Taller: Programación de un Perceptrón Simple

Autor:

JUAN HUMBERTO TABORDA ACOSTA

Tutor:

TONNY ENRIQUE JIMENEZ MARQUEZ

INTELIGENCIA ARTIFICIAL GRUPO 02

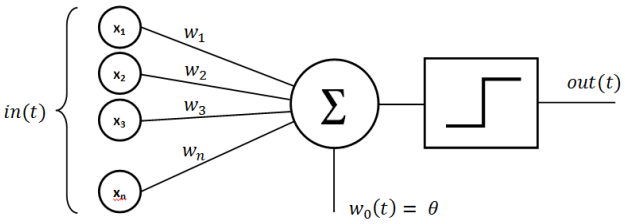
UNIVERSIDAD POPULAR DEL CESAR

FACULTAD DE INGENIERÍA DE SISTEMAS

VALLEDUPAR-CESAR

SEPRIEMBRE 2025

1. Introducción



Un perceptrón es una neurona artificial, indispensable para las redes neuronales del Deep Learning. Propuesta en 1957 por Frank Rosenblatt, esto lo hizo basándose en los primeros conceptos de neuronas artificiales, propuso la "regla de aprendizaje del perceptrón". Esta efectúa cálculos para detectar características o tendencias en los datos de entrada. Es un algoritmo para el aprendizaje supervisado de clasificadores de binarios. Ese es el algoritmo el cual permite que las neuronas artificiales aprendan y traten los elementos a partir de una serie de datos.

El perceptrón desempeña un papel esencial en los proyectos de Machine Learning. Se utiliza en gran medida para clasificar datos, o como algoritmo que permite simplificar o supervisar las capacidades de aprendizaje de los clasificadores binarios.

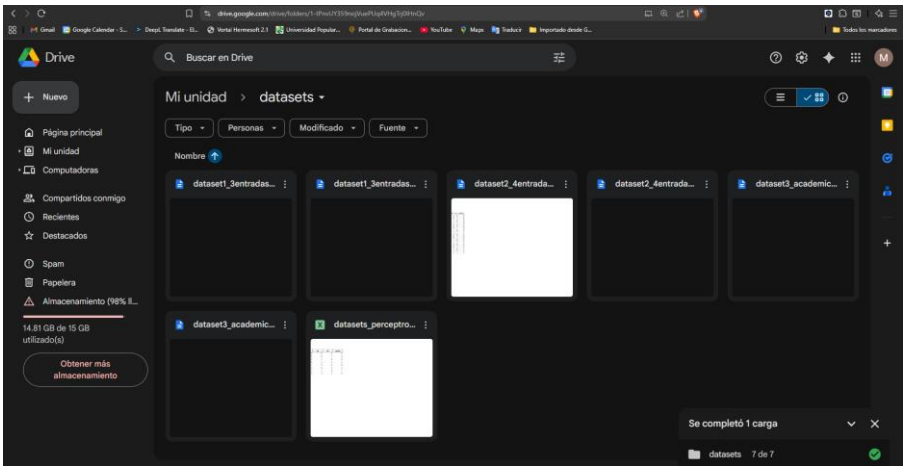
Según la **Perceptron Learning Rule** (regla de aprendizaje del perceptrón), el algoritmo enseña automáticamente los coeficientes de peso óptimo. Para determinar si una neurona "se enciende" o no, las características de los datos de entrada se multiplican por esos pesos. (DataScientest, s.f.).

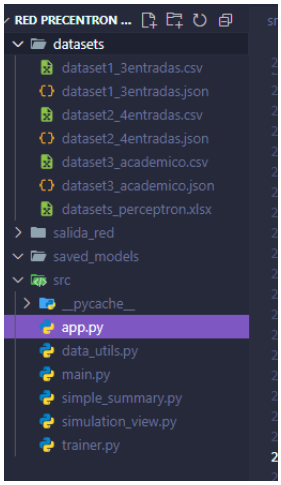
Esta es importante ya que es la base de las redes neuronales modernas. El perceptrón fue el primer modelo que mostró cómo una máquina podía aprender de los datos. Lo hizo ajustando sus pesos y sesgos en función de los errores. Si el perceptrón hiciera una predicción incorrecta, modificaría ligeramente sus parámetros para hacerlo mejor la próxima vez. Este proceso, llamado *corrección de errores*, es la esencia del aprendizaje automático. (Tahir, 2025).

1.1. Objetivo: Comprender el funcionamiento del perceptrón simple, aplicando sus fundamentos teóricos y prácticos.

2. Desarrollo

Dataset 3 simula un problema real con 3 entradas continuas/categóricas horas de estudio la cual es un numero entero que representa la cantidad de tiempo de horas de estudio de un estudiante, asistencia la cual es la cantidad de clases a la que asistió el estudiante y participación mide en 3 niveles de participación en clase de cada estudiante y 1 salida (aprueba =1 o no aprueba = 0), interpretando las variables como factores académicos que influyen en el rendimiento estudiantil. Este está alojado de manera en un drive privado para que solo las personas con acceso puedan entrar, estos archivos del drive se descargan en el dispositivo en que se vaya a utilizar la red y se guarda en una carpeta llamada "dataset", estando ahí ya podrá ser utilizable para el usuario final para empezar a trabajar.





3. Inicialización

Para el entrenamiento de este modelo, por simplicidad y por recomendación del profesor, los pesos y umbrales se inicializaron de manera aleatoria, asignándoles valores dentro de un rango entre -1 y 1, esto con el fin de evitar desviación y permitir que el algoritmo de aprendizaje vaya ajustando progresivamente estos 2 parámetros durante las distintas iteraciones.

4. Configuración del Entrenamiento

Con respecto a la configuración de todo el entorno para la realización de este modelo, se basó en la Regla Delta, técnica expuesta por el docente, la cual busca reducir la diferencia entre la salida deseada y la salida obtenida por la red a través de un ajuste de los pesos y umbral en cada iteración. La idea principal es

que, si la salida del perceptrón no coincide con la esperada, se corrigen los parámetros en proporción al error y a la tasa de aprendizaje, de modo que la red mejore progresivamente su capacidad de clasificación. Para este proceso se empleó la función de activación escalón, que transforma el potencial neto en una salida binaria (0 o 1). Los parámetros definidos por el usuario fueron: el número máximo de iteraciones, el error máximo permitido como criterio de parada, y la tasa de aprendizaje (η), que controla el tamaño de los ajustes en cada actualización, estos valores son definidos por el usuario ya que este es el encargado de ver que tanto aprendizaje o que tan rápido o lento la desea que la red aprenda, ya que en si, para esta tarea no existen valores universales ideales, solo rangos, también depende del dataset, la complejidad del problema y demás. Por ultimo para el entrenamiento, se decidió que el dataset apenas se ingrese, este mismo se particione en una proporción 80% datos de entrenamiento y 20% datos para simular y probar la red.

5. Entrenamiento del modelo

Para todo lo anteriormente dicho se utilizó la regla delta donde:

$$w_{ji}^{k+1} = w_{ji}^{k} + \propto \times EL_i \times x_j$$

w_{ji}^{k+1}= Peso Nuevo
w_{ji}^{k} = Peso actual
$\propto$ = rata de aprendizaje
EL_i = Error Lineal
x_j = Entrada
Umbral:
$u_i^{k+1} = u_i^{k} + \propto \times EL_i \times x_0$
u_i^{k+1} = Umbral Nuevo
u_i^{k} = Umbral actual
$\propto$ = rata de aprendizaje
EL_i = Error Lineal
x_0= 1

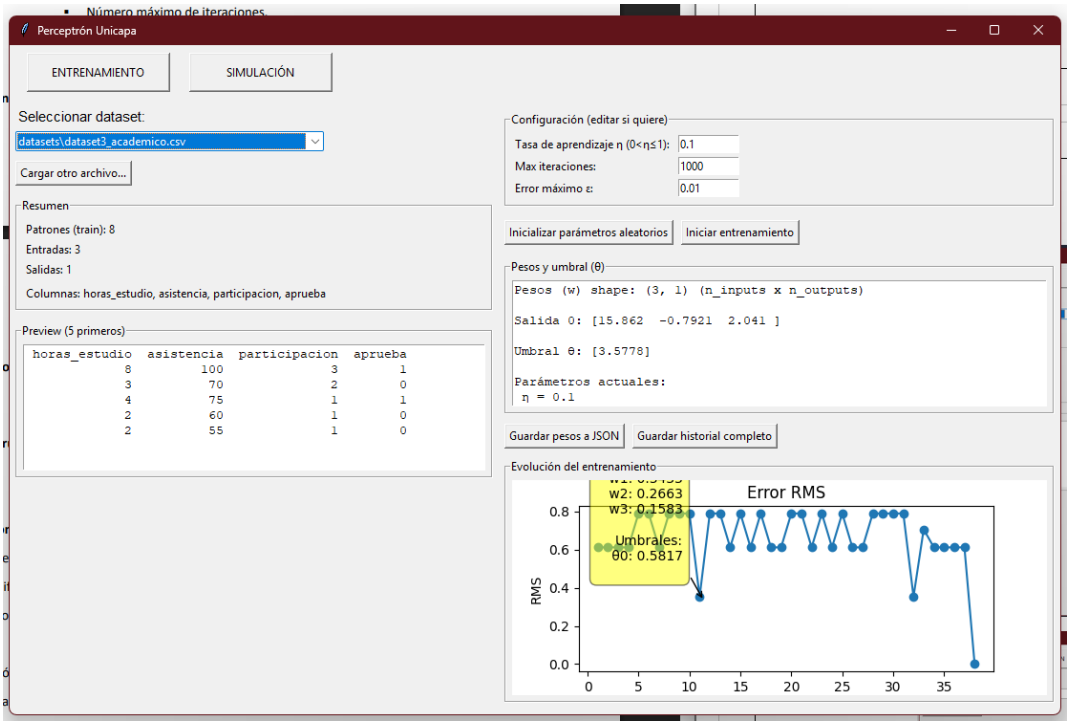
Simplificado todo esto, nos quiere decir que el nuevo peso es el mismo peso anterior, pero con una correlación basada en el error, la tasa de aprendizaje y la entrada de la red. Junto a esto tenemos la función de activación:

$$y = \begin{cases} 1 \; si \; u \geq 0 \\ 0 \; si \; u < 0 \end{cases}$$

$$\rightarrow Esto\ convierte\ el\ potencial\ neto\ en\ una\ salida\ binaria$$

Para englobar todo lo anterior, lo que nos quiere decir pues que en cada iteración del entrenamiento revisa un patrón de entrada, calcula el error entre la salida deseada y la de la red, y ajusta los pesos y el umbral proporcionalmente al error. Así, poco a poco, el perceptrón "aprende" a clasificar correctamente.

En la siguiente imagen se puede visualizar el funcionamiento de la red, donde esta selecciona alguno de los dataset propuestos, cargan los patrones, entradas y salidas, pide al usuario tasa de aprendizaje, máximo de iteraciones y error mínimo, los pesos y umbrales se cargan de manera automática y la gráfica muestra consigo cada iteración y como la red va actualizando el valor de los pesos y umbral.



Como se puede observar en la gráfica, representa la evolución del error (RMS) durante el entrenamiento del perceptrón, en el eje X se ubican las iteraciones y en el eje Y el valor del error, de modo que cada punto refleja el desempeño del modelo en esa iteración. A medida que avanza el entrenamiento, el perceptrón ajusta sus pesos y umbrales, y el RMS indica qué tan cerca están las salidas calculadas de las salidas deseadas. En este caso, se observa cómo el error fluctúa y tiende a disminuir hasta acercarse a cero, lo que significa que el perceptrón fue corrigiendo sus parámetros y aprendiendo la regla de clasificación planteada.

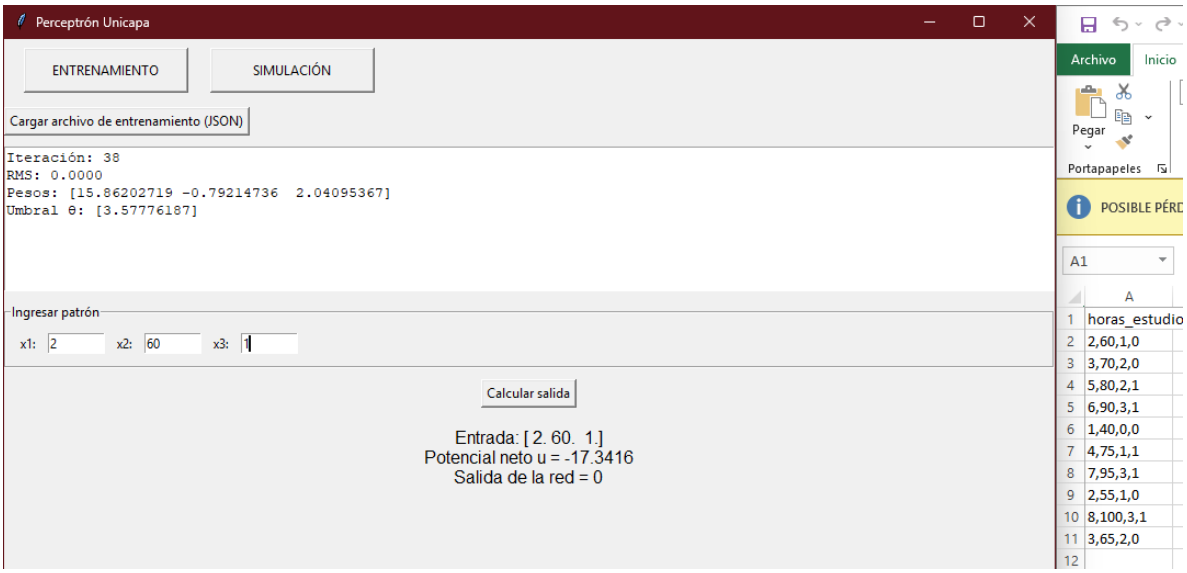
6. Condiciones de Parada

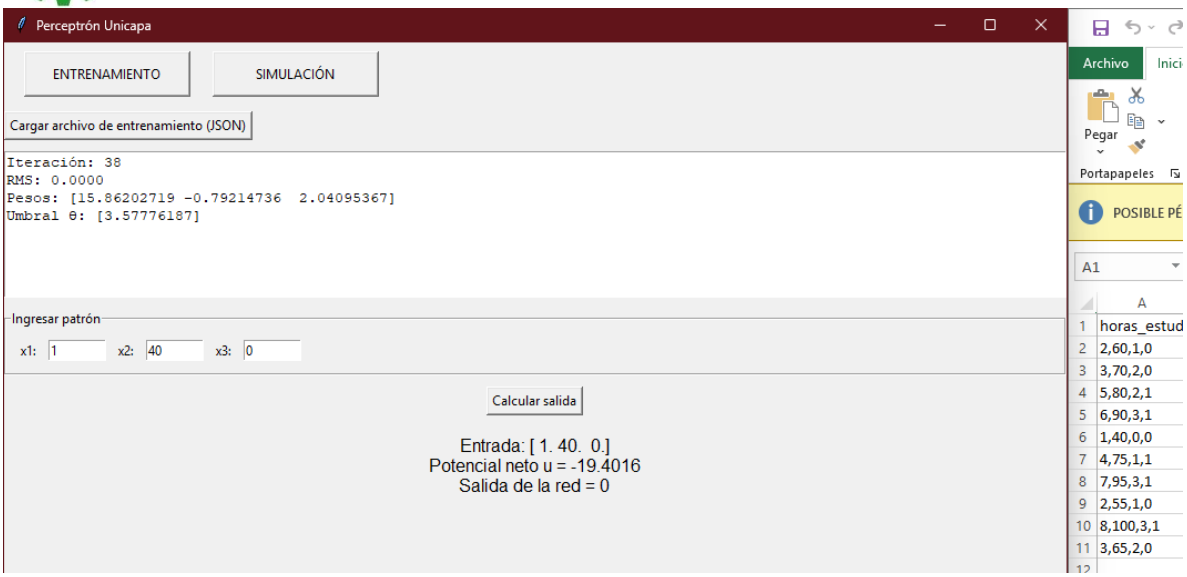
Bueno en este caso, al final del desarrollo el error (RMS) me daba muy grande, llegando hasta igualar al error máximo permitido por el usuario, por lo que me daba a entender que el procedimiento que estaba realizando estaba bien y que la red estaba aprendiendo. Por otro lado, siempre trate de mantener un alto numero en las iteraciones para que la red se tomara todas las necesarias en este proceso.

7. Pruebas y simulaciones

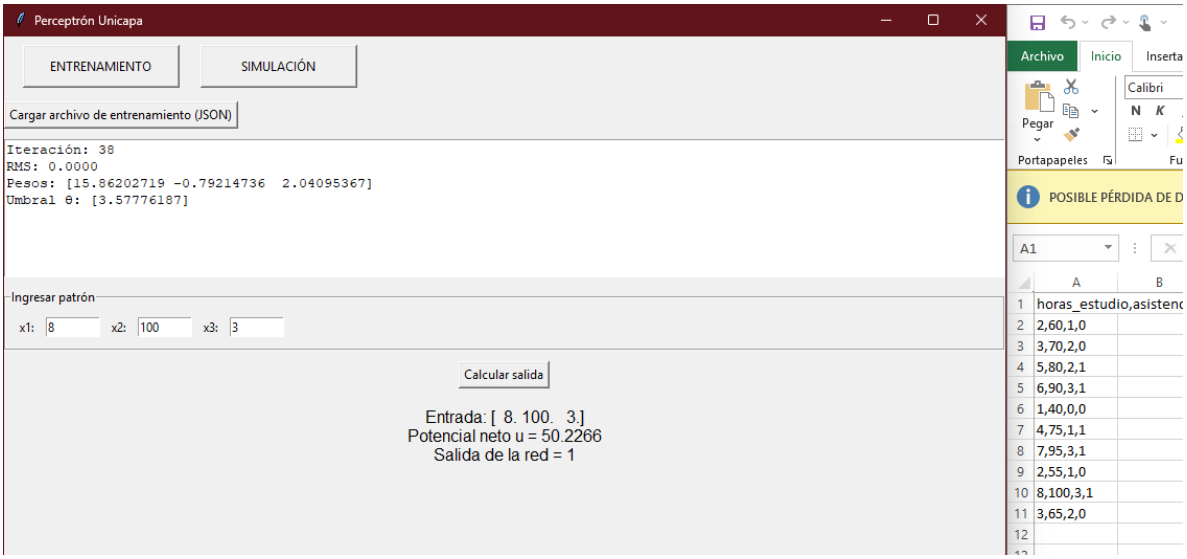
En este apartado, gracias a los pesos y umbral obtenido en cada entrenamiento (luego de haberlos conservado en un archivo aparte) de red se puede realizar una simulación para medir el correcto aprendizaje de cada una, ingresando valores deseados y propuestos por el dataset (Y_D), con la salida de la red (Y_R), aparte también probar con los valores restantes los cuales no hicieron parte del entrenamiento. A un lado se ve la UI del programa y a otro lado el dataset para una mejor visualización de los datos obtenidos.

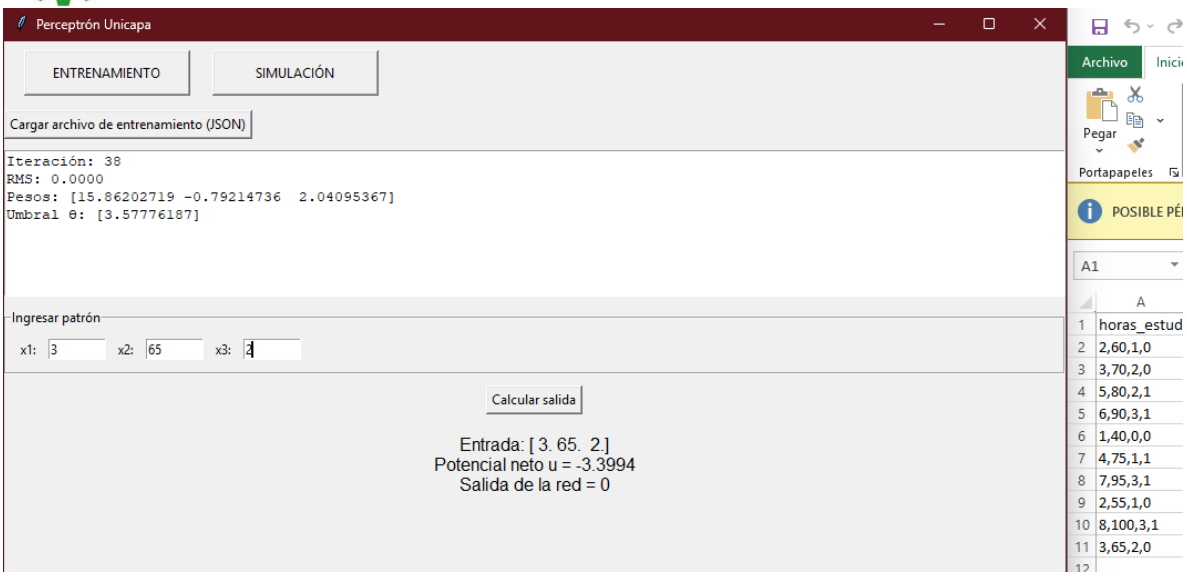
7.1 Dataset 3 con datos usados para el entrenamiento





7.2 Dataset 3 con datos NO usados para el entrenamiento





8. Conclusiones

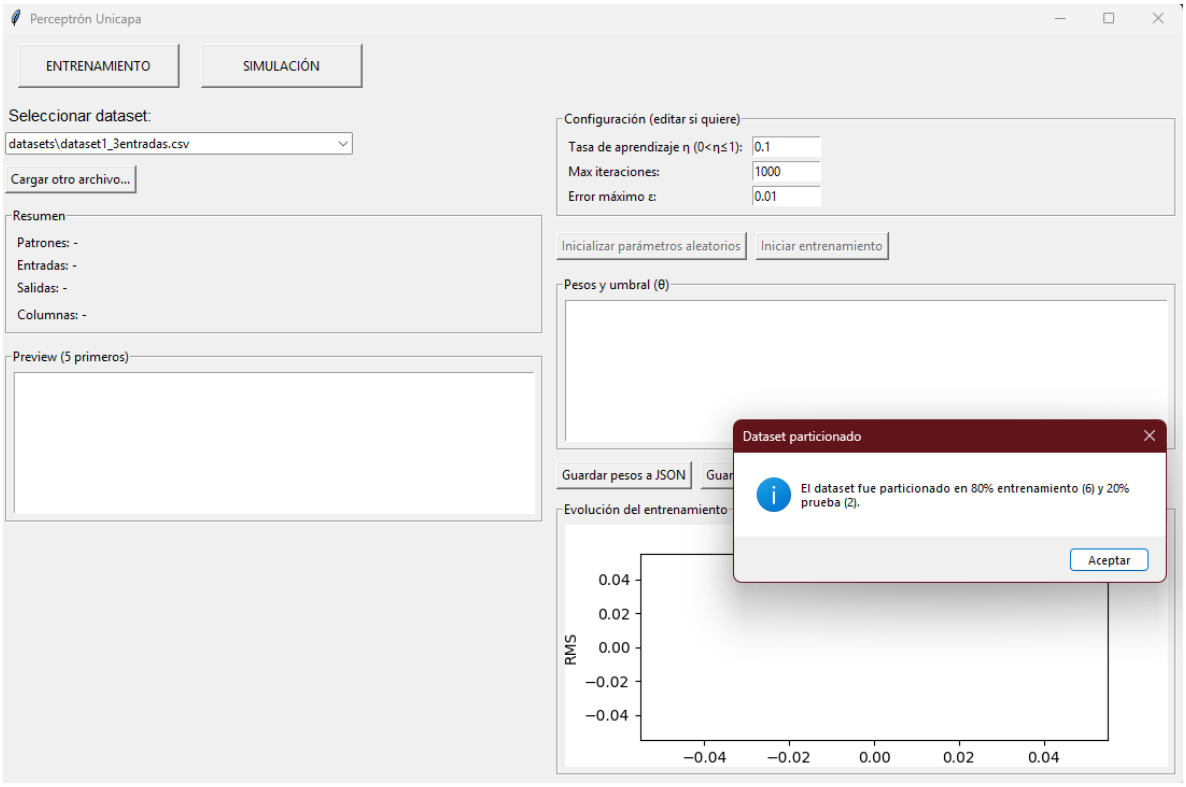
Para concluir este trabajo, primeramente en cuanto a la reflexión de la capacidad del perceptrón para resolver problemas, pues me pareció una gran manera de introducir el como funcionan las redes neuronales, al principio del trabajo, tenia una gran incertidumbre ya que no entendí el cómo iba a llegar a la salida de la red y que esta de verdad aprendiera, pero a medida de que fui investigando y entendiendo este modelo logre comprender este algoritmo, este mismo me demostró ser capaz de resolver problemas con datos binarios de manera eficiente, me gusto que mientras iba desarrollando todo iba teniendo relación directa con el paso anterior y aunque una de las dificultades fue aplicar las fórmulas propuestas de manera efectiva y precisa, leyendo las instrucciones logre completar el paso a paso. En el modelo con datos reales (dataset3) si fue necesario un ajuste mas concreto para que la salida de la red, ya sea con casos ya presentes en el dataset o datos nuevos funcionara.

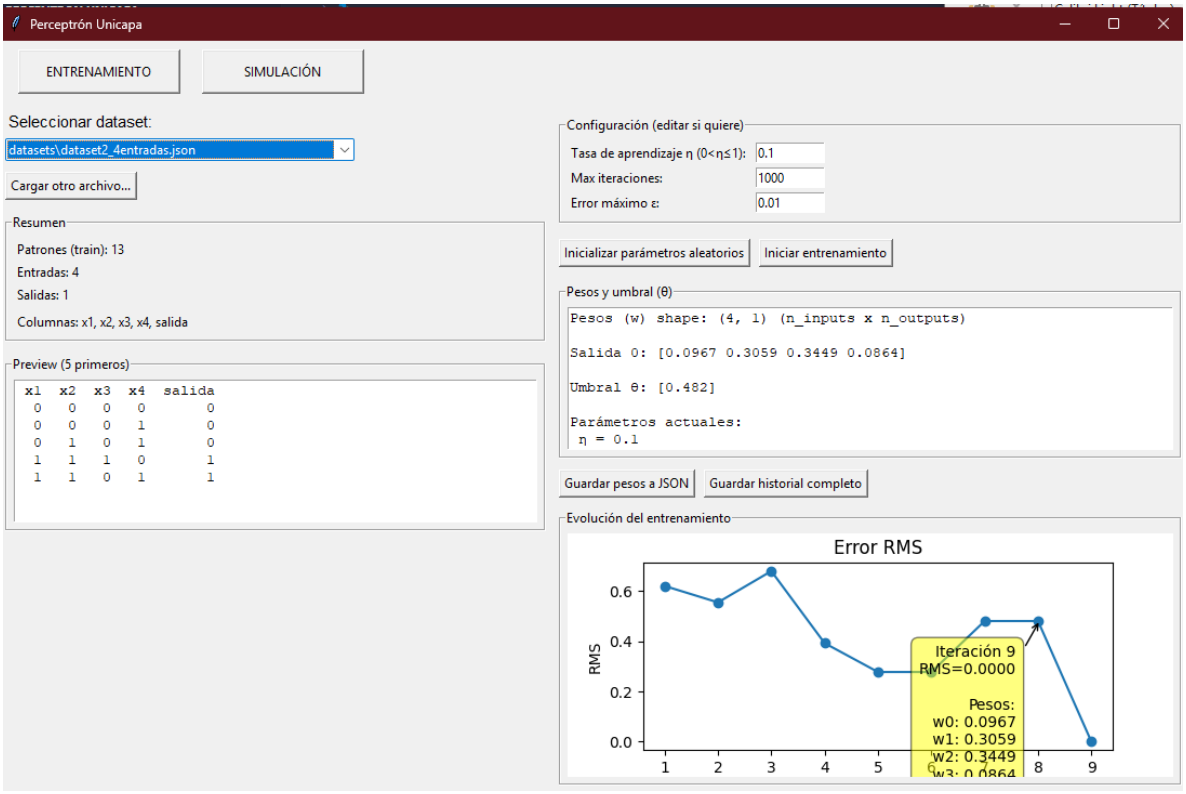
Como posibles mejoras para una futura actividad serian poder mejorar la interfaz, creo que la pude hacer mejor y poder lograr aún más presión y trabajar con más datos para mejorar la práctica.

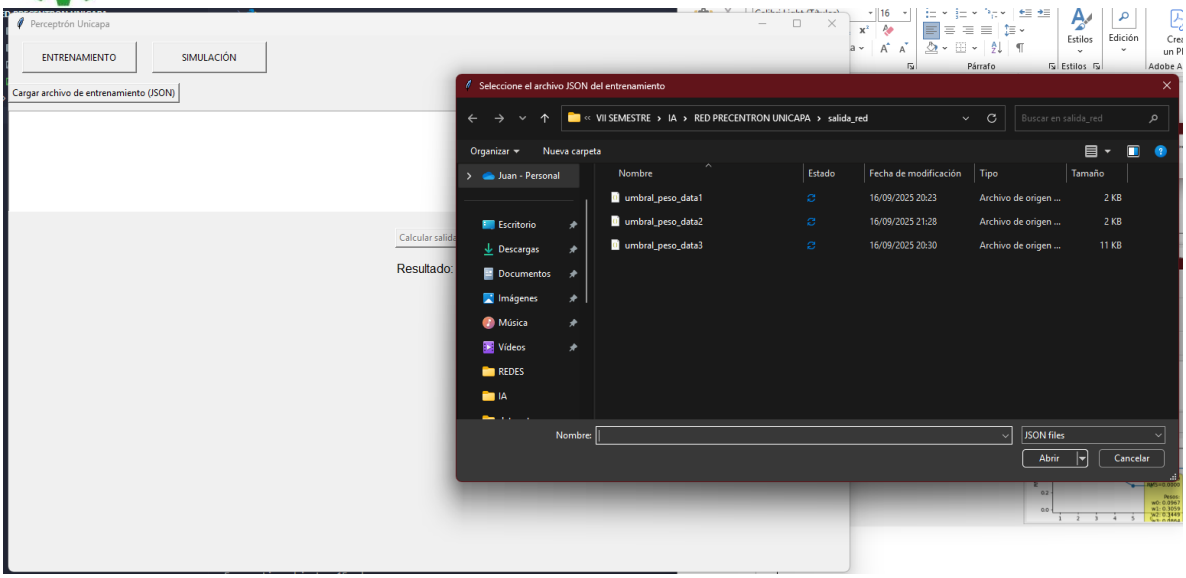
9. Anexos

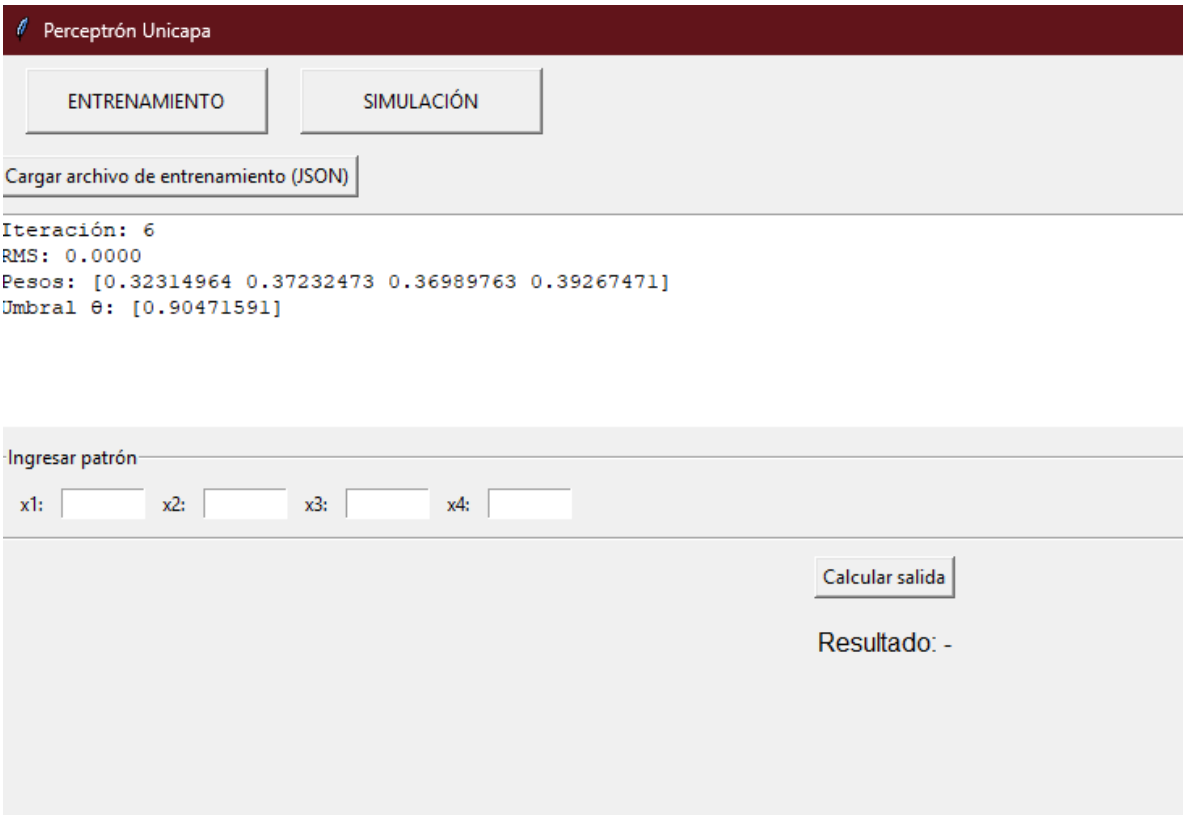
Codigo fuente:
https://github.com/juantaborda27/RED_NEURONAL_PERCETRON_SIMPLE.git

Capturas

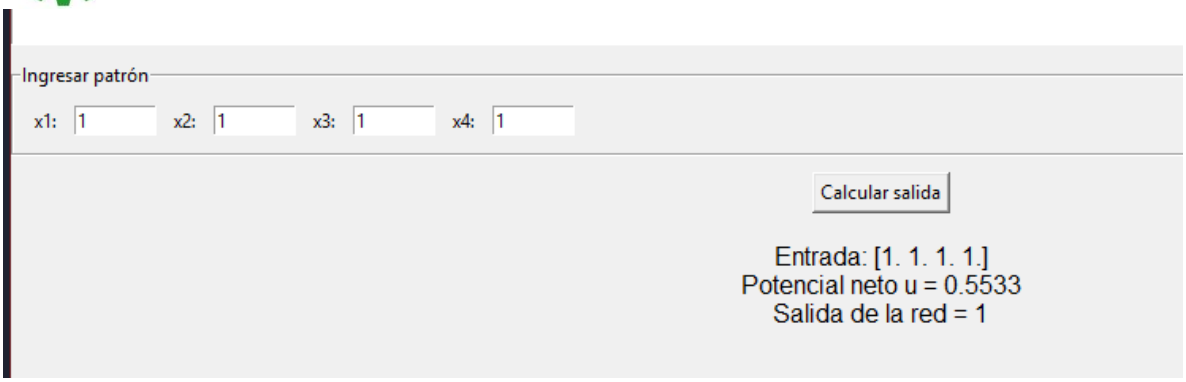

Perceptrón Unicapa
ENTRENAMIENTO
SIMULACIÓN
Seleccionar dataset:
datasets\dataset1_3entradas.csv
Cargar otro archivo...
Resumen
Patrones: -
Entradas: -
Salidas: -
Columnas: -
Preview (5 primeros)
Configuración (editar si quiere)
Tasa de aprendizaje η (0<η≤1):
0.1
Max iteraciones:
1000
Error máximo ε:
0.01
Inicializar parámetros aleatorios
Iniciar entrenamiento
Pesos y umbral (θ)
Guardar pesos a JSON
Evolución del entrenamiento
Dataset particionado
El dataset fue particionado en 80% entrenamiento (6) y 20% prueba (2).
Aceptar
RMS
0.04
0.02
0.00
−0.02
−0.04
−0.04
−0.02
0.00
0.02
0.04


Perceptrón Unicapa
ENTRENAMIENTO
SIMULACIÓN
Seleccionar dataset:
datasets\dataset2_4entradas.json
Cargar otro archivo...
Resumen
Patrones (train): 13
Entradas: 4
Salidas: 1
Columnas: x1, x2, x3, x4, salida
Preview (5 primeros)
x1 x2 x3 x4 salida
0 0 0 0 0
0 0 0 1 0
0 1 0 1 0
1 1 1 0 1
1 1 0 1 1
Configuración (editar si quiere)
Tasa de aprendizaje η (0<η≤1):
0.1
Max iteraciones:
1000
Error máximo ε:
0.01
Inicializar parámetros aleatorios
Iniciar entrenamiento
Pesos y umbral (θ)
Pesos (w) shape: (4, 1) (n_inputs x n_outputs)
Salida 0: [0.0967 0.3059 0.3449 0.0864]
Umbral θ: [0.482]
Parámetros actuales:
η = 0.1
Guardar pesos a JSON
Guardar historial completo
Evolución del entrenamiento
Error RMS
RMS
0.6
0.4
0.2
0.0
1
2
3
4
5
8
9
Iteración 9
RMS=0.0000
Pesos:
w0: 0.0967
w1: 0.3059
w2: 0.3449


Perceptrón Unicapa
ENTRENAMIENTO
SIMULACIÓN
Cargar archivo de entrenamiento (JSON)
Resultado:
Seleccione el archivo JSON del entrenamiento
VII SEMESTRE › IA › RED PRECENTRON UNICAPA › salida_red
Buscar en salida_red
Organizar
Nueva carpeta
Juan - Personal
Escritorio
Descargas
Documentos
Imágenes
Música
Vídeos
REDES
IA
Nombre
Estado
Fecha de modificación
Tipo
Tamaño
umbral_peso_data1
16/09/2025 20:23
Archivo de origen ...
2 KB
umbral_peso_data2
16/09/2025 21:28
Archivo de origen ...
2 KB
umbral_peso_data3
16/09/2025 20:30
Archivo de origen ...
11 KB
Nombre:
JSON files
Abrir
Cancelar


Perceptrón Unicapa
ENTRENAMIENTO
SIMULACIÓN
Cargar archivo de entrenamiento (JSON)
Iteración: 6
RMS: 0.0000
Pesos: [0.32314964 0.37232473 0.36989763 0.39267471]
Umbral θ: [0.90471591]
Ingresar patrón
x1:
x2:
x3:
x4:
Calcular salida
Resultado: -

X



Referencias

DataScientest. (s.f.). *Perceptrón: ¿qué es y para qué sirve?* Obtenido de https://datascientest.com/es/perceptron-que-es-y-para-que-sirve

Tahir. (17 de 03 de 2025). *Qué es el perceptrón: un modelo de red neuronal fundamental*. Obtenido de Medium: https://medium.com/@tahirbalarabe2/what-is-the-perceptron-a-foundational-neural-network-model-c722687dc51a