Aprendizaje supervisado de un perceptrón: implementación en C# y visualización del proceso de convergencia

Álvaro Peraza Garzón1, Mónica Avelina Gutiérrez Haros2, Juan Francisco Peraza Garzón3, Gloria Irene Téllez Rodríguez2, Roberto Antonio Martínez Thompson2

1Instituto Tecnológico de Mazatlán, TecNM (MÉXICO)

2Universidad Politécnica de Sinaloa (MÉXICO)

3Universidad Autónoma de Sinaloa, Facultad de Informática Mazatlán (MÉXICO)

Resumen

Este trabajo presenta una implementación didáctica y reproducible de un perceptrón de una sola capa en el lenguaje C#, con el propósito de ilustrar el proceso de aprendizaje supervisado mediante el ajuste explícito de pesos y la visualización de la frontera de decisión. Se abordan los problemas clásicos OR y AND (ambos linealmente separables) empleando el mismo conjunto de pesos iniciales. Para cada caso se genera una bitácora de entrenamiento que documenta la evolución del vector de pesos y los cambios correspondientes en la frontera de decisión a lo largo de las iteraciones. Aunque las condiciones iniciales son idénticas, el perceptrón requiere un número distinto de iteraciones para converger en cada problema, lo cual evidencia la sensibilidad del algoritmo al orden de presentación de los patrones y a su carácter incremental. Los resultados confirman la correcta convergencia del modelo y permiten visualizar de manera clara la mecánica interna del aprendizaje supervisado. La implementación propuesta constituye una herramienta formativa que facilita la comprensión del perceptrón simple y sienta las bases para el estudio de arquitecturas neuronales más avanzadas.

Palabras clave: Aprendizaje supervisado, C#, frontera de decisión, perceptrón, redes neuronales.

Abstract

This work presents a didactic and reproducible implementation of a single-layer perceptron in the C# programming language, aimed at illustrating the supervised learning process through explicit weight adjustment and the visualization of the decision boundary. The classical OR and AND classification problems (both linearly separable) are addressed using the same set of initial weights. For each case, a training log is generated to document the evolution of the weight vector and the corresponding changes in the decision boundary throughout the iterations. Although the initial conditions are identical, the perceptron requires a different number of iterations to converge in each problem, which highlights the algorithm’s sensitivity to the order in which the patterns are presented and to its incremental nature. The results confirm the correct convergence of the model and clearly illustrate the internal mechanics of supervised learning. The proposed implementation serves as an educational tool that facilitates understanding of the simple perceptron and provides a foundation for studying more advanced neural architectures.

Keywords: C#, decision boundary, neural networks, perceptron, supervised learning.

# INTRODUCción

La Inteligencia Artificial (IA) tuvo su origen formal en 1956, cuando John McCarthy acuñó el término durante la conferencia de Dartmouth College en Hanover, Estados Unidos [1]. Desde entonces, la IA ha experimentado diversas etapas de avance y estancamiento. En 1958, Frank Rosenblatt [2] propuso el perceptrón en el Cornell Aeronautical Laboratory, como un modelo computacional inspirado en las neuronas biológicas orientado al reconocimiento de patrones.

Este modelo representó uno de los primeros intentos de emular el aprendizaje humano mediante redes neuronales artificiales. No obstante, hacia 1969, Minsky y Papert [3] demostraron sus limitaciones teóricas, especialmente la incapacidad para resolver problemas no linealmente separables, como la función XOR.

Dichas restricciones provocaron una pausa en el desarrollo de las redes neuronales, periodo conocido como “el invierno de la IA”, hasta su resurgimiento en la década de 1980 con el perceptrón multicapa y el algoritmo de retropropagación [4]. El perceptrón simple continúa siendo un modelo fundamental para comprender el funcionamiento interno de las redes neuronales modernas. A través de un algoritmo de aprendizaje supervisado, ajusta los pesos de sus entradas en función de los errores de clasificación, lo que lo hace adecuado para tareas binarias y como base conceptual para arquitecturas más complejas.

En este contexto, el presente trabajo tiene como objetivo ofrecer una base académico-científica que sirva como guía para la implementación de un perceptrón en el lenguaje C#, con el propósito de ilustrar de manera didáctica el proceso de aprendizaje supervisado y el ajuste progresivo de pesos. Esta implementación busca facilitar la comprensión del modelo para estudiantes y desarrolladores, y funcionar como punto de partida para el estudio y desarrollo de redes neuronales más avanzadas.

# FUNDAMENTOS TEÓRICOS

## Perceptrón

El perceptrón es una unidad de procesamiento elemental dentro de las redes neuronales artificiales. Su funcionamiento se basa en la combinación lineal de entradas ponderadas, seguida de la aplicación de una función de activación. Matemáticamente, el modelo se expresa como [5]:

donde:

* representa las entradas del modelo
* los pesos sinápticos asociados a cada entrada
* : el sesgo
* : la suma ponderada de la neurona
* la salida de la neurona
* la función de activación, definida como una función escalón binaria:

El objetivo del entrenamiento consiste en ajustar los valores de y para minimizar el error de clasificación en un conjunto de ejemplos etiquetados.

## Aprendizaje Supervisado

El perceptrón utiliza un **algoritmo de aprendizaje supervisado** iterativo. En cada ciclo de entrenamiento, se calcula la salida para una entrada y se compara con la salida esperada . Si existe error, se actualizan los pesos conforme a la regla [6]:

donde:

* es la tasa de aprendizaje (learning rate)
* es la salida esperada (ground truth)
* es la salida obtenida por el perceptrón
* es la entrada correspondiente al peso

Esta regla ajusta los pesos cada vez que el perceptrón comete un error, acercando la salida al valor esperado. El proceso se repite para todos los ejemplos del conjunto de entrenamiento hasta alcanzar un error mínimo o un número máximo de iteraciones.

## Algoritmo de aprendizaje

Es importante comprender que el perceptrón ajusta sus pesos de manera iterativa según los errores cometidos durante el entrenamiento. Este proceso permite que el modelo refine progresivamente su frontera de decisión, acercándose a la solución.

La siguiente explicación detalla precisamente cómo se realizan estos ajustes y bajo qué condiciones converge el perceptrón hacia una clasificación correcta [7], [8]:

* Dado un vector de entrada
* Y un vector de pesos
* Se calcula la salida (ver apartado 2.1)
  + Se calcula la suma ponderada de las entradas
  + La suma se envía a la función escalón, si la salida de la función es distinta a la salida esperada, se activa el mecanismo de ajuste de pesos (ver apartado 2.2).

### Frontera de decisión

La frontera de decisión está dada por la ecuación lineal de la suma ponderada (), donde usualmente en el perceptrón .

entonces la ecuación se interpreta como [9]:

o equivalentemente a (en su forma ordenada al origen):

## Lenguaje C#

C# es un lenguaje de programación moderno y orientado a objetos que destaca por su claridad, estructura y capacidad para desarrollar software altamente organizado. Diseñado por Microsoft dentro del ecosistema .NET, C# ofrece una sintaxis coherente y un fuerte tipado que facilitan la creación de algoritmos bien estructurados, cualidades esenciales en el desarrollo de modelos computacionales como perceptrones y redes neuronales artificiales.

Su arquitectura favorece el trabajo modular, promoviendo la separación lógica del código y el uso eficiente de funciones, clases y estructuras de datos. Además, su robusto conjunto de bibliotecas y herramientas permite implementar procesos matemáticos, manipular vectores y definir funciones o flujos de entrenamiento de manera ordenada. Gracias a estas características, C# se convierte en una opción sólida para quienes buscan desarrollar soluciones de inteligencia artificial desde un enfoque didáctico, limpio y reproducible, manteniendo un control preciso sobre cada etapa del algoritmo [10].

# METODOLOGÍA

El presente trabajo adopta un enfoque experimental y explicativo, orientado a demostrar el proceso de aprendizaje supervisado de un perceptrón simple mediante su implementación computacional en el lenguaje C#. El diseño metodológico se fundamenta en la programación desde cero, evitando el uso de bibliotecas externas de aprendizaje automático, con el propósito de evidenciar de manera explícita cada paso del proceso de entrenamiento, ajuste de pesos y evaluación del modelo.

## Datos de entrenamiento

Se utilizaron las tablas de verdad OR (*Tabla 2*) y AND (*Tabla 2*) como elementos de entrenamiento, estas al ser linealmente separables en un plano bidimensional (Figura 1), son los conjuntos de datos idóneos para los primeros experimentos en la exploración del aprendizaje de un perceptrón.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Tabla 1. Tabla de verdad OR.*   |  |  |  | | --- | --- | --- | | X1 | X2 | Salida Esperada | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 1 | | *Tabla 2. Tabla de verdad AND.*   |  |  |  | | --- | --- | --- | | X1 | X2 | Salida Esperada | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 | |

Se muestra la graficación de los valores OR y AND, entradas x1, x2 corresponden a los ejes x, y. La línea punteada indica la separabilidad lineal de ambos conjuntos:

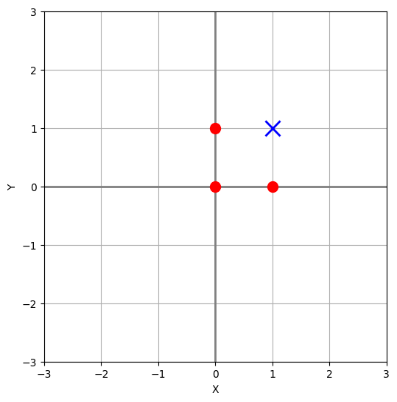
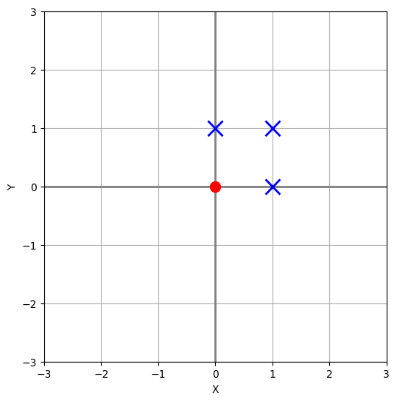


Figura 1. Las salidas de OR y AND (X para 1, **•** para 0).

## Procedimiento

Como ya se mencionó anteriormente (ver apartado 2.3 Algoritmo de aprendizaje), los pasos para el aprendizaje de una neurona siguiendo los fundamentos teóricos descritos y las instrucciones correspondientes en C#, quedan como sigue:

1. Vector de pesos iniciales, los valores oscilarán aleatoriamente entre -1 y 1.
2. Vector de entrada: Tabla del OR
   * Cada renglón de la tabla formará el vector de entrada: etc, con la forma , cabe señalar que se incluye para habilitar el peso como bias.
3. La suma ponderada de la entradas se obtiene de la multiplicación de los vectores de entrada por sus pesos correspondientes:

\* =

1. El resultado de la suma se envía a la función de activación (Escalón), esto resultaría en la salida (binaria) de la neurona .
2. La tasa de aprendizaje se define como: la diferencia entre la salida esperada menos la obtenida , se representa como , por tanto:
   * Si la salida esperada es igual a la obtenida , se vuelve al paso 2 (siguiente vector de entrada).
     + De lo contrario, se activa el mecanismo de ajuste de pesos (se generará un nuevo vector de pesos), antes de volver al paso 2.
3. El aprendizaje se detiene cuando cada uno de los vectores de entrada genere una salida igual a la esperada (utilizando todos ellos el mismo vector de pesos).

Tomando lo anterior como base algorítmica, se definen a continuación los pasos a seguir:

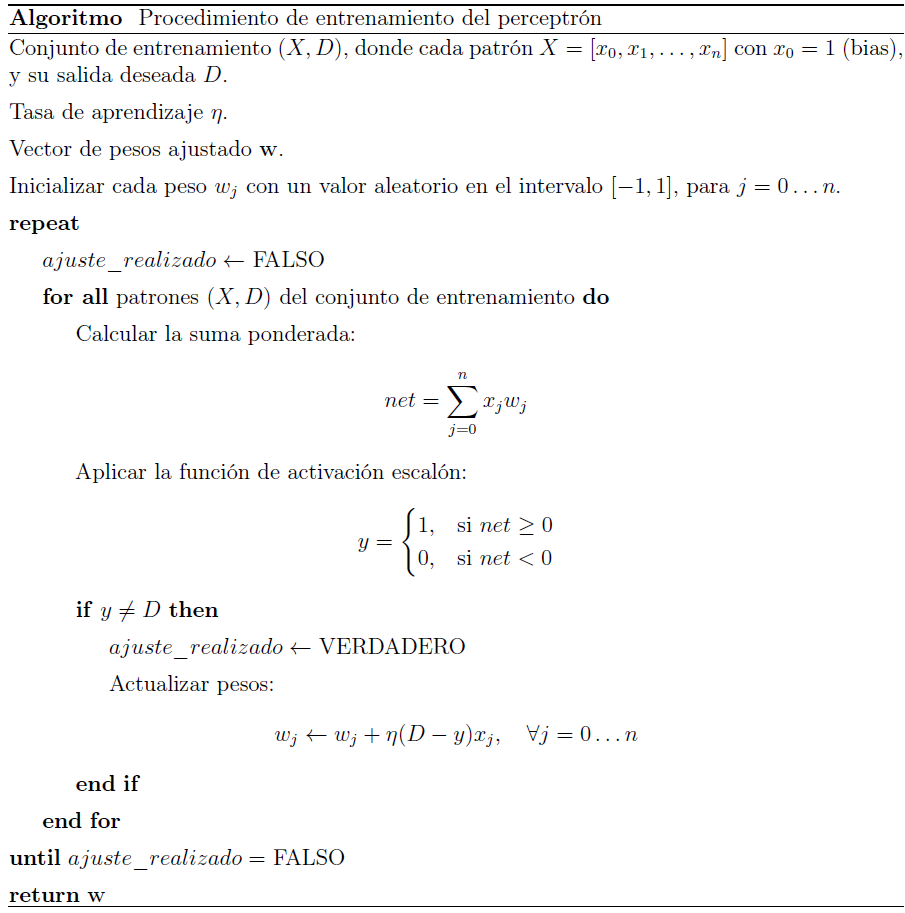


Figura 2. Algoritmo para el entrenamiento del perceptrón.

La Figura 2 presenta el procedimiento clásico de entrenamiento del perceptrón. En él se establecen los elementos fundamentales del modelo: el conjunto de entrenamiento , el vector de pesos , la tasa de aprendizaje , y la función de activación tipo escalón. El algoritmo sigue un enfoque iterativo donde, para cada patrón del conjunto de entrenamiento, se calcula primero la suma ponderada (, posteriormente este valor se evalúa mediante la función escalón, produciendo la salida binaria . Si la salida del perceptrón difiere del valor esperado , el algoritmo activa el mecanismo de aprendizaje y ajusta los pesos. Este proceso se repite hasta que no se realice ningún ajuste, indicando que el perceptrón ha clasificado correctamente el conjunto de entrenamiento.

La Figura 3 muestra la implementación directa en C# del algoritmo descrito en la Figura 2. El código reproduce fielmente cada una de las etapas del pseudocódigo, pero ahora en un entorno programable y ejecutable.

En primer lugar, la función *entrena()* recibe como parámetros el vector de pesos iniciales y la matriz de patrones. Dentro del ciclo do-while se utiliza una bandera de control (*ajuste\_realizado*). Al comenzar cada iteración, esta bandera se reinicia a false. Luego, el código recorre todos los patrones del conjunto de entrenamiento mediante un ciclo for. Para cada uno de ellos se calcula la suma ponderada (*Netj*) siguiendo exactamente la expresión mostrada en la Figura 2. Una vez calculado el valor neto, se aplica la función de activación escalón, utilizando un operador ternario que asigna 0 o 1 dependiendo del valor *Netj*.

Por último, se valida si la salida de la neurona no coincide con los esperado, el mecanismo de aprendizaje actualiza cada peso . Si se realizó algún ajuste, la bandera *ajuste\_realizado* se coloca en *true*, provocando que el ciclo do-while continúe. El proceso termina únicamente cuando en una iteración completa no se realizan ajustes, momento en el cual los pesos representan la solución aprendida por el perceptrón, y el método devuelve el vector final *w*.

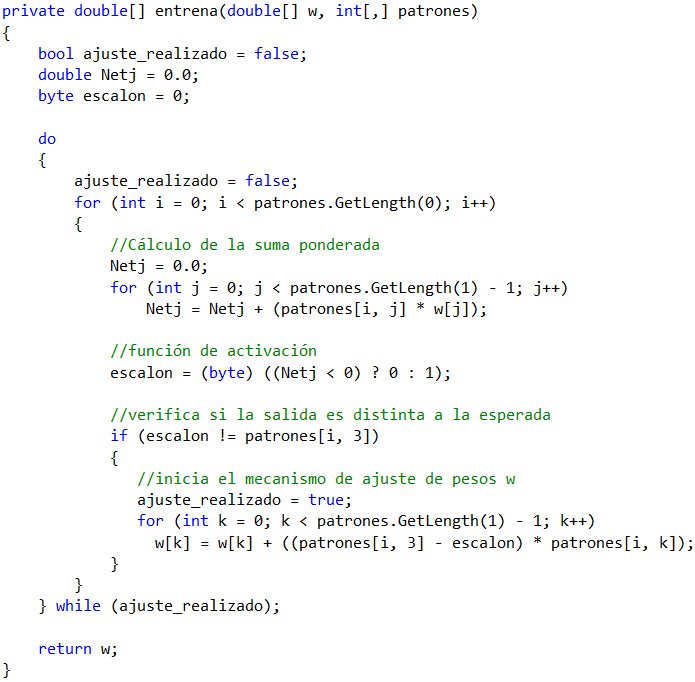


Figura 3. Implementación del algoritmo en C#

El llamado al procedimiento *entrena()* (Figura 4) se realizó utilizando la función *Clone()* del vector w, con el propósito de conservar intactos los pesos iniciales. Esto no es obligatorio y depende del criterio de quien implementa el modelo; sin embargo, en este trabajo se decidió emplearlo por razones didácticas, ya que permite mantener separados los valores iniciales, de los pesos finales (w\_finales) obtenidos tras el proceso de entrenamiento.



Figura 4. Llamado al procedimiento utilizando Clone().

A continuación se presentan los resultados obtenidos al ejecutar el procedimiento descrito, tanto para la tabla OR como para la tabla AND, con el fin de analizar el comportamiento del perceptrón durante su proceso de convergencia.

# RESULTADOS

La ejecución del algoritmo generó la bitácora de entrenamiento del perceptrón (Figura 5), en la cual se detallan los cálculos realizados y los ajustes efectuados a los pesos en cada iteración del proceso de aprendizaje. Esta bitácora permite observar de manera explícita la evolución del vector de pesos a lo largo de las distintas vueltas (o iteraciones), así como la relación entre la salida obtenida y la salida esperada para cada patrón de entrada. El registro muestra que fueron necesarias cuatro iteraciones para que el perceptrón alcanzara la convergencia, es decir, para que todos los patrones de la tabla de verdad del OR fueran clasificados correctamente sin requerir nuevos ajustes. Este proceso evidencia el funcionamiento incremental del aprendizaje supervisado en el perceptrón, donde los pesos se modifican únicamente cuando la salida producida difiere de la salida deseada. La convergencia final confirma que el conjunto de pesos resultante es capaz de separar linealmente los datos del OR, validando así la correcta aplicación del algoritmo de entrenamiento.

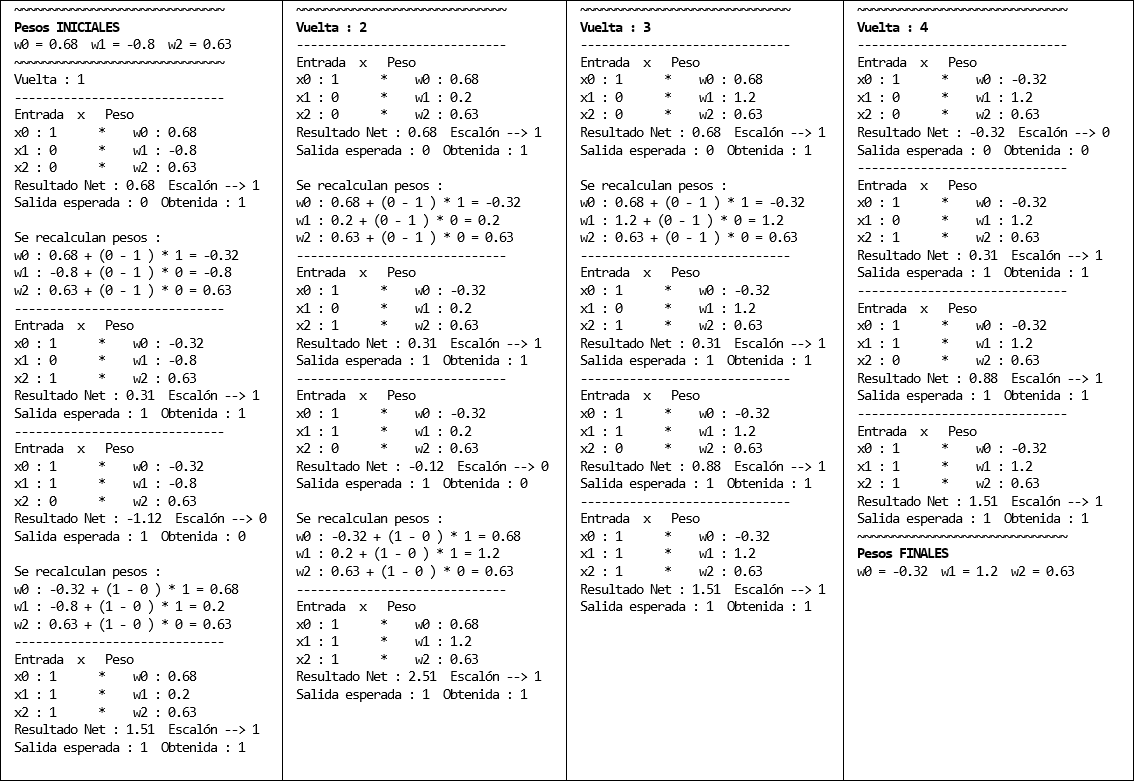


Figura 5. Bitácora de entrenamiento del perceptrón para el aprendizaje de la tabla OR.

La visualización de la frontera de decisión (ver apartado 2.3.1) en cada ajuste de pesos (Figura 6) muestra las variaciones que adopta el perceptrón en cada iteración del proceso de entrenamiento. La recta etiquetada como “inicial” corresponde a la frontera generada a partir de los pesos asignados antes de comenzar el aprendizaje. En color rojo se observa la primera vuelta, junto con los dos ajustes de pesos realizados en esa iteración. Las etiquetas siguen el formato vuelta# – ajuste#, abreviado como v1 – a1 para referirse al primer ajuste de la primera vuelta, y así sucesivamente para las iteraciones posteriores. En azul se representa la segunda vuelta y en verde la tercera; esta última es la que establece los pesos finales, los cuales no sufren modificaciones en la cuarta vuelta, alcanzándose así la condición de parada del algoritmo.

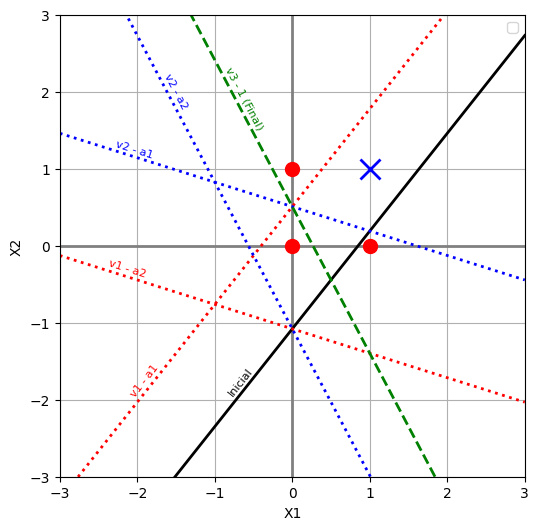


Figura 6. Ajustes en las fronteras de decisión durante el entrenamiento para el aprendizaje del OR

La Figura 7 presenta la bitácora correspondiente al entrenamiento del perceptrón utilizando la tabla de verdad del AND. Con los mismos pesos iniciales empleados previamente, se observa cómo el modelo los ajusta de manera progresiva ante cada desacierto, siguiendo el procedimiento de aprendizaje supervisado ya descrito. En este caso, el perceptrón requiere un mayor número de iteraciones para alcanzar la convergencia. A lo largo del registro se aprecia que los pesos se modifican únicamente cuando la salida obtenida difiere de la esperada, conforme a la regla de actualización del algoritmo. Este proceso culmina en un conjunto de pesos finales capaz de clasificar correctamente todos los patrones sin necesidad de ajustes adicionales, alcanzando la condición de parada. La bitácora ofrece así una evidencia clara del comportamiento incremental del modelo y del papel de las correcciones sucesivas para lograr una separación lineal adecuada en el caso del operador lógico AND.

También para este caso se representan gráficamente los cambios en la frontera de decisión (Figuras 8 y 9) que aparecen en la bitácora correspondiente al entrenamiento del perceptrón con la tabla AND. Debido al número de ajustes realizados, y con el fin de mejorar la legibilidad, las figuras se organizan en tres gráficas parciales que ilustran de manera progresiva la evolución del modelo: en (a) se muestran las primeras dos vueltas, cada una con dos ajustes; en (b) se representan los ajustes efectuados durante la tercera y cuarta vuelta; y en (c) se observa la convergencia alcanzada en la sexta vuelta, seguida de la ausencia de modificaciones en la séptima. La línea final, etiquetada como “v6–a1” (vuelta 6 – ajuste 1), evidencia la correcta separación lineal entre ambas clases. Finalmente, en (d) se integran todos los cambios registrados durante el entrenamiento, ofreciendo una vista global del proceso de ajuste de pesos.

Aunque para el caso del operador AND se emplearon exactamente los mismos pesos iniciales que en el entrenamiento del OR, el perceptrón requirió un mayor número de ajustes antes de alcanzar la convergencia. Esta diferencia no se debe a que el AND posea un área separable más restrictiva, sino a la naturaleza aleatoria de los pesos iniciales combinada con el orden en que se presentan los patrones durante el aprendizaje. En otras palabras, se sigue el mismo procedimiento de corrección incremental en ambos casos, pero la trayectoria específica que recorren los pesos para llegar a la solución depende del punto de partida y de los errores que se cometen en cada iteración.

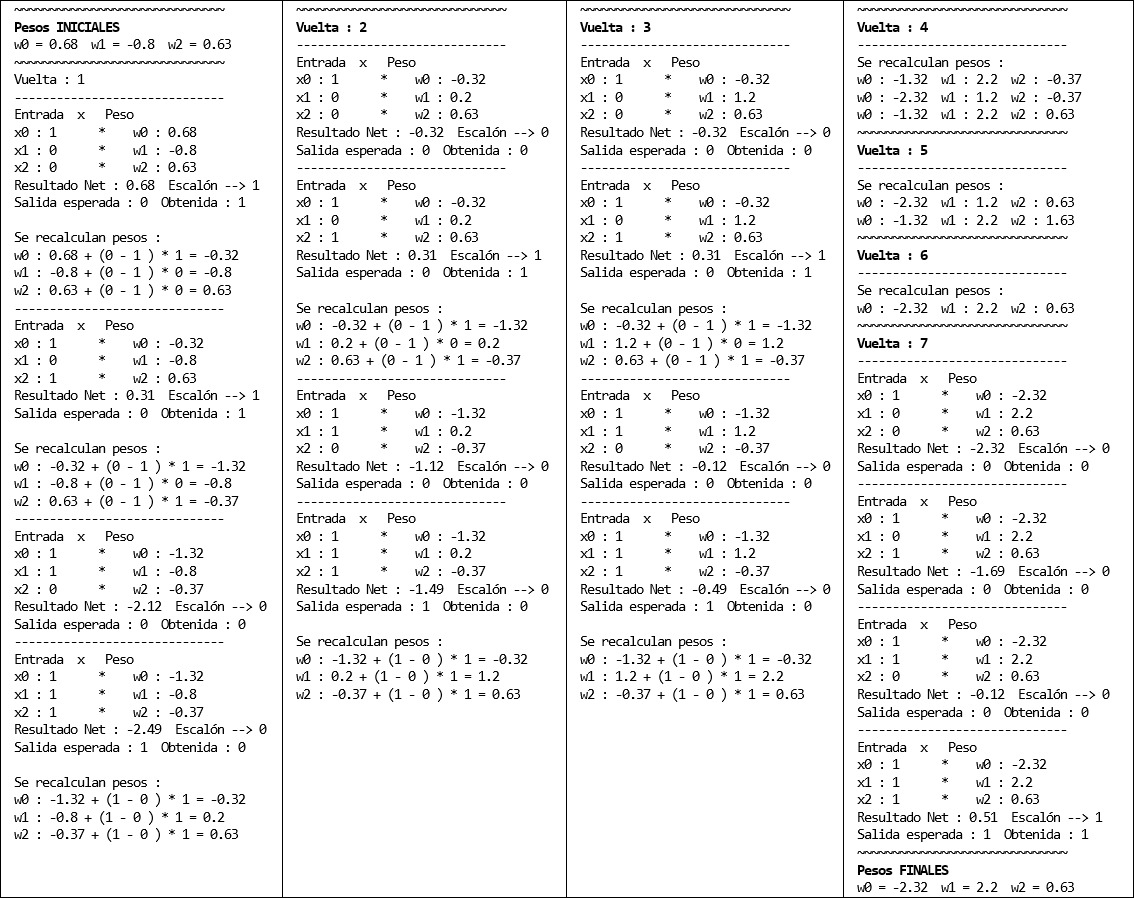


Figura 7. Bitácora de entrenamiento del perceptrón para el aprendizaje de la tabla AND.

|  |  |
| --- | --- |
| (a) | (b) |

Figura 8. Ajustes en las fronteras de decisión en las primeras 4 vueltas para el entrenamiento del AND.

|  |  |
| --- | --- |
| (c) | (d) |

Figura 9. Ajustes en las fronteras de decisión durante las 3 últimas vueltas y el concentrado.

# CONCLUSIONes

## Discusión

Los resultados obtenidos permiten analizar el comportamiento del perceptrón simple al enfrentar dos problemas clásicos de clasificación linealmente separable: las tablas OR y AND. En ambos casos, el modelo logra converger a un conjunto de pesos capaz de separar correctamente las clases, lo cual confirma la aplicabilidad del algoritmo de aprendizaje supervisado descrito. No obstante, se observan diferencias en la dinámica de convergencia.

La Figura 7 muestra el comportamiento del perceptrón durante el entrenamiento con la tabla del AND, donde el número de iteraciones requeridas para alcanzar la convergencia fue mayor que en el caso del OR. Esta diferencia no se debe a una mayor complejidad geométrica del conjunto AND, ya que ambos problemas son linealmente separables con márgenes comparables. La variación en el número de ajustes responde, más bien, a la sensibilidad del perceptrón simple frente a los pesos iniciales y al orden secuencial en que se presentan los patrones.

Tal como se observa en la bitácora, los pesos se actualizan únicamente cuando la salida producida difiere de la esperada, lo que da lugar a trayectorias distintas en el espacio de soluciones según las condiciones iniciales. En el caso del AND, los pesos iniciales indujeron un recorrido más prolongado antes de estabilizarse, mientras que para el OR la convergencia fue más directa. Este comportamiento es consistente con la naturaleza incremental del algoritmo y evidencia su dependencia de las configuraciones iniciales, aun tratándose de problemas igualmente separables.

## Conclusiones

El trabajo presentado demuestra de manera didáctica y reproducible el proceso de aprendizaje supervisado de un perceptrón simple, desde su formulación teórica hasta su implementación computacional en C#. La experimentación con las tablas OR y AND permite evidenciar el mecanismo iterativo de ajuste de pesos y la importancia de la linealidad en la separabilidad de los datos.

La bitácora de entrenamiento y las visualizaciones de las fronteras de decisión ofrecen una comprensión profunda del comportamiento del modelo, facilitando su estudio para estudiantes y desarrolladores que se inician en el campo de las redes neuronales. Asimismo, la implementación desde cero sin bibliotecas externas muestra que es posible construir modelos de inteligencia artificial de manera controlada, entendiendo explícitamente cada fase del algoritmo.

Como línea futura, este trabajo sienta las bases para explorar extensiones naturales del perceptrón, como el perceptrón multicapa y el algoritmo de retropropagación, así como para experimentar con funciones de activación alternativas y problemas no linealmente separables. La metodología presentada constituye, por tanto, un punto de partida sólido para profundizar en la comprensión y desarrollo de arquitecturas neuronales modernas.

referenciaS

|  |  |
| --- | --- |
| [1] | J. McCarthy, M. Minsky, N. Rochester y C. Shannon, «A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,» 31 08 1955. [En línea]. Available: https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html. [Último acceso: 24 02 2025]. |
| [2] | F. Rosenblatt, «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,» *Psychological Review,* pp. 386-408, 1958. |
| [3] | M. Wooldridge, A Brief History of Artificial Intelligence: What It Is, Where We Are, and Where We Are Going, Flatiron Books, 2021. |
| [4] | G. E. H. &. R. J. W. David E. Rumelhart, «Learning representations by back-propagating errors,» *Nature 323,* p. 533–536, 1986. |
| [5] | C. M. Bishop, Pattern Recognition and Machine Learning, Cambridge, UK: Springer, 2006. |
| [6] | C. M. Bishop, Neural Networks for Pattern Recognition, Birmingham, UK: Clarendon Press - Oxford, 1995. |
| [7] | C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook, NY, USA: Springer, 2018. |
| [8] | K. Gurney, An Introduction to Neural Networks, London: UCL Press, 1997. |
| [9] | S. Haykin, Neural Networks and Learning Machines 3rd ed., Ontario, Canada: Pearson Prentice Hall, 2009. |
| [10] | Microsoft, «Microsoft Learn - Tour por C#,» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/. [Último acceso: 01 10 2025]. |