A Review of Deep Learning Fundamentals – the XOR Neural Network Model

Una Revisión de los Fundamentos de Aprendizaje Profundo - El Modelo de Red Neuronal XOR

Alejo Mosso-Vazquez[1*], David Juarez-Romero[2], José Alfredo Hernández-Pérez[2], Darvi Echeverría Sosa[3], Jimer Emir Loría Yah[1], Ramiro José González Horta[1], Gerardo Israel de Atocha Pech Caraveo[1] y Carlos Alberto Decena Chan[1]

[1]Departamento de Mecatrónica, Instituto Tecnológico Superior de Calkiní en el Estado de Campeche
Av. Ah-Canul S/N Col. San Felipe CP 24900, Calkiní, Campeche, México.
*mel_22281@hotmail.com
[2] Centro de Investigación en Ingeniería y Ciencias Aplicadas, UAEM
Av. Universidad No. 1001, Col. Chamilpa, Cuernavaca, Morelos, CP 62209, México.
[3] Departamento de Ingeniería Electromecánica, Tecnológico Nacional de México. Instituto Tecnológico Superior de Motul
Carretera Mérida Motul, Tablaje Catastral 383, Motul, Yucatán, México. C.P. 97430

PALABRAS CLAVE:

Deep Learning, Redes Neuronales, Función XOR, Algoritmo Retropropagación, Algoritmo Gradiente Descendente Estocástico

RESUMEN

Este artículo explora los fundamentos de Deep Learning mediante el seguimiento, en un simple modelo de Redes Neuronales de la función XOR, de las señales en forward y backward que fluyen a través de este modelo. Nuestro objetivo es alcanzar una comprensión más profunda de algunos conceptos sobresalientes de Deep Learning, lo que nos permitiría comprender su significado mientras el modelo de Redes Neuronales de la función XOR es entrenado por el algoritmo Retropropagación. El modelo elegido contiene una sola capa oculta con cuatro neuronas y una capa de salida con una neurona. Aunque este modelo no es una red neuronal profunda, su capa oculta lleva los conceptos suficientes de Deep Learning. Se utiliza la sigmoidea como función de activación en todas las neuronas. Se presenta una derivación de una versión simple del algoritmo Gradiente Descendente Estocástico, que se usa para minimizar el error de salida, y luego al retropropagarlo llegamos al algoritmo de retropropagación. Se presentan resultados numéricos, que muestran la convergencia del error de salida y el de un peso seleccionado y su análisis resume la comprensión de los conceptos fundamentales de Deep Learning.

KEYWORDS:

Deep Learning, Neural Networks, XOR Function, Backpropagation Algorithm, Stochastic Gradient Descent algorithm

ABSTRACT

This paper explores the fundamentals of Deep Learning by searching a simple Neural Network model of the XOR function for the forward and backward signals flowing through this model. Our purpose is to reach a deeper understanding of some outstanding concepts of Deep Learning, which would enable us to get the significance of it while the Neural Network model of the XOR function is trained by the backpropagation algorithm. The chosen Neural Network model contains just one hidden layer with four neurons and an output layer with one neuron. Although this model is not a deep neural network, its hidden layer carries the enough concepts of Deep Learning. The sigmoid is used as the activation function in all neurons. A derivation of a simple version of the Stochastic Gradient Descent algorithm is presented, which is used to minimize the output error, and then by backpropagating it we come to the back-propagation algorithm. Numerical results are presented, which shows the convergence of the output error and that of a selected weight and their analysis summarize the understanding of the fundamental concepts of Deep Learning.

• **Recibido: 15 de diciembre de 2021** • **Aceptado: 2 de mayo 2022** • **Publicado en línea: 28 de febrero 2023**

ProgMat,2023,15,1; https://progmat.uaem.mx/progmat
DOI: 10.30973/progmat/2023.15.1/5

1. INTRODUCTION

Of Boolean functions of two inputs, XOR poses any difficulty. That is, the construction of neural networks that learn to compute such a function has been a challenge. The ability to compute XOR has been used as a test of variants of standard algorithms. Minsky and Papert (1969) [1] pointed out that "the computer is so much more than the sum of its parts and that we cannot ignore the nature of the components and consider only their connectivity". In the NN model of the XOR we will be dealing with, we consider the nature of the non-linear operation that take place inside the core of the neuron. As it will be shown later, the sigmoid φ is used as a forward filter and its derivative φ' as a backward symmetric low pass filter. There have been many contributions to the field of neural network (NN) that lead to what we know as deep learning algorithms [1]-[6]. We present next the most remarkable ideas of these contributions. The first mathematical model of a neuron was introduced in 1943 by W. McCulloch and W. Pitts [2], which consists of a weighted sum of input signals and its comparison to a threshold to deliver an output; the weight in this model were adjusted manually. A remarkable achievement is the perceptron introduced in 1958 by Frank Rosenblatt [3], which was the first neuron model to be described algorithmically. Another great advance in training a NN model was the backpropagation algorithm introduced in 1986 by D.E. Rumelhart, G.E. Hinton, and R.J. Williams [4],[5], which led to Deep Learning algorithms introduced in 2006 by G. E. Hinton, S. Osindero and Yee-Whye Teh [6].

The contribution of this work is to provide the reader a deeper understanding of some outstanding concepts of Deep Learning as it is known today. To present such concepts we have chosen the Neural Network (NN) model of the XOR function as it is introduced in [7], which involves just one hidden layer. Although this model is not a deep neural network, its hidden layer carries the enough concepts of Deep Learning. Our hope is that understanding the role of the forward and backward signals flowing through this model will enable us to appreciate the significance of Deep Learning. A derivation of a simple version of the Stochastic Gradient Descent (SGD) algorithm [7], [8], [9] is presented in section 2 to train the output layer of the NN model Of the XOR function by minimizing its output error. Then by backpropagating the output error we will come to the backpropagation algorithm, which is presented in section 3. Numeric and graphical results of the convergence of the output error and of a weight are presented in section 4. Future works are presented in section 5 and conclusions in section 6.

2. SGD FOR TRAINING A NN MODEL OF THE XOR FUNCTION

In this section a simple SGD algorithm is derived for minimizing the output error of a NN model of the XOR function. The aim of using this model is its simplicity. It contains just one hidden layer with four neurons and an output layer with one neuron. We present an electronic representation of the XOR function and next a NN model.

2.1. Electronic representation of the XOR function

The XOR is a digital function $x_1 \, XOR \, x_2 = x_1 \bar{x}_2 + \bar{x}_1 x_2$ that gives a true output when the values of both inputs x_1, x_2 are different and a false if both input values are equal. The true table of the XOR function and an electronic representation are shown in Figure 1.

x_1, x_2	$x_1 \, XOR \, x_2$
0 0	0
0 1	1
1 0	1
1 1	0

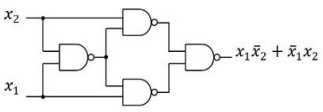
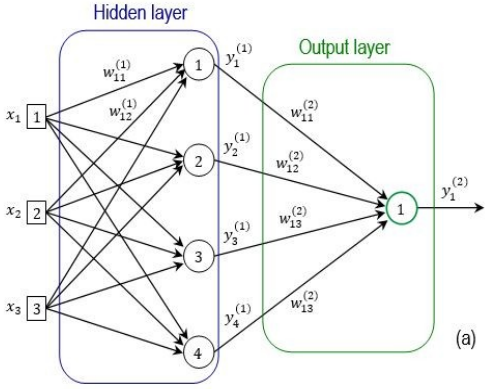


Figure 1: True table and an electronic representation of the XOR function.

2.2 Neural Network representation of the XOR function

The aim of this section is to present a NN model for the XOR function. We will use the structure of a NN proposed by Kim (2017) [7], as it is shown in Figure 2. This model consists of one hidden layer with four neurons and an output layer with one neuron. The sigmoid function is used as the activation function in all neurons. Notice that input x_1, x_2, x_3 and the target output d are known while the output $y_1^{(2)}$ is computed and it is hoped that $y_1^{(2)} \to d$. Since

the input x_1, x_2, x_3 and the target output d are known, the training process is called supervised learning. The input x_3, called the bias, is set to 1.



Input	Target output
x_1, x_2, x_3	d
0 0 1	0
0 1 1	1
1 0 1	1
1 1 1	0

(b)

Figure 2: (a) A NN model of the XOR function, which consists of three input nodes with digital inputs x_1, x_2, x_3 and a single output node with output $y_1^{(2)}$ [10]. (b) The true table of the XOR, which includes the bias x_3 and the target output d.

2.3 Learning problem formulation of the XOR function

Assume the following quantities are known: The inputs x_1, x_2, x_3 and the target output d. The initial values of weights of the hidden layer $w_{kj}^{(1)}$, $j=\{1,2,3\}; k=\{1,2,3,4\}$ and that of the output layer $w_{1,k}^{(1)}, k=\{1,2,3,4\}$. Given the data x_1, x_2, x_3, the target output d, and the initial wights $w_{kj}^{(1)}$ and $w_{1,k}^{(1)}$ of the hidden layer and the output layer respectively, the problem we want to solve is to optimally adjust the wights $w_{kj}^{(1)}$ and $w_{1,k}^{(1)}$ by means of the Backpropagation algorithm.

2.4 SGD algorithm to minimize the output error

The weights $w_{1,k}^{(1)}$ of the output layer may be adjusted minimizing the output error by means the Stochastic Gradient Descent method (SGD). The output error is shown in the output neuron of Figure 3 and is defined as the difference between the target output d and the computed output $y_1^{(2)}$, that is $e_j^{(2)}=\left(d_j-y_1^{(2)}\right)$, $j=\{1,2,3,4\}$.

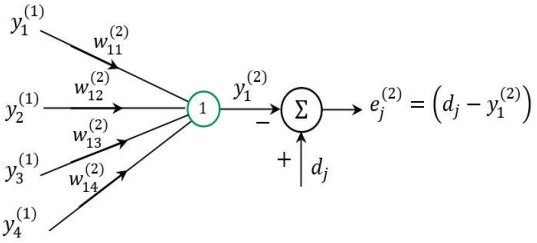


Figure 3: Definition of the output error $e_j^{(2)}$.

In order to determine the optimal weights $w_{1,k}^{(2)}$ of the output layer, an energy function is proposed to be minimized in terms on the output error $e_j^{(2)}=\left(d_j-y_1^{(2)}\right)$, $j=\{1,2,3,4\}$. The proposed energy function is specified as the following quadratic form:

$$E\equiv\frac{1}{2}\left(e_j^{(2)}\right)^2=\frac{1}{2}\left(d_j-y_1^{(2)}\right)^2 \tag{1}$$

where

$$y_1^{(2)}=\frac{1}{1+e^{-v_1^{(2)}}} \tag{2}$$

$$v_1^{(2)}=w_{11}^{(2)}y_1^{(1)}+w_{12}^{(2)}y_2^{(1)}+w_{13}^{(2)}y_3^{(1)}+w_{14}^{(2)}y_4^{(1)} \tag{3}$$

Since E depends on each weight $w_{1k}^{(2)}, k=\{1,2,\ldots,4\}$, it may be minimized with respect to it. For illustration purpose and simplicity, let us derive E only with respect to each weigh $w_{1k}^{(2)}$ of the output layer for some fixed target output d_j, $j\in\{1,2,3,4\}$:

$$\begin{aligned}\frac{\partial E}{\partial w_{1k}^{(2)}}&=\frac{\partial}{\partial w_{1k}^{(2)}}\left(\frac{1}{2}\left(d_j-y_1^{(2)}\right)^2\right)=\frac{1}{2}\frac{\partial}{\partial w_{1k}^{(2)}}\left(d_j-y_1^{(2)}\right)^2\\&=\left(d_j-y_1^{(2)}\right)\frac{\partial}{\partial w_{1k}^{(2)}}\left(d_j-y_1^{(2)}\right)\\&=-e_j^{(2)}\frac{\partial}{\partial w_{1k}^{(2)}}\left(y_1^{(2)}\right)=-e_j^{(2)}\left(\frac{\partial y_1^{(2)}}{\partial v_1^{(2)}}\right)\frac{\partial v_1^{(2)}}{\partial w_{1k}^{(2)}}\\&=-e_j^{(2)}\left(\frac{\partial y_1^{(2)}}{\partial v_1^{(2)}}\right)\frac{\partial v_1^{(2)}}{\partial w_{1k}^{(2)}}\end{aligned} \tag{4}$$

In Appendix A the derivatives of the sigmoid $y_1^{(2)}$ and the function $v_1^{(2)}$ are determined as (A.1) and (A.2) respectively and are written as follows:

$$\left(\frac{\partial y_1^{(2)}}{\partial v_1^{(2)}}\right)=y_1^{(2)}\left(1-y_1^{(2)}\right) \tag{5}$$

$$\frac{\partial v_1^{(2)}}{\partial w_{1k}^{(2)}}=y_k^{(1)} \tag{6}$$

Using results (5) and (6) in (4), we finally have:

$$\frac{\partial E}{\partial w_{1k}^{(2)}} = -e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right) y_k^{(1)} \tag{7}$$

2.5. Recursive computation of the weights of the output layer

We will focus our analysis to the weight of the output layer. Notice that we can compute these weights because we are able to minimize the output error. During this minimization, errors associated to the hidden layer are formed, which will enable us to also compute their weights. Given a current known weight $w_{1k}^{(2)}$ (connecting a neuron k of the hidden layer to the output neuron 1) of the output layer (2), the SGD method improve recursively the given weight as follows:

$$w_{1k}^{(2)} = w_{1k}^{(2)} + \Delta w_{1k}^{(2)}, \quad k=\{1,2,3,4\} \tag{8}$$

The delta rule for modifying the weights $w_{1k}^{(2)}$ of the output layer is specified as follows:

$$\begin{aligned} \Delta w_{1k}^{(2)} &= -\alpha \frac{\partial E}{\partial w_{1k}^{(2)}}, \quad \alpha \in (0,1] \text{ the step length} \\ &= -\alpha\left(-e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right) y_k^{(1)}\right) \\ &= \alpha\, e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right) y_k^{(1)} \\ &= \alpha\, \delta_1^{(2)} y_k^{(1)}, \quad k=\{1,2,3,4\} \end{aligned} \tag{9}$$

The delta function in the core of the output neuron of the output layer (2):

$$\delta_1^{(2)} = e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right) \tag{10}$$

carries the output error $e_j^{(2)}$, which cross through a low pass filter defined by the derivative $y_1^{(2)}\left(1-y_1^{(2)}\right)$ of the sigmoid, as it is illustrated in Figure 4.

3. BACKPROPAGATION ALGORITHM

The backpropagation algorithm is the method to efficiently train a deep NN and teach it to learn from its errors [11]. As it was shown in the previous sections, the knowledge of the output error enables us to compute only the weights of the output layer. The idea behind the backpropagation algorithm is to back propagate the output error through the hidden layers to have errors for each of them. Once these errors are available, we may compute the weights of the hidden layers. Figure 5 shows three key quantities y, e and δ that flows through the layers. We will show later that these quantities affect the computation of all the weights of the NN.

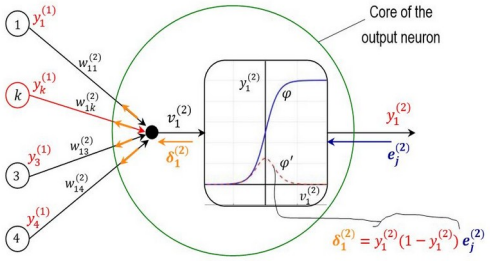


Figure 4: Output layer showing the formation of the delta function $\delta_1^{(2)}$.

- y Forward variable - output of each neuron
- e Error – backpropagated
- δ Backward variable – Delta function

The weights and delta function of the output layer are determined by these quantities:

$$w_{1k}^{(2)} = w_{1k}^{(2)} + \alpha\, \delta_1^{(2)} y_k^{(1)}, k=\{1,2,3,4\} \tag{11}$$

$$\delta_1^{(2)} = e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right), \text{for some } j \in \{1,2,3,4\} \tag{12}$$

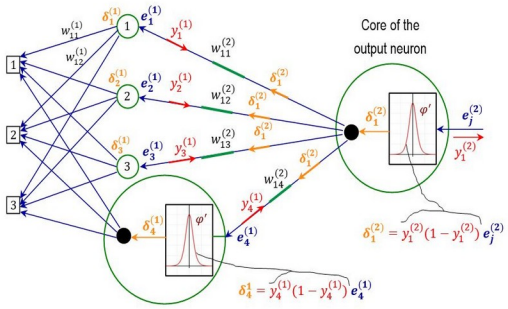


Figure 5: Backpropagation of the errors and delta functions through layers.

Backpropagation of the error: In Figure 5 we observe that the errors back entering the hidden layer may be determined considering the behavior of a dendrite. That is, the output of a dendrite is the input it receives $\delta_1^{(2)}$ multiplied by its weight $w_{1k}^{(2)}$, that is: $e_k^{(1)} = w_{1k}^{(2)} \delta_1^{(2)}, k=\{1,2,3,4\}$:

$$\begin{aligned} e_1^{(1)} &= w_{11}^{(2)} \delta_1^{(2)} \quad e_2^{(1)} = w_{12}^{(2)} \delta_1^{(2)} \\ e_3^{(1)} &= w_{13}^{(2)} \delta_1^{(2)} \quad e_4^{(1)} = w_{14}^{(2)} \delta_1^{(2)} \end{aligned} \tag{13}$$

Delta function in the hidden layer: Recall that the delta function in the output neuron is determined by passing the output error $e_j^{(2)}$ through the symmetric low pass filter $y_1^{(2)}\left(1-y_1^{(2)}\right)$, obtaining $\delta_1^{(2)} = e_j^{(2)} y_1^{(2)}\left(1-y_1^{(2)}\right)$. In the same way, errors $e_k^{(1)}, k=\{1,2,3,4\}$ in the hidden layer crossing through their corresponding filters $y_k^{(1)}\left(1-y_k^{(1)}\right), k=\{1,2,3,4\}$ should produce a

delta function in the core of each neuron of this layer:

$$\delta_4^{(1)}=y_4^{(1)}\left(1-y_4^{(1)}\right)e_4^{(1)} \quad \delta_3^{(1)}=y_3^{(1)}\left(1-y_3^{(1)}\right)e_3^{(1)}$$
$$\delta_2^{(1)}=y_2^{(1)}\left(1-y_2^{(1)}\right)e_2^{(1)} \quad \delta_1^{(1)}=y_1^{(1)}\left(1-y_1^{(1)}\right)e_1^{(1)} \tag{14}$$

Weights of the output layer: Using the delta function $\delta_1^{(2)}$ of the output neuron, the weights $w_{1k}^{(2)}, k=\{1,2,3,4\}$ of the output layer may be adjusted according to the following learning rule: $w_{1k}^{(2)}=w_{1k}^{(2)}+\alpha\,\delta_1^{(2)}\,y_k^{(1)}, k=\{1,2,3,4\}$ and $\alpha\in(0,1]$ as it was shown in (1) and (12), that is:

$$w_{14}^{(2)}=w_{14}^{(2)}+\alpha\,\delta_1^{(2)}\,y_4^{(1)} \quad w_{13}^{(2)}=w_{13}^{(2)}+\alpha\,\delta_1^{(2)}\,y_3^{(1)}$$
$$w_{12}^{(2)}=w_{12}^{(2)}+\alpha\,\delta_1^{(2)}\,y_2^{(1)} \quad w_{11}^{(2)}=w_{11}^{(2)}+\alpha\,\delta_1^{(2)}\,y_1^{(1)} \tag{15}$$

A 1×4 vector of weights of the output layer may be constructed as follows:

$$W_2=\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} & w_{14}^{(2)}\end{bmatrix} \tag{16}$$

Weights of the hidden layer: Similarly, using the delta function given in (14) $\delta_k^{(1)}, k=\{1,2,3,4\}$ of each neuron of the hidden layer, the weights $w_{jk}^{(2)}, j=\{1,2,3\}; k=\{1,2,3,4\}$ may be adjusted according to the following learning rule: $w_{jk}^{(1)}=w_{jk}^{(1)}+\alpha\,\delta_k^{(1)}\,x_j, j=\{1,2,3\}; k=\{1,2,3,4\}$ and $\alpha\in(0,1]$, that is

$$w_{j4}^{(1)}=w_{j4}^{(1)}+\alpha\,\delta_4^{(1)}\,x_j, j=\{1,2,3\}$$
$$w_{j3}^{(1)}=w_{j3}^{(1)}+\alpha\,\delta_3^{(1)}\,x_j, j=\{1,2,3\}$$
$$w_{j2}^{(1)}=w_{j2}^{(1)}+\alpha\,\delta_2^{(1)}\,x_j, j=\{1,2,3\}$$
$$w_{j1}^{(1)}=w_{j1}^{(1)}+\alpha\,\delta_1^{(1)}\,x_j, j=\{1,2,3\} \tag{17}$$

A 4×3 matrix of weights of the hidden layer may be constructed as follows:

$$W_1=\begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} \\ w_{14}^{(1)} & w_{24}^{(1)} & w_{34}^{(1)}\end{bmatrix} \tag{18}$$

The Backpropagation algorithm: Training the NN model of the XOR function is presented in Table 1. The algorithm processes the following quantities: The input data to the NN model of the XOR, the linear and non-linear operations in forward propagation, the errors and delta functions in backpropagation, and the computation of weight matrices of the hidden and output layers.

4. RESULTS IN TRAINING A NN MODEL OF THE XOR FUNCTION

The hidden layer of the NN model of the XOR classifies the inputs by identifying a specific feature of the input set [11]. Figure 6 shows how the four neurons of the hidden layer identify the four states 00, 01, 10, 11 of the logical inputs when the target XOR output is $d_j, j=\{1,2,3,4\}$.

Table 1: Backpropagation algorithm to train the NN model of the XOR function [7].

INPUT DATA:

$X=\begin{bmatrix}001\\011\\101\\111\end{bmatrix}$, the XOR input; $D=\begin{bmatrix}0\\1\\1\\0\end{bmatrix}$, the target output

$W_1=(w_{ki})$, 4×3 matrix of random entries $w_{ki}\epsilon[-1,1]$ defined in (18)

$W_2=(w_{1k})$, 1×4 vector of random entries $w_{1k}\epsilon[-1,1]$ defined in (16)

TRAINING BY BACKPROPAGATION ALGORITM:

$\alpha\epsilon(0,1]$ the step length

$N=4$

for $t=1$ to 10,000

  for $k=1$ to N

    Forward propagation:

    $x=X_k$ the 1×3 $k-th$ row of matrix X

    $d=D_k$ the $k-th$ row of vector D

    $v_1=W_1x^T$ Linear operation in the hidden layer, a 4×1 vector

    $y_1=\frac{1}{1+e^{-v_1}}$ Sigmoid in the hidden layer, a 4×1 vector

    $v_2=W_2y_1$ Linear operation in the output layer, a scalar quantity

    $y_2=\frac{1}{1+e^{-v_2}}$ Sigmoid in the output layer, a scalar quantity

    Backpropagation:

    $e_2=d-y_2$ output error

    $\delta_2=y_2(1-y_2)e_2$ delta function in output layer

    $e_1=W_2^T\delta_2$ 4×1 error vector for hidden layer

    $\delta_1=y_1(1-y_1)e_1$ 4×1 delta function vector for hidden layer (an element-wise multiplication)

    Updates of weights:

    $W_1=W_1+\alpha\delta_1x^T$ 4×3 matrix of weights of the hidden layer

    $W_2=W_2+\alpha\delta_2y_1^T$ 1×4 vector of weights of the output layer

  end

end

The numeric values shown in the output layer were computed by means of the algorithm of Table 1 for the input $x=0,0,1$and the target $d_1=0$, which was coded in Matlab language. The functions $v_1^{(2)}$ and $y_1^{(2)}$ in the output neuron may be verified manually using their definitions given in (3) and (2) respectively:

$$v_1^{(2)}=w_{11}^{(2)}\,y_1^{(1)}+w_{12}^{(2)}\,y_2^{(1)}+w_{13}^{(2)}\,y_3^{(1)}+w_{14}^{(2)}\,y_4^{(1)}$$
$$y_1^{(2)}=\frac{1}{1+e^{-v_1^{(2)}}}=\varphi$$

Training consists of improving the current weights of the NN XOR model in each step of the algorithm using the weights computed in the step before. The initial set of weights is chosen randomly. We may run the algorithm many times to observe the convergence of

quantities such as the output error. Figure 7 shows the effect of the randomness of initial values of the weights on the convergence of the output error $e_1^{(2)}=\left(d_j-y_1^{(2)}\right)\Big|_{j=1}\to 0$ for three training tests.

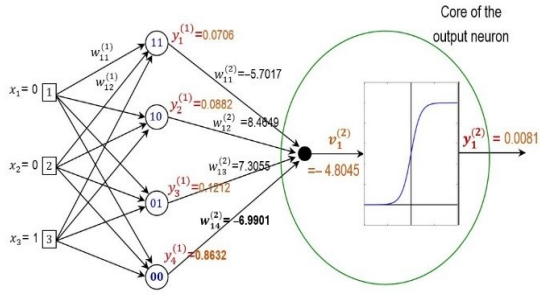


Figure 6: A numerical test of the NN XOR model. The neuron 00 of the hidden layer identify the input state $x=0,0,1$ by its output $y_4^{(1)}$=0.8632, which is greater than $y_3^{(1)}$, $y_2^{(1)}$ and $y_1^{(1)}$.

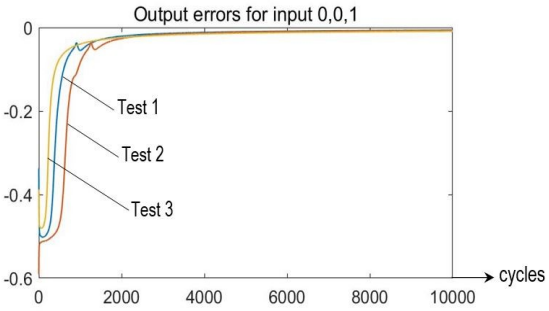


Figure 7: Convergence of the output error for three tests when the input is $x=0,0,1$ and the desired output is $d_1=0$ and $y_1^{(2)}$=0.0081. Although the initial weights are selected randomly, the final error $e_1^{(2)}=\left(d_1-y_1^{(2)}\right)\to 0$, that is $y_1^{(2)}\to d_1$ in 10,000 cycles.

Now, let us determine the convergence of one weight of the output layer, $w_{14}^{(2)}=-6.9901$ for example, for the numerical test of the NN XOR model reported in Figure 6. The numerical behavior of the weight $w_{14}^{(2)}$ is shown in Figure 8.

5. FUTURE WORKS

Once the fundamental concepts of Deep Learning have been understood, like the structure of a Deep NN, the basic functions performed by an artificial neuron, and the algorithms that make the NN model learn, the next step is to apply this technique in the treatment of a variety of technological applications [12], [13], [14]. Some of them are exposed next:

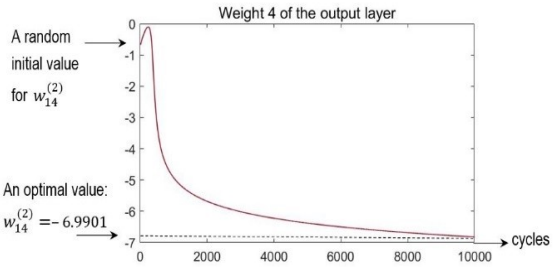


Figure 8: Convergence of the weight $w_{14}^{(2)}$ of the output layer to the optimal value −6.9901. The optimal value of any weight may be different for each random initial value. However, the output error converges always to zero, as it is shown in Figure 7.

1. *An Application of Deep Learning in Character Recognition*: Deep learning provides efficient and accurate result as compared to other techniques for character recognition.
2. *Deep Learning for Driverless Vehicles*: Automation is becoming a large component of transportation. A key technology to develop these vehicles is deep learning.
3. *Deep Learning for Brain Computer Interfaces*: Brain computer Interfaces have set new standards in the world of prosthetics, be it hearing aids or prosthetic arms, legs or vision, helping paralyzed users. Deep learning might serve as one of the translation algorithms that converts the raw signals from the brain into commands that the output devices follow.

6. CONCLUSIONS

We have explored a NN model of the XOR function to get a deeper understanding of some outstanding concepts of Deep Learning. We have examined its structure, functions, and the algorithms that make the NN model learn. The structure of the NN model includes mainly dendrites, their weights and nucleus of each neuron. We have considered the nature of the non-linear operation that take place inside the core of the neuron, that is, the sigmoid φ is used as a forward filter and its derivative φ' as a backward symmetric low pass filter.

The SGD algorithm and the Backpropagation algorithms were derived and examined. It was shown how the forward signals traverse the NN. We focused our attention to follow the backpropagation of the output error and the delta functions, which enable us for the computation of weights of the NN model of the

XOR function. Although just one hidden layer was treated, the analysis and algorithms presented in this work may be extended to deep neural networks.

Appendix A

A.1 *Derivative of the sigmoid function*: The sigmoid function given in (2) used as the activation function in all neurons of the NN model of the XOR is rewritten as follows:

$$y=\frac{1}{1+e^{-v}}=\varphi$$

and its derivative is determined next:

$$\begin{aligned}\varphi'&=\frac{d}{dv}y=\frac{d}{dv}\left(\frac{1}{1+e^{-v}}\right)=\frac{d}{dv}\left(1+e^{-v}\right)^{-1}\\&=-1\left(1+e^{-v}\right)^{-2}\frac{d}{dv}\left(1+e^{-v}\right)\\&=-\left(1+e^{-v}\right)^{-2}\left(0+e^{-v}\frac{d}{dv}(-v)\right)\\&=\left(1+e^{-v}\right)^{-1}\left(1+e^{-v}\right)^{-1}e^{-v}=\frac{1}{1+e^{-v}}\left(\frac{e^{-v}}{1+e^{-v}}\right)\\&=y\left(\frac{1+e^{-v}}{1+e^{-v}}-\frac{1}{1+e^{-v}}\right)=y\left(1-y\right)\end{aligned}\tag{A.1}$$

A.2 *Derivative of v with respect to $w_{1k}^{(2)}$ of the output layer*: The derivative of the linear function of the output layer given in (3):

$$v_1^{(2)}=w_{11}^{(2)}y_1^{(1)}+w_{12}^{(2)}y_2^{(1)}+\ldots+w_{14}^{(2)}y_4^{(1)}$$

with respect to the weights $w_{1k}^{(2)}$ of the output layer for $k=\{1,2,3,4\}$ is determined next:

$$\begin{aligned}\frac{\partial v_1^{(2)}}{\partial w_{1k}^{(2)}}&=\frac{\partial}{\partial w_{1k}^{(2)}}\left(w_{11}^{(2)}y_1^{(1)}+\ldots+w_{1k}^{(2)}y_k^{(1)}+\ldots+w_{14}^{(2)}y_4^{(1)}\right)\\&=0+\ldots\frac{\partial}{\partial w_{1k}^{(2)}}\left(w_{1k}^{(2)}\right)y_k^{(1)}\ldots+0=y_k^{(1)}\end{aligned}\tag{A.2}$$

The graph of the sigmoid function φ and its derivative φ' are shown in Figure A.1.

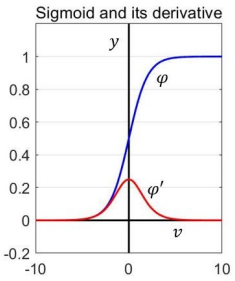


Figure A.1: The sigmoid φ and its derivative φ'.

REFERENCIAS

[1] Minsky, M. and Papert, S. (1969) Perceptrons - An introduction to computational geometry. The MIT Press.

[2] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics., Vol. 5, pp. 115–133. https://doi.org/10.1007/BF02478259

[3] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review, Vol. 65, pp. 386–408. https://psycnet.apa.org/doi/10.1037/h0042519

[4] Rumelhart, D.E., Hinton, G. E. and Williams, R. J. (1986). Learning Internal Representation by Error Propagation. In D. E. Rumelhart and J. L. McClelland. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press.

[5] David E. E. Rumelhart, James L. Mcclelland, and PDP Research Group (1986). Parallel Distributed Processing - Explorations in the Microstructure of Cognition – Foundations. The MIT press.

[6] Hinton, G. E., Osindero, S. and Yee-Whye The (2006). A Fast Learning Algorithm for Deep Belief Nets. Neural Computation 18, 1527–1554. Massachusetts Institute of Technology. https://doi.org/10.1162/neco.2006.18.7.1527

[7] Kim, P. (2017). MATLAB Deep Learning - With Machine Learning, Neural Networks and Artificial Intelligence. Apress. https://doi.org/10.1007/978-1-4842-2845-6.

[8] Snyman, J.A. (2018). Practical Mathematical Optimization-Basic Optimization Theory and Gradient-Based Algorithms. Springer.

[9] Buduma, N. and Locascio, N. (2017). Fundamentals of Deep Learning - Designing Next-Generation Machine Intelligence Algorithms. O'Reilly Media.

[10] F. Bear, M.F., Connors, B.W., and Paradiso, M.A. (2020). Neuroscience-Exploring the Brain. Jones & Bartlett Learning.

[11] Reljan-Delaney, M. and Wall, J. (2017). Solving the Linearly Inseparable XOR Problem. School of Architecture Computing and Engineering University of East London, United Kingdom.

[12] Balas, V.E., Sekhar Roy, S., Sharma, D. and Samui, P. Editors (2019). Handbook of Deep Learning Applications. Springer. https://doi.org/10.1007/978-3-030-11479-4

[13] Yu, D. and Deng, L. (2015). Automatic Speech Recognition-A Deep Learning Approach. Springer. https://doi.org/10.1007/978-1-4471-5779-3

[14] Hernandez J.A. , R.J. Romero, D. Juárez-Romero, R.F. Escobar, J. Siqueiros. A. (2009). Neural network approach and thermodynamic model of waste energy recovery in a heat transformer in a water purification processes. Desalination (ISSN: 0011-9164) 243, pp 273-285. https://doi.org/10.1016/j.desal.2008.05.015

ACERCA DE LOS AUTORES

Alejo Mosso Vázquez. Recibió el grado de Ingeniero en Comunicaciones y Electrónica de la ESIME del IPN de México en 1975. Es maestro en ciencias con especialidad en Control por el CINVESTAV-IPN en México 1986. Es también maestro en ciencias en

sistemas de la manufactura con especialidad en Robótica por el ITESM–UT (USA) en 1993. También obtuvo el grado de Doctor en Ingeniería y Ciencias Aplicadas en Cuernavaca Morelos, México en 2012 por el CIICAP-UAEM. Sus intereses de investigación son Programación Matemática aplicada a la Robótica Humanoide, Control Automático y Redes Neuronales – Deep Learning.

David Juárez-Romero. Realizó su licenciatura en Ingeniería Química en la Fac. Química-UNAM, y sus estudios de maestría y doctorado en el Colegio Imperial de la Universidad de Londres, U.K. Su línea de investigación es "la mejora del Diseño y la operación de procesos de separación-transformación relacionados con máquinas de energía". Desarrolla metodologías para analizar, diseñar, y controlar estos procesos. Pertenece al Sistema Nacional de Investigadores.

José Alfredo Hernández-Pérez. Received a Professional Diploma in chemical engineering from UV (Veracruz). He received an M.S. degree in food science from I. T. of Veracruz and a Ph.D. degree in process engineering from École Nationale Supérieure des Industries Agricoles et Alimentaires (Paris, France). He works mainly on artificial intelligence in engineering processes. His research interests include modeling and simulation processes, optimization, and state estimation with application in heat and mass transfer processes and image analysis. He has published more than 110 articles. He is also a Reviewer for Neurocomputing, Energy, Int. Journal of Heat and Mass Transfer, Int. Journal of Thermal Sciences, Int. Journal of Refrigeration, Journal of Food Engineering, JAFC, MPE, Desalination, WASJ, RMCG, CABEQ, IJACT, Arabian JSE, Desalination and Water Treatment, IJEIS, IJTS, LAAR, and others. Finally, he is a member of the editor board of Computational Intelligence and Neuroscience (Impact Factor 2.28 according to the 2020 JCR released by Clarivate Analytics in 2018).

Darvi Echeverría Sosa. Recibió el grado de Ingeniero Mecánico en el I.T. de Mérida en 1996 (cédula profesional 2689731), es Maestro en Ingeniería Mecatrónica (cédula profesional 7136525) egresado de la Universidad Modelo en 2011 en Mérida, Yucatán. Actualmente es profesor de tiempo completo en el área de Electromecánica del TecNM, Campus Motul. Sus intereses son el desarrollo tecnológico y la investigación en las áreas de Diseño y Automatización de máquinas y mecanismos así como en el diseño mecatrónico.

Jimer Emir Loría Yah. Recibió el grado de Ingeniero en la carrera de Ingeniería Mecánica del I.T. de Mérida (cédula profesional 4912966) en el año de 2006. Es maestro en planificación de empresas y desarrollo regional, egresado del I.T.de Mérida (cedula profesional 6613953) en 2010. Es también Maestro en Ingeniería Mecatrónica (cédula profesional 11648300), por la Universidad Modelo, en Méridam, en 2019. Sus intereses son la investigación y desarrollo tecnológico sustentable de la región, así como al modelado y simulación de sistemas electrónicos y mecánicos.

Ramiro José González Horta. Recibió el grado de Arquitecto del I.T. de Acapulco en 1995. Es maestro en Administración de la Construcción por el I.T. de la Construcción de la Cd. de México en 2007. Es también candidato a Maestro en Ingeniería Mecatrónica por la Universidad Modelo, Mérida Yucatán, 2018. Es también Coordinador del Programa Académico de Ingeniería Mecatrónica del I.T.S. de Calkiní en el Estado de Campeche. Sus intereses de investigación son en el área de diseño, control e impresión 3D aplicado a la Robótica.

Gerardo Israel del Atocha Pech Caraveo. Nació en la ciudad de Dzitbalché, Campeche, México. Egresado del I.T.S. de Calkiní en el Estado de Campeche como Ingeniero Industrial en 2010. Del 2010-2015 se incorpora en la Industria Petrolera de iniciativa desempeñándose en el área producción. En el 2016 inicia como Profesor asociado A en el ITESCAM y en el año 2018 alcanza una maestría en Ingeniería Mecatrónica por la Universidad Modelo, participando como asesor en diversos proyectos y asesoramiento de tesis residencial. Las áreas a fines donde se desempeña son: Mecánica, Diseño y Modelado 3D.

Carlos Alberto Decena Chan. Nació en la ciudad de Campeche, México. Eresado de la Universidad Autónoma de Campeche como Ingeniero en Comunicaciones y Electrónica en 2007, con una maestría en Ingeniería Mecatrónica en 2018 en la Universidad Modelo, además de a ver realizado anteriormente una maestría en Ingeniería Administrativa en el año 2007-2009, en el IEU del Estado de Campeche. Las áreas a fines donde se desempeña son: Electrónica Analógica, Electrónica de potencia y la electrónica digital. El maestro actualmente se desempeña como profesor de asignatura "A", Actualmente está participando como Asesor de Tesis en el área de Ing. Mecatrónica.