Heliyon 10 (2024) e27860

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Research article

Developing a multivariate time series forecasting framework based on stacked autoencoders and multi-phase feature

Dilip Kumar Sharma [a], Ravi Prakash Varshney [b], Saurabh Agarwal [c], Amel Ali Alhussan [d], Hanaa A. Abdallah [e,*]

[a] Department of Computer Engineering and Application, GLA University, Mathura 281406, India
[b] S&P Global, Noida, India
[c] College of Digital Convergence, Yeungnam University, Gyeongsan 38541, South Korea
[d] Department of Computer Sciences, College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia
[e] Department of Information Technology, College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, Riyadh 84428, Saudi Arabia

ARTICLE INFO

Keywords:
Feature selection
Autoencoder
Deep learning
Long short-term memory (LSTM)
Convolutional neural network (CNN)
Multivariate time series forecasting (MTS)
Temporal convolution network (TCN)

ABSTRACT

Time series forecasting across different domains has received massive attention as it eases intelligent decision-making activities. Recurrent neural networks and various deep learning algorithms have been applied to modeling and forecasting multivariate time series data. Due to intricate non-linear patterns and significant variations in the randomness of characteristics across various categories of real-world time series data, achieving effectiveness and robustness simultaneously poses a considerable challenge for specific deep-learning models. We have proposed a novel prediction framework with a multi-phase feature selection technique, a long short-term memory-based autoencoder, and a temporal convolution-based autoencoder to fill this gap. The multi-phase feature selection is applied to retrieve the optimal feature selection and optimal lag window length for different features. Moreover, the customized stacked autoencoder strategy is employed in the model. The first autoencoder is used to resolve the random weight initialization problem. Additionally, the second autoencoder models the temporal relation between non-linear correlated features with convolution networks and recurrent neural networks.

Finally, the model's ability to generalize, predict accurately, and perform effectively is validated through experimentation with three distinct real-world time series datasets. In this study, we conducted experiments on three real-world datasets: Energy Appliances, Beijing PM2.5 Concentration, and Solar Radiation. The Energy Appliances dataset consists of 29 attributes with a training size of 15,464 instances and a testing size of 4239 instances. For the Beijing PM2.5 Concentration dataset, there are 18 attributes, with 34,952 instances in the training set and 8760 instances in the testing set. The Solar Radiation dataset comprises 11 attributes, with 22,857 instances in the training set and 9797 instances in the testing set. The experimental setup involved evaluating the performance of forecasting models using two distinct error measures: root mean square error and mean absolute error. To ensure robust evaluation, the errors were calculated at the identical scale of the data. The results of the experiments demonstrate the superiority of the proposed model compared to existing models, as evidenced by significant advantages in various

* Corresponding author.
E-mail addresses: dilip.sharma@gla.ac.in (D. Kumar Sharma), ravi_varshney@outlook.com (R. Prakash Varshney), saurabhnsit2510@gmail.com (S. Agarwal), AAAlHussan@pnu.edu.sa (A. Ali Alhussan), haabdullah@pnu.edu.sa (H.A. Abdallah).

https://doi.org/10.1016/j.heliyon.2024.e27860
Received 12 December 2023; Received in revised form 29 February 2024; Accepted 7 March 2024
Available online 19 March 2024

metrics such as mean squared error and mean absolute error. For PM2.5 air quality data, the proposed model's mean absolute error is 7.51 over 12.45, about ~40% improvement. Similarly, the mean square error for the dataset is improved from 23.75 to 11.62, which is ~51%of improvement. For the solar radiation dataset, the proposed model resulted in ~34.7% improvement in means squared error and ~75% in mean absolute error. The recommended framework demonstrates outstanding capabilities in generalization and outperforms datasets spanning multiple indigenous domains.

1. Introduction

In the modern digital age, vast piles of information are being produced at a never-seen scale. Such information is regularly collected over a while by various sensors and other collection mechanisms from different sources. The data in context is usually of high dimension and contains numerous correlated features. With various machine learning and deep learning strategies, this data is being utilized to control the underlying process and implement various data-driven models with various aims such as forecasting, prediction, and commending proper actions. High-dimensional datasets are produced in various social, industrial, healthcare, and environmental applications. Datasets like these are analyzed and used in various applications, such as health supervising, vehicle control devices, impulsiveness assessment in the financial domain, data center monitoring, and avionics. Forecasting in multiple domains has become a promising development orientation and aids a lot in better risk management and decision-making analysis. Forecasting has become imperative for solving various present-day problems. With the advent of industrialization, air pollution has become one of the biggest global challenges. Air quality or PM2.5 concentration forecasting models are essential in curbing air pollution, resulting in good-quality air. Forecasting various key performance indicators assists multiple industries in keeping their production well within the expected demand, thus reducing the cost of the extra output, storage, and maintenance and optimizing the overall process. The forecasting models consider various factors, including various meteorological and surrounding factors, and promptly predict the air quality information, which assists government agencies in deciding to keep pollution under control.

Over the last decade, forecasting features in various domains have received a lot of interest. It has been a promising development orientation for many industries that rely on forecasting to improve their operational efficiency, risk management, and efficient decision-making. In most such use cases [1,2] across different domains, the future phenomena had some relationship or linkage with the past observations. A time series [3] is represented as a sequence of values observed in the same time sequence, where xi denotes the value of ith observation and n represents the number of observations or length of time series. Time series forecasting is scrutinizing time series data or past observations methodically in chronological order to predict the future value [4,5]. Time Series forecasting can be again classified into univariate or multivariate depending on the number of time series involved. In multivariate time series forecasting, we have time series for both the target and dependent variables, commonly known as predictor variables. In other words, a dataset with multiple time series data for all the different features is called multivariate time series data.

Accurate time series forecasting becomes challenging in the financial domain as it involves factors such as randomness and uncertainty in the dataset. The complexity of the forecasting is further enhanced by features like non-stationary, nonlinearity, and the sequence correlation of time series data. The domain is complex and exhibits a non-linear behavior comprising a surplus of factors manipulating the estimation of the relationship between target and dependent variables. Moreover, the model should be effectively capable of identifying and establishing the non-linear dynamic relationship of the influencing variables on the target variables.

The dynamic nature of the features in the real-time series data poses a serious and complex challenge to analyzing it and using it to forecast non-linear time series data with the help of computational methodologies such as neural networks. Artificial Neural Network (ANN) has been prominently used in modeling time series prediction owing to its universal approximation capabilities [6]. Shallow ANN with many layers and parameters is difficult to train. Comparatively, deep neural networks (DNNs) fare well in forecasting time series data compared to the conventional ANN [7]. DNN excels in learning the hierarchical feature mapping from the dataset [8] and does not require feeding any manually created features to learn. Recurrent neural networks (RNNs) have also been used comprehensively in numerous time series forecasting problems. Due to the recurrent feedback connections, RNNs have a short-term memory that makes them suitable for time series forecasting problems. Due to these reasons, RNN can model the time series data better than DNN or ANN. While modeling the time series data, RNN maintains a vector of activation parameters for each time step. This works well in modeling short-term dependencies. However, while modeling data with long-term dependencies with a stochastic gradient descent algorithm, RNN fail miserably to decode the dependencies in the input data because of the vanishing gradient problem [9]. The LSTM model fixed this problem by incorporating a dedicated neuron to maintain the persistent backward stream in the error signal. This lets the LSTM network decode and take in the long-term dependencies [10]. The LSTM, too, has some shortcomings. Its performance degrades with the time series, and they don't fare well in representing the complex features of sequential time series data with a high degree of nonlinearity. Research has been done where a deep LSTM (DLSTM) [7] model was proposed. The proposed DLSTM proved to be better and fixed the problems of conventional shallow LSTM.

Neuron weights are randomly initialized in the standard ANN strategies concerning stochastic gradient descent. While doing so in the cases where several non-linear parameters are part of the input dataset, the backpropagation algorithm may be entombed within the local minima. Various new approaches [11] were devised to initialize the weights in a more organized and efficient manner. Various earlier proposed models, including DLSTM, fare well on univariate time series (UTS) problems where only a single variable is modeled. At the same time, the MTS problem involves multiple highly correlated variables. If the random weight initialization is done

as a part of the MTS problem, it may lead to a bigger disaster as it can result in the model being trapped within various local minima. Numerous prior studies have substantiated that training deep learning networks with multiple layers and random weight initialization yields inferior outcomes compared to models employing shallow architectures [12,13]. Addressing this issue, using stacked convolutional layers instead of fully connected layers has proven to be a practical approach to mitigating the impact of vanishing gradients within deep neural networks [13].

In this study, we offer a hybrid multi-stage method to handle the multivariate time series data effectively and forecast the target variable. The first stage in the strategy is to identify the most relevant and fitting features from the entire feature set. A hybrid feature selection method is proposed to accomplish the same. This stage is followed by an LSTM-based autoencoder with attention and a temporal convolution network to analyze, learn, and forecast the time series data. The experimentation was conducted on four real-world datasets. An evaluation was carried out comparing the proposed method with existing reference models and other customized strategies applied to the same datasets. The results indicate a distinct superiority of the proposed model across three performance metrics. As a result of this paper, the following significant contributions have been made:

- Proposed a new hybrid feature selection method that extracts the most relevant features using Greywolf optimization and finds redundant lag values for given time series based on mutual information.
- The random weight initialization problem of DLSTM is solved using LSTM-based autoencoder learning.
- Using unlabeled time series data, a robust forecasting module is proposed to convert the data into demonstrative attributes that can be exploited and analyzed easily.
- Present a new TCN-based autoencoder to model temporal relations using deep learning algorithms, i.e., CNNs and RNNs.

The remaining paper is organized as follows: Section 2 briefly overviews the relevant work in this area. A discussion on the multivariate time series problem and methodology is presented in section 3. A description of the experimental settings and datasets used in this study is provided in Section 4. All four case studies are also analyzed and compared in this section. As a final section, section 5 summarizes the conclusions and suggests future work.

2. Related work

Analyzing multivariate time series data is complicated compared to univariate time series data. This complication is owed to the correlation in features. Additionally, it is a complex process to model the non-linear multiple features and identify the latent features. Various research and studies have been done in univariate time series forecasting. Still, owing to the complicated nature of multivariate time series forecasting, it hasn't gotten the same level of interest and exposure. We have reviewed various state-of-the-art models and strategies that have been practiced in recent times to solve the multivariate time series forecasting problem.

Research [12] has proved that artificial neural networks suffer from problems such as overfitting and local minima while approximating a complicated function as a part of regression problems. Consequently, the output is not up to the mark. Various new training strategies for conventional ANNs have been proposed [6,14]. One proposed strategy relied on extracting and formulating features capable of extracting information from the given dataset [15]. However, formulating features is an expensive operation and requires information about the domain. Another strategy proposed by Schmidhuber [16] was based on pre-learning the feature from the unlabeled data. It proposed to study the feature representation layers from the unlabeled data. This is superior to the former, as learning from unlabeled is better than learning from the formulated features. Moreover, the availability and profusion of unlabeled data are better when compared to labeled data.

Additionally, such a strategy can be coupled with stacking. A deep architecture model comprised of stacked numerous feature representation layers is more efficient in modeling complex structures and correlated features, which are the backbone of the multivariate time series dataset [12,15]. The strategy also overcomes the long due of the problem of overfitting in the modeling of systems [13].

We must experiment with fine analysis of complex real-world data to develop robust features that capture appropriate information from labeled data. For each task, such domain-specific features require expertise in data handling, are expensive, and are time-consuming to develop [15]. Another approach, which Schmidhuber first presented, is to learn feature representation layers from unlabeled data using unsupervised feature learning [16,17]. It is much more effective and efficient to use the latter approach since unlabeled data can be used for learning, and unlabeled data is more readily available and abundant. Moreover, learning the features from unlabeled data in advance is better than hand-crafting them afterward. This is called pretraining [18,19]. A significant advantage of this approach is the capability to stack different feature representation layers into deep architectures [16] that are superior to traditional ones for modeling correlated features and complex structures found in multivariate time series problems [12,15]. These unsupervised pretraining approaches [13] have fixed the underfitting and overfitting problems. Thus, they can model complex neural systems for a long time.

Classical autoencoders consist of feedforward neural networks where input cells are the same size as output cells, and hidden cells are much smaller than input and output cells [20]. By removing noise from its input, an autoencoder produces an output that reconstructs its input. Considering the hidden layers consist of fewer neurons than the input layers, the hidden layer weights are only based on the most representative features of the input data and do not include details such as outliers. A feedforward neural network-based autoencoder is often used for data that does not follow a sequential order.

It is essential to capture long-term dependencies between output variables and learn correlations between them simultaneously to generate generative models of sequence data. RNNs have successfully addressed a wide range of problem domains that require

sequential data. The use of TCNs for audio synthesis, language modeling, and machine translation [21–23] has shown that they can also perform well in less training time. This owes to their capability to learn without depending on instance recurrence. A temporal hierarchy is also introduced in the TCN architecture: the upper layers gain access to longer input subsequences and learn representations over a more extended period. Residual and skip connections propagate local information through the hierarchy [22,24].

We can identify which variables are relevant to a dataset [25] by selecting features. Furthermore, it facilitates faster training, increases computational efficiency, and improves understanding of the underlying process that generated the data by removing noise and irrelevant inputs. Feature selection involves evaluating, constructing, and transforming features to present them most appropriately. Feature evaluation is used for time series data to identify levels, trends, and seasonality in input variables and dynamic lags. As well as being adaptive to stochastic component changes, it is robust against outliers and noise. Using exogenous dummy variables in models of time series components defining principal components or factor analyses based on input variables, a new feature can be created based on the input variables.

Qi et al. [26] introduced a hybrid model for predicting PM2.5 mass concentrations over time, considering spatial and temporal variations. This model integrates a graph convolutional network and long-short-term memory, offering an optimal approach to forecasting. The graph convolutional network assisted in identifying spatial dependency among various stations, whereas the long-short-term memory helped identify temporal dependency among observations several times. On similar grounds, Huang et al. [27] proposed a hybrid model comprising the CNN model and LSTM to resolve the PM2.5 concentration time series forecasting problem. They used CNN in the model to select the features followed by LSTM to study previously extracted features and then forecast the following PM2.5 observation. A study proved that the hybrid model outclassed the single-individual models when applied individually.

3. Problem and methodology

3.1. Problem statement

A time series dataset comprises a series of observations noted at a standard interval over a fixed period. Time series prediction is forecasting the future values of a feature of time series data based on either the same feature (univariate) or several correlated features (multivariate). Thus, multivariate time series is a complex problem compared to univariate time series, as it operates on two or more correlated features.

We can define a univariate time series problem as a series of n values where n represents the number of recorded observations per Eq. (1). For forecasting the value, the method uses a set of past values. Therefore, in this method, the predicting model perceives the composition, absorbs the trend of the observations, and deduces the involved process into the future.

$$x(i) \in R : i = 1, 2, \ldots n \tag{1}$$

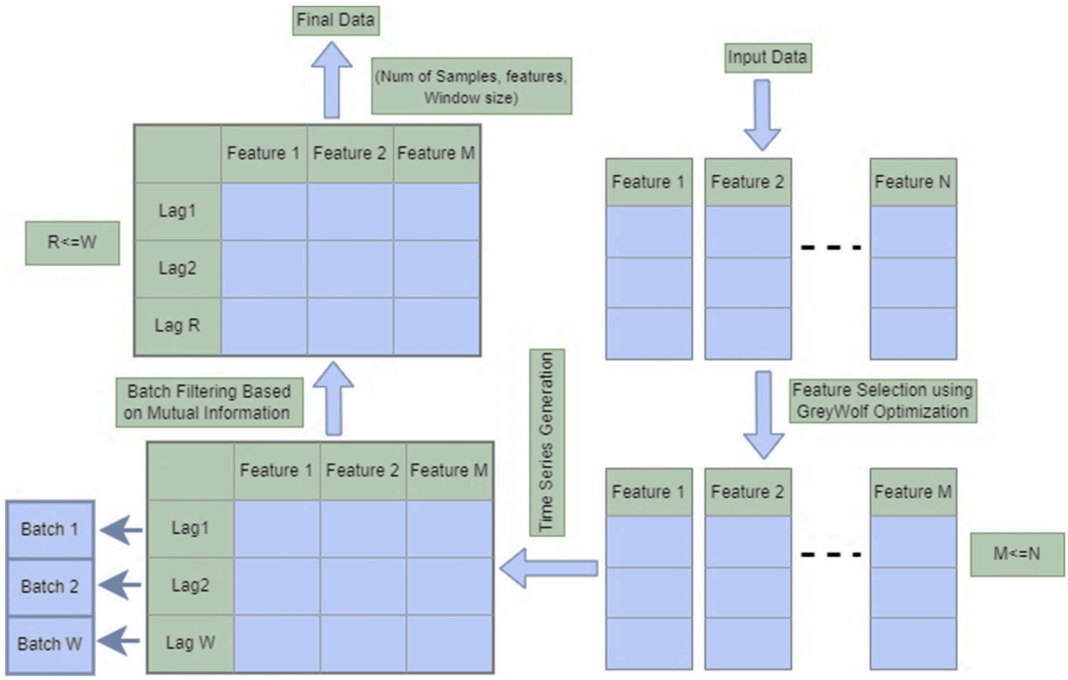

Fig. 1. Multi-phase feature selection strategy.

A multivariate time series forecasting problem can be defined as a collection of all correlated features. It is a collection of univariate series, one per feature. In a multivariate forecasting strategy, the value for the target variable is forecasted not only based on the previous values of the same variable but also on the previous values of the other correlated variables. Eq. (2) and (3) represent the definition of a dataset X, where each data point consists of a set of features (x_i) and their corresponding time series (x_j(i)).

$$X = (x\_1, x\_2, x\_3, \ldots\ldots\ x\_k) \text{ where } x\_i \text{ is the } i^{th} \text{ feature} \tag{2}$$

$$X = \{x\_j\ (i) \in R : i = 1, 2, ..n\ \}(j = 1, 2, 3..k) \text{ where } x\_j \text{ is the time series for } j^{th} \text{ feature} \tag{3}$$

In this study, we analyze a multivariate time series dataset and propose a model that aims to perform multivariate time series forecasting with high accuracy and generalization.

3.2. Feature selection

Feature selection technique is applied to the entire feature set, which returns the relevant features for further analysis. The features are selected by finding the feature subset having maximum mutual information concerning the target variable using the Greywolf optimization technique. The features with the best dimensions with the target feature are considered for the next step, and the rest of the other features are dropped from the feature set. When two variables are measured by mutual information, the uncertainty for one variable is reduced for the other variable when the value of the other variable is known.

Firstly, the Grey Wolf Optimization and the Mutual Information technique are applied to the feature set to identify the relevant features. Then, the time series data is generated using the selected features with an optimal value of window length, which is dependent on the data. During the generation of the data, some lag features do not impact the target forecast variable for different lag values of window length features. In such cases, those features have been dropped from the feature subset. In this context, lagging a time series involves adjusting the values by shifting them forward one or more-time steps, as needed. Alternatively, it can be expressed as shifting the times in its index backward by one or more steps.

Fig. 1 represents that to determine all the relevant lag values that reside between the ranges of window length, firstly, the data is generated for each possible lag value, followed by taking the mean mutual information of all the features present in the data. Lastly, all the lag value features that highly impact the target variable are filtered out. The feature selection method is hybrid, as we first make feature selection by optimizing the mutual information between the candidate and target variables. Secondly, during the time series generation, different lag values of window length are analyzed, and features that do not impact the target variable are dropped.

Greywolf optimization feature selection technique: In nature, grey wolves follow a leadership hierarchy and have a hunting mechanism like grey wolves. They optimize by mimicking their system of leadership and hunting. An optimization model for grey wolves focuses on discovering optimal sections of the complex search space over the interaction of individuals. Grey wolves have a four-layer hierarchical structure of α, β, δ, and ω.

Grey wolves are dominated by the wolf α, which has the best adaptability. In terms of fitness, β and δ are the second best. Assisting wolves with pack management and making hunting decisions is their responsibility. The main task of β and δ is to help wolf α handle wolf packs and make decisions in the hunting process. It can be described as follows: first, wolves α lead grey wolves to hunt, track, and move toward their prey; then β, δ they command ordinary wolves to attack the prey under their command until they capture it. By mimicking grey wolf behaviors such as hunting, surrounding, and attacking, the GWO algorithm simulates predation. As a result, global optimization can be achieved.

The input data with N features is taken, then after applying feature selection using grey wolf optimization, the number of features is reduced to M ($M \leq N$). After that, time series data is generated in which rows represent the lag values and columns are features. Each lag value, i.e., row (total W rows), is represented by batch. Then, we do batch filtering based on mutual information that provides the R lag values ($W \leq R$). This is the data that will be provided as input to the autoencoder.

The hybrid feature selection method, incorporating Grey Wolf Optimization and mutual information for lag values, is a crucial aspect of our research. The rationale behind choosing these methods stems from their complementary strengths. Grey Wolf Optimization is employed for its ability to search the feature space efficiently, optimizing the selection process by iteratively improving the subset of features with the highest mutual information about the target variable. This method aligns with our objective of enhancing the model's predictive performance by focusing on the most informative features. Mutual information for lag values is particularly suitable for time series data, capturing the dependencies between variables across different time points. Combining these techniques provides a comprehensive and nuanced approach to feature selection in our context.

3.3. Autoencoders

Feedforward neural networks with autoencoders are feedforward neural networks where the input and output are the same. This is accomplished by compressing the input into a lower-dimensional representation and then reconstructing the output. Codes, also known as latent-space representations, are compact summaries of inputs. Three components make up an autoencoder: encoder, code, and decoder. The encoder compresses the input and produces the code as output. The decoder is then used to reconstruct or reproduce the input provided using only the code, which is the output of the encoder. The way artificial neural networks via backpropagation are trained. Similarly, autoencoders are trained.

Even though they are technically unsupervised learning methods, they are trained with the help of supervised learning methods,

known as self-supervised methods. Most autoencoders are implemented as part of a generic model that tries to reconstruct the input. Different autoencoders serve different purposes, but perhaps the most common is to extract features automatically from input data.

In the proposed model, we have used Autoencoder as a feature extractor as represented in Fig. 2 below, which takes time series as input and produces the extracted feature vector. It solves two problems:

i. Reduces the storage and computation resource requirement of high dimensional data
ii. Using an encoder, the downstream model becomes agnostic of the lag values

These algorithms are essentially used for dimensionality reduction (and compression) algorithms with a few fundamental properties:

- Data specificity: The data specificity of autoencoders means they can only compress similar files to what they have been trained on. Unlike standard data compression algorithms such as gzip, they learn characteristics specific to the training data. Compressing landscape photos with an autoencoding algorithm trained on handwritten digits is impossible.
- Lossy: A lossy autoencoder means that the output will not be the same as the input but rather a close representation that has degraded quality. This is not the best option if you want lossless compression.
- Unsupervised: We can train an autoencoder unsupervised by feeding the input data directly to it without any fancy calculations. Autoencoders are classified as unsupervised learning strategies since they do not require explicit labels. They are self-supervised since they produce their labels using the training of the data.

3.4. LSTM

LSTM is a type of RNN. The output from the previous step is used to feed the current step's input. Highreiter & Schmidhuber [10] designed the LSTM. There was a detailed discussion in the paper of the problem of long-term dependency of RNNs. In this case, the RNN can't forecast words stored in long-term memory but can give more accurate predictions from recent data. RNN becomes inefficient with increasing gap length. LSTMs can retain information for a long time by default. The time series data it processes and predicts can be used for predicting, processing, and classifying. Eqs. (4)–(9) represent the computations involved in updating the cell state and hidden state within a LSTM.

$$f_t = \varphi_g\left(W_f . [h_{t-1}, x_t] + d_f\right) \tag{4}$$

$$i_t = \varphi_g(W_i . [h_{t-1}, x_t] + d_i) \tag{5}$$

$$X_t = \varphi_c(W_c . [h_{t-1}, x_t] + d_c) \tag{6}$$

$$C_t = (f_t * C_{t-1} + i_t * X_t) \tag{7}$$

$$O_t = \varphi(W_o . [h_{t-1}, x_t] + d_o) \tag{8}$$

$$h_t = O_t * \varphi_c(C_t) \tag{9}$$

The architecture of LSTM is displayed in Fig. 3. LSTMs are the extension or enlargement of RNNs that were brought in to address failures of RNN architecture. RNNs are a type of network that uses hidden output as input for the next time stamp, storing its memory for short periods and applying feedback from the previous output. Among the many applications of the technology, the most common ones are speech processing and music composition. RNNs suffer from some disadvantages as well. The first limitation is that information cannot be stored for an extended period. Often, a model to predict or forecast the current output must refer to recent data. It is, however, impossible for RNNs to deal with long-term dependencies of this kind.

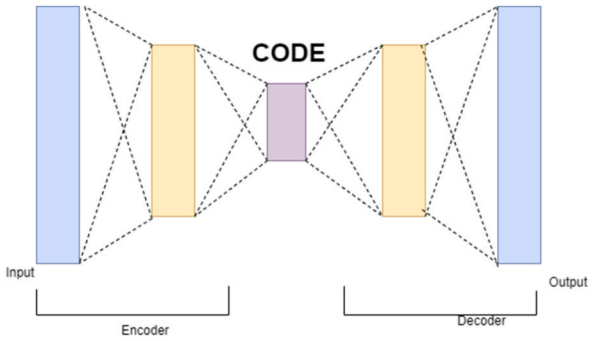

Fig. 2. Basic autoencoder model.

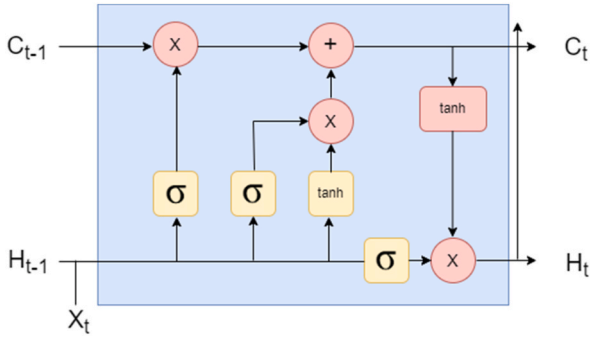

Fig. 3. The architecture of LSTM Block.

Furthermore, another limitation is that carrying forward parts of the context can be controlled more precisely while letting go of others. During the backtracking process of a network, an RNN may experience exploding and vanishing gradients. Thus, the LSTM model was developed and designed so that the vanishing gradient problem no longer exists, while the training model is left undamaged and unaltered. It can also be used to use LSTMs for solving long-time lag problems involving distributed representations, noise, and continuous values. Unlike the hidden Markov model (HMM), LSTMs do not require a finite number of states. There is an extensive range of parameter options with LSTMs, such as learning rates and input and output bias, so no fine adjustments are required. The essential advantage of the LSTM is that the weights can be updated in O (1) time, like Back Propagation Through Time (BPTT).

3.5. Convolutional network

In deep learning, a convolutional neural network is called CNN or ConvNet. This category of deep neural networks is primarily employed to analyze visual images presented as input. ConvNet visualizes its network as a matrix multiplication instead of standard neural networks. Convolution is one of the techniques used in this method. As we learned, convolution is the mathematical operation that combines two functions to produce a third function that represents how one function changes the shape of the other. A CNN, An artificial neural network designed specifically for processing pixel data, is used for image recognition and processing. Several layers of artificial neurons make up convolutional neural networks. The activation value of artificial neurons is calculated by summing multiple inputs together, like their biological counterparts. ConvNets creates multiple activation functions for each layer based on the input image. The convolutional neural network's first layer extracts the input image's essential features, such as horizontal or diagonal edges. The output produced is then passed to the next layer of the network, which is used to detect more complex features of the image, like corners or edges. As we go deeper into the network, it can detect and classify the more complex attributes of the image, such as eyes, faces, objects, etc.

CNNs generate local features using convolving filters. Local correlation in time series forecasts is reflected by continuous changes over time. Sequence problems have always been solved using RNN models, such as LSTMs. Due to their inability to operate simultaneously, RNNs are slower than CNNs. Based on the above considerations, the CNN model is the framework for the overall model design. Also, CNN is mainly used to analyze images and is applied to multivariate time series data analysis and forecasting. CNN, which analyzes images, is also suitable for analyzing multivariate time series because they have the same 2-dimensional data structures as images.

3.6. Attention mechanism

The attention mechanism was introduced to enhance the efficacy of the encoder-decoder model in machine translation. The attention mechanism prioritizes the most relevant vectors with higher weights by employing a weighted combination of all encoded input vectors. This flexibility enables the decoder to leverage the most pertinent segments of the input sequence effectively.

In an autoencoder without an attention mechanism, the output at the decoder investigates the context vector generated at the end of the encoding part, which is the bottleneck layer. In an autoencoder with an attention mechanism, the output at the decoder tries to investigate multiple sections of the encoders. It validates which input feature must be given more attention to predict the output.

There is a class of encoder-decoder models known as sequence to sequence (Seq2Seq) [16,13] models, in which the encoder and decoder are RNNs. In the encoder, variable-length input sequences $x = (x1, x2, \ldots, xTx) \in \mathbb{R}Tx \times dx$ are converted into fixed-length vector representations (also called context vectors). In the decoder, these vector representations are converted back into variable-length sequences $y = (y1, y2, \ldots, yTy) \in \mathbb{R}Ty \times dy$. In general, the hidden state of an encoder network corresponds to a vector representation that summarizes the entire sequence. Autoencoders are Seq2Seq models with aligned time ($x = y$) input and output sequences, so inputs and outputs have equal lengths ($Tx = Ty$). It is mainly due to the incapacity of the intermediate vector representation z to capture the information of the entire input sequences that Seq2Seq models fail to handle long sequences or long time series. This way, the decoder can selectively attend only to encoded hidden states relevant to its function.

To capture the dependencies between different series, we apply a self-attentional module inspired by the transformer [28] since these networks can extract many features. As a result of learning its relationships with other learned representations, including itself,

the self-attention module becomes more aware.

3.7. Temporal convolution network (TCN)

Generally, CNNs use convolving filters to generate local features. A continuous change over time and within a short time frame reflects the local correlation in a time series forecast. An RNN model like the LSTM model has always been considered the most effective standard for solving sequence problems. However, the RNNs model cannot operate in parallel, making them much slower than CNNs. Based on these considerations, CNN is used as the framework for the overall model design.

Using the TCN [29] model, we modeled temporal relations using the concepts from both CNNs and RNNs [30]. Based on two fundamental principles, the TCN model can be summarized as follows:

- Since TCN uses causal convolution, its output is independent of future inputs. TCN models only depend on inputs before time t if they produce output at time t. This means there will be no leakage of information.
- TCN models, like RNN structures, can produce sequences of any length given inputs. As a means of accomplishing this objective, a 1D fully connected layer is used.

Furthermore, TCNs use one-dimensional dilated convolutions to capture long-term patterns. This convolution does not lose resolution as it increases the network's receptive field without using pooling operations [31]. To implement the activation function, Rectified Linear Units (ReLU) layers are utilized [32]. By concatenating several TCN blocks, the network can further enhance its receptive field [33]. Thus, the learning procedure becomes more complex as deeper architectures have many more parameters. A residual connection is made between the outputs of each TCN block. In deep architectures, residual connections have been found to improve performance [34]. Inputs of TCN blocks are added to outputs of other TCN blocks. TCNs have all the characteristics that make

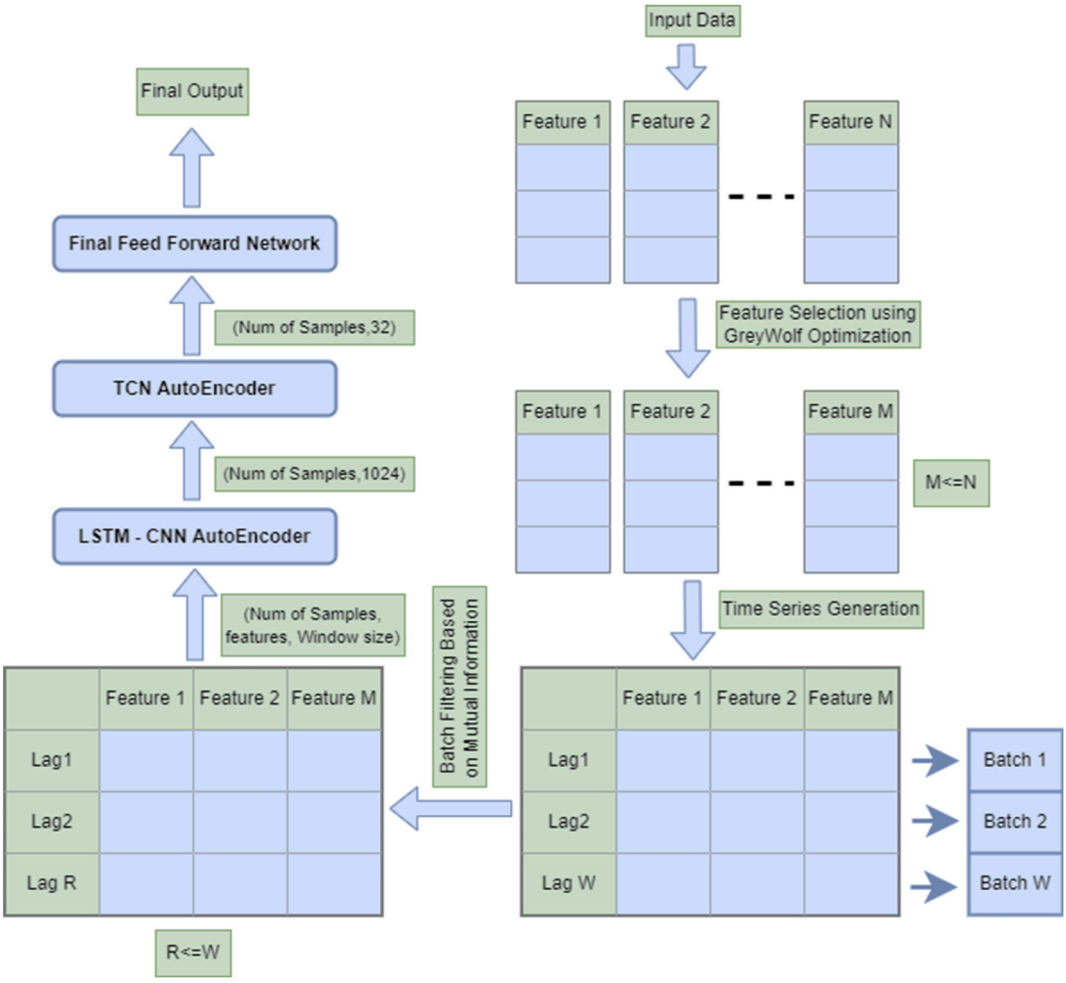

Fig. 4. Architecture of the proposed framework.

them an advantageous deep learning architecture for time series problems with complex time series representations. Like RNNs, TCNs have the advantage that their one-dimensional causal convolutional kernel can handle variable-length inputs. The shared convolution architecture of TCNs also allows them to process long sequences in parallel, making them more memory efficient than recurrent networks.

3.8. Architecture

We started our research by exploring essential data to determine the critical and insightful information from the data. Then, missing values are handled by interpolation or dropping (depending on the column and the number of null values). In the preprocessing, we also perform min-max scaling to shift all the features onto the same scale. Then, Finally, time series generation is done for different window sizes. Once the time series data was generated, we passed it through our proposed model for final forecasting.

The feature subset is selected by using the feature selection technique, allowing the relevant features to be analyzed further. Selecting the features involves using the grey wolf optimization technique to find the feature subset with maximum mutual information about the target variable. This is followed by selecting the features whose dimensions are closest to those of the target feature and dropping all the other features. Fig. 4 illustrates the graphical representation of the proposed framework's architecture.

Our research proposes a new architecture combining autoencoders and feedforward neural networks. Autoencoders enable us to extract and compress the features from the multivariate time series input. At the same time, the feedforward network helps us forecast the final output from extracted features collected using autoencoders. We can divide our complete architecture into two phases:

Pretraining Phase: Autoencoders usually come under this phase because, to use the autoencoders, we must first train them on relevant data from which we want to extract the features. Our architecture consists of two different autoencoders. The first autoencoder extracts the attributes from multivariate time series data, while the other autoencoder compresses those features into lesser dimensions.

Autoencoder1 (AE1): The main aim of this autoencoder is to extract and take out the attributes from the MTS. Firstly, the time series dataset (num of samples, window size, num of features) is passed as input and output for training the AE1. Once the AE1 gets to train, we again pass our data and collect the output vector of length 1024 from the bottleneck or shared layer, which can be treated as extracted features for time series data. Architecture of AE1 is displayed in below Fig. 5.

MTS input data is passed into parallel LSTM and Convolution layers of autoencoder (1). Convolution layers extract spatial or contiguous features from the input MTS in this setup, while LSTMs extract temporal or time-related features. LSTM extension with convolutional structures is proposed here to extract contiguous and temporal features hidden in the given multivariate time series data.

In MTS forecasting, a "temporal pattern" refers to a time-invariant pattern across the various time steps. The typical attention mechanism analyzes time steps appropriate for forecasting, which extracts information from them. MTS can be reconstructed using LSTMs by capturing the long-term temporal dependencies. A multivariate time series dataset was used to demonstrate the performance of temporal attention mechanisms on top of stacked LSTMs.

Then, the attention mechanism is applied to the model on the extracted features that checks which input features must be given more attention to predict the output. Then, the output of both layers is merged and considered the bottleneck layer. The bottleneck layer extracts the vector representation of MTS after training AE1. The output of the bottleneck layer is again passed to the LSTM and Conv layers to regain input MTS without much information loss.

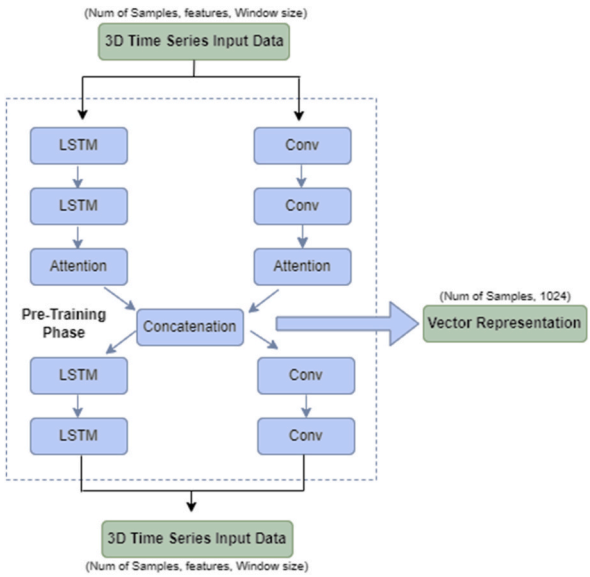

Fig. 5. Architecture of autoencoder 1.

We applied the attention mechanism because it is a technique of mimicking cognitive attention. A network should devote more attention to the small but essential parts of the input data by enhancing some parts and diminishing others. It helps in improving the results of the model.

LSTM- Conv Autoencoder Architecture:

- Input Layer: Our model begins with two separate input layers, each taking a time series input with a window size of 'k' and 'n_in' features.
- LSTM Layers: The first branch of the network comprises LSTM layers for sequence modeling. The initial LSTM layer with 64 units captures lower-level temporal features, followed by a second LSTM layer with 128 units to capture more complex dependencies. These layers use ReLU activation and return sequences for subsequent processing.
- Attention Mechanism: An attention mechanism follows the LSTM layers, enhancing the model's focus on relevant temporal features.
- Convolutional Layers: The second branch consists of convolutional layers, starting with a 1D convolution followed by max pooling. A similar pattern is repeated with increased depth (128 units). This branch is designed to extract spatial features from the time series data.
- Bottleneck Layer: The outputs from both branches are combined in a shared bottleneck layer, which serves as a fusion point of learned features.
- Upsampling Layers: After the bottleneck, the model splits into two paths again. For the LSTM path, 'RepeatVector' and LSTM layers reconstruct the time series data, while in the convolutional path, upsampling and convolution layers are used for reconstruction.

Autoencoder2 (AE2): AE2 compresses the extracted features collected from AE1 into a lesser length vector of 32. For training the AE2, we passed all of our extracted features from AE1 (number of samples, 512) as input and output. Along with it, we also passed the actual output as the output of AE2. Once the Autoencoder2 was trained, we again passed our extracted features data and collected the compressed output vector of length 32 from the bottleneck (shared) layer, which will be further passed into the final model. Architecture of AE2 is displayed in below Fig. 6.

The above architecture shows that after getting vector representation using AE1, we pass it to the TCN AE2, which will compress the size of the vector. Using the TCN autoencoder, we modeled temporal relations using the concepts from both Convolution and LSTM layers.

TCN-Conv Autoencoder Architecture:

- Input Layer: The model begins with an input layer that accepts shape sequences.
- TCN Layers: The core of the architecture comprises two TCN layers. The first TCN layer has 128 filters and uses a ReLU activation function, processing the input while maintaining sequence output. The second TCN layer reduces the number of filters to 64, continuing the pattern of sequence processing.
- Flattening and Dense Layers: After the TCN layers, the data is flattened and passed through two Dense layers with ReLU activation. The first Dense layer reduces the dimensionality to 32, and the second expands it back to 64. This configuration facilitates a form of bottleneck feature extraction within the network.
- Upsampling and Output Layers: The model then employs a 'RepeatVector' layer to reshape the data for sequence output, followed by a 'TimeDistributed' Dense layer that reconstructs the original input dimensions. This output represents the autoencoder's reconstruction of the input data. Additionally, a separate Dense layer acts as a predictor, providing another form of output from the model.

Pretraining phase Steps

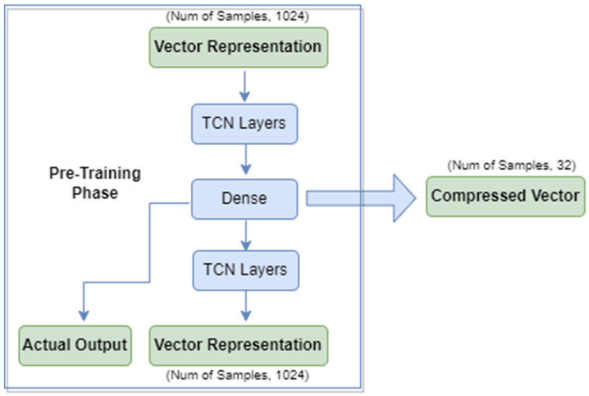

Fig. 6. The architecture of autoencoder 2.

- Train the AE1 on the whole dataset and save its best weights, which gives us the most negligible loss.
- Load the weights of AE1 to the bottleneck layer and extract the features of length 1024 for whole data in the form of a 1D vector.
- Train the AE2 on the extracted vectors from the first Autoencoder and save its best model weights.
- Load the AE2 and pass the extracted features of the train and test set collected using AE1 and the trained weights of AE2 to compress.
- The length of these features is 32.
- Train the final feedforward model on train compressed features and actual target and evaluate the final model on test data.

Fine-Tuning Phase: After getting vector representation using AE1, we pass it to the TCN AE2, which will compress the size of the vector. Using the TCN autoencoder, we modeled temporal relations using the concepts from both Convolution and LSTM layers. After applying TCN AE2 to a simple feedforward network that will forecast the target value, we pass the output we get. Because we are dealing with the regression type forecasting problem, the output layer has a linear function as activation, and for the optimizer, we used Adaptive Moment Estimation (Adam). At the end of this phase, we compare the forecasted results with the actual results and observe that our approach gives us better results than other authors' research.

Fig. 7 shows the final model we trained in the fine-tuning phase. Here, the 3D time series data first goes into AE1, and we get extracted features of length 512 from it. Then, these features pass through the AE2 to compress them in the vector of length 32, and these compressed vectors finally go from dense layers. The last dense layer with one neuron and the 'linear' activation function is responsible for final forecasting.

4. Case study

This section explains details of the case study undertaken as a part of this research. The section explains the datasets, experiment configurations, settings, and, finally, the results of the experiments.

4.1. Dataset

Table 1 displays the detail of the dataset. We have taken three case studies across domains to validate the proposed and reference models. Three benchmark datasets are taken in each of the case studies.

Beijing $PM_{2.5}$ Concentration: Numerous disciplines in the literature have used this dataset as a benchmark in their respective studies. It is a huge dataset that includes hourly air concentration measurements in several Chinese cities. In the context of our research, we have analyzed all the data collected in Beijing between January 1, 2010 and December 31, 2014 [35]. Out of the total 43, 744 observations in the dataset, we have divided it into two groups, one with 34,984 observations for training and the second group with 8760 observations for testing. This dataset is being used for experiments that require forecasting future hourly concentration based on observations of the other variables.

The Solar Radiation Dataset: The dataset utilized in this study is referred to as HI-SEAS (Hawaii Space Exploration Analog and Simulation), sourced from Kaggle under the task "Solar Radiation Prediction" from the NASA Hackathon. Weather stations were collected during the four months between Mission IV and Mission V (September through December 2016) [36,37]. The dataset contains a wide range of weather parameters that are statistically analyzed. Various meteorological parameters are included in the

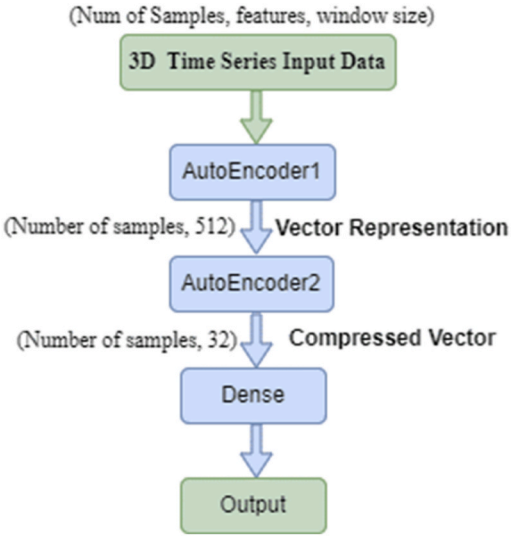

Fig. 7. Fine tuning final model architecture.

Table 1
Dataset details.

Name of Dataset	Number of Attributes	Training Size	Testing Size	Description
Energy Appliances	29	15,464	4239	A regression model was constructed using experimental data about appliances in a low-energy building.
Beijing PM2.5 Concentration	18	34,952	8760	Time series dataset(hourly) of $PM_{2.5}$amount in air over the Beijing between January 1st, 2010, and December 31st, 2014
Solar Radiation	11	22,857	9797	In the four months from September to December 2016, meteorological data was collected at the HI-SEAS weather station between Mission IV and V.

dataset, including radiation, temperatures, pressures, etc. The dataset's attributes are also statically analyzed as part of the analysis.

Energy Appliances Dataset: This dataset is sourced from the Machine Learning Repository of UCI. This dataset has a total of 19,735 instances. The dataset is composed of 29 overall features. As a part of the data cleansing, it was identified that there is no missing value in the dataset. The observations are recorded at an interval of 10 min for around 4.5 months. The ZigBee wireless sensor network is used to examine the humidity and temperature conditions of the house. Approximately 3.3 min were used to transmit humidity and temperature conditions, after which the wireless data were averaged over 10 min. Energy data was recorded every 10 min using the Mbus energy meter. A data set from Reliable Prognosis (rp5.ru) was downloaded from the nearest airport weather station (Chievres Airport, Belgium) and merged with the experimental data sets using the time and date columns. Two random variables were included in the data set for testing regression models and filtering out non-predictive features (attributes).

4.2. Performance measures

The model errors are measured to evaluate the performance of the forecasts. The errors are calculated at the identical scale of the data. In this study, we have used two different error measures, i.e., root mean square error (RMSE) and mean absolute error (MAE).

RMSE helps quantify the average magnitude of errors. An average of squared differences between the forecasted and observed values is calculated by taking the square root of that difference. It is represented as per Eq. (10).

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \overline{y_t})^2} \tag{10}$$

MAE helps quantify the average magnitude of errors irrespective of their direction. The absolute differences between the forecasted and observed values are calculated as an average over samples. It is represented as per Eq. (11).

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|y_t - \overline{y_t}| \tag{11}$$

4.3. Parameters setting

In this section, we have illustrated the settings of the models. The values of the hyper-parameters are specified in Table 2 below.

4.4. Results

This section discusses and explains the study's results that assess the proposed model. During the evaluation and benchmarking of the proposed model, we evaluate its performance, and the comparison is made against the performance and results of the reference models, DSLTM and LSTM-SAE. We used the same datasets in the reference models to compare the proposed model with the benchmark sets. The presented results on all three datasets are on the testing dataset.

4.4.1. Hourly PM2.5 concentration in Beijing

In our research, the baseline models primarily include deep LSTM models, as detailed in the base paper's abstract. The deep LSTM

Table 2
Model parameter settings.

Model Parameter	Value
Deep Learning Model Batch Size	1024
Number of Layers	3
Learning Rate	0.001
Window Size	32/64
Activation	ReLU, linear
Optimizer	Adam

models, despite their proven accuracy in time series data modeling, exhibit limitations, particularly in processing multivariate time series data with high nonlinearity and long intervals. These models suffer from issues related to random weight initialization in recurrent networks, which hinders the effective learning of latent features in correlated variables of MTS datasets. Our proposed framework addresses these limitations by incorporating a multi-phase feature selection technique and a customized stacked autoencoder strategy, enhancing the model's ability to capture complex, non-linear relationships in the data. The first autoencoder in our model explicitly targets the random weight initialization problem, offering a more robust foundation for feature learning. The second autoencoder leverages convolution and recurrent neural networks to model temporal relations more effectively. By comparing our model with these baseline LSTM models, we demonstrate significant improvements in metrics like MSE and MAE across various datasets, including PM2.5 air quality data.

Table 3 presents the results and analysis of the proposed models with the different lag and the reference models. From the evaluation metrics, it can be easily deduced that the proposed model with Adam optimizer outclasses the reference models with either lag. This proves that the proposed strategy undoubtedly improves forecasting. The proposed model is better than reference models concerning error metrics MAE and RMSE. Comparing the proposed model where the Adam optimizer is used gives better results. The proposed model with Adam optimizer attains the best overall RMSE and MAE across all the models. It brings around ∼**51.6%** improvement in RMSE and around ∼**39.7%** in MAE. Figs. 8 and 9 show the graphical relationship and variance between the predicted and original observed values.

4.4.2. Solar radiation

As mentioned in the base paper, our study compared our novel prediction framework to two baseline models, the Average Ensemble and the KNN Ensemble. The average Ensemble method is a straightforward ensemble technique that combines the outputs of various base regressors like LSTM, SVM, and NN. It calculates the mean of these outputs. While this method provides a basic ensemble approach, it may not be optimal in handling complex relationships in time series data, mainly when dealing with non-linear and intercorrelated features. This limitation becomes more pronounced in the context of solar radiation datasets, which often exhibit complex temporal dynamics. KNN is a classic regression method that bases its predictions on the proximity of data points. It identifies k nearest neighbors in the training data and uses them for prediction. One major shortcoming of this method is selecting an appropriate value of k, which can significantly impact model performance. Moreover, KNN may struggle with large datasets and not effectively capture the complex temporal and non-linear relationships inherent in solar radiation data. Our proposed model addresses these limitations with its multi-phase feature selection and custom-stacked autoencoder approach. The first autoencoder in our model addresses the random weight initialization problem common in neural networks. In contrast, the second autoencoder employs convolution and recurrent neural networks to model complex temporal relationships in the data more effectively.

Table 4 represents the results and analysis of the proposed models with different lags and the reference models. From the evaluation metrics, it can be easily deduced that the proposed model with Adam optimizer outclasses the reference models with either lag. This proves that the proposed strategy indeed improves forecasting. The proposed model is better than reference models about error metrics MAE and RMSE. Comparing the proposed model where the Adam optimizer is used gives better results. The proposed model with Adam optimizer attains the best overall RMSE and MAE across all the models. It brings around ∼**34.7%** improvement in RMSE and around ∼**75%** in MAE. Figs. 10 and 11 exhibit the visual relationship and variance between the predicted and the actual values.

4.4.3. Energy appliance dataset

The ARIMA-GARCH model combines the ARIMA and GARCH methodologies. ARIMA is renowned for its capability to model a wide range of time series data by understanding the underlying patterns, such as trends and seasonality. GARCH, on the other hand, is effective in modeling the volatility or variability in time series data, especially in financial markets. These models have certain limitations when dealing with highly non-linear and complex time series datasets, such as those involving energy appliance usage. ARIMA models, for instance, assume linearity and stationarity in the data, which might not be the case in real-world scenarios with energy datasets. GARCH models, while good at capturing volatility, might not effectively handle the non-linear interdependencies in time series data of energy appliances. With its multi-phase feature selection technique and a blend of long-short-term memory-based autoencoder and temporal convolution-based autoencoder, our proposed framework is designed to overcome these limitations. By incorporating machine learning techniques, particularly deep learning, our model is better equipped to capture and model the non-linear, complex patterns in energy appliance datasets. The experimental results, demonstrating significant improvements in metrics like MSE and MAE, underscore the efficacy of our approach in handling the complexities of real-world datasets compared to the traditional ARIMA-GARCH model.

Table 5 represents the results and analysis of the proposed models with different lags and the reference models. It can be easily deduced from evaluation metrics that the proposed model with Adam optimizer outclasses the reference models with either lag. This proves that the proposed strategy undoubtedly improves forecasting. The proposed model is better than reference models about error

Table 3
Results for $PM_{2.5}$ concentration dataset.

Model	Lag	RMSE	MAE
LSTM-SAE [1]	25	24.04	12.06
DLSTM [1]	30	23.74	12.45
Proposed model	**64**	**11.62**	**7.505**

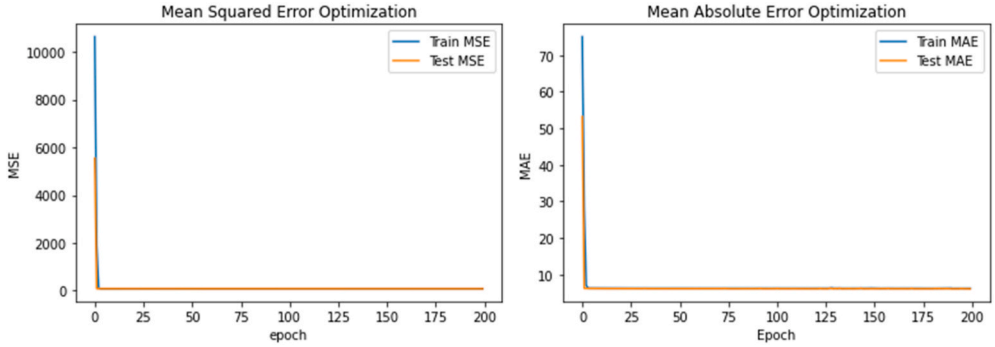

Fig. 8. Training of proposed models on $PM_{2.5}$ dataset.

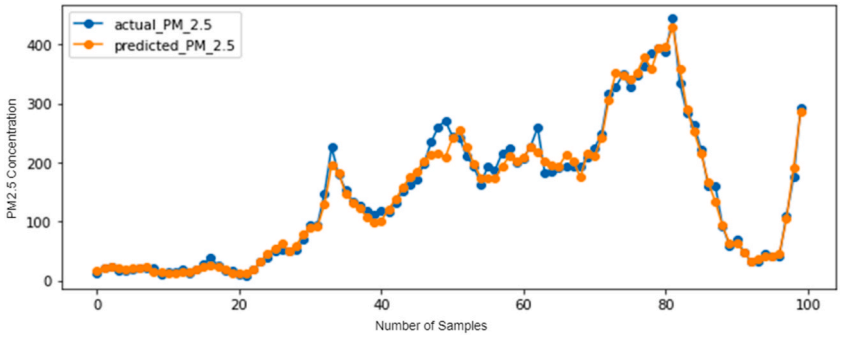

Fig. 9. Actual vs predicted for $PM_{2.5}$ dataset.

Table 4
Results of the solar radiation dataset.

Model	Lag	RMSE	MAE
Average Ensemble [38]		0.037	0.031
KNN Ensemble [38]		0.023	0.028
Proposed model	32	0.015	0.007

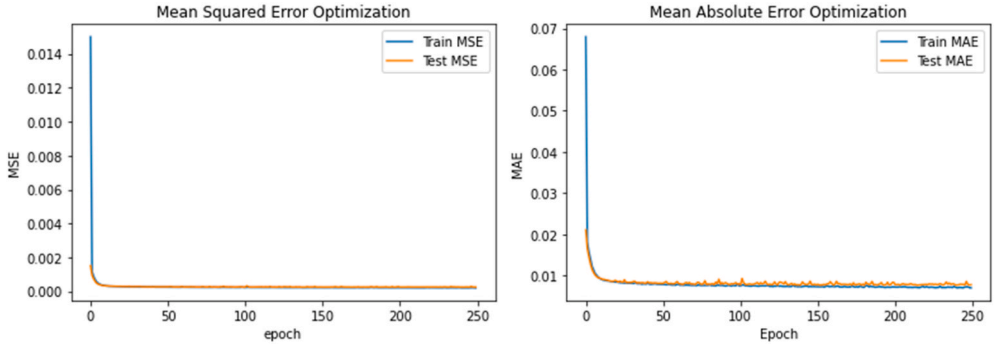

Fig. 10. Training of proposed models on solar radiation dataset.

metrics MAE and RMSE. Comparing the proposed model where the Adam optimizer is used gives better results. The proposed model with Adam optimizer attains the best overall RMSE and MAE across all the models. It brings around ~**23.8%** improvement in RMSE and around ~**10.7%** in MAE. Figs. 12 and 13 exhibit the visual relationship and variance between the predicted and the actual values.

These results establish that the proposed model is better in performance and precision than the state-of-the-art models like DLSTM and LSTM-SAE. This proves that stacking the LSTM and convolutional layers in parallel in an autoencoder fashion results in much faster and better performance. Moreover, feeding the output observation in the second autoencoder has also resulted in better results.

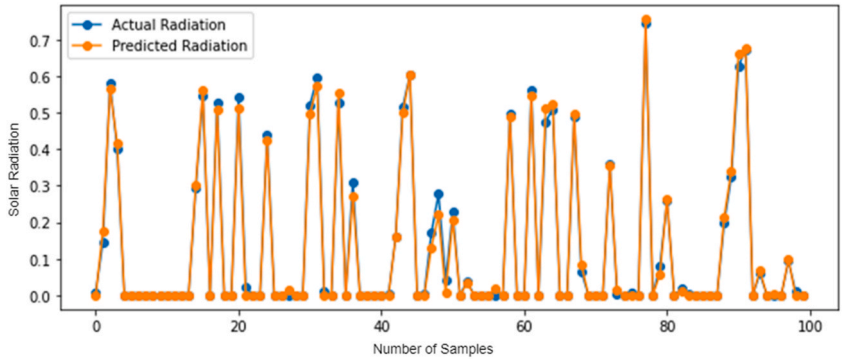

Fig. 11. Actual vs. Predicted solar radiation data.

Table 5
Results of the energy appliance dataset.

Model	Lag	RMSE	MAE
ARIMA-GARCH [2]		1.55	1.11
Attention-RNN		0.84	0.40
DNN		0.63	0.28
Proposed model	32	0.48	0.25

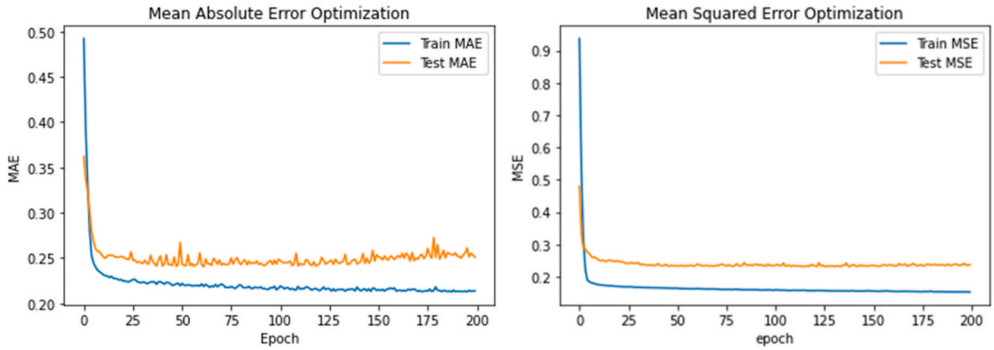

Fig. 12. Training of proposed models on the Energy Appliance dataset.

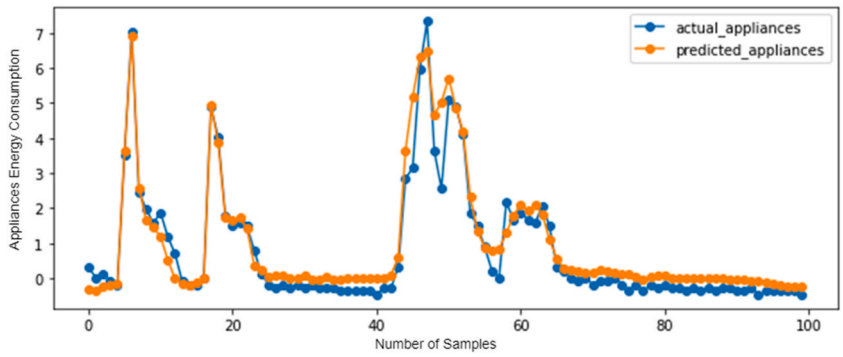

Fig. 13. Actual vs Predicted for Energy appliance dataset.

This stacking of layers results in better performance than the earlier models, which were based on random weight initialization in LSTM. In a multivariate time series problem, there is a set of highly correlated variables. When the weights or units are randomly initialized, the model can be trapped in local minima. However, the unsupervised pre-learning of the model results in better weight

initialization. It was found that LSTMs can predict words from long-term memory more accurately than RNNs based on recent information because of their ability to adjust for long-term dependencies. LSTM time series forecasting models can predict future values by analyzing previous, sequential data. Time series forecasting problems have recently shown an increasing interest in CNNs among researchers. In our base paper, the autoencoder consists of only LSTM layers, so we extend their work to figure out how conv layers are helping us in forecasting.

5. Conclusions

The study proposes and implements a novel stacked autoencoder-based deep learning forecasting framework that utilizes a combination of two autoencoders; AE1 is a combination of LSTMs and convolutional-1D layers to solve the problem of random weight initialization, and AE2 uses the TCN layers to extract the temporal features. This proposed model enhances the capability of extracting and learning the underlying non-linear relationships in time series data. A hybrid feature selection technique is proposed to solve the Curse of Dimensionality problem in multivariate time series data. This technique extracts relevant features using grey wolf optimization and then finds the relevant lag values in window lengths based on the mutual information technique to solve the Curse of Dimensionality issue. The research used four real-world-based time series datasets from different areas to substantiate the proposed forecasting model's interpretation, generalization ability, and accuracy. PM2.5, solar radiation, and electricity appliances. All these datasets are used for training purposes, and the study proposes a framework combining multiple such network stacks. To train and test, the modeling was performed using the adaptive moment optimizer. The results conclude that the proposed hybrid feature selection technique improves training performance.

The main conclusions of this study are as follows:

- The overall performance of the proposed framework has an evident and apparent advantage over the established benchmarks observed during the experiments on the case studies.
- The performance of the enhanced adaptive moment optimizer is better than the standard optimizer concluded based on the statistical measures obtained in the experiments on the case studies.
- Enhanced adaptive moment optimizer improved the overall performance of the framework by reducing the training time as well.
- The stacked deep learning model is sensitive to the optimizer and outclasses the existing models [39,22] on the same datasets. The same model could efficiently exploit the time structure across the time series of both datasets and prove to be excellent in generalization and forecasting accuracy.

Despite the proposed model achieving better results than the previous standard models, it still has some limitations. The proposed model is restricted to extracting features based on mutual information; it can be extended so that feature selection can be done on multiple objectives. Also, due to the complex nature of the proposed model, it takes computation time, storage, memory, and resources in the training and testing compared to the standard models.

Funding

This work was supported by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2024R 308), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Data availability statement

The datasets used in this paper are publicly available, and their links are provided in the reference section.

CRediT authorship contribution as it

Dilip Kumar Sharma: Writing review & editing, Writing original draft, Visualization, Validation, Supervision, Formal analysis, Conceptualization. **Ravi Prakash Varshney:** Writing review & editing, Writing original draft, Visualization Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Saurabh Agarwal:** Writing review & editing, Visualization, Formal analysis, Conceptualization. **Amel Ali Alhussan:** Writing review & editing, Writing original draft, Project administration, Investigation, Funding acquisition, Data curation, Conceptualization. **Hanaa A. Abdallah:** Writing review & editing, Writing Writing original draft, funding acquisition,formal analysis ,data curation ,conceptualization

Declaration of competing interest

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

Acknowledgments

The authors extend their appreciation to Princess Nourah bint Abdulrahman University Researchers Supporting Project number

(PNURSP2024R 308), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

References

[1] A. Sagheer, M. Kotb, Unsupervised pretraining of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems, Sci. Rep. 9 (1) (2019) 19038.
[2] Jun Hu, Wendong Zheng, Transformation-gated LSTM: efficient capture of short-term mutation dependencies for multivariate time series prediction tasks, IJCNN 2019 (2019) 1–8. July 14 - July 19.
[3] G.E.P. Box, G.M. Jenkins, G.M. Ljung, Time Series Analysis: Forecasting and Control, fifth ed., John Wiley & Sons, New Jersey, 2016.
[4] N. Meade, T. Islam, Forecasting in telecommunications and ICT-A review, Int. J. Forecast. 31 (4) (2015) 1105–1126.
[5] M. Fagiani, S. Squartini, L. Gabrielli, S. Spinsante, F. Piazza, A review of datasets and load forecasting techniques for smart natural gas and water grids: analysis and experiments, Neurocomputing 170 (25) (2015) 448–465.
[6] A. Tealab, Time series forecasting using artificial neural networks methodologies: a systematic review, Futur. Comput. Informatics J. 3 (2018) 334–340.
[7] A. Sagheer, M. Kotb, Time series forecasting of petroleum production using deep LSTM recurrent networks, Neurocomputing 323 (2019) 203–213.
[8] Y. Zheng, Q. Liu, E. Chen, Y. Ge, J. Zhao, Exploiting multi-channels deep convolutional neural networks for multivariate time series classification, Front. Comput. Sci. 10 (2016) 96–112.
[9] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, in: S.C. Kremer, J. F. Kolen (Eds.), A Field Guide to Dynamical Recurrent Neural Networks, Wiley-IEEE Press, 2001.
[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.
[11] W. Cao, X. Wang, Z. Ming, J. Gao, A review on neural networks with random weights, Neurocomputing 275 (2018) 278–287.
[12] P. Saikia, R. Baruah, An empirical study on unsupervised pretraining approaches in regression problems, Proceeding of The 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (2018) 342–349.
[13] Z. Tang, D. wang, Knowledge Transfer Pretraining. Tech. Rep, Research Institute of Information Technology, Tsinghua University, 2015.
[14] M. Yaghini, M. Khoshrafar, M. Fallahi, A hybrid algorithm for artificial neural network training, Eng. Appl. Artif. Intell. 26 (2013) 293–301.
[15] M. Längkvist, L. Karlsson, A. Loutf, Review of unsupervised feature learning and deep learning for time-series modeling, Pattern Recogn. Lett. 42 (2014) 11–24.
[16] J. Schmidhuber, Learning complex, extended sequences using the principle of history compression, Neural Comput. 4 (1992) 234–242.
[17] Netzwerkarchitekturen, Zielfunktionen und Kettenregel (Network architectures, objective functions, and chain rule), Habilitation thesis, Institut für Informatik, (TUM), 1993.
[18] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, Why does unsupervised pretraining help deep learning? J. Mach. Learn. Res. 11 (2010) 625–660.
[19] T. Le Paine, P. Khorrami, W. Han, T. Huang, An analysis of unsupervised pretraining in light of recent advances, arXiv preprint arXiv 1412 (2015) 6597.
[20] Yan Xia, Xudong Cao, Fang Wen, Gang Hua, Jian Sun, Learning discriminative reconstructions for unsupervised outlier removal, ICCV (2015) 1511–1519.
[21] Dauphin Yann N., Fan Angela, Auli Michael, Grangier David, Language modeling with gated convolutional networks, ICML'17: Proceedings of the 34th International Conference on Machine Learning. 70 (2017) 933–941.
[22] Aäron van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, Alex Graves, Nal Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: a generative model for raw audio, SSW (2016) 125.
[23] Dieleman Sander, Aaron van den Oord, Karen Simonyan, The challenge of realistic music generation: modelling raw audio at scale, arXiv preprint arXiv: 1806.10474 (2018).
[24] Shaojie Bai, J Zico Kolter, Vladlen Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv preprint arXiv: 1803.01271 (2018).
[25] J.Y. Yang, S. Olafsson, Optimization-based feature selection with adaptive instance sampling, Comput. Oper. Res. 33 (11) (Nov, 2006) 3088–3106.
[26] Y. Qi, Q. Li, H. Karimian, D. Liu, A hybrid model for spatiotemporal forecasting of pm2.5 based on graph convolutional neural network and long short-term memory, Sci. Total Environ. 664 (2019) 1–10.
[27] C. Huang, P. Kuo, A deep CNN-LSTM model for particulate matter (PM2.5) forecasting in smart cities, Sensors 18 (2018).
[28] A. Berg, M. O'Connor, M.T. Cruz, Keyword transformer: a SelfAttention model for keyword spotting, Proc. Interspeech 2021 (2021) 4249–4253.
[29] K.K.R. Samal, K.S. Babu, S.K. Das, Temporal convolutional denoising autoencoder network for air pollution prediction with missing, values 38 (2021).
[30] Hourly Heat Load Prediction Model Based on Temporal Convolutional Neural Network.
[31] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016.
[32] V. Nair, G. Hinton, Rectified linear units improve Restricted Boltzmann machines, in: Proceedings of the ICML 2010—27th International Conference on Machine Learning, Israel, Haifa, 21–24 June 2010, pp. 807–814.
[33] Van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.W.; Kavukcuoglu, K. WaveNet: A generative model for raw audio. arXiv 2016, arXiv:1609.03499.
[34] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016, pp. 770–778.
[35] X. Liang, et al., Assessing Beijing's PM2.5 pollution: severity, weather impact, apec and winter heating, in: Proceedings of the RoyalSociety A Mathematical Physical and Engineering Sciences, 2015.
[36] Hawai'i Space Exploration Analog and Simulation.Accessed: Oct. 4, 2020. [Online]. Available: https://hi-seas.org.
[37] Solar Radiation Prediction Task from NASA Hackathon.Accessed: Oct. 4, 2020. [Online]. Available: https://www.kaggle.com/dronio/SolarEnergy Schmidhuber, J.
[38] E.-S.M. El-Kenawy, et al., Advanced ensemble model for solar radiation forecasting using sine cosine algorithm and Newton's laws, IEEE Access 9 (2021) 115750–115765, https://doi.org/10.1109/ACCESS.2021.3106233.
[39] T. Tieleman, G. Hinton, Lecture 6.5 - RMSProp, COURSERA: neural networks for machine learning, Technical report (2012).