World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:10, No:12, 2016

Analysis of Multilayer Neural Network Modeling and Long Short-Term Memory

Danilo López, Nelson Vera, Luis Pedraza

***Abstract*—**This paper analyzes fundamental ideas and concepts related to neural networks, which provide the reader a theoretical explanation of Long Short-Term Memory (LSTM) networks operation classified as Deep Learning Systems, and to explicitly present the mathematical development of Backward Pass equations of the LSTM network model. This mathematical modeling associated with software development will provide the necessary tools to develop an intelligent system capable of predicting the behavior of licensed users in wireless cognitive radio networks.

***Keywords*—**Neural networks, multilayer perceptron, long short-term memory, recurrent neuronal network, mathematical analysis

I. Introduction.

ARTIFICIAL neural networks (ANNs) are models of artificial intelligence that, inspired in biological neurons, can be used in classification, optimization, pattern recognition, function approximation tasks, etc. They are considered as powerful learning models [1] and are characterized by having in its structure a set of nodes or units called neurons (interconnected processing drives) and connections equipped with weights (associated determined importance and in which most of the knowledge that the neural network has on the task in question is usually kept [2]). As a statistical model, an ANN can learn the probability density function from the given samples and then predict, according to the statistics learned, outputs for new samples that were not included in the (sample) learning set [3]. Each neuron receives a series of inputs through interconnections and emits an output. This output can be modeled as the result given by composing three functions (Fig. 1).

Propagation or excitation function. It is a linear function that usually is the weighted sum of inputs and their respective weights.

Activation function. It is almost always sigmoidal: $(1 + e^{-1})^{-1}$ o $\tanh(x)$.

Transfer (or output) function. Usually to simplify, it is assumed for practical purposes to be the same as the identity function (i.e., the previous two functions would be enough).

Danilo López S is with the Universidad Distrital "Francisco José de Caldas", Faculty of Engineering, Cra. # 40B-53, Bogotá, Colombia (Corresponding Author; phone: +573108651144; e-mail: dalopezs@udistrital.edu.co).

Nelson Vera is with the Universidad Distrital "Francisco José de Caldas", Cra. # 40B-53, Bogotá, Colombia (e-mail: neverap@udistrital.edu.co).

Luis Pedraza is with the Universidad Distrital "Francisco José de Caldas", Faculty of Technology, Cl. 68D Bis A Sur # 49F - 70, Bogotá - Colombia (e-mail: lfpedrazam@udistrital.edu.co).

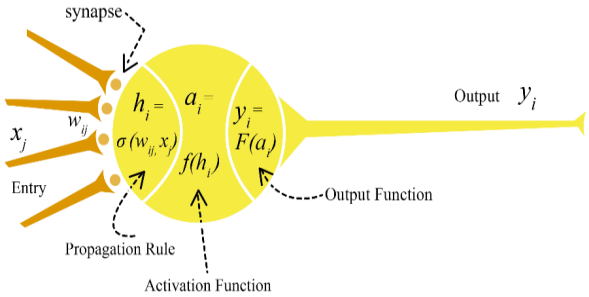

Fig. 1 Generic model of an artificial neuron [4]

Whenever a value is typed into the network (from the training set), an output is generated, therefore a function must be considered that enables quantifying the errors made and, depending on that, know if it is necessary to modify the weight values to minimize the error given by the network (i.e., until the network outputs are as close as possible to the reality that is being simulated). The process by which these weights are adjusted is known as training, and the procedure is applied as a learning algorithm. The ability to produce correct outputs for inputs not seen during training is known as generalization) [2]. An ANN is a network comprised of units called neurons which are related in some way. In order to give a more formal definition (from the mathematical point of view) we use the concept of graph.

Definition of ANN: ANNs as in Fig. 2 is a directed graph with the properties:

1. Every I node has an associated x_i state.
2. Every connection between two nodes (i and j) is assigned a weight w_ij $\in R$
3. For each i node there is a threshold θ_i.
4. For each i node a function f_i is defined, which depends on the weights of its connections, the threshold, and the states of the j nodes connected to it. This function $f_i(x_j, w_{ij}, \theta_i)$ provides the new state of the node.

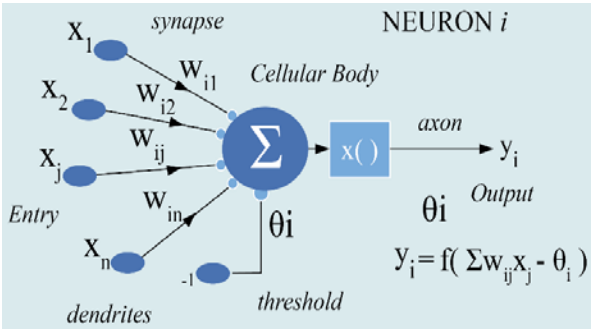

Fig. 2 Standard model of an artificial neuron [4]

A wide variety of ANNs have been developed, but can be classified, by type of connections in two groups: With cycles and without cycles. This paper briefly discusses the multilayer perceptron (MLP), which is a type of feed forward neural network without cycles; as well as some basic ideas about the RNNs), before finally focusing on LSTM networks, which is the motivation of this paper.

II. Multilayer Perceptron

MLP is an ANN whose topology (structure in which neurons are organized) is characterized by grouping neurons of the same type in substructures called layers (input, hidden, output) as shown in Fig. 3; connections between neurons only allow the flow of information in one direction (forward, so the neurons that are in the same layer are not related) and these can be totally or partially connected. MLPs are suitable for pattern recognition and prediction tasks, and are considered universal approximations of functions (especially in higher dimensions [3]), since their outputs depend only on the current inputs (or of the moment).

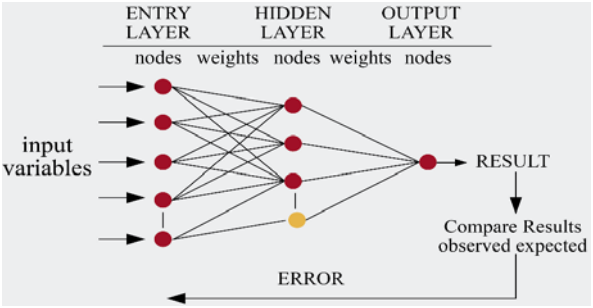

Fig. 3 Structure of a MLP [5]

TABLE I
Mathematical Nomenclature of an MLP Network

	Input layer	Hidden layer	Output layer
Subscript	i	h	k
Input	x_i	$a_h^{(l)}$	a_k
Output	x_i	$b_h^{(l)}$	y_k
Number of units	I	H_l	K

From here on, the total number of layers hidden in the MLP will be denoted by L; and the weight assigned to the connection between the neuron i of layer k and neuron j of layer $k+1$, for $w_{ij}^{(k)}$,the other notations are found in Table I.

The most common choices for activation are function j, due to its non-linearity (it allows reducing problems with multiple hidden layers to one with a single hidden layer) and differentiability (allows training the network with descending gradient) these are the functions:

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

and

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \text{(Logistic function)} \tag{1}$$

where the first derivative of these two functions are:

$$tanh'\,(x) = 1 - tanh^2\,(x)$$

and

$$\sigma(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{2}$$

In summary, we have the next notation for the h-th neuron of layer l:

$$a_h^{(l)} = \sum_{i=1}^{H_{l-1}} w_{ih}^{l-1} b_i^{(l-1)} \tag{3}$$

$$b_h^{(l)} = \theta_h(a_h^{(l)}) \tag{4}$$

It is said that an ANN has learned if it can be established that the error found in its outputs is minimal. Thus, the main purpose of applying neural networks in a problem (that could be to characterize PUs in cognitive networks) is to minimize the function of E error that depends "only" on the weights $W_{ij}^{(k)}$ (if there are activation thresholds, the function would also depend on its values). Generally, E is defined as the mean square error between the current output and the desired output:

$$E = \frac{1}{2}\sum_{i=1}^{nk}(S_i - Y_i)^2 \tag{5}$$

where, si are the test values ("real\data" of the situation); the yi are the neuron outputs of the output layer, i.e. $y_i = b_i^{(K)}$ [3]. The idea behind the descending gradient in an MLP is to find the derivative of the error function with respect to each of the weights $W_{ij}^{(k)}$, then modify those weights in the opposite direction to the derivative; more precisely, what is done is to subtract from the weight $W_{ij}^{(k)}$ *the value* or $-\alpha\frac{\partial E}{\partial w_{ij}^{(k)}}$, where α < 1(reason for learning). The partial derivative is taken because it represents the error variation when modifying a single variable, and to calculate the gradient efficiently, the technique known as Backpropagation is normally used.

III. Backpropagation for MPLs

This kind of algorithm can be interpreted as the mathematical heart of ANNs and is not only used to train feedforward networks like MLP, but can be adapted for RNNs; the LSTM model uses an adaptation of it in its learning, because of this it is necessary to understand how the method works. Backpropagation consists of repeatedly applying the chain rule for partial derivatives, and the first step consists of the derivatives of the loss function (or error) E with respect to the output neurons. For calculations presented in this document, the sigmoid function given in (1) is adopted as activation function θ_j; therefore:

$$b_h^{(1)} = \sigma\big(a_h^{(1)}\big)\sigma\big(\textstyle\sum_{i=1}^{I} x_i w_i\big) \tag{6}$$

$$b_h^{(1)} = \sigma\left(\sum_{i=1}^{H_{(l-1)}} b_i^{(l-1)}\, w_i^{(l-1)}\right), l = 2,3,\ldots,L \tag{7}$$

$$y_k = \sigma b_h^{(1)} = \sigma\left(\sum_{i=1}^{L} b_i^L \, w_{ik}^{(L)}\right) \quad (8)$$

Thus, taking into account (1) and (2), and applying the chain rule of calculation in several variables to (8), we have:

$$\frac{\partial E}{\partial a_k} = \sum_{j=1}^{K} \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial a_k} \quad (9)$$

$$\frac{\partial y_j}{\partial a_k} = y_k(1 - y_k) \quad (10)$$

The next notation will be used for any unit j in the neural network:

$$\delta_j^{(l)} = \frac{\partial E}{\partial a_j^{(l)}} \quad (1)$$

Thus, for units in the last hidden layer, we have:

$$\delta_h^{(l)} = \frac{\partial E}{\partial b_h^{(L)}} \frac{\partial b_h^{(L)}}{\partial a_h^{(L)}} = \frac{\partial b_h^{(L)}}{\partial a_h^{(L)}} \sum_{k=1}^{K} \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial b_h^{(L)}} = b_h^{(L)}(1 - b_h^{(L)}) \sum_{k=1}^{K} \delta_k w_{hk} \quad (12)$$

For the other hidden layers, we have:

$$\delta_h^{(l)} = \frac{\partial E}{\partial b_h^{(L)}} \frac{\partial b_h^{(L)}}{\partial a_h^{(L)}} = \frac{\partial b_h^{(L)}}{\partial a_h^{(L)}} \sum_{j=1}^{H_{L+1}} \frac{\partial E}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial b_h^{(L)}} = b_h^{(l)}(1 - b_h^{(l)}) \sum_{j=1}^{H_{L+1}} \delta_j w_{hj} \quad (13)$$

Once the deltas for all hidden neurons are calculated, by calculating the derivatives with respect to each of the weights, we arrive at:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ij}} = \delta_j^{(l)} b_i^{(l-1)} \quad (14)$$

IV. Recurrent Neural Networks (RNN)

Unlike the MLP, the RNNs allow one or more of the neurons that form it to feed back (graphically, cycles can be seen); the above suggests that an RNN can, in principle send the "history" of inputs previous to each output. His analysis considers an RNN with a single self-connected hidden layer (se Fig. 4).

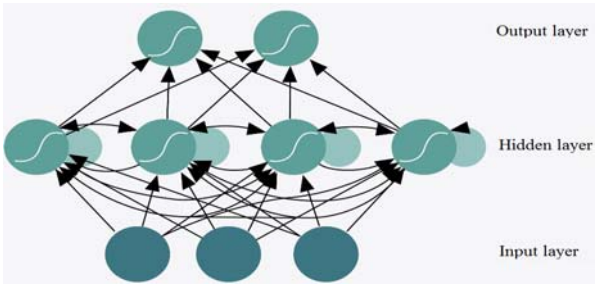

Fig. 4 Structure of an RNN [3]

The key idea is that the recurring connections allow a "memory" of previous inputs that, remaining in the internal state of the neuron, decreases in the unit output. It is possible to apply, for the RNN learning, a similar method as used for MLP. The activation functions are maintained, but the modification that the system undergoes at that moment is related to that the activations arrive at the hidden layer from two places: the input layer and from the same hidden layer.

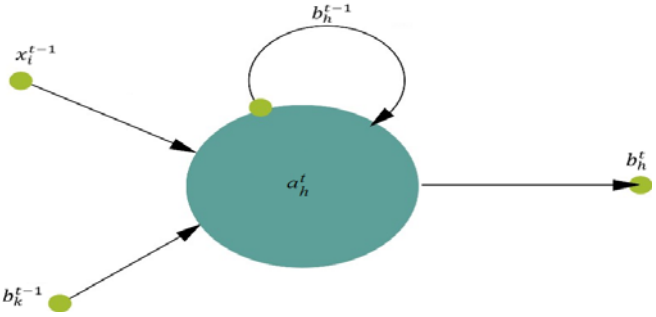

Fig. 5 Inputs and output of the hidden layer *h-th* neuron for a fixed time *t*.

TABLE II
Mathematical Nomenclature of an RNN Network

	Input layer	Hidden layer	Output layer
Number of units	I	H	K
Superscript	i	h	k
Input	x_i^t	$a_h^{(t)}$	a_k^t
Output	x_i^t	$b_h^{(t)}$	y_k^t

To mathematically analyze the RNN the notation shown in Table II shall be considered. In addition, it should be noted that: The superscript t refers to time; $b_h^{(0)} = 0$; and the weights between neurons are denoted as w_{ij}. From the previous description, we obtain:

$$a_h^t = \sum_{i=1}^{I} x_i^t w_i + \sum_{j=1}^{H} b_j^{t-1} w_{jh} \quad (15)$$

$$b_h^t = \sigma(a_h^t) \quad (16)$$

$$a_h^t = \sum_{h=1}^{H} b_h^t w_{hk} \quad (17)$$

Taking from Table II, we will use an analogue of Backpropagation but for RNN, namely: Backpropagation Through Time - BPTT. As in Backpropagation, BPTT consists of repeatedly applying the chain rule although the most important thing is that for RNNs, loss function depends on the activation of the hidden layer. Therefore, for the h-th hidden neuron we have:

$$\delta_h^t = \frac{\partial E}{\partial b_h^{(t)}} = \frac{\partial E}{\partial b_h^{(t)}} \frac{\partial b_h^{(t)}}{\partial a_h^{(t)}}$$

$$\delta_h^t = \frac{\partial E}{\partial a_h^t} = \frac{\partial E}{\partial b_h^t} \frac{\partial b_h^t}{\partial a_h^t} \frac{\partial b_h^t}{\partial a_h^t} \left(\sum_{k=1}^{K} \frac{\partial E}{\partial a_k^t} \frac{\partial a_k^t}{\partial b_h^t} + \sum_{j=1}^{H} \frac{\partial E}{\partial a_j^{t+1}} \frac{\partial a_j^{t+1}}{\partial b_h^t} \right)$$

$$\delta_h^t = b_h^t(1-b_h^t)\left(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{j=1}^{H}\delta_k^{t+1} w_{hj}\right) \quad (18)$$

Keeping in mind that the same weights are used at each time interval, we must apply the sum on all the time considered to obtain the derivatives with respect to the network weights. Therefore:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial E}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T}\delta_j^t b_i^t \quad (19)$$

In some cases, it is advisable to "unwind" the feedback neuron in order to better understand what is happening (Fig. 6); in doing so we can see a frame-by-frame of the "states" of the neuron as time progresses.

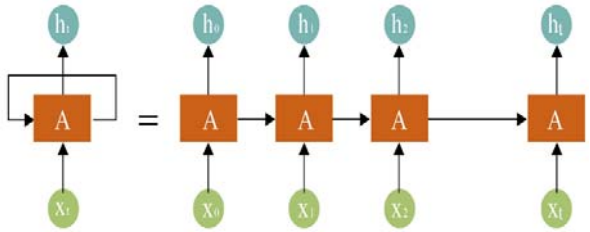

Fig. 6 Operation of a unit in an RNN [6]

V. Long Short-Term Memory

LSTM neural networks are a type of ANN whose structure consists of a set of memory blocks; basically, are recurrently connected subnets (Fig. 7). Each block has one or more self-connected cells and three "gates" that, for the cells, will perform the functions of writing (input), reading (output), and reset. This type of ANN was designed to solve the problem of the descending gradient (loss of learning achieved since the first inputs are "forgotten").

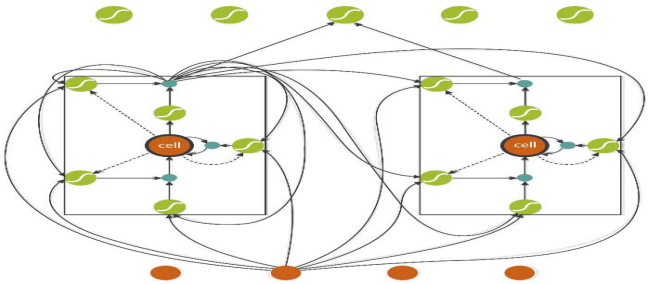

Fig. 7 Structure of an LSTM [3]

aGates allow LSTM memory cells to store and access information for long periods of time, thus cells can "remember" the first input until Input Gate is closed and the Forget Gate is open. Originally, LSTM only had the input and output gates [7]. The Forget Gate was added to enable the cell to auto-reset itself. Subsequently, peephole connections were included in order for LSTM to improve its learning ability [3]. Structurally, an LSTM is an RNN except that in the hidden layer there are memory blocks but no neurons, which have four inputs and one output. A graphic representation of a single-cell memory block can be seen in Fig. 8.

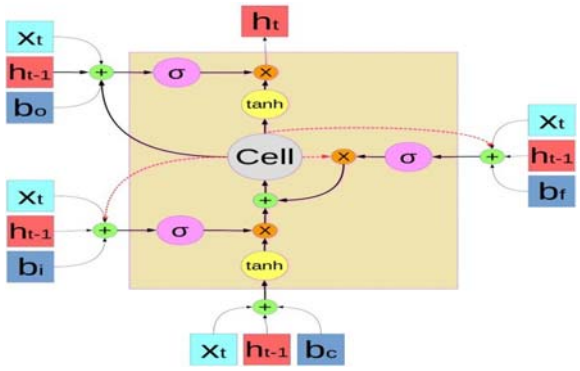

Fig. 8 Structure of a memory block [6]

Internal operation of an LSTM memory block. In order to understand the interaction of a memory block, reference will be made to Fig. 8. In principle, the three gates are units that collect the activations inside and outside the block, and their function is to control cell activation through multiplications. The input and output gates multiply the cell input and output while the forget gate multiplies the previous state of the cell. There is no activation function inside the cell. The gate activation function is usually the logistic function σ (described in (1)), where gate activations are between 0 (closed gate) and 1 (open gate). The cell input and output activation functions are generally *tanh (x)*, the logistic function or in some cases the identity function.

The weighted peephole connections start from the cell to the gates and are represented with dashed lines. The others inside the block have a fixed weight of 1. The only block outputs to the rest of the network are the result of multiplying the output gate. For more details see [3], [6], [7]. Additionally, in Figs. 9 and 10 an imaginary of what happens inside the memory block can be made as time elapses.

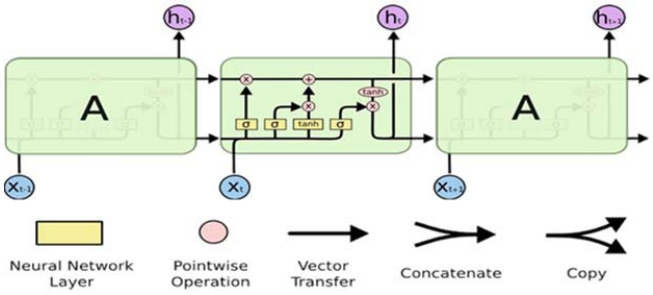

Fig. 9 Memory block operation diagram [6]

The fact of having multiplicative and sums units makes it natural to think of devising variants in the structure of the memory block. This study is focused exclusively on blocks in their standard extended form.

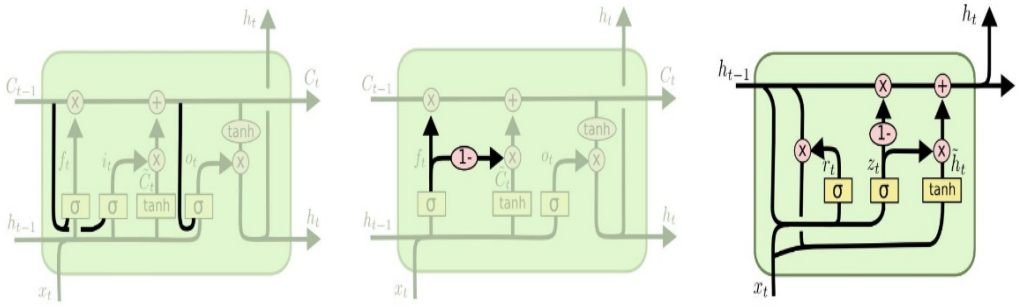

Fig. 10 Structure of a memory block [6]

Mathematical modeling of learning in LSTM. LSTM is a differentiable approximation of functions that is usually trained with the descending gradient] and although originally a doctored form of BPTT was used to approximate the error gradient [7], for the mathematical calculation we will use BPTT without doctoring based on [8]. Fig. 11 shows a schematic illustrating how information is preserved in the LSTM as the time variable elapses. The symbol "–" means that the gate is closed and/or open. The gates are located like this: output is above, forget is on the left and input is under. In addition, it should be noted that the connection weight between neurons i and j is denoted as w_{ij}; H is the number of blocks in the hidden layer; h represents the output of the other blocks in the hidden layer; the symbol s_c^t represents the state of cell c at time t.

Calculations are presented for a single block which is assumed to be single cell. It will also be assumed that the output layer has K units. The procedure for calculating the Forward Pass and Backward Pass equations (BPTT) is shown under. Before starting with the development of BPTT equations for an LSTM, the notation that will be used in the development of the equations is shown in Table III.

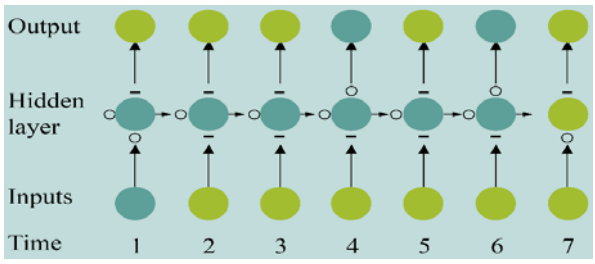

Fig. 11 Operation of learning conservation in LSTM [3]

TABLE III
MATHEMATICAL NOMENCLATURE OF AN LSTM NETWORK

	Input Block	Input Gate	Forget Gate	Output Gate	Memory Cell
Subscript	i	ι	ϕ	ω	c
input	x_i	a_ι^t	a_ϕ^t	a_ω^t	a_c^t, s_c^t
output	x_i	b_ι^t	b_ϕ^t	b_ω^t	$b_c^t = b_\omega^t(s_c^t)$
No. Units	I	n/a	n/a	n/a	C
Activation function	n/a	n/a	n/a	n/a	g (in-cell) h (out-ell)

Forward Pass Equations. From a careful analysis of Fig. 8, we can see that the mathematical representation of Table III results in (20)-(25) [3]:

$$a_l^t = \sum_{i=1}^{I} w_{il}^t x_i^t + \sum_{h=1}^{H} w_{hl}^t b_h^{t-1} + \sum_{c=1}^{C} w_{cl} s_c^{t-1} \quad (20)$$

$$b_\iota^t = f(a_\iota^t) \quad (21)$$

$$a_\emptyset^t = \sum_{i=1}^{I} w_{i\emptyset}^t x_i^t + \sum_{h=1}^{H} w_{h\emptyset}^t b_h^{t-1} + \sum_{c=1}^{C} w_{c\emptyset} s_c^{t-1} \quad (22)$$

$$b_\emptyset^t = f\left(a_\emptyset^t\right) \quad (23)$$

$$a_w^t = \sum_{i=1}^{I} w_{iw}^t x_i^t + \sum_{h=1}^{H} w_{hw}^t b_h^{t-1} + \sum_{c=1}^{C} w_{cw} s_c^t \quad (24)$$

$$b_\omega^t = f(a_\omega^t) \quad (25)$$

Forward Pass Equations. Based on the fact that a variation of the Backpropagation (as mentioned over) will be used, the chain rule must be applied to calculate the partial derivatives. Initially assume the next definitions:

$$\delta_j^t = \frac{\partial E}{\partial a_j^t}, \epsilon_c^t = \frac{\partial E}{\partial b_c^t}, \epsilon_s^t = \frac{\partial E}{\partial s_c^t}$$

Considering the over, and from a careful analysis of Fig. 8 we have the subsequent Backward Pass equations:

Output gate:

$$\delta_w^t = \frac{\partial E}{\partial a_\omega^t} = f'(a_w^t) \sum_{c=1}^{C} \epsilon_c^t h(s_c^t) \quad (26)$$

Cells:

$$\delta_c^t = \frac{\partial E}{\partial a_c^t} = \epsilon_s^t b_\iota^t g'(a_c^t) \quad (27)$$

Forget gate:

$$\delta_\phi^t = \frac{\partial E}{\partial a_\iota^t} = f'(a_\iota^t) \sum_{c=1}^{C} \epsilon_s^t g(a_c^t) \quad (28)$$

Input gate:

$$\delta_\iota^t = \frac{\partial E}{\partial a_\iota^t} = f'(a_\iota^t) \sum_{c=1}^{C} \epsilon_s^t g(a_c^t) \quad (29)$$

States

$$\epsilon_s^t = \frac{\partial E}{\partial s_c^t} = \frac{\partial E}{\partial b_c^t}\frac{\partial b_c^t}{\partial s_c^t} + \frac{\partial E}{\partial s_c^{t+1}}\frac{\partial b_c^{t+1}}{\partial s_c^t} + \frac{\partial E}{\partial b_i^{t+1}}\frac{\partial a_i^{t+1}}{\partial s_c^t} + \frac{\partial E}{\partial b_\phi^{t+1}}\frac{\partial a_\phi^{t+1}}{\partial s_c^t} + \frac{\partial E}{\partial a_\omega^t}\frac{\partial a_\omega^t}{\partial s_c^t} \quad (30)$$

Comparison between MLP and LSTM. To conclude this discussion paper, in Table IV, the most important characteristics between MLP and LSTM are compared; techniques that will be used to characterize PUs in cognitive radio networks to validate the convenience of using LSTM as a predictor of future states spectral channels use by primary users.

TABLE IV
COMPARISON BETWEEN LSTM AND MLP

	MLP	LSTM
Type of ANN	Feed forward	RNN
Feedback	No	Si
Type of Unit	Neuron	Memory block
Hidden Layers	Si	Si
Connections	Feed forward	Feed forward in the same layer
Learning	Backpropagation	BPTT

VI. CONCLUSIONS

The article presents a mathematical analysis about the operation of type MPL, ANN, finally LSTM neural networks.

From the point of view of LSTM, we show the mathematical modeling to obtain Backward Pass equations in LSTM networks; this analysis is very important to develop algorithms that lead to assess its application in fields as telecommunications in lines of research such as Cognitive Radio.

The analysis performed on LSTM to deduce the Backward Pass equations does not exist in the studied literature and there are no indications that they exist on the Internet, therefore it is a resource that is available to be used by the academic and scientific community.

REFERENCES

[1] Zachary, L., J. Berkowitz., C. Elkan., 2015. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019.
[2] Pérez, J., 2002. Predictive models based on discrete time recurrent neural networks. Department of Language and Computer Systems. PhD Thesis, University of Alicante.
[3] Alex, Graves., 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Poland: Springer, ISBN: 978-3642247965.
[4] Neural networks basics. (Online). Accessed on February 7 2015, retrieved from http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm.
[5] Artificial neuronal networks in intensive medicine. An example of application with MPM II variables. (Online). Accessed May 17 2015, retrieved from http://www.medintensiva.org/es/redes-neuronales-artificiales-medicina-intensiva-/articulo/13071859/.
[6] Yan, S. Understanding LSTM networks. (Online). Accessed on August 11 2015 retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[7] Schmidhuber, J., S, Hochrei., 1997. Long short-term memory. Journal Neural computation, 9: 1735-1780.
[8] Ke-Lin, D., M. Swamy., 2013. Neural networks and statistical learning. New York: Springer & Business Media, ISBN: 978-1447170471.