Retropropagación en el Perceptrón Multicapa

Backpropagation in multilayer perceptron

Autor 1: Kevin Leandro Bernal León, Leidy Carolina Arcila Cardona, Gustavo Alexander Dávila Ayala

*Facultad de Ingenierías. IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia*

Correo-e: [kevinleon@utp.edu.co](mailto:kevinleon@utp.edu.co), [davila7@utp.edu.co](mailto:davila7@utp.edu.co)

***Resumen*— Crearemos una red neuronal artificial en Python con las librerías de alto nivel, Keras y Tensorflow para comprender su uso. En donde se implementará la compuerta XOR, donde se logra un nivel de predicción para la salidas de la misma, en base a las posibles entradas binarias y ise compararán las ventajas del aprendizaje automático frente a la programación tradicional.**

***Palabras clave—* Aprendizaje, entrenamiento, inferencia, inteligencia artificial, neurona, patrones, perceptrón, pesos, red neuronal, umbral.**

***Abstract*— We will create an artificial neural network in Python with high-level libraries, Keras and Tensorflow to understand its use. Where the XOR gate will be implemented, where a prediction level is achieved for its outputs based on the possible binary inputs and the advantages of machine learning will be compared to traditional programming.**

***Key Word* —Learning, training, inference, artificial intelligence, neuron, patterns, perceptron, weights, red neuronal, threshold.**

1. INTRODUCCIÓN

Una de las principales razones por la que las redes neuronales desaparecieron de la mayoría de los proyectos de investigación durante la década de los 70 y parte de los 80 fue que las redes conocidas hasta entonces, cuyo exponente máximo era el perceptrón (hoy en día perceptrón simple); no eran capaces de solucionar problemas no linealmente separables.

Para problemas linealmente separables, existían soluciones algorítmicas tradicionales eficientes, luego no tenía sentido utilizar la tecnología neuronal.

Para que nos hagamos una idea, un problema linealmente separable es aquél que puede dividirse en dos áreas claramente diferenciadas mediante una línea. Vale cualquier línea para separarlos, tan compleja como se desee.

Por ejemplo, si el problema que queremos resolver es saber dadas dos entradas, cuál es el resultado de realizar la operación lógica OR entre ellas, podemos dividir el problema linealmente de forma sencilla.

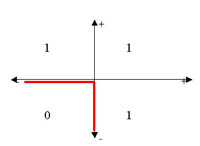


Figura 1. Operación lógica OR separada linealmente.

Sin embargo, si lo que quisiéramos es calcular el OR exclusivo de las entradas, no podemos separar linealmente el problema.

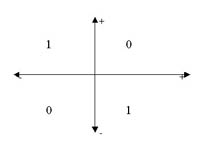


Figura 2. Representación de la operación lógica XOR.

Esta operación marca ya una barrera en las redes neuronales, que no se romperá hasta que en 1986 Rumelhart y McClelland diseñaran el *perceptrón multicapa* (o MLP, multilayer perceptron), revolucionando todo lo conocido hasta entonces en redes neuronales.

El algoritmo realiza mediante un perceptrón multicapa esta operación de OR Exclusivo. Nos permite ver varias características básicas del mismo: curva de aprendizaje, resolución de problemas no linealmente separables, tipos de aprendizaje, error y aproximaciones, etc.

1. CONTENIDO

Tenemos dos entradas binarias (1 ó 0) y la salida será 1 sólo si una de las entradas es verdadera (1) y la otra falsa (0). Es decir que de cuatro combinaciones posibles, sólo dos tienen salida 1 y las otras dos serán 0, como vemos aquí:

1. XOR(0,0) = 0
2. XOR(0,1) = 1
3. XOR(1,0) = 1
4. XOR(1,1) = 0

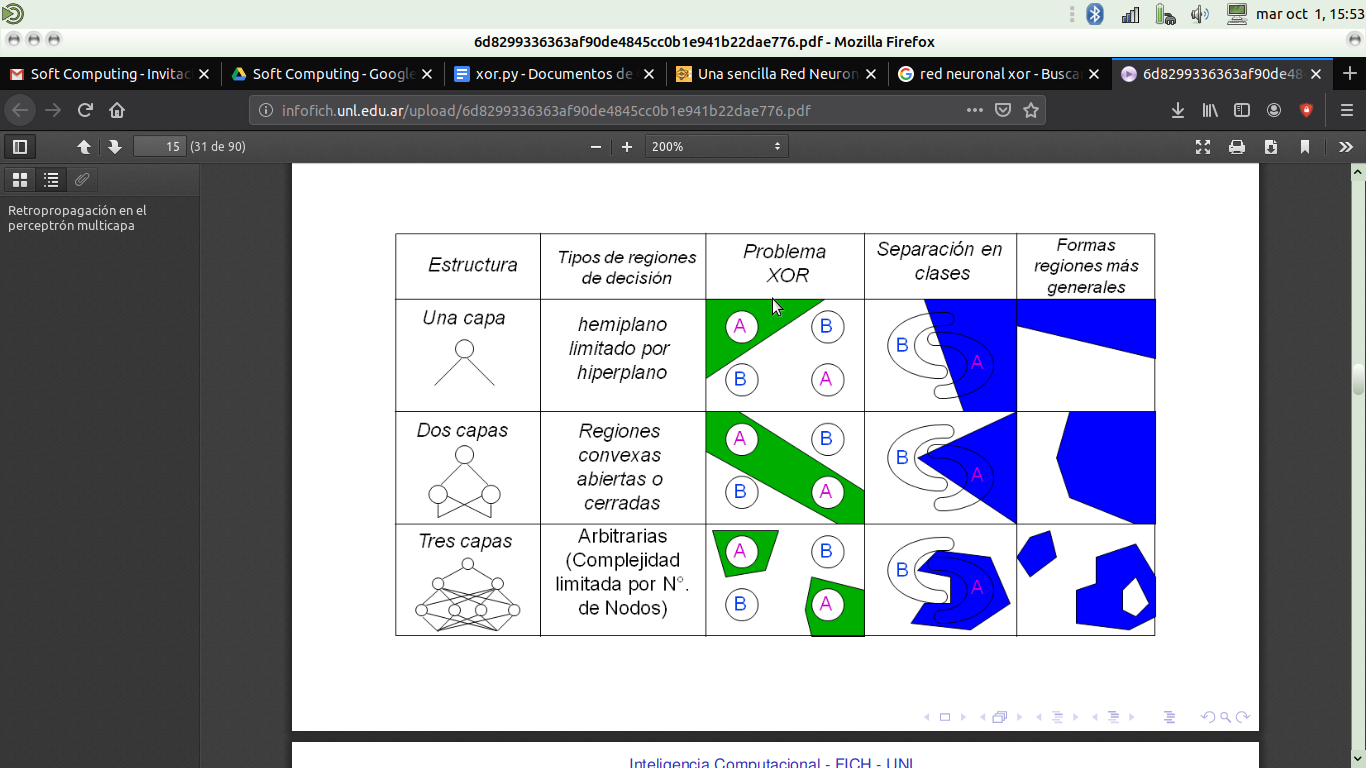
**1) Algoritmo**

El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo LMS, ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. La red Backpropagation trabaja bajo aprendizaje supervisado y por tanto necesita un set de entrenamiento que le describa cada salida.

El entrenamiento de una red neuronal multicapa se realiza mediante un proceso de aprendizaje, para realizar este proceso se debe inicialmente tener definida la topología de la red esto es: número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada, cantidad decapas ocultas y número de neuronas de cada una de ellas, número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrones objetivo y funciones de transferencia requeridas en cada capa,con base en la topología escogida se asignan valores iniciales a cada uno de los parámetros que conforma la red.

Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino mas adecuado para la actualización de los pesos y ganancias que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error medio cuadrático en cada iteración del proceso de aprendizaje.

**2) Arquitectura**

 Figura 3. Estructura del perceptrón multicapa.

La arquitectura de esta red consta de 3 capas, cada una totalmente conectada con la siguiente:

* Capa de Entrada de dos neuronas, que tomará los dos valores para hacer la operación. Aunque estos valores son binarios, la red está programada de forma que podría aceptar valores continuos y dar la respuesta adecuada.
* Capa Oculta, con un número configurable de neuronas mediante la opción *nº de neuronas ocultas*. El hecho de que exista la capa oculta es la que da al perceptrón multicapa la capacidad de resolver problemas no separables linealmente. La cantidad de neuronas que se utilicen hace que varíe el tiempo que se tarda en minimizar el error. 1 neurona hace el trabajo muy lento, 2 ó 3 son el número más adecuado (de 6000 a 40000 iteraciones para un error de 0.1). 4 ó más son muy eficientes (menos de 15000 its.), pero aumentan la carga de trabajo en cada iteración al aumentar el número de pesos a calcular.
* Capa de salida. Una sola neurona que nos da el valor del XOR entre las dos entradas. Se ha mantenido un valor continuo de salida para que pueda observarse el efecto de tolerar más o menos error.

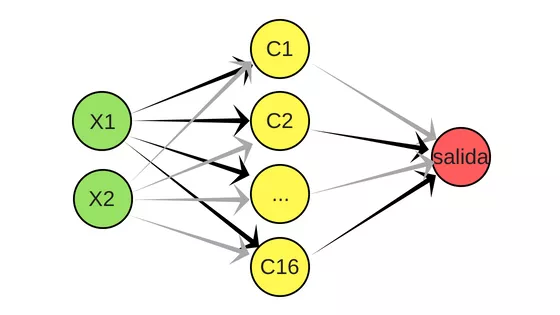


Figura 4. Arquitectura del perceptrón multicapa.

**3) Entrenamiento**

Durante el entrenamiento, mostramos a la red el juego de ensayo, que es la tabla del XOR:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **XOR** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

El entrenamiento se realiza con retropropagación *batch* (se pasan los cuatro juego de ensayo antes de modificar los pesos). En el gráfico podemos observar el descenso del error cuadrático medio según aumentan las iteraciones.

Podemos configurar:

* *Momento*: controla en qué grado el estado en el que se encuentra la red neuronal influye en el estado que tendrá en el siguiente paso del aprendizaje.
* *Factor de Aprendizaje*: controla el efecto del error en la modificación de los pesos de las capas de neuronas.
* *Nº de Iteraciones*: número máximo de iteraciones que usaremos para entrenar la red. Si no se alcanza el error máximo tolerado en esas iteraciones, la red trabajará con el error al que haya llegado.
* *Error Máximo*: error cuadrático medio (MSE, *Medium Square Error*) máximo que tolerará la red. Mientras no se haya alcanzado, continuará el entrenamiento hasta que se alcance, o hasta que agotemos las iteraciones estipuladas, lo que ocurra primero. Se observa claramente que el error máximo influye en la respuesta de la red. Con un error de 0.1, la red responde a una entrada 0,0 con un 0.15, mientras que si el error es de 0.01 nos da un 0.08.

**3) Resultados**

Al ver las salidas del entrenamiento, vemos que las primeras lineas arrojan:



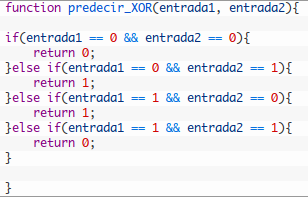
Con esto se nota que la primer iteración «tuvo algo de suerte» y acertó la mitad de las salidas (0.5), pero a partir de la segunda, sólo acierta 1 de cada 4 (0.25).

Luego en la iteración 24 recupera el 0.5 de aciertos, ya no es «por suerte», si no por haber ajustado correctamente los pesos de la red.

Y en este caso, en la iteración 107 aumenta los aciertos al 0,75 (son 3 de 4) y en la iteración 169 logra el 100% de aciertos y se mantiene así hasta finalizar. Como los pesos iniciales de la red son aleatorios, puede que las salidas que tengas en tu ordenador sean levemente distintas en cuanto a las iteraciones, pero llegarás a la «precisión binaria» (binara\_accuracy) de 1.0.

III. CONCLUSIÓN

Luego de lo anteriormente visto, ¿no conviene hacer una programación «tradicional» en vez de entrenar una red neuronal? Pues siempre dependerá del caso. Por ejemplo para la función XOR tendríamos algo así:



Vemos que es una función con 4 condicionales que evalúa cada caso. ¿Pero que pasaría si en vez de 2 entradas tuviéramos más parámetros? Seguramente la cantidad de condicionales aumentaría creando un código caótico y propenso a errores, difícil de mantener. No quiere decir que haya que reemplazar todo el código del mundo con redes neuronales, pero sí pensar en que casos las redes neuronales nos brindan una flexibilidad y un poder de predicción increíbles y que se justifican el tiempo de desarrollo.

REFERENCIAS.

[1] Red neuronal de 3 capas. http://infofich.unl.edu.ar/upload/6d8299336363af90de4845cc0b1e941b22dae776.pdf

[2] Regla del aprendizaje del perceptrón. https://es.slideshare.net/aerdna07/regla-de-aprendizaje-del-p erceptrn-simple

[3] El perceptrón Simple. http://www.lcc.uma.es/~munozp/documentos/modelos\_com putacionales/temas/Tema4MC-05.pdf

[5] Red Neuronal en Python. https://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/

[6] Redes Multicapa. ftp://decsai.ugr.es/pub/usuarios/castro/Actividades/Redes-Neuronales/Apuntes/Apuntes%20Javier%20Rodriguez%20Blazquez/Redes%20Multicapa%20-%20Algoritmo%20Backpropagation.pdf