INTELIGENCIA ARTIFICIAL

Camila Andrea Borja Sánchez, Alejandro Sántibañes Sánchez, Daniel Cardoz

18 de octubre de 2019

I. Resumen

El trabajo científico y los grandes avances del ser humano han permitido dar pasos agigantes en muchas áreas del conocimiento, entre ellas la tecnología que cumple un papel fundamental en la sociedad actual por la manera en cómo nos relacionamos con la misma. La inteligencia Artificial y las redes neuronales permiten brindar soluciones a problemáticas de la vida real que no han sido resueltas por la computación tradicional. El objetivo del ejercicio experimental, consiste en la implementación y evaluación de redes neuronales, con el fin de analizar su comportamiento, su aprendizaje y su variación en cuanto al ambiente en el que se ejecuta. Para ellos se hace uso de redes neuronales tales como: Perceptrón, Red Neuronal Simple y Red Neuronal Multicapa; así como también se realizan bucles de datos y repeticiones, para validas el funcionamiento de la neurona y determinar su salida y el error. Logrando con esto un aprendizaje por épocas que permite resolver problemas complejos en cuanto al uso de reconocimiento de patrones.

i. Palabras Claves

- Red
- Neurona
- Red Neuronal
- Época
- Aprenizaje
- Peso Sináptico
- Perceptrón
- Gradiante
- BackPropagation
- Delta
- Matrices
- Función de Activación

II. Abstract

The scientific work and the great advances of the human being have allowed us to take leaps and bounds in many areas of knowledge, including technology that plays a fundamental role in today's society because of the way in which we relate to it. Artificial intelligence and neural networks allow us to provide solutions to real-life problems that have not been solved by traditional computing. The objective of the experimental exercise is the implementation and evaluation of neural networks, in order to analyze their behavior, their learning and their variation in the environment in which it is executed. For them use is made of neural networks such as: Perceptron, Simple Neural Network and Multilayer Neural Network; as well as data loops and repetitions, to validate the functioning of the neuron and determine its output and error. Achieving with this a learning by epochs that allows to solve complex problems regarding the use of pattern recognition.

i. Key Words

- Net
- Neurona
- Simple Neural Network
- Epoch
- Learning
- Synaptic
- Perceptron
- Gradient
- BackPropagation
- Delta
- Matrices
- Activation Function

III. Introducción

La aplicación de la Inteligencia Artificial es una realidad en los ámbitos modernos, debido al auge de las tecnologías de la información y la comunicación, y en gran medida también a los avances en materia de software que permiten investigar con lenguajes altamente tipados, que exigen arquitecturas robustas que permitan un nivel de procesamiento superior. Todo esto ha conllevado a llevar el uso de la inteligencia artificial a los ámbitos cotidianos para resolver problemas complejos que exijan toma de decisiones y aprendizaje de máquina, automatizando procesos y permitiendo eficiencia en los recursos.

IV. Metodología Utilizada

Paradigma de programación orientada a Objetos (POO)

i. Algoritmos Usados

- Perceptrón
- Red Neuronal Simple
- Red Neuronal Multicapa
- Funciones de Activación
- Sigmoide
- HardLim
- HardLims
- Tanhgential
- Derivada de la Sigmoidal
- Derivada de la Tanhgential

V. Lenguaje de Programación usado (Breve descripción)

El lenguaje de programación usado en este proyecto ha sido Python en su versión más reciente 3.7.4, haciendo uso de módulos creados por terceros y/o propios de Python como lo son tkinter, matplotlib, time, numpy, random.

VI. Código de los algoritmos implementados

i. Perceptrón

```
class Perceptron():
def __init__(self):
super(Perceptron, self).__init__()
self.cantidad_entradas = int()
self.pesos = list()
self.bahia = float()
self.epocas = int()
self.entradas_entrenamiento = list()
self.salidas_entrenamiento = list()
self.entrenamiento = list()
self.error = float()
self.esperados = list()
```

```
self.errores = list()

def run(self, cantidad_entradas,
pesos, bahia, epocas,
entradas_entrenamiento,
salidas_entrenamiento):

self.cantidad_entradas =
cantidad_entradas
self.pesos = pesos
self.bahia = bahia
self.epocas = epocas
self.entradas_entrenamiento =
entradas_entrenamiento
self.salidas_entrenamiento =
salidas_entrenamiento

self.__cargar_set_entrenamiento()
self.__cargar_pesos_sinapticos()
self.__entrenamiento()

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

def __cargar_set_entrenamiento(self):
if (self.entradas_entrenamiento and
self.salidas_entrenamiento) and len
(self.entradas_entrenamiento)
== len(self.salidas_entrenamiento):
for i,salida in enumerate
(self.salidas_entrenamiento):
self.entrenamiento.append
((array(self.entradas_entrenamiento[i]),
salida))
else:
for i in range(self.cantidad_entradas):
self.entrenamiento.append
((random.rand(self.cantidad_entradas),
randrange(1)))

def __cargar_pesos_sinapticos(self):
if type(self.pesos) is int or
len(self.pesos) != self.cantidad_entradas:
# self.pesos = random.rand
(self.cantidad_entradas)
self.pesos = 2 * random.random
((self.cantidad_entradas,1)) - 1
print(len(self.pesos))

def __entrenamiento(self):
self.start_time = time()
for i in range(self.epocas):
 x, esperado = choice(self.entrenamiento)
 resultado = dot(self.pesos, x)
 self.error = esperado -
 self.__funcion_activiacion(resultado)
 self.esperados.append(esperado)
     self.errores.append(self.error)
#ajuste
self.pesos += self.bahia * self.error * x
self.elapsed_time = time() - self.start_time

def prediccion(self, entradas):
entrada_neta = dot(self.pesos, entradas)
resultado =
self.__funcion_activiacion(entrada_neta)
return resultado

def obtener_pesos(self):
return self.pesos

def obtener_errores(self):
return self.errores

def obtener_esperados(self):
return self.esperados
```

ii. Red Neuronal Simple

```
class RedNeuronalSimple():
def __init__(self):
    self.pesos_sinapticos = list()
    self.errores = list()

def __sigmoide(self, x):
    return 1 / (1 + exp(-x))

def __sigmoide_derivado(self, x):
    return x * (1 - x)

def __entrenamiento
(self,entradas,salidas,
numero_iteraciones):
self.start_time = time()
entradas = array(entradas)
```

```
if type(salidas) is list:
salidas = [salidas]
salidas = array(salidas).T
for i in range(numero_iteraciones):
salida = self.prediccion(entradas)
error = salidas - salida
ajuste = dot(entradas.T, error *
self.__sigmoide_derivado(salida))
self.errores.append(error)
self.pesos_sinapticos += ajuste
self.elapsed_time = time() -
self.start_time
def prediccion(self,entrada):
return self.__sigmoide
(dot(entrada, self.pesos_sinapticos))
def run(self, cantidad_entradas,
entradas,salidas,numero_iteraciones):
self.pesos_sinapticos = 2 *
random.random((cantidad_entradas,1)) - 1
self.__entrenamiento(entradas,
salidas,numero_iteraciones)
```

iii. Red Multicapa

```
class RedNeuronalMulticapa():
def __init__(self):
 #Iniciarlizar pesos
 self.pesos = []
 self.deltas = []

def run(self, capas,
activacion='tangente'):
if activacion == 'sigmoide':
self.activacion = sigmoide
self.activacion_prima =
sigmoide_derivado
elif activacion == 'tangente':
self.activacion = tangente
self.activacion_prima =
tangente_derivada

    self.pesos = []
    self.deltas = []

for i in range(1, len(capas) -1):
r = 2 * np.random.random
((capas[i-1] + 1, capas[i] + 1)) -1
self.pesos.append(r)

#asignar aleatorios a la capa de salida
r = 2 * np.random.random
((capas[i] + 1, capas[i + 1])) - 1
self.pesos.append(r)
def ajuste(self, X, y,
factor_aprendizaje = 0.2,
epocas = 1000):
self.start_time = time()
ones = np.atleast_2d(np.ones(X.shape[0]))
X = np.concatenate((ones.T, X), axis = 1)

for k in range(epocas):
    i = np.random.randint(X.shape[0])
    a = [X[i]]

for l in range(len(self.pesos)):
dot_value = np.dot(a[l], self.pesos[l])
activacion = self.activacion(dot_value)
a.append(activacion)

#Calculo la diferencia entre la capa de
salida y el valor obtenido
error = y[i] - a[-1]
deltas = [error * self.activacion_prima
(a[-1])]

#Empezamos en la segunda capa hasta
la ultima
for l in range(len(a) - 2, 0, -1):
deltas.append(deltas[-1].dot
(self.pesos[l].T)
self.activacion_prima(a[l]))
self.deltas.append(deltas)

#invertir
deltas.reverse()

#Backpropagation
for i in range(len(self.pesos)):
capa = np.atleast_2d(a[i])
delta = np.atleast_2d(deltas[i])
self.pesos[i] += factor_aprendizaje *
capa.T.dot(delta)

if k % 10000 == 0: print
('epocas:', k)
```

```
self.elapsed_time = time() -
self.start_time

def predecir(self, x):
unos = np.atleast_2d
(np.ones(x.shape[0]))
a = np.concatenate((np.ones(1).T,
np.array(x)), axis = 0)for l in
range(0, len(self.pesos)):
a = self.activacion(np.dot
(a, self.pesos[l]))
return a
```

iv. Función de Activación

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

def sigmoide_derivado(x):
return sigmoide(x) *
(1 - sigmoide(x))

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

def tangente_derivada(x):
return 1 - x**2
```

VII. Red Perceptrón

Se capturan las entradas, los pesos sinápticos de cada una de ellas, se captura el sesgo de bahías y se capturan los datos de entrenamiento, lo que son las épocas o repeticiones que va a tener la red neuronal, y consigo las entradas y salidas esperadas, para que la red aprenda. Se define la función de activación la cual regresa un cero (0) si el valor analizado es menor que cero y el caso contrario retorna uno(1). Para el entrenamiento, se recrea un bucle que es dirigido u orquestado por las épocas capturadas anteriormente, se seleccionan al azar una entrada una salida de los datos o set de entrenamiento, en donde las entradas se les aplica la operación punto. El resultado de la operación es evaluado por la función de activación, que posteriormente se le sustrae a la salida esperada seleccionada para calcular el error. El error es utilizado para realizar el ajuste a los pesos sinápticos, siendo el nuevo valor de los pesos sinápticos la suma de los pesos más el resultado del producto el sesgo de bayas, los pesos sinápticos y la entrada seleccionada.

VIII. Red Neuronal Simple

Se capturan los pesos sinápticos, el sesgo de bahías, la cantidad de entradas, y cantidad de salidas, además se define la función de activación y su derivada. Por otra parte, se capturan los datos de entrenamiento para conocer la cantidad de repeticiones o épocas en las que se va entrenar y ajustar la red; seguido, el conjunto de entradas y salidas esperadas que sirven de guía para el aprendizaje de la red neuronal. Se realiza el entrenamiento en donde se ajustan los pesos sinápticos de cada una de las entradas de la red neuronal. Este consiste hacer un bucle repetitivo con la sentencia “for”, el cual va a iterar entre la cantidad de épocas capturadas anteriormente, dentro del bucle, se evalúa en la función de activación el resultado de la operación producto punto entre los vectores de “entradas” y “pesos sinápticos”, seguido a esto, se calcula el error para esta iteración del entrenamiento, el cual consiste en restarle al vector de “salidas” el resultado de la operación producto punto calculada anteriormente. Por último, se realiza el ajuste al vector de “pesos sinápticos”, sumándole el resultado de la operación producto punto entre la transpuesta del vector “entradas” y el producto entre la multiplicación del vector “error”, por la evaluación en la función de activación derivada del resultado entre la operación producto punto de los vectores “entradas” y “pesos sinápticos”. Luego de haber terminado el entrenamiento, la red neuronal debe de estar en la capacidad de hacer predicciones con cualquier otro dato de entrada dado, para esto basta en evaluar en la función de activación el resultado de la operación producto punto entre el

vector de “entrada” ingresado y el vector de “pesos sinápticos”

IX. Red Neuronal Multicapa

Para la red neuronal multicapa es necesario capturar o indicar los siguientes parámetros: cantidad de neuronas en la capa de entrada, oculta y de salida, el factor de aprendizaje y los datos de entrenamiento como una matriz de dos columnas (valores de entrada y salidas esperadas) y n cantidad de datos ingresados Antes de realizar el entrenamiento es necesario cargar la matriz de “pesos sinápticos” agregando en sus filas el vector de “pesos sinápticos” de cada una de las capas, con valores iniciales aleatorios o un conjunto de valores indicados previamente. También es necesario definir la función de activación y la derivada de la misma. Se crea una matriz con todos sus valores en 1, de una fila por la misma cantidad de columnas que los datos ingresados como set de entrenamiento Se agrega la transpuesta de la matriz de unos creada anteriormente, a la matriz o set de entrenamiento por lo largo del eje de las columnas Se crea un bucle repetitivo que iterará desde 0 hasta la cantidad de épocas indicadas, dentro de este bucle se seleccionara de la matriz entrenamiento un arreglo o fila con un conjunto de datos de entrada y su salida esperada al azar. Se crea un sub bucle repetitivo que iterar desde 0 hasta la longitud de la matriz de “pesos sinápticos”, dentro de este sub bucle se carga una lista con las salidas calculadas de cada capa, en otras palabras, una lista cargada con los valores devueltos por la función de activación al evaluar el resultado de la operación producto punto entre cada valor del conjunto de datos de entrada seleccionado al azar por cada una de las filas pertenecientes a la matriz de pesos sinápticos Fuera del sub bucle se calcula el error para esa iteración del entrenamiento, restándole a la salida esperada seleccionada al azar, la lista cargada con los valores de salida calculados, solo hasta el penúltimo elemento, ósea omitiendo el valor de salida calculado de la capa de salida. Se cargan los deltas de esa iteración del entrenamiento, multiplicando el error calculado por el valor retornado de evaluar en la función derivada de activación las salidas calculadas de la capa de entrada y la capa oculta de la red. Se crea un sub bucle que se repite desde el penúltimo elemento hasta el primero de la lista de salidas calculadas, dentro de este sub bucle se le agrega a la lista deltas el producto de la multiplicación entre: el producto de la multiplicación matricial de la misma lista por el arreglo transpuesto de la matriz de pesos sinápticos en la posición de iteración del bucle; multiplicado por el resultado de evaluar en la función derivada de activación el elemento de la lista de salidas calculadas en la posición de iteración del bucle. Por fuera del sub bucle, se invierte la lista deltas Por último, en el ciclo de entrenamiento se hace el ajuste de los pesos sinápticos de cada una de las capas, con el uso del algoritmo de “Backpropagation”, que consiste en crear un sub bucle entre el rango de 0 hasta la longitud de la matriz de pesos sinápticos. Dentro del sub bucle se crea dos matrices, una en la que se almacenará el elemento de la lista de salidas calculadas en la posición de iteración del sub bucle, y la otra que contendrá el elemento de la lista deltas en la posición de iteración del sub bucle. Luego se hace el ajuste en la matriz de pesos sinápticos en la posición de iteración del sub bucle, sumándole el producto del factor de aprendizaje, por el resultado de la multiplicación matricial entre las matrices capa y delta creadas anteriormente. Ya culminado el entrenamiento, la red neuronal multicapa estará en la posición de hacer predicciones de salidas de cualquier conjunto de datos de entrada. Para realizar dicha predicción se crea una matriz “A” concatenando a lo largo del eje de las filas, la transpuesta de un arreglo con valores de unos (“1”) y de dimensiones (1 x 1) al arreglo de la entrada a predecir. Luego en un ciclo desde 0 hasta la longitud de los pesos sinápticos de la red neuronal multicapa se calcula la salida, asignado a la matriz “A” el resultado proporcionado por la función de activación al evaluar el resultado de la ope-

ración producto punto entre la matriz creada "A" por el valor el arreglo de la matriz de pesos sinápticos en la posición de iteración de ciclo. Se retorna la matriz "A" como respuesta a salida para el conjunto de entradas a predecir.

X. Resultados Alcanzados

Para contrastar el funcionamiento de cada tipo de red neuronal, se capturo el tiempo de procesamiento en el entrenamiento de cada una de ellas (Figura 3 y 4), la capacidad de aprendizaje que tiene cada tipo de red variando su factor de aprendizaje (Figura 5, 6 y 7), e intercambiando entre el uso de diferentes funciones de activación (Figura 8, 9 y 10).

En todos los casos se utilizo el conjunto de entrenamiento visualizado en la Figura 1. También se comprobo la capacidad de aprendizaje de cada red neuronal con los datos de prueba mostrados en la Figura 2, contrastando la salidad proporcionada por la red neuronal luego de ser entrenada, con una salida esperada.

DATOS ENTRENAMIENDO			
ENTRENAMIENDO			SALIDAS
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Figura 1: Datos de entrenamiento

Conjunto de datos de comprobación, usados para corroborar el aprendizaje y el funcionamiento de la red.

DATOS DE COMPROBACION APRENDIZAJE			
ERNTRADAS			SALIDA
0	1	0	0
0	1	1	1
1	1	0	0
1	1	1	1

Figura 2: Datos de prueba

las tres redes neuronales se parametrizaron así: bahia = 0, cada red cuenta con 3 entradas y 1 salid

	VALORES					
	1000			10000		
MÁQUINA	PERCEPTRÓN	RNS	RNM	PERCEPTRÓN	RNS	RNM
PC_Asus	0,0220	0,0010	0,0781	0,0751	0,0010	0,7717
PC_Acer	0,0440	0,1520	0,5160	0,4920	1,2919	4,4557
PC_Lenovo	0,5000	0,0360	0,1093	0,0938	0,2910	1,0781

Figura 3: Diferentes cantidad de Épocas parte 1

ida, en el caso de la red neuronal multicapa, la estructura de las capas es [3,3,1]

	100000			1000000		
	PERCEPTRON	RNS	RNM	PERCEPTRÓN	RNS	RNM
7	0,6929	1,5112	7,6742	6,9968	14,8242	73,4985
7	4,5054	11,2594	41,2351	47,1916	110,3699	407,1802
1	0,9844	2,7206	9,9225	10,5319	21,5639	98,5065

Figura 4: Diferentes cantidad de Épocas parte 2

El factor de aprendizaje inside en gran medida en el aprendizaje y adaptación de la red neuronal a las distintas situaciones, como se puede ver demostrado en las Figuras 5, 6 y 7. Allí se realizo el ejercicio de entrenar cada red neuronal con un factor de aprendizaje distinto. Cada red estuvo afrontada a un entrenamiento de 1 millon de epocas.

					FACTOR = 0.3			
MÁQUINA	PERCEPTRÓN				RNS	RNM		
PC_Acer	0	0	1	0	no parametrizable	0,569	0,184	0,784	0,500
PC_Lenovo	1	1	0	0	no parametrizable	0,896	0,102	0,627	0,500

Figura 5: Diferentes Factores de Aprendizaje parte - 1

Diferentes Fatores de Aprendizaje								
				FACTOR = 0.5				
PERCEPTRÓN				RNS	RNM			
1	1	0	0	no parametrizable	0,500	0,499	0,731	0,500
0	0	0	0	no parametrizable	0,500	0,497	0,742	0,500

Figura 6: Diferentes Factores de Aprendizaje parte - 2

				FACTOR = 0.9				
PERCEPTRÓN				RNS	RNM			
0	0	0	0	no parametrizable	0,500	2,816	0,610	0,500
1	1	1	1	no parametrizable	0,500	0,499	0,781	0,500

Figura 7: Diferentes Factores de Aprendizaje parte - 3

Se hizo uso de diferentes funciones de activación sigmoidal, tangencial y escalonada. este ejercicio se puede ver evidenciado en las Figuras 8, 9 y 10, en donde se realizo un entrenamiento de 1 millón de epocas en cada una de las redes y un facto de aprendizaje = 0.

Las redes neuronales se parametrizarón así: epocas = 1000000, bahias=0.3, y se define la función de activacion en

MÁQUINA	PERCEPTRON ESCALONADA			
	COMPROBACION 1	COMPROBACION 2	COMPROBACION 3	COMPROBACION 4
PC_Asus	0	1	0	0
PC_Acer	1,000	0,000	0,000	1,000
PC_lenovo	0	0	0	0

Figura 8: Uso de diferentes funciones de activación parte - 1

las redes neuronales simples y multicapas

FUNCIONES DE ACTIVACIÓN							
RNS							
SIGMOIDE				TANGENTE			
COMPROBACION 1	COMPROBACION 2	COMPROBACION 3	COMPROBACION 4	COMPROBACION 1	COMPROBACION 2	COMPROBACION 3	COMPROBACION 4
0,5326	0,3771	0,8015	0,6821	0,988	0,995	0,999	0,999
0,532	0,377	0,801	0,682	0,988	0,996	1,000	0,999
0,532	0,377	0,801	0,682	0,988	0,995	0,999	0,999

Figura 9: Uso de diferentes funciones de activación parte - 2

RNM							
SIGMOIDE				TANGENTE			
COPROBACION 1	COPROBACION 2	COPROBACION 3	COPROBACION 4	COMPROBACION 1	COMPROBACION 2	COMPROBACION 3	COMPROBACION 4
0,818	2,584	0,747	0,500	0,995	0,000	0,996	0,000
0,569	0,184	0,784	0,500	0,996	0,001	0,999	0,003
0,896	0,102	0,627	0,500	0,999	-0,005	0,999	-0,005

Figura 10: Uso de diferentes funciones de activación parte - 3

XI. Conclusión

El presente trabajo representó un reto, debido a la complejidad de la investigación y de la abstracción que este sugiere. Después de analizar, evaluar y entrenar tres algoritmos neuronales, se puede hablar de un factor de aprendizaje variable que da luces acerca del error en el aprendizaje, del sesgo de bahías y de la mejora del agoritmo.

Además de la aplicación de los algoritmos, se aplican diferentes funciones de activación, que ayudan a acercarse más al restultado espera, y ayuda de igual manera en el proceso de aprendizaje y de entrenamiento de la neurona. Los datos se pueden interpretar como información que permita establecer una metodología de aprendizaje continuo y que permite resolver problemas más cercanos a la realidad cotidiana.

La experimentación, la práctica y demás actividades desarrolladas, estuvieron orientadas al entrenamiento y análisis de datos tanto de entrada como de salida, de las epocas a la cual se sometia y a los diferentes factores de aprendizaje.