Neurocomputing 325 (2019) 1–19

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

On extreme learning machines in sequential and time series prediction: A non-iterative and approximate training algorithm for recurrent neural networks

Yara Rizk, Mariette Awad*

Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 29 December 2017
Revised 27 June 2018
Accepted 12 September 2018
Available online 9 October 2018

Communicated by Ning Wang

Keywords:
Recurrent neural networks
Non-iterative training
Extreme learning machines
Random vector functional link
Time series prediction
Partially observable Markov decision process

ABSTRACT

Recurrent neural networks (RNN) are a type of artificial neural networks (ANN) that have been successfully applied to many problems in artificial intelligence. However, they are expensive to train since the number of learned weights grows exponentially with the number of hidden neurons. Non-iterative training algorithms have been proposed to reduce the training time, mainly on feedforward ANN. In this work, the application of non-iterative randomized training algorithms to various RNN architectures, including Elman RNN, fully connected RNN, and long short-term memory (LSTM), are investigated. The mathematical formulation and theoretical computational complexity of the proposed algorithms are presented. Finally, their performance is empirically compared to other iterative RNN training algorithms on time series prediction and sequential decision-making problems. Non-iteratively-trained RNN architectures showed promising results as significant training speedup of up to 99%, and improved repeatability were achieved compared to backpropagation-trained RNN. Although the decrease in prediction accuracy was found to be statistically significant based on Friedman and ANOVA testing, some applications like real-time embedded systems can tolerate and make use of that.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Sequential decision-making and time series prediction problems are an integral part of smart cities with applications in smart electric grids, intelligent transportation systems, robotic systems, Internet of things and others. These applications generally require computationally efficient algorithms that are difficult to deploy on energy aware and computationally limited platforms.

Artificial neural networks (ANN), and recurrent neural networks (RNN) specifically, have been successfully applied to many problems in artificial intelligence and machine learning including time series prediction and sequential decision-making problems. While they are the most powerful neural networks and have exceeded human performance on some applications [1], RNN training, which mainly relies on iterative optimization, is computationally expensive and faces difficulty in resource-challenged applications. Thus, non-iteratively trained ANN such as random weight neural networks (RWNN) [2], random vector functional link (RVFL) [3], random activation weight network (RAWN) [4], and extreme learning machines (ELM) [5] have been investigated to reduce training costs.

In this paper, we investigate training various RNN architectures non-iteratively, to reduce the computational complexity of RNN training. Specifically, we apply a non-iterative randomized offline training algorithm based on Moore–Penrose pseudoinverse, which was proposed by Schmidt et al. in 1992 [2], Pao et al. in 1994 [3], Te Braake and Van Straten in 1995 [4] and Huang et al. in 2004 [5], on a variety of RNN architectures. The following RNN architectures are considered in this work: Elman [6], Jordan [7], fully connected [8], non-linear auto regressive moving average with exogenous input (NARMAX) [9], and long short-term memory (LSTM) [10]. Networks with and without direct links (DL), connections between the input and output layer neurons [11], are also considered.

Since learning the weights of the recurrent connections is not a linear problem, we propose two training algorithms to address this issue: the first, ELM-rand, randomly assigns values to the recurrent connection weights, while the second, ELM-lin, linearizes the non-linear activation functions and learns the recurrent connection weights using one of the non-iterative training algorithms mentioned above. Table 1 summarizes the various network architecture/training algorithm combinations investigated in this work

* Corresponding author.
E-mail addresses: yar01@aub.edu.lb (Y. Rizk), mariette.awad@aub.edu.lb (M. Awad).

https://doi.org/10.1016/j.neucom.2018.09.012
0925-2312/© 2018 Elsevier B.V. All rights reserved.

Acronyms

Acronym	Definition
ANN	Artificial neural network
Avg	Average
BPTT	Back-propagation through time
DL	Direct links
ELM	Extreme learning machines
LSTM	Long short-term memory
MAE	Mean absolute error
MSE	Mean square error
NARMAX	Non-linear auto regressive moving average with exogenous input
NARMA	Non-linear auto regressive moving average
NARX	Non-linear auto regressive with exogenous input
ELM-rand	ELM-based non-iterative training algorithm with random recurrent weights
ELM-lin	ELM-based non-iterative training algorithm with a linearized activation function
nMAE	Normalized mean absolute error
nRMSE	Normalized root mean square error
PID	Proportional-integral-derivative
POMDP	Partially observable Markov decision process
RAWN	Random activation weights network
RMSE	Root mean square error
RNN	Recurrent neural network
RVFL	Random vector functional link
RWNN	Random weight neural network
std	Standard deviation
SVR	Support vector regression

Nomenclature

Symbol	Definition
b	Neuron bias vector
b_i	Element in neuron bias vector
b_λ	Neuron bias for a forget gate in an LSTM block
b_{in}	Neuron bias for an input gate in an LSTM block
b_o	Neuron bias for an output gate in an LSTM block
b_c	Neuron bias for a state cell in an LSTM block
$c(t)$	State cell vector at time t in an LSTM block
e	Prediction error
$E[.]$	Expectation
$f(t)$, $f_i(t)$	Output of a hidden neuron at time t
F	Dimension of an input vector
h_{ij}	Element of H
H	Output matrix of hidden layer
i, j, k, l	Indexing variables
$in(t)$	Input gate vector at time t in an LSTM block
M	Number of hidden layer neurons
N	Number of input vectors
$o(t)$	Output gate vector at time t in an LSTM block
Q	Degree of the output polynomial
R	Degree of the error polynomial
t	Time
U_λ	Recurrent connections' weight matrix for a forget gate in an LSTM block
U_{in}	Recurrent connections' weight matrix for an input gate in an LSTM block
U_o	Recurrent connections' weight matrix for an output gate in an LSTM block
U_c	Recurrent connections' weight matrix for a state cell in an LSTM block
w	Input layer weights vector
w_i, w_{ij}	Elements in input layer weights vector
W_λ	Weight matrix for a forget gate in an LSTM block
W_{in}	Weight matrix for an input gate in an LSTM block
W_o	Weight matrix for an output gate in an LSTM block
W_c	Weight matrix for a state cell in an LSTM block
x	Input vector
y	Target output vector
y_{max}, y_{min}	Maximum, minimum value in the target output vector
$\hat{y}$	Predicted output vector
z	Dummy variable
α	Recurrent connections' weights
α_i, α_{ik}, α_{ikl}	Element of the recurrent connections' weights vector
β	Output weights vector
β_i	Element in an output weights vector
$\lambda(t)$	Forget gate vector at time t in an LSTM block
θ	Parameter vector
ϕ_i	Functional link transformation in ANN and RNN networks with direct links
c	Constant
$g(.)$	Activation function
$g_\lambda(.)$	Activation function for a forget gate in an LSTM block
$g_{in}(.)$	Activation function for an input gate in an LSTM block
$g_o(.)$	Activation function for an output gate in an LSTM block
$g_f(.)$	Activation function for a cell gate in an LSTM block

(denoted by "X") and those presented in the literature. In addition, "N" denotes the absence of DL while "Y" denotes their presence.

The proposed algorithms are validated on the double inverted pendulum problem modeled as a partially observable Markov decision process (POMDP) and 14 publicly available time series prediction databases. Compared to RNN trained using the iterative back-propagation algorithm, non-iteratively trained RNN achieved faster training time and better repeatability while incurring an increase in prediction error.

The contributions of this work include:

1. proposing two non-iterative training algorithms for multiple RNN architectures including fully connected RNN and LSTM,
2. comparing the theoretical computational complexity of various RNN architectures when trained non-iteratively,

Table 1
ANN architecture and training algorithm combinations proposed in this work (denoted by X).

Network	DL	Iterative training	Non-iterative training
Elman	N	[6]	X
Elman	Y	–	X
Jordan	N	[7]	[12]
Jordan	Y	[13]	X
NARX/NARMAX	N	[9]	X
NARX/NARMAX	Y	–	X
Fully connected RNN	N	[8]	X
Fully connected RNN	Y	–	X
LSTM	N	[10]	X
LSTM	Y	–	X
Feedforward	N	[14]	[2,4,5]
Feedforward	Y	[11]	[3]

3. and empirically comparing the proposed algorithms to other algorithms in the literature on multiple publicly available datasets for time series regression and control problems.

The rest of this paper is organized as follows. Section 2 summarizes related work. Section 3 presents the proposed training algorithms. Section 4 presents a theoretical analysis of the proposed algorithms before discussing the simulation results in Section 5. Finally, Section 6 concludes with some final remarks.

2. Literature review

In this section, we survey existing work on RNN architectures and non-iterative training algorithms for various applications including classification, regression and clustering of sequential and non-sequential data.

2.1. Recurrent neural networks

RNNs are the most powerful ANN and are considered general computers [1]; the network's weights are its program, and changing them implies changing the program it runs. The main difference between RNN and ANN is the presence of recurrent connections that form cyclic graphs. Many RNN architectures and training algorithms have been proposed in the literature for various applications; we discuss a few in what follows.

2.1.1. Network architectures

Various methods of introducing cycles in a graph yield different types of RNN. In this work, we consider the six architectures illustrated in Fig. 1. Some more complex RNN architectures, which are not within the scope of this work, include bidirectional RNN which combine the output of two RNN, each processing the input sequence in a different direction [15], hierarchical RNN which feed the output of one RNN to another RNN [16], continuous time RNN [8], complex-valued RNN [17], and time-delayed RNN with reaction-diffusion terms [18,19].

The simplest forms of RNN include Elman [6] and Jordan RNN [7], described by (1) and (2) respectively, which are single hidden layer networks with context neurons. Context neurons introduce recurrence into the network by feeding back signals in the network. Jordan networks provide context in terms of the predicted output. Although simpler to training using traditional backpropagation, Jordan RNN cannot represent an arbitrary dynamical system [20]. On the other hand, Elman networks provide context in terms of the internal state of the network which allows them to represent any Qth order dynamical system, where Q is the number of context neurons [21]. However, these networks are difficult to train using backpropagation as the network depth and size increases, due to the vanishing gradient problem [10].

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i f_i(t)$$

$$f_i(t) = g\left(w_i^T x(t) + \sum_{k=1}^{Q} \alpha_{ik} f_i(t-k) + b_i \right) \tag{1}$$

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i g\left(w_i^T x(t) + \sum_{k=1}^{Q} \alpha_{ik} \hat{y}(t-k) + b_i \right) \tag{2}$$

Recurrent NARMAX networks, described by (3) where $e(t) = y(t) - \hat{y}(t)$, have been proposed for non-linear time series prediction using ANN [9]. They were trained using BPTT but proven to be special cases of fully connected RNN architectures [9]. Determining the order of the output and error polynomials is generally based on a grid search and affects the number of input layer neurons, as shown in Fig. 1(c). In a fully connected RNN architecture, which is described by (4), neurons are connected to all other neurons which include neurons in the same, previous and subsequent layers. A fully connected RNN is the most general RNN architecture.

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i g\Bigg(\sum_{j=1}^{F} w_{ij} x_j + \sum_{j=F+1}^{F+Q} w_{ij} y(t-j+F) + \sum_{j=F+Q+1}^{F+Q+R} w_{ij} e(t-j+F+Q) + b_i \Bigg) \tag{3}$$

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i f_i(t)$$

$$f_i(t) = g\left(w_i^T x(t) + \sum_{k=1}^{Q} \sum_{l=1}^{M} \alpha_{ikl} f_l(t-k) + b_i \right) \tag{4}$$

LSTM networks were proposed to solve the vanishing gradient problem, present when training RNN with backpropagation [10]. LSTM cells replace neurons as the network's basic units, as shown in Fig. 1(e). An LSTM cell consists of a typical linear neuron and multiplicative gates that feedback outputs and inputs to the neuron while controlling their effects on the linear unit through forgetting factors and weights. The output of the LSTM block is described by (5), where $\circ$ represents the Hadamard product of two matrices.

$$\begin{aligned}
&\hat{y}(t) = \sum_{i=1}^{M} \beta_i f_i(t) \\
&\lambda(t) = g_\lambda(W_\lambda x(t) + U_\lambda f(t-1) + b_\lambda) \\
&in(t) = g_{in}(W_{in} x(t) + U_{in} f(t-1) + b_{in}) \\
&o(t) = g_o(W_o x(t) + U_o f(t-1) + b_o) \\
&c(t) = \lambda(t) \circ c(t-1) + in(t) \circ g_c(W_c x(t) + U_c f(t-1) + b_c) \\
&f(t) = o(t) \circ g_f(c(t))
\end{aligned} \tag{5}$$

Finally, converting RVFL ANN from a feedforward to a recurrent RVFL architecture, i.e. RNN with DL, simply requires us to pass the input through a preprocessing phase before applying it to any one of the RNN architectures. Fig. 1(f) depicts a Jordan RNN architecture with DL, which is described by (6). However, similar illustrations can be derived for Elman, fully connected, NARMAX and LSTM networks.

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i \phi_i(t) + \sum_{k=1}^{Q} \alpha_k \hat{y}(t-k) \tag{6}$$

2.1.2. Training algorithms

Many training algorithms have been proposed for RNN. Gradient descent based methods are iterative approaches and include backpropagation through time (BPTT) [22] and real-time recurrent learning [23]. Given a sequence of input–output pairs and initial weight values, BPTT unfolds the recurrence in the network through time to transform an RNN to a feedforward network trained using backpropagation. BPTT performs a forward and backward pass at each time step. However, this method is susceptible to local minima and suffers from the vanishing gradient problem when run on a fully connected RNN. Furthermore, BPTT can only be iteratively applied in batch mode. Therefore, an online alternative, real-time recurrent learning, was introduced which performs one forward pass only and computes RNN's derivatives with respect to its parameters without storing hidden states.

Other algorithms that optimally train RNN include a Hessian free optimization algorithm which reduces the computational complexity of the optimization algorithm by avoiding the expensive

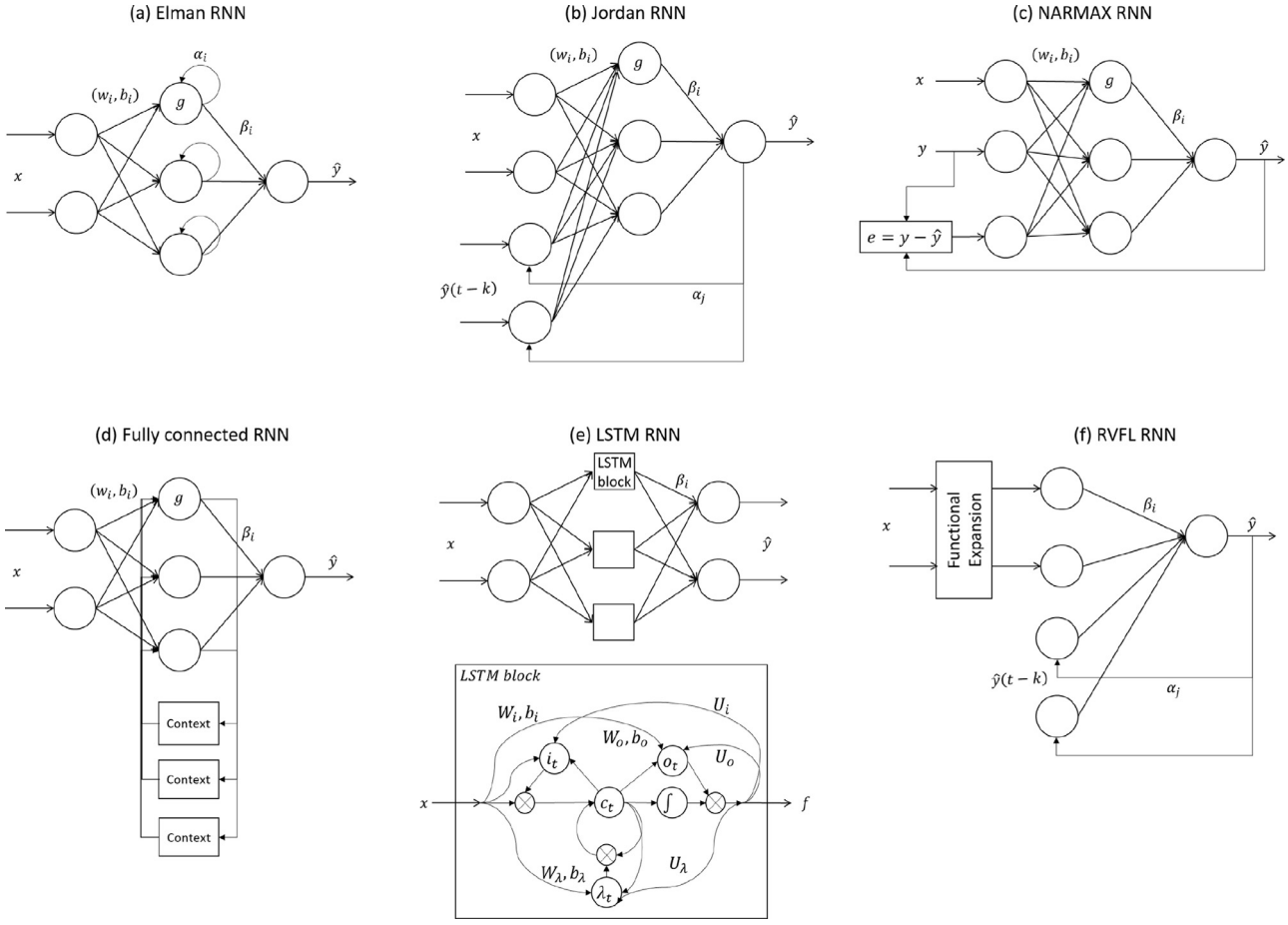

Fig. 1. Different RNN Architectures.

computation of the Hessian matrix using the truncated Newton method [24]. Fu et al. [25] adopted the Levenberg–Marquardt optimization algorithm to learn RNN weights. Bayesian filters including the extended Kalman filters [26] have been used to compute RNN weights and improve RNN training convergence. On the other hand, search algorithms have also been adopted including particle swarm optimization [27], genetic algorithm [28], artificial immune system and ant colony optimization [27].

Even though these iterative training algorithms have performed well in many applications, they remain computationally expensive and require the manual tuning of many hyper-parameters. A non-iteratively trained Jordan RNN was proposed by Ertugrul [12] for electricity load forecasting. The non-iteratively trained network fed back the output layer values to the input layer and was tested on only one-time series prediction problem. The approach was only validated on this simple RNN architecture which cannot learn more complex real-world problems. In our work, we attempt to generalize this non-iterative approach to more powerful RNN architectures.

2.1.3. Reservoir computing

Reservoir computing [29] is a type of RNN with random connections used to map temporal data to higher dimensions. The readout is then trained using some machine learning algorithm which is generally iterative. One example is the echo state machine, a sparsely connected RNN with randomly assigned weights [30]. Liquid state machines are spiking RNNs with random connections that map time-varying inputs to spatio-temporal neuron activation patterns [31]. Instead, we investigate the effect of random connections on densely connected RNN with deterministic neurons using a non-iterative training algorithm.

2.2. Non-iteratively trained artificial neural networks

Many researchers have attempted to develop computationally less expensive algorithms to train ANN. Randomized training algorithms, surveyed in [32], are one approach to reduce training complexity, with some references developing non-iterative training algorithms which improved training time significantly since the connection weights are computed once. To that end, RWNN in 1992 [2], RVFL in 1994 [3], RAWN in 1995 [4] and ELM in 2004 [5] were proposed to train single hidden layer feedforward networks by randomly assigning input weights and computing output weights using the least squares method.

2.2.1. Random weight neural networks

To reduce the training time of feedforward ANN, Schmidt et al. [2] applied non-iterative training to the single hidden layer feedforward ANN. This network architecture, shown in Fig. 2 (without DL), appended the hidden layer with a bias neuron to produce the input–output function in (7). The predicted output of the network $\hat{y}$ is a non-linear function of the input due to the presence of $g(.)$ which represents the activation function. Hereafter, Schmidt et al.'s approach is referred to as RWNN.

The supervised training of ANN involves minimizing the square of the error between the expected output y and the predicted output $\hat{y}$: $\min e^2 = (y - \hat{y})^2$. Since the objective function is quadratic, least squares can be used to solve for these variables in one

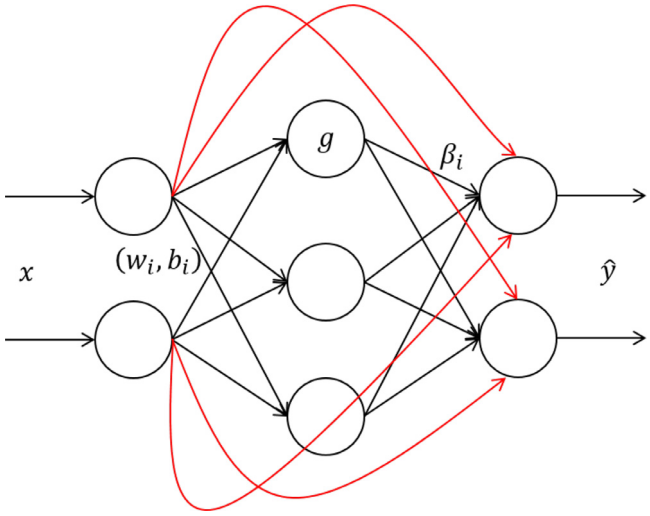

Fig. 2. Single hidden layer feedforward network trained non-iteratively with randomized input weights.

step when the minimization problem is linear in the optimization variables.

The algorithm proposed in [2] randomly assigns input weights w_i from a uniform distribution which leads to a linear optimization problem, as shown in (7). Therefore, the output weights β may be analytically computed. Considering the single hidden layer feedforward ANN in [2], the output matrix H is computed using (8) where an element of H is $h_{ji} = g(w_i^T x_j), i = 1, \ldots, M, j = 1, \ldots, N$, and the $(M+1)$th column contains 1 that multiplies the bias neuron. Note that w_i has dimensionality MF, where F is the number of features in the input vector. The output weights form the parameter vector $\theta = \beta$ with dimension $M+1$ and are learned using the generalized Moore–Penrose pseudoinverse, $\theta = (H^T H)^{-1} H^T y$, where y is the target matrix or desired output in a supervised learning paradigm.

$$\hat{y} = \sum_{i=1}^{M} \beta_i g\left(w_i^T x\right) + \beta_{M+1} \tag{7}$$

$$H = \begin{bmatrix} g(w_1^T x_1) & \ldots & g(w_M^T x_1) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ g(w_1^T x_N) & \ldots & g(w_M^T x_N) & 1 \end{bmatrix}_{N \times (M+1)} \tag{8}$$

2.2.2. Random vector functional link networks

RVFL networks added direct connections from the input to the output layer, as shown in Fig. 2 and described by (9). It randomly assigned weights to the input weights and biases from a uniform distribution and learned the output weights, whose dimension is $M+F$, using a non-iterative algorithm to reduce training complexity [3].

$$\hat{y}(t) = \sum_{i=1}^{M} \beta_i g(w_i^T x(t) + b_i) + \sum_{i=M+1}^{M+F} \beta_i x_{i-M}(t) \tag{9}$$

RVFL, which is based on a functional link ANN architecture results in the matrix shown in (10). The output weights are computed as in the previous subsection. Therefore, the main differences between RVFL and RWNN are adding a bias to each hidden neuron and modeling a linear dependence between the input and output through direct input–output connections.

$$H = \begin{bmatrix} x_1^T & g(w_1^T x_1 + b_1) & \ldots & g(w_M^T x_1 + b_M) \\ \vdots & \vdots & \ddots & \vdots \\ x_N^T & g(w_1^T x_N + b_1) & \ldots & g(w_M^T x_N + b_M) \end{bmatrix}_{N \times (M+F)} \tag{10}$$

These direct connections have been empirically shown to improve the performance of such networks over networks without these connections. For example, Tang et al. [33] showed the significance of the direct input–output connections when applying an ensemble RVFL learning paradigm, which outperformed ELM networks, to the crude oil price prediction problem. Mesquita et al. [34] reached a similar conclusion with regard to the effectiveness of DL when they formulated an ensemble method based on randomized networks including RVFL and ELM with successive projection for regression. A comprehensive analysis of RVFL was performed in [35], discussing parameter sensitivity, activation functions and other effects on 121 UCI classification data sets. They demonstrated RVFL's superior performance compared to ELM and empirically illustrated the importance of DL between the input and output, revealing a superposition of linear and non-linear dependence of the output on the input which is not captured in ELM's network architecture. This was also evident [36] who applied RVFL to electricity load prediction.

2.2.3. Random activation weight networks

Random activation weight network (RAWN), introduced by Te Braake and Van Straten [4], modified RWNN's training algorithm by randomly assigning the activation or input weights of the network in Fig. 2 (without the DL), describing the input–output relationship in (11), from a normal distribution and performing regularization to ensure that H, described by (12), was not singular. The hidden neurons have biases, and the output vector β has dimension M. As a result, this algorithm was shown to be suitable for control applications and dynamic systems.

$$\hat{y} = \sum_{i=1}^{M} \beta_i g\left(w_i^T x + b_i\right) \tag{11}$$

$$H = \begin{bmatrix} g(w_1^T x_1 + b_1) & \ldots & g(w_M^T x_1 + b_M) \\ \vdots & \ddots & \vdots \\ g(w_1^T x_N + b_1) & \ldots & g(w_M^T x_N + b_M) \end{bmatrix}_{N \times M} \tag{12}$$

2.2.4. Extreme Learning machines

Huang et al. [5] applied non-iterative training to the single hidden layer feedforward ANN shown in Fig. 2 (without the red arrows), naming the approach ELM. This single hidden layer architecture relates the input x and output of the network $\hat{y}$ by the function in (11) where $g(.)$ represents the activation function, and input weights w_i and biases b_i are randomly sampled from a uniform distribution, performing regularization to H in (12) when necessary.

The input weights are an MF-dimension vector and the biases an M-dimension vector. Unlike RWNN, ELM adds a bias term to each hidden neuron. However, Refs. [35,36] have shown through extensive experiments with numerous datasets that biases do not contribute much to performance. Compared to RAWN [4], ELM samples the input weights and biases from a uniform distribution as opposed to a normal distribution as in [4]. In addition, this differs from RVFL's formulation which is a function of two summations, a non-linear and a linear weighted sum of the input. These direct input–output connections, omitted in [5], have been empirically shown to improve performance as RVFL has outperformed ELM on various applications in [33–36].

ELM has been generalized to regression problems [37] and was adopted in many applications from modeling ship roll motion [38] to tanker motion dynamics [39]. Online variants [40–43] have also been developed for real-time prediction. ELM has been mainly applied to feedforward ANN, except for Ertugrul [12] which non-iteratively trained a simple Jordan RNN architecture. However, Jordan RNN cannot handle very complex problems. In this work, we

Table 2
Pseudocode for ELM-rand training algorithm.

1. Randomly assign w_i, b_i, α_i
2. Compute H
3. Perform regularization on H to avoid singular matrices
4. Compute β using the generalized Moore–Penrose pseudoinverse

Table 3
Pseudocode for ELM-lin training algorithm.

1. Randomly assign w_i, b_i
2. Compute H with a linearized activation function
3. Perform regularization on H to avoid singular matrices
4. Compute $\theta = [\beta, \alpha]^T$ using the generalized Moore–Penrose pseudoinverse

investigate the effectiveness of training other more complex RNN architectures non-iteratively that can be applied to a wider range of regression problems.

3. Non-iterative proximal randomized recurrent neural networks

In this work, we propose to train RNN using a non-iterative training algorithm. While Ertugrul [12] proposed to apply non-iterative learning to the Jordan RNN architecture, we extend it to other RNN architectures. Based on (1) and the new output matrix elements in (13), it is evident that the problem of learning the weights of the recurrent connections, α, in Elman RNN is not linear. Similarly, we can train a fully connected RNN non-iteratively based on (4) and obtain the output matrix elements in (14), which is also not linear in α. The output matrix elements of a NARMAX architecture are described by (15), and an LSTM architecture is described by (16). As for RVFL networks, a fully connected RNN with DL is described by (17), while a NARMAX and LSTM with DL are described by (18) and (19), respectively.

$$h_{ij}(t) = g\left(w_i^T x_j(t) + b_i + \sum_{k=1}^{Q} \alpha_{ik} f_i(t-k)\right) \tag{13}$$

$$h_{ij}(t) = g\left(w_i^T x_j(t) + b_i + \sum_{k=1}^{Q}\sum_{l=1}^{M} \alpha_{ilk} f_l(t-k)\right) \tag{14}$$

$$h_{ij}(t) = g\left(w_i^T \left[x_j^T(t), y_j^T(t), e_j^T(t)\right]^T + b_i\right) \tag{15}$$

$$h_{ij}(t) = o(t) \circ g_f(c(t)) \tag{16}$$

$$h_{ij}(t) = \begin{cases} g\left(w_i^T x_j + b_i\right) & \text{if } 1 \le i \le M \\ \hat{y_j}(t-i+M) & \text{if } M < i \le M+Q \end{cases} \tag{17}$$

$$h_{ij}(t) = \begin{cases} [x_j^T(t), y_j^T(t), e_j^T(t)]^T & \text{if } 1 \le i \le F \\ g\left(w_i^T \left[x_j^T(t), y_j^T(t), e_j^T(t)\right]^T + b_i\right) & \text{if } F < i \le M+F \end{cases} \tag{18}$$

$$h_{ij}(t) = \begin{cases} x_j^T(t) & \text{if } 1 \le i \le F \\ o(t) \circ g_f(c(t)) & \text{if } F < i \le M+F \end{cases} \tag{19}$$

3.1. Randomized non-iterative RNN training algorithm

Our first attempt to resolve this non-linearity is to randomly assign the values of α, like the input weights and biases. Therefore, the non-iterative training algorithm can be applied as described in [2–5] with the output matrix elements as shown in (13) for the Elman RNN and (14) for the fully connected RNN architecture, in addition to LSTM and networks with DL, without changing the parameter vector θ, i.e. $\theta = \beta$. Hereafter, RNN architectures trained non-iteratively with randomly assigned α values are referred to as ELM-rand. Table 2 summarizes the ELM-rand algorithm for RNN architectures. Input weights, biases and recurrent connections are randomly assigned. The output matrix H corresponding to the network architecture is computed; regularization is performed if necessary. Finally, the output weights are computed using the generalized pseudo-inverse.

3.2. Linear approximation of non-iterative RNN training algorithm

Instead of randomly assigning the recurrent connection weights, these weights can be learned by approximating the activation function with a linear function. More specifically, we consider linearizing the sine and sigmoid activation functions as follows. The sine function is approximated by its Taylor series expansion, shown in (20), and results in the output matrix elements in (21), when using a first order polynomial. The sigmoid function is approximated by a piecewise linear function using the centered linear approximation method to obtain the function in (22) [44]. In this case, we are estimating both β and α, resulting in a parameter vector of $\theta = [\beta, \alpha]^T$. Hereafter, ELM-lin represents RNN architectures trained non-iteratively with a linearized activation function. Table 3 summarizes the ELM-lin algorithm for RNN architectures which randomly assigns the input weights and biases. The output matrix H corresponding to the network architecture is computed based on the linearized activation functions. Finally, the output and recurrent weights are computed using the generalized pseudo-inverse.

$$g(z) = sin(z) = \sum_{j=0}^{\infty} \frac{(-1)^j z^{2j+1}}{(2j+1)!} \approx z \tag{20}$$

$$h_{ij}(t) = w_i x_j(t) + b_i + \sum_{k=1}^{Q}\sum_{l=1}^{M} \alpha_{ilk} f_l(t-k) \tag{21}$$

$$g(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{2}\left(1 + \frac{z}{2}\right) \tag{22}$$

4. Theoretical analysis

In this section, we perform a theoretical comparison of the various non-iteratively-trained RNN architectures.

4.1. Computational complexity

The computational complexity of training a network and predicting an output given an input vector depends on the network architecture and training algorithm. Therefore, to analyze the computational complexity of various RNN architectures when trained non-iteratively, we first derive the complexity of the non-iterative training algorithm for a generic feedforward single hidden layer network then consider the output matrix dimensions of the different RNNs.

The non-iterative algorithm consists of three main steps: (1) randomly assigning values to the input weights and biases, (2) computing the output matrix H and (3) solving for the parameter vector θ which contains the output weights β, using the Moore–Penrose pseudoinverse. We consider a network with F input neurons, M hidden neurons and 1 output neuron trained on N input vectors. Assigning random values requires a constant amount of time and the memory requirements are linear in the number of assigned values, equal to the sum of input layer weights and biases.

Table 4
Computational complexity of ELM.

	Time complexity	Space complexity
Randomly assign weights and biases	$O(1)$	$O(FM+M)$
Compute H	$O(NM(F+c))$	$O(NM)$
Compute θ	$O(NM)$	$O(M)$
Training total	$O(NM(F+c)+NM)$	$O(FM+2M+NM)$

Table 5
Number of weights per layer for ANN architectures.

Network	DL	Input weights: w	Biases: b	Recurrent weights: α	Output weights: β
Feedforward	N	FM	M	0	M
Feedforward	Y	FM	M	0	$M+F$
Elman	N	FM	M	MQ	M
Elman	Y	FM	M	MQ	$M+F$
Jordan	N	$(F+Q)M$	M	0	M
Jordan	Y	$(F+Q)M$	M	0	$M+F$
Fully connected	N	FM	M	QM^2	M
Fully connected	Y	FM	M	QM^2	$M+F$
NARMAX	N	$(F+Q+R)M$	M	0	M
NARMAX	Y	$(F+Q+R)M$	M	0	$M+F$
LSTM	N	$4MF$	$4M$	$4M$	M
LSTM	Y	$4MF$	$4M$	$4M$	$M+F$

The output matrix H is an $N\times M$ matrix and consists of multiplying F-dimensional input vectors by F-dimensional input weights and applying an activation function with a constant number of floating-point computations c. Therefore, computing the H matrix requires a total of $NM(F+c)$ computations. Finally, calculating the pseudoinverse of H and multiplying the result by the target output requires $O(NM)$ operations when $M\leq N$ and $O(NM+N^2M)$ otherwise. Since RNN are trained on a large number of data points compared to the number of hidden neurons, computing θ requires $O(NM)$ operations. On the other hand, testing requires computing the H matrix and multiplying it by β, equivalent $O(M(F+c)+M)$ computations for one test point, to find $\hat{y}$. Table 4 summarizes the time and space complexity of the non-iterative training algorithm. The space complexity indicates how many floating-point numbers are saved in memory.

Focusing on the RNN architectures discussed in Section 3, the number of input layer weights, biases, hidden layer weights and output layer weights will vary based on the architecture under consideration. Table 5 summarizes the number of weights for each layer, based on the architecture. When training Elman, fully connected and LSTM RNN using ELM-rand, the hidden recurrent weights will be randomly assigned and will not affect the size of the H matrix or the number of parameters to estimate. However, for Elman and fully connected RNN trained using ELM-lin, the parameter vector becomes $\theta=[\beta,\alpha]^T$, and the size of the H matrix is also affected. The number of input weights is based on the dimensionality of the feature vector (F) and the number of hidden layer neurons (M). Jordan and NARMAX networks increase the dimensionality of the input vector by feeding back the predicted output which results in a higher number of input weights, as shown in Table 5. The LSTM cell contains four gates that contain input weights, recurrent connections and biases, as shown in Fig. 1, leading to $4MF$ input weights per cell. The number of biases is equal to the number of hidden layer neurons (M), except in the LSTM network where the LSTM cell contains four bias weights. The number of recurrent connections in the hidden layer depends on the number of hidden layers and number of time steps fed back into the neuron. For example, it is possible to feedback only the previous value of the neuron output, which means $Q=1$. The number of output weights depends on the number of direct connections from the hidden and input layers to the output neurons. Input to output layer connections increase the dimensionality of the parameter vector by the dimensionality of the input vector, i.e. the dimensionality of θ becomes $M+F$.

Finally, the computational complexity of training a network involves combining the number of weights in each architecture to the number of computations per weight for a given training algorithm. For example, a fully connected RNN has M output weights, MQ recurrent weights, NM input weights and M bias weights. Therefore, the ELM-rand algorithm requires a total of $O(NM(F+Q^2+c)+NM)$ computations and $O(FM+M+MQ+NM+M)$ of space on this network. On the other hand, training an Elman RNN using ELM-lin would require $O(N(MQ+M))$ computations to find β, $O(NM(F+c))$ computations to find H and $O(FM+M+NM+M+MQ)$ of memory.

4.2. Model transformations

In this section, we investigate the relationship between the various RNN architectures. Connoret al. [9] proved that a non-linear autoregressive moving average(NARMA) network is a special case of the fully connected RNN architecture; extending it to NARMAX network simply involves adding the exogenous input to both input layers. Moreover, α_{ikl} in (4) is set to $-\alpha_{ik}\beta_i$ and w_{ij} in (4) is set to $w_{ij}+\alpha_{ij}, j\leq Q$ or $w_{ij}, j\leq F$ for $F>Q$. Jordan RNN are also a special case of fully connected RNN and can be created by setting α_{ikl} in (4) to $\alpha_{ik}\beta_i$ from (2). Fully connected RNN can be converted to Elman RNN by setting $\alpha_{ikl}=0$ if $k\neq 1$ and $i\neq l$. Finally, Elman and Jordan RNN are equivalent when α_{ik} in (1) is set to $\alpha_{ik}\beta_j$ for $i=j$ where $j=[1,M]$. The model equivalences could help to improve the performance of non-iteratively-trained RNN. For example, one could train a NARMAX network, then convert it to a fully connected RNN based on the weight relationships mentioned above. This would result in faster training. Similarly, this methodology could be applied to other architectures as well. However, the empirical performance of these theoretical transformations on real-world problems should be investigated to confirm their merit.

5. Experimental results

In this section, we present the simulation results of applying non-iteratively-trained RNN on multiple benchmarks compared to other ANN algorithms in the literature, based on prediction accuracy and training time. Before presenting the results, a

Table 6
Description of the time series benchmarks.

Database	Number of training instances	Number of testing instances	Exogenous input dimension	Output mean	Output standard deviation	Output min	Output max
Fortaleza	104	45	0	1445.0	496.95	468.0	2836.0
Consumo	108	46	0	120.90	27.24	75.39	232.0
Bebida	130	57	0	92.66	20.35	75.39	232.0
Ipi	130	57	0	103.60	18.28	65.81	148.3
Atmosfera Temperature	255	110	0	15.35	3.16	5.6	21.6
Atmosfera Humidity	255	110	0	81.15	7.95	53.34	95.79
Lavras	268	116	0	127.60	122.06	0	718
Santa Fe Laser	700	300	0	59.9	46.9	2	255
Quebec Births	1000	4113	0	250.8	41.9	−23.08	366.0
Deep Tesla	1050	450	240	0.8523	3.2355	−6	6
Japan population	1778	762	0	0.1399×10^7	0.1384×10^7	0.0123×10^7	1.3514×10^7
SP500	12,053	5165	1	0.0899×10^{10}	0.1533×10^{10}	0.0001×10^{10}	1.1456×10^{10}
AEMO	12,264	5256	0	7981.57	1190.91	5113.03	13787.85
Electricity Load Balance	140,257	140,257	1	2.7×10^{14}	2.6×10^{14}	0	9.9×10^{14}

brief description of the benchmarks and experimental setup is presented.

5.1. Experimental setup

The experiments were run in Matlab R2016b on an Intel Xeon 64-bit processor with 12 cores, 2.0 GHz clock speed and 24 GB of RAM. Matlab's neural networks toolbox was used to train NARMAX-BPTT while the statistics and machine learning toolbox was used to train support vector regression (SVR). The SVR hyperparameters were obtained using Matlab's built-in automatic hyperparameter optimization which search for the optimal kernel, box constraint, epsilon value and other hyperparameters using Bayesian optimization. The ELM code was downloaded from Zhu and Huang [45], and all other algorithms were written in Maltab R2016b. Hereafter, ELM is referred to as feedforward network without DL trained using ELM-rand, and RVFL is referred to as feedforward network with DL trained using ELM-rand or simply feedforward networks when no confusion is possible.

Multiple error measures were used to compare the performance of the various algorithms including mean square error (MSE), $E[(y-\hat{y})^2]$, root MSE (RMSE), $\sqrt{E[(y-\hat{y})^2]}$, normalized RMSE (nRMSE), $\frac{1}{y_{max}-y_{min}}\sqrt{E[(y-\hat{y})^2]}$, mean absolute error (MAE), $E[|y-\hat{y}|]$, and normalized MAE (nMAE), $\frac{1}{y_{max}-y_{min}}\sqrt{E[|y-\hat{y}|]}$. Furthermore, the training time and error measures of each algorithm were averaged over 5 independent runs to assess repeatability since the considered algorithms possess random characteristics. A repeatability analysis allows us to determine how sensitive the algorithms' performance is to the random number generator.

5.1.1. Time series prediction

The proposed methodology was validated on time series prediction problems. Table 6 summarizes the characteristics of the publicly available databases, sorted in ascending order of total database size. The Sante Fe Laser data is stationary [46], while Quebec Births is non-stationary [47]. The data in Sante Fe Laser was normalized to improve the condition number of the input matrix and avoid matrix inversion singularities. For the electricity load balance data set [48], the MT166 substation of the electricity load balance data set was selected for training and MT257 for testing, and the input vector consists of the day of the measurement after normalizing the data. AEMO data [49] reports the electricity load demand in Australia; the first 70% of the data was used for training and the remaining 30% for testing. The same was done for the Atmosfera Temperature, Atmosfera Humidity, Bebida, Consumo, Fortaleza, Ipi, and Lavras, obtained from Morettin and Toloi [50]. Deep tesla is an autonomous driving database whose purpose is to predict the steering angle of the steering wheel given an image of the road ahead as seen from the front window shield of a car [51]. The first 1500 frames of the database are used in our experiments since they correspond to the a single driving simulation. Japan population tracks the population of various Japanese regions [52]. Finally, the SP 500 database records the stock prices since 1950 [53].

5.1.2. POMDP-modeled decision-making

The inverted double pendulum on a sliding cart benchmark, also referred to as the double pole balancing problem, is a continuous sequential decision-making problem. The benchmark was run using the Simulink model of an inverted double pendulum on a sliding cart in Matlab R2015b, on an Intel Core i7, 64-bit machine with 2.4 GHz clock speed and 12 GB of RAM. The networks were trained on 1200 data points, then they were tested on 300 data points. Usually, the feature vector contained the cart's position and velocity in addition to the poles' angles and their velocities. The force applied to the cart was predicted. Excluding the cart and poles' velocities makes the problem partially observable and can be modeled using POMDP, which was considered in this work. The data was collected by saving the input and output values of a PID controller designed to control the plant. The controller was run for 10 seconds sampled at approximately 6.67 ms. Performance was measured by computing the MSE of the predicted force that should be applied to the cart and the actual force that was applied by a PID controller.

5.2. Non-iteratively-trained RNN performance analysis

A grid search was performed over the number of hidden neurons and types of activation functions (sigmoid and sine) to find the best architecture. Fig. 3 plots the predicted output values for ELM-lin trained fully connected RNN on the AEMO database, which shows that the algorithms could learn the trend in the data relatively well.Tables 7–17 summarize average (avg) and standard deviation (std) of the MSE, RMSE, MAE and training time of the various RNN architectures for both ELM-rand and ELM-lin on some of the databases, when sampling the random weights from a uniform distribution. Results of BPTT-trained RNN and SVR are also included. The hyperparameter column reports the number of hidden neurons for ANN algorithms and (kernel,box constraint) for SVR.

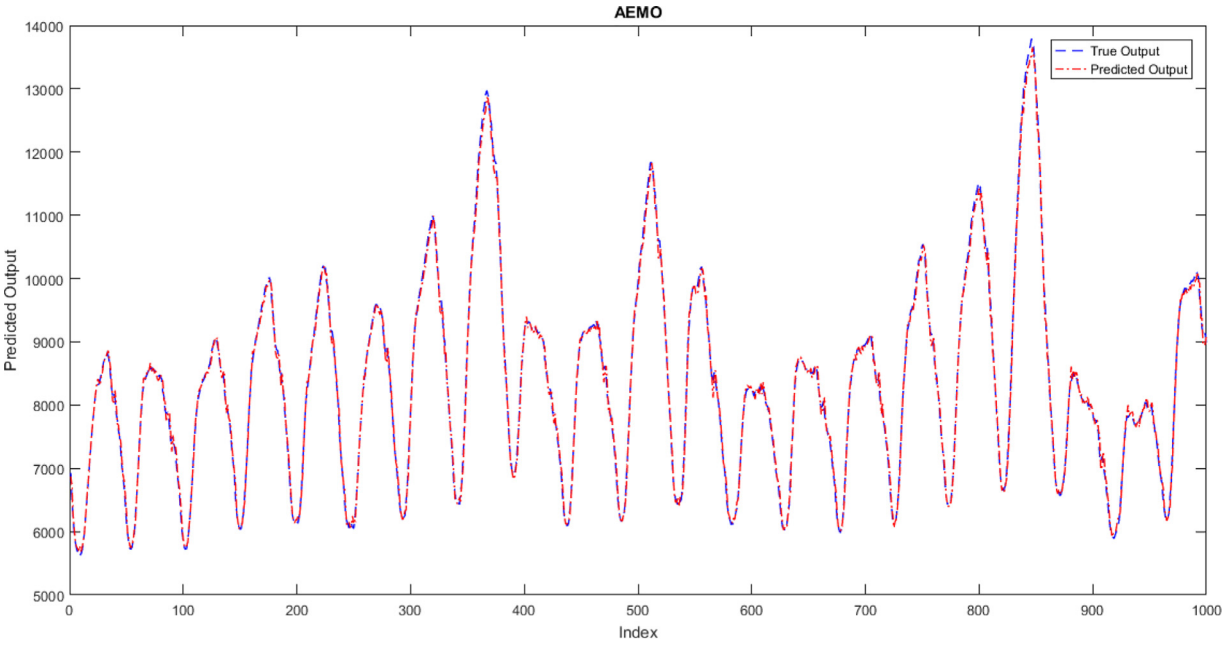

Fig. 3. Predicted output for AEMO of a fully connected RNN (without DL) trained using ELM-lin.

Sampling the random weights from a normal distribution was tested and the results showed either almost identical MSE to uniform distribution or significantly worse values. We also compared the RNN variants derived from the feedforward network in Refs. [2] and [5], where the difference was the bias terms. The results showed that the latter performed as well as the former on all RNN variants and databases, which is consistent with Ren et al. [36] who concluded that the bias term was irrelevant.

ELM-lin generally achieved lower MSE but required more time to train than ELM-rand. For example, for the Santa Fe database, ELM-lin had a lower MSE than ELM-rand on almost all RNN architectures. Fully connected RNN with DL was an exception. A similar pattern was observed for other databases as well with some exceptions.

Furthermore, non-iteratively-trained fully connected RNN performed better than non-iteratively-trained Elman RNN but was also more computationally expensive. For example, the inverted pendulum problem experienced a 56% improvement in MSE with the Elman RNN with ELM-lin and fully connected RNN with ELM-lin but required 87% more time to train. Comparing non-iteratively-trained Elman and fully connected RNN on the Santa Fe laser data set, non-iteratively-trained fully connected RNN achieved 77% improvement in MSE but resulted in a slight increase in MSE, 1.6%, for the Electricity load balance data set. The training time varied based on the database size but generally reflected an increase in training time for non-iteratively-trained fully connected RNN compared to non-iteratively-trained Elman RNN. Non-iteratively-trained LSTM did not perform well in general on the smaller databases due to the curse of dimensionality. Larger databases are required to learn a better model. Considering the electricity load database, a 3.5 $\times$ increase in MSE was experienced compared to non-iteratively-trained feedforward networks. Therefore, additional research should be performed to assess the merit of training an LSTM network non-iteratively.

Next, we compared the performance of non-iteratively-trained Elman and fully connected RNN to the Jordan RNN architecture proposed in [12] and non-iteratively-trained feedforward networks based on Refs. [2,3,5]. Note that the difference between Refs. [2] and [5] is the presence of a bias neuron in the hidden layer as opposed to the input layer. Considering the inverted pendulum problem, the feedforward network trained using ELM had the best performance with an MSE of 0.0168 only 4.8% better than the best non-iteratively-trained RNN algorithm which was fully connected RNN trained with ELM-rand. However, the disadvantage of using feedforward networks is that the best autoregressive model with exogenous input, i.e. the order of the autoregressive output, must be obtained by trial and error. Non-iteratively-trained RNN does not require this grid search. In addition, learning the additional weights did not cause a significant reduction in training speed.

Focusing on architectures with and without DL, a consistent pattern was not observed, i.e. RNN architectures with DL sometimes performed better than their counterparts without DL and vice versa. For example, a fully connected RNN trained using ELM-rand performed better without DL on the Atmosphere Temperature database but much worse on the Consumo database. However, ELM-rand trained LSTM networks with DL consistently outperformed LSTM without DL on all the mentioned databases. Therefore, adding DL to non-iteratively-trained RNN architectures should be investigated on a case by case basis.

Compared to NARX network trained using BPTT, a speedup of 69% in training time of non-iteratively-trained RNN was achieved with only a 4.3% degradation in prediction accuracy. NARMAX with ELM achieved faster training time than NARX with BPTT but resulted in larger MSE. In addition, non-iteratively-trained RNN achieved better repeatability given their lower standard deviations compared to BPTT-trained networks. This is due to the lower dimensional search spaces of non-iterative training algorithms compared to iterative algorithms, since the input weights are not learned. Similar results were obtained on the larger data sets where significant speedups, up to 99% for Electricity load balance, were achieved but at the cost of an increase in MSE.

Compared to SVR, non-iteratively-trained RNN are approximately 7.7 $\times$ faster than SVR, when averaging across all databases; the speed up was more significant as the size of the database increased and reached almost 37 $\times$ on the Electricity load database.

Table 7
Performance on atmosphere humidity data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training Time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	45	3.12E-3 ± 6.98E-3	4.95E+1 ± 4.03E-12	7.04E+0 ± 2.86E-13	5.86E+0 ± 2.54E-13
Elman	N	ELM-rand	10	3.1E-1 ± 9.0E-2	5.01E+1 ± 2.3E+0	7.08E+0 ± 1.62E-1	5.85E+0 ± 8.62E-2
Elman	N	BPTT	10	2.18E-1 ± 2.70E-2	7.61E+1 ± 1.86E+1	8.67E+0 ± 1.06E+0	7.21E+0 ± 7.90E-1
Elman	Y	ELM-lin	25	1.88E-1 ± 6.3E-2	4.95E+1 ± 4.00E-12	7.04E+0 ± 2.84E-13	5.86E+0 ± 2.49E-13
Elman	Y	ELM-rand	10	3.9E-1 ± 7.8E-2	5.12E+1 ± 6.87E+0	7.14E+0 ± 4.62E-1	5.82E+0 ± 1.55E-1
Fully connected	N	ELM-lin	45	1.50E-1 ± 8.54E-3	4.95E+1 ± 6.93E-12	7.04E+0 ± 4.92E-13	6.0E+0 ± 3.5E-1
Fully connected	N	ELM-rand	10	3.12E-3 ± 6.98E-3	5.2E+1 ± 5.6E+0	7.2E+0 ± 3.7E-1	3.70E+1 ± 6.90E+1
Fully connected	Y	ELM-lin	45	1.50E-1 ± 8.54E-3	4.95E+1 ± 7.23E-12	7.04E+0 ± 5.13E-13	5.86E+0 ± 4.37E-13
Fully connected	Y	ELM-rand	10	2.6E-1 ± 7.0E-2	5.34E+1 ± 1.17E+1	7.28E+0 ± 7.55E-1	5.84E+0 ± 1.87E-1
Jordan	N	ELM-rand	20	9.36E-3 ± 8.54E-3	6.83E+1 ± 1.41E+1	8.23E+0 ± 8.20E-1	6.61E+0 ± 4.68E-1
Jordan	Y	ELM-rand	50	9.36E-3 ± 8.54E-3	6.35E+1 ± 4.40E-2	7.97E+0 ± 2.76E-3	6.39E+0 ± 1.94E-3
NARMAX	N	ELM-rand	15	9.36E-3 ± 8.54E-3	5.83E+1 ± 1.06E+1	7.61E+0 ± 6.64E-1	6.31E+0 ± 5.83E-1
NARMAX	Y	ELM-rand	50	2.34E-1 ± 9.06E-2	1.54E+2 ± 1.37E-1	1.24E+1 ± 5.54E-3	1.05E+1 ± 5.48E-3
NARX	N	BPTT	15	1.99E-1 ± 1.30E-2	9.71E+1 ± 2.30E+1	9.79E+0 ± 1.21E+0	7.73E+0 ± 8.20E-1
Feedforward [5]	N	ELM-rand	20	3.9E-1 ± 7.8E-2	6.18E+1 ± 2.87E+0	7.86E+0 ± 1.82E-1	6.6E+0 ± 1.85E-1
Feedforward [2]	N	ELM-rand	15	6.25E-3 ± 1.40E-2	6.28E+1 ± 3.99E+0	7.92E+0 ± 2.55E-1	6.66E+0 ± 2.16E-1
Feedforward [3]	Y	ELM-rand	10	3.6E-1 ± 1.1E-1	**4.75E+1 ± 6.69E-1**	**6.89E+0 ± 4.8E-2**	**5.73E+0 ± 4.4E-2**
SVR	–	–	Linear, 593	7.71E-2 ± 0.0E+0	4.99E+1 ± 0.0E+0	7.07E+0 ± 0.0E+0	5.83E+0 ± 0.0E+0

Table 8
Performance on atmosphere temperature data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training Time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	35	3.1E-1 ± 5.9E-2	2.48E+0 ± 3.50E-14	1.58E+0 ± 1.11E-14	1.27E+0 ± 1.28E-14
Elman	N	ELM-rand	10	1.56E-2 ± 2.70E-2	9.17E+0 ± 8.15E+0	2.81E+0 ± 1.26E+0	1.42E+0 ± 1.57E-1
Elman	N	BPTT	20	2.65E-1 ± 7.26E-2	1.18E+1 ± 6.93E+0	3.3E+0 ± 1.06E+0	2.63E+0 ± 9.75E-1
Elman	Y	ELM-lin	10	6.24E-3 ± 8.54E-3	2.48E+0 ± 4.74E-14	1.58E+0 ± 1.51E-14	1.27E+0 ± 1.82E-14
Elman	Y	ELM-rand	10	3.2E-1 ± 0.0E+0	5.05E+0 ± 2.03E+0	2.21E+0 ± 4.63E-1	1.39E+0 ± 8.28E-2
Fully connected	N	ELM-lin	25	5.62E-2 ± 8.54E-3	2.48E+0 ± 1.32E-13	1.58E+0 ± 4.20E-14	1.27E+0 ± 4.75E-14
Fully connected	N	ELM-rand	10	3.12E-3 ± 6.98E-3	3.47E+0 ± 2.49E+0	1.79E+0 ± 5.73E-1	1.27E+0 ± 1.20E-1
Fully connected	Y	ELM-lin	45	2.15E-1 ± 2.56E-2	2.48E+0 ± 1.35E-13	1.58E+0 ± 4.30E-14	1.27E+0 ± 6.41E-14
Fully connected	Y	ELM-rand	10	2.7E-1 ± 6.8E-2	6.95E+1 ± 1.49E+2	5.05E+0 ± 7.42E+0	1.63E+0 ± 7.49E-1
Jordan	N	ELM-rand	40	9.36E-3 ± 8.54E-3	1.69E+1 ± 3.32E+0	4.1E+0 ± 4.11E-1	3.58E+0 ± 3.77E-1
Jordan	Y	ELM-rand	10	6.24E-3 ± 8.54E-3	4.37E+0 ± 8.45E-2	2.09E+0 ± 2.02E-2	1.78E+0 ± 2.18E-2
LSTM	N	ELM-rand	10	3.74E-2 ± 8.54E-3	1.27E+2 ± 1.29E+2	9.9E+0 ± 6.04E+0	9.53E+0 ± 6.23E+0
LSTM	Y	ELM-rand	25	1.66E-1 ± 1.59E-2	4.31E+0 ± 7.33E-2	2.08E+0 ± 1.77E-2	1.77E+0 ± 1.83E-2
NARMAX	N	ELM-rand	15	1.25E-2 ± 6.98E-3	3.28E+1 ± 6.19E+0	5.71E+0 ± 5.43E-1	5.23E+0 ± 5.82E-1
NARMAX	Y	ELM-rand	10	1.78E-1 ± 1.25E-2	8.19E+0 ± 2.66E-2	2.86E+0 ± 4.64E-3	2.54E+0 ± 4.31E-3
NARX	N	BPTT	10	1.05E+0 ± 1.74E+0	6.75E+0 ± 9.66E-1	2.59E+0 ± 1.83E-1	1.97E+0 ± 6.01E-2
Feedforward [5]	N	ELM-rand	50	3.0E-1 ± 1.2E-1	5.05E+0 ± 1.27E+0	2.23E+0 ± 2.76E-1	1.74E+0 ± 1.61E-1
Feedforward [2]	N	ELM-rand	50	6.25E-3 ± 1.40E-2	5.01E+0 ± 8.81E-1	2.23E+0 ± 1.99E-1	1.73E+0 ± 1.30E-1
Feedforward [3]	Y	ELM-rand	10	3.1E-1 ± 1.1E-1	2.74E+0 ± 1.87E-3	1.66E+0 ± 5.64E-4	1.31E+0 ± 1.79E-3
SVR	–	–	Linear, 0.26927	1.65E+0 ± 0.0E+0	**2.15E+0 ± 0.0E+0**	**1.47E+0 ± 0.0E+0**	**1.19E+0 ± 0.0E+0**

Table 9
Performance on Bebida data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	30	3.12E-3 ± 6.98E-3	1.93E+2 ± 2.11E-12	1.39E+1 ± 7.60E-14	1.04E+1 ± 9.87E-14
Elman	N	ELM-rand	20	2.6E-1 ± 7.4E-2	**1.71E+2 ± 1.06E+1**	**1.31E+1 ± 4.05E-1**	**9.6E+0 ± 2.71E-1**
Elman	N	BPTT	10	8.94E-1 ± 1.34E+0	6.71E+2 ± 5.46E+2	2.46E+1 ± 9.04E+0	2.02E+1 ± 8.52E+0
Elman	Y	ELM-lin	10	2.8E-1 ± 6.3E-2	1.93E+2 ± 3.27E-12	1.39E+1 ± 1.18E-13	1.04E+1 ± 1.72E-13
Elman	Y	ELM-rand	20	3.20E-1 ± 0.0E+0	1.85E+2 ± 6.11E+0	1.36E+1 ± 2.24E-1	9.91E+0 ± 1.49E-1
Fully connected	N	ELM-lin	45	7.49E-2 ± 6.98E-3	1.93E+2 ± 7.21E-12	1.39E+1 ± 2.60E-13	1.04E+1 ± 1.40E-13
Fully connected	N	ELM-rand	10	3.20E-1 ± 0.0E+0	2.57E+2 ± 1.36E+2	1.57E+1 ± 3.79E+0	1.26E+1 ± 3.88E+0
Fully connected	Y	ELM-lin	35	4.37E-2 ± 1.31E-2	1.93E+2 ± 1.29E-11	1.39E+1 ± 4.65E-13	1.04E+1 ± 1.10E-13
Fully connected	Y	ELM-rand	10	2.9E-1 ± 5.8E-2	2.01E+2 ± 1.24E+0	1.42E+1 ± 4.38E-2	1.03E+1 ± 2.46E-2
Jordan	N	ELM-rand	45	6.24E-3 ± 8.54E-3	2.29E+2 ± 4.47E+1	1.51E+1 ± 1.46E+0	1.17E+1 ± 7.57E-1
Jordan	Y	ELM-rand	45	6.24E-3 ± 8.54E-3	2.24E+2 ± 1.63E-1	1.50E+1 ± 5.46E-3	1.06E+1 ± 4.96E-3
LSTM	N	ELM-rand	10	1.56E-2 ± 1.10E-2	3.46E+3 ± 6.25E+3	4.34E+1 ± 4.44E+1	4.09E+1 ± 4.42E+1
LSTM	Y	ELM-rand	45	1.69E-1 ± 1.53E-2	2.25E+2 ± 9.35E-2	1.50E+1 ± 3.12E-3	1.06E+1 ± 1.94E-3
NARMAX	N	ELM-rand	25	6.24E-3 ± 8.54E-3	2.35E+2 ± 1.13E+2	1.50E+1 ± 3.62E+0	1.24E+1 ± 2.67E+0
NARMAX	Y	ELM-rand	45	1.13E-1 ± 9.19E-2	2.37E+2 ± 1.98E-1	1.54E+1 ± 6.44E-3	1.20E+1 ± 5.45E-3
NARX	N	BPTT	15	1.22E+0 ± 2.26E+0	4.27E+2 ± 9.03E+1	2.06E+1 ± 2.08E+0	1.67E+1 ± 1.85E+0
Feedforward [5]	N	ELM-rand	10	2.8E-1 ± 6.3E-2	8.29E+2 ± 8.36E+1	2.88E+1 ± 1.40E+0	2.57E+1 ± 2.25E-1
Feedforward [2]	N	ELM-rand	20	3.0E-1 ± 8.4E-2	9.02E+2 ± 1.03E+2	3.00E+1 ± 1.70E+0	2.67E+1 ± 1.04E+0
Feedforward [3]	Y	ELM-rand	15	3.6E-1 ± 7.4E-2	1.78E+2 ± 3.51E+0	1.33E+1 ± 1.31E-1	9.67E+0 ± 1.03E-1
SVR	–	–	Linear, 1.0514	5.33E-2 ± 0.0E+0	2.03E+2 ± 0.0E+0	1.42E+1 ± 0.0E+0	1.11E+1 ± 0.0E+0

Table 10
Performance on consumo data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	10	3.12E-3 ± 6.98E-3	2.0E+2 ± 3.4E-11	1.4E+1 ± 1.2E-12	1.05E+1 ± 1.52E-12
Elman	N	ELM-rand	20	2.7E-1 ± 1.3E-1	2.47E+2 ± 7.43E+1	1.55E+1 ± 2.29E+0	1.24E+1 ± 2.33E+0
Elman	N	BPTT	10	2.37E-1 ± 1.27E-1	1.37E+3 ± 9.76E+2	3.49E+1 ± 1.38E+1	2.85E+1 ± 1.06E+1
Elman	Y	ELM-lin	20	1.8E-1 ± 5.2E-2	2.01E+2 ± 4.26E-11	1.42E+1 ± 1.50E-12	1.05E+1 ± 1.90E-12
Elman	Y	ELM-rand	30	2.3E-1 ± 7.8E-2	2.01E+2 ± 2.87E+0	1.42E+1 ± 1.01E-1	1.03E+1 ± 5.83E-2
Fully connected	N	ELM-lin	10	3.12E-3 ± 6.98E-3	2.0E+2 ± 2.3E-11	1.4E+1 ± 8.1E-13	1.05E+1 ± 1.14E-12
Fully connected	N	ELM-rand	10	1.6E-1 ± 0.0E+0	4.74E+2 ± 1.33E+2	2.15E+1 ± 3.36E+0	1.49E+1 ± 2.44E+0
Fully connected	Y	ELM-lin	40	4.37E-2 ± 6.98E-3	2.0E+2 ± 2.8E-10	1.4E+1 ± 9.8E-12	1.05E+1 ± 1.22E-11
Fully connected	Y	ELM-rand	15	1.9E-1 ± 5.1E-2	**1.99E+2 ± 2.83E+0**	**1.41E+1 ± 1.0E-1**	**1.03E+1 ± 5.9E-2**
Jordan	N	ELM-rand	45	3.1E-1 ± 5.1E-2	3.74E+2 ± 6.44E+1	1.93E+1 ± 1.69E+0	1.33E+1 ± 1.35E+0
Jordan	Y	ELM-rand	50	1.25E-2 ± 6.98E-3	5.12E+2 ± 2.19E-1	2.26E+1 ± 4.84E-3	1.71E+1 ± 3.97E-3
LSTM	N	ELM-rand	30	1.87E-2 ± 6.98E-3	5.14E+3 ± 4.21E+3	6.62E+1 ± 3.07E+1	6.43E+1 ± 3.15E+1
LSTM	Y	ELM-rand	35	1.06E-1 ± 8.75E-2	5.13E+2 ± 1.42E-1	2.27E+1 ± 3.12E-3	1.71E+1 ± 2.42E-3
NARMAX	N	ELM-rand	10	3.12E-3 ± 6.98E-3	2.74E+2 ± 5.66E+1	1.65E+1 ± 1.64E+0	1.32E+1 ± 1.57E+0
NARMAX	Y	ELM-rand	50	2.59E-1 ± 8.82E-2	3.63E+2 ± 1.11E-1	1.90E+1 ± 2.92E-3	1.62E+1 ± 2.98E-3
NARX	N	BPTT	10	9.64E-1 ± 1.77E+0	9.46E+2 ± 7.36E+2	2.88E+1 ± 1.20E+1	1.95E+1 ± 6.12E+0
Feedforward [5]	N	ELM-rand	20	2.7E-1 ± 6.7E-2	7.98E+2 ± 3.13E+1	2.82E+1 ± 5.53E-1	2.54E+1 ± 4.81E-1
Feedforward [2]	N	ELM-rand	30	2.6E-1 ± 7.0E-2	8.57E+2 ± 1.09E+2	2.92E+1 ± 1.80E+0	2.57E+01 ± 1.31E+0
Feedforward [3]	Y	ELM-rand	10	3.2E-1 ± 1.5E-1	7.56E+2 ± 6.32E+2	2.55E+1 ± 1.14E+1	1.43E+1 ± 3.02E+0
SVR	–	–	Linear, 2.1304	1.62E+0 ± 0.0E+0	2.01E+2 ± 0.0E+0	1.42E+1 ± 0.0E+0	8.84E+0 ± 0.0E+0

Table 11
Performance on IPI data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	20	3.0E-1 ± 4.9E-2	1.21E+2 ± 2.93E-11	1.10E+1 ± 1.33E-12	**8.9E+0 ± 1.0E-12**
Elman	N	ELM-rand	20	2.9E-1 ± 5.4E-2	1.76E+2 ± 4.38E+1	1.32E+1 ± 1.75E+0	1.11E+1 ± 1.66E+0
Elman	N	BPTT	10	3.06E-1 ± 1.16E-1	5.00E+2 ± 3.92E+2	2.10E+1 ± 8.44E+0	1.81E+1 ± 7.79E+0
Elman	Y	ELM-lin	20	3.1E-1 ± 0.0E+0	1.21E+2 ± 2.18E-11	1.10E+1 ± 9.90E-13	**8.9E+0 ± 7.6E-13**
Elman	Y	ELM-rand	10	2.6E-1 ± 7.3E-2	**1.14E+2 ± 2.49E+0**	**1.1E+1 ± 1.16E-1**	9.15E+0 ± 1.79E-1
Fully connected	N	ELM-lin	45	8.42E-2 ± 8.54E-3	1.21E+2 ± 1.88E-10	1.10E+1 ± 8.53E-12	8.92E+0 ± 6.43E-12
Fully connected	N	ELM-rand	10	3.2E+0 ± 0.0E+0	5.49E+2 ± 8.10E+1	2.34E+1 ± 1.73E+0	1.88E+1 ± 1.57E+0
Fully connected	Y	ELM-lin	50	1.34E-1 ± 1.40E-2	1.21E+2 ± 2.17E-10	1.10E+1 ± 9.88E-12	**8.9E+0 ± 8.0E-12**
Fully connected	Y	ELM-rand	10	2.7E-1 ± 6.8E-2	1.14E+2 ± 1.94E+0	1.07E+1 ± 9.08E-2	9.12E+0 ± 1.34E-1
Jordan	N	ELM-rand	45	3.12E-3 ± 6.98E-3	2.37E+2 ± 5.81E+1	1.53E+1 ± 1.92E+0	1.24E+1 ± 1.55E+0
Jordan	Y	ELM-rand	50	3.0E-1 ± 5.4E-2	1.57E+2 ± 1.39E-1	1.25E+1 ± 5.55E-3	1.02E+1 ± 4.30E-3
LSTM	N	ELM-rand	15	1.56E-2 ± 1.10E-2	6.29E+3 ± 6.87E+3	6.81E+1 ± 4.54E+1	6.60E+1 ± 4.62E+1
LSTM	Y	ELM-rand	25	2.81E-2 ± 5.63E-2	1.57E+2 ± 5.15E-2	1.25E+1 ± 2.06E-3	1.02E+1 ± 1.44E-3
NARMAX	N	ELM-rand	45	9.36E-3 ± 8.54E-3	5.17E+2 ± 2.13E+2	2.24E+1 ± 4.36E+0	1.90E+1 ± 3.5E+0
NARMAX	Y	ELM-rand	50	1.91E-1 ± 4.57E-2	6.76E+2 ± 1.53E-1	2.60E+1 ± 2.95E-3	2.05E+1 ± 2.21E-3
NARX	N	BPTT	10	1.02E+0 ± 1.8E+0	7.69E+2 ± 5.10E+2	2.65E+1 ± 9.29E+0	2.22E+1 ± 8.15E+0
Feedforward [5]	N	ELM-rand	45	3.0E-1 ± 1.1E-1	4.75E+2 ± 4.96E+1	2.18E+1 ± 1.11E+0	2.03E+1 ± 9.53E-1
Feedforward [2]	N	ELM-rand	40	6.25E-03 ± 1.40E-2	4.58E+2 ± 3.32E+1	2.14E+1 ± 7.70E-1	1.96E+1 ± 5.60E-1
Feedforward [3]	Y	ELM-rand	10	3.12E-3 ± 6.98E-3	1.34E+2 ± 2.85E+1	1.15E+1 ± 1.18E+0	9.76E+0 ± 9.99E-1
SVR	–	–	Linear, 897.3	1.6E+0 ± 0.0E+0	2.90E+2 ± 0.0E+0	1.70E+1 ± 0.0E+0	1.58E+1 ± 0.0E+0

Table 12
Performance on inverted pendulum data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	35	6.24E-3 ± 8.54E-3	1.30E-1 ± 4.34E-16	3.61E-1 ± 6.05E-16	2.60E-1 ± 3.02E-16
Elman	N	ELM-rand	10	9.36E-3 ± 2.09E-2	1.08E-1 ± 1.00E-2	3.28E-1 ± 1.52E-2	2.32E-1 ± 5.34E-3
Elman	N	BPTT	10	5.00E-1 ± 1.30E-1	1.41E-1 ± 2.33E-2	3.74E-1 ± 3.00E-2	2.73E-1 ± 2.44E-2
Elman	Y	ELM-lin	15	2.5E-1 ± 9.1E-2	1.30E-1 ± 9.83E-16	3.61E-1 ± 1.35E-15	2.60E-1 ± 5.54E-16
Elman	Y	ELM-rand	10	3.12E-3 ± 6.98E-3	1.03E-1 ± 6.44E-3	3.21E-1 ± 9.90E-3	2.31E-1 ± 3.15E-3
Fully connected	N	ELM-lin	25	2.84E-1 ± 3.00E-2	1.30E-1 ± 8.15E-15	3.61E-1 ± 1.13E-14	2.60E-1 ± 8.06E-15
Fully connected	N	ELM-rand	30	3.12E-3 ± 6.98E-3	1.01E-1 ± 7.12E-3	3.18E-1 ± 1.11E-2	2.30E-1 ± 7.44E-3
Fully connected	Y	ELM-lin	45	4.39E+0 ± 3.80E-1	1.30E-1 ± 1.09E-14	3.61E-1 ± 1.51E-14	2.60E-1 ± 9.42E-15
Fully connected	Y	ELM-rand	40	6.24E-3 ± 8.54E-3	1.04E-1 ± 5.43E-3	3.23E-1 ± 8.44E-3	2.33E-1 ± 5.79E-3
Jordan	N	ELM-rand	35	5.93E-2 ± 1.31E-2	1.45E-1 ± 5.80E-4	3.80E-1 ± 7.62E-4	2.77E-1 ± 4.72E-4
Jordan	Y	ELM-rand	10	3.74E-2 ± 1.40E-2	9.96E-2 ± 1.11E-3	3.16E-1 ± 1.75E-3	2.31E-1 ± 1.08E-3
LSTM	N	ELM-rand	45	3.34E-1 ± 3.92E-2	1.45E-1 ± 6.97E-3	3.81E-1 ± 9.30E-3	2.76E-1 ± 7.02E-3
LSTM	Y	ELM-rand	40	3.70E+1 ± 1.11E+0	1.88E-2 ± 4.35E-5	1.37E-1 ± 1.59E-4	7.65E-2 ± 5.23E-4
NARMAX	N	ELM-rand	35	6.55E-2 ± 1.71E-2	1.45E-1 ± 2.05E-3	3.81E-1 ± 2.69E-3	2.77E-1 ± 1.82E-3
NARMAX	Y	ELM-rand	10	7.63E-1 ± 8.64E-2	1.93E-2 ± 7.60E-5	1.39E-1 ± 2.74E-4	7.84E-2 ± 9.40E-4
NARX	N	BPTT	10	1.27E+0 ± 1.97E+0	7.04E-3 ± 7.82E-3	7.61E-2 ± 3.94E-2	4.57E-2 ± 1.33E-2
Feedforward [5]	N	ELM-rand	20	3.5E-1 ± 7.9E-2	4.24E-3 ± 3.36E-3	6.09E-2 ± 2.55E-2	3.16E-2 ± 1.26E-2
Feedforward [2]	N	ELM-rand	20	9.38E-3 ± 2.10E-2	8.64E-3 ± 9.23E-3	7.91E-2 ± 5.46E-2	3.40E-2 ± 1.97E-2
Feedforward [3]	Y	ELM-rand	15	3.12E-3 ± 6.98E-3	9.51E-2 ± 1.11E-3	3.08E-1 ± 1.80E-3	2.25E-1 ± 1.45E-3
SVR	–	–	Gaussian, 7.3806	2.57E+0 ± 0.0E+0	**2.63E-4 ± 0.0E+0**	**1.62E-2 ± 0.0E+0**	**1.16E-2 ± 0.0E+0**

Table 13
Performance on Quebec births data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	40	5.20E-3 ± 9.01E-3	1.15E-2 ± 3.25E-16	1.07E-1 ± 1.51E-15	8.43E-2 ± 1.32E-15
Elman	N	ELM-rand	10	3.12E-3 ± 6.98E-3	2.15E-2 ± 1.17E-2	1.42E-1 ± 3.91E-2	8.55E-2 ± 8.01E-4
Elman	N	BPTT	20	6.55E-1 ± 3.15E-1	1.07E-2 ± 2.99E-3	1.03E-1 ± 1.38E-2	8.18E-2 ± 9.63E-3
Elman	Y	ELM-lin	35	9.36E-3 ± 8.54E-3	1.15E-2 ± 2.69E-16	1.07E-1 ± 1.25E-15	8.43E-2 ± 1.04E-15
Elman	Y	ELM-rand	10	3.12E-3 ± 6.98E-3	1.39E-2 ± 2.25E-3	1.17E-1 ± 9.50E-3	8.50E-2 ± 5.67E-4
Fully connected	N	ELM-lin	15	4.37E-2 ± 6.98E-3	1.15E-2 ± 1.75E-15	1.07E-1 ± 8.16E-15	8.43E-2 ± 6.40E-15
Fully connected	N	ELM-rand	40	1.04E-2 ± 9.01E-3	1.57E-2 ± 2.33E-3	1.25E-1 ± 9.44E-3	9.50E-2 ± 5.08E-3
Fully connected	Y	ELM-lin	25	2.00E-1 ± 1.31E-2	1.15E-2 ± 6.84E-16	1.07E-1 ± 3.19E-15	8.43E-2 ± 2.54E-15
Fully connected	Y	ELM-rand	10	2.7E-1 ± 7.1E-2	1.65E-2 ± 7.15E-3	1.26E-1 ± 2.52E-2	8.10E-2 ± 1.10E-3
Jordan	N	ELM-rand	20	3.43E-2 ± 6.98E-3	4.37E-2 ± 2.05E-2	2.05E-1 ± 4.72E-2	1.75E-1 ± 5.07E-2
Jordan	Y	ELM-rand	15	4.06E-2 ± 1.40E-2	3.04E-2 ± 6.17E-3	1.74E-1 ± 1.80E-2	1.42E-1 ± 1.91E-2
LSTM	N	ELM-rand	20	1.37E-1 ± 6.98E-3	1.63E-1 ± 1.25E-1	3.73E-1 ± 1.71E-1	3.51E-1 ± 1.78E-1
LSTM	Y	ELM-rand	20	2.59E-1 ± 5.64E-2	3.22E-2 ± 1.40E-2	1.75E-1 ± 3.87E-2	1.48E-1 ± 4.02E-2
NARMAX	N	ELM-rand	10	3.12E-2 ± 1.59E-15	4.27E-2 ± 2.42E-2	2.00E-1 ± 5.66E-2	1.74E-1 ± 6.07E-2
NARMAX	Y	ELM-rand	10	2.75E-1 ± 6.96E-2	3.08E-2 ± 2.46E-3	1.75E-1 ± 7.09E-3	1.42E-1 ± 8.53E-3
NARX	N	BPTT	20	6.35E-1 ± 4.22E-2	1.29E-2 ± 1.74E-3	1.14E-1 ± 7.89E-3	8.89E-2 ± 5.07E-3
Feedforward [5]	N	ELM-rand	25	3.1E-1 ± 8.3E-2	3.89E-3 ± 9.84E-5	6.23E-2 ± 7.88E-4	4.86E-2 ± 6.44E-4
Feedforward [2]	N	ELM-rand	35	3.5E-1 ± 1.2E-1	**3.88E-3 ± 6.17E-5**	**6.23E-2 ± 4.97E-4**	**4.85E-2 ± 3.10E-4**
Feedforward [3]	Y	ELM-rand	20	3.9E-1 ± 7.8E-2	1.14E-2 ± 4.74E-08	1.07E-1 ± 2.22E-07	8.37E-2 ± 3.65E-07
SVR	–	–	Polynomial, 0.16565	3.12E-2 ± 0.0E+0	2.06E-1 ± 0.0E+0	4.54E-1 ± 0.0E+0	4.28E-1 ± 0.0E+0

Table 14
Performance on Santa Fe data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	40	3.12E-3 ± 6.98E-3	2.38E-2 ± 5.30E-16	1.54E-1 ± 1.72E-15	1.20E-1 ± 1.84E-15
Elman	N	ELM-rand	10	9.36E-3 ± 2.09E-2	5.73E-2 ± 5.10E-2	2.23E-1 ± 9.65E-2	1.05E-1 ± 1.14E-2
Elman	N	BPTT	15	5.10E-1 ± 1.28E-1	4.86E-2 ± 3.11E-2	2.10E-1 ± 7.51E-2	1.57E-1 ± 4.71E-2
Elman	Y	ELM-lin	45	3.12E-3 ± 6.98E-3	2.38E-2 ± 2.44E-16	1.54E-1 ± 7.90E-16	1.20E-1 ± 8.38E-16
Elman	Y	ELM-rand	15	3.12E-3 ± 6.98E-3	1.28E-1 ± 7.39E-2	3.46E-1 ± 1.04E-1	1.73E-1 ± 2.13E-2
Fully connected	N	ELM-lin	15	3.43E-2 ± 6.98E-3	2.38E-2 ± 2.80E-15	1.54E-1 ± 9.08E-15	1.20E-1 ± 9.30E-15
Fully connected	N	ELM-rand	45	3.2E-1 ± 7.5E-2	2.92E-2 ± 1.65E-2	1.63E-1 ± 5.10E- 2	1.11E-1 ± 4.50E-2
Fully connected	Y	ELM-lin	30	4.49E-1 ± 9.95E-2	2.38E-2 ± 3.39E-15	1.54E-1 ± 1.10E-14	1.20E-1 ± 1.28E-14
Fully connected	Y	ELM-rand	50	3.12E-3 ± 6.98E-3	1.26E-2 ± 2.58E-3	1.11E-1 ± 1.12E-2	7.46E-2 ± 8.87E-3
Jordan	N	ELM-rand	15	2.50E-2 ± 8.54E-3	6.08E-2 ± 6.12E-3	2.46E-1 ± 1.26E-2	1.78E-1 ± 6.51E-3
Jordan	Y	ELM-rand	10	2.81E-2 ± 6.98E-3	6.01E-2 ± 6.03E-3	2.45E-1 ± 1.24E-2	1.77E-1 ± 6.05E-3
LSTM	N	ELM-rand	35	1.40E-1 ± 1.56E-2	7.57E-2 ± 4.01E-2	2.69E-1 ± 6.66E-2	2.23E-1 ± 5.83E-2
LSTM	Y	ELM-rand	10	4.38E-1 ± 3.31E-1	5.01E-2 ± 1.35E-3	2.24E-1 ± 3.03E-3	1.51E-1 ± 2.72E-3
NARMAX	N	ELM-rand	20	2.18E-2 ± 8.54E-3	6.16E-2 ± 6.41E-3	2.48E-1 ± 1.29E-2	1.80E-1 ± 6.53E-3
NARMAX	Y	ELM-rand	25	2.84E-1 ± 7.49E-2	5.24E-2 ± 1.01E-3	2.29E-1 ± 2.21E-3	1.57E-1 ± 1.66E-3
NARX	N	BPTT	15	5.63E-1 ± 3.37E-1	5.03E-2 ± 6.49E-3	2.24E-1 ± 1.41E-2	1.75E-1 ± 1.13E-2
Feedforward [5]	N	ELM-rand	10	3.1E-1 ± 1.3E-1	**1.41E-2 ± 2.52E-3**	**1.18E-1 ± 1.10E-2**	**1.03E-1 ± 1.29E-2**
Feedforward [2]	N	ELM-rand	10	2.9E-1 ± 1.1E-1	2.34E-2 ± 1.71E-2	1.46E-1 ± 5.11E-2	1.29E-1 ± 5.43E-2
Feedforward [3]	Y	ELM-rand	15	3.6E-1 ± 7.4E-2	3.42E-2 ± 1.97E-4	1.85E-1 ± 5.32E-4	1.40E-1 ± 6.68E-4
SVR	–	–	Polynomial, 2.949	1.77E+0 ± 0.0E+0	1.69E-2 ± 0.0E+0	1.30E-1 ± 0.0E+0	1.22E-1 ± 0.0E+0

Table 15
Performance on Deep Tesla data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	10	6.6E-28 ± 4.9E-28	6.3E-3 ± 1.3E-2	1.0E-1 ± 2.2E-14	1.0E-1 ± 2.2E-14
Elman	N	ELM-rand	15	1.0E-4 ± 1.1E-4	3.1E-3 ± 6.3E-3	1.0E-1 ± 2.1E-3	9.9E-2 ± 1.3E-3
Elman	N	BPTT	15	1.1E-2 ± 1.2E-2	1.8E-2 ± 1.5E-3	1.5E-1 ± 4.7E-2	1.4E-1 ± 4.3E-2
Elman	Y	ELM-lin	10	8.6E-28 ± 8.3E-28	6.3E-3 ± 1.3E-2	1.0E-1 ± 2.7E-14	1.0E-1 ± 2.7E-14
Elman	Y	ELM-rand	10	2.1E-12 ± 4.0E-12	1.0E-2 ± 2.0E-12	1.0E-1 ± 1.4E-6	1.0E-1 ± 1.4E-6
Fully connected	N	ELM-lin	10	1.2E-27 ± 2.0E-27	1.0E-2 ± 8.4E-28	1.0E-1 ± 2.9E-14	1.0E-1 ± 2.9E-14
Fully connected	N	ELM-rand	10	1.9E-3 ± 1.2E-3	6.3E-3 ± 1.3E-2	1.4E-1 ± 5.5E-2	1.1E-1 ± 1.2E-2
Fully connected	Y	ELM-lin	15	2.3E-28 ± 1.8E-28	6.3E-3 ± 1.3E-2	1.0E-1 ± 1.6E-14	1.0E-1 ± 1.6E-14
Fully connected	Y	ELM-rand	10	1.6E-12 ± 3.1E-12	1.0E-2 ± 1.4E-12	1.0E-1 ± 1.2E-6	1.0E-1 ± 1.2E-6
Jordan	N	ELM-rand	10	2.3E-2 ± 4.6E-4	1.9E-2 ± 1.5E-2	1.2E-1 ± 7.7E-4	1.1E-1 ± 5.0E-4
Jordan	Y	ELM-rand	45	3.4E-7 ± 8.0E-8	3.1E-3 ± 6.3E-3	1.0E-1 ± 2.6E-5	1.0E-1 ± 3.2E-5
LSTM	N	ELM-rand	20	3.8E-1 ± 1.6E-1	3.1E-3 ± 6.3E-3	4.7E-1 ± 3.4E-1	4.5E-1 ± 3.5E-1
LSTM	Y	ELM-rand	15	1.1E-8 ± 7.3E-9	3.1E-3 ± 6.3E-3	1.0E-1 ± 6.9E-5	1.0E-1 ± 6.8E-5
NARMAX	N	ELM-rand	10	2.3E-2 ± 2.4E-4	3.1E-3 ± 6.3E-3	1.2E-1 ± 2.4E-4	1.1E-1 ± 2.4E-4
NARMAX	Y	ELM-rand	10	1.5E-8 ± 8.0E-9	1.6E-2 ± 2.0E-2	1.0E-1 ± 1.0E-5	1.0E-1 ± 1.3E-5
NARX	N	BPTT	30	3.3E-4 ± 1.4E-5	4.7E-2 ± 3.2E-3	2.5E-1 ± 7.7E-2	2.3E-1 ± 7.3E-2
Feedforward [5]	N	ELM-rand	15	6.7E-5 ± 1.0E-4	**1.6E-2 ± 2.0E-2**	**9.9E-2 ± 1.7E-3**	**9.9E-2 ± 1.8E-3**
Feedforward [3]	Y	ELM-rand	10	1.5E-12 ± 1.5E-12	1.0E-2 ± 6.6E-13	1.0E-1 ± 8.1E-7	1.0E-1 ± 8.2E-7
SVR	–	–	Linear, 41.44	6.2E-7 ± 0.0E+0	6.9E-4 ± 0.0E+0	1.0E-1 ± 0.0E+0	1.0E-1 ± 0.0E+0

Table 16
Performance on Japan population data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	10	1.8E-3 ± 4.0E-18	1.3E-2 ± 1.5E-2	4.8E-2 ± 2.9E-16	1.3E-2 ± 1.9E-15
Elman	N	ELM-rand	10	1.6E-3 ± 2.4E-5	9.0E-3 ± 3.4E-3	9.5E-2 ± 5.8E-2	1.5E-2 ± 3.8E-3
Elman	N	BPTT	30	1.3E-2 ± 1.4E-2	2.5E-2 ± 4.7E-3	1.1E-1 ± 6.4E-2	6.7E-2 ± 5.4E-2
Elman	Y	ELM-lin	10	1.8E-3 ± 0.0E+0	6.3E-3 ± 1.3E-2	4.8E-2 ± 0.0E+0	1.3E-2 ± 2.0E-16
Elman	Y	ELM-rand	10	1.4E-3 ± 9.0E-5	3.2E+2 ± 1.2E+3	1.8E+1 ± 3.4E+1	7.3E-1 ± 1.3E+0
Fully connected	N	ELM-lin	10	1.6E-3 ± 1.9E-5	7.9E-3 ± 1.4E-3	8.9E-2 ± 3.7E-2	1.4E-2 ± 2.0E-3
Fully connected	N	ELM-rand	10	1.8E-3 ± 4.0E-18	6.3E-3 ± 1.3E-2	4.8E-2 ± 1.6E-15	1.3E-2 ± 1.2E-14
Fully connected	Y	ELM-lin	10	1.8E-3 ± 4.0E-18	1.3E-2 ± 1.5E-2	4.8E-2 ± 6.7E-16	1.3E-2 ± 4.8E-15
Fully connected	Y	ELM-rand	35	1.1E-3 ± 1.2E-4	1.3E-2 ± 1.5E-2	2.4E-1 ± 3.2E-1	2.3E-2 ± 1.6E-2
Jordan	N	ELM-rand	15	1.6E-1 ± 3.9E-3	6.3E-3 ± 7.7E-3	4.5E-1 ± 6.0E-2	4.4E-1 ± 5.9E-2
Jordan	Y	ELM-rand	10	1.5E-1 ± 8.1E-3	1.6E-2 ± 9.9E-3	3.7E-1 ± 6.8E-2	3.6E-1 ± 6.7E-2
LSTM	N	ELM-rand	10	8.7E-2 ± 9.5E-2	1.3E-2 ± 6.3E-3	3.2E-1 ± 1.0E-1	3.1E-1 ± 1.0E-1
LSTM	Y	ELM-rand	10	1.3E-1 ± 3.2E-2	1.6E-2 ± 0.0E+0	4.0E-1 ± 4.3E-2	3.9E-1 ± 4.2E-2
NARMAX	N	ELM-rand	15	1.5E-1 ± 1.0E-2	1.6E-2 ± 0.0E+0	4.3E-1 ± 3.3E-2	4.2E-1 ± 3.3E-2
NARMAX	Y	ELM-rand	15	1.5E-1 ± 6.2E-3	1.6E-2 ± 1.7E-2	3.9E-1 ± 2.3E-2	3.8E-1 ± 2.2E-2
NARX	N	BPTT	20	2.1E-3 ± 7.8E-4	6.7E-2 ± 3.4E-3	7.1E-2 ± 5.9E-3	2.5E-2 ± 4.9E-3
Feedforward [5]	N	ELM-rand	10	1.6E-3 ± 8.9E-6	**3.1E-3 ± 6.3E-3**	**3.9E-2 ± 3.2E-4**	**8.7E-3 ± 2.9E-4**
Feedforward [3]	Y	ELM-rand	20	1.5E-3 ± 2.2E-5	1.3E-3 ± 2.7E-7	3.7E-2 ± 5.2E-4	9.2E-3 ± 4.8E-5
SVR	–	–	Linear, 0.0010492	1.9E-3 ± 0.0E+0	1.5E-3 ± 0.0E+0	5.0E-2 ± 0.0E+0	8.1E-3 ± 0.0E+0

Table 17
Performance on SP500 data set.

Architecture	DL	Training algorithm	Hyper-parameters	Training time (avg ± std)	Testing MSE (avg ± std)	Testing RMSE (avg ± std)	Testing MAE (avg ± std)
Elman	N	ELM-lin	10	2.0E-6 ± 0.0E+0	1.3E-2 ± 1.2E-2	1.0E+0 ± 3.6E-14	9.7E-1 ± 3.4E-14
Elman	N	ELM-rand	45	7.7E-3 ± 1.6E-3	**1.6E-2 ± 1.4E-2**	**1.7E-1 ± 8.7E-3**	**1.2E-1 ± 3.6E-3**
Elman	N	BPTT	50	5.8E-4 ± 2.7E-4	1.2E-1 ± 1.1E-2	6.9E-1 ± 3.3E-1	6.2E-1 ± 3.1E-1
Elman	Y	ELM-lin	10	2.0E-6 ± 4.1E-21	6.3E-3 ± 7.7E-3	1.0E+0 ± 6.8E-14	9.7E-1 ± 6.4E-14
Elman	Y	ELM-rand	10	2.0E-6 ± 1.4E-8	1.0E+0 ± 2.6E-6	1.0E+0 ± 1.6E-3	9.7E-1 ± 1.4E-3
Fully connected	N	ELM-lin	10	2.0E-6 ± 0.0E+0	2.8E-2 ± 6.3E-3	1.0E+0 ± 7.8E-14	9.7E-1 ± 7.3E-14
Fully connected	N	ELM-rand	10	2.3E-2 ± 4.5E-3	1.6E-2 ± 0.0E+0	2.9E-1 ± 3.8E-2	2.4E-1 ± 4.4E-2
Fully connected	Y	ELM-lin	10	2.0E-6 ± 4.1E-21	5.9E-2 ± 1.8E-2	1.0E+0 ± 1.8E-13	9.7E-1 ± 1.8E-13
Fully connected	Y	ELM-rand	10	2.0E-6 ± 1.4E-8	1.6E-2 ± 9.9E-3	1.0E+0 ± 9.6E-4	9.7E-1 ± 7.3E-4
Jordan	N	ELM-rand	35	6.7E-1 ± 3.7E-3	7.2E-2 ± 1.6E-2	5.0E-1 ± 1.3E-3	4.7E-1 ± 1.1E-3
Jordan	Y	ELM-rand	15	3.5E-6 ± 8.9E-8	9.1E-2 ± 1.8E-2	1.0E+0 ± 7.7E-5	9.6E-1 ± 6.2E-5
LSTM	N	ELM-rand	10	7.0E-2 ± 3.5E-2	3.0E-1 ± 2.5E-2	5.8E-1 ± 2.9E-1	5.5E-1 ± 3.0E-1
LSTM	Y	ELM-rand	10	3.8E-6 ± 7.4E-8	2.8E-1 ± 3.2E-2	1.0E+0 ± 4.2E-5	9.6E-1 ± 3.5E-5
NARMAX	N	ELM-rand	30	6.7E-1 ± 9.6E-3	1.0E-1 ± 1.6E-2	5.0E-1 ± 2.3E-4	4.7E-1 ± 2.4E-4
NARMAX	Y	ELM-rand	25	3.3E-6 ± 1.3E-8	8.4E-2 ± 1.3E-2	1.0E+0 ± 5.2E-5	9.6E-1 ± 4.2E-5
NARX	N	BPTT	20	3.2E-6 ± 2.2E-8	3.7E-1 ± 3.7E-2	1.0E+0 ± 1.5E-1	9.8E-1 ± 1.5E-1
Feedforward [5]	N	ELM-rand	20	9.2E-3 ± 1.6E-3	3.2E-2 ± 1.4E-4	1.8E-1 ± 1.2E-2	1.2E-1 ± 9.7E-3
Feedforward [3]	Y	ELM-rand	10	1.9E-6 ± 4.1E-8	1.0E+0 ± 8.4E-6	1.0E+0 ± 2.9E-3	9.7E-1 ± 2.5E-3
SVR	–	–	Polynomial, 0.15218	1.0E+4 ± 0.0E+0	1.1E+0 ± 0.0E+0	1.0E+0 ± 0.0E+0	9.9E-1 ± 0.0E+0

However, feedforward networks' MSE and MAE were on average 2.2 × and 1.03 × larger than SVR's MSE and MAE, respectively.

ELM have been known to overfit, especially as the number of hidden layer neurons increases. To minimize the possibility of choosing models that were overfitting, we penalized networks with a large number of hidden layers by choosing less complex models when the performance of both models was arbitrarily close. In future work, we will also investigate incorporating a regularization term in the objective function that has been shown to avoid overfitting [54].

Finally, we note that some data sets resulted in large prediction errors due to data ill conditioning and random initialization. Since the network architecture contains recurrent connections, unsuitable initial conditions can cause the error to increase dramatically and the network is unable to correct it. Therefore, for recurrent architectures, non-iterative training techniques should take certain measures to insure a good initialization and matrix conditioning, which will be investigated in future work.

5.3. Non-iteratively-trained RNNstatistical analysis

We performed a statistical analysis to compare the performance of the variants on all the databases in this work. The tests were performed on 16 non-iteratively-trained RNN variants considered in this work. Specifically, the Friedman ranking and ANOVA tests resulted in p-values equal to 0 and 0.0086, respectively. Both values are less than the significance level set to $\alpha = 0.05$, which implies the presence of a significant difference between the algorithms. Furthermore, the Nemenyi post-hoc was performed once a significant difference was observed, to rank the algorithms. The rankings are shown in Fig. 4. The critical distance was found to be equal to 0.2.

Based on Nemenyi's ranking algorithm, LSTM trained using ELM-rand without DL was the highest ranked algorithm, even though it did not always have the lowest error measure (as observed in Tables 7–17). LSTM trained using ELM-rand with DL came in second place. However, NARMAX trained using ELM-rand without DL was the worst algorithm. Furthermore, the difference between Elman trained using ELM-lin with DL and fully connected RNN trained using ELM-rand with DL was less than the critical distance which implies that difference is insignificant. Feedforward networks ranked fourth and ninth which implies that RNN architectures were more suitable for the databases given their time series properties. All algorithms joined by vertical lines in Fig. 4 were not significantly better than each other. Furthermore, we notice that the fully connected RNN, which is the most general architecture (as discussed in Section 4.2), is the most difficult to train since it ranked in the bottom half. Its special cases, obtained from fully connected RNN by pruning some connections, had smaller search spaces that allowed the convergence to better solutions on the given datasets. The rankings were based on the databases used in this work which may not have been suitable for the lower ranked architectures. One should keep in mind that these architectures may still be beneficial for other applications since research has shown that the optimal network architecture is data dependent.

5.4. Parameter sensitivity analysis

Table 18 compares the performance of non-iteratively-trained RNN when different activation functions are used. The number of hidden neurons was fixed to 20 for this analysis. We notice that the sine activation function produced less ill-conditioned problems. Fully connected RNN trained using ELM-rand with a sigmoid activation function would sometimes lead to an ill-conditioned H matrix, resulting in very large MSE. This problem was also aggravated when the number of hidden neurons increased. In addition, the

Table 18
Effect of kernel choice on nRMSE, nMAE and training time.

Architecture	DL	Training algorithm	Training time (s)		nRMSE		nMAE	
			Sigmoid	Sine	Sigmoid	Sine	Sigmoid	Sine
Bebida								
Elman	N	ELM-lin	0.003	0.012	0.301	0.375	0.225	0.314
Elman	N	ELM-rand	0.045	0.030	0.283	1.959	0.208	1.730
Elman	N	BPTT	0.220	0.185	0.794	0.750	0.631	0.632
Elman	Y	ELM-lin	0.018	0.013	0.301	0.375	0.225	0.314
Elman	Y	ELM-rand	0.032	0.012	0.295	0.337	0.215	0.251
Fully connected	N	ELM-lin	0.009	0.037	0.301	0.375	0.225	0.314
Fully connected	N	ELM-rand	0.029	0.030	3.457	2.544	0.768	2.398
Fully connected	Y	ELM-lin	0.022	0.050	0.301	0.375	0.225	0.314
Fully connected	Y	ELM-rand	0.007	0.003	0.372	0.348	0.242	0.252
Jordan	N	ELM-rand	0.006	0.009	0.419	2.461	0.315	2.430
Jordan	Y	ELM-rand	0.006	0.013	0.325	0.325	0.229	0.229
LSTM	N	ELM-rand	0.016	0.044	2.667	2.037	2.655	2.017
NARMAX	N	ELM-rand	0.003	0.012	0.513	2.436	0.442	2.392
NARX	N	BPTT	0.254	0.186	0.796	0.732	0.572	0.567
Feedforward [5]	N	ELM-rand	0.031	0.031	1.4441	2.585	0.8304	2.453
Feedforward [3]	Y	ELM-rand	0.030	0.046	0.311	0.434	0.226	0.294
IPI								
Elman	N	ELM-lin	0.047	0.046	0.183	0.436	0.148	0.388
Elman	N	ELM-rand	0.032	0.032	0.219	1.067	0.185	0.810
Elman	N	BPTT	0.285	0.205	0.489	0.417	0.377	0.342
Elman	Y	ELM-lin	0.030	0.032	0.183	0.436	0.148	0.388
Elman	Y	ELM-rand	0.015	0.012	0.185	0.227	0.156	0.188
Fully connected	N	ELM-lin	0.016	0.050	0.183	0.436	0.148	0.388
Fully connected	N	ELM-rand	0.015	0.012	0.3775	2.207	2.608	2.026
Fully connected	Y	ELM-lin	0.016	0.062	0.183	0.436	0.148	0.388
Fully connected	Y	ELM-rand	0.031	0.012	0.1974	0.192	1.930	0.159
Jordan	N	ELM-rand	0.010	0.012	0.342	2.002	0.273	1.975
Jordan	Y	ELM-rand	0.006	0.010	0.208	0.208	0.170	0.170
LSTM	N	ELM-rand	0.016	0.041	1.728	1.699	1.703	1.675
NARMAX	N	ELM-rand	0.006	0.012	0.379	2.019	0.311	1.961
NARX	N	BPTT	0.212	0.186	0.606	0.570	0.513	0.458
Feedforward [5]	N	ELM-rand	0.016	0.012	0.402	0.3875	0.356	1.964
Feedforward [3]	Y	ELM-rand	0.050	0.047	0.240	0.4037	0.194	6.653
Lavras								
Elman	N	ELM-lin	0.016	0.012	0.158	0.152	0.114	0.108
Elman	N	ELM-rand	0.031	0.047	0.176	0.254	0.122	0.182
Elman	N	BPTT	0.432	0.267	0.543	0.178	0.443	0.125
Elman	Y	ELM-lin	0.015	0.047	0.158	0.152	0.114	0.108
Elman	Y	ELM-rand	0.016	0.031	0.152	0.179	0.110	0.121
Fully connected	N	ELM-lin	0.034	0.037	0.158	0.152	0.114	0.108
Fully connected	N	ELM-rand	0.031	0.015	0.1604	0.264	0.117	0.191
Fully connected	Y	ELM-lin	0.037	0.050	0.158	0.152	0.114	0.108
Fully connected	Y	ELM-rand	0.031	0.012	0.237	0.180	0.126	0.124
Jordan	N	ELM-rand	0.009	0.031	0.166	0.250	0.118	0.180
Jordan	Y	ELM-rand	0.012	0.037	0.174	0.174	0.114	0.114
LSTM	N	ELM-rand	0.034	0.078	0.232	0.213	0.170	0.151
NARMAX	N	ELM-rand	0.009	0.053	0.197	0.251	0.152	0.181
NARX	N	BPTT	1.617	0.208	0.152	0.150	0.111	0.110
Feedforward [5]	N	ELM-rand	0.047	0.012	0.1345	0.257	0.0910	0.189
Feedforward [3]	Y	ELM-rand	0.0469	0.031	0.189	0.184	0.119	0.127
Inverted Pendulum								
Elman	N	ELM-lin	0.016	0.019	0.074	0.075	0.044	0.046
Elman	N	ELM-rand	0.003	0.003	0.065	0.070	0.038	0.040
Elman	N	BPTT	1.788	1.780	0.083	0.117	0.047	0.068
Elman	Y	ELM-lin	0.025	0.016	0.074	0.075	0.044	0.046
Elman	Y	ELM-rand	0.009	0.003	0.066	0.067	0.039	0.039
Fully connected	N	ELM-lin	0.568	0.512	0.074	0.075	0.044	0.046
Fully connected	N	ELM-rand	0.016	0.019	0.684	0.122	0.071	0.064
Fully connected	Y	ELM-lin	0.580	0.484	0.074	0.075	0.044	0.046
Fully connected	Y	ELM-rand	0.022	0.019	0.071	0.078	0.041	0.044
Jordan	N	ELM-rand	0.296	0.306	0.097	0.100	0.063	0.062
Jordan	Y	ELM-rand	0.321	0.312	0.078	0.083	0.047	0.043
LSTM	N	ELM-rand	1.822	2.065	0.099	0.096	0.064	0.059
NARMAX	N	ELM-rand	0.293	0.312	0.096	0.098	0.062	0.063
NARX	N	BPTT	1.837	2.012	0.013	0.014	0.005	0.005
Feedforward [5]	N	ELM-rand	0.006	0.003	0.008	0.033	0.003	0.012
Feedforward [3]	Y	ELM-rand	0.006	0.003	0.085	0.0801	0.041	0.502

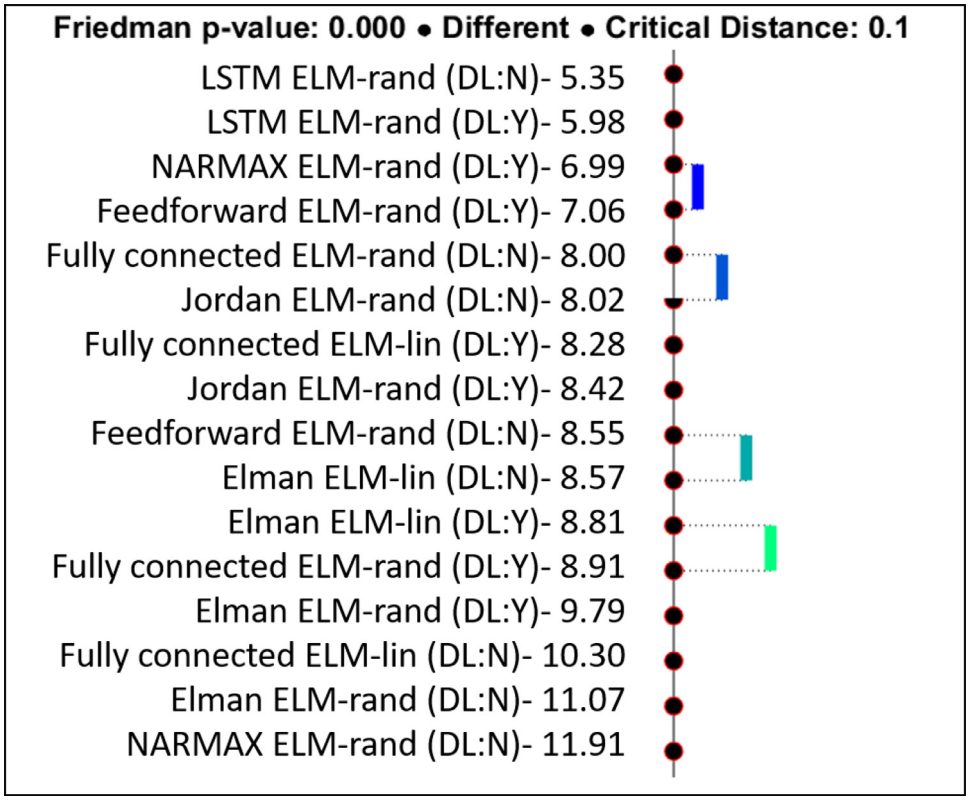

Fig. 4. Non-iteratively-trained RNN variants ranking based on Nemenyi post-hoc test.

Table 19
Effect of number of hidden neurons on MSE and training time.

Architecture	DL	Training algorithm	Number of hidden neurons	Test set MSE Electricity load balance	Training time (s)	Test set MSE Inverted pendulum	Training time (s)
Elman	N	ELM-lin	15	0.0686	0.720	0.0175	0.002
			20	0.0686	1.092	0.0175	0.004
			25	0.0686	1.320	0.0175	0.004
			30	0.0686	1.660	0.0175	0.008
Elman	N	ELM-rand	15	0.0694	0.204	0.0135	0.002
			20	0.0700	0.318	0.0135	0.016
			25	0.0700	0.492	0.0138	0.002
			30	0.0695	0.790	0.0132	0.006
Fully connected	N	ELM-lin	15	0.0686	14.258	0.0175	0.072
			20	0.0686	33.346	0.0175	0.258
			25	0.0686	59.298	0.0175	0.702
			30	0.0686	83.766	0.0175	2.646
Fully connected	N	ELM-rand	15	0.0691	1.056	0.0699	0.010
			20	0.0679	1.262	0.0141	0.010
			25	0.0686	1.476	0.0485	0.010
			30	0.0755	1.706	0.0167	0.014
LSTM	N	ELM-rand	15	2.0650	6.879	0.0314	1.323
			20	0.2729	13.167	0.0293	1.822
			25	0.1438	15.025	0.0318	2.493
			30	0.6983	18.798	0.0293	3.179

linearized sigmoid activation function resulted in better RNN models than ELM-rand. The sine activation function exhibited the opposite behavior for some data sets. This could be due to the low order approximation and could be remedied by adopting higher order approximations. Furthermore, the approximation might not be valid for certain data distributions. This should also be considered in the approximation procedure.

Focusing on the effect of the number of hidden neurons on non-iteratively-trained RNN, Table 19 summarizes the obtained results for the inverted pendulum and electricity load data sets. A sigmoid activation function was adopted for this analysis. As expected, the computational cost of non-iteratively-trained fully connected RNN is greater than Elman RNN, especially as the number of hidden neurons increases. As previously mentioned, fully connected RNN with ELM-rand can result in ill-conditioned models that get worse as the network size increases. Comparing ELM-rand to ELM-lin, the former is less computationally expensive than the latter, since the output matrix and parameter vector are smaller

in size. For example, considering the Electricity load balance data set, fully connected RNN with ELM-rand is 15% to 49% times faster than fully connected RNN with ELM-lin.

6. Conclusion

In this paper, we proposed to train RNN using a non-iterative randomized algorithm for time series prediction and sequential decision-making problems, to reduce training time. We focused on Elman, Jordan, fully connected RNN, NARMAX and LSTM RNN architectures with and without DL. Since learning the recurrent connections for these architectures is a non-linear problem, we proposed two approaches to train RNN non-iteratively. The first approach randomly assigns the values of the recurrent connections while the second approach linearizes the activation function and learns these weights.

In addition to performing a theoretical computational complexity of various non-iteratively trained RNN architectures, experimental validation was performed on 14 publicly available time series prediction data sets and the double inverted pendulum POMDP sequential decision-making problem. Multiple error measures, training time and repeatability were compared to other methods in the literature including non-iteratively-trained Jordan RNN and feedforward ANN, BPTT trained RNN and SVR. Results showed that even though RNN trained using BPTT and SVR achieve lower MSE, non-iteratively-trained RNN require significantly less time to train. This tradeoff is especially beneficial in real-time decision-making problems on computationally challenged platforms such as in communication challenged robotics, i.e. robots in environments that prevent them from offloading their computations to the cloud. While the Nemenyi post hoc test showed the superiority of the proposed approaches compared to other work in the literature, the proposed algorithms can be further improved to avoid diverging solutions when recurrent weights are badly initialized and assess its performance on deeper networks which will be subject of future work. Furthermore, future work will investigate the proposed algorithms in anytime control problems where the computationally efficient learner is used to provide a quick first guess and a more accurate learner enhances this guess.

In addition, non-iteratively-trained Elman RNN required less time to train than non-iteratively-trained fully connected RNN even though it performed worse. ELM-lin performed better than ELM-rand on most databases. This prompts the investigation of higher order approximations of the activation function which will be the subject of future work. Finally, LSTM did not perform well compared to other network architectures. This could be due to the size of the training data and high non-linearity of LSTM cells. Therefore, further research is necessary to optimize the non-iterative training algorithm for LSTM networks.

Furthermore, future work will also include the investigation of other activation functions and the practical implications of the theoretical model transformations on real-world problems. Methods to reduce the incurred loss of accuracy while maintaining the computational gains and LSTM with linearized activation functions will also be investigated.

Acknowledgement

This work was supported by the CNRS-L/AUB PhD scholarship from the National Council of Scientific Research in Lebanon and the University Research Board at the American University of Beirut. The authors would like to thank Hmayag Partamian for collecting the pole balancing data.

References

[1] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[2] W.F. Schmidt, M.A. Kraaijveld, R.P. Duin, Feedforward neural networks with random weights, in: Proceedings on the 11th IAPR International Conference on Pattern Recognition, Vol. II. Conference B: Pattern Recognition Methodology and Systems, IEEE, 1992, pp. 1–4.
[3] Y.-H. Pao, G.-H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.
[4] H.A. Te Braake, G. Van Straten, Random activation weight neural net (RAWN) for east non-iterative training, Eng. Appl. Artif. Intell. 8 (1) (1995) 71–80.
[5] G.B. Huang, Q.Y. Zhu, C.K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of the International Joint Conference on Neural Networks, 2, IEEE, 2004, pp. 985–990.
[6] J.L. Elman, Finding structure in time, Cognit. Sci. 14 (2) (1990) 179–211.
[7] M. Jordan, Serial order: a parallel distributed processing approach, Technical Report, Institute for Cognitive Science, California University, La Jolla, San Diego, USA, 1986. June 1985–March 1986
[8] R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural Comput. 1 (2) (1989) 270–280.
[9] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (2) (1994) 240–254.
[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[11] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79.
[12] Ö.F. Ertugrul, Forecasting electricity load by a novel recurrent extreme learning machines approach, Int. J. Electr. Power Energy Syst. 78 (2016) 429–435.
[13] H. Zhao, J. Zhang, Nonlinear dynamic system identification using pipelined functional link artificial recurrent neural network, Neurocomputing 72 (13) (2009) 3046–3054.
[14] D.E. Rumelhart, J.L. McClelland, P.R. Group, et al., Parallel distributed processing: explorations in the microstructures of cognition, Foundations, 1, MIT Press Cambridge, 1986.
[15] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (11) (1997) 2673–2681.
[16] S. El Hihi, Y. Bengio, Hierarchical recurrent neural networks for long-term dependencies., in: Proceedings of the NIPS, Citeseer, 1995, pp. 493–499.
[17] J. Hu, J. Wang, Global stability of complex-valued recurrent neural networks with time-delays, IEEE Trans. Neural Netw. Learn. Syst. 23 (6) (2012) 853–865.
[18] Z. Tu, N. Ding, L. Li, Y. Feng, L. Zou, W. Zhang, Adaptive synchronization of memristive neural networks with time-varying delays and reaction–diffusion term, Appl. Math. Comput. 311 (2017) 118–128.
[19] L. Duan, L. Huang, Z. Guo, X. Fang, Periodic attractor for reaction–diffusion high-order Hopfield neural networks with time-varying delays, Comput. Math. Appl. 73 (2) (2017) 233–245.
[20] D. Pham, S. Oh, A recurrent back propagation neural network for dynamic system identification, J. Syst. Eng. 2 (4) (1992) 213–223.
[21] D. Pham, X. Liu, Dynamic system identification using partially recurrent neural networks, J. Syst. Eng. 2 (2) (1992) 90–97.
[22] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[23] R.J. Williams, D. Zipser, Gradient-based learning algorithms for recurrent networks and their computational complexity, Back-propagation: Theory, architectures and applications (1995) 433–486.
[24] J. Martens, I. Sutskever, Learning recurrent neural networks with hessian-free optimization, in: Proceedings of the 28th International Conference on Machine Learning, 2011, pp. 1033–1040.
[25] X. Fu, S. Li, I. Jaithwa, Implement optimal vector control for lcl-filter-based grid-connected converters by using recurrent neural networks, IEEE Trans. Ind. Electron. 62 (7) (2015) 4443–4454.
[26] X. Wang, Y. Huang, Convergence study in extended Kalman filter-based training of recurrent neural networks, IEEE Trans. Neural Netw. 22 (4) (2011) 588–600.
[27] S. Omkar, D. Mudigere, J. Senthilnath, M.V. Kumar, Identification of helicopter dynamics based on flight data using nature inspired techniques, Int. J. Appl. Metaheuristic Comput. (IJAMC) 6 (3) (2015) 38–52.
[28] A. Blanco, M. Delgado, M.C. Pegalajar, A real-coded genetic algorithm for training recurrent neural networks, Neural Netw. 14 (1) (2001) 93–105.
[29] B. Schrauwen, D. Verstraeten, J. Van Campenhout, An overview of reservoir computing: theory, applications and implementations, in: Proceedings of the 15th European Symposium on Artificial Neural Networks, 2007, pp. 471–482.
[30] H. Jaeger, Adaptive nonlinear system identification with echo state networks, Networks 8 (9) (2003) 17.
[31] W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: a new framework for neural computation based on perturbations, Neural Comput. 14 (11) (2002) 2531–2560.
[32] L. Zhang, P. Suganthan, A survey of randomized algorithms for training neural networks, Inf. Sci. 364 (2016) 146–155.
[33] L. Tang, Y. Wu, L. Yu, A non-iterative decomposition-ensemble learning paradigm using RVFLnetwork for crude oil price forecasting, Appl. Soft Comput. 70 (2018) 1097–1108.
[34] D.P. Mesquita, J.P.P. Gomes, L.R. Rodrigues, S.A. Oliveira, R.K. Galvão, Building selective ensembles of randomization based neural networks with the successive projections algorithm, Appl. Soft Comput. 70 (2018) 1135–1145.

[35] L. Zhang, P. Suganthan, A comprehensive evaluation of random vector functional link networks, Inf. Sci. 367 (2016) 1094–1105.
[36] Y. Ren, P. Suganthan, N. Srikanth, G. Amaratunga, Random vector functional link network for short-term electricity load demand forecasting, Inf. Sci. 367 (2016) 1078–1093.
[37] N. Wang, M.J. Er, M. Han, Generalized single-hidden layer feedforward networks for regression problems, IEEE Trans. Neural Netw. Learn. Syst. 26 (6) (2015) 1161–1176.
[38] J.-C. Yin, A.N. Perakis, N. Wang, A real-time ship roll motion prediction using wavelet transform and variable RBF network, Ocean Eng. 160 (2018) 10–19.
[39] N. Wang, M. Han, N. Dong, M.J. Er, Constructive multi-output extreme learning machine with application to large tanker motion dynamics identification, Neurocomputing 128 (2014) 59–72.
[40] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, N. Sundararajan, On–line sequential extreme learning machine, Comput. Intell. 2005 (2005) 232–237.
[41] N. Wang, M.J. Er, X.-Y. Meng, X. Li, An online self-organizing scheme for parsimonious and accurate fuzzy neural networks, Int. J. Neural Syst. 20 (05) (2010) 389–403.
[42] Y. Gu, J. Liu, Y. Chen, X. Jiang, H. Yu, Toselm: timeliness online sequential extreme learning machine, Neurocomputing 128 (2014) 119–127.
[43] N. Wang, M.J. Er, M. Han, Parsimonious extreme learning machine using recursive orthogonal least squares, IEEE Trans. Neural Netw. Learn. Syst. 25 (10) (2014) 1828–1841.
[44] K. Basterretxea, J. Tarela, I. Del Campo, Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons, IEEE Proc. Circuits Dev. Syst. 151 (1) (2004) 18–24.
[45] M. Q.-Y. ZHU, D. G.-B. HUANG, Basic elm algorithms, 2013. ntu.edu.sg/home/egbhuang/elm_codes.html (accessed 18 October 2018).
[46] A. Weigend, N. Gershenfeld, Time series prediction: forecasting the future and understanding the past. 1994, in: Proceedings of a NATO Advanced Research Workshop on Comparative Time Series Analysis, Santa Fe, New Mexico.
[47] DataMarket, Time series data library, 2016. https://datamarket.com/data/list/?q=provider%3Atsdl (accessed 18 October 2018).
[48] M. Lichman, UCI machine learning repository, 2013. https://archive.ics.uci.edu/ml/index.php (accessed 18 October 2018).
[49] Australian energy market operator, 2015. https://www.aemo.com.au/ (accessed 18 October 2018).
[50] P.A. Morettin, C. Toloi, Analise de series Temporais, Blucher, 2006.
[51] L. Friedman, End-to-end learning from tesla autopilot driving data, 2018. https://github.com/lexfridman/deeptesla (accessed 18 October 2018).
[52] Kaggle, Japan population data, 2018. https://www.kaggle.com/jd1325/japan-population-data (accessed 18 October 2018).
[53] Kaggle, Sp500 since 1950, 2018. https://www.kaggle.com/benjibb/sp500-since-1950 (accessed 18 October 2018).
[54] Y. Cui, J. Zhai, X. Wang, Extreme learning machine based on cross entropy, in: Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC), 2, IEEE, 2016, pp. 1066–1071.

Yara Rizk is a Ph.D. student enrolled in the electrical and computer engineering department at the American University of Beirut (AUB). Prior, she has received her BE in computer and communication engineering from AUB, Lebanon, in 2012. Her research interests span robotics, multi-agent systems, machine learning, classification, clustering, and artificial intelligence. Rizk has attended a technical internship (2013–2014) at Intel in Hillsboro, Oregon, USA and is an active researcher with multiple peer-reviewed publications.

Mariette Awad (M'08) obtained her Ph.D. in Electrical Engineering from the University of Vermont (2007). Her current research focuses on HMI, artificial intelligence, machine learning, data analytics and Internet of Things. Dr. Awad has received 21 grants to support her research including 2 multidisciplinary multi-million dollar grants from the Qatar National Research Fund (QNRF) and Intel. Her work culminated in a book, Efficient Machine Learning, in 2015 as well as more than 70 conference, book chapter, and journal publications. Awad has been an invited speaker and reviewer at a number of international conferences and journals. Prior to her academic position, she was with IBM System and Technology group in Vermont for six years as a wireless product engineer where her technical leadership and innovative spirit has earned her management recognition twice, two business awards, and 10 patents.