Differential Neural Networks (DNN)

SERGIO LEDESMA1,2, DORA-LUZ ALMANZA-OJEDA1, MARIO-ALBERTO IBARRA-MANZANO1, (Member, IEEE), EDUARDO CABAL YEPEZ2, (Member, IEEE), JUAN GABRIEL AVINA-CERVANTES1, AND PASCAL FALLAVOLLITA2

1School of Engineering, University of Guanajuato, Salamanca 36885, Mexico
2Faculty of Health Sciences, Interdisciplinary School of Health Sciences, University of Ottawa, Ottawa, ON K1N 6N5, Canada
Corresponding author: Sergio Ledesma (selo@ugto.mx)

ABSTRACT In this work, we propose an artificial neural network topology to estimate the derivative of a function. This topology is called a differential neural network because it allows the estimation of the derivative of any of the network outputs with respect to any of its inputs. The main advantage of a differential neural network is that it uses some of the weights of a multilayer neural network. Therefore, a differential neural network does not need to be trained. First, a multilayer neural network is trained to find the best set of weights that minimize an error function. Second, the weights of the trained network and its neuron activations are used to build a differential neural network. Consequently, a multilayer artificial neural can produce a specific output, and simultaneously, estimate the derivative of any of its outputs with respect to any of its inputs. Several computer simulations were carried out to validate the performance of the proposed method. The computer simulation results showed that differential neural networks are capable of estimating with good accuracy the derivative of a function. The method was developed for an artificial neural network with two layers; however, the method can be extended to more than two layers. Similarly, the analysis in this study is presented for two common activation functions. Nonetheless, other activation functions can be used as long as the derivative of the activation function can be computed.

INDEX TERMS Differential neural network, artificial intelligence, neural network structure, derivative estimation, multilayer network.

I. INTRODUCTION
Artificial neural networks are used when a representation model of a system does not exist or when this model is too complex for practical purposes. The main advantage of artificial neural networks is that they can create a system model directly from a set of samples. Furthermore, artificial neural networks can operate even under noise or distortion in the experimental data [1].

In the last few decades, artificial neural networks have been used for pattern recognition, prediction, control, classification, and many other applications, see [2]–[5]. Current research related to artificial neural networks includes methods that can reduce training time, [6]. Research also focuses on reducing human intervention for data preparation and processing [7]. In the last few years, deep learning has become popular in a broad range of fields to solve different kinds of problems, see [8]–[11]. For instance, the authors of [12] use feed-forward artificial neural networks to solve the nonlinear second-order corneal shape model. Thus, scientific efforts to extend the capabilities of artificial neural networks are very important.

In several fields, it is important to analyze how one variable changes over time or how this variable changes with respect to other variables. For instance, in physics, the derivative of the position with respect to time provides the velocity. In business and economics, it is possible to optimize profit using derivative information of other variables. In the same sense, Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) are used in a broad range of fields such as mathematics, physics, astronomy, chemistry, biology, economics, and many other. In past years, artificial neural networks including functional link artificial neural networks have been used to solve differential equations, [13]–[15].

The associate editor coordinating the review of this manuscript and approving it for publication was Wen-Sheng Zhao.
In the same context, the authors of [16] analyzed the solution of linear fractional-order ordinary differential equations using artificial neural networks. Some previous works to apply artificial intelligence to solve differential equations include the methods in [17], [18]. Mall and Chakraverty [19] presented a review of how artificial neural networks have been used through the years to solve differential equations. The authors in [20] approximated the solution of fractional differential equations (FDEs) by using the fundamental properties of artificial neural networks. Recently, the authors in [21] have demonstrated the precision and effectiveness of artificial neural networks to solve high-order linear fractional differential equations. Despite all these works, there is little research performed about how to estimate the partial derivative of a signal using artificial neural networks.

In this article, we propose an artificial network topology called a differential neural network. This type of network can be used to estimate the partial derivative of a variable with respect to another one. In other words, the method proposed in this study can be used to compute the partial derivative of one output of an artificial neural network with respect to one of its inputs. The main contribution of this work is a method that allows the estimation of the partial derivative using an artificial neural network. Therefore, it is possible to use an artificial neural network to create a model directly from the data, and then use our method, to create a second model for the partial derivative of the parameters involved in the process under analysis. For instance, if an artificial neural network is used to model the stock market, then a differential neural network can be used to estimate the rate of change of the stocks.

II. ARTIFICIAL NEURAL NETWORK

In this section, we present a quick review of feed-forward artificial neural networks. First, we begin by describing how the networks are organized in layers. Second, we will explain how scaling can be applied at the input and at the output of the network to normalize the range of the data in the training set. Third, we will describe some common activation functions used in neural networks. These concepts will be used throughout the paper to compute several partial derivatives in different parts of the network. The main objective of this work is to find an expression to compute the partial derivative of one output of the network with respect to one of its inputs.

A feed-forward artificial neural network is composed of a set of layers of neurons [22]. Each neuron in the network is connected to other neurons through a numeric value called weight. Additionally, each neuron produces an activation which is computed by taking a weighted sum of the layer inputs, and then, using a non-linear function, see [23]. During training, the weights are adjusted so that for each input applied to the neural network, each network output is as close as possible to a target value [24].

Figure 1 shows an artificial neural network with $M$ inputs and $R$ outputs. The neural network has two layers. The hidden layer has $N$ neurons, and the output layer has $R$ neurons. Each neuron has anadder and an activation function $f(\cdot)$.

A. INPUT SCALER

The neural network in Figure 1 has a set of input scalers to modify the input values range. The main purpose of input scaling is to reduce the space for the network weight’s possible values, and thus, ease the the network’s training. These scalers can be represented by the following set of linear equations

$$\tilde{x}_i = \mu_i x_i + \beta_i, \quad i = 1, 2, \cdots, M$$  \hspace{1cm} (1)

where $\mu_i$ and $\beta_i$ are constant values determined from the input range.

B. OUTPUT SCALER

If the neurons in the output layer of the network have an activation function that limits the output values, it is necessary to scale the output of the network. For instance, if the activation function is the hyperbolic tangent ($\tanh$), the range for $\tilde{z}_j$ is from $-1$ to $1$. In this case, the output scaler must transform each output $\tilde{z}_j$ to the actual range of the target values. The artificial neural network in Figure 1 has a scaler for each neuron in the output layer. These scalers are represented by

$$z_j = \delta_j \tilde{z}_j + \lambda_j, \quad j = 1, 2, \cdots, R$$  \hspace{1cm} (2)

where $\delta_j$ and $\lambda_j$ are constants determined using the range of the target values in the training set and the type of activation function of the neurons in the network.

C. ACTIVATION FUNCTION

The activation function is an important component of an artificial neural network because it affects the ability of the network to converge during training. An activation function is
a mathematical expression that defines the output of a neuron using the input value [25], [26]. One common activation function is the logistic function, \( \text{logsig} \); it has a sigmoid curve with equation
\[
f(y) = \logsig(y) = \frac{1}{1 + e^{-y}}.
\]  
(3)

Another common activation function that has been used in artificial neural networks is
\[
f(y) = \tanh(ay)
\]  
(4)

where the constant \( a \) commonly takes a value of 1.5 [23]. The main difference between the \( \logsig \) function and the \( \tanh \) function is the output range produced by the neuron. Another very common activation function in deep learning is the Rectified Linear Unit, ReLU, described as
\[
f(y) = \begin{cases} 
  y & y > 0 \\
  0 & \text{otherwise}.
\end{cases}
\]

The ReLU function is computationally efficient and allows the artificial neural network to converge quickly, see [27].

III. METHODOLOGY

For clarity, the discussion presented in this section is focused on the artificial neural network of Figure 1. However, our approach can be extended to networks with more than two layers.

Consider the artificial neural network in Figure 2. In this case, we are interested in the estimation of the partial derivative of output \( z_j \) with respect to input \( x_i \), \( \partial z_j / \partial x_i \). This derivative can be computed using the chain rule as
\[
\frac{\partial z_j}{\partial x_i} = \frac{\partial x_i}{\partial x_i} \cdot \frac{\partial q_j}{\partial x_i} \cdot \frac{\partial z_j}{\partial q_j}.
\]

The first partial derivative in Equation 5, \( \partial x_i / \partial x_i \), will be referenced as the input derivative. The second partial derivative, \( \partial q_j / \partial x_i \), will be referenced as the internal derivative. Finally, the last partial derivative, \( \partial z_j / \partial q_j \), will be referenced as the output derivative. Figure 2 shows these three partial derivatives; they are necessary to compute \( \partial z_j / \partial x_i \). The figure also shows those parts of the artificial neural network that are involved in the computation of each partial derivative.

A. INPUT DERIVATIVE

The input derivative, \( \partial x_i / \partial x_i \), for each input of the network can be easily computed from Equation 1. Thus, this set of input derivatives can be expressed as
\[
\frac{\partial x_i}{\partial x_i} = \mu_i
\]  
(6)

where \( i = 1, 2, \ldots, M \) and the values of \( \mu_i \) are scaling constants.

B. INTERNAL DERIVATIVE

In this section, the internal derivative \( \partial q_j / \partial x_i \) indicated in Figure 2 is computed. From this figure, it can be seen that the values of \( q_j \) can be computed as
\[
q_j = \sum_{k=1}^{N} h_jk \cdot p_k + h_jB
\]

where \( j = 1, 2, \ldots, R \). Thus, the partial derivative of \( q_j \) with respect \( x_i \) can be computed as
\[
\frac{\partial q_j}{\partial x_i} = \sum_{k=1}^{N} h_jk \cdot \frac{\partial p_k}{\partial x_i}.
\]

By applying the chain rule in Equation 8 and Figure 2, we get
\[
\frac{\partial q_j}{\partial x_i} = \sum_{k=1}^{N} h_jk \cdot \frac{\partial p_k}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i}.
\]

The value of \( \partial p_k / \partial y_k \) can be computed from the activation function used in the neurons of the artificial neural network. For the \( \logsig \) activation function of Equation 3, this derivative can be expressed as
\[
\frac{\partial p_k}{\partial y_k} = p_k (1 - p_k).
\]

where \( p_k \) is the activation produced by neuron \( k \) in the hidden layer of the network in Figure 2. However, if the neurons in the hidden layer use the activation function \( \tanh \) of Equation 4, the value of \( \partial p_k / \partial y_k \) can be expressed as
\[
\frac{\partial p_k}{\partial y_k} = a (1 - p_k^2)
\]

\( a \) is the Rectified Linear Unit, ReLU, described as
\[
f(y) = \begin{cases} 
  y & y > 0 \\
  0 & \text{otherwise}.
\end{cases}
\]

\( a \) commonly takes a value of 1.5 [23]. The main difference between the \( \logsig \) function and the \( \tanh \) function is the output range produced by the neuron. Another very common activation function in deep learning is the Rectified Linear Unit, ReLU, described as
\[
f(y) = \begin{cases} 
  y & y > 0 \\
  0 & \text{otherwise}.
\end{cases}
\]
By substituting Equation 11 in Equation 9, we obtain
\[
\frac{\partial q_j}{\partial x_i} = \sum_{k=1}^{N} h_{jk} (1-p_k^2) \frac{\partial y_k}{\partial x_i}.
\] (12)
The last partial derivative in Equation 12, \(\partial y_k/\partial x_i\), can be obtained from the network in Figure 2 as
\[
\frac{\partial y_k}{\partial x_i} = w_{kj}, \quad i = 1, 2, \ldots, M.
\] (13)
Consequently, a final expression for the internal derivative can be computed by substituting Equation 13 in Equation 12 to get
\[
\frac{\partial q_j}{\partial x_i} = a \sum_{k=1}^{N} h_{jk} (1-p_k^2) w_{ki}
\] (14)
where the values of \(w_{ki}\) are the weights in the hidden layer, \(i = 1, 2, \ldots, M\) and \(j = 1, 2, \ldots, R\). That is, there is a value of \(i\) for each input in the network, and there is a value of \(j\) for each output.

**C. OUTPUT DERIVATIVE**

In this section, we evaluate the last partial derivative of Equation 5, \(\partial z_j/\partial q_j\). This derivative can be determined by applying the chain rule to the neurons in the output layer of the network in Figure 2. Thus, we get
\[
\frac{\partial z_j}{\partial q_j} = \frac{\partial z_j}{\partial \bar{z}_j} \frac{\partial \bar{z}_j}{\partial q_j}.
\] (15)
The first partial derivative of Equation 15, \(\partial \bar{z}_j/\partial q_j\), can be computed using the activation function of the neurons in the output layer. If the neurons in the output layer use the activation function \(tanh\) of Equation 4, this derivative can be expressed as
\[
\frac{\partial \bar{z}_j}{\partial q_j} = a (1-\bar{z}_j^2)
\] (16)
where \(j = 1, 2, \ldots, R\) and \(\bar{z}_j\) is the activation produced by neuron \(j\) in the output layer of the network in Figure 2. The second partial derivative of Equation 15 can be computed directly from the equation of the output scaler. By using Equation 2, the output derivative can be expressed as
\[
\frac{\partial z_j}{\partial \bar{z}_j} = \delta_j
\] (17)
where \(j = 1, 2, \ldots, R\) and \(\delta_j\) is a constant determined by the activation function used in neuron \(j\) of the output layer and the range of the target values for output \(j\). Thus, the output derivative, \(\partial \bar{z}_j/\partial q_j\), can be computed by substituting Equation 16 and Equation 17 in Equation 15 to obtain
\[
\frac{\partial z_j}{\partial q_j} = a (1-\bar{z}_j^2) \delta_j
\] (18)
where \(j = 1, 2, \ldots, R\).

**D. DIFFERENTIAL NEURAL NETWORK (DNN)**

Finally, the value of \(\partial z_j/\partial x_i\) can be computed by substituting Equations 6, 14 and 18 in 5, we get
\[
\frac{\partial z_j}{\partial x_i} = \mu_i a \sum_{k=1}^{N} h_{jk} (1-p_k^2) w_{ki} a (1-\bar{z}_j^2) \delta_j.
\] (19)
By moving to the left of the summation those elements that do not depend on \(k\), Equation 19 can be expressed as
\[
\frac{\partial z_j}{\partial x_i} = \mu_i \delta_j a^2 (1-\bar{z}_j^2) \left( \sum_{k=1}^{N} h_{jk} (1-p_k^2) w_{ki} \right).
\] (20)
Equation 20 can be used to estimate the derivative of output \(j\) with respect to input \(i\) when the neurons use the activation function \(tanh\). When the neurons use the \(logsig\) activation function, the derivative can be estimated using the following equation
\[
\frac{\partial z_j}{\partial x_i} = \mu_i \delta_j \bar{z}_j (1-\bar{z}_j) \sum_{k=1}^{N} h_{jk} p_k (1-p_k) w_{ki}
\] (21)
where \(i = 1, 2, \ldots, M\) and \(j = 1, 2, \ldots, R\). The interesting fact about Equation 20 (or Equation 21) is that all the values in this equation are directly available from an artificial neural network. For instance, the weights in the hidden layer \(w_{ki}\) and the weights in the output layer \(h_{jk}\) are estimated during the normal training of the network. In the same sense, the activations of the neurons in the hidden layer \(p_k\) and the activations in the output layer \(\bar{z}_j\) are simultaneously computed when the complete artificial neural network is activated. This implies that it is possible to create an artificial neural network that can simultaneously compute the output and its derivative. Figure 3 shows the structure of a differential neural network to compute the derivative of output \(j\) with respect to input \(i\). As it can be seen from the diagram, the partial derivative \(\partial z_j/\partial x_i\) only involves those weights that are coming from input \(i\), \(w_{1i}, w_{2i}, \ldots w_{Ni}\) and those weights that are going to output \(j\), \(h_{j1}, h_{j2}, \ldots, h_{jN}\).

Figure 3 shows the artificial neural network topology to estimate the derivative. In this study, this topology or configuration is denominated a differential neural network, DNN. The hidden layer of a DNN is similar to the hidden layer of a feed-forward network. However, the output layer of a DNN is completely different from the output layer of the feed-forward network and requires the derivatives of the activation function as shown in Figure 3.

Figure 4 describes how to use the algorithm proposed in this article to estimate the derivative of a function using an artificial neural network. First, the weights of a multilayer neural network are estimated using an appropriate training set. When the training of the network is completed, each case in a dataset is applied to the network to compute the activations of the neurons in the hidden layer; \(p_1, p_2, \ldots, p_N\), see Figures 1 and 3. At this point, the neuron activation \(\bar{z}_j\) is computed. Finally, Equation 21 is used to compute the desired derivative.
IV. COMPUTER SIMULATIONS AND RESULTS

Several computer simulations were performed to establish the validity of the method proposed in this article. These computer simulations were performed using the simulator Neural Lab [28]. The analysis presented in this section focuses on three functions that have well-known derivatives: \( \sin(x) \), \( x^2 \) and \( \ln(x) \).

A. DERIVATIVE ESTIMATION OF THE SINE FUNCTION, \( \sin(x) \)

The simulations to estimate the derivative of \( \sin(x) \) began by creating a training set with 4000 uniformly distributed values for \( x \) in the range from zero to \( 2\pi \). The respective target for each case was the value of \( \sin(x) \). Once the training set was created, an artificial neural network with sixteen neurons in the hidden layer was trained using the parameters shown in Table 1. The artificial neural network had the structure shown in Figure 1 with one hidden layer and one output layer. However, the network had only one input and one output. The input scaler was designed to transform the value of \( x \) to the range \([-1 1]\). Similarly, the output scaler was designed to transform the range \([-0.9 0.9] \) to \([-1 1]\). From Table 1, it can be seen that the network was trained using the method of simulated annealing, and then, the training was improved using the method of the conjugate gradient, see [29]–[31]. After the training was completed, a validation set with 800 random values for \( x \) in the range from zero to \( 2\pi \) was created. As the mean-squared error obtained during training was similar to the mean-squared error obtained during validation, it was concluded that the network was properly trained, see [32].

As the neurons of the artificial neural network had the activation function hyperbolic tangent of Equation 4, Equation 20 was used to estimate the derivative of \( \sin(x) \). Figure 5 shows the estimation of the derivative of \( \sin(x) \) using the artificial neural network. Additionally, Figure 6 shows the error of the estimation made by the artificial neural network. As it can be seen from the figure, the derivative has an error that is less than 0.001 for most of the values in the range from zero to \( 2\pi \).

B. DERIVATIVE ESTIMATION OF THE PARABOLA, \( x^2 \)

For this function, a training set with 4000 uniformly distributed values \( x \) in the range from \(-2 \) to \( 2 \) was created.
FIGURE 6. Error for the approximation of the derivative of \( \sin(x) \).

The target values of the training set were the values of \( x^2 \). Then, an artificial neural network with 16 neurons in the hidden layer using the same training parameter in Table 1. After the network’s training was completed, the network performance was validated using a data set with 800 random values for \( x \) in the range from \(-2\) to \(2\). The computer simulations showed that the mean squared error for both training and validation was similar. The derivative of \( x^2 \) was estimated substituting the weights and activations of the network in Equation 20. Figure 7 shows the derivative estimation for \( x^2 \), as it can be seen from the graph, the estimation of the derivative is very good for most of the range from \(-2\) to \(2\). However, the estimation of the derivative presents an abnormal behavior around \(-2\) and \(2\). As the artificial neural network was not trained with values outside this range, the network does not have enough information to properly estimate the derivative in the lower limit as well as in the upper limit. Figure 8 shows the difference between the actual value of the derivative \( 2x \) and the estimation produced by the network. From the graph, it can be seen that the error remains below 0.01 for most values of \( x \) in the range from \(-2\) to \(2\). Observe that, in this case, it is possible to try improving the performance of the network by increasing the number of training cases and the number of neurons in the hidden layer.

C. DERIVATIVE ESTIMATION FOR THE NATURAL LOGARITHM FUNCTION, \( \ln(x) \)

The last function used to validate the method proposed in this work is the natural logarithm function. The training set was built with 4000 samples with uniformly distributed values in the range from 1 to 10. Once the training set was ready, one artificial neural network with sixteen neurons in the hidden layer was trained using the parameters in Table 1. Then, the artificial neural network’s performance was validated using a dataset with 800 random samples in the range from 1 to 10.

Figure 9 shows the estimation of the derivative of \( \ln(x) \) computed using the artificial neural network. As it can be seen from the graph in the figure, the estimation of the derivative is very good in most of the range. However, there is a little deviation in the lower limit around \( x = 1 \). Figure 10 shows the difference between the exact value of the derivative and the estimation computed by the artificial neural network. For this function, the error is below 0.001 for most of the range.
V. CONCLUSION

In the last few decades, artificial neural networks have been applied to a broad range of fields. Artificial neural networks can learn from data samples, and therefore, can be used in those applications where a model is missing or when the model is too complex. In this work, we propose a new network topology called differential neural network. This network topology can be used to estimate the partial derivative of a function. Differential neural networks use the same weights of a multi-layer neural network but in a different configuration. First, a multi-layer neural network is trained with a dataset to find the best set of weights that minimizes the network error. Second, the weights of this network and its neurons activations can be used to estimate the derivative of any output of the network with respect to any of its inputs. Therefore, an artificial neural network can be extended to simultaneously estimate the partial derivative of any of its outputs with respect to any of its inputs.

Three different artificial neural networks were designed and trained. The first network was trained using a data set with samples from the sine function. The second network was trained with samples from a parabola, and the last network was trained with samples from the natural logarithm. Once the three networks were trained and validated, three different differential neural networks were build using values taken from the trained networks. The results from the computer simulations showed that a differential neural network could accurately estimate the partial derivative directly from data samples. The main advantage of the proposed method is that the same weights that were obtained during the training of a neural network are used for a differential neural network. Thus, a differential neural network does not need further training.

ACKNOWLEDGMENT

The authors would like to thank DAIP, University of Guanajuato, and the Faculty of Health Sciences, University of Ottawa, Canada, for their sponsorship in the realization of this work. This work was developed during the sabbatical stay of Sergio Ledesma at the Faculty of Health Sciences, University of Ottawa.

REFERENCES

[1] S. Marsland, Machine Learning: An Algorithmic Perspective (Chapman & Hall/CRC Machine Learning & Pattern Recognition), 2nd ed. Boca Raton, FL, USA: CRC Press, 2015, pp. 190–210.
[2] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” Science, vol. 349, no. 6245, pp. 255–260, Jul. 2015.
[3] C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” IEEE Access, vol. 5, pp. 21954–21961, 2017.

[4] W. Liu, M. Zhang, Z. Luo, and Y. Cai, “An ensemble deep learning method for vehicle type classification on visual traffic surveillance sensors,” IEEE Access, vol. 5, pp. 24417–24425, 2017.

[5] B. Jin, Z. Jing, and H. Zhao, “Incremental and decremental extreme learning machine based on generalized inverse,” IEEE Access, vol. 5, pp. 20852–20865, 2017.

[6] S. Ledesma, M.-A. Ibarra-Manzano, E. Cabal-Yepez, D.-L. Almanza-Ojeda, and J.-G. Avina-Cervantes, “Analysis of data sets with learning conflicts for machine learning,” IEEE Access, vol. 6, pp. 45562–45570, 2018.

[7] R. J. Martin, S. L. Swapna, and S. Sujatha, “Adopting machine learning models for data analytics—A technical note,” Int. J. Comput. Sci. Eng., vol. 6, no. 10, pp. 359–364, Oct. 2018.

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, pp. 436–444, May 2015.

[9] V. Mnih, K. Kavukcuoglu, and D. Silver, “Human-level control through deep reinforcement learning,” Nature, vol. 518, pp. 529–533, 2015.

[10] W. Liu, Z. Wang, Y. Liu, N. Zheng, Y. Liao, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” Neurocomputing, vol. 234, pp. 11–26, Apr. 2017.

[11] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, “Deep convolutional neural network for inverse problems in imaging,” IEEE Trans. Image Process., vol. 26, no. 9, pp. 4509–4522, Sep. 2017.

[12] M. Umar, F. Amin, H. A. Wahab, and D. Baleanu, “Unsupervised constrained neural network modeling of boundary value cornel model for eye surgery,” Appl. Soft Comput., vol. 85, Dec. 2019, Art. no. 105826.

[13] M. A. Z. Raja, J. A. Khan, and I. M. Qureshi, “A new stochastic approach for solution of Riccati differential equation of fractional order,” Ann. Math. Artif. Intell., vol. 60, nos. 3–4, pp. 229–250, Dec. 2010.

[14] S. Mall and S. Chakraverty, “Application of Legendre neural network for solving ordinary differential equations,” Appl. Soft Comput., vol. 43, pp. 347–356, Jun. 2016.

[15] M. A. Z. Raja, M. A. Manzar, and R. Samar, “An efficient computational intelligence approach for solving fractional order Riccati equations using ANN and SQP,” Appl. Math. Model., vol. 39, nos. 10–11, pp. 3075–3093, Jun. 2015.

[16] A. Jafari, S. Measoomy Nia, A. K. Golmankhaneh, and D. Baleanu, “On artificial neural networks approach with new cost functions,” Appl. Math. Comput., vol. 339, pp. 546–555, Dec. 2018.

[17] A. L. Meade and A. A. Fernandez, “The numerical solution of linear ordinary differential equations by feedforward neural networks,” Math. Comput. Model., vol. 19, no. 12, pp. 1–25, Jun. 1994.

[18] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” IEEE Trans. Neural Netw., vol. 9, no. 5, pp. 987–1000, Sep. 1998.

[19] S. Mall and S. Chakraverty, “Comparison of artificial neural network architecture in solving ordinary differential equations,” Adv. Artif. Neural Syst., vol. 2013, pp. 1–12, Jan. 2013.

[20] M. Pakdaman, A. AMDadian, S. Effati, S. Salahshour, and D. Baleanu, “Solving differential equations of fractional order using an optimization technique based on training artificial neural network,” Appl. Math. Comput., vol. 293, pp. 81–95, Jan. 2017.

[21] F. Rostami and A. Jafari, “A new artificial neural network structure for solving high-order linear fractional differential equations,” Int. J. Comput. Math., vol. 95, no. 3, pp. 528–539, Mar. 2018.

[22] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. London, U.K.: Pearson, 2020, pp. 122–204.

[23] T. Masters, Deep Belief Nets in C++ and CUDA C: Restricted Boltzmann Machines and Supervised Feedforward Networks, vol. 1. Ithaca, NY, USA: Apress, 2017, pp. 1–11.

[24] T. Masters, Deep belief Nets in C++ and CUDA C: Convolutional Nets, 2nd ed. Ithaca, NY, USA: Apress, 2018, pp. 28–37.

[25] J. D. Kelche, B. M. Namee, and A. D’Arcy, Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies, 2nd ed. Cambridge, MA, USA: MIT Press, 2020.

[26] C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook, 1st ed. Cham, Switzerland: Springer, 2018, pp. 53–103.

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (Adaptive Computation and Machine Learning Series). Cambridge, MA, USA: MIT Press, 2016, pp. 267–309.

[28] S. Ledesma, M.-A. Ibarra-Manzano, M.-G. Garcia-Hernandez, and D.-L. Almanza-Ojeda, “Neural lab a simulator for artificial neural networks,” in Proc. IEEE Comput. Conf., Jul. 2017, pp. 716–721.

[29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes in C++: The Art of Scientific Computing, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 2007, pp. 549–555.
EDUARDO CABAL YEPEZ (Member, IEEE) received the M.Eng. degree from the Facultad de Ingeniería Mecánica Electrica y Electrónica (FIMEE), Universidad de Guanajuato, Mexico, in 2001, and the Ph.D. degree from the University of Sussex, U.K., in 2007. In April 2008, he joined the División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato, where he is currently aLeading Professor and the Dean of the Multidisciplinary Studies Department. He has been the author/coauthor of more than 31 research articles published in journals registered in the journal citation report (JCR). His current research interests include digital instrumentation, digital image and signal processing, artificial intelligence, robotics, smart sensors, real-time processing, mechatronics, FPGAs, power quality, and embedded systems. He is an Active Member of the IEEE Instrumentation and Measurement Society, the IEEE Industrial Electronics Society, and the IEEE Signal Processing Society. He serves as an Area Editor for the International Journal of Computers and Electrical Engineering (Elsevier) and an Associate Editor for the International Journal of Digital Signal Processing (Elsevier) and the IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT.

JUAN GABRIEL AVINA-CERVANTES received the B.S. degree in communications and electronics engineering and the master’s degree in electrical engineering (instrumentation and digital systems) from the University of Guanajuato, in 1998 and 1999, respectively, and the Ph.D. degree in informatics and telecommunications from the Institut National Polytechnique de Toulouse and the LAAS-CNRS, France, in 2005. He is currently a Researcher and a full-time Professor with the University of Guanajuato. His research interests include the artificial vision for outdoor mobile robotics, pattern recognition, optimal control systems, and image processing.

PASCAL FALLAVOLLITA led the research activities in medical augmented reality with the Chair for Computer Aided Medical Procedures, Technische Universität München. He is currently an Associate Professor with the Interdisciplinary School of Health Sciences, University of Ottawa, Canada. His research interests include mixed-reality visualization, design and evaluation of health technologies, and application of artificial intelligence for better health. He received the Postdoctoral Fellowship in collaboration with the teams at the Perk Laboratory, Queen’s University, and the Johns Hopkins University (JHU) School of Medicine, from 2008 to 2010.

***