Expert Systems With Applications 223 (2023) 119879

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Real-time forecasting of time series in financial markets using sequentially trained dual-LSTMs

Kelum Gajamannage [a,*], Yonggi Park [a], Dilhani I. Jayathilake [b]

[a] Department of Mathematics and Statistics, Texas A&M University - Corpus Christ, 6300 Ocean Dr., Corpus Christi, TX 78412, USA
[b] Department of Physical and Environmental Sciences, Texas A&M University - Corpus Christ, 6300 Ocean Dr., Corpus Christi, TX 78412, USA

ARTICLE INFO

Keywords:
Many-to-one LSTM
Sequential training
Real-time forecasting
Time series
Financial markets

ABSTRACT

Financial markets are highly complex and volatile; thus, accurate forecasting of such markets is vital to make early alerts about crashes and subsequent recoveries. People have been using learning tools from diverse fields such as financial mathematics and machine learning to make trustworthy forecasting on such markets. However, the accuracy of such techniques had not been adequate until artificial neural network frameworks such as long short-term memory (LSTM) were utilized. Moreover, making accurate real-time forecasting, also known as nowcasting, of financial time series is highly subjective to the LSTM's architecture in use and the procedure of training it. Herein, we forecast financial markets in real-time by training a dual version of LSTM which forecasts only one time step at each iteration so that the forecast for this iteration will be in the input for the next iteration. Semi-convergence is a prominent issue in a recurrent LSTM setup as the error could propagate through iterations; however, the duality of this LSTM aids in dwindling this issue. Especially, we employ one LSTM to find the best number of epochs associated with the least loss and train the second LSTM only through that many epochs to make forecasting. We treat the current forecast as a part of the training set for the next forecast and train the same LSTM. While classic ways of training cause more error when the forecast is made further away through the test period, our approach offers superior accuracy as the training increases when it proceeds through the testing period. The forecasting accuracy of our approach is validated using three time series from each of the three diverse financial markets: stock, cryptocurrency, and commodity. The results are compared with those of a single LSTM, an extended Kalman filter, and an autoregressive integrated moving average model.

1. Introduction

Financial markets refer broadly to any marketplace that entitles the trading of securities, commodities, and other fungibles, and the financial security market includes stock market, cryptocurrency market, etc. Bahadur et al. (2019). Among the three markets, which are stock, cryptocurrency, and commodity, the stock markets are well known to people while the other two are not. The cryptocurrency market exchanges digital or virtual currencies between peers without the need for a third party such as a bank (Squarepants, 2022), but a commodity market trades raw materials such as gold and oil rather than manufactured products. These markets are both highly complex and volatile due to diverse economical, social, and political conditions (Qiu et al., 2020). Learning such markets for the sake of making forecasting is vital because that aids market analysts to make early alerts about crashes and subsequent recoveries so that the investors can either obtain better precautions for future crashes or gain more profit under future recoveries. Since it is unreliable and inefficient to rely only on a trader's personal experience and intuition for the analysis and judgment of such markets, traders need smart trading recommendations derived from scientific research methods.

The classical methods of forecasting time series data are mostly linear statistical approaches where they assume linear relationships between the current output and the previous inputs. Autoregressive (AR) (Shumway et al., 2000), is such a model that predicts the value for the current time step based on a linear relationship between some fixed number of recent observations as many time series exhibit autocorrelation, i.e., linear association between lagged observations. Moving average (MA) (Shumway et al., 2000), is another linear model that captures serial autocorrelation in a time series by expressing the conditional

* Corresponding author.
E-mail addresses: kelum.gajamannage@tamucc.edu (K. Gajamannage), yonggi.park@tamucc.edu (Y. Park), dilhani.jayathilake@tamucc.edu (D.I. Jayathilake).

https://doi.org/10.1016/j.eswa.2023.119879
Received 9 May 2022; Received in revised form 1 March 2023; Accepted 13 March 2023
Available online 16 March 2023

mean as a function of past innovations. Autoregressive integrated moving average (ARIMA) (Shumway et al., 2000), is made by combining a differential version of the AR model into the MA model. Autoregressive fractionally integrated moving average (ARFIMA) (Shumway et al., 2000), provides a parsimonious parameterization of the long-memory process by extending the short-memory process of the ARMA model by combining a fractional differential version of the AR model into the MA model. Thus, these frameworks often do not capture nonlinear relationships in the data and cannot cope with certain complex time series. Since financial time series are nonstationary, nonlinear, and contaminated with high noise (Bontempi et al., 2013), traditional statistical models have some limitations in predicting financial time series with high precision.

Nonlinear time series forecasting models have shown trustworthy performance in the forecasting of financial time series data. The threshold autoregressive (TAR) model (Tsay, 1989), is only piecewise linear in which the thresholding process divides one-dimensional Euclidean space into some fixed number of regimes where a linear autoregressive model is utilized in each regime. Smooth transition autoregressive (Terasvirta & Anderson, 1992), allows transitions along a continuous scale making the regime-switching process smooth which helps overcome the abrupt switching of parameter values characteristic of simpler TAR model. The Markov-switching model (Hamilton, 1989), is also a regime-switching model which assumes that unobserved states are determined by an underlying stochastic process, known as a Markov chain, determining how uncertain an unobserved outcome is to occur. Both nonlinear and purely data-driven machine learning approaches such as Artificial Neural Networks (ANNs) are adopted to forecast nonlinear and nonstationary time series data with both high efficiency and better accuracy (Gu et al., 2020). ANNs are accurate time series predictors due to adaptive self-learning capabilities (Gajamannage et al., 2022).

Recurrent Neural Networks (RNNs) are powerful and robust types of ANNs that belong to the most promising algorithms in use because of their internal memory (Gajamannage et al., 2023). This internal memory remembers its inputs and helps RNNs to find solutions for a vast variety of problems (Ma & Principe, 2018). Especially, the regular RNN framework has been recently improved by utilizing diverse smart extensions including wavelet denoising-based ANN (Wang et al., 2012), globally optimized ANN (Kim & Ahn, 2012), and ARIMA-RNN hybrid model (Rather et al., 2015), to name a few. RNN is optimized with respect to its weights to fit the training data by adopting a technique called backpropagation that requires the gradient of the RNN. However, the gradient of RNN may vanish and explode during the optimization routine which hampers RNN's learning ability of long data sequences (Allen-Zhu et al., 2019). As a solution to these two problems (Le & Zuidema, 2016), the LSTM architecture (Hochreiter & Schmidhuber, 1997), which is a special type of RNN, is often used. LSTMs are explicitly designed to learn long-term dependencies of time-dependent data by remembering information for long periods. LSTM performs faithful learning in applications such as speech recognition (Kim et al., 2017; Tian et al., 2017) and text processing (Shih et al., 2018; Simistira et al., 2015). Especially, LSTMs are suitable to analyze complex data sequences such as financial market time series since they have internal memory, have the capability of customization, and are free from gradient-related issues (Bhandari et al., 2022). Kim and Won (2018) forecasts the volatility of the stock prices using a hybrid model integrating LSTMs with generalized autoregressive conditional heteroskedasticity models. Baek and Kim (2018) forecasts stock prices using an LSTM model that has guarantees of preventing overfitting. Adaptative batch-normalized versions of LSTMs are also used to forecast financial time series as presented in Fang et al. (2023). Thus, LSTMs are widely applicable for the analysis of stock markets that include next day's close price prediction (Gao et al., 2017), stock indices forecasting (Roondiwala et al., 2017), and stock characteristics analysis (Liu et al., 2018), to name a few.

We adopt a real-time iterative approach to sequentially train LSTMs that predicts only one time step ahead at each iteration so that the forecast for this iteration will be in the input for the next iteration. To implement the notion of a single time step future prediction for an input sequence of immediate past time series, many-to-one ANN architecture is adopted so that the many input nodes receive the input sequence and the one output node passes the prediction. This single time step forecasting technique is also known as nowcasting; however, we call it real-time forecasting in this paper. Due to our recurrent training, validation, and forecasting procedure, effective optimization of the loss function with respect to the epochs is nonviable if we only used one LSTM. Especially, semi-convergence of the loss function could be prominent under a such procedure. In fact, this semi-convergence of the loss function during some iterations of such recurrent training/forecasting routing might be intensified as it propagates through iterations and eventually encounters significant forecasting error. Validation, e.g., walk-forward validation, is widely used in deep learning of sequential data, including time series, to determine the best number of epochs (Goodfellow et al., 2016). Moreover, some research suggests that the validation could be incorporated into LSTM-based time series forecasting schemes if autocorrelation is present in the data (Broersen, 2006), which exists in most financial time series. Validation with a single LSTM requires either initialization of the weights of the ANN before each recurrent training/forecast iteration or initialization of them only one time for all the iterations. However, while the initialization of the weights of the ANN at each recurrent forecasting iteration avoids the capability of learning beyond the input data sequence, training the same ANN over each iteration would degrade the training performance by weak samples so the overall forecast might be erroneous. Early stopping is another technique in deep learning to determine the best number of epochs in which the user specifies a threshold for loss so that the deep learning tool, e.g. LSTM, stops training through epochs when it meets the threshold-loss (Goodfellow et al., 2016). However, such a single threshold strategy does not fit well to an experimental setup that executes independent training of the LSTM at each training/validation/forecasting iteration, since while some user input threshold-loss might be too big for one iteration cycle, it might be too small for the other. Thus, to avoid semi-convergence with respect to epochs, we train two LSTMs, such that while one LSTM is employed to find the best number of epochs at each iteration, the second LSTM is trained only with that many epochs to make a forecast during that iteration.

The two LSTMs intake known lengths, say T time steps, of previous data while setting the loss function to be the mean square error between labels and forecasts. This dual-LSTM predicts only one time step ahead during each of the current iterations that we treat as a training instance during the next iteration. The best number of epochs of the training is an essential hyperparameter that ensures the best training of an LSTM. This best number for epochs is found through walk-forward validation which should be found and set at each iteration in our dual-LSTM to achieve a higher accuracy as the dual-LSTM makes a forecast at each time step. Thus, we utilize one LSTM, say LSTM_1, for finding the best number of epochs at each iteration and the second LSTM, say LSTM_2, is trained only with that many number of epochs to make a forecast for the next time step. Consider that this dual-LSTM has been implemented for forecasting until some arbitrary time step, say $(t-1)$th time step; below, we explicitly state the process for the tth time step. First, we initialize the weights of LSTM_1 with the weights of LSTM_2. Then, we train LSTM_1 with the data sequence for the tth time step through a fixed number of epochs, say L, and find the best number of epochs that contributes the least loss based on a validation step. We train LSTM_2 with exactly that many epochs and forecast for the $(t+1)$th time step. Similarly, we keep on training and predicting this duel-LSTM until the entire time series of interest is processed. Here, both of the LSTMs should have the same architecture, i.e, number of layers, nodes in each layer, etc., and we define an iteration as one training/forecasting cycle.

Since the best number of epochs is revealed at each iteration by the duality, this real-time dual-LSTM is adaptive to forecasting with time series of spiky and abrupt changes such as financial time series. This model is capable of incorporating every new future observation of the time series into the ongoing training process to make forecasting. Since we use a sequence of observed time series to forecast only one time step ahead, the forecasting accuracy increases significantly. Moreover, this dual-LSTM processes $t-2$ previous observations along with the current forecast to make a prediction for the next time step; thus, the forecasting error associated with the current prediction will not propagate to the next prediction as it is processed with $t-2$ previous observations. While classic ways of training LSTMs cause error propagation when the forecasting is performed further away in the testing period, our approach is capable of maintaining a superior accuracy when the forecasting is performed further away.

Our sequentially trained many-to-one dual-LSTM scheme presented in this paper makes the following scientific contributions to the literature of machine learning based financial market forecasting:

- Main contribution
  - This dual-LSTM avoids the state of semi-convergence since the LSTM performing forecasting, i.e., LSTM_2, is only trained with the best number of epochs revealed by LSTM_1 at each iteration. Thus, this framework is adaptive to the natural characteristics of the time series including spikes, abrupt fluctuations, etc., which is pertinent for financial market forecasting.
- Minor contributions
  - This real-time dual-LSTM is capable of incorporating every future observation of the time series, as they are available over time, into the ongoing training process so that the computational cost per iteration remains constant while increasing the accuracy as new data is incorporated over time.
  - Forecasting only one time step ahead for a sequence of data input helps learn the long-term dependencies of the data to the immediate future so that the forecast accuracy is increased significantly.
  - The length of the sequential input can be optimized so that this dual-LSTM apprehends essential patterns of the time series to make precise forecasting.

This paper is structured with five sections, namely, introduction (Section 1), single-LSTMs versus dual-LSTMs (Section 2), state-of-the-art-methods (Section 3), performance analysis (Section 4), and conclusions (Section 5). In Section 2, first, we present the notion of real-time time series forecasting using a single-LSTM, and then, we provide the mathematical formulation of our many-to-one dual-LSTM architecture for sequential training. Section 3 presents two statistical state-of-the-art time series forecasting methods with their formulations: first, a non-linear method called extended Kalman filters (EKF), and then, a linear method called ARIMA. Section 4 provides a detailed analysis of the performance of our dual-LSTM architecture against that of single-LSTM, EKF, and ARIMA using three financial stocks (Apple, Microsoft, Google), three cryptocurrencies (Bitcoin, Ethereum, Cardano), and three commodities (gold, crude oil, natural gas). We present the conclusions along with a discussion in Section 5. Tables 1 and 2 present notations used in the paper along with their descriptions and abbreviations along with their descriptions, respectively.

2. Single-LSTMs versus dual-LSTMs

In this section, first, we provide technical details of real-time time series forecasting using a many-to-one single-LSTM scheme with walk-forward validation. Then, we present the dual-LSTM framework, a predecessor of the single LSTM, that improves real-time time series forecasting by avoiding semi-convergence of the loss with respect to epochs.

Table 1
Notations used in this paper and their descriptions.

Notation	Description
t	Index for time steps
T	Length of the training period
N	Forecasting length
L	Number of epochs
K	Number of stacked LSTMs
$\mathcal{L}_l^{(t)}$	Training loss at the lth epoch of the tth iteration
σ	Sigmoid function in LSTM
p	Order of the AR model
q	Number of past innovations in MA model
$\mathcal{E}$	Relative root mean square error
$\boldsymbol{x}^{(t)}$	The observation at the tth time step where $1 \le t \le T$
$\boldsymbol{X}^{(t)} = [\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(T+t-2)}]$	tth input training window
$\hat{\boldsymbol{x}}^{(t)}$	The forecast at the tth time step where $T < t \le T+N$
$\boldsymbol{w}^{(t)}$	White Gaussian noise vector with zero mean in EKF
$\boldsymbol{y}^{(t)}$	Observation vector at the tth time step in EKF
$\boldsymbol{v}^{(t)}$	White Gaussian noise vector with zero mean in EKF
$\alpha_i, 1 \le i \le p$	Parameters of AR model
$\boldsymbol{\epsilon}^{(t)}$	White Gaussian noise vector with zero mean in AR model
$\beta_i, 1 \le i \le q$	Parameters of MA model
$\boldsymbol{a}^{(t)}$	tth past innovation of MA model
$\boldsymbol{c}$	Bias vector in ARIMA
$\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_c$	Bias vectors in LSTM
f	System dynamics in EKF
h	Measurement function in EKF
L^i	ith level lag operator
$\Delta^D(\cdot)$	Dth differential time series
$Q^{(t)}$, $R^{(t)}$, $P^{(t)}$	Covariance matrices of $\boldsymbol{w}^{(t)}$, $\boldsymbol{v}^{(t)}$, $\boldsymbol{x}^{(t)}$, respectively, in EKF
J_f, J_h	Jacobian matrices of $f(\cdot)$, $h(\cdot)$, respectively, in EKF
W_i, W_f, W_o	Weight matrices in LSTM

Table 2
Abbreviations used in this paper and their descriptions.

Abbreviations	Description
LSTM	Long Short-Term Memory
KF	Kalman Filter
EKF	Extended Kalman filter
AR	AutoRegressive
MA	Moving Average
ARMA	AutoRegressive Moving Average
ARIMA	AutoRegressive Integrated Moving Average

2.1. Many-to-one single-LSTM architecture with walk-forward validation for real-time forecasting

Here, first, we present the many-to-one LSTM architecture; then, we formulate the optimization framework of this LSTM; finally, we present the procedure of real-time training/validation/forecasting of time series using this single-LSTM along with the issues associated with that single-LSTM setup.

2.1.1. Many-to-one LSTM

Since we make forecasts only for one time step ahead at a time for an input time series, the LSTM architecture implemented here is the many-to-one type (Goodfellow et al., 2016). To attain a better prediction precision, we impose the deep learning notion into the ANN by stacking the regular LSTMs, see Fig. 1(a) for an ANN architecture with K stacks of LSTMs. Each stacked LSTM after the first LSTM acts as a hidden layer to the ANN where each LSTM hidden layer provides a series of memory cells to improve the learning accuracy. An LSTM consists of a series of nonlinear recurrent modules, denoted as $M_j^{(t)}$ for $t = 1, \dots, T$ and $j = 1, \dots, N$ in Fig. 1, where each module processes data related to one time step. LSTM introduces a memory cell, a special type of the hidden state, that has the same shape as the hidden state which is engineered to record additional information. Each recurrent module in an LSTM filters information through four hidden layers where three of them are gates, namely, forgotten gate, input gate, and output gate, and

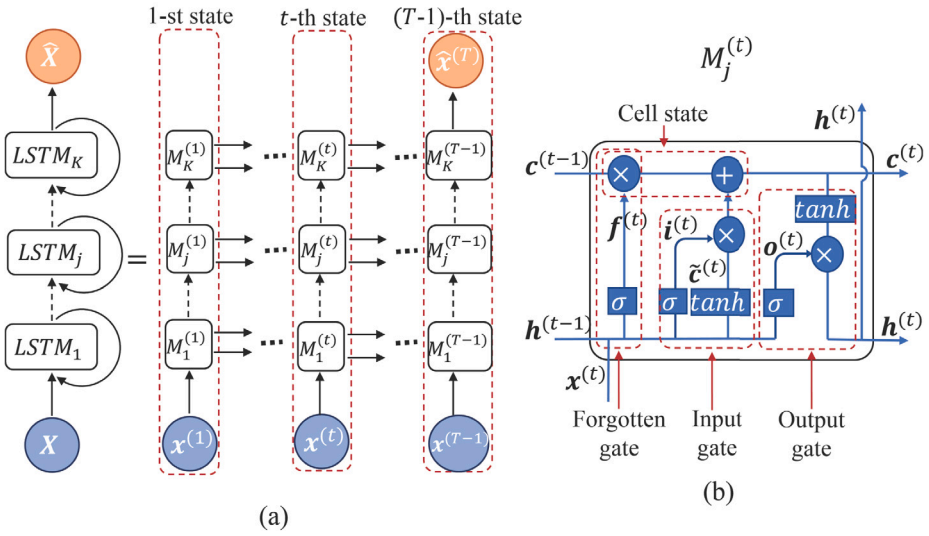

Fig. 1. K-stacked LSTMs for many-to-one forecasting of a single feature time series where each LSTM is a collection of recurrent modules, denoted as M's. (a) The diagram on the left side of the equal sign shows a folded version of the artificial neural network (ANN) whereas the one on the right side of the equal sign shows its unfolded version. Here, the input for the ANN is $\boldsymbol{X} = \left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(T-1)}\right]$ and output from that is $\hat{\boldsymbol{x}}^{(T)}$. (b) Each M filters information through four hidden layers where three of them are gates, namely, forgotten, input, and output, and the other is called the cell state. The forgotten gate resets the content of the memory cell, the input gate decides what new information is stored in the memory cell, the cell state stores long-term information in the memory, and the output gate sends out a filtered version of the memory cell's stored information from the M. Operations in an M are given as follows: $\boldsymbol{f}^{(t)} = \sigma\left(W_f \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_f\right)$, $\boldsymbol{i}^{(t)} = \sigma\left(W_i \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i\right)$, $\tilde{\boldsymbol{c}}^{(t)} = tanh\left(W_C \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_c\right)$, $\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} \oplus \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}$, $\boldsymbol{o}^{(t)} = \sigma\left(W_o \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_o\right)$, and $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot tanh\left(\boldsymbol{c}^{(t)}\right)$, where $\odot$ and $\oplus$ are point-wise multiplication and point-wise addition, respectively.

the other is called the cell state that maintains and updates long-term memory, see Fig. 1(b).

The forgotten gate resets the content of the memory cell by deciding what information should be forgotten or retained. This gate produces a value between zero and one where zero means completely forgetting the previous hidden state and one means completely retaining that. Information from the previous hidden state, i.e., $\boldsymbol{h}^{(t-1)}$, and the information from the current input, i.e., $\boldsymbol{x}^{(t)}$, is passed through the *sigmoid* function, denoted as σ, according to

$$\boldsymbol{f}^{(t)} = \sigma\left(W_f \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_f\right), \tag{1}$$

where W_f and $\boldsymbol{b}_f$ are a weighting matrix and a bias vector, respectively. The input gate consisting of two components decides what new information is to be stored in the cell state. The first component is a *sigmoid* layer that decides which values to be updated based on the previous hidden state and the information from the current input such that

$$\boldsymbol{i}^{(t)} = \sigma\left(W_i \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_i\right), \tag{2}$$

where W_i and $\boldsymbol{b}_i$ are a weighting matrix and a bias vector, respectively. The next component is a *tanh* layer that creates a vector of new candidate values, $\tilde{c}^{(t)}$, based on the previous hidden state and the information from the current input as

$$\tilde{\boldsymbol{c}}^{(t)} = tanh\left(W_c \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_c\right), \tag{3}$$

where W_c and $\boldsymbol{b}_c$ are a weighting matrix and a bias vector, respectively.

Cell state updates the LSTM's memory with new long-term information. For that, first, it multiplies point wisely the old cell state $c^{(t-1)}$ by the forgetting state $\boldsymbol{f}^{(t)}$, i.e., $\boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}$, to assure that the information retained from the old cell state is what is allowed by the forgetting state. Then, we add the pointwise product $\boldsymbol{i} \odot \tilde{c}^{(t)}$ into $\boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}$, i.e.,

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} \oplus \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)}, \tag{4}$$

as the information from the current input state which is found relevant by the ANN. The output gate determines the value of the next hidden state with the information from the current cell state, current input state, and previous hidden state. First, a *sigmoid* layer decides how much of the current input and the previous hidden state are going to output. Then, the current cell state is passed through the *tanh* layer to scale the cell state value between −1 and 1. Thus, the output $\boldsymbol{h}^{(t)}$ is

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot tanh\left(\boldsymbol{c}^{(t)}\right), \text{ with } \boldsymbol{o}^{(t)} = \sigma\left(W_o \cdot [\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}_o\right), \tag{5}$$

where W_o and $\boldsymbol{b}_o$ are weighting matrix and bias vector, respectively. Based upon $\boldsymbol{h}^{(t)}$, the network decides which information from the current hidden state should be carried out to the next hidden state where the next hidden state is used for forecasting. To conclude, the forget gate determines which relevant information from the prior steps is needed. The input gate decides what relevant information can be added from the current cell state, and the output gates finalize the input to the next hidden state.

2.1.2. Optimization of LSTM

Training an LSTM is the process of minimizing a relevant reconstruction error function, also called loss function, with respect to weights and bias vectors of Eqs. (1), (2), (3), (4), and (5). Such a minimization problem is implemented in four steps: first, forward propagation of input data through the ANN to get the output; second, calculate the loss, between forecasted output and the true output; third, calculate the derivatives of the loss function with respect to the LSTM's weights and bias vectors using backpropagation through time (BTT) (Werbos, 1990); and fourth, adjusting the weights and bias vectors by gradient descent method (Gruslys et al., 2016).

BTT unrolls backward all the dependencies of the output onto the weights of the ANN (Manneschi & Vasilaki, 2020), which is represented from the left side to the right side in Fig. 1(a). At each iteration, say $t \in [1, N+1]$, we train the LSTM by only one instance of input-label where the input is $\boldsymbol{X}^{(t)} = \left[\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(T)}, \hat{\boldsymbol{x}}^{(T+1)}, \ldots, \hat{\boldsymbol{x}}^{(T+t-2)}\right]$ and the label is $\hat{\boldsymbol{x}}^{(T+t-1)}$. Due to this process, at the tth iteration, the ANN is trained with tth input-label instance and predicts for the $(T+t-1)$th time step. Thus, we formulate the loss function at the tth iteration of the LSTM as the relative mean square error,

$$\mathcal{L}^{(t)} = \frac{\left\|\hat{\boldsymbol{x}}^{(T+t-1)} - \boldsymbol{x}^{(T+t-1)}\right\|_F^2}{\left\|\boldsymbol{x}^{(T+t-1)}\right\|_F^2}, \tag{6}$$

where F denotes the Frobenius norm and $\hat{\boldsymbol{x}}^{(T+t-1)}$ is the output of the LSTM for the input $\boldsymbol{X}^{(t-1)}$. We use BTT to compute the derivatives of Eq. (6) with respect to the weights and bias vectors. We update the weights using the gradient descent-based method, called Adaptive Moment Estimation (ADAM) (Kingma & Ba, 2015). ADAM is an iterative optimization algorithm used in recent machine learning algorithms to minimize loss functions where it employs the averages of both the first moment and the second moment of the gradient for computations. Specifically, the gradient descent uses the gradient of the objective function to navigate on the search space; whereas, ADAM updates the gradient descent automatically by adaptive step size for each input variable using a decaying average of partial derivatives. It generally converges faster than standard gradient descent methods and saves memory by not accumulating the intermediate weights (Indolia et al., 2018).

2.1.3. Real-time time series forecasting

We adopt a sequential approach to efficiently train LSTMs with time series data and predict the future. For fixed-length sequential data input, one implementation of an LSTM model is to predict only one future time step at an iteration where the process runs until the required length of the forecast is performed. Such a real-time forecasting approach is capable of incorporating every new data point of the time series into the ongoing training process to forecast the next time step. Let, for some $T, N \in \mathbb{N}$ such that $N < T$, the observed portion of the time series is $[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T)}]$ and the unobserved portion, i.e, future, of the time series is $\left[\boldsymbol{x}^{(T+1)}, \dots, \boldsymbol{x}^{(T+N)}\right]$.

For the first iteration, we train this LSTM with the input $\boldsymbol{X}^{(1)} = \left[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T-2)}\right]$ and the label $\boldsymbol{x}^{(T-1)}$ over a fixed number of epochs, say L, which is given by the user. At each epoch, a walk-forward validation step is carried out by computing the loss between the output $\hat{\boldsymbol{x}}^{(T)} = \text{LSTM}\left(\boldsymbol{X}^{(1)}\right)$ and the observed value $\boldsymbol{x}^{(T)}$. We reveal the best number for epochs that involve the least loss during this validation step. This validation step assures better convergence of the loss function since choosing the best number of epochs is challenging for a non-convex or semi-convergent loss function. Now, we reinitialize this LSTM and retrain the LSTM with that many epochs. We pass $\boldsymbol{X}^{(1)}$ through this trained LSTM and forecast the $(T+1)$th time step, i.e., $\hat{\boldsymbol{x}}^{(T+1)} = \text{LSTM}\left(\boldsymbol{X}^{(1)}\right)$, see Fig. 2 for the training/validation/testing procedure of this many-to-one real-time single-LSTM. In the second iteration, we reinitialize and retrain this LSTM with the input $\boldsymbol{X}^{(2)}$ and the label $\boldsymbol{x}^{(T)}$ over the same fixed number of epochs. Similar to Iteration 1, a validation step is carried out by computing the loss, between the current output $\hat{\hat{\boldsymbol{x}}}^{(T+1)} = \text{LSTM}\left(\boldsymbol{X}^{(2)}\right)$ and the current observed value $\hat{\boldsymbol{x}}^{(T+1)}$, at each epoch. Note that, $\hat{\boldsymbol{x}}^{(T+1)}$ was the forecast for the $(T+1)$th time step during Iteration 1 that we used as an observed point for the validation step in Iteration 2. Similar to Iteration 1, we perform the forecasting for the $(T+2)$th time step, i.e., $\hat{\boldsymbol{x}}^{(T+2)} = \text{LSTM}\left(\boldsymbol{X}^{(3)}\right)$ after training the LSTM with the revealed best number of epochs associated with the least validation loss. We keep on doing this process until the forecasting is performed for all the N time steps.

We run this real-time single-LSTM model with the closing prices of the Apple stock to analyze the variation of the validation loss (L_l) with respect to the number of epochs (l). Here, we input a sequence of 1227 days of prices into this single-LSTM and generate the price for the 1228th day where the loss is computed as the relative mean square error between the predicted price and the observed prices for the 1228th day. We proceed with this single-day training for 60 epochs that we show in Fig. 3 which exhibits non-convex characteristics of the validation loss. While there could be multiple reasons for such fluctuations of the loss function with respect to epochs, the main reason is that LSTMs are optimized with a variant of gradient descent techniques, especially, it is *Adam* in our case, which causes oscillations during the descent on the loss function (Goodfellow et al., 2016). The validation step of this single-LSTM is capable of revealing the best number of epochs associated with the least loss. However, this single-LSTM is initialized before each iteration that performs training, validation, and forecasting. Due to the initialization, this single-LSTM is not capable of learning the data outside of the current input data sequence of length $T-2$. To overcome this limitation of the learning process, we came up with a unique way of training a duel-LSTM along with a walk-forward validation step that determines the best number of epochs at each iteration.

2.2. Dual-LSTM for real-time time series forecasting

In this dual approach, we maintain two LSTMs, denoted as LSTM_1 and LSTM_2, that are uniquely trained through each iteration, see Fig. 4 for the architecture of this many-to-one real-time dual-LSTM. For a fixed-length input sequence of data, this real-time duel-LSTM model forecasts only one future time step at each iteration such that the model is capable of incorporating every new data point of the time series into the ongoing training process to make a forecast for the next time step. Similar to the single-LSTM, let, the currently observed time series be $[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T)}]$ for some T and the unobserved future portion of the time series is $\left[\boldsymbol{x}^{(T+1)}, \dots, \boldsymbol{x}^{(T+N)}\right]$ for some $N < T$, see Fig. 4. For the first iteration, we train LSTM_1 with the input $\boldsymbol{X}^{(1)} = \left[\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T-2)}\right]$ and the label $\boldsymbol{x}^{(T-1)}$ over a fixed number of epochs, say L, that is given by the user. At each epoch, a validation step is carried out by computing the loss between the output $\hat{\boldsymbol{x}}^{(T)} = \text{LSTM}_1\left(\boldsymbol{X}^{(2)}\right)$ and the label $\boldsymbol{x}^{(T)}$. During this validation step of LSTM_1, we determine the best number for epochs that involves with the least loss and train LSTM_2 through that many epochs with input $\boldsymbol{X}^{(2)}$ and the label $\boldsymbol{x}^{(T)}$. We pass $\boldsymbol{X}^{(3)}$ though trained LSTM_2 and perform the forecasting for the $(T+1)$th time step, i.e., $\hat{\boldsymbol{x}}^{(T+1)} = \text{LSTM}_2\left(\boldsymbol{X}^{(3)}\right)$. In the second iteration, we initialize LSTM_1 as LSTM_2 which involves assigning LSTM_1 with the same weights and the bias vectors of those in LSTM_2. Now, we train LSTM_1 with the input $\boldsymbol{X}^{(2)}$ and the label $\boldsymbol{x}^{(T)}$ over the same fixed number of epochs. Similar to Iteration 1, at each epoch, a validation step is carried out by computing the loss between the output $\hat{\hat{\boldsymbol{x}}}^{(T+1)} = \text{LSTM}_1\left(\boldsymbol{X}^{(3)}\right)$ and the forecast of the previous iteration, i.e., $\hat{\boldsymbol{x}}^{(T+1)}$. Note that, (1) the forecast in Iteration 1, $\hat{\boldsymbol{x}}^{(T+1)}$, is used as an observed point for the validation step in Iteration 2, and (2) for the $(T+1)$th time step, we used $\hat{\hat{\boldsymbol{x}}}^{(T+1)}$ to denote the output of LSTM_1 in Iteration 2 since $\hat{\boldsymbol{x}}^{(T+1)}$ was already used to denote the output of LSTM_2 in Iteration 1. We perform the forecasting for the $(T+2)$th time step, i.e., $\hat{\boldsymbol{x}}^{(T+2)} = \text{LSTM}_2\left(\boldsymbol{X}^{(4)}\right)$ after training LSTM_2, with the input $\boldsymbol{X}^{(3)}$ and the label $\hat{\boldsymbol{x}}^{(T+1)}$, through the best number of epochs associated with the least validation loss. We keep on doing this process until the forecasting is performed for all the N time steps. Algorithm 1 summarizes the procedure of training, validation, and forecasting of our real-time many-to-one dual-LSTM scheme.

3. State-of-the-art methods

Here, we present two state-of-the-art time series forecast methods, namely, extended Kalman filter (EKF) and autoregressive integrated moving average (ARIMA), that we use to validate the performance of our dual-LSTM scheme. Here, we utilize the same sequential training as we did for LSTMs to produce real-time forecasts on the same financial time series.

3.1. Extended Kalman Filter (EKF)

EKF is a nonlinear version of the standard Kalman filter where the formulation of EKF is based on the linearization of both the state and the observation equations. In an EKF, the state Jacobian and the measurement Jacobian replace the state transition matrix and the measurement matrix, respectively, of a linear KF (Valade et al., 2017). This process essentially linearizes the non-linear function around the current estimate. Linearization enables the propagation of both the state and state covariance in an approximately linear format. Here, the extended Kalman filter is presented in three steps, namely, dynamic process, model forecast step, and data assimilation step.

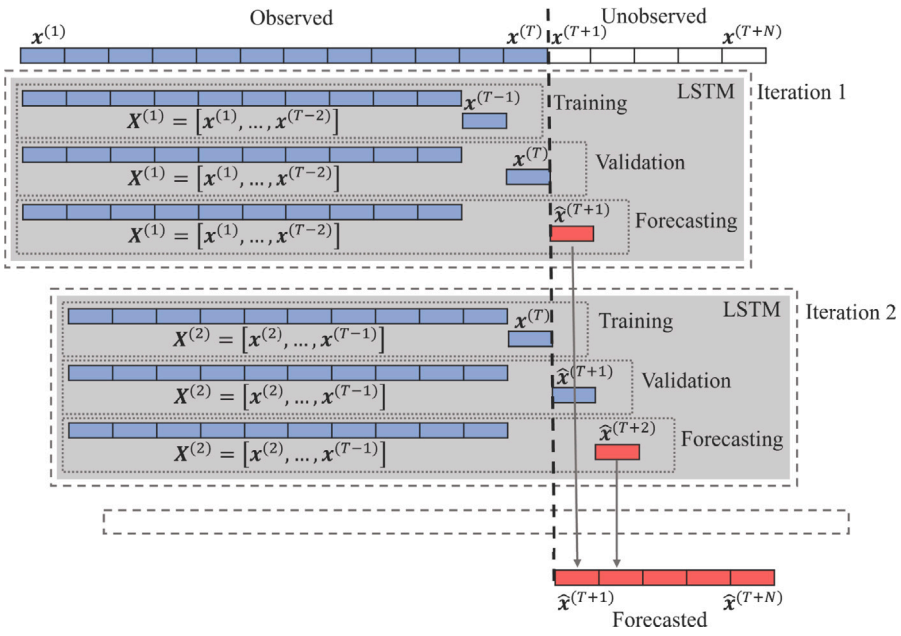

Fig. 2. Real-time time series forecasting scheme using an architecture of a single-LSTM where the observed portion of the time series, $\left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\right]$ for some T, at the current iteration is shown in blue color. The initial unobserved portion of the time series, $\left[\boldsymbol{x}^{(T+1)}, \ldots, \boldsymbol{x}^{(T+N)}\right]$ for some $N < T$, i.e., future, is shown in white color while the forecast for the future at each iteration is shown in red color. For the first iteration, we train this single-LSTM with the input $\boldsymbol{X}^{(1)} = \left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T-2)}\right]$ and the label $\boldsymbol{x}^{(T-1)}$. A walk-forward validation step is performed using the input $\boldsymbol{X}^{(2)} = \left[\boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T-2)}\right]$ and the label $\boldsymbol{x}^{(T)}$ to determine the best number of epochs of the training that gives the least forecasting loss. We reinitialize and retrain this LSTM with that many epochs, and use the trained LSTM to predict for the time step $(T+1)$, which we denoted by $\hat{\boldsymbol{x}}^{(T+1)}$, such that $\hat{\boldsymbol{x}}^{(T+1)} = \text{LSTM}\left(\boldsymbol{X}^{(1)}\right)$. In the second iteration, we retrain the LSTM with the input $\boldsymbol{X}^{(2)} = \left[\boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T-1)}\right]$ and the label $\hat{\boldsymbol{x}}^{(T)}$. The validation set in Iteration 2 includes the first iteration's forecast $\hat{\boldsymbol{x}}^{(T+1)}$ as an observed value. Similar to Iteration 1, we make a new forecast $\hat{\boldsymbol{x}}^{(T+2)}$ based on the retrained LSTM with the best number of epochs. We iteratively perform this process until the forecasting of the entire unobserved portion is available.

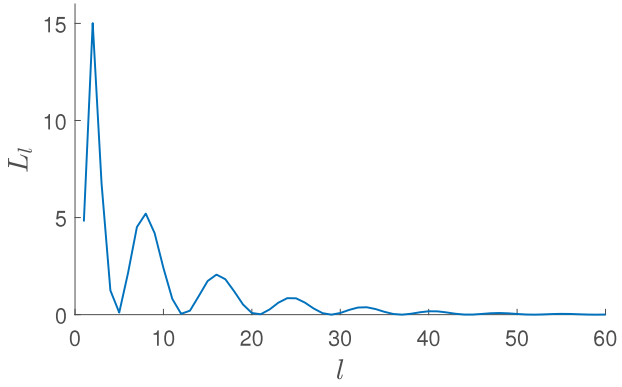

Fig. 3. Non-convex characteristics of the loss function with respect to the number of epochs. We apply our real-time training scheme to train a many-to-one single-LSTM with the closing prices of the Apple stock. We input a sequence of 1227 days of prices into the LSTM and generate the price for the 1228th day. The loss is computed during the validation step as the relative mean square error between the predicted price and the observed prices for the 1228th day. We proceed with this single-day training through 60 epochs and carry out the validation step at each epoch where the validation loss for the lth epoch is denoted as L_l.

Dynamic process. Here, we present both the state model and the observation model of a nonlinear dynamic process. The current state, $\boldsymbol{x}^{(t+1)}$, is modeled as a sum of the nonlinear function of the previous state, $\boldsymbol{x}^{(t)}$, and the noise, $\boldsymbol{w}^{(t)}$, as

$$\boldsymbol{x}^{(t+1)} = f\left(\boldsymbol{x}^{(t)}\right) + \boldsymbol{w}^{(t)}, \tag{7}$$

where $\boldsymbol{x}^{(t)}, \boldsymbol{w}^{(t)} \in \mathbb{R}^n$. Here, the random process $\{\boldsymbol{w}^{(t)}\}$ is Gaussian white noise with zero mean and covariance matrix of $Q^{(t)} = E\left[\boldsymbol{w}^{(t)}\left(\boldsymbol{w}^{(t)}\right)^T\right]$. The initial state $\boldsymbol{x}^{(0)}$ is a random vector with known mean $\boldsymbol{\mu}_0 = E\left(\boldsymbol{x}^{(0)}\right)$ and covariance $P^{(0)} = E\left[(\boldsymbol{x}^{(0)} - \boldsymbol{\mu}_0)(\boldsymbol{x}^{(0)} - \boldsymbol{\mu}_0)^T\right]$. The Jacobian of the predicted state with respect to the previous state, denoted as J_f, is obtained by partial derivatives as $J_f = f_{\boldsymbol{x}}(\cdot)$.

The current observation, $\boldsymbol{y}^{(t+1)}$, is modeled as a sum of the nonlinear function of the current state, $\boldsymbol{x}^{(t+1)}$, and the noise, $\boldsymbol{v}^{(t+1)}$, as

$$\boldsymbol{y}^{(t)} = h\left(\boldsymbol{x}^{(t)}\right) + \boldsymbol{v}^{(t)}, \tag{8}$$

where $\boldsymbol{y}^{(t+1)}, \boldsymbol{v}^{(t+1)} \in \mathbb{R}^n$. Here, the random process $\{\boldsymbol{v}^{(t+1)}\}$ is Gaussian white noise with zero mean and covariance matrix of $R^{(t)} = E\left[\boldsymbol{v}^{(t)}\left(\boldsymbol{v}^{(t)}\right)^T\right]$. The Jacobian of the predicted observation with respect to the previous state, denoted as J_h, is obtained by partial derivatives as $J_h = h_{\boldsymbol{x}}(\cdot)$.

Model forecast step. The state Jacobian and the measurement Jacobian replace linear KF's state transition matrix and the measurement matrix, respectively (Valade et al., 2017). Let, the initial estimates of the state and the covariance are $\boldsymbol{x}^{(0|0)}$ and $P^{(0|0)}$, respectively. The state and the covariance matrices are propagated to the next step using

$$\hat{\boldsymbol{x}}^{(t+1)} = J_f\left(\hat{\boldsymbol{x}}^{(t)}\right)\hat{\boldsymbol{x}}^{(t)} + A\boldsymbol{u}^{(t+1)}, \tag{9}$$

where $\boldsymbol{u}^{(t+1)}$ is the control signal at the time $t+1$ and A is a real valued matrix, and

$$P^{(t+1)} = J_f\left(\hat{\boldsymbol{x}}^{(t)}\right)P^{(t)}\left[J_f\left(\hat{\boldsymbol{x}}^{(t)}\right)\right]^T + Q^{(t)}, \tag{10}$$

respectively.

Data assimilation step. The measurement at the $t+1$ step is given by

$$\boldsymbol{y}^{(t+1)} \approx h\left(\hat{\boldsymbol{x}}^{(t+1)}\right) \tag{11}$$

Use the difference between the actual measurement and the predicted measurement to correct the state at the $t+1$ step. To correct the state, the filter must compute the Kalman gain. First, the filter computes the measurement forecast covariance (innovation) as

$$S^{(t+1)} = J_h\left(\boldsymbol{x}^{(t+1)}\right)P^{(t+1)}\left[J_h\left(\boldsymbol{x}^{(t+1)}\right)\right] + R^{(t+1)} \tag{12}$$

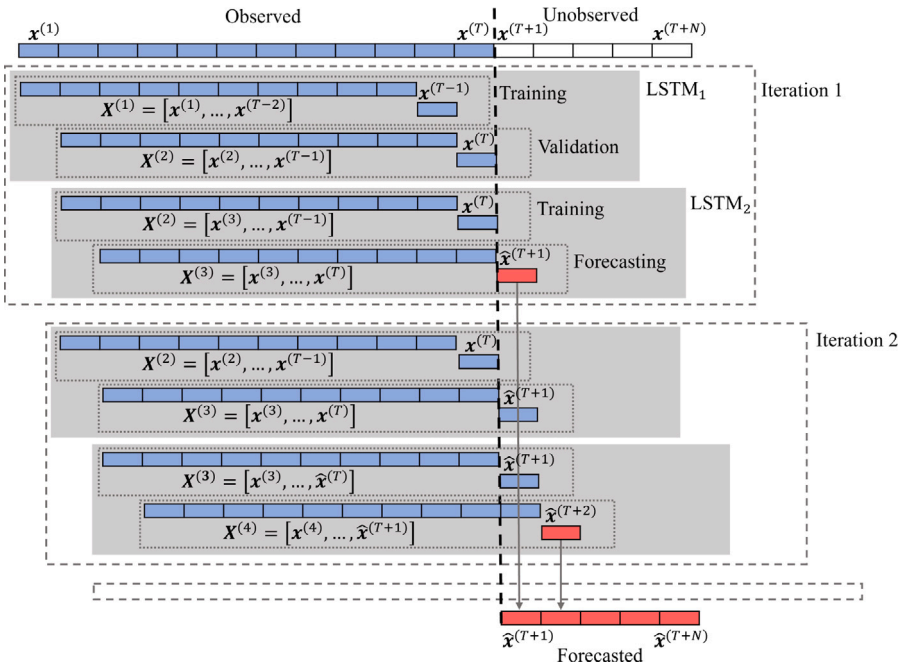

Fig. 4. Real-time time series forecasting scheme of the dual-LSTM where the observed portion of the time series, $\left[\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(T)}\right]$ for some T, at the current iteration is shown in blue color. The initial unobserved portion of the time series, $\left[\boldsymbol{x}^{(T+1)},\ldots,\boldsymbol{x}^{(T+N)}\right]$ for some $N<T$, i.e., future, is shown in white color while the forecast for the future at each iteration is shown in red color. For the first iteration, we train LSTM_1 with the input $\boldsymbol{X}^{(1)}=\left[\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(T-2)}\right]$ and the label $\boldsymbol{x}^{(T-1)}$ over a fixed number of epochs. A walk-forward validation step is performed using the output $\hat{\boldsymbol{x}}^{(T)}=\text{LSTM}_1\left(\boldsymbol{X}^{(2)}\right)$ and the label $\boldsymbol{x}^{(T)}$ to determine the best number of epochs of the training that gives the least forecasting loss. Now, we train LSTM_2 with that best number of epochs and predict for the time step $(T+1)$, i.e., $\hat{\boldsymbol{x}}^{(T+1)}=\text{LSTM}_2(\boldsymbol{X}^{(2)})$. In the second iteration, we initialize the weights and bias vectors of LSTM_1 as those of LSTM_2 and train LSTM_1 with the input $\boldsymbol{X}^{(2)}$ and the label $\boldsymbol{x}^{(T)}$ over the same fixed number of epochs. Similar to Iteration 1, a validation step is carried out by computing the loss between the output $\hat{\boldsymbol{x}}^{(T+1)}=\text{LSTM}_1\left(\boldsymbol{X}^{(3)}\right)$ and the label $\hat{\boldsymbol{x}}^{(T+1)}$ at each epoch. Note that, the validation set in Iteration 2 includes the first iteration's forecast $\hat{\boldsymbol{x}}^{(T+1)}$ as an observed value. Similar to Iteration 1, we perform the forecasting for the $(T+2)$th time step, i.e., $\hat{\boldsymbol{x}}^{(T+2)}=\text{LSTM}_2\left(\boldsymbol{X}^{(4)}\right)$ after training LSTM_2 with the best number of epochs associated with the least validation loss. We keep on doing this process until the forecasting is performed for all the N time steps.

Then, the filter computes the Kalman gain as

$$K_g^{(t+1)}=P^{(t)}\left[J_h\left(\boldsymbol{x}^{(t+1)}\right)\right]^T\left[S^{(t)}\right]^{-1} \tag{13}$$

The filter corrects the predicted estimate by using observation. The estimate, after the correction using the observation $\boldsymbol{y}^{(t+1)}$, is

$$\begin{aligned}\hat{\boldsymbol{x}}^{(t+1)}&=\hat{\boldsymbol{x}}^{(t)}+K^{(t+1)}\left[\boldsymbol{y}^{(t+1)}-h\left(\hat{\boldsymbol{x}}^{(t)}\right)\right]\\ P^{(t+1)}&=\left[I-K_g^{(t+1)}J_h\left(\hat{\boldsymbol{x}}^{(t+1)}\right)\right]P^{(t)}\end{aligned} \tag{14}$$

The corrected state is often called the a posteriori estimate of the state, because it is derived after including the observation.

3.2. Autoregressive integrated moving average (ARIMA) model

ARIMA model is made by combining a differential version of the Autoregression (AR) model into a moving average (MA) model. Many observed time series exhibit serial autocorrelation which is known to be the linear association between lagged observations. The AR model predicts the value for the current time step, $\boldsymbol{x}^{(t)}$, based on a linear relationship between p-recent observations, $\boldsymbol{x}^{(t-1)}$, $\boldsymbol{x}^{(t-2)}$, …, $\boldsymbol{x}^{(t-p)}$, where this p is known as the order of the model (Geurts et al., 1977). Let, $\alpha_1,\ldots,\alpha_p\in\mathbb{R}$ are the coefficients, order p AR model is given by

$$\boldsymbol{x}^{(t)}=\boldsymbol{c}+\alpha_1\boldsymbol{x}^{(t-1)}+\alpha_2\boldsymbol{x}^{(t-2)}+\cdots+\alpha_p\boldsymbol{x}^{(t-p)}+\boldsymbol{\epsilon}^{(t)}, \tag{15}$$

where $\boldsymbol{\epsilon}^{(t)}$ is uncorrelated noise with a zero mean. Let, the lag operator polynomial notation is $L^i\boldsymbol{x}^{(t)}=\boldsymbol{x}^{(t-i)}$. We define order p autoregression lag operator polynomial as $\alpha(L)=(1-\alpha_1L-\cdots-\alpha_pL^p)$. Thus, the AR model is given by

$$\alpha(L)\boldsymbol{x}^{(t)}=c+\boldsymbol{\epsilon}^{(t)}. \tag{16}$$

The solution for the AR model is given by

$$\boldsymbol{x}^{(t)}=\alpha^{-1}(L)\left(c+\boldsymbol{\epsilon}^{(t)}\right). \tag{17}$$

MA model captures serial autocorrelation in a time series $\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(T)}$ by expressing the conditional mean of $\boldsymbol{x}^{(t)}$ as a function of past innovations, $\boldsymbol{a}^{(t)},\boldsymbol{a}^{(t-1)},\ldots,\boldsymbol{a}^{(t-q)}$ (Shumway et al., 2000). An MA model that depends on q past innovations is called an MA model of order q, denoted by MA(q). Let, $\beta_1,\ldots,\beta_q\in\mathbb{R}$ are the coefficients, then the MA(q) model can be represented by the formula

$$\boldsymbol{x}^{(t)}=\boldsymbol{\mu}+\boldsymbol{a}^{(t)}+\beta_1\boldsymbol{a}^{(t-1)}+\cdots+\beta_q\boldsymbol{a}^{(t-q)}, \tag{18}$$

where $\boldsymbol{a}^{(t)}$'s are uncorrelated innovation processes with a zero mean and $\boldsymbol{\mu}$ is the unconditional mean of $\boldsymbol{x}^{(t)}$ for all t.

For some observed time series, a higher-order AR or MA model is needed to capture the underlying process well. In this case, a combined ARMA model can sometimes be a parsimonious choice. An ARMA model expresses the conditional mean of $\boldsymbol{x}^{(t)}$ as a function of both recent observations, $\boldsymbol{x}^{(t-1)},\boldsymbol{x}^{(t-2)},\ldots,\boldsymbol{x}^{(t-p)}$, and recent innovations, $\boldsymbol{a}^{(t)},\boldsymbol{a}^{(t-1)},\ldots,\boldsymbol{a}^{(t-q)}$. The ARMA model with AR degree of p and MA degree of q is denoted by ARMA (p,q), which is given by

$$\begin{aligned}\boldsymbol{x}^{(t)}=\boldsymbol{c}&+\alpha_1\boldsymbol{x}^{(t-1)}+\alpha_2\boldsymbol{x}^{(t-2)}+\cdots+\alpha_p\boldsymbol{x}^{(t-p)}\\&+\beta_0\boldsymbol{a}^{(t)}+\beta_1\boldsymbol{a}^{(t-1)}+\cdots+\beta_q\boldsymbol{a}^{(t-q)},\end{aligned} \tag{19}$$

(Shumway & Stoffer, 2017).

The ARIMA process generates nonstationary series that are integrated of order D where that nonstationary process can be made stationary by taking D differences. A series that can be modeled as a stationary ARMA(p,q) process after being differenced D times is denoted by ARIMA(p,D,q), which is given by

$$\begin{aligned}\Delta^D\boldsymbol{x}^{(t)}=\mu&+\alpha_1\Delta^D\boldsymbol{x}^{(t-1)}+\alpha_2\Delta^D\boldsymbol{x}^{(t-2)}+\cdots+\alpha_p\Delta^D\boldsymbol{x}^{(t-p)}\\&+\boldsymbol{a}^{(t)}-\beta_1\boldsymbol{a}^{(t-1)}-\beta_2\boldsymbol{a}^{(t-2)}-\cdots-\beta_q\boldsymbol{a}^{(t-q)},\end{aligned} \tag{20}$$

where $\Delta^D\boldsymbol{x}^{(t)}$ denotes a Dth differential time series, $\boldsymbol{a}^{(t)}$'s are uncorrelated innovation processes with a zero mean, and μ is the unconditional

Algorithm 1 :Dual-LSTM architecture for real-time time series forecasting.

Denotations: $\boldsymbol{X}^{(1)} = \left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T-2)}\right]$, $\boldsymbol{X}^{(2)} = \left[\boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T-1)}\right]$, and $\boldsymbol{X}^{(t)} = \left[\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(T)}, \hat{\boldsymbol{x}}^{(T+1)}, \ldots, \hat{\boldsymbol{x}}^{(T+t-3)}\right]$ for $3 \leq t \leq N+2$.

Inputs: training portion of the time series $\left(\left[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\right]\right)$; forecasting length ($N$); number of maximum epochs (L).

Outputs: time series forecast $\left(\left[\hat{\boldsymbol{x}}^{(T+1)}, \ldots, \hat{\boldsymbol{x}}^{(T+N)}\right]\right)$; trained $LSTM$ (LSTM_2).

Initialization: two LSTMs, denoted as LSTM_1 and LSTM_2, with the weights $W_f = W_i = W_c = W_o = 0$ and bias vectors $\boldsymbol{b}_f = \boldsymbol{b}_i = \boldsymbol{b}_c = \boldsymbol{b}_o = \boldsymbol{h}^{(0)} = \boldsymbol{0}$.
for $t \in [1, N+2]$ **do**
  for $l \in [1, L]$ **do**

Training:
  Compute $\hat{\boldsymbol{x}}^{(T+t-2)} = \text{LSTM}_1\left(\boldsymbol{X}^{(t)}\right)$ according to the map in Fig. 1(a).
  Minimize $L_l^{(t)} = \left\|\hat{\boldsymbol{x}}^{(T+t-2)} - \boldsymbol{x}^{(T+t-2)}\right\|_F^2 \Big/ \left\|\boldsymbol{x}^{(T+t-2)}\right\|_F^2$ using BTT with respect to the weights W_i, W_i, W_c, and W_o, and the bias vectors $\boldsymbol{b}_f$, $\boldsymbol{b}_i$, $\boldsymbol{b}_c$, and $\boldsymbol{b}_o$ of the composite representation of the functions in Eqns. (1), (2), (3), (4), and (5).
  Update the weights and bias vectors of LSTM_1 using the gradient descent-based method ADAM.

Validation:
  Compute $\hat{\boldsymbol{x}}^{(T+t-1)} = \text{LSTM}_1\left(\boldsymbol{X}^{(t+1)}\right)$.
  Compute $L_l^{(t+1)} = \left\|\hat{\boldsymbol{x}}^{(T+t-1)} - \boldsymbol{x}^{(T+t-1)}\right\|_F^2 \Big/ \left\|\boldsymbol{x}^{(T+t-1)}\right\|_F^2$.
  Record $L_l^{(t+1)}$.
  end for

Forecasting:
  Compute l corresponding to the least the $L_l^{(t+1)}$.
  Train LSTM_2 with input $\boldsymbol{X}^{(t+1)}$ and $\boldsymbol{x}^{(T+t-1)}$ through l many epochs.
  Forecast for $(T+t)$-th time step as $\hat{\boldsymbol{x}}^{(T+t)} = \text{LSTM}_2\left(\boldsymbol{X}^{(t+2)}\right)$.
  Replicate LSTM_2 and initialize LSTM_1 as that replication.
end for
Return $\left(\left[\hat{\boldsymbol{x}}^{(T+1)}, \ldots, \hat{\boldsymbol{x}}^{(T+N)}\right]\right)$ and LSTM_2.

mean of $\boldsymbol{x}^{(t)}$ for all t (Newbold, 1983). With the lag operator $L^i \boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-i)}$, the ARIMA model can be written as

$$\alpha(L)(1-L)^D \boldsymbol{x}^{(t)} = c + \beta(L)\boldsymbol{a}^{(t)} \tag{21}$$

where $\alpha(L) = (1 - \alpha_1 L - \cdots - \alpha_p L^p)$ and $\beta(L) = (1 + \beta_1 L + \cdots + \beta_q L^q)$. Thus, the solution for the ARIMA model is given by

$$\boldsymbol{x}^{(t)} = \left(\alpha(L)(1-L)^D\right)^{-1}\left(c + \beta(L)\boldsymbol{a}^{(t)}\right). \tag{22}$$

4. Performance analysis

The performance analysis of dual-LSTM is conducted using nine financial time series obtained from three markets, namely, stocks, cryptocurrencies, and commodities. We chose Apple, Google, and Microsoft for stocks; Bitcoin, Ethereum, Cardano for cryptocurrencies; and Gold, Oil, and Natural Gas for commodities. These diverse examples validate the broad applicability of the dual-LSTM in analyzing and predicting financial time series.

We follow the procedure given in Fig. 4 to train the real-time many-to-one dual-LSTM architecture given in Fig. 1. Setting the LSTM to run for a specific number of epochs and then using that trained network to make forecasts often do not perform the best training and then do not perform accurate forecasts since the loss function undergoes semi-convergence as shown in Fig. 3. To avoid this issue; first, we train the LSTM for 100 epochs; second, we compute the best number of epochs associated with the least loss; and finally, train again a new LSTM with that many epochs. Moreover, the parameter choices for the training length and forecast length are shown in Table 3. We arbitrarily choose the last 30 days from the time series as the forecasting length and the remainder as the training length; however, we choose arbitrary long forecasting lengths from long time series. The performance of the forecasting method is mostly independent of these two parameter choices except for the case where significantly small training lengths might reduce the thoroughness of the training and then reduce the forecasting performance. However, the training lengths of the time series in this paper are sufficiently long for thorough training.

Table 3
Parameter choices for the training length, forecast length, and number of epochs used in our real-time many-to-one dual-LSTM.

Time series	Training length (T)	Forecast length (N)
Apple	1228	30
Microsoft	1228	30
Google	1228	30
Bitcoin	1064	30
Ethereum	1064	30
Cardano	1064	30
Oil	8248	200
Natural gas	5802	150
Gold	816	30

Now, we incorporate the same one-day recursive forecasting procedure into the other three state-of-the-art methods, namely, single-LSTM, EKF, and ARIMA, and predict the above financial time series. The best parameter values for methods are determined using a validation step performed on a validation set of the latest 10% of the observed portion of the time series where the remaining 90% is utilized as the training set. For that, first, we discretize the parameter space of each method into a set of parameter combinations. Then, for each parameter combination, we recursively fit the time series model of interest with the data $\boldsymbol{X}^{(t)} = \left[\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t+\lfloor 0.9T \rfloor - 1)}\right]$ where $1 \leq t \leq \lfloor 0.1T \rfloor$ and forecast for the $(t + \lfloor 0.9T \rfloor)$th time step, i.e., $\boldsymbol{x}^{(t+\lfloor 0.9T \rfloor)}$. Here, $\lfloor \cdot \rfloor$ is the floor function that rounds down the function value to the nearest integer. This process makes forecasting for the entire validation set but one day of forecasting at a time. Now, we compute the mean of the relative absolute difference between the observed time series and the predicted time series for the validation period using

$$\mathcal{E}_v = \frac{1}{\lfloor 0.1T \rfloor} \sum_{t=\lfloor 0.9T \rfloor}^{T} \frac{\left\|\hat{\boldsymbol{x}}^{(t)} - \boldsymbol{x}^{(t)}\right\|_2}{\left\|\boldsymbol{x}^{(t)}\right\|_2}, \tag{23}$$

as a measure of the validation performance. Finally, we select the parameter combination corresponding to the best validation performance. We implement each of the methods with its best parameter combination to make predictions for the future.

The matrices Q and R in Eqs. (10) and (12) are single-valued since the time series of interest in this study are one-dimensional. Moreover, $\boldsymbol{u}^{(t+1)} = 0$ in Eq. (9) since there is no control signal for these financial time series. We discretize Q and P in the domain $\{0, .2, \ldots, 1\}$ and then run EKF 36 times, each with one combination of values of P and Q, over the training period. We compute the validation error as the relative absolute difference between the observed and forecasted time series using Eq. (23). For each time series, the parameter combination associated with the least validation error is adopted for the subsequent analysis. We found that the best (P, Q)'s for EKF are (.2, .4), (.2, .4), and (.2, .2), for Apple, Microsoft, and Google, respectively; (.2, .4), (.4, .4), and (.2, .2), for Bitcoin, Ethereum, and Cardano, respectively; and (.4, .4), (.2, .6), and (.2, .4), for Oil, Natural gas, and Gold, respectively.

Now, we set EKF with the aforementioned best parameter values and forecast the corresponding time series.

As explained in the previous paragraph, we discretize the parameter space of (p, D, q) as $\{(p, D, q)|p = 0, 2, \ldots, 12; D = 0, 1, 2, 3; q = 0, 1, 2, 3, 4\}$. For each of the nine time series, we fit ARIMA model 140 times over the training period each with one combination of parameters, and forecast the time series for the validation period. We compute the validation error as the relative absolute difference between the observed time series and the forecasted time series using Eq. (23). For each time series, the parameter combination associated with the least validation error is utilized for the subsequent analysis. We found that the best (p, D, q)'s for the ARIMA model are (10, 0, 2), (10, 2, 1), and (0, 1, 1), for Apple, Microsoft, and Google, respectively; (6, 0, 2), (6, 1, 1), and (8, 2, 1), for Bitcoin, Ethereum, and Cardano, respectively; and (4, 1, 1), (10, 1, 2), and (8, 2, 0), for Oil, Natural gas, and Gold, respectively. Now, we set the ARIMA model with the aforementioned best parameter values and forecast the corresponding time series.

We compute the mean of the relative absolute difference between the predicted and the observed time series for the forecast period using

$$\mathcal{E} = \frac{1}{N}\sum_{t=T+1}^{T+N}\frac{\left\|\hat{\boldsymbol{x}}^{(t)} - \boldsymbol{x}^{(t)}\right\|_2}{\left\|\boldsymbol{x}^{(t)}\right\|_2}, \tag{24}$$

as an error measure of the forecast that we show in Table 4. We observe hereby that the order of the best to the worst forecast performance is dual-LSTM, single-LSTM, ARIMA, and EKF. To experimentally justify whether the prediction performance of each method is significantly different from other methods, we utilize the Diebold–Mariano test available in Diebold and Mariano (2002) and Harvey et al. (1997).

Before presenting the Diebold–Mariano test; first, we define some technical terms that are used in the test. Let, $\{\hat{\boldsymbol{x}}_1^{(t)} | \ T < t \leq T + N\}$ and $\{\hat{\boldsymbol{x}}_2^{(t)} | \ T < t \leq T + N\}$ be two forecastings of the true time series $\{\boldsymbol{x}^{(t)} | \ T < t \leq T + N\}$ performed by two time series models; $\{\boldsymbol{r}_1^{(t)} | \ T < t \leq T + N\}$ and $\{\boldsymbol{r}_2^{(t)} | \ T < t \leq T + N\}$ be two residuals of the two forecastings such that $\boldsymbol{r}_1^{(t)} = \hat{\boldsymbol{x}}_1^{(t)} - \boldsymbol{x}_1^{(t)}$ and $\boldsymbol{r}_2^{(t)} = \hat{\boldsymbol{x}}_2^{(t)} - \boldsymbol{x}_2^{(t)}$; and, $d^{(t)} = \|\boldsymbol{r}_1^{(t)}\|_2^2 - \|\boldsymbol{r}_2^{(t)}\|_2^2$. For $1 \leq k < N$, a lag-k autocorrelation function, denoted as γ_k, is defined as

$$\gamma_k = \frac{1}{N}\sum_{t=k+T}^{T+N}\left(d^{(t)} - \bar{d}\right)\left(d^{(t-k)} - \bar{d}\right), \quad \text{where} \quad \bar{d} = \frac{1}{N}\sum_{t=T+1}^{T+N} d^{(t)}. \tag{25}$$

Now, for $h \geq 1$, the test statistics of the Diebold–Mariano test is given by

$$TS = \frac{\bar{d}}{\sqrt{\left(\gamma_0 + 2\sum_{k=1}^{h-1}\gamma_k\right)/N}}, \tag{26}$$

where it is generally sufficient to use $h = N^{1/3} + 1$ (Diebold & Mariano, 2002; Harvey et al., 1997). Under the assumption $\bar{d} = 0$, that is the null hypothesis, TS follows a standard normal distribution, i.e., $TS \sim N(0, 1)$. Since this is a two-tailed test, the null hypothesis is rejected if p-value $< \alpha/2$, where α is the significant level, which concludes that two time series forecastings are significantly different if p-value $< \alpha/2$.

We implement the Diebold–Mariano test for each pair of forecasting produced by five methods (e.g., dual-LSTM, EKF, etc.) applied to nine financial time series (e.g., Apple, Microsoft, etc.). Table 5 shows p-values of the Diebold–Mariano test applied to each pair of time series where each (i, j)th entry of a matrix represents the Diebold–Mariano test's p-value between the forecasts of the ith method and the jth method where $i, j = 1, 2, 3$, and 4 represent dual-LSTM, EKF, ARIMA, and single-LSTM, respectively. The red color represents the p-values that are not smaller than $\alpha/2$ where $\alpha = 0.05$. We observe that the forecastings of dual-LSTM for three cryptocurrencies and three commodities are different than that of the other three methods with a significant level of 0.05. For the three stocks, the forecastings of dual-LSTM are only different than that of EKF with a significant level of 0.05. Diebold–Mariano test's results along with the forecasting error in

Table 4
This table shows forecasting error, quantified as the means of the relative difference between the predicted and the observed time series for the forecasting period, of the five methods dual-LSTM, EKF, ARIMA, and single-LSTM. The analysis is conducted on three stocks Apple, Microsoft, and Google; three cryptocurrencies, Bitcoin, Ethereum, and Cardano; and three commodities, oil, natural gas, and gold. On average, dual-LSTM performs 17 times better than EKF, 4 times better than ARIMA, and 3 times better than single-LSTM. Moreover, the average forecast errors of the dual-LSTM are 0.05, 0.22, and 0.14 for stocks, cryptocurrencies, and commodities, respectively.

Time series	Dual-LSTM	EKF	ARIMA	Single-LSTM
Apple	$5.9 * 10^{-2}$	$4.5 * 10^{-1}$	$7.5 * 10^{-2}$	$6.9 * 10^{-2}$
Microsoft	$5.5 * 10^{-2}$	$4.2 * 10^{-1}$	$6.3 * 10^{-2}$	$6.1 * 10^{-2}$
Google	$3.5 * 10^{-2}$	$4.7 * 10^{-1}$	$5.5 * 10^{-2}$	$4.8 * 10^{-2}$
Bitcoin	$2.1 * 10^{-1}$	$2.7 * 10^{0}$	$3.8 * 10^{-1}$	$3.2 * 10^{-1}$
Ethereum	$1.8 * 10^{-1}$	$2.9 * 10^{0}$	$1.0 * 10^{0}$	$5.9 * 10^{-1}$
Cardano	$2.7 * 10^{-1}$	$4.3 * 10^{0}$	$1.7 * 10^{0}$	$9.2 * 10^{-1}$
Oil	$1.5 * 10^{-1}$	$4.6 * 10^{0}$	$3.0 * 10^{-1}$	$2.2 * 10^{-1}$
Natural gas	$2.0 * 10^{-1}$	$3.4 * 10^{0}$	$1.0 * 10^{0}$	$8.5 * 10^{-1}$
Gold	$7.7 * 10^{-2}$	$2.4 * 10^{0}$	$1.1 * 10^{0}$	$3.3 * 10^{-1}$

Table 5
p-values of the Diebold–Mariano test for each pair of financial time series (e.g., Apple, Microsoft, etc.) forecasting performed by dual-LSTM, EKF, ARIMA, and single-LSTM. Each (i, j)th entry of a matrix represents Diebold–Mariano test's p-value between the ith method and the jth method where $i, j = 1, 2, 3$, and 4 represent dual-LSTM, EKF, ARIMA, and single-LSTM, respectively. The red colors represent the cases where the p-values are not smaller than $\alpha/2$, where α is the significance level of 0.05. That is, if an (i, j)th entry of a matrix is red, the predictions made by the ith and the jth methods are not different with a significant level α of 0.05.

Apple			
	0.00	0.14	0.21
0.00		0.00	0.00
0.14	0.00		0.35
0.21	0.00	0.35	

Microsoft			
	0.00	0.35	0.15
0.00		0.00	0.00
0.35	0.00		0.48
0.15	0.00	0.48	

Google			
	0.00	0.07	0.18
0.00		0.00	0.00
0.07	0.00		0.28
0.18	0.00	0.28	

Bitcoin			
	0.00	0.02	0.02
0.00		0.00	0.00
0.02	0.00		0.27
0.02	0.00	0.27	

Ethereum			
	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00		0.00
0.00	0.00	0.00	

Cardano			
	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00		0.00
0.00	0.00	0.00	

Oil			
	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00		0.00
0.00	0.00	0.00	

Natural gas			
	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00		0.00
0.00	0.00	0.00	

Gold			
	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00		0.00
0.00	0.00	0.00	

Table 4 justify that the dual-LSTM framework outperforms the other three methods.

Fig. 5 shows the forecasted prices of the three stocks, Apple (first row), Microsoft (second row), and Google (third row), using our sequentially trained many-to-one dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). Here, Fig. 5(a), (d), and (g) show the original time series where the training period is on the left side of the black discrete vertical line and the forecasting period is on the right side of that line. Fig. 5(b), (e), and (h) show the original time series along with the forecasts. Since some of the forecasts closely mimic the observed time series to overlap, we compute the absolute difference between the observations and the forecasts, see Fig. 5(c), (f), and (i). We observe that all four methods are capable of capturing the pattern of the time series with the order of the best to the worst forecast performance is dual-LSTM, single-LSTM, ARIMA, and EKF. Moreover, while dual-LSTM, single-LSTM, and ARIMA perform similarly well, EKF performs weakly.

Fig. 6 shows the forecasted prices of the three cryptocurrencies, Bitcoin (first row), Ethereum (second row), and Cardano (third row), using dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). Here, the first column of Fig. 6 shows the original time series where the training period is on the left side of the black discrete vertical line and the forecasting period is on the right side of this line. The second column of Fig. 6 presents the original time series with

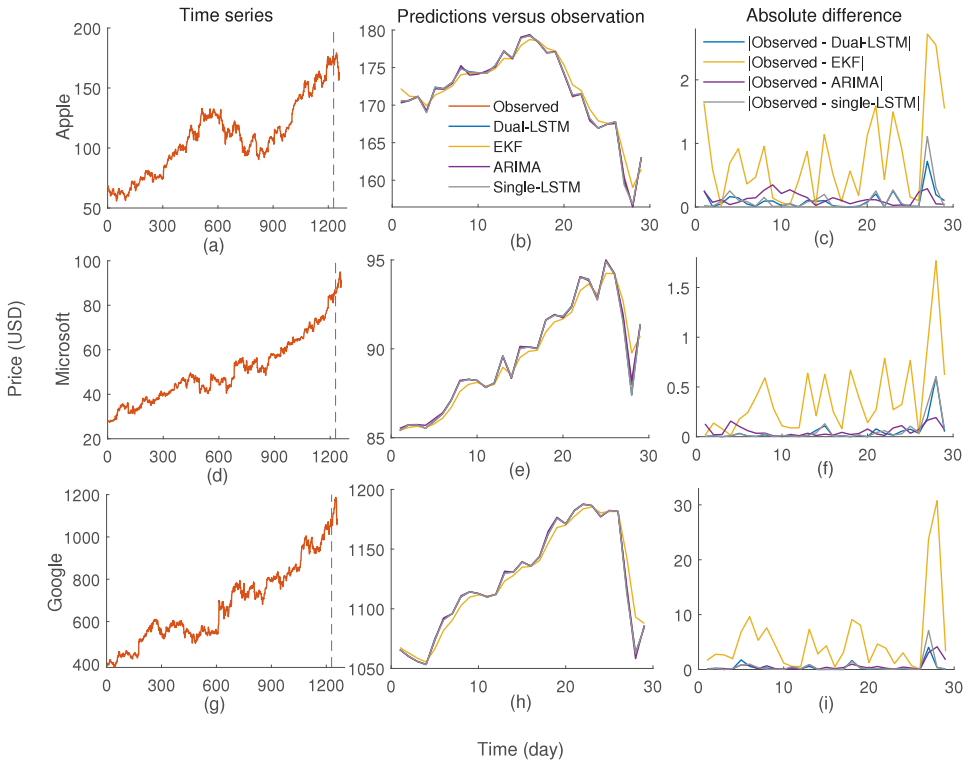

Fig. 5. Prices of the three stocks, Apple (first row), Microsoft (second row), and Google (third row), forecasted using our sequentially trained many-to-one dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). The first column shows the entire time series, the second column shows the observed and predicted time series for the forecast period, and the third column shows the absolute difference between the observed and predicted time series. We observe that the order of the best to the worst performance is dual-LSTM, single-LSTM, ARIMA, and EKF. Note that, some of the time series are not visible as they are covered by others.

the forecasts. Since some of the forecasts closely mimic the observed time series to overlap, we compute the absolute difference between the observations and the forecasts, see the third column of Fig. 6. We observe that dual-LSTM, single-LSTM, and ARIMA are capable of capturing the pattern of the time series in contrast to the weak forecast of EKF. The order of the best to the worst forecast performance is dual-LSTM, single-LSTM, ARIMA, and EKF.

Fig. 7 shows the prices of the three commodities, Oil (first row), Natural gas (second row), and Gold (third row), forecasted using dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). The first column of Fig. 7 shows the original time series where the training period is on the left side of the black discrete vertical line and the forecasting period is on the right side of this line. The second column of Fig. 7 presents the original time series with the predictions for the testing period. We compute the absolute difference between the observations and the forecasts, see the third column of Fig. 7, since some of the forecasts are similar to observations. We observe that mostly dual-LSTM, single-LSTM, and ARIMA are capable of capturing the pattern of the time series. The order of the best to the worst forecast performance is dual-LSTM, single-LSTM, ARIMA, and EKF.

The performance of this real-time many-to-one dual-LSTM is highly influenced by the number of epochs that it is executed. To check this assertion, we compute the forecast performance of all four methods with respect to different numbers of epochs. This analysis is performed on the prices of Apple, Bitcoin, and Gold, which we show in the first column, the second column, and the third column, respectively, in Fig. 8. In Fig. 8, the methods dual-LSTM, EKL, ARMA, and single-LSTM are represented in blue, yellow, purple, and gray, respectively. According to Eq. (24), the forecast performance is computed as the mean of the relative absolute difference, i.e. $\mathcal{E}$, between the forecast and the observed time series. Since EKF and ARIMA are independent of epochs, we represent their $\mathcal{E}$ as a straight line. We observe that the performance of the dual-LSTM improves from the worst to the best when the number of epochs is increased.

5. Conclusions

The classical methods of solving temporal chaotic systems are either both linear and model-based which the model assumes linear relationships between systems' previous outputs for stationary time series or both nonlinear and model based which the model assumes nonlinearity in such relationships. Thus, while the classic linear models have no capability of capturing non-linear and non stationary relationships in the data, nonlinear models might not cope with certain nonlinear and non-stationary signals as the underlying model does not fit with the data. Because the financial time series are often nonstationary, nonlinear, and contain noise (Bontempi et al., 2013), traditional statistical model-based techniques encounter some limitations in forecasting time series with high precision. However, ANN-based approaches such as LSTMs are not only nonlinear but also model free so that a neural network learns to mimic the data as permitted during the training process. In this paper, we have presented a real-time forecasting technique for financial markets using a sequentially trained many-to-one dual-LSTM. We applied this technique for some time series obtained from stock market, cryptocurrency market, and commodity market; then, compared the forecasting performance of the dual-LSTM against three state-of-the-art methods, namely, EKF, ARIMA, and single-LSTM.

Here, we train a many-to-one dual-LSTM with sequential data sampled using a moving window approach such that the succeeding window is shifted forward by one data instance from the preceding window. Such sequential window training plays an important role in time series forecasts since it, (1) helps generate more data from a given limited time series and then thorough training of the ANN; (2) makes the data heterogeneous so that the overfitting issue of the ANN can be reduced; and (3) facilitates the learning of patterns of the data not only for the entire time series but also for short segments of sequential data. Sequential window training maximizes the performance of this dual-LSTM as it accelerates LSTMs' learning capability as well as it increases the LSTMs' robustness to new data.

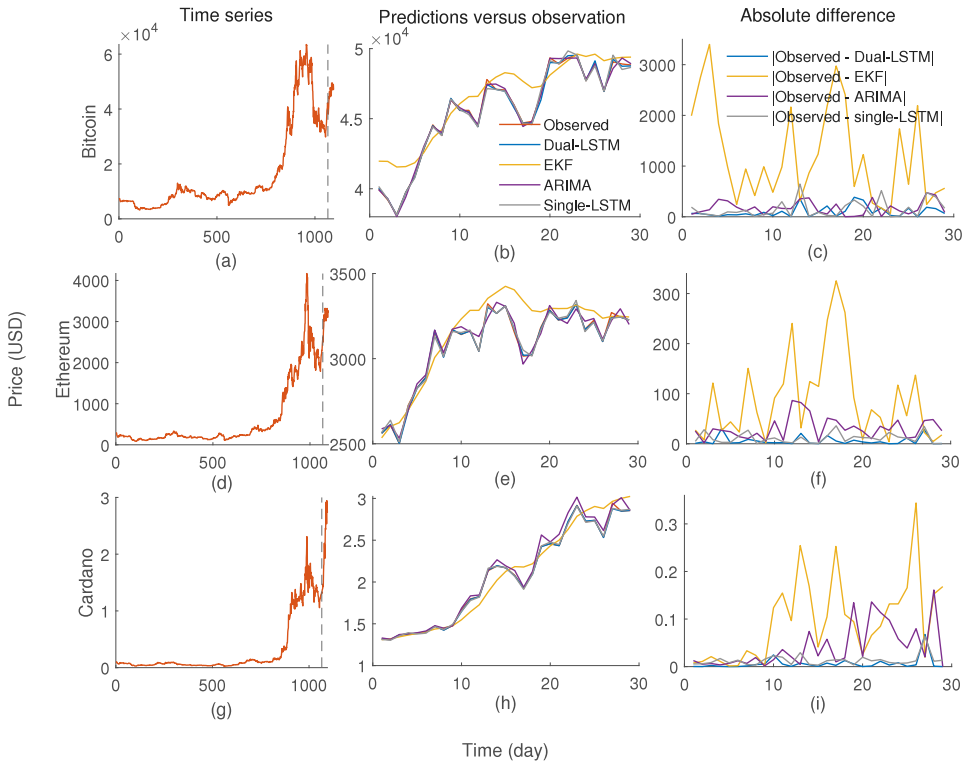

Fig. 6. Forecasts of the prices of three cryptocurrencies, Bitcoin (first row), Ethereum (second row), and Cardano (third row), performed by the sequentially trained many-to-one dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). The first, second, and third columns show the entire time series, observed and predicted time series for the forecast period, and the absolute difference between the observed and predicted time series, respectively. We observe that dual-LSTM, single-LSTM, ARIMA, and EKF is the order of the best to the worst performance. Note that, some of the time series are not visible as they are covered by others.

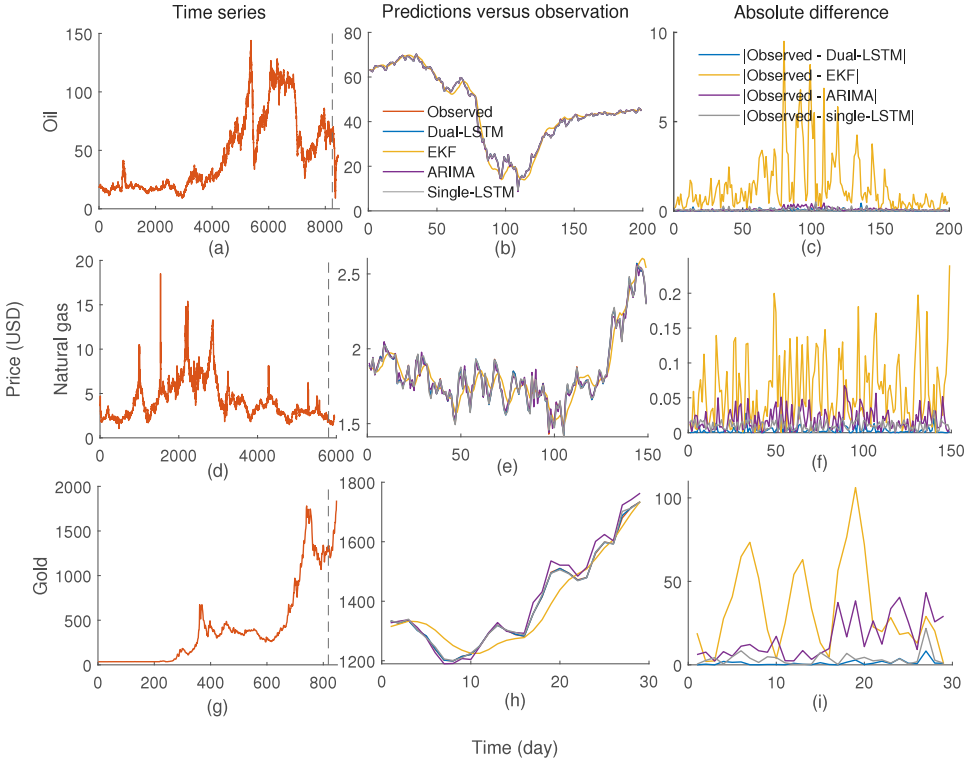

Fig. 7. Three commodities, Oil (first row), Natural gas (second row), and Gold (third row) are forecasted using our sequentially trained many-to-one dual-LSTM (blue), EKF (yellow), ARIMA (purple), and single-LSTM (gray). The first, second, and third columns show the entire time series, observed and predicted time series for forecast period, and the absolute difference between the observed and predicted time series, respectively. The order of the best to the worst performance is dual-LSTM, single-LSTM, ARIMA, and EKF. Note that, some of the time series are not visible as they are covered by others.

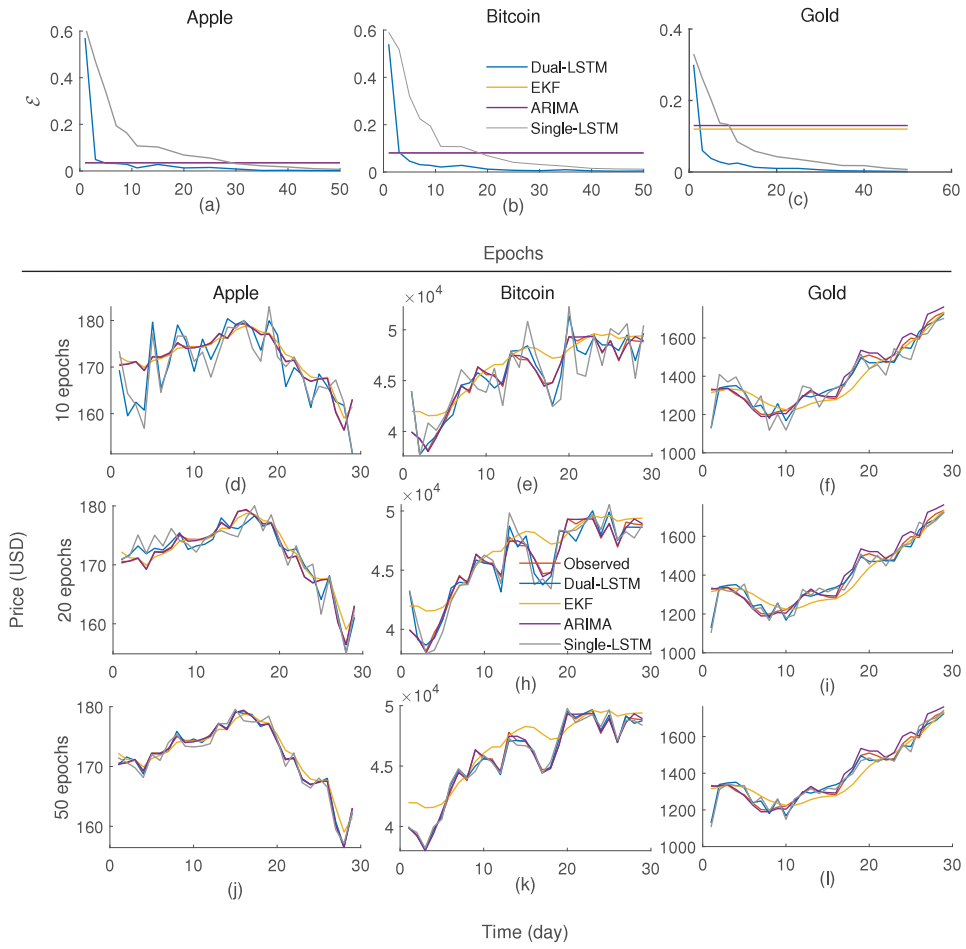

Fig. 8. Forecast performance of the real-time many-to-one dual-LSTM (blue) with respect to the different numbers of epochs and its comparison with EKF (yellow), ARIMA (purple), and single-LSTM (gray). The first row shows the mean of the relative absolute difference, denoted as $\mathcal{E}$, between the forecast and the observed time series for Apple, Bitcoin, and Gold. Note that, EKF and ARIMA are independent of epochs; however, we represent a straight line for their $\mathcal{E}$ in the first row. The second, third, and fourth rows show the forecast and the observed time series of the prices in the forecast periods of Apple, Bitcoin, and Gold for 10 epochs, 20 epochs, and 50 epochs, respectively. We observe that the performance of the dual-LSTM improves from worst to the best when the number of epochs is increased. Note that some of the time series and plots are not visible as they are covered by others.

The literature proposes some techniques based on Partial Autocorrelation Function (PAF), Information Criteria (IC), and validation, to determine the lag parameters p and the smoothing parameter q in ARIMA (Hastie et al., 2009). In the PAF approach, the best p and q are chosen based on some threshold value applied to the PAF. Such approaches are capable of determining the aforesaid parameters based on a threshold-specified significant partial autocorrelation; however, there might be some other parameter values that attain better autocorrelation even though they are not significant based on the threshold (Hastie et al., 2009). IC includes two famous techniques, namely, Bayes Information Criterion (BIC) and Akaike Information Criterion (AIC), each is an estimation of the lag length p. The basic idea of both criteria is that the first term dealing with the square sum of residues decreases as lags are added whereas the second term weighting the lags increases as lags are added. However, as BIC and AIC are model-based frameworks, not only they rely on assumptions such as the data follows a distribution, independence of the observations, and no uncertainty in the data, but also they need much more observations than model parameters (Hastie et al., 2009). Validation is proven to be free from the aforesaid issues pertaining to the other techniques except the issue of small sample size (Varoquaux, 2018). However, all our nine time series datasets had a sufficient amount of data. Thus, to offer a competitive comparison in this study, each of the three state-of-the-art methods EKF and ARIMA was implemented with the best combination of parameters that were determined by a validation scheme. Therein, we separated a validation set of 10% of the recent data from the training set and then trialed diverse parameter combinations in the validation set to determine the combination that gives the best forecasting performance. This validation approach provided a remarkable performance of these forecasting frameworks as we do not determine the best parameter values based on a user input threshold.

The performance analysis of this study covers the dual-LSTM applied to nine time series obtained from three financial markets, stocks (Apple, Microsoft, Google), cryptocurrencies (Bitcoin, Ethereum, Cardano), and commodities (gold, crude oil, natural gas). We observed that the dual-LSTM performs exceptionally better than the other three methods for all the nine datasets where the performance of EKF was significantly weak. We have seen in Table 4, on average, dual-LSTM performs 17 times better than EKF, 4 times better than ARIMA, and 3 times better than single-LSTM. The average forecast errors of the dual-LSTM are 0.05, 0.22, and 0.14 for stocks, cryptocurrencies, and commodities, respectively. The reason for that is while the forecast on less volatile time series like in the stock market is easy, the forecast on high volatile time series like in the cryptocurrency market is challenging.

In future work, we are planning to extend this sequentially trained many-to-one dual-LSTM to employ as a real-time fault detection technique in industrial production processes. This real-time fault detection scheme will be capable of producing an early alarm to alert a shift in the production process so the quality controlling team can take necessary actions. Moreover, trajectories of collectively moving agents can

be represented on a low-dimensional manifold that underlies a high-dimensional data cloud (Gajamannage et al., 2015; Gajamannage & Paffenroth, 2021; Gajamannage et al., 2019). However, some segments of these trajectories are not tracked by multi-object tracking methods due to natural phenomena such as occlusions. Thus, we are planning to utilize our dual-LSTM architecture to make forecasts for the fragmented segments of the trajectories. As another future direction, we are planning to utilize this dual-LSTM architecture for streamflow forecasting. Streamflow modeling and forecasting is a sophisticated task in the field of hydrology due to its high volatility and frequent spikes (Jayathilake & Smith, 2021). We employed RNNs in Gajamannage et al. (2023), to model and forecast streamflow with a sliding window approach. The sliding window approach is capable of generating a big amount of homogeneous data from a given time series. However, we will utilize this dual-LSTM to understand hydrologic catchments' behaviors and then produce their daily expected streamflow throughout the year.

We empirically validated that our real-time dual-LSTM outperforms the performance of EKF and ARIMA. In the future, we are planning to compare the performance of our real-time dual-LSTM with that of the other famous ANN-based methods such as Facebook developed Prophet (Taylor & Letham, 2018), Amazon developed DeepAR (Salinas et al., 2020), Google developed Temporal Fusion Transformer (Lim et al., 2021), and Element AI developed N-BEATS (Oreshkin et al., 2019). Prophet was designed for automatic forecasting of univariate time series data. DeepAR is a probabilistic forecasting model based on recurrent neural networks. Temporal Fusion Transformer is a novel attention-based architecture that combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics. N-BEATS is a custom deep learning algorithm that is based on backward and forward residual links for univariate time series point forecasting. Moreover, since our dual-LSTM forecasts stock prices in this paper rather than returns, we are planning to analyze the forecasting performance of our framework on returns. While stock prices are non-stationary, returns are stationary; thus, this future analysis would be helpful for understanding the wide applicability of our dual-LSTM framework.

We presented both nonlinear and real-time forecast technique for financial time series that is made by a many-to-one dual-LSTM which is sequentially trained with windows of data. The sequential window training approach has significantly improved LSTM's learning ability while dramatically reducing LSTM's over-fitting issues. We empirically justified that our dual-LSTM possesses superior performance even for highly volatile time series such as those in cryptocurrencies and commodities.

CRediT authorship contribution statement

Kelum Gajamannage: Conceptualization, Methodology, Validation, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition. **Yonggi Park:** Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Dilhani I. Jayathilake:** Software, Validation, Formal analysis, Investigation, Data curation, Writing – review & editing, Visualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Kelum Gajamannage reports financial support and equipment, drugs, or supplies were provided by Google Cloud Platform.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to thank the Google Cloud Platform for granting Research Credit to access its GPU computing resources under project number 397744870419.

References

Allen-Zhu, Z., Li, Y., & Song, Z. (2019). On the convergence rate of training recurrent neural networks. In *Advances in neural information processing systems, Vol. 32* (pp. 1310–1318). PMLR, arXiv:1810.12065.

Baek, Y., & Kim, H. Y. (2018). Modaugnet: A new forecasting framework for stock market index value with an overfitting prevention lstm module and a prediction lstm module. *Expert Systems with Applications*, *113*, 457–480.

Bahadur, N., Paffenroth, R., & Gajamannage, K. (2019). Dimension estimation of equity markets. In *Proceedings - 2019 IEEE international conference on big data, big data 2019* (pp. 5491–5498). Institute of Electrical and Electronics Engineers Inc, http://dx.doi.org/10.1109/BigData47090.2019.9006343.

Bhandari, H. N., Rimal, B., Pokhrel, N. R., Rimal, R., Dahal, K. R., & Khatri, R. K. (2022). Predicting stock market index using lstm. *Machine Learning with Applications*, *9*, Article 100320.

Bontempi, G., Ben Taieb, S., & Le Borgne, Y. A. (2013). Machine learning strategies for time series forecasting. In *LNBIP*, *Lecture notes in business information processing, Vol. 138* (pp. 62–77). Springer, http://dx.doi.org/10.1007/978-3-642-36318-4_3.

Broersen, P. M. (2006). *Automatic autocorrelation and spectral analysis*. Springer Science & Business Media.

Diebold, F. X., & Mariano, R. S. (2002). Comparing predictive accuracy. *Journal of Business & Economic Statistics*, *20*, 134–144.

Fang, Z., Ma, X., Pan, H., Yang, G., & Arce, G. R. (2023). Movement forecasting of financial time series based on adaptive lstm-bn network. *Expert Systems with Applications*, *213*, Article 119207.

Gajamannage, K., Butail, S., Porfiri, M., & Bollt, E. M. (2015). Identifying manifolds underlying group motion in vicsek agents. *European Physical Journal: Special Topics*, *224*, 3245–3256. http://dx.doi.org/10.1140/epjst/e2015-50088-2.

Gajamannage, K., Jayathilake, D. I., Park, Y., & Bollt, E. M. (2023). Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *33*(1), 013109. http://dx.doi.org/10.1063/5.0088748.

Gajamannage, K., & Paffenroth, R. (2021). Bounded manifold completion. *Pattern Recognition*, *111*, Article 107661. http://dx.doi.org/10.1016/j.patcog.2020.107661.

Gajamannage, K., Paffenroth, R., & Bollt, E. M. (2019). A nonlinear dimensionality reduction framework using smooth geodesics. *Pattern Recognition*, *87*, 226–236. http://dx.doi.org/10.1016/j.patcog.2018.10.020.

Gajamannage, K., Park, Y., Paffenroth, R., & Jayasumana, A. P. (2022). Reconstruction of fragmented trajectories of collective motion using Hadamard deep autoencoders. *Pattern Recognition*, *131*, 108891. http://dx.doi.org/10.1016/j.patcog.2022.108891.

Gao, T., Chai, Y., & Liu, Y. (2017). Applying long short term momory neural networks for predicting stock closing price. In *2017 8th IEEE international conference on software engineering and service science* (pp. 575–578). IEEE.

Geurts, M., Box, G. E. P., & Jenkins, G. M. (1977). Time series analysis: Forecasting and control. *Journal of Marketing Research*, *14*, 269. http://dx.doi.org/10.2307/3150485.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., & Graves, A. (2016). Memory-efficient backpropagation through time. In *Advances in neural information processing systems, Vol. 29* (pp. 4132–4140). arXiv:1606.03401.

Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, *33*, 2223–2273.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 357–384.

Harvey, D., Leybourne, S., & Newbold, P. (1997). Testing the equality of prediction mean squared errors. *International Journal of Forecasting*, *13*, 281–291.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction, Vol. 2*. Springer.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Indolia, S., Goswami, A. K., Mishra, S. P., & Asopa, P. (2018). Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia Computer Science*, *132*, 679–688.

Jayathilake, D. I., & Smith, T. (2021). Assessing the impact of pet estimation methods on hydrologic model performance. *Hydrology Research*, *52*, 373–388.

Kim, K.-j., & Ahn, H. (2012). Simultaneous optimization of artificial neural networks for financial forecasting. *Applied Intelligence*, *36*, 887–898.

Kim, J., El Khamy, M., & Lee, J. (2017). Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. In *Proceedings of the annual conference of the international speech communication association, Vol. 2017-Augus* (pp. 1591–1595). http://dx.doi.org/10.21437/Interspeech.2017-477.

Kim, H. Y., & Won, C. H. (2018). Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models. *Expert Systems with Applications*, *103*, 25–37.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd international conference on learning representations, ICLR 2015 - conference track proceedings*. http://dx.doi.org/10.48550/arXiv.1412.6980, arXiv:1412.6980.

Le, P., & Zuidema, W. (2016). Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs. (pp. 87–93). http://dx.doi.org/10.18653/v1/w16-1610, arXiv preprint arXiv:1603.00423.

Lim, B., Aräık, S., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*, 1748–1764. http://dx.doi.org/10.1016/j.ijforecast.2021.03.012, arXiv:1912.09363.

Liu, S., Liao, G., & Ding, Y. (2018). Stock transaction prediction modeling and analysis based on lstm. In *2018 13th IEEE conference on industrial electronics and applications* (pp. 2787–2790). IEEE.

Ma, Y., & Principe, J. (2018). Comparison of static neural network with external memory and RNNs for deterministic context free language learning. In *Proceedings of the international joint conference on neural networks, Vol. 2018-July* (pp. 1–7). IEEE, http://dx.doi.org/10.1109/IJCNN.2018.8489240.

Manneschi, L., & Vasilaki, E. (2020). An alternative to backpropagation through time. *Nature Machine Intelligence*, *2*, 155–156. http://dx.doi.org/10.1038/s42256-020-0162-9.

Newbold, P. (1983). ARIMA model building and the time series analysis approach to forecasting. *Journal of Forecasting*, *2*, 23–35. http://dx.doi.org/10.1002/for.3980020104.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437. URL: http://arxiv.org/abs/1905.10437.

Qiu, J., Wang, B., & Zhou, C. (2020). Forecasting stock prices with long-short term memory neural network based on attention mechanism. *PLoS One*, *15*, Article e0227222. http://dx.doi.org/10.1371/journal.pone.0227222.

Rather, A. M., Agarwal, A., & Sastry, V. (2015). Recurrent neural network and a hybrid model for prediction of stock returns. *Expert Systems with Applications*, *42*, 3234–3241.

Roondiwala, M., Patel, H., & Varma, S. (2017). Predicting stock prices using lstm. *International Journal of Science and Research (IJSR)*, *6*, 1754–1756.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*, 1181–1191. http://dx.doi.org/10.1016/j.ijforecast.2019.07.001, arXiv:1704.04110.

Shih, C. H., Yan, B. C., Liu, S. H., & Chen, B. (2018). Investigating siamese LSTM networks for text categorization. In *Proceedings - 9th Asia-Pacific signal and information processing association annual summit and conference, APSIPA ASC 2017, Vol. 2018-Febru* (pp. 641–646). IEEE, http://dx.doi.org/10.1109/APSIPA.2017.8282104.

Shumway, R. H., & Stoffer, D. S. (2017). *ARIMA models* (pp. 75–163). Cham: Springer, http://dx.doi.org/10.1007/978-3-319-52452-8_3.

Shumway, R. H., Stoffer, D. S., & Stoffer, D. S. (2000). *Time series analysis and its applications, Vol. 3*. Springer.

Simistira, F., Ul-Hassan, A., Papavassiliou, V., Gatos, B., Katsouros, V., & Liwicki, M. (2015). Recognition of historical greek polytonic scripts using LSTM networks. In *Proceedings of the international conference on document analysis and recognition, ICDAR, Vol. 2015-Novem* (pp. 766–770). IEEE, http://dx.doi.org/10.1109/ICDAR.2015.7333865.

Squarepants, S. (2022). Bitcoin: A peer-to-peer electronic cash system. *SSRN Electronic Journal*, 21260. http://dx.doi.org/10.2139/ssrn.3977007.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *American Statistician*, *72*, 37–45. http://dx.doi.org/10.1080/00031305.2017.1380080.

Terasvirta, T., & Anderson, H. M. (1992). Characterizing nonlinearities in business cycles using smooth transition autoregressive models. *Journal of Applied Econometrics*, *7*, S119–S136.

Tian, X., Zhang, J., Ma, Z., He, Y., Wei, J., Wu, P., Situ, W., Li, S., & Zhang, Y. (2017). Deep LSTM for large vocabulary continuous speech recognition. http://dx.doi.org/10.48550/arXiv.1703.07090, arXiv:1703.07090.

Tsay, R. S. (1989). Testing and modeling threshold autoregressive processes. *Journal of the American Statistical Association*, *84*, 231–240.

Valade, A., Acco, P., Grabolosa, P., & Fourniols, J. Y. (2017). A study about kalman filters applied to embedded sensors. *Sensors (Switzerland)*, *17*, 2810. http://dx.doi.org/10.3390/s17122810.

Varoquaux, G. (2018). Cross-validation failure: Small sample sizes lead to large error bars. *Neuroimage*, *180*, 68–77.

Wang, J.-J., Wang, J.-Z., Zhang, Z.-G., & Guo, S.-P. (2012). Stock index forecasting based on a hybrid model. *Omega*, *40*, 758–766.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*, 1550–1560. http://dx.doi.org/10.1109/5.58337.