Differential radial basis function network for sequence modelling

Kojo Sarfo Gyamfi, James Brusey and Elena Gaura

DOI 10.1016/j.eswa.2021.115982
ISSN 0957-4174

Publisher: Elsevier

Expert Systems With Applications 189 (2022) 115982

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Differential radial basis function network for sequence modelling

Kojo Sarfo Gyamfi *, James Brusey, Elena Gaura

School of Computing, Electronics and Mathematics, Coventry University, CV1 5FB, Coventry, United Kingdom

ARTICLE INFO

Keywords:
Radial basis function
Neural network
Sequence modelling

ABSTRACT

We propose a differential radial basis function (RBF) network termed RBF-DiffNet—whose hidden layer blocks are partial differential equations (PDEs) linear in terms of the RBF—to make the baseline RBF network robust to noise in sequential data. Assuming that the sequential data derives from the discretisation of the solution to an underlying PDE, the differential RBF network learns constant linear coefficients of the PDE, consequently regularising the RBF network by following modified backward-Euler updates. We experimentally validate the differential RBF network on the logistic map chaotic timeseries as well as on 30 real-world timeseries provided by Walmart in the M5 forecasting competition. The proposed model is compared with the normalised and unnormalised RBF networks, ARIMA, and ensembles of multilayer perceptrons (MLPs) and recurrent networks with long short-term memory (LSTM) blocks. From the experimental results, RBF-DiffNet consistently shows a marked reduction in the prediction error over the baseline RBF network (e.g., 41% reduction in the root mean squared scaled error on the M5 dataset, and 53% reduction in the mean absolute error on the logistic map); RBF-DiffNet also shows a comparable performance to the LSTM ensemble but requires 99% less computational time. Our proposed network consequently enables more accurate predictions—in the presence of observational noise—in sequence modelling tasks such as timeseries forecasting that leverage the model interpretability, fast training, and function approximation properties of the RBF network.

1. Introduction

The radial basis function network (RBFN) is an artificial neural network first introduced in the 1980s (Broomhead & Lowe, 1988) but still very much in vogue now (Dey et al., 2019; Masnadi-Shirazi & Subramaniam, 2020; Que & Belkin, 2019; Teng, 2018) due to its robustness as a universal function approximator. Architecturally, it is a three-layer network having one input layer, one hidden layer, and one output layer, as shown in Fig. 1, with activation units in the hidden layer made up of radial basis functions (RBFs).

The input layer is often connected to the hidden layer via direct connections whose weights are frozen at unity and thus not trainable, while the output layer is a simple linear layer. Though simple (compared to more complex architectures (Kaviani & Sohn, 2021)), this architecture is particularly efficient as a universal function approximator (Girosi & Poggio, 1990; Park & Sandberg, 1991; Rizaner & Rizaner, 2018; Scarselli & Tsoi, 1998); one reason for this is that the hidden layer of the RBF network performs a similar kernel transformation to an infinite-dimensional inner product space employed by other kernel machines such as the support vector machine (Abu-Mostafa et al., 2012; Que & Belkin, 2019; dos Santos et al., 2012). The RBF network has therefore seen many uses especially in timeseries forecasting, function approximation, control and classification problems that occur in many real-world applications such as fraud detection, speech recognition, manufacturing, medical diagnosis, and face recognition (Dash et al., 2016; Li & Verma, 2016; Wong et al., 2011). One other area in which the radial basis function network has seen increasing adoption is in the linear approximation of the value function in reinforcement learning in terms of the state–action variables (Barreto & Anderson, 2008; Kretchmar & Anderson, 1997; Sutton & Barto, 2018).

One reason for the ubiquity of RBF networks in many machine learning tasks is the interpretability of their outputs, since the hidden layer essentially performs a fuzzy nearest-neighbour association of an input vector to a set of well-defined exemplars in the training data; thus, for a given input, the influence of different features on the output can be estimated from the relative importance of the features in these exemplars. Another reason for the sustained use of the RBF network is the speed in training the network, since only the final linear layer is often trained; for the least-squares error (with ridge regularisation), there is, in fact, a closed-form solution for the network weights. This comes at the expense of a usually unsupervised step of selecting the

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Correspondence to: Twitter, 901 King Street W, Toronto, ON M5V 3H5, Canada.
E-mail addresses: kgyamfi@twitter.com (K.S. Gyamfi), james.brusey@coventry.ac.uk (J. Brusey), elena.gaura@coventry.ac.uk (E. Gaura).

https://doi.org/10.1016/j.eswa.2021.115982
Received 20 January 2021; Received in revised form 24 September 2021; Accepted 25 September 2021
Available online 13 October 2021

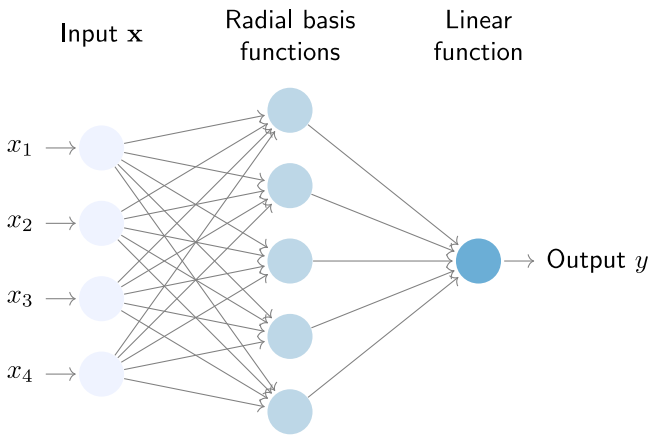

Fig. 1. Radial basis function network (RBFN).

number, centres and widths of the radial basis function activation units. Thus, the eventual performance of the RBF network is highly susceptible to the widths and centre locations of the RBF units (Lee et al., 2009; Lim et al., 2019; Orr, 1995; Scheibel et al., 1999), which in turn can be heavily influenced by the presence of noise in the data (Dey et al., 2019; Masnadi-Shirazi & Subramaniam, 2020). Other neural network architectures may not be so exceptionally sensitive to noise. Several other approaches mainly using forward selection (Chen et al., 2008, 2006; Gomm & Yu, 2000; Mehrabi et al., 2009) and sophisticated clustering methodologies or the self-organising map (Dash et al., 2016; Huilan et al., 2005; Kamalabady & Salahshoor, 2008; Lim et al., 2019) have thus been proposed to better locate the RBF centres.

Crucially however, for sequence modelling tasks such as timeseries forecasting, the sequence data has to be reshaped such that the RBF network takes as inputs the l lagged values of the sequence (in addition to any exogenous inputs) and outputs some next points in the sequence; therefore, if there is a single noisy observation in the sequence, this observation likely gets replicated l times in the reshaped data that is input to the RBF network. In the extreme case, the sequence data may be so corrupted that the data contains no more meaningful information for prediction (Masnadi-Shirazi & Subramaniam, 2020). This profoundly degrades the optimal placement of the RBF centres and consequently the performance of the network. It is worth noting that this problem of noise propagation, as described, is not prevalent in the application of the RBF network to other regression or classification tasks where there are no temporal correlations in the data, in which case each noisy observation occurs only once in training. For example, in reinforcement learning settings, since the state transition in the sequential decision process is typically considered Markovian, there is functional dependence on only the last state vector, i.e. $l = 1$, and thus a single noisy observation does not replicate itself in the reshaped training data.

In this paper, we propose a novel neural network architecture, termed the differential RBF network (RBF-DiffNet), that is designed to learn a representation of the sequence data that ignores signal noise. By utilising activation blocks made up of partial differential equations (PDEs) linear in terms of the radial basis function —with the constant linear coefficients of the PDE being trainable —we subject the network to a regularisation based on backward Euler updates that makes the network robust to noise. The intuition behind this architecture stems from the observation that many real-world sequence data derive from phenomena (such as in biology, economics or physics) whose underlying dynamics, in the absence of noise or control inputs, may be modelled by a set of partial differential equations, however complex. For example, the sequence of air temperature data observed in a car cabin may very well be described by a set of heat balance equations (Brusey et al., 2018), which are PDEs.

Our main contributions in this paper are therefore as follows: (1) we solve the known problem of the susceptibility of radial basis function networks to noise in timeseries forecasting applications; (2) we introduce the differential RBF network architecture, whose hidden layer blocks are partial differential equations (PDE) linear in terms of the radial basis function; (3) we analyse the mathematical properties of the proposed network and show how the hidden layer blocks regularise the network and make it robust to noisy perturbations to sequential data that easily degrades the performance of the baseline RBF network; (4) we propose a fast recursive algorithm for training the proposed network weights including the linear coefficients of the PDE. The proposed network, its analysis and training are detailed in Section 3.

Section 4 presents an experimental validation of the proposed architecture on the logistic map chaotic timeseries (Farmer & Sidorowich, 1987; Maathuis et al., 2017) and 30 different real-world retail timeseries from the M5 competition (Makridakis, 2020); this section also includes performance comparisons with the baseline RBF network, autoregressive integrated moving average (ARIMA) model, ensembles of MLPs, and recurrent neural networks with long–short-term memory (LSTM) blocks (Makridakis, 2020). From the experimental results, our proposed network consistently shows a marked reduction over the baseline RBF network in terms of the prediction error (e.g., 41% reduction in the root mean squared scaled error on the M5 dataset); Our network also shows a comparable performance to the LSTM ensemble at less than one-hundredth the LSTM computational time. Conclusions and future work are given in Section 5, while the problem statement and related work are presented in the next section.

2. Problem statement and related work

We begin by considering a set of N input–output pairs $\{\mathbf{x}_n, y_n\}_{1:N}$ from which we wish to train a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathbf{x}_n \in \mathcal{X}$ is a d-dimensional input vector and $y_n \in \mathcal{Y}$ is a scalar-valued output. In the case of a univariate series $s_1, s_2, \ldots, s_T$, denoted as $\{s_\pi\}_{1:T}$, where π indexes the timestep, we reshape the data into input–output pairs $\{\mathbf{x}_n, y_n\}_{1:N}$ as before, such that $\mathbf{x}_n$ is given by the l lagged values of the series prior to time $t_{\pi+1}$, i.e., $\mathbf{x}_n^\top = [s_{\pi-l+1}, \ldots, s_\pi]$, and y_n is the true output $s_{\pi+1}$. If there are other exogenous inputs $\mathbf{e}$ on which the sequence $\{s_\pi\}_{1:T}$ has dependence, we assume this is again captured in $\mathbf{x}_n$ as:

$$\mathbf{x}_n^\top = [\mathbf{e}^\top, s_{\pi-l+1}, \ldots, s_\pi]. \tag{1}$$

Note that if there are exogenous inputs, then $l < d$; otherwise $l = d$.

The radial basis function network (RBFN) as shown in 1, can be written concisely as:

$$f(\mathbf{x}_n) = \sum_{j=1}^{c} w_j \phi_j(\mathbf{x}_n) + w_0, \tag{2}$$

where c is the number of RBF centres, corresponding to the number of neurons in the hidden layer, and

$$\phi_j(\mathbf{x}_n) = e^{-\beta_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2} \tag{3}$$

is the Gaussian radial basis function, which is the RBF used in this paper. The choice of non-linearity in other radial basis functions has been found not to be crucial to the performance of the network (Chen et al., 1991). In (3), $\boldsymbol{\mu}_j$ are exemplars in the training data; β_j is inversely proportional to the width of the jth RBF, and w_j, w_0 are the RBF weights and bias to be optimised.

From (2) and (3), the hidden layer of the RBF network essentially performs a fuzzy nearest-neighbour association of the input vector $\mathbf{x}_n$ to the set of exemplars $\boldsymbol{\mu}_j$, based on a Mahalanobis distance criterion with spherical covariance in terms of β_j. Thus, for any given input vector, the influence of different features on the output can be estimated from the relative importance of the features in each of the c exemplars, each weighted by its corresponding RBF weight w_j; this property makes the output of the RBF network interpretable in terms of its inputs.

For the sake of brevity, we drop the subscript n in $\mathbf{x}_n$ in the following. In matrix form, (2) is equivalent to:

$$f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \tilde{\mathbf{w}}, \tag{4}$$

where,

$$\boldsymbol{\phi}(\mathbf{x}) = [1, \phi_1(\mathbf{x}), \dots, \phi_c(\mathbf{x})]^\top, \tag{5}$$

and

$$\tilde{\mathbf{w}} = [w_0, w_1, \dots, w_c]^\top \tag{6}$$

If one considers the entire training set, then we have the following system of equations:

$$\mathbf{f} = \boldsymbol{\Phi}\tilde{\mathbf{w}}, \tag{7}$$

where

$$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_N)]^\top, \tag{8}$$

and

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_c(\mathbf{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}_N) & \cdots & \phi_c(\mathbf{x}_N) \end{bmatrix} \tag{9}$$

For the least-squares error (used in many regression tasks) and low to moderate number of RBF nodes, $\tilde{\mathbf{w}}$ can be optimised in closed-form as:

$$\tilde{\mathbf{w}}^* = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \gamma\mathbf{I})^{-1}\boldsymbol{\Phi}\mathbf{y}, \tag{10}$$

where γ is a regularisation coefficient, $\mathbf{I}$ is the identity matrix of size $c+1$, and $\mathbf{y}^\top = [y_1, \dots, y_n]$. In general, the weights $\tilde{\mathbf{w}}$ can be trained for any arbitrary loss function using different optimisation routines.

Separate from the RBF network weights are the hyperparameters c, $\boldsymbol{\mu}_j$ and β_j that require tuning. Most commonly, these are obtained from an unsupervised preprocessing step; the number of RBF centres is fixed and the exemplars $\boldsymbol{\mu}_j$ are determined as the cluster centres obtained from some variants of K-Means clustering (Lim et al., 2019; Masnadi-Shirazi & Subramaniam, 2020; Que & Belkin, 2019; Scholkopf et al., 1997), while β_j is derived from the compactness of the individual clusters. Specifically, β_j is defined as:

$$\beta_j = \frac{1}{2\sigma_j^2}, \tag{11}$$

where popular choices of σ_j (Benoudjit & Verleysen, 2003; McCormick, 2013; Moody & Darken, 1989; Wu et al., 2012) include:

$$\sigma_j := \frac{d_{max}}{\sqrt{2c}}, \quad \forall j \in \{1, \dots, c\} \tag{12}$$

$$\sigma_j := \frac{1}{n_j}\sum_{\mathbf{x}\in C_j} \|\mathbf{x} - \boldsymbol{\mu}_j\|, \quad \forall j \in \{1, \dots, c\} \tag{13}$$

$$\sigma_j := \frac{1}{o}\sum_{\mathbf{x}\in \mathcal{R}_j} \|\mathbf{x} - \boldsymbol{\mu}_j\|, \quad \forall j \in \{1, \dots, c\} \tag{14}$$

$$\sigma_j := \frac{1}{oc}\sum_{j=1}^{c}\sum_{\mathbf{x}\in \mathcal{R}_j} \|\mathbf{x} - \boldsymbol{\mu}_j\|, \quad \forall j \in \{1, \dots, c\}, \tag{15}$$

where d_{max} is the maximum distance between the centres of any two clusters, C_j is the set of all points in the jth cluster, n_j is the number of points in the jth cluster, and $\mathcal{R}_j$ is the set of the o closest points to the jth cluster centre. An alternative to K-Means clustering employed for RBF networks is the self-organising map which allows for a more intuitive determination of the number of cluster centres c when the data is projected onto two or three dimensions (Kamalabady & Salahshoor, 2008; Lin & Chen, 2005).

Since the RBF network parameters affect the network's performance quite significantly, other approaches utilising other performance metrics have been proposed for the selection of the hyperparameters. For example, the RBF nodes may be incrementally added in order to maximise the Fisher class-separability ratio and the leave-one-out cross validation RMSE for classification and regression tasks, respectively (Chen et al., 2008, 2006). Alternatively, the incremental additions of the neurons can be based on the concept of "neuron significance" that evaluates an RBF node's contribution to the overall performance of the network (Lee et al., 2009). Nevertheless, the presence of observational noise tends to influence the optimal placement of centres (Dey et al., 2019) or any metrics computed based on them, such as the Fisher class-separability ratio. The prevalence of observational noise tends to have an even more profound effect for sequence data (Masnadi-Shirazi & Subramaniam, 2020), where upon reshaping into input–output pairs, noisy observations get replicated throughout the data if the lag l is much greater than 1.

One way to make the RBF network robust is to normalise the network to achieve the *partition of unity* property where the c normalised radial basis functions sum up to one for every point in the input space $\mathcal{X}$, making the RBF network less susceptible to noisy observations in arbitrary regions of the input space (Shorten & Murray-Smith, 1994, 1996). Normalisation thus results in the RBF network losing its local characteristics, but often results in improved generalisability (Bugmann, 1998). The normalised RBF network f_{norm} is given mathematically as:

$$f_{norm}(\mathbf{x}_n) = \frac{\sum_{j=1}^{c} w_j\phi_j(\mathbf{x}_n)}{\sum_{j=1}^{c}\phi_j(\mathbf{x}_n)} + w_0, \tag{16}$$

Since normalising the RBF network has several known side effects such as shifts in the maxima of the RBFs (Shorten & Murray-Smith, 1994, 1996), in this paper, we take a different perspective towards making the RBF network robust specifically for its application to modelling sequential data. This involves utilising activation blocks made up of partial differential equations linear in terms of the radial basis function and based on backward Euler discretisation of the sequence data.

It is worth mentioning that our proposed differential RBF network architecture, although it utilises a similar Euler discretisation as the neural ordinary differential equation (ODE-Net) (Chen et al., 2018), differs from the ODE-Net as follows: while the ODE-Net parameterises the derivatives of the hidden state of a residual network with another neural network in the limit of a large number of hidden layers when the discrete step size approaches 0, the differential RBF network parameterises the solution to the differential equation with an RBF network and directly evaluates the derivatives of this solution according to the backward Euler updates given in Section 3 in such a way as to regularise the original RBF network.

Furthermore, our work differs from other works that have employed the RBF network in solving ordinary or partial differential equations (Chen et al., 2016; Lagaris et al., 1998; Larsson et al., 2017; Mai-Duy & Tran-Cong, 2001; Schaback & Wendland, 1970) in that, instead of a well-defined set of differential equations to solve, we have only some discrete realisations from phenomena that are assumed to be described by unknown differential equations. Thus, we start with a solution to an unknown differential equation in the form of the RBF network and work backwards to find an optimal differential equation with constant coefficients that best describes the sequence data, based on backward Euler updates.

Finally, the DiffNet architecture for deep learning proposed by Ward et al. (2021) is similar only in name with our proposed RBF-DiffNet architecture, but it differs fundamentally from our approach in the learning task it is used for: while our approach is set up for sequential modelling tasks, the model by Ward et al. is set up for supervised dimensionality reduction using autoencoders.

3. Proposed differential RBF network

In this section, we provide the mathematical intuition behind our proposed differential RBF network and analyse how the network achieves regularisation against noise. We further propose a fast recursive algorithm to train the network.

3.1. Intuition

We first consider the univariate series $\{s_\pi\}_{1:T}$, and assume that this sequence data is generated by some underlying differential equation given by:

$$\frac{dz}{dt} = g(t, z(t)), \tag{17}$$

that is, the sequence $\{s_t\}_{1:T}$ are instances sampled from the solution to the differential equation given by (17) at specific timesteps. These discrete samples may be approximated by backward Euler updates (Atkinson et al., 2011) of the form:

$$s_{\pi+1} = s_\pi + hg(t_{\pi+1}, s_{\pi+1}), \tag{18}$$

where h is some small interval between the occurrences of s_π and $s_{\pi+1}$, i.e., $h = t_{\pi+1} - t_\pi$. Note that in a noiseless system, $s_\pi = z(t_\pi)$.

We wish to approximate the function z with the RBF network f. However, since we wish to predict the next state of the sequence based on its prior values, we replace the functional dependence on time t with the last l realisations of the sequence in the function f. Thus, for $l = 1$ (and no exogenous inputs), we wish to approximate the local value of $s_{\pi+1} = z(t_{\pi+1})$ as:

$$s_{\pi+1} = z(t_{\pi+1}) \approx f(s_\pi). \tag{19}$$

With this approximation, the differential equation equivalent to (17) becomes:

$$\frac{df}{ds_\pi} = \sum_{j=1}^{c} w_j \phi_j'(s_\pi), \tag{20}$$

given the definition of the RBF network in (2), and the Euler update equations become:

$$s_{\pi+1} = s_\pi + hf'(s_\pi). \tag{21}$$

We may then train the RBF network f on the input–output pairs $\{s_\pi, s_{\pi+1}\}$ (for example, via the pseudo-inverse solution in (10)), while also constraining the network to satisfy the Euler update equations in (21). This constraint helps regularise the network weights against noisy samples, since the noisy samples may be unlikely to satisfy the backward Euler update in (21), for a sufficiently small h.

The backward Euler update in (21) can be thought of as a first-order Taylor's approximation, and thus to improve this approximation and reduce the truncation errors, we may consider a higher-order Taylor's expansion up to degree ν as follows:

$$s_{\pi+1} = s_\pi + hf'(s_\pi) + \cdots + \frac{h^\nu}{\nu!} f^{(\nu)}(s_\pi). \tag{22}$$

One of the utilities in approximating the solution z to the underlying differential equation in (17) as an RBF network f is in its parameter efficiency, i.e., we are able to obtain closed-form expressions for its higher-order derivatives without any increase in the number of variables that parameterise f or these higher derivatives.

If we now consider larger values for the lag l, as well as include any exogenous inputs, then the RBF network is no longer a function of s_π only, but $\mathbf{x}_n$ as defined in (1). Again, as in Section 2, we drop the subscript n for brevity. In this case, the gradient and higher order derivatives of f is computed for all the components of $\mathbf{x}$, so that we have a form of the Euler update analogous to (22) using multi-index notation (Folland, 2005) as:

$$s_{\pi+1} = \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \sum_{0 \le |k| \le \nu} \frac{\mathbf{h}^k}{k!} D^{(k)} f(\mathbf{x}), \tag{23}$$

which is equivalent to:

$$s_{\pi+1} = \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \mathbf{h}^\top Df(\mathbf{x}) + \frac{1}{2}\mathbf{h}^\top D^2 f(\mathbf{x})\mathbf{h} + \cdots \tag{24}$$

Here, $\mathbf{h}$ is now a small vector interval, $D^i f$ is an ith-order tensor, with Df and $D^2 f$ being the gradient and Hessian respectively of f, and $\lambda^\top \mathbf{s}_{\pi-l+1:\pi}$ is the weighted sum of the last l realisations of the sequence, with λ being the vector of weights.

For the special case where $\nu = 2$, since Df and $D^2 f$ are respectively the gradient and Hessian of f, we have from (24) that,

$$s_{\pi+1} \approx \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \sum_{i=1}^{d} h_i \frac{\partial f(\mathbf{x})}{\partial x_i} + 0.5 \sum_{i=1}^{d}\sum_{p=1}^{d} h_i h_p \frac{\partial^2 f(\mathbf{x})}{\partial x_i x_p}, \tag{25}$$

where h_i, h_p are respectively the ith and pth components of $\mathbf{h}$. We note here that the computational complexity is $O(d^2)$.

In general, for an arbitrary ν, the computational complexity would be $O(d^\nu)$, which may be intractable for high-dimensional datasets. Thus, to keep the order of complexity in (24) linear and tractable, we ignore all mixed derivatives (such as $\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_p}$ in the summation in (25) involving the second-order tensor, where $i \neq p$) thus approximating (24) with the following partial differential equation:

$$s_{\pi+1} \approx \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \sum_{k=1}^{\nu}\sum_{i=1}^{d} a_{k,i} \frac{\partial^{(k)} f(\mathbf{x})}{\partial x_i^k}. \tag{26}$$

Because we ignore the mixed derivatives for computational reasons, the Taylor approximation no longer holds, and thus we introduce in (26) a new set of learnable parameters $a_{k,i}$ (which replace $\mathbf{h}$ in (24)) which now have to be optimised to maximise the fit to the data. In the case where $\nu = 1$, $a_{k,i}$ is equal to h as given in (21).

With f defined as in (19), we have from (26) that:

$$f(\mathbf{x}) = \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \sum_{k=1}^{\nu}\sum_{i=1}^{d} a_{k,i} \frac{\partial^{(k)} f(\mathbf{x})}{\partial x_i^k}. \tag{27}$$

The relationship in (27) then represents a regularisation of the parameters of f in (2). Specifically, while the radial basis function network f is ordinarily given by (2), we are now subjecting it to the additional constraint that the next observation $s_{\pi+1} = f(\mathbf{x})$ is such that it satisfies the multivariate, higher-order extension to the backward Euler updates as given in (26), assuming that the sequence data are sampled from the solution to an underlying differential equation; this can be expressed by the following optimisation problem:

$$\min_{\mathbf{w}} \sum_n L(f(\mathbf{x}_n), y_n), \quad \text{where} \quad f(\mathbf{x}_n) = \sum_{j=1}^{c} w_j \phi_j(\mathbf{x}_n) + w_0$$

subject to :

$$s_{\pi+1} = f(\mathbf{x}_n) = \lambda^\top \mathbf{s}_{\pi-l+1:\pi} + \sum_{k=1}^{\nu}\sum_{i=1}^{d} a_{k,i} \frac{\partial^{(k)} f(\mathbf{x}_n)}{\partial x_i^k}, \tag{28}$$

where L is an arbitrary loss function, and $\mathbf{w}$ are the RBF network weights. Note that this optimisation framework presumes knowledge of $a_{k,i}$ and λ; in Section 3.3, we detail how to recursively optimise the network parameters $\mathbf{w}$ jointly with $a_{k,i}$ and λ.

By requiring that the RBF network satisfies the backward-Euler updates, the regularisation we introduce in (28) ensures that the network weights $\mathbf{w}$ are trained to discount noisy training instances that do not follow the sequential behaviour given by the backward-Euler update equations. Thus, the proposed network relatively shows more robustness to noisy observations in sequence data which affect the choice of RBF centres and widths, causing the vanilla RBF network to underperform. Consequently, while fitting the data to the differential equation given by (27), we are implicitly only learning a regularised version of the parameters w_j, i.e., regularised by the equality constraint in (28).

3.2. Higher-order derivatives of the RBF

The fundamental idea in our proposed approach is fitting the data to the backward-Euler-regularised RBF network given by the differential equation in (27), rather than training the RBF network to approximate f directly as a function of $\mathbf{x}$ as in (2). The form of (27) thus requires

us to obtain the expressions for the first and higher-order derivatives of the radial basis function network.

Deriving from (2), the first-order partial derivative of f is given by:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \sum_{j=1}^{c} w_j \frac{\partial \phi_j(\mathbf{x})}{\partial x_i}, \tag{29}$$

where

$$\frac{\partial \phi_j(\mathbf{x})}{\partial x_i} = -2\beta_j (x_i - \mu_{j,i}) e^{-\beta_j \|\mathbf{x}-\boldsymbol{\mu}_j\|^2} \tag{30}$$

Accordingly, the second-order partial derivative of f (ignoring mixed derivatives) is given by:

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i^2} = \sum_{j=1}^{c} w_j \frac{\partial^2 \phi_j(\mathbf{x})}{\partial x_i^2}, \tag{31}$$

where

$$\frac{\partial^2 \phi_j(\mathbf{x})}{\partial x_i^2} = 2\beta_j \left[2\beta_j (x_i - \mu_{j,i})^2 - 1\right] e^{-\beta_j \|\mathbf{x}-\boldsymbol{\mu}_j\|^2}. \tag{32}$$

This generalises to a higher-order ν as follows:

$$\frac{\partial^{(\nu)} z(\mathbf{x})}{\partial x_i^{\nu}} = \sum_{j=1}^{c} w_j \frac{\partial^{(\nu)} \phi_j(\mathbf{x})}{\partial x_i^{\nu}}, \tag{33}$$

Similar results have been obtained for the Gaussian RBF (Mai-Duy & Tran-Cong, 2003) and the multiquadric radial basis functions (Mai-Duy & Tran-Cong, 2001).

Due to the form of the Gaussian radial basis function in (3), $\phi_j(\mathbf{x})$ is infinitely differentiable. In the following, we derive the formula for finding the kth derivative of $\phi_j(\mathbf{x})$ given by (3). First, we consider the generalised Leibniz rule for finding the kth derivative of the product uv given as:

$$(uv)^{(k)} = \sum_{m=0}^{k} \binom{k}{m} u^{(k-m)} v^{(m)}. \tag{34}$$

If we start from the first-order partial derivative of $\phi_j(\mathbf{x})$ in (30), we can express this as a product uv, where:

$$u = -2\beta_j (x_i - \mu_{j,i}), \tag{35}$$

and

$$v = e^{-\beta_j \|\mathbf{x}-\boldsymbol{\mu}_j\|^2} = \phi_j(\mathbf{x}). \tag{36}$$

From Leibniz rule, we can then derive the kth order derivative of the RBF as:

$$\frac{\partial^{(k)} \phi_j(\mathbf{x})}{\partial x_i^k} = \begin{cases} \phi_j(\mathbf{x}), & k = 0 \\ \sum_{m=0}^{k-1} \binom{k-1}{m} u^{(k-m-1)} \frac{\partial^{(m)} \phi_j(\mathbf{x})}{\partial x_i^{(m)}}, & k \geq 1, \end{cases} \tag{37}$$

which computes the kth partial derivative recursively.

If we now substitute the radial basis function network f and its higher-order partial derivatives into the PDE in (26), we obtain:

$$f = \lambda^{\top} \mathbf{s}_{\pi-l+1:\pi} + \sum_{j=1}^{c} w_j \sum_{k=1}^{\nu} \sum_{i=1}^{d} a_{k,i} \frac{\partial^{(k)} \phi_j(\mathbf{x})}{\partial x_i^k}, \tag{38}$$

whose network architecture is given in Fig. 2.

In matrix form, (38) is equivalent to:

$$f = \lambda^{\top} \mathbf{s}_{\pi-l+1:\pi} + \mathbf{w}^{\top} \boldsymbol{\Theta}(\mathbf{x}) \mathbf{a}, \tag{39}$$

where,

$$\mathbf{w} = [w_1, \ldots, w_c]^{\top}, \tag{40}$$

$$\mathbf{a} = [a_{1,1}, \ldots, a_{1,d}, \ldots, a_{\nu,1}, \ldots, a_{\nu,d}]^{\top}, \tag{41}$$

and

$$\boldsymbol{\Theta}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^{(1)}\phi_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial^{(1)}\phi_1(\mathbf{x})}{\partial x_d} & \cdots & \frac{\partial^{(\nu)}\phi_1(\mathbf{x})}{\partial x_1^{\nu}} & \cdots & \frac{\partial^{(\nu)}\phi_1(\mathbf{x})}{\partial x_d^{\nu}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial^{(1)}\phi_c(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial^{(1)}\phi_c(\mathbf{x})}{\partial x_d} & \cdots & \frac{\partial^{(\nu)}\phi_c(\mathbf{x})}{\partial x_1^{\nu}} & \cdots & \frac{\partial^{(\nu)}\phi_c(\mathbf{x})}{\partial x_d^{\nu}} \end{bmatrix} \tag{42}$$

3.3. Training RBF-DiffNet

Using (39), we may then optimise $\mathbf{w}, \lambda$ and $\mathbf{a}$ jointly on the training data for arbitrary objective functions. For example, for the mean-squared error ϵ given as:

$$\epsilon = \frac{1}{N} \sum_{n=1}^{N} (y_n - f_n)^2 = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \lambda^{\top} \mathbf{s}_{\pi-l+1:\pi,n} - \mathbf{w}^{\top} \boldsymbol{\Theta}(\mathbf{x}_n) \mathbf{a}\right)^2, \tag{43}$$

where $\mathbf{s}_{\pi-l+1:\pi,n}$ is the l lagged values of the series for the nth training instance, we can optimise the network weights $\mathbf{w}, \lambda$ and $\mathbf{a}$ to an arbitrary accuracy with a given choice of an optimiser such as stochastic gradient descent (SGD). Here, we denote f_n as $f_n = f(\mathbf{x}_n)$.

However, when we consider the matrix form of the differential RBF network f in (39), we observe that given any two out of the three sets of network parameters: $\mathbf{w}, \mathbf{a}$ and λ, the third set of parameters has a closed form solution from (43), since the system of all n training instances becomes linear. For example, given λ and $\mathbf{a}$, we may obtain a closed-form solution for $\mathbf{w}$ as follows:

$$\bar{\mathbf{w}} = (\boldsymbol{\Psi}_{\mathbf{w}}^{\top} \boldsymbol{\Psi}_{\mathbf{w}})^{-1} \boldsymbol{\Psi}_{\mathbf{w}}^{\top} \eta_{\mathbf{w}}, \tag{44}$$

where $\boldsymbol{\Psi}_{\mathbf{w}}$ is an N-by-c matrix whose rows are given by $\boldsymbol{\Theta}(\mathbf{x}_n)\mathbf{a}$, and $\eta_{\mathbf{w}}$ is an N-dimensional vector whose components are $y_n - \lambda^{\top} \mathbf{s}_{\pi-l+1:\pi,n}$.

Similarly, given λ and $\mathbf{w}$, we may obtain a closed-form solution for $\mathbf{a}$ as follows:

$$\bar{\mathbf{a}} = (\boldsymbol{\Psi}_{\mathbf{a}}^{\top} \boldsymbol{\Psi}_{\mathbf{a}})^{-1} \boldsymbol{\Psi}_{\mathbf{a}}^{\top} \eta_{\mathbf{a}}, \tag{45}$$

where $\boldsymbol{\Psi}_{\mathbf{a}}$ is an N-by-νd matrix whose rows are given by $\boldsymbol{\Theta}(\mathbf{x}_n)^{\top}\mathbf{w}$, and $\eta_{\mathbf{w}}$ is an N-dimensional vector whose components are $y_n - \lambda^{\top} \mathbf{s}_{\pi-l+1:\pi,n}$.

In the same way, given $\mathbf{a}$ and $\mathbf{w}$, we may obtain a closed-form solution for λ as follows:

$$\bar{\lambda} = (\boldsymbol{\Psi}_{\lambda}^{\top} \boldsymbol{\Psi}_{\lambda})^{-1} \boldsymbol{\Psi}_{\lambda}^{\top} \eta_{\lambda}, \tag{46}$$

where $\boldsymbol{\Psi}_{\lambda}$ is an N-by-l matrix whose rows are given by $\mathbf{s}_{\pi-l+1:\pi,n}$, and η_{λ} is an N-dimensional vector whose components are $y_n - \mathbf{w}^{\top} \boldsymbol{\Theta}(\mathbf{x}_n)\mathbf{a}$.

This leads us to propose a relatively inexpensive recursive approach to training the RBF-DiffNet which is given in Algorithm 1:

The choices of weight initialisations for the proposed network are justified as follows:

1. By setting each component of λ as $\frac{1}{l}$, equal weights are assigned to the last l values of the sequence.
2. By setting $a_{i,k}$ as $a_{i,k} = \frac{0.1^k}{k!}$, the PDE weights are defined as Taylor expansion coefficients with 0.1 being the interval between any two realisations of the series.
3. By setting $\mathbf{w}$ as $\mathbf{w} = [\tilde{w}_1^*, \ldots, \tilde{w}_c^*]$, we utilise the weights obtained from the unnormalised RBF network.

The main computational load in the proposed algorithm is in lines 14 to 16, which involve obtaining closed-form solutions via the pseudo-inverse; the matrix sizes used in the inversions are c, l, and νd respectively, where c is the number of RBF centres, l is the number of lags or lookback window, ν is the order of the PDE, and d is the dimensionality of the input space (i.e., d equals the number of lags l plus the number of any exogenous inputs). Since $l \leq d$, and $\nu \geq 1$ for the Euler update, then $l \leq \nu d$, and hence the computational performance of RBF-DiffNet depends mainly on the relative magnitudes of νd and c. Since matrix inversion for the pseudo-inverse solution is cubic in the number of input dimensions, the order of complexity of RBF-DiffNet, as shown in Algorithm 1, is $O(I_{max}(\max(c, \nu d)^3)$. In comparison, the order of complexity of the RBF network is $O((c+1)^3)$. Thus, if $c \gg \nu d$, then RBF-DiffNet achieves similar computational performance as the vanilla RBF network.

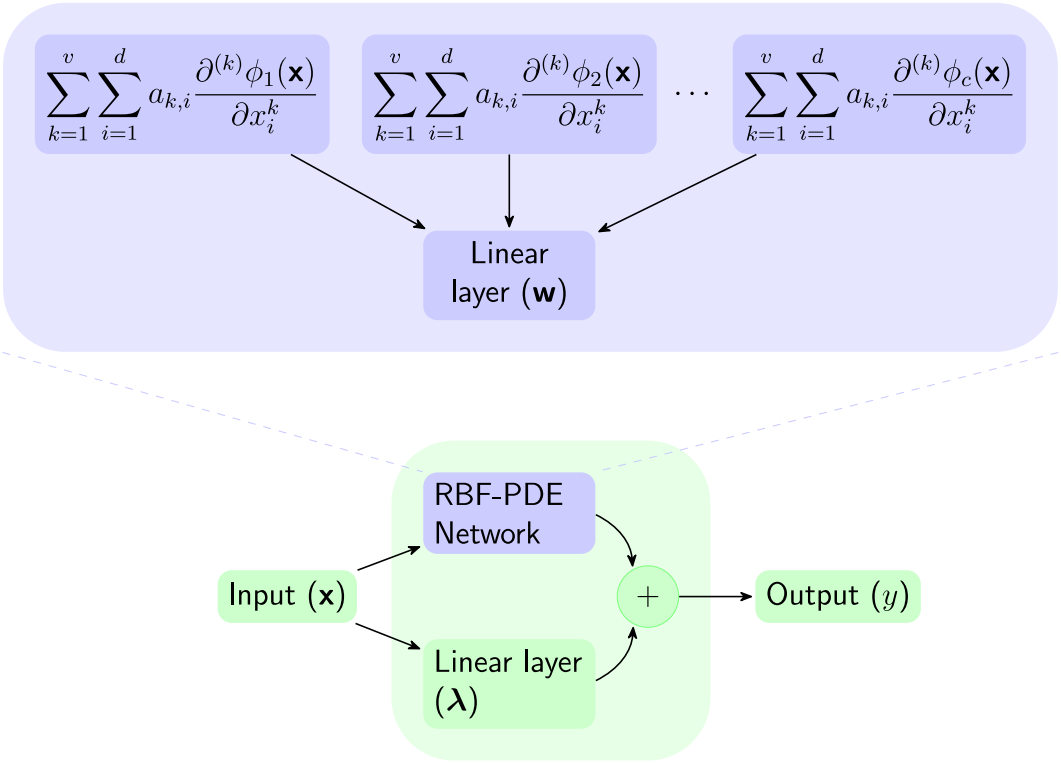

Fig. 2. Proposed differential RBF network architecture.

Algorithm 1 RBF-DiffNet training

procedure RBF-DiffNet($X_{train}, y_{train}, c, \nu, I_{max}$)
  Obtain RBF centres by K-Means clustering for c clusters.
  Obtain RBF widths according to one of (12) to (15).
  Initialise the entries of $\mathbf{a}$ as $a_{i,k} = \frac{0.1^k}{k!}$, where i, k index the kth-order partial derivative of the RBF with respect to the component x_i of the input $\mathbf{x}_n$, and $k \in \{1, ..., \nu\}$.
  Initialise each component of λ to $\frac{1}{l}$, where l denotes the number of lags of the sequence being considered as inputs to the model.
  Initialise $\mathbf{w}$ as $\mathbf{w} = [\tilde{w}_1^*, ..., \tilde{w}_c^*]$, where $\tilde{w}_1^*, ..., \tilde{w}_c^*$ are obtained from (10) with no regularisation, given that $\tilde{\mathbf{w}}^* = [\tilde{w}_0^*, \tilde{w}_1^*, ..., \tilde{w}_c^*]^\top$.
  while Number of iterations < I_{max} **do**
    Evaluate the loss function
    if Loss function is the minimum found so far **then**
      $\mathbf{a}^\dagger := \mathbf{a}$
      $\mathbf{w}^\dagger := \mathbf{w}$
      $\lambda^\dagger := \lambda$
    end if
    Set $\mathbf{a}, \lambda$ to previous iteration's values, and obtain a closed-form solution for $\mathbf{w}$ from (44) on $\{X_{train}, y_{train}\}$.
    Set $\mathbf{a}, \mathbf{w}$ to previous iteration's values, and obtain a closed-form solution for λ from (46) on $\{X_{train}, y_{train}\}$.
    Set $\mathbf{w}, \lambda$ to previous iteration's values, and obtain a closed-form solution for $\mathbf{a}$ from (45) on $\{X_{train}, y_{train}\}$.
    Update newly computed $\mathbf{a}, \lambda, \mathbf{w}$ for current iteration.
  end while
  Return optimal network parameters: $\mathbf{a}^\dagger, \mathbf{w}^\dagger, \lambda^\dagger$.
end procedure

4. Experimental work

In this section, we validate the proposed differential RBF network on the logistic map chaotic timeseries (Farmer & Sidorowich, 1987; Maathuis et al., 2017), as well as on 30 different timeseries from the M5 competition (Makridakis, 2020); the reasons for the choice of these datasets are given in Sections 4.1 and 4.2 respectively.

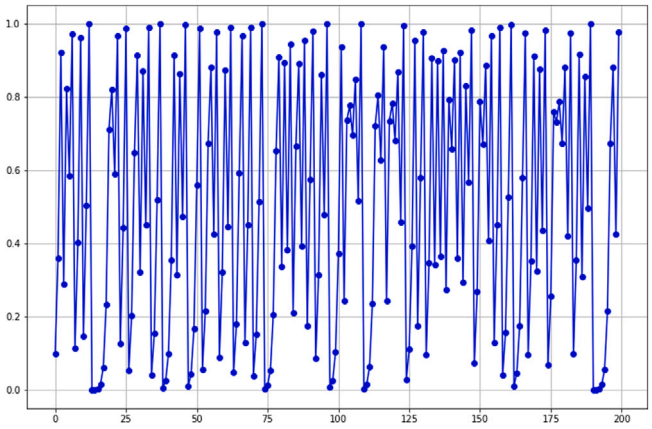

Fig. 3. Logistic map chaotic timeseries.

4.1. Logistic map

We consider the logistic map given as:

$$s_{\pi+1} = r s_\pi [1 - s_\pi], \tag{47}$$

and we generate 1000 observations in the forward pass as our timeseries using an initial value of 0.1, and $r \in \{3.8, 3.9, 4.0\}$; the first 200 observations for $r = 4.0$ are shown in Fig. 3: We split the timeseries into training and testing sets, with the last 100 observations as the test set. We have selected the logistic map because, although it is an archetypal chaotic series, it arises from simple polynomial mappings that can be easily modelled by the RBF network. Furthermore,

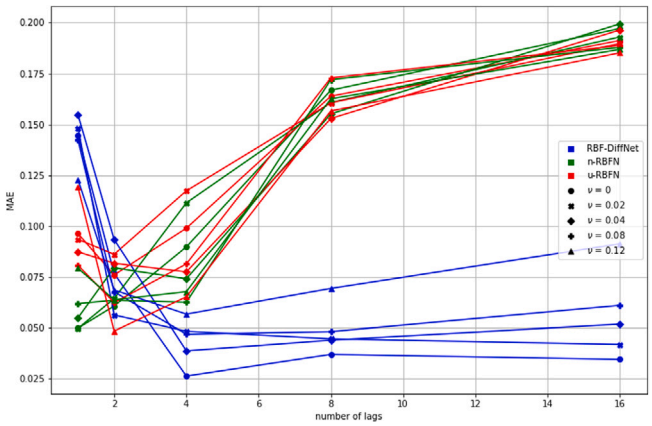

Fig. 4. MAE results on logistic map: $r = 3.8$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. ν is the noise variance.

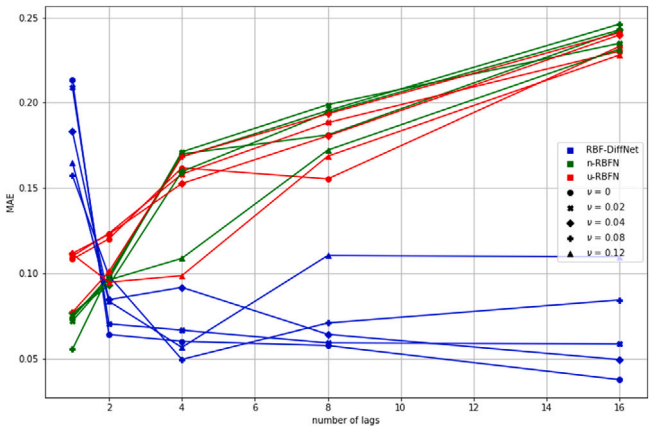

Fig. 5. MAE results on logistic map: $r = 3.9$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. ν is the noise variance.

Table 1
Model training times on three generated logistic map timeseries: $r = 3.8, 3.9, 4.0$ given in the form: mean ± s.d. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively.

Lags l	RBF-DiffNet	n-RBFN	u-RBFN
$l = 1$	0.52 ± 0.07	0.22 ± 0.02	0.22 ± 0.02
$l = 2$	0.65 ± 0.05	0.26 ± 0.02	0.26 ± 0.01
$l = 4$	0.84 ± 0.05	0.326 ± 0.02	0.26 ± 0.02
$l = 8$	2.15 ± 0.20	0.37 ± 0.01	0.37 ± 0.02
$l = 16$	5.78 ± 0.17	0.61 ± 0.03	0.62 ± 0.04

to simulate observational noise, we add some Gaussian noise with variance $\omega \in \{0, 0.02, 0.04, 0.08, 0.12\}$ to the timeseries (whose original chaotic behaviour is not due to noise) in order to demonstrate the noise susceptibility of the RBF network. We have selected ω so that it does not exceed the variance of the original timeseries which is 0.12, in order not to corrupt the information contained in the series. We perform mean-normalisation, and then reshape the training data into input–output pairs using a lags of $l \in \{1, 2, 4, 8, 16\}$ with which we train the unnormalised and normalised RBF networks and the proposed differential RBF network. Although $l = 1$ suffices for the logistic map because it depends only its previous input, as shown in (47), we experiment with exponentially increasing l in order to demonstrate the replication of noisy observations as inputs to the RBF network. The number of RBF centres used is $c = \max(5, 2l)$, which is what showed the best performance after experimenting with different configurations. On the test set, we perform one-step prediction and report the mean absolute error (MAE). We defer multi-step prediction to Section 4.2.

For all the RBF networks (i.e., unnormalised, normalised and differential variants), we determine the exemplars $\boldsymbol{\mu}_j$ via K-Means clustering while the RBF parameters β_j are defined according to (13). We minimise the least-squares error with lasso regularisation, with the regularisation coefficient determined via cross-validation. Furthermore, the order ν of the PDE in the differential RBF network is set empirically to $\nu = 3$ based on cross-validation results. We subsequently optimise the network weights according to Algorithm 1 using $I_{max} = 100$. All the RBF networks (including RBF-DiffNet) are coded in Python using the scikit-learn linear models package to fit the networks' final linear layer.

The MAE results from these experiments are given in Figs. 4 to 6, and the computational performance given in Table 1.

The logistic map experiments are run on a 8GB-RAM Intel Core i5-1035G1 CPU @1.00 GHz 1.19 GHz. All the RBF networks (including RBF-DiffNet) are coded in Python using the scikit-learn linear models package to fit the networks' final linear layer.

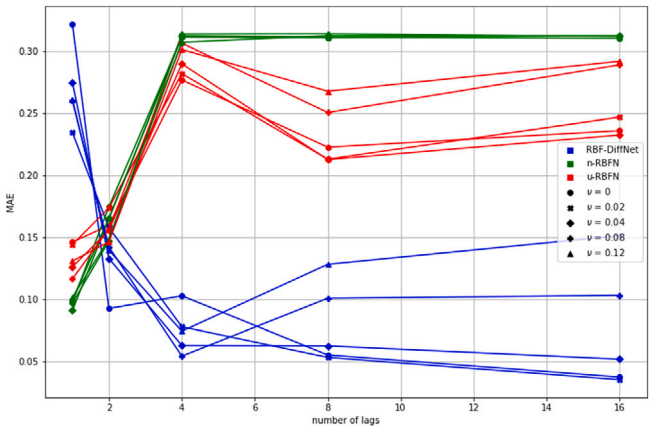

Fig. 6. MAE results on logistic map: $r = 4.0$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. ν is the noise variance.

4.2. M5 dataset

The M5 competition is the latest in a series of M-forecasting competitions organised by the University of Nicosia that has drawn the interest of top forecasting practitioners including Google (Fry & Brundage, 2019) and Uber (Smyl, 2020). The M5 competition comprises 42840 individual timeseries data made available by Walmart. This dataset consists of unit sales of products across the United States at different levels of aggregation. This dataset has been chosen because, like many real-world datasets, it exhibits a non-deterministic (noisy) behaviour. Furthermore, by not including exogenous inputs such as prices that can affect product sales, we impose further *deterministic noise* (Abu-Mostafa et al., 2012) on the series. Thus, the dataset is able to sufficiently illustrate the susceptibility of the baseline RBF network to noisy inputs in sequential data.

Due to computational constraints, we work at the level 8 aggregation that consists of 30 different timeseries data of unit sales of products aggregated for each of 10 stores (across three US states) and 3 product categories (i.e., foods, household and hobbies). Each of the 30 different timeseries has 1941 daily observations (spanning the period between 29 January 2011 and 19 June 2016), with the last 28 days set for validation.

Different benchmark algorithms (categorised under statistical, machine learning and combinations of both) have been provided by the M5 competition organisers, and in this section, we limit our comparison

to one statistical benchmark: autoregressive integrated moving average (ARIMA), and one machine learning benchmark: the multi-layer perceptron (MLP). The benchmark MLP architecture specified in the M5 competition is given as follows (Makridakis, 2020):

1. Number of input layer nodes = 14, corresponding to the last 14 days of sales data
2. Number of hidden layer nodes = 28,
3. Number of output nodes = 1
4. Hidden layer activation function: logistic; output layer activation function: linear,
5. Number of iterations = 500,
6. Ensemble size = 10, i.e, 10 different MLPs are trained and their predictions aggregated in terms of the median operator to account for poor weight initialisations.

Using the above network parameters, we also implement an ensemble of 10 recurrent networks with LSTM blocks which use tanh activation for further comparison.

To make the RBF networks comparable to the MLP architecture, we set d (the RBF network input size) to 14 and c (the number of RBF nodes) to 28. The RBF network centres are again obtained via K-Means clustering, and the RBF widths are set according to (14), using $o = 10$; the order ν of the PDE in the differential RBF network is set to empirically to $\nu = 1$, based on cross-validation results.

The MLP and LSTM networks are implemented using Keras and optimised with an Adam optimiser, while the ARIMA model is implemented using the statsmodels package in Python. Similar to the logistic map experiments, all the RBF networks (including RBF-DiffNet) are coded in Python using the scikit-learn linear models package to fit the networks' final linear layer.

For each timeseries, we set the last 28 observations as the test set, and the remaining as the training set. Since preprocessing for timeseries forecasting typically involves stationarising the series to make the series easier to predict (Gheyas & Smith, 2009; Pal & Prakash, 2017), we first perform a stationarity test on the training set using the augmented Dickey–Fuller test (Cheung & Lai, 1995); if the timeseries does not exhibit stationarity according to the test's null hypothesis, we proceed to difference the timeseries incrementally until it shows stationarity, up to a differencing order of 10. We then perform mean-normalisation and then reshape the data into input–output pairs using a lag of $l = 14$.

On the test set, we perform reverse-differencing and normalisation operations to recover the original series and range, and we perform multi-step prediction over the forecast horizon of 28 days. We report the root mean squared scaled error (RMSSE), which is the error metric used in the M5 competition. The RMSSE is defined as:

$$\text{RMSSE} = \frac{(n_{train}-1)}{n_{test}} \frac{\sum_{\pi=n_{train}+1}^{n_{train}+n_{test}} (s_\pi - \hat{s}_\pi)^2}{\sum_{\pi=2}^{n_{train}} (s_\pi - s_{\pi-1})^2}, \tag{48}$$

where $\hat{s}_\pi$ is a model's estimate of the timeseries at time t_π, $n_{train} = 1913$ is the length of training series, and $n_{test} = 28$ is the length of the forecast horizon.

We do not investigate cross-learning, where information from other timeseries are used in training a single global model to predict each timeseries (Makridakis et al., 2020); this is out of the scope of this paper.

Due to the large ensemble of neural networks involved here, the M5 experiments are run on a 32GB-RAM Tesla P100-PCIE 128GB GPU processor with 8 Intel E5-2650 CPU cores @2.2 GHz. The MLP and LSTM networks are implemented using Keras and optimised with an Adam optimiser, while the ARIMA model is implemented using the statsmodels package in Python. All the RBF networks (including RBF-DiffNet) are coded in Python using the scikit-learn linear models package to fit the networks' final linear layer.

Table 2
Root mean squared scaled error (RMSSE) results on the M5 dataset (level 8 aggregation). RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. Best values across all algorithms are in boldface. Best values across the three RBF networks (RBF-DiffNet, u-RBFN, n-RBFN) are in italics.

Series ID	RBF-DiffNet	u-RBFN	n-RBFN	ARIMA	MLP	LSTM
1	0.558	**0.389**	0.393	0.513	0.433	0.46
2	**0.641**	0.717	0.668	0.686	0.76	0.749
3	**0.463**	1.02	0.783	0.512	0.467	0.489
4	*0.781*	1.49	1.38	0.867	0.801	**0.683**
5	*0.725*	1.21	1.06	0.884	0.833	**0.581**
6	**0.593**	0.94	0.657	0.749	0.664	0.598
7	*0.439*	0.569	0.581	0.492	**0.438**	0.653
8	*0.592*	1.7	1.05	0.582	**0.552**	0.554
9	**0.511**	2.3	2.23	0.556	0.528	0.616
10	**0.605**	0.987	1.06	0.691	0.743	0.663
11	*0.957*	1.34	1.15	0.978	0.935	**0.928**
12	**0.856**	2.67	2.58	0.875	1.12	1.16
13	*0.794*	1.03	0.887	0.862	**0.697**	1
14	**0.709**	1.26	0.798	0.923	0.759	0.725
15	*0.848*	1.08	1.08	**0.83**	0.837	1.03
16	0.659	*0.641*	0.658	0.55	**0.521**	0.571
17	*0.649*	0.841	0.782	0.667	0.686	**0.637**
18	*0.546*	1.38	0.991	**0.536**	0.578	0.659
19	1.57	1.08	**1.02**	1.11	1.38	1.12
20	**0.795**	1.91	1.27	0.916	0.877	0.817
21	**0.742**	0.91	0.913	0.809	0.75	1.33
22	**0.592**	0.781	0.717	0.753	0.704	0.634
23	*0.499*	0.606	0.514	0.522	0.648	**0.485**
24	**0.593**	1.09	0.958	0.687	0.78	0.788
25	*1.49*	1.75	1.59	1.87	1.46	**1.44**
26	**0.726**	1.21	1.42	0.741	0.731	0.73
27	*1.06*	2.22	2.35	1.12	0.988	**0.975**
28	*0.751*	1.77	1.36	1.03	**0.705**	0.761
29	0.838	*0.803*	0.807	0.821	**0.778**	0.81
30	*0.731*	2.04	1.51	0.697	0.67	**0.639**
Mean	**0.744**	1.26	1.11	0.794	0.76	0.776
Median	*0.717*	1.09	1.01	0.751	0.737	**0.704**

5. Results and discussion

In this section, we present the results of the experiments on the logistic map and the 30 M5 timeseries data. We discuss the performance of the proposed RBF-DiffNet in comparison with the existing methods tested on these experiments.

5.1. Logistic map data

In Figs. 7 to 9, we show the original timeseries (ground truth) as well as the one-step predictions by the different algorithms for visual comparisons; due to space limitations, we show this for only three cases: at noise level $\nu = 0.8$ for lag values $l = 1, 4, 16$. Other experimental settings yield similar graphs.

From the MAE results in Figs. 4 to 6, there is an increase in the MAE values for the unnormalised RBF network (u-RBFN) as the lag l increases. This is especially observed for high noise levels ω, since the noisy observations gets replicated many times in the reshaped data that is input to the RBF network, illustrating the susceptibility of the unnormalised RBF network to overfitting on noisy observations. The normalised RBF network (n-RBFN), on the other hand, shows robustness in its predictions across different noise levels at low lag values, with the MAE values remaining relatively constant. However, as the lag l increases, its performance also deteriorates for each of the noise levels. This performance degradation is mainly due to the side effects of normalisation (Shorten & Murray-Smith, 1994, 1996) whereby the maxima of the RBF shifts when the RBFs have unequally spaced centres or different widths. As Shorten (Shorten & Murray-Smith, 1994) notes, this side effect becomes more pronounced with increase in the input dimension. This causes points far away from the RBF centre to have high functional values, and thus, rather than a few

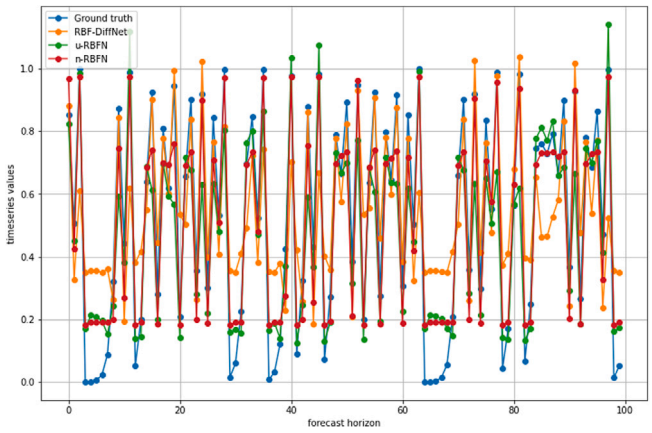

Fig. 7. Model predictions on logistic map: $r = 4.0, l = 1, \nu = 0.8$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. l is the number of lags; ν is the noise variance.

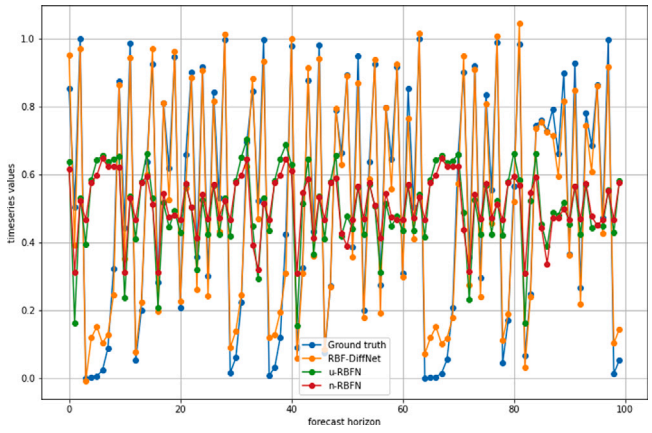

Fig. 8. Model predictions on logistic map: $r = 4.0, l = 4, \nu = 0.8$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. l is the number of lags; ν is the noise variance.

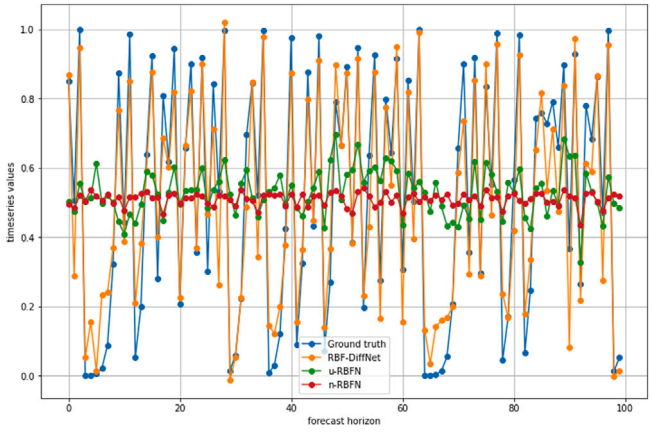

Fig. 9. Model predictions on logistic map: $r = 4.0, l = 16, \nu = 0.8$. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. l is the number of lags; ν is the noise variance.

RBFs getting activated, many more RBFs tend to contribute uniformly to the output for any given input. The differential RBF network (RBF-DiffNet), however, shows robustness across different noise variance ω and different values of l, significantly outperforming the benchmark

Table 3
P-values from Wilcoxon signed rank test for differences in median between proposed differential RBF network (RBF-DiffNet) and other algorithms. u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively.

Algorithm	p-value
u-RBFN	5.558×10^{-6}
n-RBFN	1.654×10^{-5}
ARIMA	5.886×10^{-4}
MLP	2.000×10^{-1}
LSTM	1.798×10^{-1}

Table 4
Model training time on the M5 datasets (level 8 aggregation) given in the form: mean ± s.d. obtained on a 32GB-RAM Tesla P100-PCIE 128GB GPU processor with 8 Intel E5-2650 CPU cores @2.2 GHz. RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively. The MLP and LSTM networks are implemented using Keras and optimised with an Adam optimiser, while the ARIMA model is implemented using the statsmodels package in Python. Similar to the logistic map experiments. All the RBF networks (including RBF-DiffNet) are coded in Python using the scikit-learn linear models package to fit the networks' final linear layer.

Algorithm	Training time (seconds)
RBF-DiffNet	4.8 ± 0.13
u-RBFN	1.27 ± 0.19
n-RBFN	1.20 ± 0.16
ARIMA	33.34 ± 10.74
MLP	307.23 ± 4.30
LSTM	633.61 ± 6.84

RBF networks. In particular, for low to medium noise levels, RBF-DiffNet is able to utilise more information in the lagged values of the series as l increases to improve the prediction accuracy, without any noise enhancement or side effects of normalisation, unlike for u-RBFN and n-RBFN. While the MAE performance can be seen to degrade as the noise variance increases, this is to be expected as the information in the timeseries increasingly gets corrupted.

5.1.1. *Influence of lags l on RBF-DiffNet performance*

However, RBF-DiffNet noticeably underperforms for $l = 1$ across all noise levels as seen in Figs. 4 to 6. At $l = 2$, it achieves comparable performance to the baseline RBF networks. The accuracy gains with RBF-DiffNet only starts to show from $l \geq 4$. This phenomenon is also observed in Figs. 7 to 9, where the RBF-DiffNet has poor predictive performance for $l = 1$ but achieves superior performance in Figs. 8 and 9. The reason for this behaviour is in the nature of the network output as summarised in (39):

$$f = \lambda^{\top}\mathbf{s}_{\pi-l+1:\pi} + \mathbf{w}^{\top}\boldsymbol{\Theta}(\mathbf{x})\mathbf{a}.$$

For small values of the lag l, e.g., $l <= 2$, the first term of the equation $\lambda^{\top}\mathbf{s}_{\pi-l+1:\pi}$ comprises only the last couple of realisations of the series; if the series is noisy or chaotic, then the weighted sum of the last one or two realisations is hardly predictive on unseen data, but has a negative impact on the performance of the network in terms of overfitting on noise. As l increases however, the first term $\lambda^{\top}\mathbf{s}_{\pi-l+1:\pi}$ tends towards the expected value of the series, and with that, the network learns the trend in the series, ultimately yielding an improved predictive performance. In contrast, in the case of the baseline RBF networks, there is no additive term for the weighted sum of the last l realisations, as they are all mapped to prototypes in the hidden layer; hence the baseline RBF networks tend to perform better for low values of l. However, for high values of l, any noisy observations get replicated many times in the reshaped data, causing the baseline RBF network to overfit on noise, and underperform compared to the proposed RBF-DiffNet. For this reason, the RBF-DiffNet is most suited to timeseries forecasting tasks for which the series to be predicted is influenced by more than one or two previous realisations of the series: an example of this is demand forecasting of grocery items, where future sales may

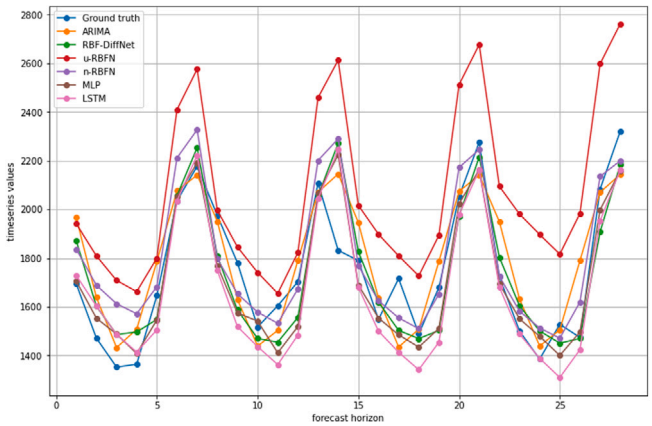

Fig. 10. Model predictions on M5 (series id 9). RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively.

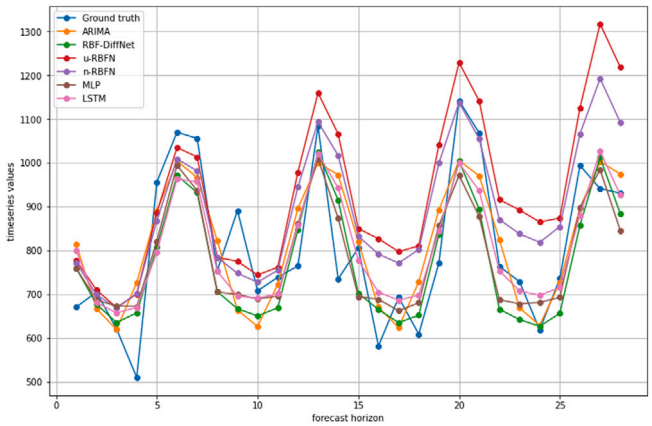

Fig. 11. Model predictions on M5 (series id 30). RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively.

be autocorrelated with historical sales up to $l = 14$ days prior; other examples include weather forecasting, household energy consumption forecasting, or sentence completion in natural language processing.

5.1.2. Computational considerations

In terms of computational performance as given in Table 1, u-RBFN and n-RNFN having closed-form solutions given in (10) outperform the proposed RBF-DiffNet which requires a recursive optimisation routine given in Algorithm 1. In particular, the training time increases more profoundly with the number of lags l (this is equivalently an increase in the dimensionality d) for RBF-DiffNet than for n-RBFN and u-RBFN, because RBF-DiffNet has computational complexity of order $O(I_{max}(\max(c, vd)^3)$, while n-RBFN has $O((c+1)^3)$; the computational performance is detailed in Section 3.3. For a large number of RBF nodes, the solution given by (10) may be computationally intractable and u-RBFN and n-RBFN may require iterative optimisation techniques such as SGD. Since u-RBFN and n-RBFN each has $c + 1$ optimising variables as compared to $c + dv + l + 1$ for the differential RBF network, the existing RBF networks may still have a computational edge over RBF-DiffNet in an iterative optimisation technique.

5.2. M5 datasets

In Figs. 10 and 11, we show the original timeseries (ground truth) as well as the 28-steps predictions by the different algorithms for visual comparisons; due to space limitations, we show this for only series id 9 and 30. Other series show similar graphs.

Although the different models are able to maintain relatively constant prediction accuracy across the forecast horizon due to the seasonalities in the timeseries, the results in Table 2 show that the differential RBF network (RBF-DiffNet) achieves superior predictive accuracy over the unnormalised RBF network (u-RBFN) and normalised RBF network (n-RBFN). Over the 30 different timeseries, RBF-DiffNet achieves a 41% reduction over u-RBFN and a 33% reduction over n-RBFN in terms of the mean RMSSE. This performance improvement is shown to be statistically significant at the 0.95 confidence level, as seen from the p-values in Table 3.

The RBF-DiffNet is also shown to yield a 6% reduction over ARIMA (significant at the 0.95 confidence level), 2% over the MLP (statistically insignificant), and 4% over the LSTM (statistically insignificant) in terms of the RMSSE.

The reason for the performance gain of RBF-DiffNet over u-RBFN and n-RBFN is the relative robustness of RBF-DiffNet to noise as described in Section 5.1 due to its backward-Euler regularisation, while the performance gain of RBF-DiffNet over ARIMA is because the RBF network generally learns complex non-linear mappings as compared to the simple linear mapping in the ARIMA model.

Despite their comparable performance, it is worth noting that RBF-DiffNet uses only one set of weight initialisation as given in Algorithm 1, whereas the LSTM and MLP ensembles respectively use 10 different weight initialisations and the median operator to mitigate the effects of poor weight initialisations. The single weight initialisation is possible in RBF-DiffNet because its network weights have more intuitive meanings which allow for an informed choice of initialisation in such a way that multiple random restarts yield only minor performance improvements. Consequently, for the comparable performance given in Table 2, RBF-DiffNet requires at least 132 times less computational time than the LSTM ensemble and 64 times less time than the MLP ensemble, as shown in Table 4. Moreover, RBF-DiffNet has the added advantage of being interpretable in terms of the influence of different input features on the output, since the hidden layer performs a fuzzy nearest-neighbour association of a given input to the RBF centres.

5.3. Qualitative comparisons of algorithms

Despite the relative superiority of the proposed RBF-DiffNet architecture in terms of predictive accuracy, there are a few limitations to its application, most notably, it is only suited to sequence modelling tasks and does not currently work for general regression tasks, unlike the baseline RBF network or the multilayer perceptron (MLP). Based on the results of the two sets of experiments, the relative performance of the proposed algorithm with respect to the existing algorithms are summarised in Table 5 across 6 different performance criteria:

6. Conclusion and future work

The radial basis function (RBF) network has good function approximation properties, fast training time and is easily interpretable in terms of the influence of different input features on the output. However, it is especially sensitive to noisy inputs due to the usually unsupervised step in selecting the RBF centres and widths. This paper extends the RBF network for use on noisy sequential data. For this purpose, we have introduced a novel differential RBF network (RBF-DiffNet) whose hidden layer blocks are higher-order constant-coefficient partial differential equations in terms of the RBF. RBF-DiffNet exhibits robustness to noisy perturbations to the input sequence by regularising the network following backward-Euler updates, assuming the sequence data originates from an underlying differential equation. Notably, RBF-DiffNet differs from the neural ordinary differential equation (ODE-Net) in that RBF-DiffNet parameterises the solution to the underlying differential

Table 5
RBF-DiffNet is the proposed differential RBF network; u-RBFN and n-RBFN are the unnormalised and normalised RBF networks respectively.

Criterion	RBF-DiffNet	u-RBFN/n-RBFN	LSTM/MLP	ARIMA
Accuracy	Superior accuracy	Poor in the presence of noise	Superior accuracy	Average
Training time	Average: requires only a few more matrix inversions than the baseline RBF network)	Low: has a simple pseudo-inverse solution	High: requires several epochs to train, as well as multiple initialisations	Average: requires maximising the likelihood for a given ARIMA order
Ability to interpret	Easy, since hidden layer maps any test input to relevant prototypes in the data	Easy, since hidden layer maps any test input to relevant prototypes in the data	Difficult, due to many non-linear hidden layers between input and output	Easy, due to its autoregressive setup
Application domain	Only sequential modelling	All regression tasks	MLP can be used for all regression tasks	Only sequential modelling
Robustness	Robust to noise	Susceptible to noise	Robust to noise	Robust to noise
Weight initialisation	Easy to initialise, because weights have intuitive explanations	Not applicable	Different weight initialisations are typically required to give good performance	Not applicable

equation with an RBF network and directly evaluates the derivatives of the network based on the backward-Euler discretisation, while the ODE-Net parameterises the derivatives of the hidden state of a residual network with another neural network.

The proposed differential RBF network is experimentally validated on the logistic map and 30 other real-world and noisy timeseries data from the M5 competition (level 8 aggregation). Experimental results show RBF-DiffNet significantly outperforms the normalised and unnormalised RBF networks at different noise levels on the logistic map. On the M5 dataset, RBF-DiffNet outperforms the normalised and unnormalised RBF networks by 33% and 41% respectively; furthermore, because its network weights have intuitive meanings, RBF-DiffNet allows for an informed choice of initialisation for the network weights, thereby achieving comparable performance to an ensemble of 10 LSTM networks (built from 10 different weight initialisations) at less than one-hundredth the LSTM computational time.

Our proposed network therefore enables more accurate predictions—in spite of the presence of observational noise—in sequence modelling tasks such as timeseries forecasting that leverage the fast training and function approximation properties of the RBF network, where model interpretability is desired. Such timeseries forecasting tasks may include demand forecasting in retail, weather forecasting, or energy consumption forecasting.

Nevertheless, the current development of RBF-DiffNet is limited to scalar outputs, and on-going work is investigating its extension to vector outputs. This involves training the network to approximate backward Euler updates involving vector sequences, rather than training a different network for each output. The feasibility of the proposed approach to cross-learning, where information from multiple series is used in training a single global model, is also an area we are currently researching. Finally, convergence analysis of the proposed recursive training algorithm for RBF-DiffNet is being conducted.

CRediT authorship contribution statement

Kojo Sarfo Gyamfi: Conceptualisation, Methodology, Formal analysis, Writing – original draft, Writing – review & editing. **James Brusey:** Validation, Writing – review & editing, Formal analysis, Visualisation, Supervision. **Elena Gaura:** Validation, Writing – review & editing, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The authors would like to acknowledge the financial support of the Engineering and Physical Science Research Council (EPSRC) for funding the EnergyREV project (EP/S031863/1).

References

Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H.-T. (2012). Learning from data, vol. 4. AMLBook New York, NY, USA.

Atkinson, K., Han, W., & Stewart, D. E. (2011). *Numerical solution of ordinary differential equations, vol. 108*. John Wiley & Sons.

Barreto, A. d. M. S., & Anderson, C. W. (2008). Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, *172*(4–5), 454–482.

Benoudjit, N., & Verleysen, M. (2003). On the kernel widths in radial-basis function networks. *Neural Processing Letters*, *18*(2), 139–154.

Broomhead, D. S., & Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks. Royal Signals and Radar Establishment Malvern (United Kingdom).

Brusey, J., Hintea, D., Gaura, E., & Beloe, N. (2018). Reinforcement learning-based thermal comfort control for vehicle cabins. *Mechatronics*, *50*, 413–421.

Bugmann, G. (1998). Normalized Gaussian radial basis function networks. *Neurocomputing*, *20*(1–3), 97–110.

Chen, S., Cowan, C. F., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, *2*(2), 302–309.

Chen, Y., Gottlieb, S., Heryudono, A., & Narayan, A. (2016). A reduced radial basis function method for partial differential equations on irregular domains. *Journal of Scientific Computing*, *66*(1), 67–90.

Chen, S., Hong, X., Luk, B. L., & Harris, C. J. (2008). Construction of tunable radial basis function networks using orthogonal forward selection. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *39*(2), 457–466.

Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in neural information processing systems* (pp. 6571–6583).

Chen, S., Wang, X., Hong, X., & Harris, C. J. (2006). Kernel classifier construction using orthogonal forward selection and boosting with Fisher ratio class separability measure. *IEEE Transactions on Neural Networks*, *17*(6), 1652–1656.

Cheung, Y.-W., & Lai, K. S. (1995). Lag order and critical values of the augmented Dickey–Fuller test. *Journal of Business & Economic Statistics*, *13*(3), 277–280.

Dash, C. S. K., Behera, A. K., Dehuri, S., & Cho, S.-B. (2016). Radial basis function neural networks: a topical state-of-the-art survey. *Open Computer Science*, *1*(open-issue).

Dey, P., Gopal, M., Pradhan, P., & Pal, T. (2019). On robustness of radial basis function network with input perturbation. *Neural Computing and Applications*, *31*(2), 523–537.

Farmer, J. D., & Sidorowich, J. J. (1987). Predicting chaotic time series. *Physical Review Letters*, *59*(8), 845.

Folland, G. (2005). Higher-order derivatives and Taylor's formula in several variables. (pp. 1–4). Preprint Retrieved 17/01/2021, 2021, from https://sites.math.washington.edu/~folland/Math425/taylor2.pdf.

Fry, C., & Brundage, M. (2019). The M4 forecasting competition-A practitioner's view.

Gheyas, I. A., & Smith, L. S. (2009). A neural network approach to time series forecasting. In *Proceedings of the world congress on engineering*, vol. 2 (pp. 1–3).
Girosi, F., & Poggio, T. (1990). Networks and the best approximation property. *Biological Cybernetics*, *63*(3), 169–176.
Gomm, J. B., & Yu, D. L. (2000). Selecting radial basis function network centers with recursive orthogonal least squares training. *IEEE Transactions on Neural Networks*, *11*(2), 306–314.
Huilan, J., Ying, G., Dongwei, L., & Jianqiang, X. (2005). Self-adaptive clustering algorithm based RBF neural network and its application in the fault diagnosis of power systems. In *2005 IEEE/PES transmission & distribution conference & exposition: asia and pacific* (pp. 1–6). IEEE.
Kamalabady, A. S., & Salahshoor, K. (2008). New siso and miso adaptive nonlinear predictive controllers based on self organizing rbf neural networks. In *2008 3rd international symposium on communications, control and signal processing* (pp. 703–708). IEEE.
Kaviani, S., & Sohn, I. (2021). Application of complex systems topologies in artificial neural networks optimization: An overview. *Expert Systems with Applications*, Article 115073.
Kretchmar, R. M., & Anderson, C. W. (1997). Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of international conference on neural networks (ICNN'97), vol. 2* (pp. 834–837). IEEE.
Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000.
Larsson, E., Shcherbakov, V., & Heryudono, A. (2017). A least squares radial basis function partition of unity method for solving PDEs. *SIAM Journal on Scientific Computing*, *39*(6), A2538–A2563.
Lee, C.-C., Chiang, Y.-C., Shih, C.-Y., & Tsai, C.-L. (2009). Noisy time series prediction using M-estimator based robust radial basis function neural networks with growing and pruning techniques. *Expert Systems with Applications*, *36*(3), 4717–4724.
Li, M. M., & Verma, B. (2016). Nonlinear curve fitting to stopping power data using RBF neural networks. *Expert Systems with Applications*, *45*, 161–171.
Lim, E. A., Tan, W. H., & Junoh, A. K. (2019). Distance weighted K-means algorithm for center selection in training radial basis function networks. *IAES International Journal of Artificial Intelligence*, *8*(1), 54.
Lin, G.-F., & Chen, L.-H. (2005). Time series forecasting by combining the radial basis function network and the self-organizing map. *Hydrological Processes: An International Journal*, *19*(10), 1925–1937.
Maathuis, H., Boulogne, L., Wiering, M., & Sterk, A. (2017). Predicting chaotic time series using machine learning techniques. In *Preproceedings of the 29th benelux conference on artificial intelligence. Groningen* (pp. 326-340).
Mai-Duy, N., & Tran-Cong, T. (2001). Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Networks*, *14*(2), 185–199.
Mai-Duy, N., & Tran-Cong, T. (2003). Approximation of function and its derivatives using radial basis function networks. *Applied Mathematical Modelling*, *27*(3), 197–220.
Makridakis, S. (2020). The M5 competition.
Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74.
Masnadi-Shirazi, M., & Subramaniam, S. (2020). Attractor ranked radial basis function network: A nonparametric forecasting approach for chaotic dynamic systems. *Scientific Reports*, *10*(1), 1–10.
McCormick, C. (2013). Radial basis function network (rbfn) tutorial.
Mehrabi, S., Maghsoudloo, M., Arabalibeik, H., Noormand, R., & Nozari, Y. (2009). Application of multilayer perceptron and radial basis function neural networks in differentiating between chronic obstructive pulmonary and congestive heart failure diseases. *Expert Systems with Applications*, *36*(3), 6956–6959.
Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, *1*(2), 281–294.
Orr, M. J. (1995). Regularization in the selection of radial basis function centers. *Neural Computation*, *7*(3), 606–623.
Pal, A., & Prakash, P. (2017). *Practical time series analysis: master time series data processing, visualization, and modeling using python*. Packt Publishing Ltd.
Park, J., & Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation*, *3*(2), 246–257.
Que, Q., & Belkin, M. (2019). Back to the future: radial basis function network revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(8), 1856–1867.
Rizaner, F. B., & Rizaner, A. (2018). Approximate solutions of initial value problems for ordinary differential equations using radial basis function networks. *Neural Processing Letters*, *48*(2), 1063–1071.
dos Santos, G. S., Luvizotto, L. G. J., Mariani, V. C., & dos Santos Coelho, L. (2012). Least squares support vector machines with tuning based on chaotic differential evolution approach applied to the identification of a thermal process. *Expert Systems with Applications*, *39*(5), 4805–4812.
Scarselli, F., & Tsoi, A. C. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, *11*(1), 15–37.
Schaback, R., & Wendland, H. (1970). Using compactly supported radial basis functions to solve partial differential equations. *WIT Transactions on Modelling and Simulation*, *23*.
Scheibel, F., Steele, N. C., & Low, R. (1999). Centre and variance selection for Gaussian radial basis function artificial neural networks. In *Artificial neural nets and genetic algorithms* (pp. 141–147). Springer.
Scholkopf, B., Sung, K.-K., Burges, C. J., Girosi, F., Niyogi, P., Poggio, T., & Vapnik, V. (1997). Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, *45*(11), 2758–2765.
Shorten, R., & Murray-Smith, R. (1994). On normalising radial basis function networks. In *Proceedings of the fourth irish conference on neural networks, INNC*, vol. 94, (pp. 213–217).
Shorten, R., & Murray-Smith, R. (1996). Side effects of normalising radial basis function networks. *International Journal of Neural Systems*, *7*(02), 167–179.
Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, *36*(1), 75–85.
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.
Teng, P. (2018). Machine-learning quantum mechanics: Solving quantum mechanics problems using radial basis function networks. *Physical Review E*, *98*(3), Article 033305.
Ward, M. D., Zimmerman, M. I., Meller, A., Chung, M., Swamidass, S., & Bowman, G. R. (2021). Deep learning the structural determinants of protein biochemical properties by comparing structural ensembles with DiffNets. *Nature Communications*, *12*(1), 1–12.
Wong, Y. W., Seng, K. P., & Ang, L.-M. (2011). Radial basis function neural network with incremental learning for face recognition. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *41*(4), 940–949.
Wu, Y., Wang, H., Zhang, B., & Du, K.-L. (2012). Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, *2012*.