



OPEN

# Effective weight optimization strategy for precise deep learning forecasting models using EvoLearn approach

Jatin Bedi<sup>1</sup>, Ashima Anand<sup>1</sup>, Samarth Godara<sup>2</sup>, Ram Swaroop Bana<sup>3</sup>, Mukhtar Ahmad Faiz<sup>3,4</sup>, Sudeep Marwaha<sup>2</sup> & Rajender Parsad<sup>2</sup>

Time series analysis and prediction have attained significant attention from the research community in the past few decades. However, the prediction accuracy of the models highly depends on the models' learning process. In order to optimize resource usage, a better learning methodology, in terms of accuracy and learning time, is needed. In this context, the current research work proposes EvoLearn, a novel method to improve and optimize the learning process of neural-based models. The presented technique integrates the genetic algorithm with back-propagation to train model weights during the learning process. The fundamental idea behind the proposed work is to select the best components from multiple models during the training process to obtain an adequate model. To demonstrate the applicability of EvoLearn, the method is tested on the state-of-the-art neural models (namely MLP, DNN, CNN, RNN, and GRU), and performances are compared. Furthermore, the presented study aims to forecast two types of time series, i.e. air pollution and energy consumption time series, using the developed framework. In addition, the considered neural models are tested on two datasets of each time series type. From the performance comparison and evaluation of EvoLearn using a one-tailed paired *T*-test against the conventional back-propagation-based learning approach, it was found that the proposed method significantly improves the prediction accuracy.

**Keywords** Time series prediction, Genetic algorithm, Back-propagation, Neural models, Learning optimization

The process of extracting meaningful patterns/information from time-indexed data, termed “Time series analytics and modelling”<sup>1</sup>, has gained significant attention from the research community in recent years. It covers a broad spectrum of aspects, including time-series clustering, classification, regression/prediction and visualization. Time series analysis and prediction have shown great potential in nearly all application domains, including healthcare, utility services, transportation, manufacturing, financial services and agriculture<sup>2-5</sup>. Moreover, in the past few years, forecasting air pollution and energy consumption have been focal points for researchers as the insights from these domains help governments in decision-making on an extensive scale<sup>6,7</sup>.

With the precise forecasting of air pollution, it becomes more manageable to address and mitigate the risks of air pollution and secure a safe level of pollutant concentration in the target region. It also helps estimate risks to the atmosphere and the climate caused by inferior air quality standards. Precise forecasting can also ease day-to-day planning activities, avoid locations with high alert zones, and execute effective pollution control measures.

Furthermore, global energy consumption has also been rising every year. Governments and power companies need to explore models to forecast and plan energy use precisely, as it acts as the backbone of nationwide economies. Monitoring the state of electrical power loads, especially in the early detection of unusual loads and behaviours, is essential for power grid maintenance and energy theft detection.

In this direction, the presented study is aimed at forecasting air pollution time series and energy consumption time series corresponding to a total of three States/UTs of India. A wide variety of statistical, machine learning and deep neural models have been employed for the modelling and prediction tasks in the listed domains<sup>8-11</sup>.

<sup>1</sup>Thapar Institute of Engineering And Technology, Patiala, Punjab, India. <sup>2</sup>ICAR-Indian Agricultural Statistics Research Institute, New Delhi, India. <sup>3</sup>ICAR-Indian Agricultural Research Institute, New Delhi, India. <sup>4</sup>Afghanistan National Agricultural Sciences and Technology University, Kandahar, Afghanistan. ✉email: samarth.godara@icar.gov.in; maf\_maher@yahoo.com; maf.maher.anastu@gmail.com

In the earlier studies, statistical/traditional techniques<sup>12–15</sup> including Autoregressive (AR), Autoregressive with moving average (ARMA), Autoregressive integrated moving average (ARIMA), and seasonal autoregressive integrated moving average (SARIMA) models have been widely utilized for the time series modelling tasks. Despite their vast popularity, these models have been found less useful at improvising the generalization capabilities of the prediction approaches.

With the latest advancements in information technology and sensors, machine learning (ML) models<sup>16–20</sup>, including linear and polynomial regression models, support vector machines, decision trees, k-nearest neighbours, bayesian networks and artificial neural networks (ANN), were adopted for achieving the improved prediction accuracy. These ML models have achieved good generalization capabilities but could not efficiently capture highly non-linear featural aspects and sudden variations of the time-series data. To deal with this, various deep neural network models were introduced in the past to automatically capture nonlinear and chaotic hidden features of the temporal data. Several research studies have employed these different neural models<sup>21–26</sup>, including recurrent neural network (RNN), convolutional neural network (CNN), deep neural network (DNN), gated recurrent unit (GRU) and long short term memory network (LSTM) for the time-series modelling/prediction tasks. Deep neural models are very potent at capturing different featural and non-linear aspects of the time series. However, determining optimal architectural parameters of the neural network models in terms of the number of layers, activation functions, and the number of neurons per layer is a complex task. These architectural parameters have a direct impact on the target model's accuracy and generalization capabilities.

To deal with the problem of tuning the hyperparameters of the neural-based models, one field that has attained great interest from the research community is integrating evolutionary algorithms with learnable models. This has greatly helped researchers in achieving improved prediction accuracy on data belonging to different application domains. In the past decades, numerous research studies have successfully explored different statistical, deep neural models and hybrid techniques for time-series forecasting on data of various domains like transportation, rainfall estimation, financial service, utility services etc.

Ho and Xie<sup>27</sup> presented an approach to investigate the use of auto-regressive models at system reliability forecasting. The performance comparison is done using the conventional equations driven by the Duane model. Dastorani et al.<sup>14</sup> examined different time-series models to forecast monthly rainfall events. Statistical methods, including AR, ARMA, ARIMA, and SARIMA were implemented to do accurate rainfall forecasting at nine different stations in Northeast Iran. Based on the experimental results, the authors concluded that the best time-series models could change based on the input data variability.

Nicholas et al.<sup>17</sup> proposed a support vector regression-based approach to capture non-linear and non-stationary time-series variations. Also, the authors stated some critical challenges to be considered while employing the proposed SVR approach for the prediction task, such as selecting optimal hyper-parameters and performance metrics. Martinez et al. deployed a k-nearest neighbour algorithm to estimate time-series. The performance of the proposed approach is assessed on the N3 competition dataset. Cuautle et al. introduced an artificial neural network-based method for estimating chaotic signals. In addition to statistical and machine learning models, sequential deep learning prediction models are also well utilized for the time-series prediction task. Connor et al.<sup>28</sup> proposed a robust learning algorithm to improve the stability of the recurrent neural network models. The proposed method was based on dealing with outliers present in the data. Ruan et al.<sup>26</sup> developed a deep learning-based approach for load-balancing and adaptive scheduling tasks in real-time systems. The approach has implemented an LSTM model for the specified task.

Soltani<sup>29</sup> proposed a way to combine wavelet decomposition and neural models for targeting higher accuracy. Shi et al.<sup>30</sup> integrated the ARIMA model with ANN and SVR to efficiently estimate wind speed and energy load time-series. Xu et al.<sup>31</sup> blended SARIMA models support vector regression models to estimate the demand of the aviation industry. From the performance comparison with conventional models, the authors stated that hybrid models provide better prediction accuracy. Du et al.<sup>21</sup> provided a deep neural network-based hybrid model for air pollution forecasting. The proposed hybrid approach (CNN + LSTM) captures the Spatial-temporal aspect of the series parameters to improve prediction accuracy.

Sheikhan and Mohammadi<sup>32</sup> proposed a Genetic Algorithm (GA)-based approach for electricity load forecasting. The authors employed ant-colony optimization algorithm for the feature selection task. Experimental results achieved on the real-world energy consumption dataset showed that the GA-ant colony optimization integrated approach provides better results than the traditional MLP model. Dong et al.<sup>33</sup> proposed a deep learning and k-nearest neighbours-based approach for load time series forecasting. Initially, the k-nearest neighbour algorithm was implemented to capture the historical featural aspect of the input series. Subsequently, a GA was employed to achieve multi-objective optimization by determining the smallest category of k-nearest neighbour and the highest accuracy. In a similar context, Anh et al.<sup>34</sup> utilized Bayesian optimization and GA for hyper-parameters optimization and feature selection (respectively) in a time-series prediction problem/task.

Niska et al.<sup>35</sup> identified that defining the architectural model for non-linear chaotic time-series is a challenging task. So, the authors, in their research study, proposed a GA-based approach to decide the architectural details of the multi-layer perceptron model. The performance evaluation of the model is done on a real-world air quality dataset. Puneet et al.<sup>36</sup> proposed a GA-based faster approach to discover the hyper-parameters of the DNN, RNN and CNN models. Compared to the existing hyperparameters optimization approach, which optimizes each hyperparameter in isolation, the proposed approach targets simultaneous optimization of multiple hyper-parameters. The parameter estimation speed of the proposed approach is compared with the Grid search method, and it is found that the proposed GA-based approach is 6.71 times faster than the Grid search method.

Li et al.<sup>37</sup> employed GA to determine the optimal value of the number of trainable parameters for training a convolutional neural network. The proposed approach converged with 97% precision in less than 15 generations. In a similar context, Yuan et al.<sup>38</sup> proposed a novel GA-based hierarchical evolutionary strategy for deciding the hyper-parameters of graph neural networks. The authors presented a fast evaluation approach for graph neural

networks using the early stopping criteria. The performance evaluation of the approach is done on two deep-graph neural network models. Shahid et al.<sup>39</sup> integrated a GA with the LSTM model (GLSTM) for the sequential modelling or prediction task. In the proposed approach, two parameters, window size and number of neurons in LSTM layers, were determined by using the GA. The prediction results were compared with the support vector machine model, and it was found that GLSTM provides 6% to 30% improvement in the prediction accuracy.

Ahmet kara<sup>40</sup> developed a hybrid approach for influenza outbreak forecasting using the LSTM model and GA. The author employed GA to obtain the optimal value for many LSTM hyper-parameters such as the number of epochs, number of LSTM layers, size of units in each LSTM layer, and window size. The experimental results of the proposed hybrid are shown on a dataset collected from the Centers for Disease Control and Prevention (CDC), USA. Huang et al.<sup>41</sup> proposed a hybrid approach integrating variational mode decomposition, GA and LSTM for financial data forecasting. In this research study, GA was utilized to determine the optimal parameters of the variational mode decomposition method. Cicek and Ozturk<sup>42</sup> integrated ANN with a biased random key GA for the time-series forecasting task. In the study, biased random key GA was implemented to determine the optimal value of neural model parameters, namely the number of hidden neurons, hidden neurons' bias values, and the connection weights between nodes. The performance evaluation of this approach is done by comparing prediction results with conventional ANN model with back-propagation and Support vector machines and ARIMA models.

The research by Pin et al.<sup>43</sup> concentrates on enhancing predictive biomedical intelligence through transfer learning from biomedical text data for specific downstream tasks. In contrast, the proposed EvoLearn approach focuses on improving and optimizing the learning process of neural-based models for time series analysis and prediction. Moreover, the EvoLearn approach integrates GA with back-propagation for model training, focusing on selecting the best components from multiple models during the training process. Conversely, the existing research utilizes transfer learning from pre-trained language models (PLMs) and proposes a Stacked Residual Gated Recurrent Unit-Convolutional Neural Networks.

Another similar method employing the artificial bee colony (ABC)-integrated extreme learning machine (ELM) approach is introduced by Yang and Duan<sup>44</sup>. In this work, the ABC algorithm is applied to optimize the weights of a simpler model, the ELM, which inherently possesses fewer parameters. In contrast, the EvoLearn approach diverges from the existing method, primarily regarding the optimization methodology employed. In EvoLearn, GA is utilized to optimize the weights of DL models. Additionally, EvoLearn adopts an iterative alternation between GA and the backpropagation algorithm, which enhances optimization speed. This iterative process facilitates a more rapid convergence towards optimized weights. In contrast, the ABC-integrated ELM approach does not incorporate this iterative alternation, potentially resulting in a slower optimization process.

Another work by Wang et al.<sup>45</sup> introduced the adaptive differential evolution back propagation neural network (ADE-BPNN). The ADE-BPNN approach combines the adaptive differential evolution (ADE) algorithm with the backpropagation neural network (BPNN) to improve forecasting accuracy. In contrast, the proposed EvoLearn approach focuses on optimizing the parameters of pre-designed neural network architectures to enhance their predictive performance across various domains. Moreover, in the existing approach, ADE is initially employed to search for the global initial connection weights and thresholds of BPNN, followed by the thorough optimization of weights and thresholds using BPNN itself. Overall, while both approaches seek to enhance forecasting accuracy, they employ different optimization algorithms and strategies. EvoLearn focuses on optimizing the weights of pre-designed deep learning models, while ADE-BPNN combines ADE with BPNN to automatically design the initial connection weights and thresholds, thereby improving the forecasting performance of BPNN.

The approach introduced by Jalali et al.<sup>46</sup> focuses on electricity load forecasting using a deep neuroevolution algorithm to design CNN structures automatically. Specifically, it utilizes a modified evolutionary algorithm called enhanced grey wolf optimizer (EGWO) to optimize the architecture and hyperparameters of CNNs for improved load forecasting accuracy. This method aims to address the complexity of electricity load forecasting and the challenges in CNN architecture design by employing a novel optimization technique. While both approaches (EGWO-based method and EvoLearn) involve optimization algorithms, they differ in their specific focus and methodology. The EvoLearn approach aims to optimize the weights of pre-designed deep learning models, while the mentioned approach focuses on automatically designing CNN structures using a neuroevolution algorithm. Additionally, the mentioned approach specifically targets electricity load forecasting, whereas EvoLearn can be applied to various prediction tasks in different domains.

Based on the literature review, it is evident that the proposed EvoLearn method represents various novel contributions to the field. While previous research has commonly employed evolutionary algorithms for hyperparameter tuning, only a limited number have utilized them for tuning model weights, as we have done. Furthermore, in those studies where model weights are tuned akin to our approach, the utilization has typically involved separate stages of evolutionary optimization and backpropagation, rather than simultaneous and iterative integration as demonstrated in our methodology. This distinctive approach accelerates the optimization process significantly, distinguishing our method as notably innovative compared to prior research endeavors.

EvoLearn integrates GA with back-propagation by initially training neural networks using back-propagation to establish a population of diverse models with different weights. The GA then optimizes these weights through selection, crossover, and mutation, focusing on minimizing a combined error from both training and validation sets. This hybrid approach allows for a broader exploration of the weight space, preventing overfitting and improving generalization. Compared to conventional back-propagation alone, EvoLearn enhances model robustness and performance by leveraging evolutionary strategies to escape local minima.

Furthermore, the proposed strategy aids neural models in achieving better accuracy while avoiding overfitting. The major research contributions of the current research study are summarized as follows:

- The present work introduces a methodology to train models to their greater potential and achieve better model accuracy. Moreover, the proposed fitness function used in the presented technique keeps the models from over-training.
- The proposed approach provides a mechanism to generate optimal weight parameters. Best combinations of network weights are produced by integrating GA with the back-propagation.
- The presented work introduces an evolutionary algorithm-based model-selection technique against the conventional method of selecting the best model among the set of trained learnable models.
- With the integration of the GA along with the back-propagation algorithm, it was found that the trained models attain saturation earlier than the conventional learning technique. Therefore, the proposed approach can be used to train neural-based models faster.

## Results

### Dataset description

To assess the performance of EvoLearn in comparison with the conventional Back-Propagation algorithm, in this study, several neural-based forecasting models are developed on two different datasets. The details of used datasets are as follows:

- *Air pollution dataset* Increasing air pollution is of critical concern for each nation. Early prediction of rising air pollution (PM2.5 concentration) is critical for proactively mitigating the adverse effects. In the current study, we have employed the proposed EvoLearn approach to predict the PM2.5 concentration of two major states (Punjab and Delhi) in India.
  - *Punjab* Punjab has been well known for its agricultural activities and healthy lifestyle. However, with the increase in industrial activities and traffic, there has been a considerable rise in air pollution over the past few years. We gathered data on Punjab's air pollution ( $AP_{DT1}$ ) for three years, from January 2018 to January 2021. The dataset is sampled at an interval of 24 hours.
  - *Delhi* Delhi is the capital of the country India. For the past few years, it has been suffering from a very high concentration of air pollution. The Delhi government has taken many initiatives to control the level of air pollution in the capital. In this study, we have collected the air pollution dataset of Delhi ( $AP_{DT2}$ ) for a period of four years, starting from January 2017 to January 2021, with a sampling rate of 24 hours.
- *Energy consumption dataset* Providing sufficient energy supply to all the needy areas is a top priority for each nation. In this context, predicting timestamps ahead of energy demand can help government and private agencies coordinate many activities, such as generation plant scheduling, capacity planning, transmission network optimization etc. The current study gathers data from two Indian states (**Haryana** ( $E_{DT1}$ ) and **Punjab** ( $E_{DT2}$ )) to accurately predict their future energy demand. The data for both states is collected every 24 hours for a total of seven years, starting in January 2014 and ending in April 2021.

### Performance evaluation

#### *Hyper-parameters selection*

The learning efficiency and accuracy of any prediction model highly depend on the model's architectural parameters, also known as hyper-parameters. There are several hyper-parameters related to the neural network models used in the present study, like the number of layers in the models, the number of neurons per layer, the type of activation functions, the number of iterations and many more. There exist many strategies to estimate the optimal hyper-parameters value, such as Random search, Grid search and Bayesian optimization. The current research study implements a Grid search strategy for the hyper-parameters tuning task. The method works by initially defining a set of possible values corresponding to each hyper-parameter available for a target neural model. Subsequently, we train a model for each possible combination of hyper-parameters defined and related to the target model. Later, the accuracy of each model is evaluated separately using several performance metrics. Finally, we select the hyper-parameters corresponding to the model with the best prediction accuracy.

#### *Evaluation procedure for comparing learning performance*

In order to have a fair performance estimation of the conventional method for weight optimization and the proposed EvoLearn approach, the below-listed training strategy is adopted.

- *The procedure of model selection for the evaluation of models trained using the conventional method:* Optimal hyperparameters were first determined using the grid search method to select the best model developed with the conventional learning technique. Moreover, twenty models were trained separately for various numbers of epochs using the obtained hyperparameters. Later, the best-performing model was chosen based on its MSE performance on the testing dataset for the observations.
- *The procedure of model selection for the evaluation of models trained using the EvoLearn method:* Optimal hyperparameters were first determined using the grid search method to select the best model developed with the proposed learning technique. An initial population of twenty models (weight matrices) was generated using the obtained hyperparameters. Each model present in the population was trained for various epoch numbers (represented by 'x'), with two GA cycles after every five epochs. The best solution obtained from the

last generation was chosen for the observations. The values of 'x' used in the development of MLP, DNN4, DNN7, CNN, RNN, and GRU-based models are 150, 150, 200, 250, 150, and 200, respectively.

## Results comparison and prediction visualization

Six conventional and deep neural network models, namely MLP, DNN4 (DNN with four hidden layers), DNN7 (DNN with seven hidden layers), CNN, RNN, and GRU, were used to estimate the generalization capability and performance improvement achieved by employing the proposed EvoLearn approach. These prediction models were trained using the two different mechanisms. Firstly, the traditional back-propagation-based training was implemented to build prediction models on the four datasets listed in the "Results" Section. Secondly, the models were trained using the proposed EvoLearn (back-propagation integrated GA) approach on the same datasets. Lastly, the prediction performance achieved by using both mechanisms was compared in terms of two different metrics listed in the "Methods" section.

## MSE and MAE comparison

In the current research study, we have trained six time-series prediction models corresponding to each training approach, namely the BackPropagation Approach and EvoLearn Approach. These models are trained on four datasets belonging to two different application domains (Energy Load and Air Pollution). Hence, a total of  $(6 \times 2 \times 4 = 48)$  models have been trained. The performance of these trained models is compared (Tables 1 and 2) in terms of Mean Squared Validation (MSE\_Validation), Mean Squared Testing (MSE\_Test) and Mean Absolute Testing errors (MAE\_Test). In order to test if the proposed approach significantly improved the models' performances, a one-tailed paired T-test was performed over the two groups of each of the performance metrics. The p-value was found to be less than 0.05 in every case, i.e.  $9.43E-05$ ,  $5.60E-06$ , and  $1.38E-07$  for MSE\_Validation, MSE\_Test, and MAE\_Test, respectively. The results reveal that the models' performances were significantly improved by

Model_Name/performance metric	Energy dataset - Haryana ( $E_{DT1}$ )			Energy dataset - Punjab ( $E_{DT2}$ )		
	MSE_Validation ( $\ast 10^{-3}$ )	MSE_Test ( $\ast 10^{-3}$ )	MAE_Test ( $\ast 10^{-2}$ )	MSE_Validation ( $\ast 10^{-3}$ )	MSE_Test ( $\ast 10^{-3}$ )	MAE_Test ( $\ast 10^{-2}$ )
MLP	5.926	9.047	7.148	4.817	7.327	5.672
EvoLearn + MLP	<b>5.835</b>	<b>8.525</b>	<b>6.890</b>	<b>4.385</b>	<b>6.134</b>	<b>5.421</b>
DNN4	6.159	9.239	7.302	5.915	8.552	6.194
EvoLearn + DNN4	<b>5.618</b>	<b>7.876</b>	<b>6.666</b>	<b>4.516</b>	<b>7.090</b>	<b>5.678</b>
DNN7	5.352	7.716	6.569	4.052	5.618	5.223
EvoLearn + DNN7	<b>5.192</b>	<b>6.653</b>	<b>6.120</b>	<b>3.824</b>	<b>5.230</b>	<b>4.757</b>
CNN	9.219	11.308	8.763	8.921	12.088	7.478
EvoLearn + CNN	<b>9.099</b>	<b>11.129</b>	<b>8.146</b>	<b>8.323</b>	<b>11.625</b>	<b>7.053</b>
RNN	5.459	8.070	6.834	4.033	6.130	5.231
EvoLearn + RNN	<b>5.180</b>	<b>7.152</b>	<b>6.445</b>	<b>3.871</b>	<b>5.687</b>	<b>5.030</b>
GRU	<b>4.976</b>	4.982	5.201	3.552	4.278	4.506
EvoLearn + GRU	5.005	<b>4.917</b>	<b>5.184</b>	<b>3.549</b>	<b>4.046</b>	<b>4.307</b>

**Table 1.** Performance comparison results on energy datasets (in terms of MSE and MAE). Significant values are in bold.

Model_Name/performance metric	Air pollution dataset- Delhi ( $AP_{DT2}$ )			Air pollution dataset - Punjab ( $AP_{DT1}$ )		
	MSE_Validation ( $\ast 10^{-3}$ )	MSE_Test ( $\ast 10^{-3}$ )	MAE_Test ( $\ast 10^{-2}$ )	MSE_Validation ( $\ast 10^{-3}$ )	MSE_Test ( $\ast 10^{-3}$ )	MAE_Test ( $\ast 10^{-2}$ )
MLP	2.400	6.556	5.816	2.184	5.457	5.273
EvoLearn + MLP	<b>2.223</b>	<b>4.690</b>	<b>4.687</b>	<b>2.077</b>	<b>5.292</b>	<b>5.020</b>
DNN4	4.730	8.989	7.095	2.711	6.791	5.830
EvoLearn + DNN4	<b>2.073</b>	<b>6.124</b>	<b>5.560</b>	<b>1.891</b>	<b>4.934</b>	<b>4.649</b>
DNN7	<b>1.786</b>	9.307	6.044	1.772	5.843	5.208
EvoLearn + DNN7	1.799	<b>6.636</b>	<b>5.408</b>	<b>1.715</b>	<b>5.344</b>	<b>4.569</b>
CNN	2.342	8.737	6.539	2.808	7.404	5.725
EvoLearn + CNN	<b>2.215</b>	<b>8.199</b>	<b>6.189</b>	<b>2.517</b>	<b>7.160</b>	<b>5.255</b>
RNN	2.314	5.878	5.388	<b>1.742</b>	4.768	4.748
EvoLearn + RNN	<b>2.067</b>	<b>4.799</b>	<b>4.494</b>	1.762	<b>3.946</b>	<b>3.730</b>
GRU	1.948	4.949	4.684	1.494	3.952	3.768
EvoLearn + GRU	<b>1.849</b>	<b>4.632</b>	<b>4.526</b>	<b>1.476</b>	<b>3.874</b>	<b>3.685</b>

**Table 2.** Performance comparison results on air pollution datasets (in terms of MSE and MAE). Significant values are in bold.

using the EvoLearn approach. Figure 1a–d depicts a comparative analysis of the prediction performance achieved by different neural models combined with the conventional and EvoLearn training approaches. From the MAE and MSE graphs shown in Fig. 1, it is evident that the DNN4, DNN7, RNN and GRU models have performed well at capturing the non-linear variations of input time-series datasets. Out of these trained models, GRU models in combination with both model training approaches, have achieved the lowest prediction errors on all four datasets. Furthermore, it can be observed that a significant performance improvement has been gained by all neural models when trained in amalgamation with the EvoLearn approach compared to conventional back-propagation-based training. Hence, it can be concluded that the proposed EvoLearn approach helps neural models in achieving better prediction performance by learning better during the training phase.

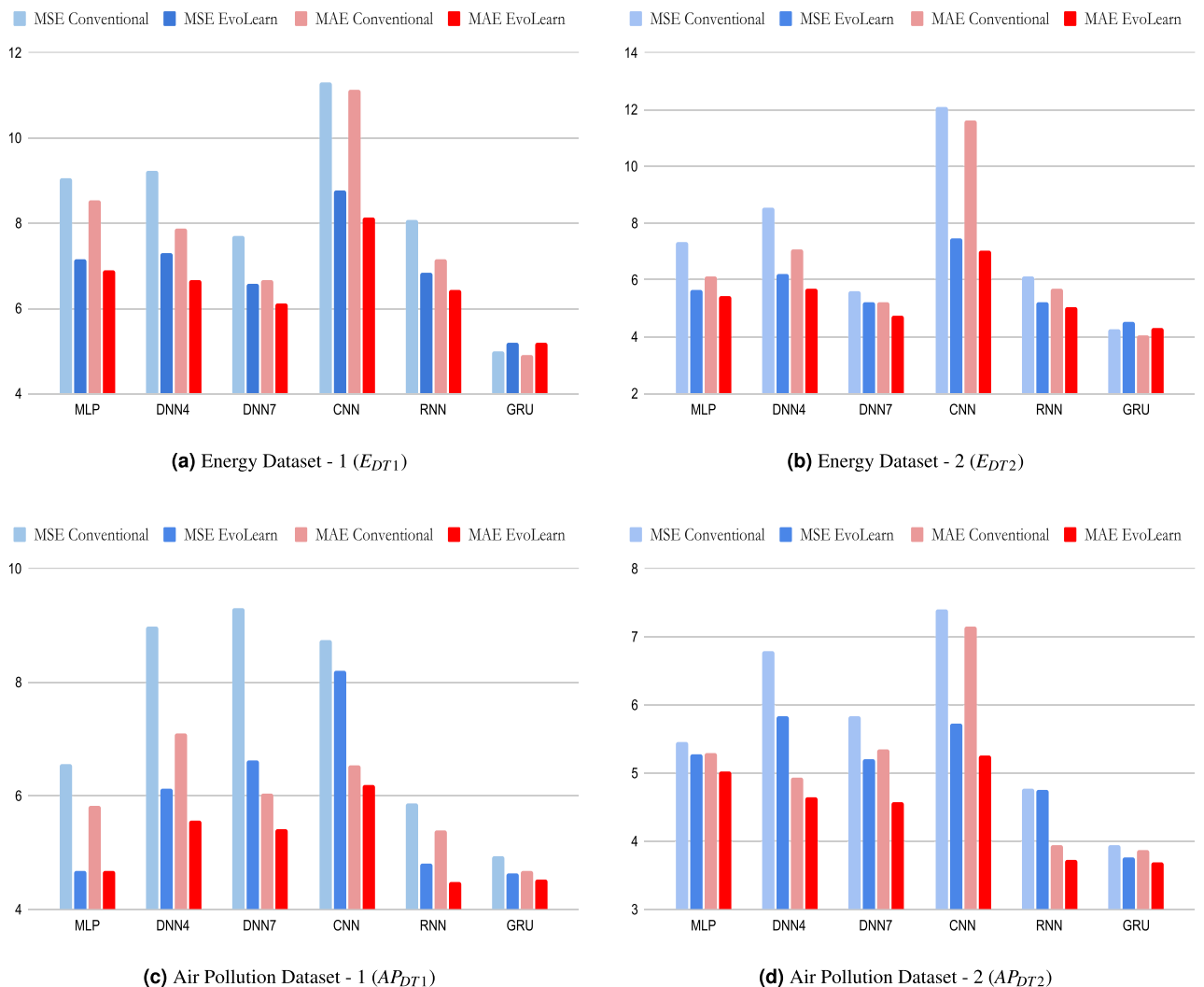
**Normalized difference MSE and MAE**

In order to measure the performance improvement achieved by combining the proposed EvoLearn approach with different neural models, we have proposed the following two performance measures:

- *Normalized difference mean squared error* It evaluates the normalised difference between the mean squared error value obtained by training neural models using the traditional and EvoLearn approaches. Mathematically, it is given as:

$$Diff\_Mean\_Squared\_Error(DMSE) = MSE_{EvoLearn} - MSE_{conventional}, \tag{1}$$

$$Norm\_DMSE_i = \frac{DMSE_i - \min(DMSE)}{\max(DMSE) - \min(DMSE)}, \tag{2}$$



**Figure 1.** Models’ performances (MSE and MAE) on various datasets, trained with the EvoLearn approach and the conventional back-propagation approach.

- where  $MSE_{EvoLearn}$  and  $MSE_{conventional}$  represent the MSE value of a neural model trained using the EvoLearn approach and the Back-propagation approach, respectively.
- **Normalized difference mean absolute error** It evaluates the normalised difference between the mean absolute error value achieved by training neural models using the traditional and EvoLearn approaches.

$$Diff\_Mean\_Absolute\_Error(DMAE) = MAE_{EvoLearn} - MAE_{conventional}, \tag{3}$$

$$Norm\_DMAE_i = \frac{DMAE_i - \min(DMAE)}{\max(DMAE) - \min(DMAE)}, \tag{4}$$

where  $MAE_{EvoLearn}$  and  $MAE_{conventional}$  represent the MAE value of a neural model trained using the EvoLearn approach and back-propagation approach, respectively.

The current research work involves calculating the above-listed metrics for all neural learning models trained in this study. These normalized performance parameters are calculated on two datasets used in the study, and the corresponding results are depicted in Fig. 2. From the Figure, it can be seen that the integration of the EvoLearn approach has contributed significantly to improving the efficiency of all neural models. However, the highest performance improvement has been achieved in the DNN4 neural network model on all four datasets. Following the same pattern, RNN and DNN7 have also seen a notable reduction in prediction errors by employing the proposed EvoLearn approach. Hence, it can be stated that the proposed method can be efficiently adopted to improve the prediction performance of the time-series prediction models.

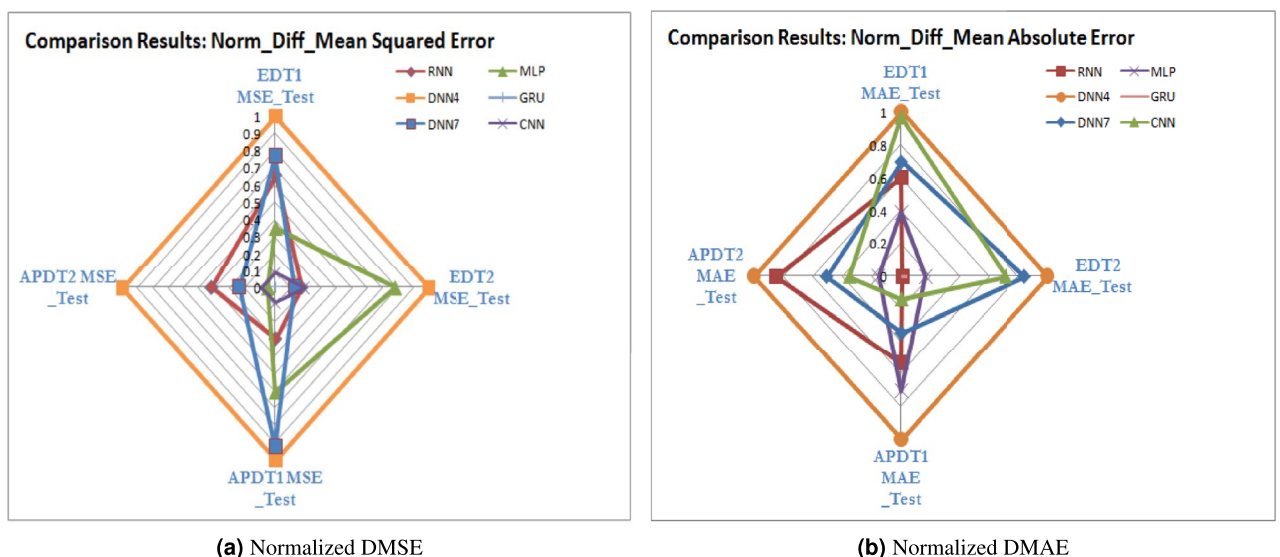
### Prediction outputs/visualization

The output prediction plots of the proposed EvoLearn + GRU model (Best model) on these datasets are illustrated in Fig. 3. The prediction results in the figure are shown in red and blue colour, where blue represents the actual values, and the predicted values are shown in red colour. These prediction outputs represent the one-step-ahead forecasting outputs on the testing datasets comprising 15% of the complete datasets used in the study. From the prediction results depicted in Fig. 3, it is evident that the proposed EvoLearn approach works well at capturing the non-linear and chaotic variations present in the data. Hence, the proposed approach can be efficiently utilized for the time-series analysis and prediction tasks.

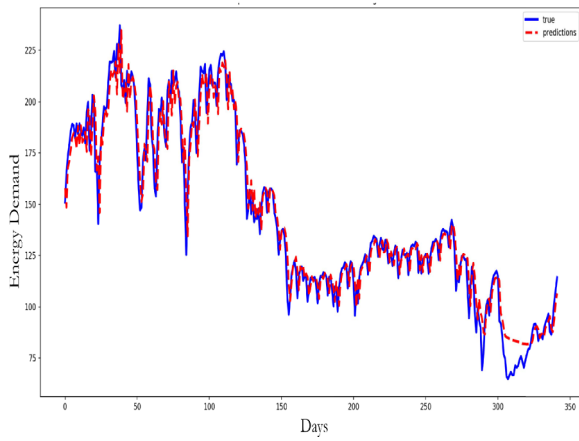
### Discussion

#### Early saturation

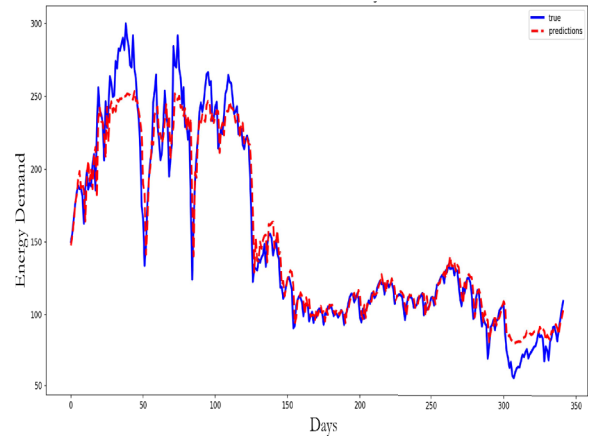
From the experiments, it has been observed that the models trained through EvoLearn attained saturation earlier than the conventional training method. As shown in Fig. 4, the red line denoted the MSE of the models at the difference of five epochs trained using Conventional and EvoLearn approaches, whereas the blue line denoted the Validation MSE of the models trained using these approaches. From the figures, it is clearly noticeable that the model trained using EvoLearn reaches the saturation point earlier than the other models. This is because, in each cycle of GA, the models' weights get optimized by the sequential procedure of crossover, selection, evaluation, and mutation. It helps the new generation (model weights) to have the best node-weights from the previous generation (models).



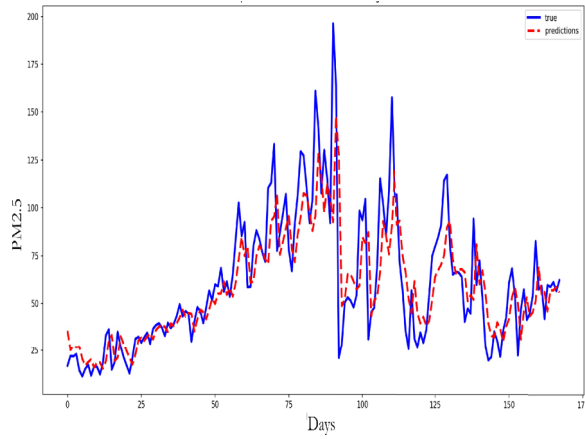
**Figure 2.** Norm\_DMSE and Norm\_DMAE comparison of the models trained using EvoLearn and back-propagation.



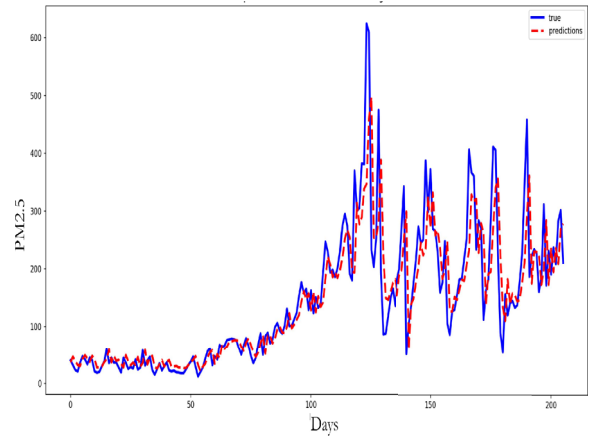
(a) Energy Dataset - 1 ( $E_{DT1}$ )



(b) Energy Dataset - 2 ( $E_{DT2}$ )

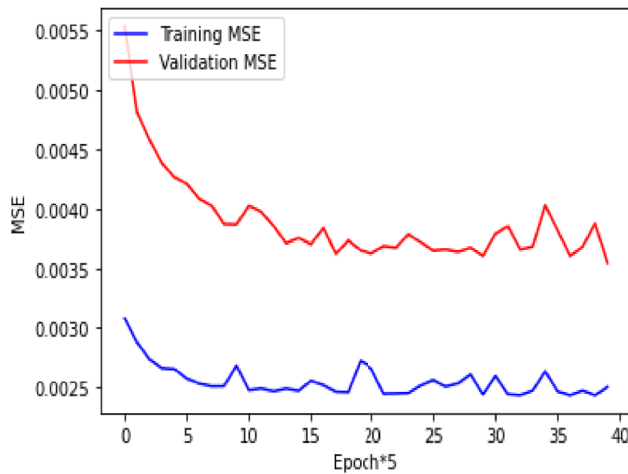


(c) Air Pollution Dataset - 1 ( $AP_{DT1}$ )

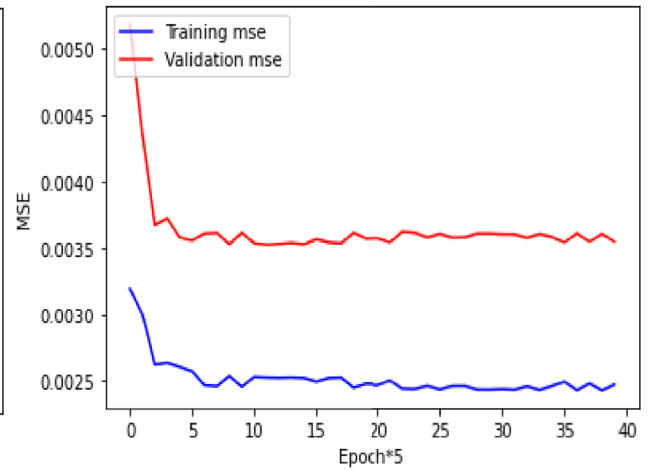


(d) Air Pollution Dataset - 2 ( $AP_{DT2}$ )

**Figure 3.** Prediction results of proposed EvoLearn approach on testing datasets.



(a) Conventional Approach



(b) EvoLearn Approach

**Figure 4.** Early saturation results.



Furthermore, another reason behind better (model) generation is the proposed fitness function (Sect. "Effects of data complexity/non-linear variations": Eq. (15)) that is designed to rank the models based on the performance achieved on both the training and validation set. Consequently, EvoLearn produces such models that perform better not only on the training set but also on the validation set. Therefore, it was also observed that the models trained using EvoLearn are less prone to overfitting. A point to be noted here is that the validation set is not used for "learning", it is only used to assess the performance of the model on unseen data.

### Effect of learnable parameters

Through the experiments, it has been observed that there is a slight inverse correlation ( $-0.34$ ) between the number of learnable parameters in models and DMSE. A possible explanation for this observation is that in previous studies, GA needs an exponentially higher number of generations to optimize a linearly-bigger solution<sup>47</sup>. Since GA works on search space optimization, a higher dimensional input exponentially increases the space to be explored for the optimal solution. Moreover, in our case, the input to GA is the weights matrices, and the bigger the matrices are, the more generations are needed in order to optimize the values. In our experiments, however, the number of generations to be reproduced while training is fixed irrespective of the size of the weight matrices. Therefore, it is acceptable to have a lesser optimization effect on larger models.

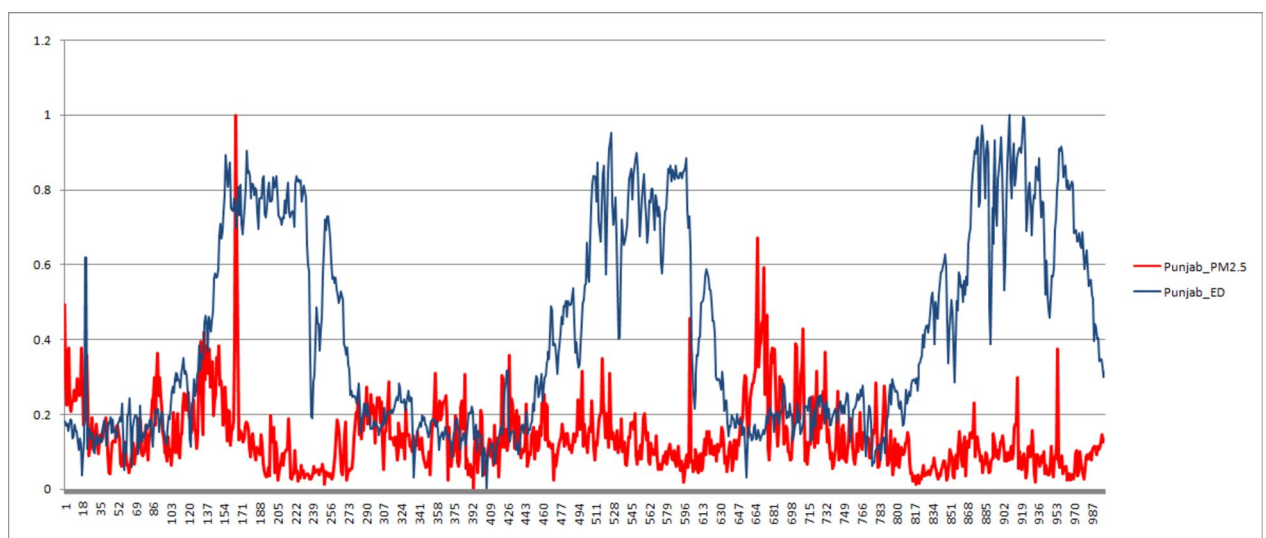
### Effects of data complexity/non-linear variations

From the experiments, it has been noticed that the models trained with EvoLearn performed much better than the conventional method when trained on highly non-linear datasets. In other words, it has been observed that the models' DMSE and DMAE trained using EvoLearn against the models trained with the conventional approach on a highly non-linear dataset (Air pollution) is more (Fig. 5) than on comparatively less non-linear datasets (Energy Demand). A possible reason behind this is that the proposed fitness function is designed to rank the models based on the models' performance on both training and validation datasets. As discussed before, this observation is also the result of the fact that EvoLearn chooses the models for crossover based on their ability to perform better on unseen data.

EvoLearn's computational complexity and resource requirements are substantial, particularly when training large-scale neural networks. The evolutionary algorithms demand significant computational power and memory for population-based optimization and fitness evaluations. The process involves numerous iterations and parallel computations, making high-performance computing resources essential. This complexity can limit practical applicability, especially for resource-constrained environments, but offers powerful optimization capabilities for well-resourced setups.

Potential limitations of EvoLearn include high computational costs, slow convergence rates, and difficulty in balancing exploration and exploitation. These challenges can be addressed by optimizing algorithms, employing efficient parallel computing, and using adaptive parameter tuning to enhance performance and scalability. Additionally, hybrid approaches combining evolutionary strategies with other optimization techniques can improve effectiveness and feasibility.

EvoLearn's findings enable more accurate and efficient predictive models in environmental monitoring and energy management, leading to improved decision-making and resource optimization. Organizations can leverage this method to enhance their predictive analytics capabilities by integrating adaptive and robust model training processes into their workflows.



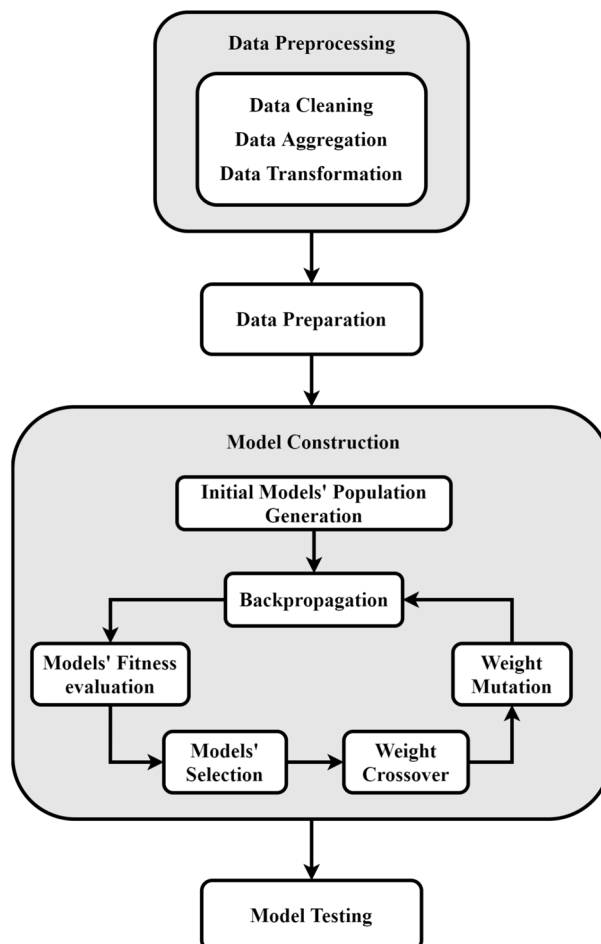
**Figure 5.** Non-linear variations present in air pollution versus energy dataset.

## Conclusion

Time-series forecasting has been one of the hot subjects among scholars for the past several decades. Moreover, many neural-based designs have been proposed to make predictions using the prior sequence of data points. However, the forecasting precision of the models profoundly depends on the training method. Presently, there is a need for a better/faster training methodology in terms of accuracy and training time. In this direction, the present work proposed EvoLearn, a new technique to enhance the training method of neural-based models for forecasting air pollution and energy consumption time series. The presented procedure combines an evolutionary algorithm with the conventional back-propagation algorithm to optimize the model weights throughout the learning process. The basic concept behind EvoLearn is to pick the best parts from multiple models during the training process to produce a superior model. To validate the performance of the proposed method, five different models (MLP, DNN, CNN, RNN, and GRU) were trained on two time-series datasets (Air Pollution and Energy Consumption). Each model was separately trained with two different learning methods (Back-propagation and EvoLearn), and performances were compared. Statistical tests revealed that the proposed EvoLearn approach significantly improved the models' forecasting performances over the models trained with a simple back-propagation learning algorithm. Furthermore, the GRU-based model with EvoLearn showed the highest forecasting accuracy. Additionally, from the sensitivity analysis, it was found that the models trained with EvoLearn avoided over-fitting and also attained the saturation point earlier than the conventional method. In future, the authors intend to incorporate evolutionary algorithms with NLP and image-based models to enhance the models' classification accuracy.

## Methods

This section describes the overall methodology of the proposed EvoLearn approach (shown in Fig. 6). Initially, it involves applying data preprocessing and data preparation to generate the desired target representation from the raw input data. Subsequently, several neural network models are developed using the conventional back-propagation and proposed EvoLearn training/learning strategy separately. Lastly, the performance comparison of both learning strategies (Conventional and EvoLearn) is carried out on datasets belonging to two different application domains. The detailed working of each step is explained as follows:



**Figure 6.** Methodology of the proposed EvoLearn approach.

## Data preprocessing

Data preprocessing entails handling inconsistencies or irregularities present in the data. The goal is to improve data quality, which will aid learning models in attaining strong generalization capabilities. Hence, it is one of the crucial steps in building a forecasting model to solve the target problem. The sequence of preprocessing steps required to achieve the desired data quality is not always the same and may differ from problem to problem. In the present research work, we have applied the following preprocessing steps to achieve the desired data quality.

- *Data cleaning* Data cleaning involves filling in missing entries, outlier identification and noise removal. Out of the four datasets used in the study, only a few missing values were present in the air pollution datasets. As a result, linear interpolation is employed in the present study to fill in those missing values.
- *Data aggregation* The dataset collected from the WebCrawler is not a suitable format for input to a machine learning model. So, data aggregation is applied to combine the data from multiple .csv files to a single file required for building/implementing the target model.
- *Data transformation* In the present study, min-max scaling is applied to transform the data into a particular range (0,1). The mathematical equation that corresponds to the scaling strategy used is as follows:

$$X' = \frac{x_i - \min(X)}{\max(X) - \min(X)}, \quad (5)$$

where  $X$  and  $X'$  represent the input feature and the normalized output feature, respectively.

## Data preparation

Data preparation involves structuring the time series to feed it to the prediction models as input. Defining the degree/amount of historical time-indexed information, a.k.a lag value or look-back, to be used for current timestamp forecasting is essential for building a time-series prediction model. The present research study follows the procedure below to prepare a time series for input to the prediction model.

Consider that we have an input time-series  $\mathcal{T}$  represented as  $\langle t_1, t_2, t_3, \dots, t_n \rangle$  where  $t_i$  denotes the time-series value at timestamp  $i$ , and  $n$  is the total number of samples present in the input time-series. For the purpose of input to the prediction models, we create a new set  $S$  from the existing time series, in which each object consists of a tuple of two time-series as shown below by Eq. (6).

$$S_i = (\langle X_i \rangle, Y_i) \quad i \in N \mid 16 \leq i \leq n, \quad (6)$$

Where,  $X_i$  denotes the set of independent variables to be used as input features for the prediction models and is given by Eq. (7). Here,  $Y$  is the target prediction variable and is given by  $\langle t_i \rangle$ .

$$X_i = \langle t_{i-16}, t_{i-15}, t_{i-14}, \dots, t_{i-1} \rangle, \quad (7)$$

### Splitting dataset

The following step is to divide the dataset  $S_i$  into three parts: training (70%), validation (15%), and testing dataset (15%). The training and validation sections are used to develop/select neural models and obtain an unbiased evaluation of a model fit while estimating optimal hyper-parameters respectively. The testing section is used to assess the accuracy of the trained model on the unknown data.

---

```

1: Keras_ga ← []
2: n_epochs ← 5
3: Specify the size of the initial population (init_pop_size = 20)
4: for i ← 1 to init_pop_size do
5:   Specifying the architecture of neural learning model (layers, neurons per layers, activation function etc.)
6:   Randomly Initialize the weights parametrs of the defined neural model
7: end for
8: for i ← 1 to init_pop_size do
9:   Compilation and fitting of the defined model (Step 4)
10:  Keras_ga.append(weights_vector)
11: end for

```

---

**Algorithm 1.** Models\_Initialization\_BackPropagation().

## Model construction and training

This phase involves building a hybrid approach for the time-series prediction task. The proposed approach works by integrating back-propagation with GA for weight optimization of the shallow and deep neural models. The internal working of each of the undertaken models is as follows:

1. Multi-layer perceptron (MLP) and deep neural network (DNN) ANN consists of a sequence of layers connected by means of interconnections. Furthermore, each layer in an ANN comprises a set of nodes. Every MLP architecture consists of three main layers, namely the input layer, hidden layers, and output layer. The

input layer is responsible for feeding input to the network and does not involve performing any computation on the input data. Following the input layer, there can be one or more hidden layers which are responsible for capturing hidden complexities or features present in the input data. The mathematical equation for a node ( $j$ ) present in the hidden layer is given as:

$$y_j = \text{act} \left( \sum_{i=1}^n w_{ij} \cdot x_i \right), \quad (8)$$

where  $w_{ij}$  represents the connection weight from node  $i$  in the previous layer to current node  $j$  in the hidden layer,  $\text{act}$  represents the activation function,  $n$  denotes the number of nodes in the previous layer and  $x_i$  is the input value at node  $i$  in the input layer. This process is repeated for each neuron present in the hidden layer, and the whole process is executed for each hidden layer present in the MLP. Moreover, depending on the complexity of the given problem, there can be multiple hidden layers present in an MLP architecture that allows naming it as a Deep Neural Network model (as it involves defining a large number of hidden layers). The major aim of defining multiple hidden layers is to capture non-linear complexities and multiple feature aspects of the input data.

- Convolutional neural networks (CNN) Convolutional neural networks<sup>48</sup> are a class of deep neural network models that researchers have widely adopted to solve complex problems relating to several different application domains. Compared to the traditional neural network models, CNN is based on the concept of receptive field to capture both local and global characteristics patterns of the data. The CNN architecture can be defined as an integration of two separate components, namely the convolutional part and the Regression/Classification part. The convolutional part involves defining several components, such as convolutional filters and pooling to extract deep featural aspects of the input data. The second part connects the extracted feature to the MLP, such as architecture for the target regression or classification tasks. The details of the several CNN components are as follows: The input layer may consist of ' $k$ ' neurons where ' $k$ ' denotes the size of the input time-series vector. Subsequently, there can be multiple convolutional filters (with different lengths). The aim of defining these filters is to extract different hidden features present in the input-time series, i.e. to define a non-linear transformation function ' $f$ ' in each filter to capture different timescale features. The convolutional layer may be followed by a pooling layer (min, max or average pooling) to downsample the convolution output. Finally, after multiple layers of convolution and pooling operations, the output time-series will represent the series of featural maps extracted from the input series. These featural maps are then fed to the dense neural connected layers to generate the target regression or classification output.
- Recurrent neural networks (RNN) ANNs are based on the concept that each input sample is independent of the other samples present in the input data. However, the scenario may vary depending on the type of problem under study. In the time-series domain, variations in different timescale aspects might be related to each other and constitute an important factor in estimating the current and future timescale variations. RNNs<sup>49</sup> introduced in 1980, are based on using feedback loops for remembering the previous events' occurrence information and then using the captured hidden information for estimating the future or current timestamps. The architectural details of the model at each timestamp are given as follows:

At any timestamp  $t$ , the activation of the current state is given by:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t), \quad (9)$$

And, the corresponding output at timestamp  $t$  is given by:

$$y_t = W_{hy} \cdot h_t, \quad (10)$$

where  $W_{hh}$ ,  $W_{xh}$ ,  $W_{hy}$  are the shared weight coefficients,  $x_t$ ,  $y_t$  and  $h_t$  represents the input, output and hidden state at timestamp  $t$ .

- Gated recurrent unit (GRU) The RNN model works well at capturing the sequential information present in the input data. However, they suffer from the exploding and vanishing gradient problem, which is why these models are not very efficient and reliable at handling the long-term sequential dependencies present in the series. This vanishing/exploding gradient problem has been resolved with the introduction of Gated Recurrent Unit (GRU) models<sup>50</sup>. The GRU model implements a gating mechanism to control several activities, such as the amount of information to be retained from the previous state, which information to retain, information to be thrown, updating the current state, etc. The network involves implementing two gates: reset gate and update gate.

- Reset gate** This gate entails using the information from both the previous timestamp hidden state and the input at the current timestamp. The aim is to identify the more relevant information from the hidden state and define the new state based on current input and previously filtered relevant information.

$$res_{gate} = \sigma(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1}), \quad (11)$$

$$result = \tanh(res_{gate} \odot (W \cdot h_{t-1}) + W \cdot x_t), \quad (12)$$

- Update gate** This gate involves computing the final output based on the state and input information. Mathematically, it is given as:

$$Upd_{gate} = \sigma(W \cdot x_t + W \cdot h_{t-1}), \quad (13)$$

$$final_{output} = Upd_{gate} \odot h_{t-1}, \quad (14)$$

$$h_t = result \odot (1 - Upd_{gate}) + final_{output}, \quad (15)$$

Following are the architectural details of each of the considered models:

- Deep neural network (DNN): A sequential neural network with seven layers. The input layer has 16 neurons for the input of shape '(window\_size,)', followed by five hidden layers with 15, 12, 10, 9, and 5 neurons, respectively, using ReLU activation, and a single output neuron with sigmoid activation.
- Multilayer perceptron (MLP): A sequential model with two layers. The input layer has 4 neurons with ReLU activation and L2 regularization, accepting input of shape '(window\_size,)', and an output layer with a single neuron using sigmoid activation.
- Gated recurrent unit (GRU): A neural network with three layers. The input layer accepts sequences of shape '(window\_size, 1)', followed by a GRU layer with 4 neurons using ReLU activation and a dense output layer with 1 neuron using linear activation.
- Convolutional neural network (CNN): A sequential model with three convolutional layers and two dense layers. It includes Conv1D layers with 32, 16, and 4 filters, respectively, each followed by MaxPooling1D layers with a pool size of 2, then a Flatten layer. The dense layers include 5 neurons with ReLU activation and a single output neuron with sigmoid activation.
- Recurrent neural network (RNN): A sequential model with two layers. It starts with a SimpleRNN layer with 4 units and no return sequences, accepting input of shape '(x\_train.shape[1], 1)', followed by a dense output layer with 'output\_size' neurons using sigmoid activation.

All the models use the Adam optimizer and mean squared error (MSE) loss function. The process of integrating GA with the above-mentioned models involves the following sequence of steps:

---

**Require:** Global  $X_{train}, Y_{train}, X_{valid}, Y_{valid}, model$

- 1:  $predictions \leftarrow model.predict(X_{train})$
  - 2:  $mse_{train\_error} \leftarrow mse(Y_{train}, predictions)$
  - 3:  $predictions \leftarrow model.predict(X_{valid})$
  - 4:  $mse_{valid\_error} \leftarrow mse(Y_{valid}, predictions)$
  - 5:  $solution\_fitness \leftarrow 1.0 / (mse_{train\_error} + mse_{valid\_error})$
  - 6: **return** solution\_fitness
- 

**Algorithm 2.** *fitness\_function* (model\_weights).

- **Step 1** The very first step in implementing genetic algorithm for the optimization task is to create an initial population. The present research work provides a unique strategy to generate an initial population to be used by the GA-integrated back-propagation algorithm. The sequence of sub-steps involved in this step is as follows:
  - The strategy works by initially defining the architectural representation of the target neural learning models with randomly initialized weight parameters (lines 2 to 7 in Algorithm 1).
  - Following this, the defined models are compiled and trained using the back-propagation<sup>51</sup>. The set of weights generated after running  $n_{epochs}$  of back-propagation represents one candidate of the initial population. This training procedure with random initialization is repeated  $init\_pop\_size$  number of times, where  $init\_pop\_size$  represents the number of candidates or size of the initial population (lines 8 to 11 in Algorithm 1).
- **Step 2** The next step in the sequence is to evaluate the generated chromosomes (models' weights) through the proposed fitness function for selecting the candidate models.
 

The proposed fitness function (Eq. 16) helps in avoiding over-training of the models, as the models' assessment is based on the combined error value obtained on the training as well as the validation dataset. The detailed step-wise working of the fitness function is summarized in Algorithm 2.

$$fitness(M, data_{train}, data_{valid}) = \frac{1}{MSE(M, data_{train}) + MSE(M, data_{valid})}, \quad (16)$$

Here,  $M$  represents the model which is to be assessed,  $data_{train}$  represents the training dataset, and  $data_{valid}$  represents the validation dataset.

---

```

1: for  $j$  in range (0, num_ iterations) do
2:   Assesing the fitness of each candiate in the input population
3:   Choosing the best candidates for input to the reproduction phase.
4:   Creating the new population (ga_instances) by crossover and mutation
5:   Demonstrating the performance of best_solution or best_weights on the training and validation parts
6:   if  $j! = (\text{num\_iterations} - 1)$  then
7:     for  $i$  in range (len (ga_instances)) do
8:       Initializing the neural architectural model (defined in Algorithm 3) weights with ga_instances
9:       Train the defined model by using back-propagation (runing n_epochs)
10:      Use these new_weights as new population (ga_instances) for input to the GA Model
11:    end for
12:  end if
13: end for
14: Evaluating the performance of the best solution or optimized model weights on the test part

```

---

**Algorithm 3.** *Keras\_GA\_Model* (models\_weights).

- *Step 3* This step entails developing/training the proposed GA and back-propagation integrated learning models for the prediction tasks. The overall working of this stage is described in Algorithm 3. It consists of carrying out a two-phase procedure, each of which includes a series of sub-steps.
  1. *GA-phase* This phase involves employing GA in the learning task. The sequence of steps involved in this phase is as follows: Firstly, the initial population generated by employing the back-propagation (explained in step 1) is passed as an input to the GA. Secondly, the fitness value of all candidate solutions in the input population is calculated. Thirdly, the rank selection (based on the fitness function value) chooses  $init\_pop\_size/4$  best weights candidates for input to the reproduction phase. Finally, the reproduction phase is utilized to generate a new population of weights using the scattered crossover with 0.5 probability and random mutation with a mutant of 10%.
  2. *BP-phase* The second phase comprises incorporating back-propagation to the above stated GA-based learning task. In this back-propagation phase, the last generation (weights candidates) from the previous phase is used to initialize the weights of the neural models. A total of  $init\_pop\_size$  neural models are trained by running  $n\_epochs$  iterations of the back-propagation algorithm with candidates set-based weight initialization. The final weights matrix obtained (size =  $init\_pop\_size * \text{the number of learnable weights parameters}$ ) after back-propagation becomes new input to the GA, which is then passed to GA phase of this step. This training process is repeated for a defined number of cycles ( $num\_iterations$  times), alternating between GA and back-propagation-based learning during each epoch.
- *Step 4* In this step, the performance of the proposed GA and back-propagation integrated neural models is evaluated on the test dataset. After the last cycle, the best model among the generated population is selected for validation.

### Model evaluation and performance metrics

Model evaluation is crucial to assess the accuracy and reliability of any prediction model. In the present study, the following well-known metrics are employed to evaluate the performance of the proposed GA and back-propagation integrated neural models.

- *MSE (mean squared error)*<sup>52</sup>: It estimates the quality of the prediction model by calculating the average of square of errors between the actual and the predicted values. Mathematically, it is given as:

$$MSE = \frac{1}{m} \sum_{s=1}^m (y_s - \bar{y}_s)^2, \quad (17)$$

where  $m$  denotes the number of samples present in the dataset,  $y_s$  &  $\bar{y}_s$  represent the actual and predicted values respectively.

- *MAE (mean absolute error)*<sup>52</sup>: It measures the average magnitude of a model's prediction error by calculating the absolute difference between the actual and predicted outcomes.

$$MAE = \frac{1}{m} \sum_{s=1}^m |y_s - \bar{y}_s|, \quad (18)$$

## Data availability

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Received: 8 December 2023; Accepted: 2 August 2024

Published online: 29 August 2024

## References

- Shumway, R. H., Stoffer, D. S. & Stoffer, D. S. *Time Series Analysis and its Applications* Vol. 3 (Springer, 2000).
- Hussain, W., Hussain, F. K., Saberi, M., Hussain, O. K. & Chang, E. Comparing time series with machine learning-based prediction approaches for violation management in cloud slas. *Future Gen. Comput. Syst.* **89**, 464–477 (2018).
- Weng, Y. *et al.* Forecasting horticultural products price using Arima model and neural network based on a large-scale data set collected by web crawler. *IEEE Trans. Comput. Soc. Syst.* **6**, 547–553 (2019).
- Hsu, C.-Y. & Liu, W.-C. Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing. *J. Intell. Manuf.* **32**, 823–836 (2021).
- Bedi, J. & Toshniwal, D. Deep learning framework to forecast electricity demand. *Appl. Energy* **238**, 1312–1326 (2019).
- Bai, L., Wang, J., Ma, X. & Lu, H. Air pollution forecasts: An overview. *Int. J. Environ. Res. Public Health* **15**, 780 (2018).
- Nti, I. K., Teimeh, M., Nyarko-Boateng, O. & Adekoya, A. F. Electricity load forecasting: A systematic review. *J. Electr. Syst. Inf. Technol.* **7**, 1–19 (2020).
- Chang, Y.-S. *et al.* An lstm-based aggregated model for air pollution forecasting. *Atmos. Pollut. Res.* **11**, 1451–1463 (2020).
- Samal, K. K. R., Babu, K. S., Das, S. K. & Acharaya, A. Time series based air pollution forecasting using sarima and prophet model. In *Proc. of the 2019 International Conference on Information Technology and Computer Communications*, 80–85 (2019).
- Somu, N. *et al.* A hybrid model for building energy consumption forecasting using long short term memory networks. *Appl. Energy* **261**, 114131 (2020).
- Wei, N., Li, C., Peng, X., Zeng, F. & Lu, X. Conventional models and artificial intelligence-based models for energy consumption forecasting: A review. *J. Petrol. Sci. Eng.* **181**, 106187 (2019).
- Cortez, P., Rocha, M. & Neves, J. Evolving time series forecasting arma models. *J. Heuristics* **10**, 415–429 (2004).
- Hernandez-Matamoros, A., Fujita, H., Hayashi, T. & Perez-Meana, H. Forecasting of covid19 per regions using arima models and polynomial functions. *Appl. Soft Comput.* **96**, 106610 (2020).
- Dastorani, M., Mirzavand, M., Dastorani, M. T. & Sadatinejad, S. J. Comparative study among different time series models applied to monthly rainfall forecasting in semi-arid climate condition. *Nat. Hazards* **81**, 1811–1827 (2016).
- Malki, Z. *et al.* Arima models for predicting the end of covid-19 pandemic and the risk of second rebound. *Neural Comput. Appl.* **33**, 2929–2948 (2021).
- He, W., Wang, Z. & Jiang, H. Model optimizing and feature selecting for support vector regression in time series forecasting. *Neurocomputing* **72**, 600–611 (2008).
- Sapankevych, N. I. & Sankar, R. Time series prediction using support vector machines: A survey. *IEEE Comput. Intell. Mag.* **4**, 24–38 (2009).
- Yan, W. Toward automatic time-series forecasting using neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **23**, 1028–1039 (2012).
- Borghi, P. H., Zakordonets, O. & Teixeira, J. P. A covid-19 time series forecasting model based on mlp ann. *Procedia Comput. Sci.* **181**, 940–947 (2021).
- Chandran, L. R. *et al.* Residential load time series forecasting using ann and classical methods. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, 1508–1515 (IEEE, 2021).
- Du, S., Li, T., Yang, Y. & Horng, S.-J. Deep air quality forecasting using hybrid deep learning framework. *IEEE Trans. Knowl. Data Eng.* <https://doi.org/10.48550/arXiv.1812.04783> (2019).
- Bedi, J. & Toshniwal, D. Energy load time-series forecast using decomposition and autoencoder integrated memory network. *Appl. Soft Comput.* **93**, 106390 (2020).
- Kwak, N. W. & Lim, D. H. Financial time series forecasting using adaboost-gru ensemble model. *Korean Data Inf. Sci. Soc.* **32**, 267–281 (2021).
- Hewamalage, H., Bergmeir, C. & Bandara, K. Recurrent neural networks for time series forecasting: Current status and future directions. *Int. J. Forecast.* **37**, 388–427 (2021).
- Al-Gabalawy, M., Hosny, N. S. & Adly, A. R. Probabilistic forecasting for energy time series considering uncertainties based on deep learning algorithms. *Electr. Power Syst. Res.* **196**, 107216 (2021).
- Ruan, L., Bai, Y., Li, S., He, S. & Xiao, L. Workload time series prediction in storage systems: A deep learning based approach. *Clust. Comput.* **26**, 1–11 (2021).
- Ho, S. L. & Xie, M. The use of arima models for reliability forecasting and analysis. *Compu. Ind. Eng.* **35**, 213–216 (1998).
- Connor, J. T., Martin, R. D. & Atlas, L. E. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Netw.* **5**, 240–254 (1994).
- Soltani, S. On the use of the wavelet decomposition for time series prediction. *Neurocomputing* **48**, 267–277 (2002).
- Shi, J., Guo, J. & Zheng, S. Evaluation of hybrid forecasting approaches for wind speed and power generation time series. *Renew. Sustain. Energy Rev.* **16**, 3471–3480 (2012).
- Xu, S., Chan, H. K. & Zhang, T. Forecasting the demand of the aviation industry using hybrid time series sarima-svr approach. *Transport. Res. E Logist. Transport. Rev.* **122**, 169–180 (2019).
- Sheikhan, M. & Mohammadi, N. Neural-based electricity load forecasting using hybrid of ga and aco for feature selection. *Neural Comput. Appl.* **21**, 1961–1970 (2012).
- Dong, Y., Ma, X. & Fu, T. Electrical load forecasting: A deep learning approach based on k-nearest neighbors. *Appl. Soft Comput.* **99**, 106900 (2021).
- Anh, N. N., Anh, N. H. Q., Tung, N. X. & Anh, N. T. N. Feature selection using genetic algorithm and Bayesian hyper-parameter optimization for lstm in short-term load forecasting. In *The International Conference on Intelligent Systems & Networks*, 69–79 (Springer, 2021).
- Niska, H., Hiltunen, T., Karppinen, A., Ruuskanen, J. & Kolehmainen, M. Evolving the neural network model for forecasting air pollution time series. *Eng. Appl. Artif. Intell.* **17**, 159–167 (2004).
- Kumar, P., Batra, S. & Raman, B. Deep neural network hyper-parameter tuning through twofold genetic approach. *Soft Comput.* **25**, 8747–8771 (2021).
- Li, C. *et al.* Genetic algorithm based hyper-parameters optimization for transfer convolutional neural network. Preprint at <https://arxiv.org/2103.03875> (2021).
- Yuan, Y., Wang, W., Coghill, G. M. & Pang, W. A novel genetic algorithm with hierarchical evaluation strategy for hyperparameter optimisation of graph neural networks. Preprint at <http://arxiv.org/abs/2101.09300> (2021).
- Shahid, F., Zameer, A. & Muneeb, M. A novel genetic lstm model for wind power forecast. *Energy* **223**, 120069 (2021).

40. Kara, A. Multi-step influenza outbreak forecasting using deep lstm network and genetic algorithm. *Expert Syst. Appl.* **180**, 115153 (2021).
41. Huang, Y., Gao, Y., Gan, Y. & Ye, M. A new financial data forecasting model using genetic algorithm and long short-term memory network. *Neurocomputing* **425**, 207–218 (2021).
42. Cicek, Z. I. E. & Ozturk, Z. K. Optimizing the artificial neural network parameters using a biased random key genetic algorithm for time series forecasting. *Appl. Soft Comput.* **102**, 107091 (2021).
43. Ni, P., Li, G., Hung, P. C. & Chang, V. Staresgru-cnn with cmedlms: A stacked residual gru-cnn with pre-trained biomedical language models for predictive intelligence. *Appl. Soft Comput.* **113**, 107975 (2021).
44. Yang, Y. & Duan, Z. An effective co-evolutionary algorithm based on artificial bee colony and differential evolution for time series predicting optimization. *Complex Intell. Syst.* **6**, 299–308 (2020).
45. Wang, L., Zeng, Y. & Chen, T. Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. *Expert Syst. Appl.* **42**, 855–863 (2015).
46. Jalali, S. M. J. *et al.* A novel evolutionary-based deep convolutional neural network model for intelligent load forecasting. *IEEE Trans. Ind. Inf.* **17**, 8243–8253 (2021).
47. Katoch, S., Chauhan, S. S. & Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **80**, 8091–8126 (2021).
48. Albawi, S., Mohammed, T. A. & Al-Zawi, S. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, 1–6 (IEEE, 2017).
49. Sherstinsky, A. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D* **404**, 132306 (2020).
50. Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. Preprint at <http://arxiv.org/abs/1412.3555> (2014).
51. Chauvin, Y. & Rumelhart, D. E. *Backpropagation: Theory, Architectures, and Applications* (Psychology Press, 2013).
52. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

### Author contributions

J.B., A.A. and S.G. designed the proposed models, conceived and conducted the experiments. R.S.B. and M.A.F. collected the data and analysed the results. S.M. and R.P. designed the study and interpreted the results. All authors reviewed the manuscript.

### Competing interests

The authors declare no competing interests.

### Additional information

**Correspondence** and requests for materials should be addressed to S.G. or M.A.F.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2024