Redes Neuronales - Perceptrón Simple

Sistemas de Inteligencia Artifical - ITBA

Carlos Sessa Lucas Pizzagalli Nicolás Purita

Resumen

Se implementó una red neuronal con aprendizaje supervisado que resuelve los operadores *OR* y *AND* lógicos, donde posee N (con $2 \leq N \leq 5$) entradas con *1* salida.

Desarrollo

La implementación del perceptrón simple se separa en tres funciones principales:

$$[w_{out}] = learn(g, g', w, input, expected, \eta) \quad (1)$$

$$[output] = calculate(g, w, input) \quad (2)$$

$$[error] = calculateError(g, w, input, expected) \quad (3)$$

Donde g es la función de activación, *g'* su derivada, *input* son los patrones utilizados para el entrenamiento del perceptrón, *expected* es la salida esperada, η es la constante de proporcionalidad de aprendizaje y w el vector de pesos.
La función 1 se utiliza para entrenar el perceptrón simple con un determinado patrón y retorna el vector de pesos correspondiente al mismo.
La función 3 calcula el error en una época en sentido de cuadrados mínimos. Esta función es la encargada de determinar si es necesario continuar con el entrenamiento de la red, la decisión es tomada en base a un cota de error determinada.
La función 2 es la encargada de calcular la salida de la red una vez finalizado el entrenamiento.
Las pruebas se realizan variando la función de activación, el η y manteniendo una cota de error de 0.01. Cabe destacar que el orden en que se muestran los patrones a la red es aleatorio.

Problemas encontrados

En un principio tuvimos inconvenientes por la utilización de los números binarios para la representación de la entrada y salida esperada. Esto se notó en la corrección de los pesos ya que en ciertos casos la variación era 0. La causa de este problema es que la entrada es 0 generando que toda la variación de un peso dé 0. Para solucionarlo se realiza una transformación de entrada, para que todos los valores en 0 sean mapeados a -1.
Otro problema encontrado se deriva del anterior, ya que la función de activación exponencial al tener en la entrada un -1 no provocaba ninguna variación en el error, es decir, se saturaba la red neuronal. Para solucionarlo evitamos la transformación de la entrada.

Por otro lado, como luego se observará en los resultados, la red se saturaba en valores incorrectos al utlizar algunas funciones de activación, por lo que nunca se alcanzaba un estado satisfactorio. Es por esto, que se debió fijar una política de corte. La misma se basa en que si el error es muy grande (en nuestro caso se decidió $error > 10\epsilon$), pero el error varía muy poco entre las distintas épocas, entonces significa que la red se encuentra saturada en una configuración errónea. Teniendo todo esto en cuenta, se decidio por la siguiente condición de corte:

$$error\_var < 0.0000001 \&\& -0.09 < error - \epsilon > 0.1 \quad (4)$$

donde *error_var* es la diferencia cuadrada entre el promedio de las últimas 200 épocas y el error de la época actual.

Resultados obtenidos

Como se puede observar, no fue posible aprender ninguno de los operadores con la función lineal, sin embargo, el objetivo fue alcanzado por el resto de las funciones. Por otro lado, al utilizar la función de activación escalón, se llega a la solución rapidamente, mientras que utlizando la sigmoidea, el resultado es alcanzado más lentamente. Otro dato apreciable, es que por lo general, al utilizar un η mayor, se obtiene un aprendizaje más rápido.

Conclusiones

A simple vista se puede observar, que los comportamientos de todas las funciones de activación son muy similares tanto en el aprendizaje de la función *OR* como la del *AND*, ya que son problemas muy similares y linealmente separables. También se puede observar, que a mayor cantidad de entradas, le es mas difícil a la red aprender lo deseado, debido a que hay mas variables a las que atender. Como se puede observar en los gráficos de las que se ulitizó la función lineal, no se logra el aprendizaje con el error querido, esto se debe a que dicha función no posee transformaciones apreciables para poder corregir los pesos de forma adecuada. Por último, es importante destacar, que los ηs más grandes aceleran el aprendizaje sin mayores consecuencias debido a que se trata de problemas muy sencillos, en caso contrario, podrían causar estancamiento.

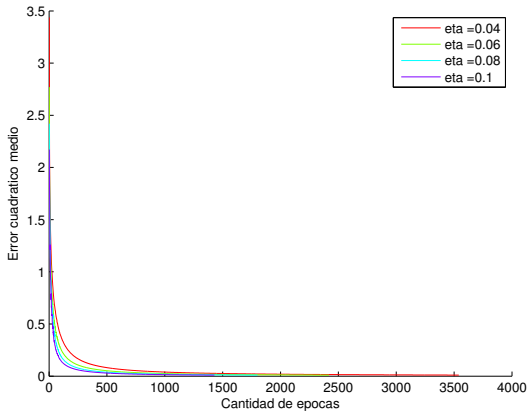

Figura 1: Función AND, N = 5 y función de activación $g(h) = tanh(\beta h)$

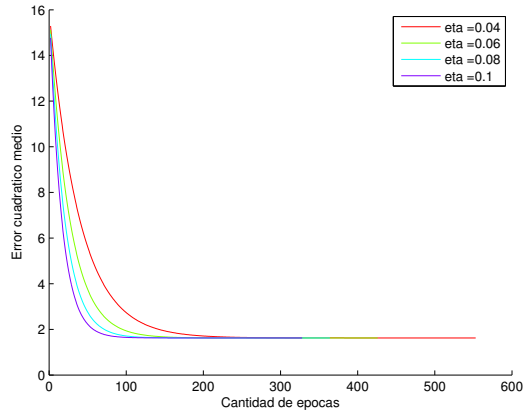

Figura 2: Función AND, N = 5 y función de activación $g(h) = h$

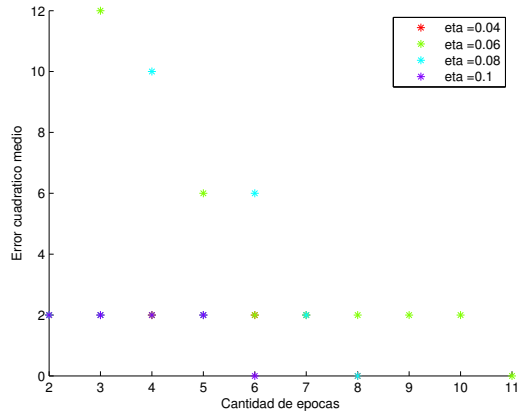

Figura 3: Función AND, N = 5 y función de activación $g(h) = sgn(h)$

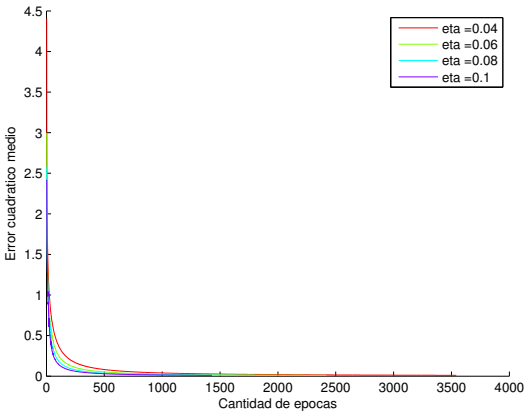

Figura 4: Función OR, N = 5 y función de activación $g(h) = tanh(\beta h)$

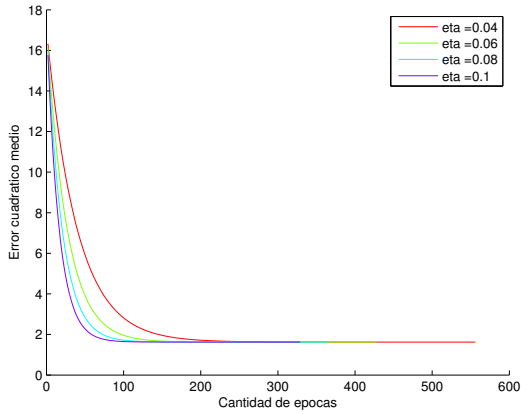

Figura 5: Función OR, N = 5 y función de activación $g(h) = h$

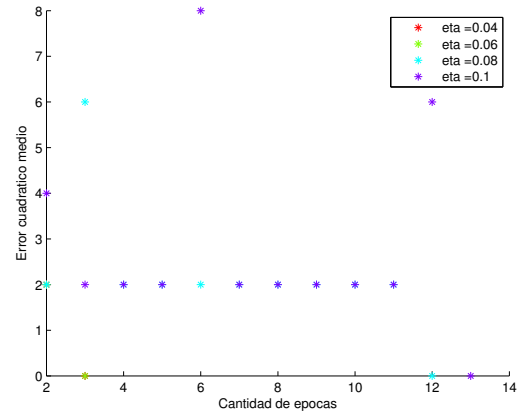

Figura 6: Función OR, N = 5 y función de activación $g(h) = sgn(h)$

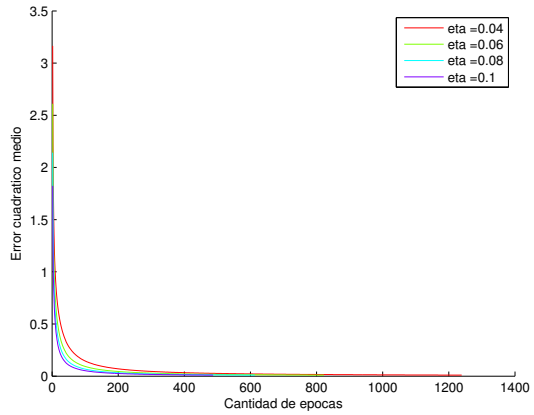

Figura 7: Función AND, N = 3 y función de activación $g(h) = tanh(\beta h)$

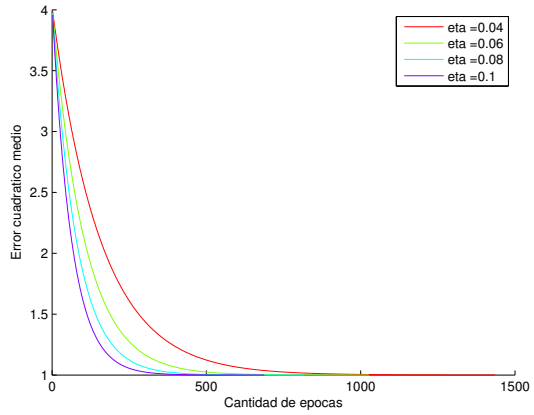

Figura 8: Función AND, N = 3 y función de activación $g(h) = h$

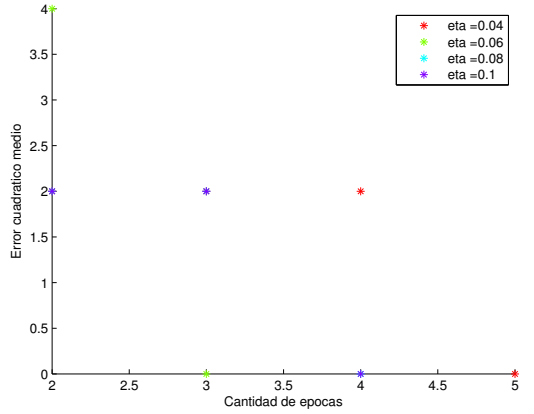

Figura 9: Función AND, N = 3 y función de activación $g(h) = sgn(h)$

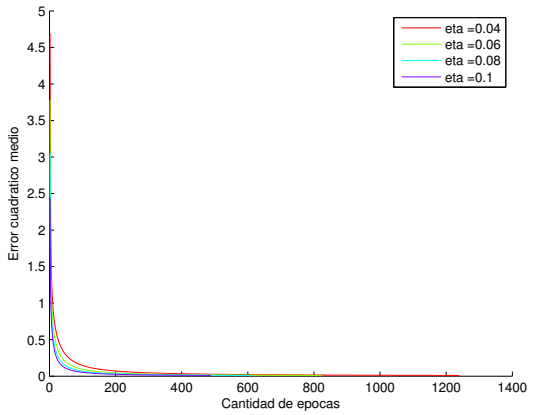

Figura 10: Función OR, N = 3 y función de activación $g(h) = tanh(\beta h)$

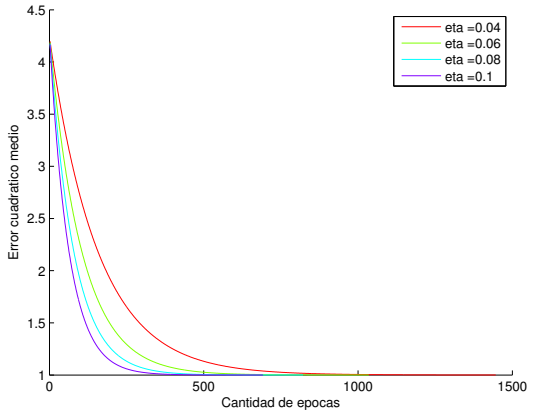

Figura 11: Función OR, N = 3 y función de activación $g(h) = h$

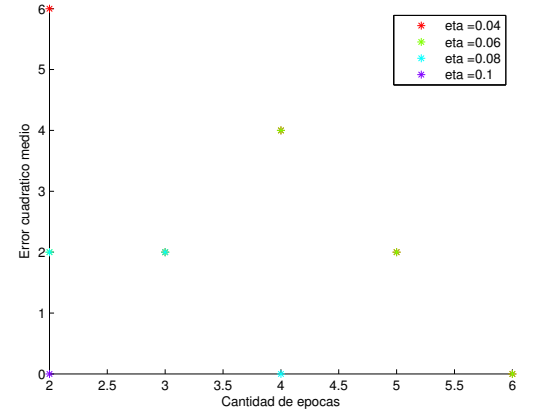

Figura 12: Función OR, N = 3 y función de activación $g(h) = sgn(h)$