

Universidad Politécnica de Chiapas



Inteligencia Artificial

Perceptrón

GARCÍA THOMAS ANDRES ALEJANDRO

Tuxtla Gutiérrez, Chiapas.

Índice	
Introducción	2
Métodos	2
Implementación	3
Resultados	6
Bibliografía	9

Índice de figuras

Figura 1. Neurona biológica y neurona artificial (representación)	2
Figura 2. Bucle entrenamiento parte 1	4
Figura 3. Bucle entrenamiento parte 2	4
Figura 4. Bucle entrenamiento parte 3	5
Figura 5. Gráfica comparativa	6
Figura 6. Gráfica de pesos para η_1	7
Figura 7. Gráfica de pesos para η_2	7
Figura 8. Gráfica de pesos para η_3	8
Figura 9. Gráfica de pesos para η_4	8
Figura 10. Gráfica de pesos para η_5	9

Índice de tablas

1.Representación de entradas.txt	3
---	---

Introducción

Un perceptrón es una red neuronal antigua, implementada en la computadora IBM 704 en el año 1957, la idea del perceptrón fue planteada por Frank Rosenblatt, consistía en un clasificador binario o discriminador lineal, esto quiere decir que a partir de un entrenamiento con datos el perceptrón era capaz de reconocer patrones y tomar decisiones.

Una red neuronal en informática es un conjunto de neuronas artificiales cuya implementación tiene como modelo a las neuronas biológicas.

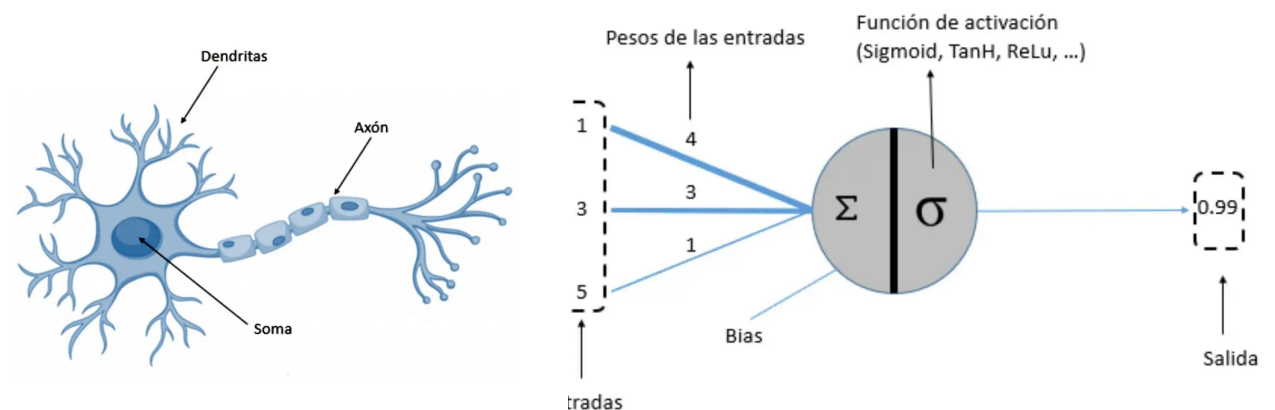


Figura 1. Neurona biológica y neurona artificial (representación)

Métodos

Durante este desarrollo se entrenará a la red neuronal ó perceptrón ajustando el peso de las entradas y al parámetro extra denominado bias, el ajuste que se hará será para que la salida obtenida tenga el menor error posible.

Función de activación : En función de la entrada la neurona se activa y emite/ transmite su salida

El algoritmo de entrenamiento es el siguiente

1. Iniciar el vector de pesos **W** con valores aleatorios
2. Iniciar el bucle de entrenamiento (optimización - minimización):

a. Calcular $u = \sum_{i=0}^n x_i * w_i$

- b. Evaluar la salida de la función de activación $FA(u)$
La función a utilizar será la de escalón

- c. Calcular el error $e_k = y_1 - y_0$
- d. Corregir en el vector **W**, la posición **W[k]** tomando en cuenta el error e observado

$$W_{k+1} = W_k + \eta x e_k$$

3. Detener el proceso si la magnitud del error es menor a un umbral ϵ , $|e| < \epsilon$

y_1 indica a la salida esperada

y_0 indica a la salida obtenida

η taza de aprendizaje

Implementación

Se ha escrito un programa en Python, el cual consta de una GUI donde el usuario introduce las 5 tazas de aprendizaje o se generarán de manera aleatoria, durante la ejecución del programa se harán diferentes operaciones matriciales.

Se requiere previamente un archivo en formato txt, llamado **entradas.txt**, archivo que se estructura de la siguiente manera

bias	x_0	x_1	x_n
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

1.Representación de **entradas.txt**

La fila resaltada por el color cyan no se incluirá dentro del archivo, es una representación de que hace referencia cada columna.

La parte central del algoritmo o la parte del bucle fue definida en el siguiente código fuente que ha sido dividido

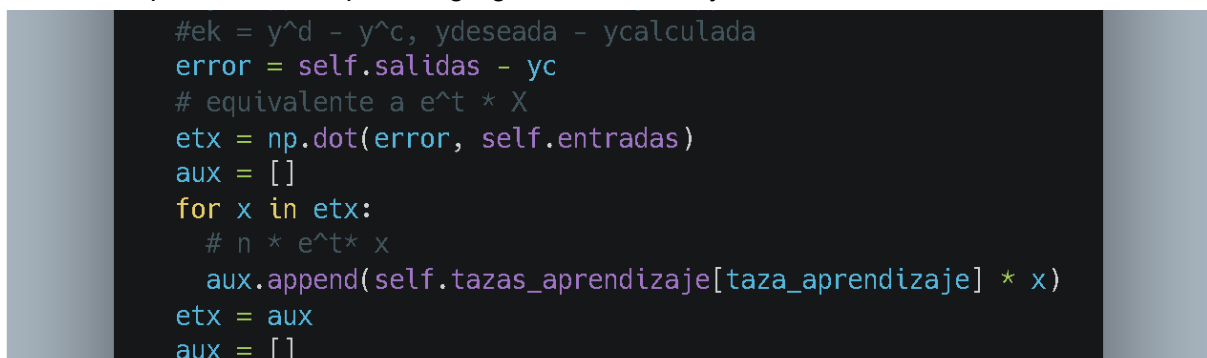
A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in Python and defines the first part of a training loop. It starts with a while loop that continues as long as self.norma is greater than self.error. Inside the while loop, it calculates u as the dot product of w and self.entradas using np.dot. A comment indicates that Y^c(u) = FA(u). Then, it initializes yc as an empty list. A comment explains that yc[i] will be 0 if u[i] < 0 and 1 if u[i] >= 0. A for loop iterates over each element i in u, and for each i, it appends 1 to yc if i >= 0, otherwise it appends 0.

```
while(self.norma > self.error):
    u = np.dot(w,self.entradas)
    # Y^c(u) = FA(u)
    yc = []
    # 0 si u < 0, 1 si u >= 0
    for i in u:
        yc.append(1) if i >= 0 else yc.append(0)
```

Figura 2. Bucle entrenamiento parte 1

Se establece la condición de parada del bucle y luego se procede a calcular el valor de U_k llamando a la función **dot** de la librería **np(numpy)** el cual retorna un array con el valor de la multiplicación entre el vector **w** y **x(self.entradas)**.

Inicializa el vector **yc** donde se almacena las salidas calculadas al pasarlas por la función de activación que determina que se agrega un 1 si $u \geq 0$ y un 0 si $u < 0$

A screenshot of a code editor with a dark background. The code continues the training loop from the previous figure. It calculates the error as self.salidas minus yc. A comment indicates that the error is equivalent to e^t * X. Then, it calculates etx as the dot product of error and self.entradas using np.dot. It initializes aux as an empty list. A for loop iterates over each element x in etx, and for each x, it appends the product of self.tazas_aprendizaje[taza_aprendizaje] and x to aux. Finally, it assigns etx to aux and resets aux to an empty list.

```
#ek = y^d - y^c, ydeseada - ycalculada
error = self.salidas - yc
# equivalente a e^t * X
etx = np.dot(error, self.entradas)
aux = []
for x in etx:
    # n * e^t * x
    aux.append(self.tazas_aprendizaje[taza_aprendizaje] * x)
etx = aux
aux = []
```

Figura 3. Bucle entrenamiento parte 2

Siguiendo con el entrenamiento se calcula un vector de error **error** al restar las salidas esperadas (**self.salidas**) con las salidas calculadas (**yc**), posteriormente se obtiene el producto entre **error** y **x(self.entradas)** lo que es equivalente a $e^t \times X$, se utiliza un vector **aux** para multiplicar cada valor obtenido en **etx** por η (valor obtenido del vector **self.tazas_aprendizaje**) que es la tasa de aprendizaje, se asigna el valor de **aux** a **etx** y **aux** se limpia para volver a utilizarla adelante.

```

# Wk+1 = Wk + n * e^t * x
for i in range(0, len(w)):
    aux.append(w[i]+etx[i])
# wk+1
w = aux
# para sacar la norma es la raiz cuadrada de la
#sumatoria de los cuadrados de la lista de errores
# sumatoria de cuadrados
temp = 0
for e in error:
    temp += e**2
self.norma = temp
# raiz cuadrada
self.norma = math.sqrt(self.norma)

```

Figura 4. Bucle entrenamiento parte 3

En esta parte el ciclo for es para obtener el vector W_{k+1} que es almacenado temporalmente en **aux** para así después asignarlo al vector **w**.

Como último paso se calcula la norma primero realizando la sumatoria de los cuadrados de los errores obtenidos y como último extraer la raíz cuadrada de la sumatoria, en una

expresión se vería de la siguiente forma: $||V|| = \sqrt{\sum_{i=0}^n v_i^2}$

Resultados

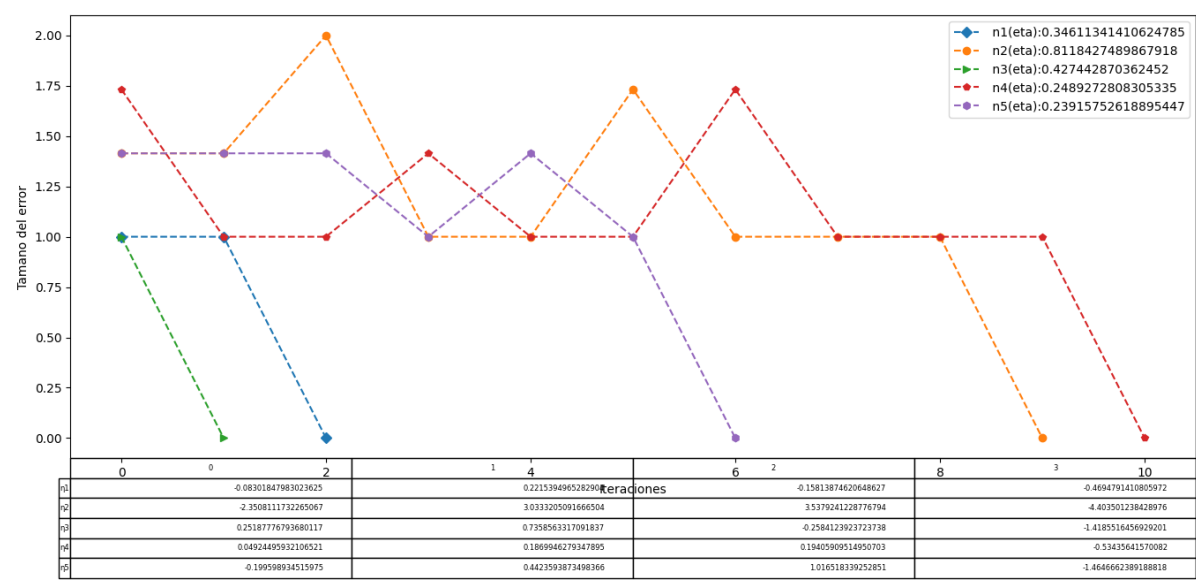


Figura 5. Gráfica comparativa

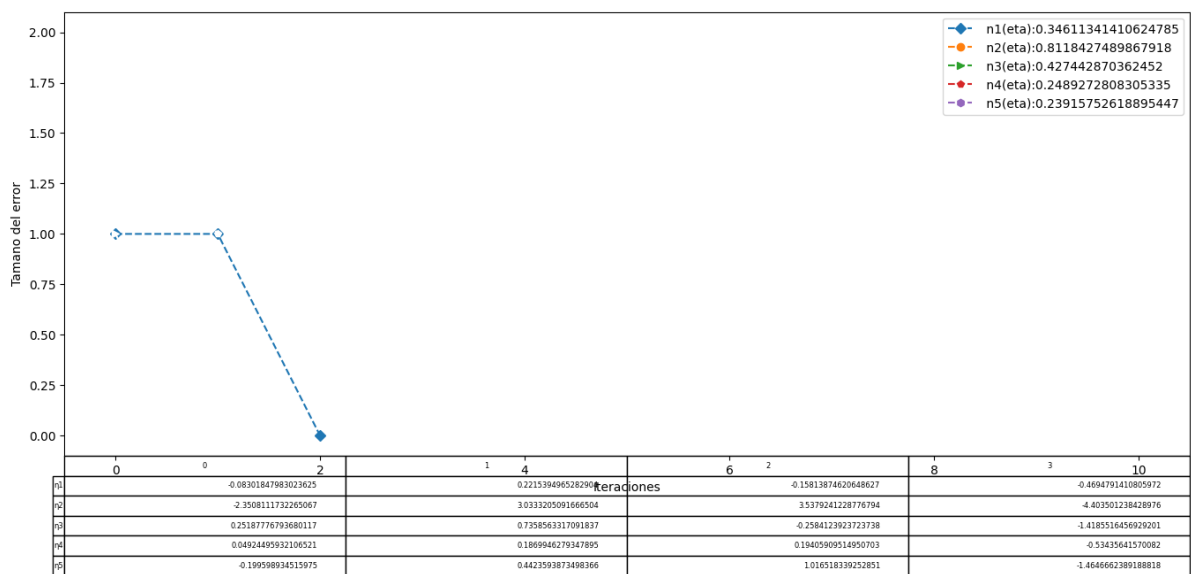


Figura 6. Gráfica de pesos para η_1

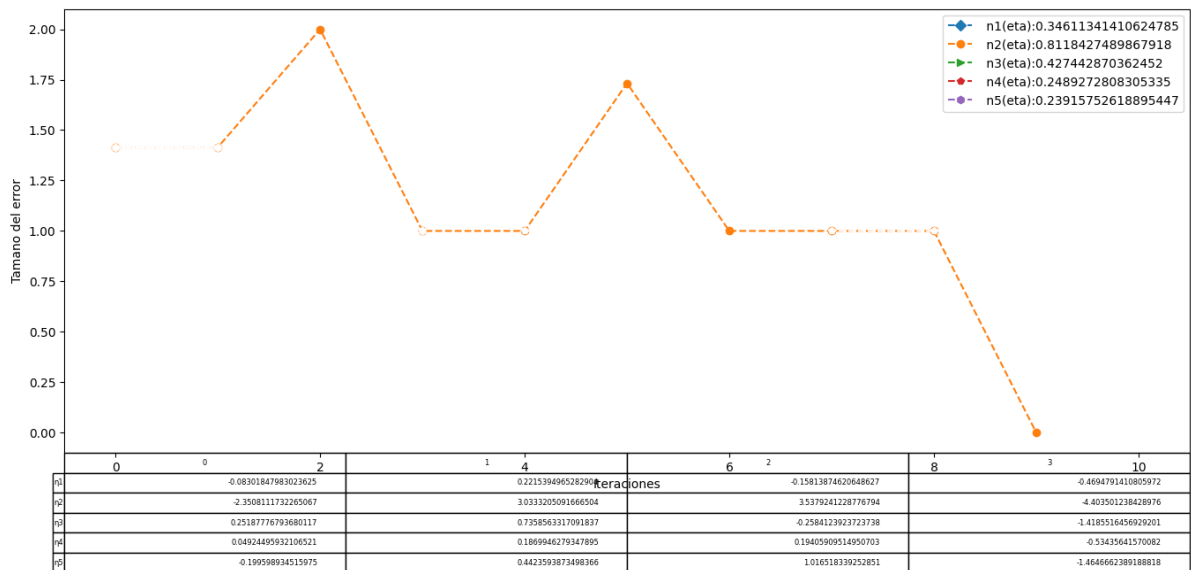


Figura 7. Gráfica de pesos para η_2

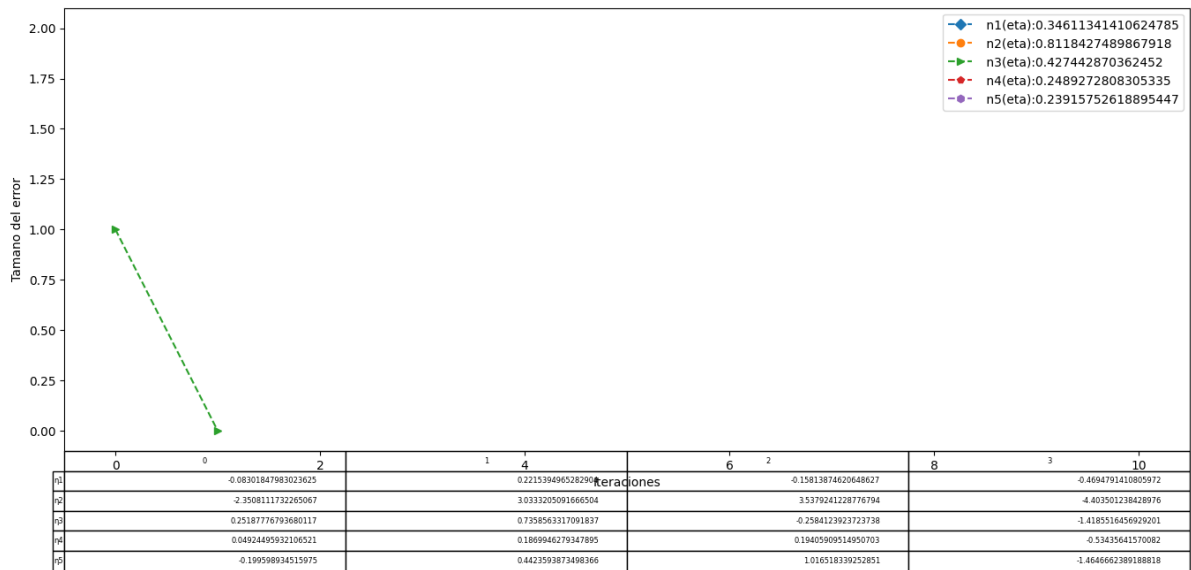


Figura 8. Gráfica de pesos para η_3

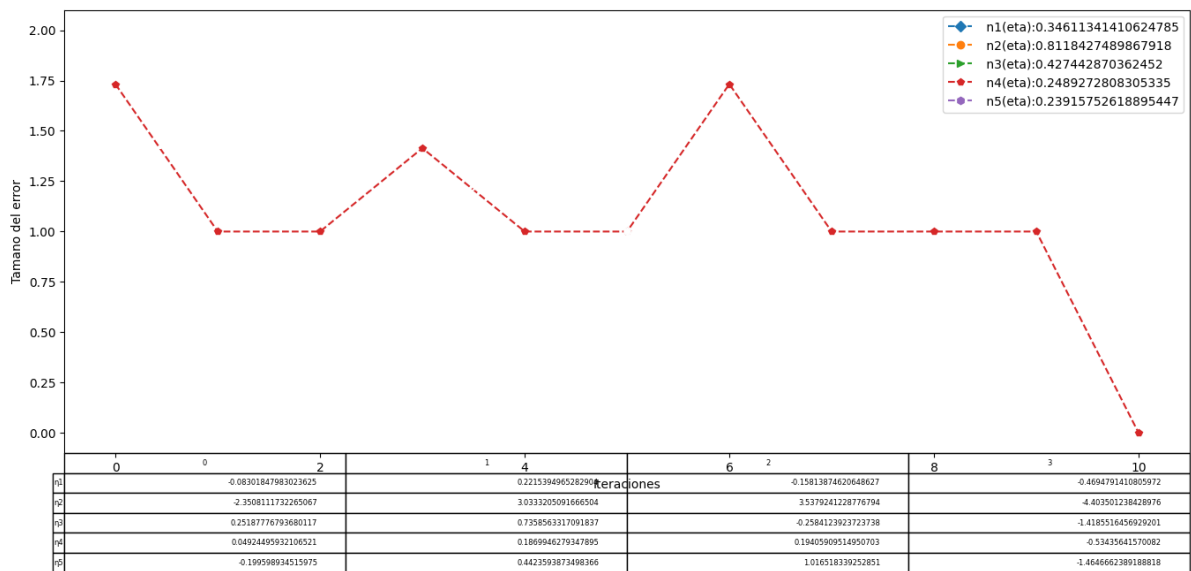


Figura 9. Gráfica de pesos para η_4

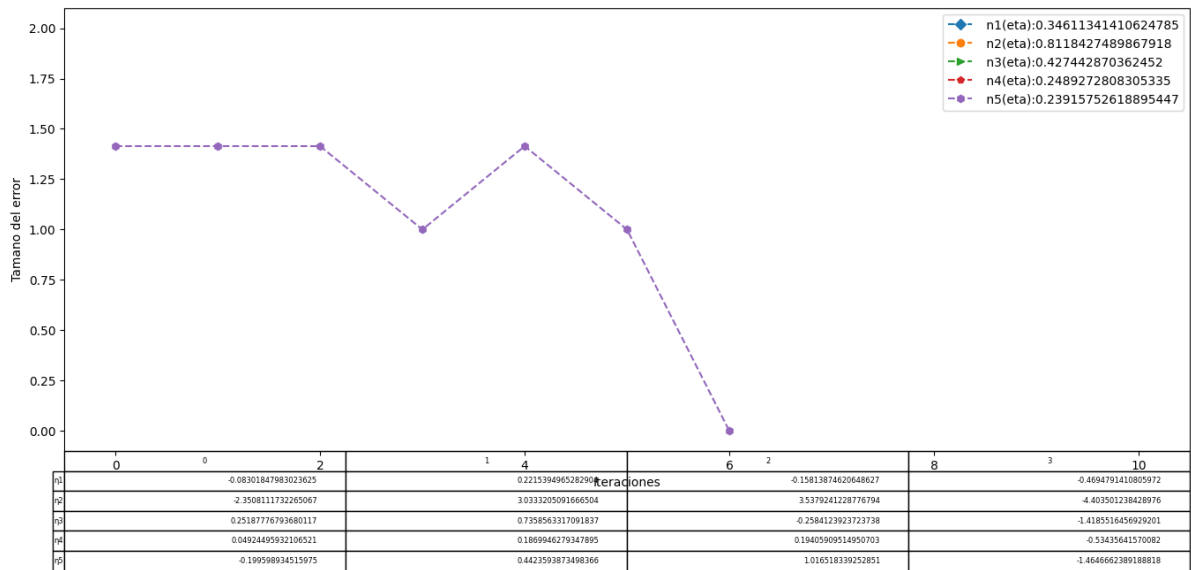


Figura 10. Gráfica de pesos para η_5

Bibliografía

Autor/a. "Título." Título del contenedor (autocontenido si el libro) , Otros contribuyentes (traductores o editores), Versión (edición), Número (vol. Y / o no.), Editor/a, Fecha de publicación, Ubicación (páginas, párrafos y / o URL, DOI o enlace permanente). 2nd título del contenedor, Otros contribuyentes, Versión, Número, editor/a, fecha de publicación, ubicación, fecha de acceso (si es aplicable).

Islas Ximena, ¿Qué es un perceptrón? La red neuronal artificial más antigua, 5 de Febrero del 2021, Ubicación: [¿Qué es un perceptrón? | Inteligencia Artificial \[2021\]](#), fecha de acceso: 01 de marzo del 2022

Olalla Oscar García, Redes Neuronales artificiales: Qué son y cómo se entrenan, 16 de septiembre del 2019, Ubicación: [Redes Neuronales artificiales: Qué son y cómo se entrenan](#), fecha de acceso: 01 de marzo del 2022

LUCA AI Powered Decisions, [¿Qué es la Función de Activación?](#), fecha de acceso: 01 de marzo de 2022