Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 1, No. 1, January : 2024

APPLICATION IN SOLVING INTEGRO-DIFFERENTIAL EQUATIONS

Mr. Sushil Kumar, Senior Lecturer, Dept. Of Mathematics, IIMT College of Polytechnic Greater Noida (U. P.).
Mr. Dev Kumar Singh, Lecturer, Dept. Of Mathematics, IIMT College of Polytechnic Greater Noida (U. P.).
Mr. Pramesh Kumar, Lecturer, Dept. Of Mathematics, IIMT College of Polytechnic Greater Noida (U. P.).

Abstract
In this paper, we propose using LSTM-RNNs (Long Short-Term Memory- Recurrent Neural Networks) to learn and represent nonlinear integral op- erators that appear in nonlinear integro-differential equations (IDEs). The LSTM-RNN representation of the nonlinear integral operator allows us to turn a system of nonlinear integro-differential equations into a system of or- dinary differential equations for which many efficient solvers are available. We illustrate the efficiency and robustness of this Long Short-Term Memory- Recurrent Neural Networks based numerical IDE solver with a model problem. Additionally, we highlight the generalizability of the learned integral opera- tor by applying it to IDEs driven by different external forces. As a practical application, we show how this methodology can effectively solve the Dyson's equation for quantum many-body systems.
Keywords: Machine learning, Integro-differential equations

I. Introduction
II. Integro-differential equations (IDEs) arise in many scientific applications ranging from nonequilibrium quantum dynamics (Stefanucci, 2013; Cohen, 2011; Reeves, 2023B,A), the dynamics of non-Markovian colloidal particles (Zhu, 2022, 2018, 2020; Zwanzig, 2001) the modeling of dispersive waves
III. (Whitham, 1967; Debnath, 2005) and electronic circuits (Bohner, 2022). One particularly important example is the application of the Kadanoff-Baym equation (Kadanoff, 2018; Stefanucci, 2013; Reeves, 2023B) for the time evo- lution of quantum correlators. This equation describes the propagation in time of non-equilibrium Green's functions, and finding efficient methods of solving this equation is of extreme importance in the study of driven quantum systems. A general type of IDEs can be written as:
IV. d
where both F and the integral kernel K are functions of and t and G(t). An IDE is computationally challenging to solve due to the presence of the integral term in (1). A numerical time evolution scheme typically requires performing a numerical integration of the integral term at each time step. If nT time steps are taken to evolve the numerical solution of 1 to time T , the overall computational complexity in time is proportional to at least n2 , which can be high for a large nT .
The main motivation of this research is to use machine-learning ap- proaches to reduce the computational complexity of solving IDE to the order of nT . More specifically, by representing and learning the nonlinear integral operator I(G(t), t) = t K(G(t s), s)G(s)ds in IDE 1 via neural network (NN), we turn an IDE into an ordinary differential equation (ODE) of the form

$$\frac{D}{dt} G(t) = F(G(t), t) + I(G(t), t), \qquad (2)$$

where I(G(t), t) is a functional of G(t) and t that can be evaluated with a constant cost, i.e., without performing numerical integration. A standard ODE scheme can then be applied to solve 2 with a temporal complexity of O(nT).

In principle, the mapping from G(t) to I(G(t), t), which is implicitly defined by the integral t K(G(t s), s)G(s)ds always exists, although its (memoryless) analytical form is generally unknown (Venturi, 2014; Smirne, 2010). In this work, we seek to represent and learn such a mapping by training a recurrent neural network (RNN) using snapshots of the numerical solution of 1 within a small time window. Hence our approach falls into the category of operator learning methods, which encompasses recently devel- oped machine-learning methodologies such as DeepONet(Lu, 2021) and the Fourier neural operator(Li, 2020; Kovachki, 2023). In an operator learning method, we view the solution to a given ODE or partial differential equation (PDE) as the output of a neural network parameterized operator that maps between function spaces. For instance, solving an initial value problem for a given PDE can be reformulated as searching for a neural network parame- terized operator, denoted as S : u(x, 0) u(x, t). Carefully designed neural networks, such as those employed in DeepONet and the Fourier neural op- erator, can approximate this operator S. By applying the learned operator S to the initial condition u(x, 0) we can readily obtain the solution to the PDE. One notable advantage of the operator learning approach lies in the generalizability of the trained neural network model. Once the approxima- tion to the operator is obtained, it can be applied to other initial conditions or inputs to the model.

In this work, we adopt an operator-learning perspective and choose to use a long-short term memory (LSTM)-based RNN (Hochreiter, 1997) to learn the mapping between G(t) and I(G(t), t). Instead of a simple feed-forward neural network (FFNN), we utilize RNNs because they can better preserve the causality of both G(t) and I(G(t), t). Furthermore, instead of learning the operator that yields the solution to the IDE 1 directly, we choose to learn the mapping between G(t) and I(G(t), t) for the following reasons. First of all, although the solution of 1 depends on both F (G(t), t) (referred to as the streaming term) and I(G(t), t) (referred to as the memory integral or collision integral), the cost of evaluating the memory term typically far exceeds that of the streaming term. Secondly, when the streaming term F (G(t), t) is time- dependent and creates atypical dynamics outside of the training window time data. Consequently, the extrapolated prediction of the solution of 1 for large t can be poor. On the other hand, in many physically relevant scenarios, the integral kernel in 1 is well behaved. As a result, we expect the mapping between G(t) and I(G(t), t), which is independent of the solution G(t) itself, can be learned more easily. Furthermore, in addition to increasing the time window and training the RNN with different initial conditions, we can augment the training data by considering solutions of 1 with different streaming terms within a small time window. We will refer to this type of training as multi-trajectory training in Section 3. This type of training is important for learning an operator that maps from one function space to another. Once the integral operator I(t) is well approximated by a properly trained RNN, the solution of the IDE can be obtained using any standard ODE solver with I(t) evaluated by the RNN. Furthermore, transferability of the learning algorithm is clearly achievable. Once the integral operator with a fixed integral kernel K is learned via an LSTM-based RNN, it can be used to solve the IDE 1 associated with different streaming forces F (G(t), t) by using a standard ODE solver.

What distinguishes the approach presented in this work from other ex- isting neural-network-based differential equation solver is that we do not use the RNN to solve the IDEs directly. Instead, we use it to approximate the time dependent nonlinear integral operator that is costly to evaluate. We combine the RNN based operator learning with a standard ODE solver to obtain numerical solutions to the IDEs. To demonstrate the efficiency of this method, we will benchmark our result against a direct operator learned model that learns the entire solution to the right hand side of Eqn (2), as well as against using DMD to extrapolate the dynamics.

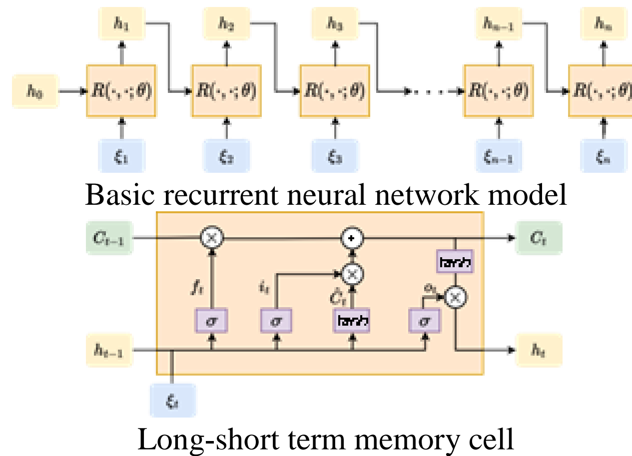
1. Methods

Recurrent Neural Network

Recurrent neural networks (RNN) and their various adaptations have become prevalent tools to analyze and forecast the patterns in time series data (Zhang, 1998; Karim, 2017; Hu, 2020) across a wide range of domains and scenarios (Cho, 2014; Can, 2018).

=Basic RNN model

The workflow of the RNN model is illustrated in Figure 1 (Top), which comprises several key components: including inputs, hidden states, and an RNN cell. The hidden states are calculated recursively within the RNN cell, utilizing sequential inputs. These hidden states play an important role in



Basic recurrent neural network model

Long-short term memory cell

Figure 1: (Top) The workflow of a basic RNN model. The model processes the inputs $\{\xi_1, \xi_2, \cdots, \xi_n\}$ one by one in sequence and produces a corresponding sequence of hidden states $\{h_1, h_2, \cdots, h_n\}$. The same set of model parameters, θ, is employed in the RNN cell (i.e., $R(\cdot, \cdot; \theta)$) to calculate each h_i, $i = 1, \cdots, n$. (Bottom) An LSTM cell. The update of the cell state C_t and the hidden state h_t relies on the current time input ξ_t and the preceding states, h_{t-1} and C_{t-1}, through several information gates.

Specifically, consider a time series dataset $\Xi = \xi_0, \xi_1, \xi_2, \ldots, \xi_N$, where ξ_i represents the ith time index within our time series. The RNN cell, repre- sented as a function $R(\, , \, ; \theta)$, takes the preceding hidden state and the cur- rent time data as input and produces a new hidden state as output. Namely, $h_t = R(\xi_t, h_{t-1}; \theta)$, $t = 1, , N$. Here, h_0 is initialized as a zero vector and θ represents the set of shared and trainable parameters. Mathematically, the hidden states can be written as a composition of the RNN cells given by:

$h_n = R(\xi_n, h_{n-1}; \theta) = R(\xi_n, R(\xi_{n-1}, h_{n-2}; \theta); \theta) = R(R(\ldots R(\xi_2, R(\xi_1, h_0; \theta); \theta), \ldots; \theta); \theta)$.

The sharing of θ enables the RNN to learn general patterns across time and incorporate a memory effect into the model. This capability also allows RNN models to handle input sequences of varying lengths.

Long-short term memory (LSTM)

The LSTM model is a type of RNN that distinguishes itself by employing different gates within its LSTM cell to regulate the flow of information, as shown in Figure 1 (Bottom). Compared with the standard RNN models, LSTM models exhibit better performance in capturing long-range dependen- cies and mitigating the vanishing gradient issue (Hochreiter, 1997). Their gated structure allows them to effectively preserve information across longer sequences. As a result, LSTM models have gained widespread applications in dynamics modeling tasks (Zhu, 2023; Harlim, 2021). In our specific context, we will employ an LSTM model to model the memory integral of the IDE.

The LSTM cell features two distinct states: the cell state, denoted as C_t, and the hidden state, represented as h_t. The cell state serves as a reposi- tory for long-term memory, capable of retaining and propagating information throughout different time steps.

Let i_t, f_t, and o_t represent the input, forget, and output gates at time t. Each of these gates employs a nonlinear activation function, such as $\sigma(x) = 1-$, in conjunction with learnable parameters to govern the selection of information to be incorporated into the state updates. This can be expressed as follows:

$i_t = \sigma(\xi_t, h_{t-1}; \theta_{input})$, $f_t = \sigma(\xi_t, h_{t-1}; \theta_{forget})$, $o_t = \sigma(\xi_t, h_{t-1}; \theta_{output})$.

Here, θ_{input}, θ_{forget}, and θ_{output} are parameter sets within total collection of shareable parameters θ.

Using these gates, the current cell state is updated according to:

$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$,

which combines both long-term memory (e.g., C_{t-1}) and new information (e.g., ξ_t used in the computation of $\hat{C}_t = \tanh(\xi_t, h_{t-1}; \theta_c)$). Here, $\theta_c \in \theta$ is

Minimize the difference

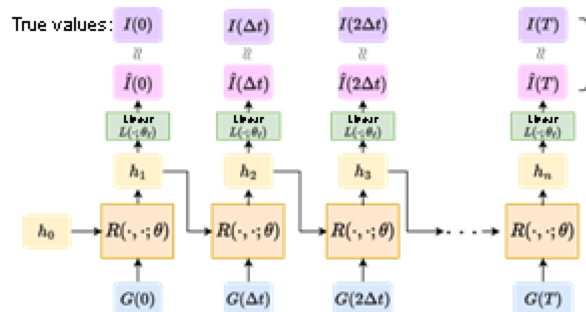


Figure 2: Learning and predicting diagram of the RNN model. In the training phase, the input sequence $G(0)$, $G(\Delta t)$, $G(2\Delta t)$, ..., $G(T)$ is fed into the model to generate pre- dictions $\hat{I}(0)$, $\hat{I}(\Delta t)$, $\hat{I}(2\Delta t)$, ..., $\hat{I}(T)$, using the parameters θ. From here, we can use the final output $\hat{I}(T)$ to produce numerically extrapolated dynamics using the procedure outlined in Model Architecture and illustrated in Figure 3.

the parameter set. Finally, we compute the current hidden state h_t as:

$h_t = o_t \cdot \tanh(C_t)$.

The hidden state h_t can subsequently undergo further trainable transfor- mations, such as a linear transformation, to predict the specific quantity of interest.

RNN customized for learning time-dependent nonlinear integral operator

Having introduced the basic architecture of RNN and the integro-differential equation we aim to solve, in this section, we customize a specific RNN model that enables us to efficiently learn the integral operator of an IDE within a short time window and use that to predict its long-time dynam- ics. As mentioned in the introduction, the RNN we seek should learn a map

$I : G(t)\ I(t)$ for any given function $G(t)$. Since $I(t)$ depends on all the history values of $G(t)$, to capture this memory effect, we propose to use the LSTM cells as the basic modeling modules to build the RNN model. This leads to a learning-predicting diagram as illustrated in Figure 2.

Model architecture and dynamics extrapolation

The RNN in Figure 2 consists of an LSTM model and a linear transforma- tion layer attached to it that maps the output of LSTM cells into the shape

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 1, No. 1, January : 2024

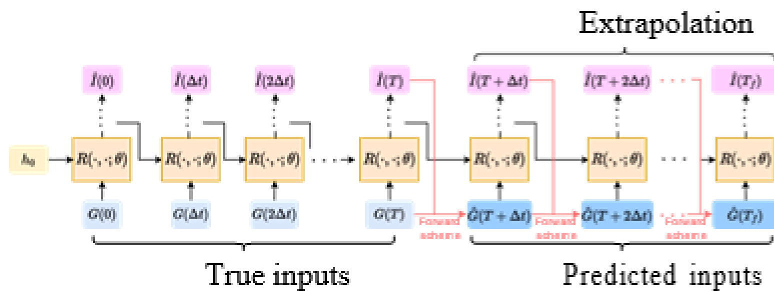


Figure 3: Extrapolation of dynamics using the RNN model. The training phase will output predictions {Iˆ(iΔt)}T . Using Iˆ(T), we can use a forward numerical method to obtain the numerically extrapolated G(T + Δt). From here, we can recursively feed the numerically extrapolated G(T + Δt) into the model to obtain the prediction for Iˆ(T + Δt).

As an example, if we employ the forward Euler scheme to solve the differential equation, then for IDE 2, we have

$$G(T + (i + 1)\Delta t) = G(T + i\Delta t) + \Delta t[F (T + i\Delta t, G(T + i\Delta t)) + \hat{I}(T + i\Delta t)]$$

where Iˆ(T +iΔt) is the output of the RNN generated recursively by feeding in the input G(T + iΔt) as illustrated in Figure 3. The forward scheme for any multi-step method can be similarly derived. The specific model parameters, such as the hidden size of each layer, will be discussed in Section 3 per IDE considered. In this work, we utilize a PyTorch backend to implement all of our NN models.

Data preparation and loss functions

To generate the training data, we solve the IDE numerically using an Adams-Bashforth third-order method (AB3) and Simpson's rule to approx- imate the collision integral I(t). This generates a time series data that is recorded in an interval of [0, T], discretized by Δt, eventually forming dataset consisting of (

Theweaknessesofthebenchmarkmethodscomefromthefactthatthese

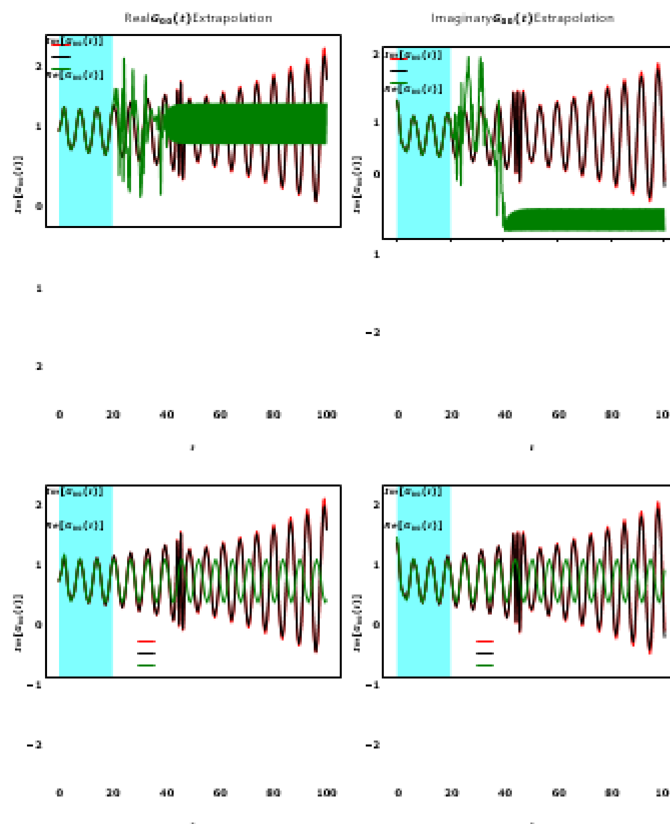


Figure 4: Sample trajectories of G00(t), I00(t), selected from the database created by solving Eqn 4 with α1, α2 [1, 20] [1, 20], σ 1, 2, 3, 4, 5 , and β = 1. Subsequently, we use the same database to do multi-trajectory training of the RNN model.

values that are considered in our applications, we will further split the real and imaginary parts of G(t) and I(t) when we generate the input sequence and compare Iˆ(t) with I(t). This has to be done since most popular ML frameworks such as PyTorch(Paszke, 2019) can only do real-valued arith- metics when training the NN. It is chosen to adjust the parameter values to minimize the mean-squared error (MSE) function:

$$f(I, \hat{I}; \theta) := \frac{1}{N}\sum_{i=0}^{N} (I(i\Delta t)\ \hat{I}(i\Delta t))^2, \qquad (3)$$

where Iˆ(iΔt) is the predicted collision integral generated by the RNN and I(iΔt) is the ground truth. The parameters are learned by propagating the gradients of each hidden state's inputs.

Training method

As we mentioned before, for all the numerical simulations considered in this paper, the RNN training is performed in a small time window (T rela- tively small), and the trained model is used for long-time (Tf) extrapolation in which Tf T , see Figure 3. We employ two training strategies to learn the integral operator I(t):

1. (Single trajectory training) For this case, the RNN is trained using a single trajectory dataset {(G(iΔt), I(iΔt)}T . The numerical advan- tages of employing single trajectory training stem from its relatively low computational cost throughout the optimization process. Accord- ingly, since the single trajectory data corresponds to a specific choice of the steaming term F (G(t), t), the optimized RNN normally yields bad generalization results if we use the learned RNN to solve the IDE with different F (G(t), t) terms.
2. (Multi-trajectory training) In contrast with the first case, the RNN can also be trained using batch training techniques, where the input of the neural network are multiple trajectories generated by choosing differ- ent steaming terms F (G(t), t). The training cost is obviously higher but the obtained RNN model has greater generalizability and hence can be used to predict dynamics for IDE with new steaming terms. From the operator-learning point of view, the multi-trajectory training method is preferred since the enlarged dataset contains different input- output G(t) and I(t), which essentially provides more test functions for learning the mapping I : G(t) → I(t).

Computational cost

All computations are performed using the Perlmutter cluster. The login node has one AMD EPYC 7713 as the CPU and one 40GB NVIDIA A100 as the GPU. For Eqn 4, a typical training with input sequence length 2000 across 750 epochs for an RNN with 2 LSTM layers and hidden size 64 would take approximately 2 hours for the multi-trajectory training and 30 min- utes for the single trajectory training. Under the same setting, for Eqn 5, it would take approximately 30 minutes for a multi-trajectory training and approximately 15 minutes for a single trajectory training. After the training, in the extrapolation phase, the computational time would be spent on the evaluation of the multi-step forward scheme and for RNN to generate new output I(T + iΔt).

2. Numerical results

To demonstrate our method, we first consider a nonlinear complex-valued IDE given by:

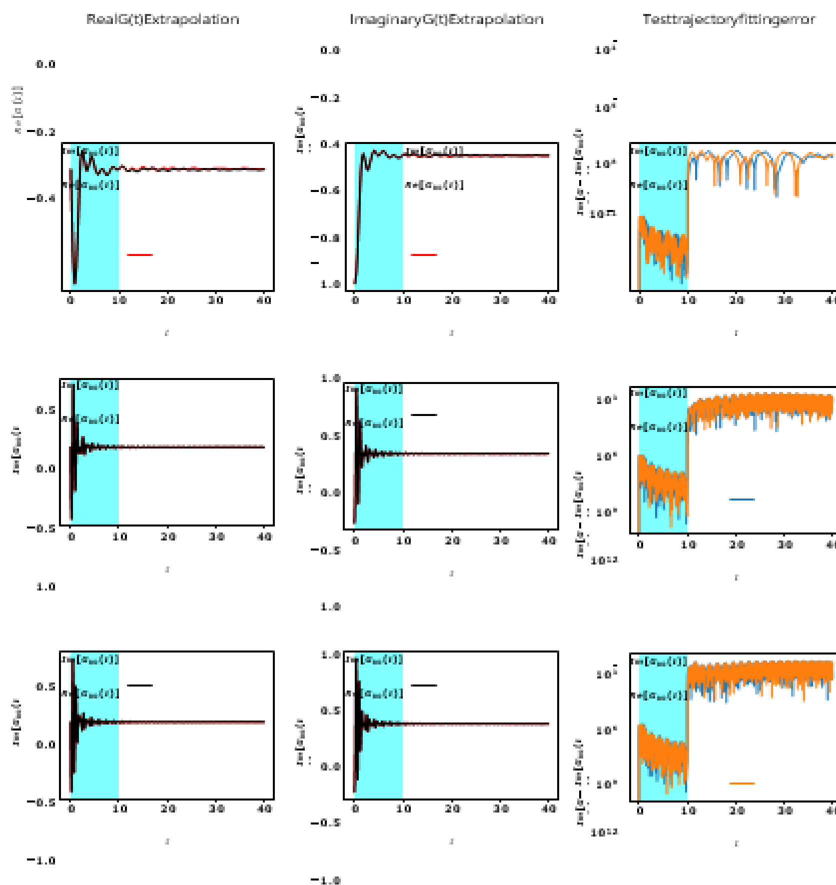


Figure 5: Single trajectory learning of IDE 4 where the RNN is trained on $\alpha_1 = 10$, $\alpha_2 = 15$, $\sigma = 2$ and $\beta = 1$ and tested by extrapolating the dynamics up to T = 120 into the future. The shaded window represents the training regime. The black curve represents the true dynamics. The red curve represents the simulated dynamics using the learned Iˆ(t) produced by the RNN.

For Eqn 4, we will employ both the single trajectory training and multi- trajectory training approaches to approximate the integral operator I(t) := t K(t s)G(s)ds and then combine the learned I(t) and AB3 as the ODE solver to obtain long-time trajectories. Before we present the training details, we note in advance that the RNN training results are benchmarked in three ways:

1. We compare the learned and extrapolated G(t) with a highly accurate numerical solution of IDE 4 and show the accuracy in the fitting region and RNN's predictability of the future dynamics. Due to the fact that the RNN is designed to learn the integral operator I(t) = t K(G(t s))G(s)ds, we also expect this predictability to be generally valid for IDE 4 with different A(t) terms in the multi-trajectory case.
2. We further compare our learning strategy, i.e. learning the map I(t) = t K(ts)G(s)ds, with two existing dynamics extrapolation methods. For the first one, we consider a direct learning strategy that uses an RNN with the same architecture while the output now changes to be

Multi-trajectory learning of IDE 4 where the RNN is trained on $\alpha_1, \alpha_2 \in [1, 20] \times [1, 20]$, $\sigma \in \{1, 2, 3, 4, 5\}$, $\beta = 1$ and tested on $\alpha_1 = 45$, $\alpha_2 = 45$, $\sigma = 5$ and $\beta = 1$ (First column) and $\alpha_1 = 45$, $\alpha_2 = 35$, $\sigma = 2$ and $\beta = 14$ (Second column). The shaded window represents the training regime. The black curve represents the true dynamics. The red curve represents the simulated dynamics using the learned Iˆ(t) produced by the RNN model.

the next timestep G((i + 1)∆t) value. Accordingly, we modify the loss function (3) as:

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 1, No. 1, January : 2024

$$f(G, G_N^R; \theta) := \frac{1}{N} \sum_{i} (G(i\Delta t) - G_N^R(i\Delta t))^2.$$

and other settings are the same. Secondly, we also compare our re- sults with what was obtained using the dynamical mode decomposi- tion (DMD) (Schmid, 2010, 2011; Kutz, 2016; Yin, 2023, 2022; Reeves, 2023A) approach. With these comparisons, one can clearly see the better generalizability of our RNN model.

3. We calculate the runtime of our simulation and compare it with what was obtained using a standard integro-differential equation solver. This would demonstrate the numerical speedup we gain using the RNN to approximate the collision integral I(t).

The weaknesses of the benchmark methods come from the fact that these

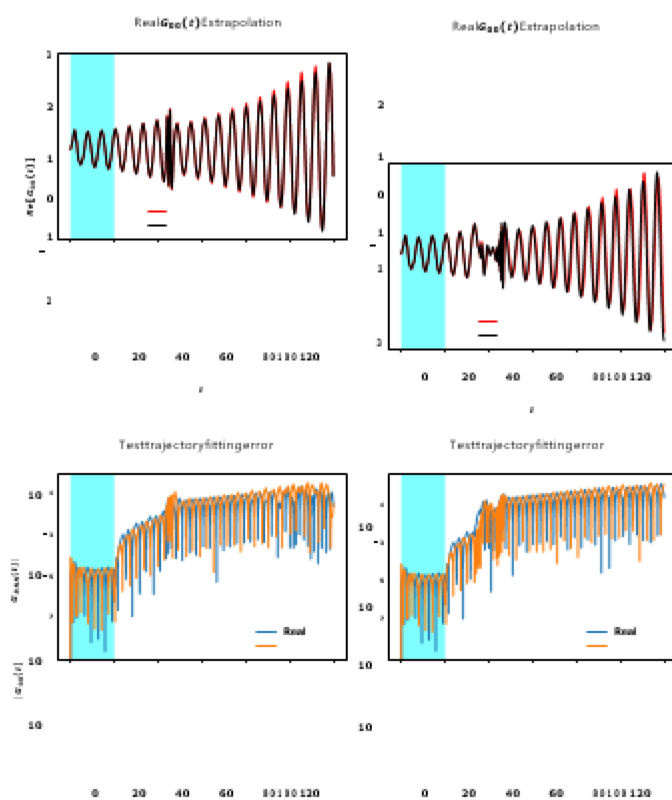


Figure 7: Multi-trajectory learning of IDE 4 where the RNN is trained on α1, α2 ∈ [1, 20] [1, 20], σ 1, 2, 3, 4, 5 , β = 1 and tested on α1 = 45, α2 = 45, σ = 5 and β = 1. The shaded window represents the training regime. The black curve represents the true dynamics. The red curve represents the simulated dynamics using the learned Iˆ(t) produced by the RNN model. The green curve in the top row represents the simulated dynamics using the RNN model that directly predicts G(t). The green curve in the bottom row represents the DMD extrapolation result.

Two approaches rely heavily on the training data used to predict future timesteps. For instance, since DMD constructs the model based on a small window of the dynamics within a specific time frame to extrapolate the future dynamics, if we are to observe a streaming term in Eqn (2) that has forcing applied outside of the training window, the method will fail simply because the training data lacking in this aspect.

Training details

For single trajectory training, we fix the modeling parameters in Eqn 4 to be α1 = 10, α2 = 15, σ = 2, and β = 1. The RNN is trained by feeding in
{G(iΔt)}N for Δt = 0.01 and N = 2000, and then we generate the extrap-
olated trajectory of G(t) up to T = 120 (10000 timesteps into the future).

For multi-trajectory training, the batch dataset is generated by solving IDE 4 with different parameters α1, α2, σ, β. Specifically, we prepare a dataset by

choosing α1, α2 from the lattice grid [1, 20] [1, 20], σ 1, 2, 3, 4, 5 and β = 1. This results in 2000 different trajectories. Plotted in Figure 4 is a reference of our input data G(t) and targets I(t). The displayed result is the first component of the 2 x 2 matrices G(t) and I(t). The dynamic difference between different matrix components is minimal.

The RNN model contains 2 LSTM layers where the hidden size for each layer is 64. The input is an 8-dimensional vector that consists of the flattened
2 2 matrix G(t) decomposed into the real and imaginary parts. Similarly, the output is 8-dimensional, consisting of the real and imaginary parts of the 2 2 matrix collision integral I(t).

Results discussion

The training and testing results are summarized in Figures 5-7. The single trajectory training result is displayed in Figure 5. We see that the RNN model is able to accurately predict the dynamics of the system using only 1 of the total trajectory for training. In particular, the modes and amplitudes of the oscillations match well with the ground truth dynamics. The multi- trajectory training results are shown in Figure 6. The batch training data are obtained by varying the parameters of Eqn 5 to be α1, α2 ∈ [1, 20]×[1, 20], σ ∈
{1, 2, 3, 4, 5}, and β = 1. In the first column of Figure 6, the learned integral
operator I(t) is used to solve IDE 4 for a new set of parameters: α1 =
45, α2 = 45, σ = 5, and β = 1, which is outside of the parameter range of the training dataset. We see that our RNN model is still able to accurately predict the dynamics of this test trajectory, despite the highly oscillatory regime of the test trajectory appearing much later, which is never seen in the dataset. This result is further highlighted in the second column of Figure 6 where the test trajectory corresponds to α1 = 45, α2 = 35, σ = 5, and β = 14, and a large chirp oscillation is created after the training regime. The RNN predicting result still matches well with the true dynamics.

The result of integral operator learning is further compared with what was obtained using the direct learning approach and the DMD method. The testing results are summarized in Figure 7. As we introduced before, the di- rect learning approach used the same RNN architecture and training dataset to learn the solution map G(iΔt) G((i + 1)Δt). The DMD method can only do time extrapolation for a single trajectory therefore the training data is just the G(t), t [0, 20] for fixed parameter values α1 = 45, α2 = 45, σ = 5 and β = 1. We can see clearly from Figure 7 that the integral operator learning strategy significantly outperforms other approaches.

Total simulation time	AB3 + RNN	FE + SR
20	0.8684s	60.3825s
40	1.5028s	166.0921 s
80	2.5220s	475.0468 s
160	4.7412s	1602.866 4s

Table 1: Comparison of wall clock time using the RNN method and a regular IDE solver to extrapolate dynamics with Δt = 0.01 for both methods. Here, total simulation time refers to the final time T used in solving the IDE.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 1, No. 1, January : 2024

where O(nT)A results from the numerical method, and O(nT)B results from querying the RNN. In contrast, using a numerical integration for the integral term and a forward solver to solve IDE 1 has O(n2) scaling due to the integral term needing to be computed at each time step for all the history. We also note that for both the RNN and the FE + SR methods in Table 1 use Δt = 0.01.

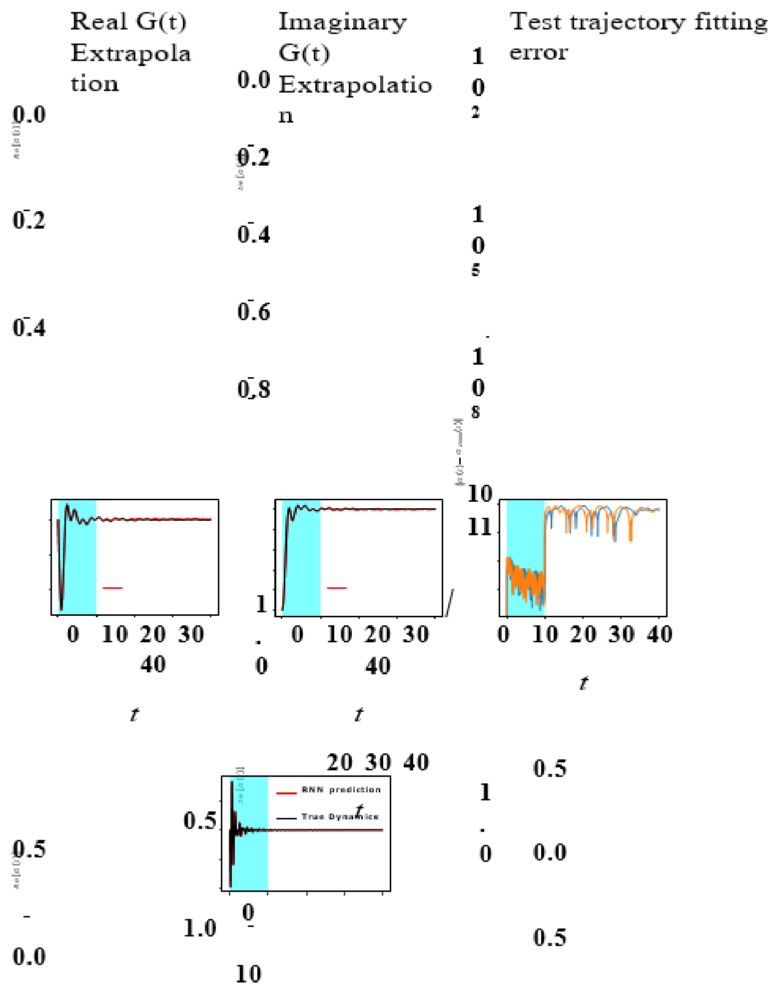
4. Application in quantum dynamics simulation

In this section, we apply the RNN model to numerically solve and extrap- olate the dynamics of Dyson's equation, which is a special class of complex IDEs that is fundamentally important for the study of quantum many-body systems (Stefanucci, 2013). To benchmark our result, we consider an equi- librium Dyson's equation for hopping electrons in the Bethe lattice (Kaye, 2023; Mahan, 2000). When t > 0, the equation of motion reads:

$$i\partial_t G^R(t) = hG^R(t) + \int_0^t c^2 G^R(t-s)G^R(s)ds. \qquad (5)$$

In the context of quantum many-body theory, the time-dependent quantity GR(t) is called the retarded Green's function, which contains important phys- ical information such as the single-particle energy spectrum of the physical system. The memory kernel K(t s) = c2GR(t s) is the self-energy of the system. h, c are the modeling parameters that will be varied when we do multi-trajectory training. Throughout this section, the initial condition GR(0) = i is chosen. According to the analysis by Kaye et al.(Kaye, 2023), Eqn 5 admits the analytical solution:

$$G^R(t) = -ie^{-iht}\frac{J_1(2ct)}{ct}, \quad t > 0,$$



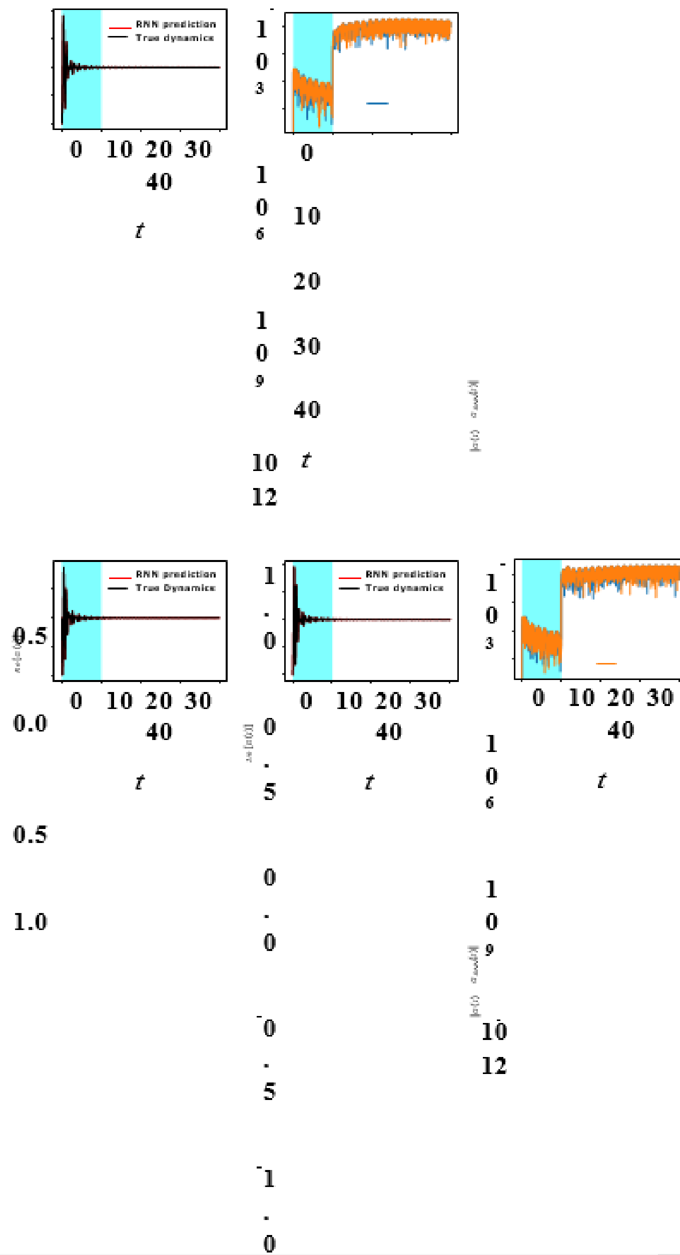


Figure 9: RNN training and extrapolating results for the Dyson's equation 5. The first row displays the single trajectory training result for h = 1, c = 1. The second row is the multi-trajectory training results where the RNN is trained for h [1, 10], c = 1, and tested on h = 8, c = 1. The third row uses the same multi-trajectory model, but tests on h = 11.5, c = 1. where J1(t) is the Bessel function of the first kind. We will use this analytical solution to benchmark the extrapolation results generated by RNN. The training procedures are almost the same as the previous example. The slight differences are summarized in the following subsection:

Training details

For IDE 5, we use an RNN model with 2 layers of LSTM cells and the hidden state size for each layer is 128. The input and output are the same as it was for IDE 4. For the single trajectory training case, we set h = 1 and c = 1 to build the dataset. For the multi-trajectory training, we made a slight modification to the training procedure. We first randomly sample h from the domain [1, 10] for 2000 times, while keeping c = 1. Then for

Industrial Engineering Journal

ISSN: 0970-2555

Volume : 53, Issue 1, No. 1, January : 2024

each epoch, we randomly choose a batch of 10 data pairs G(iΔt), I(iΔt) N from the whole database (possibly with replacement) to optimize the RNN. In this fashion, we create a more robust model since for each epoch, the data pair {G(iΔt), I(iΔt)}N used in optimization is different.

Results discussion

All the training and testing results are summarized in Figure 9. In the first row, we show the single trajectory training result where the data is collected for t   [0, 10] and we use the RNN to extrapolate the same trajectory up to T = 40 (3000 timesteps into the future). The second row shows the multi-trajectory training results where the parameters for the test trajectory are set to be h = 8, c = 1. Note that h is chosen from the sampling domain [1, 10] but is not chosen as a parameter to be used in the training database. The third row is for the same multi-trajectory training while showing an out-of-sampling domain example where h = 11.5, c = 1.

As we can see, both the single trajectory training and the multi-trajectory training yield precise predictions of the future-time dynamics of the Dyson's equation. Moreover, the RNN also predicts the correct asymptotic behavior of G(t), i.e., G(t)   0 as t   . All these findings are consistent with what we found in the previous example. In comparison, we also see more accurate and robust results in the multi-trajectory case, both visually and in terms of the error plots. Since we have used many trajectories in training, the RNN is able to learn a more concrete mapping from G(t)   I(t). This leads to better predictability of the RNN on unseen dynamics, as well as greater generalizability to unseen system parameters, as shown in the final row, where h = 11.5 [1, 10].

5. Conclusion

In this paper, we introduced an RNN-based machine-learning technique that uses LSTM as the basic modeling module to learn the nonlinear integral operator in an IDE. Such a learning scheme allows us to turn an IDE to an ODE that can be solved efficiently by a standard ODE solver for a large t. We showed that a more effective way to learn a nonlinear integral operator is to include multiple training trajectories generated from different solutions of IDEs defined by different streaming terms within a small time window in the aining data. The effectiveness of this approach was demonstrated with two test examples. The generalizability of the learned operator was demonstrated by using the learned map to predict the dynamics of a new IDE that is driven by a completely different streaming term that is outside of the parameter range of the training data. Moreover, since the RNN consists of layers of function composition, it is almost immediate to generate a next timestep collision integral I(t) given the input. This leads to an overall O(nT) scaling of computational cost (the same as an ODE solver) when we use the RNN to solve the IDE, in contrast with the O(n2) scaling of a regular IDE solver. Due to the scalability of the RNN architecture, we expect that the methodology can be generalized and used to solve high-dimensional IDEs, such as the Kadanoff-Baym equations in nonequilibrium quantum many-body theory.

6. CRediT authorship contribution statement

Hardeep Bassi: Methodology, Validation, Software, Data Curation, Writ- ing – Original draft preparation Yuanran Zhu: Methodology, Validation, Data Curation, Software, Writing- Original draft preparation. Senwei Liang: Methodology, Validation, Visualization, Software, Writing – Original draft preparation Jia Yin: Validation, Writing – Reviewing & Editing Cian C. Reeve : Methodology, Validation, Writing – Reviewing & Editing Vojtech Vlcek: Methodology, Validation, Writing – Reviewing & Editing, Supervi- sion, Project administration, Funding acquisition Chao Yang: Methodology, Validation Writing- Reviewing and Editing, Supervision, Project administra- tion, Funding acquisition

7. Declaration of competing interest

The authors have no competing interests.

8. Data availability

The authors will provide data upon request.

9. Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research
and Office of Basic Energy Sciences, Scientific Discovery through Advanced Computing (SciDAC) program under Award Number DE-SC0022198 and contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Sci- ence User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award BES-ERCAP0020089 and ASCR-ERCAP-m1027.

References

Bohner, M., and Tun¸c, O. (2022). Qualitative analysis of integro-differential equations with variable retardation. Discrete & Continuous Dynamical Systems-Series B, 27(2).
Cohen, G., and Rabani, E. (2011). Memory effects in nonequilibrium quan- tum impurity models. Physical Review B, 84(7), 075150.
Chen, X., Duan, J., and Karniadakis, G. E. (2021). Learning and meta- learning of stochastic advection–diffusion–reaction systems from sparse measurements. European Journal of Applied Mathematics, 32(3), 397-420.
Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. Advances in neural information pro- cessing systems, 31.
Cho, K., Van Merri¨enboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
Can, E. F., Ezen-Can, A., and Can, F. Multilingual sentiment analy- sis: an RNN-based framework for limited data (2018). arXiv preprint arXiv:1806.04511.
Debnath, L., and Debnath, L. (2005). Nonlinear partial differential equations for scientists and engineers (pp. 528-529). Boston: Birkhauser.
Harlim, J., Jiang, S. W., Liang, S., and Yang, H. (2021). Machine learning for prediction with missing dynamics. Journal of Computational Physics, 428, 109922.
V.