Randomising the Simple Recurrent Network: A lightweight, energy-efficient RNN model with application to forecasting problems

Mohammed Elmahdi Khennour[1*], Abdelhamid Bouchachia[2], Mohammed Lamine Kherfi[1,3] and Khadra Bouanane[1]

[1]Lab of Artificial intelligence & Data science, Kasdi Merbah University, Ouargla, Algeria.
[2]Dept of Computing & Informatics, Bournemouth University, Poole, UK.
[3]LAMIA Laboratory, Université du Québec à Trois-Rivières, Canada.

*Corresponding author(s). E-mail(s): khennour.mahdi@univ-ouargla.dz;
Contributing authors: abouchachia@bournemouth.ac.uk; Mohammedlamine.Kherfi@uqtr.ca; bouanane.khadra@univ-ouargla.dz;

Abstract

Multi-variate Time-Series (MTS) forecasting is the prediction of future for a sequence of data. The process of analysing obtained data can benefit the community financially and securely, for instance observing stock exchange trends and predicting malicious attacks whenabout. MTS forecasting models faces many problems including data and model complexity, energy constraints and computational cost. These problems could affect budget allocation, latency and carbon emission. Recurrent neural networks are one of these models, which are knowns for their computational complexity due to slow learning process which requires more energy to train. Contributing to green AI, in this paper, we propose a competitive and energy-efficient light-weight recurrent neural network based on a hybrid neural architecture that combines Random Neural Network (RaNN) and Simple Recurrent Network (SRN), namely

Random Simple Recurrent Network (RSRN). We consider RaNN for its distinctive probabilistic properties and SRN for adding light-weight recurrent ability to the RaNN to process sequential data. The paper shows how RSRN is trained using adapted and optimised versions of back propagation (BP), back propagation through time (BPTT) and truncated BPTT (TBPTT). The latter two algorithms use penalised gradient descent to prevent gradient explosion problems by employing the average of total gradient over time. Evaluated on several datasets, RSRN achieves best performance when using TBPTT. Moreover, we performed a comparative study against well-known recurrent models showing its superiority compared to the state-of-the-art models, while requiring much less computational time and training parameters. In addition, we investigated the multi-layer architecture and its properties.

Keywords: Random Neural Network (RaNN), Simple Recurrent Network (SRN), Random Simple Recurrent Network (RSRN), Time-series data

1 Introduction

Time-series(TS) forecasting is the process of predicting the future for a sequence of data like fuel prices, weather forecasting and power consumption. This prediction helps specialists to take proactive or precautionary actions such as stocks exchange and monitoring packet traffics from malicious attacks. Various models have been proposed to address this kind of prediction. Each one of those models has its advantages and limitations. Recurrent Neural Networks(RNNs) is a dedicated model for handling sequential data; but they suffer from gradient vanishing/explosion problem especially in long sequences. Gated Recurrent Units(GRU) and Long Short-Term Memory(LSTM) are other variants of RNN that have been applied to TS forecasting and solved the RNN problems. Although LSTM/GRU has a remarkable performance; they are complex model, slow to train and draw a lot of energy. The complexity also increases for long sequence multi-variate data. These problems affect in limited-power devices with low computational capabilities, low budget project with complex data or energy concern criteria. To address those difficulties, certain norms must be met such as energy efficiency, low computational cost, ability to handle multi-variate sequential data and high performance. Accordingly, we suggest an energy-efficient lightweight recurrent model based on hybrid of Random Neural Network (RaNN) and Simple Recurrent Network (SRN) with an adapted and penalised gradient descent (GD) algorithms.

RaNN is a model inspired from the biological behaviour of spiking neurons. It was proposed by Gelenbe [1] to simulate the excitation and inhibition signals of neurons in the form of positive and negative signals. Regardless the stochastic behaviour of RaNN, it has a significant property that in steady state, the probability distribution of the network has an easy "Product Form" solution. Also, RaNN shows universal approximator's properties under certain

conditions. G-network [2] is a mathematical model inspired from RaNN and has been applied to several optimisation problems like the work of authors in [3, 4, 5]. RaNN was used to monitor network packet [6] and detect DoS attacks [7]. Also, it was used in high video compression [8] and task assignment in distributed systems [9]. Reinforcement learning approach was utilised for QoS improvement [10] and a generalized reinforcement learning scheme for RaNN[11]. Interestingly, several RaNN-based deep learning models, such as the non-negative auto-encoders with simplified RaNN [12] and convolution RaNN [13], have been proposed in [14] to serve as tools for extracting high-level abstract features from raw data. One of those extensions are used as hybrid deep RaNN with sheep flock optimization approach to enhance video transmission quality[15]. Also, an accurate business management decision model based on deep RaNN with genetic algorithms[16].
Serrano [17] proposed a RaNN model to predict the price of fuel at time t with an input window of k previous input $[t-k, t-1]$. Such method works only on uni-variate time series and the window is fixed and cannot be changed unless we change the structure of the network. On the contrast, our proposed hybrid model, Random Simple Recurrent Network (RSRN), is designed to work on multi-variate time series with flexible windows. Most of the proposed RaNN models deal with discrete data achieving excellent results [5, 14, 6, 7, 8, 10, 9]. However, there has not been much research regarding time-series forecasting data using RaNN, hence the present research.

RaNN has two topologies: feed-forward topology for casual data and recurrent topology for sequential data. The problem with recurrent topology is that most of the effective learning algorithms depend on second order terms and the Hessian matrix approximation, making them computationally demanding. For this reason we will use the feed-forward topology with GD and adapt it to be able to process time-series data.

On the other hand, Simple Recurrent Network(SRN - also called Elman net) is a simple lightweight recurrent network with a context layer for capturing temporal dependencies. It is commonly used to forecast sequential data. The main reason for choosing SRN instead of another Recurrent NN variant is due to the fact that in RaNN, the probability of a neuron i firing a signal to itself is $P(i, i) = 0$. We overcome this issue by using a context layer that holds a copy of the hidden layer, this copy is used next iteration as temporal dependencies like SRN do. In the beginning of this work, we relied on SRN instead of LSTM and GRU nets as recurrent model to merge with RaNN because these nets are more demanding and combining either LSTM or GRU with RaNN is not straightforward due to their different logic.

The proposed RSRN model is a combination of feed-forward RaNN and SRN. We have a three layered feed-forward RaNN and add a context layer to the hidden layer as shown is Fig. 3. The output of the hidden layer will be copied to the context layer and used as temporal dependencies for the next step. To train such a network, we adapt Back Propagation (BP), Back Propagation Through Time (BPTT) and Truncated Back Propagation Through

Time (TBPTT). We solve the problem of gradient explosion by adopting in particular penalised GD, where the update of the weights uses the average of the gradient over time instead of the total gradient. This makes the learning process slower but stable. RSRN is able to process multi-variate sequential data with less trainable parameters than state-of-the-art models. RaNN has proved to be energy-efficient [14] than existing methods making the RSRN also energy-efficient.

In a nutshell, the paper presents an introduction of a new recurrent neural architecture, RSRN, that lightweight in terms of complexity leading to energy efficiency that supports its deployment on limited power platforms. It is capable to accommodate multi-variate data with flexible windows compared to the previous studies. Also, the adaptation of GD using the average gradient in back propagation through time (BPTT) and truncated BPTT (TBPTT) in training RSRN to prevent gradient explosion. Finally, a reduction of general calculation complexity of BPTT/TBPTT from $O(N^2/2)$ to $O(2N)$. A thorough evaluation on a number of datasets to show the performance of RSRN.

In the rest of this paper, we will provide a brief description of RaNN and SRN (Section 2). In Section 3, we describe the proposed RSRN model, covering both the architecture and the learning algorithms. In Section 4, we present and discuss the performance results of various experiments.

2 Overview of the neural models

2.1 Random NN

RaNN is a mathematical model derived from the spiking behaviour of neurons in the brain cortex and the queuing system making it the closest model to the neurological connectivity. The spiking behaviour appears in the signalling activity of the network. Even thought those spikes occur in stochastic random manner, studies shows that the probability distribution of each neuron comes from the sum of positive signal divided by the sum of negative signals(see Eq. 1). Another important study indicate that in steady state, the joint probability distribution of very large system is the multiplication of individual probabilities(cell or neurons)[18]. Those properties makes RaNN helpful in various applications such as cyber-security, optimal routing and tumor detection.
RaNN [19] is a feed-forward network (see Fig. 2) where neurons exchange positive and negative signals as shown in Figure 1. A Random Neuron (RN) is the main component of RaNN. It may receive a positive or negative signals from previous neurons. Each RN i has a positive parameter called potential k_i. It increases by 1 when a RN i receives a positive signal, $k_i \leftarrow k_i + 1$, and decreases by 1 when it receives a negative signal until it reaches 0, $k_i \leftarrow max(k_i - 1, 0)$. A RN i has a probability of sending a positive or negative signal to other neurons. Another parameter is the firing rate r_i, which controls the frequency of signal firing. Positive/Negative signals in RaNN are expressed as positive weights ($W^+/W^- \geq 0$). The stationary probability distribution ϱ (could be

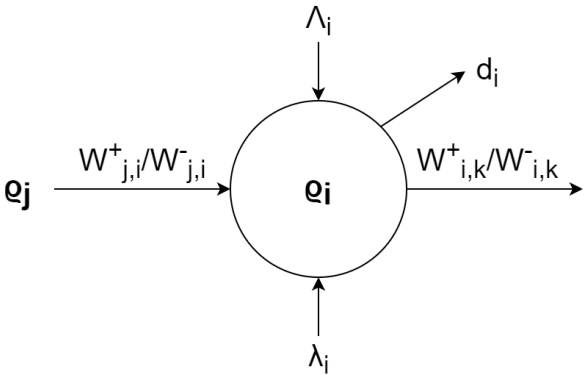


Fig. 1: Internal structure of a random neuron: Λ_j and λ_j are excitatory and inhibitory input signals and d_i is the outgoing signal to environment.

described as an activation function) of each neuron i is defined as follows:

$$\varrho_i = \frac{T_i^+}{r_i + T_i^-} = \frac{\Lambda_i + \sum_j^J \varrho_j w_{j,i}^+}{r_i + \lambda_i + \sum_j^J \varrho_j w_{j,i}^-} \tag{1}$$

where T_i^+ and T_i^- are the total arrival of positive/negative signals from the environment and other neurons. r_i is the firing rate, which is the total of the outgoing weights from neuron i ($r_i = \sum_j^J W_{ij}^+ + W_{ij}^-$). Λ_i and λ_i are the excitatory and inhibitory signals from the environment(expressed as input data), associated with neuron i. For each data input x_i:

$$\Lambda_i = \begin{cases} x_i & if x_i \geq 0 \\ 0 & otherwise \end{cases} \tag{2}$$

$$\lambda_i = \begin{cases} |x_i| & if x_i \leq 0 \\ 0 & otherwise \end{cases} \tag{3}$$

The quantities $w_{j,i}^+$ and $w_{j,i}^-$ are the positive and negative connection weights between neuron j and i. If the neurons keep firing positive signals, the next neurons will saturate [19]. To prevent such a saturation, a limit of firing is set as:

$$\varrho_i = min(\varrho_i, 1) \tag{4}$$

where $min(a, b)$ produces the minimum between the quantities a and b. In a feed-forward network with N neurons, an input layer with I nodes, a hidden layer with H nodes and an output layer with O nodes, the excitation probability of neurons in each layer can be calculated using Eqs. 1 and 4. To train this network, connection pairs ($w_{u,v}^+$, $w_{u,v}^-$) of nodes u and v are updated using GD rule, which update the weight by computing the gradient of the loss function

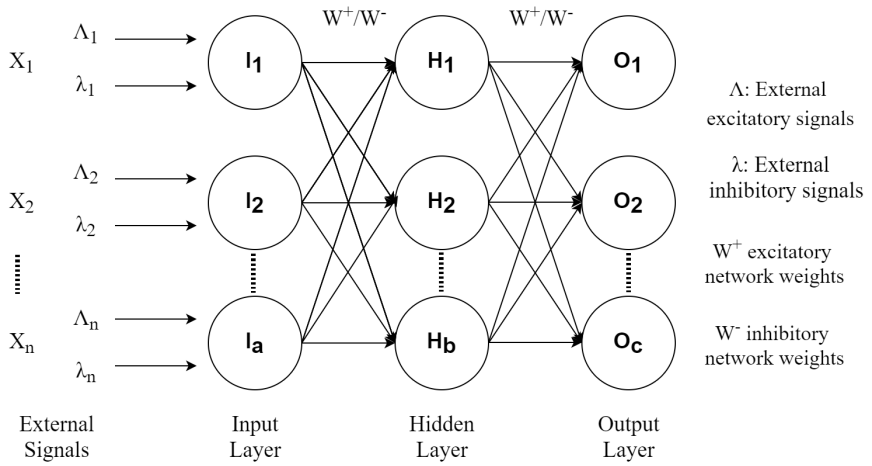


Fig. 2: The structure of a feed-forward RaNN: I_a, H_b and O_c are input, hidden and output nodes respectively.

with respect to the weight and subtract it from the old value of the weight with a controlled step:

$$w_{u,v}^{t} = w_{u,v}^{t-1} - \eta \frac{\partial E}{\partial w_{u,v}^{t-1}} \tag{5}$$

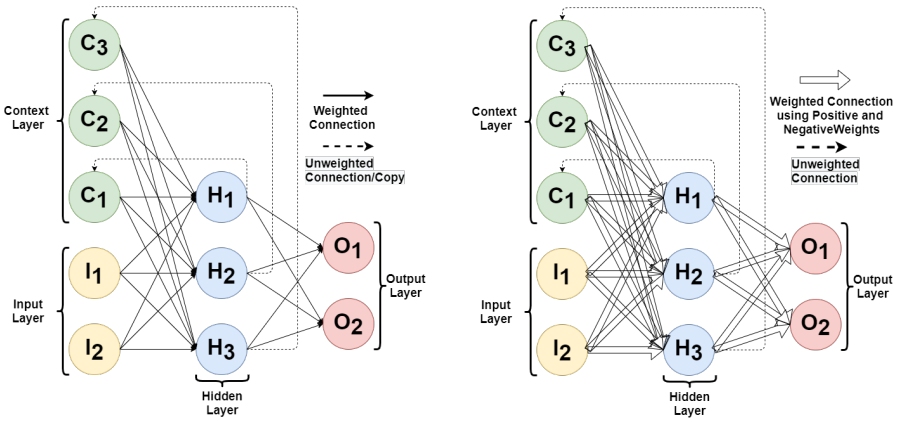
where $w_{u,v}^{t-1}$ denotes the previous value of the connection between u and v, E is the loss function and η is the learning rate. In this paper, we will use the Mean Squared Error (MSE) as a loss function E, which is the average quadratic difference between the network output and the desired results:

$$E = \frac{1}{2} \sum^{O} (\varrho_o - d_o)^2 \tag{6}$$

where ϱ_o and d_o are the computed output and the desired output of node o in the output layer. By applying Eq. 5 and Eq. 6, we get the following GD update:

$$\begin{aligned} \frac{\partial E}{\partial w_{h,o}^{+}} &= \frac{\partial E}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial w_{h,o}^{+}} \\ &= \sum_{o}^{O} (\varrho_o - y) \frac{\varrho_h}{r_o + \sum_h^H \varrho_h w_{h,o}^{-}} \end{aligned} \tag{7}$$

$$\begin{aligned} \frac{\partial E}{\partial w_{h,o}^{-}} &= \frac{\partial E}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial w_{h,o}^{-}} \\ &= \sum_{o}^{O} (\varrho_o - y) \frac{-\varrho_h \varrho_o}{r_o + \sum_h^H \varrho_h w_{h,o}^{-}} \end{aligned} \tag{8}$$



(a) Simple Recurrent Network (b) Random Simple Recurrent Network

Fig. 3: From to SRN to RSRN

$$\begin{aligned}\frac{\partial E}{\partial w^{+}_{i,h}} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w^{+}_{i,h}} \\ &= \sum_{o}^{O}(\varrho_o - y)\frac{w^{+}_{h,o} - w^{-}_{h,o}\varrho_o}{(r_o + \sum_{h}^{H}\varrho_h w^{-}_{h,o})}\frac{\varrho_i}{(r_h + \sum_{i}^{I}\varrho_i w^{-}_{i,h})}\end{aligned} \tag{9}$$

$$\begin{aligned}\frac{\partial E}{\partial w^{-}_{i,h}} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w^{-}_{i,h}} \\ &= \sum_{o}^{O}(\varrho_o - y)\frac{w^{+}_{h,o} - w^{-}_{h,o}\varrho_o}{(r_o + \sum_{h}^{H}\varrho_h w^{-}_{h,o})}\frac{-\varrho_i\varrho_h}{(r_h + \sum_{i}^{I}\varrho_i w^{-}_{i,h})}\end{aligned} \tag{10}$$

The update of the weights is done through Eq. 7 to Eq. 10.

2.2 Simple Recurrent Network (SRN)

Elman Network, known as Simple Recurrent Network, is a recurrent neural network [20], encompassing a context layer that captures the temporal/order dependencies in sequential data. Figure 3a shows the structure of SRN for processing discrete-time sequences. As shown in the figure, SRN is a simple network with additional layer(context layer) to add a recurrent behaviour. This layer serves as pocket holding past memory/information for next sequence. As we can see, we have an input layer I_x, a hidden layer H_x and an output layer O_x. The context layer C_x receives a copy of the hidden layer information(unweighted connection) and resend it to the hidden layer H_x next iteration with weighted connection. SRN considered as unrolled version of recurrent neural network where the context layer does not exist.

During training, at each time step t, when an input is presented, a feed-forward pass to the output layer and a copy of hidden layer's firing values is

saved in the context layer and used in the next time step $t+1$. The calculation of this process is as follows:

$$h_t = \sigma(W_{ih}x_i + W_{ch}h_{t-1} + b_h) \tag{11}$$

$$o_t = \sigma(W_{ho}h_t + b_o) \tag{12}$$

where W_{ij} is the connection weight between neuron i and j. The index c refers to the context nodes, b_h, b_o are the biases of the hidden and output layers and σ is an activation function. SRN can be trained using the standard back-propagation (BP) [21, 22] using the update rule expressed by Eq. 5.

3 Proposed Random Simple Recurrent Network (RSRN)

The purpose of this paper is to build an efficient light-weight recurrent model to process time-series data. This process is power consuming and computationally expensive using state-of-art models such as LSTM and GRU. Hence, RSRN is an energy-efficient model that is able to solve time-series problems with less complexity. This will allow the deployment of RSRN in a power limited devices and achieving better results with minimum consumed energy. The proposed RSRN brings Elman's SRN and Gelenbe's RaNN together. RSRN combines the architecture of SRN with the connectivity and learning properties of RaNN, resulting in a new neural architecture. In order to develop this model, we tweak the RaNN's architecture in the form of SRN to preserve RaNN's properties and add the recurrence capability. We use the SRN structure instead of a general recurrent network because we cannot add internal loop explicitly. This is due to the fact that internal loops in RaNN $p(i,i) = 0$ are disallowed [19]. SRN has a context layer that serves as an internal loop implicitly and saves the temporal dependencies (see Fig. 3b). To build the proposed model RSRN, we consider a 3-layer SRN that has a context layer and augment the individual weight connections with the random behaviour to result in pairs of positive and negative signals or weights as shown in Figure 3b. The context layer behaves like the input layer but receives its input from the previous time step. RSRN's probability distribution of the nodes is derived by combining the probability distribution equation (Eq. 1) and the activation function of SRN (Eq. 11) to result in following :

$$\varrho_i = \frac{T_i^+}{r_i + T_i^-} = \frac{\Lambda_i + \sum_{j=1}^{J} \varrho_j w_{j,i}^+ + \sum_{c=1}^{C} \varrho_c w_{c,i}^+}{r_i + \lambda_i + \sum_{j=1}^{J} \varrho_j w_{j,i}^- + \sum_{c=1}^{C} \varrho_c w_{c,i}^-} \tag{13}$$

$$\varrho_i = min(\varrho_i, 1) \tag{14}$$

where T_i^+ and T_i^- are the total arrival of positive and negative signals from the environment and the previous neurons (input and context layers). r_i is the firing rate calculated as the total of the outgoing weights from neuron

i to other neurons S, $r_i = \sum_s^S (W_{i,s}^+ + W_{i,s}^-)$. Λ_i and λ_i are the excitatory and inhibitory signal from the environment. $w_{j,i}^+$ and $w_{j,i}^-$ are the positive and negative connection weights between neuron j and i. $w_{c,i}^+$ and $w_{c,i}^-$ are the positive and negative connection weights between nodes in the context layer (denoted as index $c = 1 \dots C$) and other nodes ($i = 1 \dots I$). A limit for probability distribution is set to prevent it from saturating the next neurons.

This change in the structure will require a learning algorithm that is adapted to RSRN. An offline GD-based algorithm is used to train the network. Here, we use BP, BPTT and TBPTT. The last two algorithms are used in recurrent networks calculating the gradient loss with respect to each time step. If we consider the context layer as part of the input layer and use the BP algorithm, we can apply the same update rule of RaNN while adding connections between the hidden layer and the context layer W_{ch}. Specifically, we apply Eq. 5 and Eq. 6 to update the positive and negative weights. The connections W_{ho}^+ and W_{ho}^- are updated using the Eqs.7 and 8 respectively. The connection $W_{ih}^+, W_{ih}^-, W_{ch}^+$ and W_{ch}^- can be concluded from Eqs.9 and 10 and taking the probability distribution of Eq. 13 into consideration as follows:

$$\begin{aligned}\frac{\partial E}{\partial w_{i,h}^+} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w_{i,h}^+} \\ &= \sum_o^O (\varrho_o - y)\frac{w_{h,o}^+ - w_{h,o}^- \varrho_o}{(r_o + \sum_h^H \varrho_h w_{h,o}^+)}\frac{\varrho_i}{(r_h + \sum_i^I \varrho_i w_{i,h}^- + \sum_c^C \varrho_c w_{c,h}^-)}\end{aligned} \tag{15}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{i,h}^-} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w_{i,h}^-} \\ &= \sum_o^O (\varrho_o - y)\frac{w_{h,o}^+ - w_{h,o}^- \varrho_o}{(r_o + \sum_h^H \varrho_h w_{h,o}^+)}\frac{-\varrho_i \varrho_h}{(r_h + \sum_i^I \varrho_i w_{i,h}^- + \sum_c^C \varrho_c w_{c,h}^-)}\end{aligned} \tag{16}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{c,h}^+} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w_{c,h}^+} \\ &= \sum_o^O (\varrho_o - y)\frac{w_{h,o}^+ - w_{h,o}^- \varrho_o}{(r_o + \sum_h^H \varrho_h w_{h,o}^+)}\frac{\varrho_c}{(r_h + \sum_i^I \varrho_i w_{i,h}^- + \sum_c^C \varrho_c w_{c,h}^-)}\end{aligned} \tag{17}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{c,h}^-} &= \frac{\partial E}{\partial \varrho_o}\frac{\partial \varrho_o}{\partial \varrho_h}\frac{\partial \varrho_h}{\partial w_{c,h}^-} \\ &= \sum_o^O (\varrho_o - y)\frac{w_{h,o}^+ - w_{h,o}^- \varrho_o}{(r_o + \sum_h^H \varrho_h w_{h,o}^+)}\frac{-\varrho_c \varrho_h}{(r_h + \sum_i^I \varrho_i w_{i,h}^- + \sum_c^C \varrho_c w_{c,h}^-)}\end{aligned} \tag{18}$$

where $\varrho_c = \varrho_h^{t-1}$.

For a start, we consider a forecasting case. Given a sequential dataset: $\mathbf{u}_t = \{u_0, u_1, \dots, u_T\}$ where the outcome of step $t+1$ results from the previous time step t. RSRN will try to map $f(u_t) \rightarrow u_{t+1}$. The training process for the adapted GD-based back propagation algorithm is fully described in Alg. 1.

Data is normalised in the unit interval $[0, 1]$ because the output of the network is bounded between 0 and 1. The weights are kept positive after each update.

Algorithm 1 Training RSRN using BP

Inputs: Dataset $\mathbf{u}_t = \{u_0, u_1, \ldots, u_T\}$, learning rate η
Updated parameters: $W^+_{ih}, W^-_{ih}, W^+_{ho}, W^-_{ho}, W^+_{ch}, W^-_{ch}$
Normalize the data u_t between 0 and 1
Initialize weights $W^+, W^- > 0$
while not stopping criterion **do**
  Calculate r_i for all $i \in I, H$ and O
  for all time-step t in $[0, 1, \ldots, T-1]$ **do**
    $\Lambda = u_t, y = u_{t+1}$
    Calculate ϱ_i using Eq.13 for $i \in [I, H, O]$
    Calculate error $E_t(\varrho_o, y)$ using Eq.6
    $\varrho_c = \varrho_h$
  end for
  Calculate the gradient of W^+, W^- using Eqs.7, 8, 15, 16, 17 and 18
  Update the weight using Eq.5
  $W = max(0, W)$
end while

penalised BPTT and Truncated BPTT

These algorithms are adapted versions of back propagation used in recurrent networks. This algorithm unrolls the network and back propagates the gradient error through the previous time steps. BPTT [23] calculates the gradient error with respect to each time step from T back to 0. In the following, we reformulate the previous equations to obtain BPTT.
Likewise, we apply the Truncated BPTT which behaves the same way as BPTT but does not roll back to beginning of the series. Instead, we divide the dataset into batches of k instances and treat each batch k as separate dataset. Then, we feed the state of batch k to the next batch $k+1$. Algorithm 2 describes the full steps of BPTT (and TBPTT) algorithm. During the training process of BPTT, the gradient of the current time-step t is dependent on the previous one $t-1$. Accordingly, we employ the dynamic programming concept to optimize the training process. In normal cases, the calculation of the gradient using BPTT occur at the end of sequence, which unrolls the data to recalculate every gradient in the sequence whereas for our case, the gradient is calculated during the feed-forward process therefore α and β(for the weights $W^{+/-}_{ih}$ and $W^{+/-}_{ch}$ respectively). Those variables preserve the temporal dependencies from the beginning of the series $t = 0$ to the current time step T by adding the gradient each time the data is presented, hence we do not need to calculate it at the end of sequence. The use of the latter variables reduces the complexity of

Algorithm 2 BPTT algorithm for RSRN model

Inputs: Dataset $\mathbf{u}_t = \{u_0, u_1, \ldots, u_T\}$, learning rate η
Updated parameters: $W^+_{ih}, W^-_{ih}, W^+_{ho}, W^-_{ho}, W^+_{ch}, W^-_{ch}$
Normalize the data u_t between 0 and 1
Initialize weights $W^+, W^- > 0$
while not stopping criterion **do**
  $\alpha^+, \alpha^-, \beta^+, \beta^-, \varrho_c = 0$
  // For TBPTT we add a loop of batches here
  for all time-step t in [0,1,...,T] **do**
    Calculate r_i for all $i \in I, H$ and O
    $\Lambda = u_t, y = u_{t+1}$
    Calculate ϱ_i using Eq.13 for $i \in [I, H, O]$
    Calculate error $E(\varrho_o, y)$ using Eq.6
    Calculate the gradient of $W^+_{h,o}, W^-_{h,o}$ using Eqs.7, 8
    Calculate terms $\frac{\partial E}{\partial \varrho_{h^t}}, \frac{\partial \varrho_{h^t}}{\partial W^+_{i,h}}, \frac{\partial \varrho_{h^t}}{\partial W^+_{c,h}}, \frac{\partial \varrho_{h^t}}{\partial W^-_{i,h}}, \frac{\partial \varrho_{h^t}}{\partial W^-_{c,h}}, \frac{\partial \varrho_{h^t}}{\partial \varrho_{c^t}}$
    $\alpha^+ = \frac{\partial \varrho_{h^t}}{\partial W^+_{i,h}} + (\frac{\partial \varrho_{h^t}}{\partial \varrho_{c^t}} \alpha^+), \frac{\partial E}{W^+_{i,h}} = \frac{\partial E}{\partial \varrho_{h^t}} \alpha^+$
    $\alpha^- = \frac{\partial \varrho_{h^t}}{\partial W^-_{i,h}} + (\frac{\partial \varrho_{h^t}}{\partial \varrho_{c^t}} \alpha^-), \frac{\partial E}{W^-_{i,h}} = \frac{\partial E}{\partial \varrho_{h^t}} \alpha^-$
    $\beta^+ = \frac{\partial \varrho_{h^t}}{\partial W^+_{c,h}} + (\frac{\partial \varrho_{h^t}}{\partial \varrho_{c^t}} \beta^+), \frac{\partial E}{W^+_{c,h}} = \frac{\partial E}{\partial \varrho_{h^t}} \beta^+$
    $\beta^- = \frac{\partial \varrho_{h^t}}{\partial W^-_{c,h}} + (\frac{\partial \varrho_{h^t}}{\partial \varrho_{c^t}} \beta^-), \frac{\partial E}{W^-_{c,h}} = \frac{\partial E}{\partial \varrho_{h^t}} \beta^-$
    $\varrho_c = \varrho_h$
  end for
  Update the weight using Eq.24
  $W = max(0, W)$
end while

BPTT (and TBPTT) algorithm from $O(\frac{N^2}{2})$ to $O(2N)$, where N is the total number of time-steps.

$$\frac{\partial E_T}{\partial W^+_{i,h}} = \sum_{t=0}^{T} \frac{\partial E_T}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial \varrho_{h^T}} \frac{\partial \varrho_{h^T}}{\partial \varrho_{h^t}} \frac{\partial \varrho_{h^t}}{\partial W^+_{i,h}} \tag{19}$$

$$\frac{\partial E_T}{\partial W^-_{i,h}} = \sum_{t=0}^{T} \frac{\partial E_T}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial \varrho_{h^T}} \frac{\partial \varrho_{h^T}}{\partial \varrho_{h^t}} \frac{\partial \varrho_{h^t}}{\partial W^-_{i,h}} \tag{20}$$

$$\frac{\partial E_T}{\partial W^+_{c,h}} = \sum_{t=0}^{T} \frac{\partial E_T}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial \varrho_{h^T}} \frac{\partial \varrho_{h^T}}{\partial \varrho_{h^t}} \frac{\partial \varrho_{h^t}}{\partial W^+_{c,h}} \tag{21}$$

$$\frac{\partial E_T}{\partial W^-_{c,h}} = \sum_{t=0}^{T} \frac{\partial E_T}{\partial \varrho_o} \frac{\partial \varrho_o}{\partial \varrho_{h^T}} \frac{\partial \varrho_{h^T}}{\partial \varrho_{h^t}} \frac{\partial \varrho_{h^t}}{\partial W^-_{c,h}} \tag{22}$$

$\frac{\partial \varrho_{h^T}}{\partial \varrho_{h^t}}$ is the partial derivative of the hidden layer at the current time T with respect to its previous states t. We can get the previous formulas using the

following equation:

$$\begin{aligned}\frac{\partial \varrho_{h^t}}{\partial \varrho_{h^{t-1}}} &= \frac{\partial \varrho_{h^t}}{\partial \varrho_c} \\ &= \frac{W^+_{c,h} - W^-_{c,h}\varrho_h}{(r_h + \sum_i^I \varrho_i W^-_{i,h} + \sum_c^C \varrho_c W^-_{c,h})}\end{aligned} \tag{23}$$

The problem with BPTT (and some situations with TBPTT) is that it tends to reach infinite and large values (gradient explosion) due to the additive process of multiple gradient. To solve this issue, we use the average of the gradient total over time to limit the gradient from reaching to infinite or large values. Therefore, instead of using Eq. 5 as an update rule, we use the following equation to tackle the explosion problem:

$$w^t_{u,v} = w^{t-1}_{u,v} - \eta \frac{1}{T} \frac{\partial E}{\partial w^{t-1}_{u,v}} \tag{24}$$

where T is the length of the sequence. This update rule makes the learning process slower than the original one, but it is stable and less prone to the weight explosion problem. The term $\frac{1}{T}$ could be considered as part of the learning rate η, where $(T > 1)$ all the time which makes $\frac{1}{T} < 1$, hence minimize the learning rate η and made the learning process slow compared to the update rule in Eqs.5. Apart from this, the term $\frac{1}{T}$ will divide the total gradient and prevent it from reaching to large values.

As we can conclude from the training process, Algo. 1 is the standard version of GD hence less complex and fast during training so it is suitable for small sequences data and where the training time is crucial factor. Meanwhile, Algo. 2 is an adapted and optimized version of BPTT strategy but it is more complex and memory dependent. This algorithm is designed for longer sequences of data with downside of more training time. Both algorithms are less complex than second order approaches designed to train RaNN such as the Gauss-Newton based methods and Levenberg-Marquardt algorithm [24].

4 Evaluation

To evaluate the proposed neural architecture, RSRN, a number of time-series datasets are used. Moreover, a comparative study is presented involving some state-of-the-art recurrent NNs such as LSTM and GRU. Before delving into the experiments, we introduce the three datasets used: Britain's power grid sources [25], Global Land Temperature [26] and Weekly fuel prices [27]. The Britain's power grid dataset consists of 10 features and covers 196453 time-step. This dataset describes the output electricity power generated by each power source like coal, wind and nuclear as well as the demand and supply changes from 2012 to 2020. The Global Land Temperature dataset, known as Global Climate Change Data contains monthly recorded land and ocean

temperatures and other measurements about climate change from 1750 to 2015. It consists of 3180 samples and 8 features. The third dataset, weekly fuel prices, describes the price of the fuel (petrol and diesel) each week and their changes since 2013. It has 881 samples and 10 features.

In all experiments, these datasets are split into training (75%) and testing (25%) sets. Note that RSRN was implemented and trained on a laptop (i5-8300H 2.30 GHz, 8GB of RAM, GTX 1050 4GB) and Kaggle platform (cloud computing system for training deep-learning models). During our experiments, we found out that the large datasets prefer small learning rate, so we set the learning rate η to 10^{-2}, 10^{-3} and 10^{-4} for the fuel, temperature and GridWatch datasets respectively. The models were trained on 100 epoch, this choice is due to the fact that generally the performance is stable after 100-th epoch for all models and increasing the number of epochs will yield the same results. While for the batch-size of data in TBPTT method, the empirical experiments proved that the batch size of 100 is the best choice for these data.

4.1 Evaluation of the RSRN learning algorithms

The goal of this experiment is to evaluate the proposed training algorithms: BP, BPTT and TBPTT for different datasets. Tables 1 and 2 show the MSE testing results and the training time (in seconds) by setting the number of hidden neurons to various values using different training algorithms. The simulations show that the best results are obtained by TBPTT, highlighting its ability to cope better with the temporal dependencies in the data without suffering the gradient explosion problem. On the other hand, BP achieves lower performance than that of BPTT and TBPTT. A general observation indicate that the performance (MSE) of TBPTT is twice than BPTT/BP, which proves the effectiveness of TBPTT in training this model. We notice that the best results are achieved by the model having the smallest number of neurons in the hidden layer (in this case 5 neurons). It seems that RSRN with less neurons performs better for these dataset recall that each neuron has two weights and increasing the number of neurons in hidden layer potentially leading to overfitting.

In terms of computational time, Tab. 2 shows that the training time increases as the size of the hidden layer increases. Also, BP has the lowest training time by far compared to BPTT and TBPTT as expected, because in the updating process we consider only the last gradient, whereas TBPTT seems a bit slower than BPTT due to the weight update at each batch k while this happens once in BPTT. In conclusion, for best accuracy we recommend TBPTT at the cost of more training time, whereas BP offers an acceptable performance in short period if the training time were a crucial factor. When building an RSRN model, it is advisable to choose the lesser complex model as it saves time while obtaining good results.

Table 1: Mean Squared Error (MSE) of RSRN model under different datasets and configurations, H is the number of neurons in hidden layer

Dataset	Fuel			Temp			GridWatch		
Configurations	BP	BPTT	TBPTT	BP	BPTT	TBPTT	BP	BPTT	TBPTT
H = 5	0,087	0,106	**0,036**	0,114	0,104	**0,059**	0,103	0,096	**0,029**
H = 10	0,119	0,115	**0,037**	0,122	0,121	**0,079**	0,109	0,117	**0,066**
H = 15	0,17	0,127	**0,049**	0,97	0,222	**0,109**	0,118	0,12	**0,075**

Table 2: Training time (in seconds) of RSRN using different datasets and configurations, H is the number of hidden neurons.

Dataset	Fuel			Temp			GridWatch		
Configurations	BP	BPTT	TBPTT	BP	BPTT	TBPTT	BP	BPTT	TBPTT
H = 5	1,32	6,26	7,33	5,18	25,07	25,21	348	1482	1586
H = 10	1,7	6,47	7,35	6,53	25,79	25,5	413	1613	2001
H = 15	1,96	7,95	7,95	8,26	28,83	27,46	495	3426	3987

4.2 Comparative study

In this section, we aim to validate our model against state-of-art recurrent models. The goal of this study is to highlight its properties and address its limitations. Based on empirical results and general observation as far as we know, we assembled the Table 3. This tables describe the performance of LSTM/-GRU and RSRN from different perspective. As shown in Tab. 3, the criteria

Table 3: model comparison according to multiple criteria

Criteria	classification	forecasting	framework	training time	energy consumption	mobility usage
LSTM/GRU	high	high	high	slow	high	low
RSRN	low	high	low	fast	low	high

presented are the classification and forecasting performance, framework availability, training time, energy consumption and mobility usage(availability for limited-power devices). We notice that LSTM/GRU has great performance and there is plenty of dedicated libraries while it suffer from slow training time, high energy consumption and low mobility usage for its heavy architecture. On the contrary, RSRN covers the limitation of LSTM/GRU with fast training time, low energy consumption and high mobility usage with drawback of misclassification and lack of reusable code.

On this part, we focus on empirical results. In this experiment, RSRN, trained with TBPTT, is compared against well known recurrent models such as LSTM and GRU using the Kaggle platform. In each model, the hidden layer consists of 5 neurons. For the sake of comparison, we use the same structure in each model even thought that each model has its own optimal structure/mechanism. The purpose of this experiment is to determine the performance and training time of RSRN against other models under the same conditions and

parameters. We trained the three models over 100 epoch and measured the MSE and training time (in seconds).

Table 4 shows the MSE testing results and the training time on three datasets. The major observation is that RSRN outperforms LSTM and GRU in the case of small datasets, Fuel and Temperature with a wide gap. On the other hand, RSRN achieved better and comparable performance on the GridWatch dataset against LSTM/GRU. This indicate that RSRN is able to extract meaningful patterns faster from relatively small sequences, while this performance decreases a little bit for large data when compared against other models. It appears that this is due to forgetting of temporal dependencies over longer sequences. In terms of training time, RSRN requires less training time compared to the other models, this is due to the use of less trainable parameters than other models (see table 5) and the optimization of the training process. As presented in table 5, RSRN has less weights than LSTM and GRU which mean less arithmetic operations, hence RSRN is able to perform the same task with less consumed energy than other models. This shows in particular the energy-efficient lightweight nature of RSRN.

Specifically, we notice that RSRN performs better than LSTM and GRU on small datasets while it acquires comparable results against them on large datasets with less training time and fast convergence.

Table 4: RSRN vs. LSTM and GRU: MSE results and training time (seconds)

Dataset (# of time-steps)	MSE			Time(seconds)		
	LSTM	GRU	RSRN	LSTM	GRU	RSRN
Fuel(881)	0.0874	0.0931	**0.0338**	18	30	**5**
Temp(3180)	0.0696	0.0692	**0.0339**	40	78	**19**
GridWatch(196453)	0.0272	0.0282	**0.0232**	2295	5081	**1183**

Table 5: Number of trainable parameters(weights) in each model

Dataset (# of features)	# of trainable parameters		
	LSTM	GRU	RSRN
Fuel/GridWatch(10)	400	335	**250**
Temp(8)	344	289	**210**

4.3 Multi-Layer RSRN

In this section, we investigate the development of a multi-layer RSRN architecture. We compare four RSRN networks with different number of parameters (neurons) and number of hidden layers using MSE as a performance measure. The first two models correspond to a one-hidden layer RSRN model with 5

(called RSRN-1-5) and 10 hidden neurons (RSRN-1-10) respectively. The second two models correspond to a two-hidden layer RSRN model with 5 and 10 neurons respectively in each hidden layer (called RSRN-2-5 and RSRN-2-10 respectively). These architectures are selected based on observations of the last two experiments. The four models are trained using 100 epochs and tested on the three datasets.

From Tab. 6 it appears that RSRN-1-5 achieves the best result on the Fuel and GridWatch datasets, while RSRN-2-5 performs best on the Temperature dataset. Temp dataset is more complex because each feature of the data has a direct relationship with other features and has an impact on each others (strong correlation), while the other two datasets each entry is an individual data (weak correlation); Thus, temperature dataset requires more complex model (more layers) to extract meaningful features.

If we compare RSRN-1-10 against RSRN-2-5 (having the same number of parameters, but with different architectures), RSRN-2-5 performs slightly better than RSRN-1-10. Thus, using the optimal structure with multiple layer is better for extracting higher level representation of the time-series. The last model RSRN-2-10 has the lowest performance, likely because it has a sub-optimal structure for these data and tend to overfit.

In conclusion, when dealing with data having a weak correlation between features we recommend the use of single layer; whereas for strong correlated data, one could choose a multi-layer RSRN architecture with small number of parameters.

Table 6: MSE results using the four RSRN models

ID#	Configuration	Fuel	Temp	GridWatch
RSRN-1-5	$H = 5$	**0,0443**	0,0994	**0,027**
RSRN-1-10	$H = 10$	0,0459	0,1044	0,1111
RSRN-2-5	$H_1 = 5, H_2 = 5$	0,0445	**0,0954**	0,1045
RSRN-2-10	$H_1 = 10, H_2 = 10$	0,063	0,1295	0,1229

5 Conclusion

In this paper, we proposed a light-weight energy-efficient recurrent network, that is able to solve time-series problems. It combines Gelenbe's RaNN behaviour and Elman's SRN architecture to devise the proposed recurrent network, RSRN. The model is able to process multi-variate sequential data fast and accurately. To train such a model, three offline learning algorithms were adapted and applied, namely BP, BPTT and TBPTT. The latter two approaches were penalised to prevent gradient explosion problem by using the average of total gradient over time. The empirical scores showed that TBPTT produces the best results compared to BPTT and BP as well as some competitors like LSTM and GRU. Another result is that the tested data prefer the RSRN model with less neurons to achieve better inference. The paper

investigates also the multi-layer RSRN architecture showing that for highly dimensional and correlational data it prefers a multi-layer design that may leads to better results. Even though RSRN had a great performance in forecasting data; it performs poorly on time-series classification. This results came from an implicit experiments done earlier to test its ability in categorising sequential data such as activity recognition from body sensors. Interestingly, RSRN is an energy-efficient model that offers a competitive performance with less computational time compared to the stat-of-the-art models making it suitable for devices with energy constraints. RSRN is adequate for proactive models like traffic monitoring, where it will predict future packet flow and analyse it for potential threats to take a precautious actions.
In future work, we consider optimizing RSRN with well-known optimizers such as Adam and RMSProp, applying regularization to avoid over-fitting and incorporating it with deep learning tools to broaden its application potential. RSRN was designed to work in offline mode, an online approach is considered for next work as daily data arrives. Finally, we aim to design a suitable framework for ease of accessibility and reusability.

Declarations

All authors declare that they have no conflicts of interest.

Data availability

The datasets generated during and/or analysed during the current study are available publicly in the "data.world" repository.

- Britain's power grid sources dataset is available at https://data.world/makeovermonday/2019w32
- Global Land Temperature dataset is available at https://data.world/data-society/global-climate-change-data
- Weekly fuel prices dataset is available at https://data.world/makeovermonday/2020w17-weekly-road-fuel-prices

References

[1] Gelenbe, E. Random neural networks with negative and positive signals and product form solution. *Neural computation* **1** (4), 502–510 (1989) .

[2] Gelenbe, E. G-networks: a unifying model for neural and queueing networks. *Annals of Operations Research* **48** (5), 433–461 (1994) .

[3] Gelenbe, E. A probabilistic generalization of the sat problem. *COMPTES RENDUS DE L ACADEMIE DES SCIENCES SERIE I-MATHEMATIQUE* **315** (3), 339–342 (1992) .

[4] Gelenbe, E., Koubi, V. & Pekergin, F. *Dynamical random neural network approach to the traveling salesman problem*, Vol. 2, 630–635 (IEEE, 1993).

[5] Timotheou, S. The random neural network: a survey. *The computer journal* **53** (3), 251–267 (2010) .

[6] Fröhlich, P. & Gelenbe, E. *Optimal fog services placement in sdn iot network using random neural networks and cognitive network map*, 78–89 (Springer, 2020).

[7] Öke, G. & Loukas, G. A denial of service detector based on maximum likelihood detection and the random neural network. *The Computer Journal* **50** (6), 717–727 (2007). https://doi.org/10.1093/comjnl/bxm066 .

[8] Gelenbe, E., Sungur, M., Cramer, C. & Gelenbe, P. Traffic and video quality with adaptive neural compression. *Multimedia Systems* **4** (6), 357–369 (1996) .

[9] Aguilar, J. & Gelenbe, E. Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences* **97** (1-2), 199–219 (1997) .

[10] Gelenbe, E. Steps toward self-aware networks. *Communications of the ACM* **52** (7), 66–75 (2009) .

[11] Lent, R. A generalized reinforcement learning scheme for random neural networks. *Neural Computing and Applications* **31** (7), 2699–2716 (2019) .

[12] Yin, Y. & Gelenbe, E. *Non-negative autoencoder with simplified random neural network*, 1–6 (IEEE, 2019).

[13] Yin, Y. & Gelenbe, E. *Single-cell based random neural network for deep learning*, 86–93 (IEEE, 2017).

[14] Yin, Y. Deep learning with the random neural network and its applications. *arXiv preprint arXiv:1810.08653* (2018) .

[15] Benisha, R. An efficient sheep flock optimization-based hybrid deep rann for secure and enhanced video transmission quality. *Neural Computing and Applications* 1–16 (2023) .

[16] Serrano, W. Genetic and deep learning clusters based on neural networks for management decision structures. *Neural Computing and Applications* **32** (9), 4187–4211 (2020) .

[17] Serrano, W. The random neural network in price predictions. *Neural Computing and Applications* 1–19 (2022) .

[18] Science, S. Random neural networks. URL https://serious-science.org/random-neural-networks-9974.

[19] Gelenbe, E. Stability of the random neural network model. *Neural computation* **2** (2), 239–247 (1990) .

[20] Elman, J. L. Finding structure in time. *Cognitive science* **14** (2), 179–211 (1990) .

[21] Sotirov, S. *Modelling the backpropagation algorithm of the elman neural network by a generalized net*, 49–55 (2012).

[22] Ren, G., Cao, Y., Wen, S., Huang, T. & Zeng, Z. A modified elman neural network with a new learning rate scheme. *Neurocomputing* **286**, 11–18 (2018) .

[23] Hermans, M. & Schrauwen, B. Training and analysing deep recurrent neural networks. *Advances in neural information processing systems* **26** (2013) .

[24] Basterrech, S. & Rubino, G. Random neural network model for supervised

learning problems. *Neural Network World* **25** (5), 457 (2015) .

[25] Kommenda, N. Britain's power sources (2021). URL https://data.world/makeovermonday/2019w32.

[26] berkeleyearth.org. Global climate change data (2021). URL https://data.world/data-society/global-climate-change-data.

[27] Department for Business, E. . I. S. Weekly road fuel prices (2021). URL https://data.world/makeovermonday/2020w17-weekly-road-fuel-prices.