Available online at www.isas.org.in/jisas
Journal of the Indian Society of Agricultural Statistics 77(1) 2023 71–78

LSTM based Stacked Autoencoder Approach for Time Series Forecasting

K.N. Singh, Kamal Sharma, G. Avinash, Rajeev Ranjan Kumar, Mrinmoy Ray, Ramasubramanian V., Achal Lama and S.B. Lal

ICAR-Indian Agricultural Statistics Research Institute, New Delhi

Received 16 March 2023; Revised 03 April 2023; Accepted 20 April 2023

SUMMARY

This study proposes a novel approach for multi-step time series forecasting using a stacked long-short term memory (LSTM) sequence-to-sequence autoencoder (LSTM-SAE) to handle the volatility of edible oil prices in the Indian market. The approach was implemented on Ruchi Soya Ltd. stock price dataset and compared with other deep learning models like Gated Recurrent Unit (GRU), LSTM, and Bi-directional LSTM. The LSTM-SAE outperformed other models in closing price prediction based on evaluation metrics like Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). The proposed approach has significant implications for stakeholders in the edible oil and oilseeds industry, including farmers, traders, and policymakers.

Keywords: Deep learning, LSTM, Autoencoder, Gated Recurrent Unit (GRU), Stock price.

1. INTRODUCTION

Edible oil price forecasting is a critical activity that has significant impacts on both the producers and consumers of edible oils. The price of edible oils is subject to various factors, including supply and demand, weather conditions, geopolitical events, and commodity trading. Accurately forecasting future prices can help producers and traders to make better-informed decisions about when to buy and sell, thus mitigating risks and maximising profits. At the same time, consumers can benefit from accurate price forecasting, as they can make informed purchasing decisions and avoid sudden price increases. Governments also rely on accurate price forecasts to manage food security, stabilise prices, and ensure adequate supplies of edible oils for their citizens. Overall, edible oil price forecasting plays a crucial role in supporting a stable, efficient, and sustainable food system. Capturing the edible stock price through classical models is impossible due to the market's complexity and chaotic dynamics and many non-decidable, nonstationary stochastic variables involved (Marszalek and Burczynski, 2014). Therefore, deep learning models were used to combat these above issues. In this paper,LSTM-based Stacked Autoencoder (LSTM- SAE) was used to capture the above-stated problems and compared with other deep learning models like Gated Recurrent Unit (GRU), Long-Short Term Memory (LSTM), Bi-directional LSTM (Bi-LSTM).

LSTM autoencoders have gained significant attention in recent years as a powerful tool for time series forecasting. An LSTM autoencoder is a type of neural network that utilises Long-Short Term Memory (LSTM) units to learn a compact representation of the input time series. The network is trained to encode the input time series into a low-dimensional latent space and then decode it back to the original time series. By minimising the reconstruction error between the input and the output, the network learns to capture the most important features of the time series in the latent space. The trained LSTM autoencoder can then be used for time series forecasting by predicting the future values of the time series based on the learned representation in the latent space. Unlike traditional statistical models, LSTM autoencoders can capture complex temporal

Corresponding author: K.N. Singh
E-mail address: kn.singh@icar.gov.in

patterns and non-linear relationships in the data, making them a powerful tool for time series forecasting.

In the remainder of this paper, section 2 reviews the literature used for the financial market prediction and other domains. Section 3 formulates the problem and gives detail on LSTM-Stacked Autoencoders (LSTM-SAE). Furthermore, it deals with the experimental analysis with the proposed model and compares the obtained results with those given by other prediction models. Finally, sections 4 and 5 deal with the results of the real dataset, its conclusions, and references.

2. REVIEW OF LITERATURE

Analysing Univariate Time Series (UTS) data is easy and common; however, analysing Multivariate Time series (MTS) data is complex due to the correlated signals involved (Wei *et al.,* 2006).The key challenge in MTS problems is to model such complex real-world data and automatically learn the latent features from the correlated input data (Yi *et al.,* 2017). Several studies use autoencoders to predict time series (Gensler *et al.*, 2016; Bao *et al.*, 2017; Sagheer and Kotb, 2019). Autoencoders mainly used for dimensionality reduction (Schmidhuber *et al.*, 1992 and Hinton *et al.*, 2006), Classification (Vincent *et al.*, 2010), data denoising (Romeu*et al.*, 2015), image generation (Zha *et al.*, 2018), anomaly detection (Guo *et al.*, 2018, Kieu *et al.*, 2019), generation of data (Zha *et al.*, 2018), missing value imputation (Pan *et al.*, 2022) and time series forecasting (Sagheer and Kotb, 2019, Wu *et al.*, 2019, Xie and Yu, 2021). Recent literature reviews about the Autoencoder are listed in Table 1.

Table 1. Review of time series data and modelling tasks using Autoencoder

Authors	Application	Values in task	Model category
(Pan*et al.*, 2022)	Imputation	Median of the input data	AM, DAE
(Zha *et al.*, 2018)	Generation	Smart grid data	VAE, GRU
(Zhang *et al.,* 2019)	Anomaly Detection and Generation	Time series	LSTM, VAE, Encoder-Decoder
(Guo *et al.*, 2018)	Anomaly Detection	Simulations on real world dataset	GRU, VAE
(Kieu *et al.*, 2019)	Anomaly Detection	Electrocardiography (ECG)	S-RNNs
(Oleksandra *et al.*, 2019)	Anomaly Detection	Rare sound events dataset	Encoder-decoder
(Zhou *et al.*, 2022)	Anomaly Detection	Medical and Meteorologic data	AE, CNN and GRU
(Wu*et al.*, 2023)	Anomaly Detection	Air quality data (Indore)	LSTM, AE
(Li Xie and Sheng Yu, 2021)	Forecasting	SPY, NASDAQ, DJIA index	ANN, CNN, RNN, LSTM
(Sagheer and Kotb, 2019)	Forecasting	Medical and meteorologic data	LSTM-RN and GRU
(Wu*et al.*, 2019)	Forecasting	Heart rate ECG signals	CNN, RNN
(Jung and Choi., 2021)	Forecasting	FXVIX, BPVIX, EUVIX, BPVIX	Hybrid ANN based on AE and LSTM

3. MATERIALS AND METHODS

3.1 Gated recurrent unit (GRU)

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho *et al.* (2014) as an alternative to the LSTM. GRU uses gating mechanisms to control the flow of information between cells in the neural network and is composed of two gates: an update gate and a reset gate. These gates help filter out unnecessary information, solving the problems of vanishing and exploding gradients commonly encountered in traditional RNNs. Compared to LSTM, GRU has fewer parameters due to the absence of one gate and can only store long and short-term memory in the hidden state, lacking a separate cell state. Recent studies have shown that GRUs outperform LSTMs on certain smaller and less frequent datasets. Fig. 1 displays the internal architecture of a GRU unit cell.

The process can be described as:

$$Z_t = \sigma\left(x_t w^z + h_{t-1} U^z + b_z\right)$$

$$r_t = \sigma\left(x_t w^r + h_{t-1} U^r + b_z\right)$$

$$\widetilde{h}_t = tan\left(r_t \times h_{t-1} U + x_t W + b\right)$$

$$h_t = \left(1 - Z_t\right) \times \widetilde{h}_t + Z_t \times h_{t-1}$$

where w^z, w^r, W denote the weight matrices for the corresponding connected input vector. U^z, U^r, U represent the weight matrices of the previous time step, and b_r,b_z and b are bias. The σ denotes the sigmoid function, r_t denotes the reset gate, z_t denotes the update gate, and $\widetilde{h}_t$ denotes the candidate hidden layer.

3.2 Long-Short Term Memory (LSTM)

Long short-term memory (LSTM) is a specific type of recurrent neural network (RNN) architecture proposed by Hochreiter *et al.* in 1997. Unlike a traditional feed-forward neural network, it includes feedback connections, making it suitable for single-point and sequential data. The essential components of LSTM are an input gate, an output gate, and a forget gate. The LSTM network was developed to overcome the vanishing gradient problem that commonly occurs in traditional RNNs during training. LSTM is a cell memory unit, which means that it can remove or add information to the cell state. By incorporating a specific internal structure into the model, LSTM has resolved the vanishing and exploding gradients problems commonly encountered in RNNs. Nowadays, LSTM is recognised as a powerful method for processing, classifying, and predicting time series data.

There are three gate controls: input gate (i_t), output gate (o_t), and forget gate (f_t) in LSTM cell. The structure of the LSTM unit is shown in Fig. 2 and the main calculation process is described as follows

The main calculation process is described as follow:

$$i_t = \sigma\left(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}C_{t-1} + b_i\right)$$

$$f_t = \sigma\left(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}C_{t-1} + b_f\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g\left(W_{cx}x_t + W_{cm}m_{t-1} + b_c\right)$$

$$o_t = \sigma\left(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}C_{t-1} + b_o\right)$$

$$m_t = o_t \odot h\left(c_t\right)$$

$$y_t = \varphi(W_{ym}m_t + b_y)$$

where c_t represent the memory cell, w and b are the weight matrices and the bias vectors, respectively, x_t denotes the input data at a present time step t, σ is the sigmoid function, and φ is the output activation function.

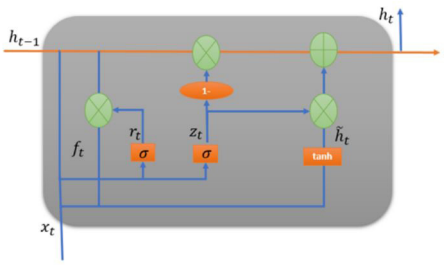

Fig. 1. Architecture of the GRU

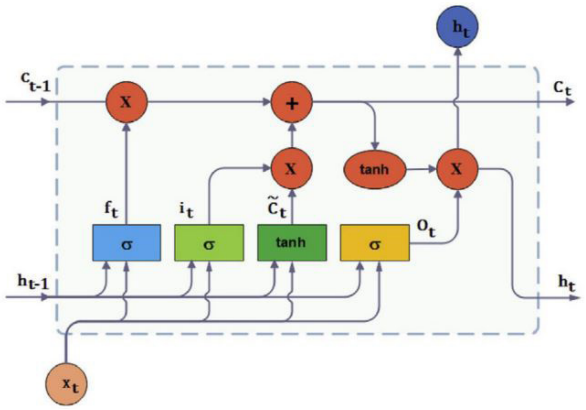

Fig. 2. Architecture of the LSTM

3.3 Bidirectional LSTM (Bi-LSTM)

The bidirectional LSTM (Bi-LSTM) model, illustrated in Fig. 3, consists of a forward LSTM layer and a backward LSTM layer. The LSTM hidden vectors of the forward and backward layers at time t are denoted as $\vec{h}_t$ and $\overleftarrow{h}_t$, respectively. These hidden vectors are independent of each other and are only related to their respective LSTM layers, as shown in the figure. The output of the Bi-LSTM (y_t) is obtained by taking a weighted combination of these two hidden layers. This process can be described as:

$$\vec{h}_t = LSTM\left(x_t, \vec{h}_{t-1}\right)$$

$$\overleftarrow{h}_t = \text{LSTM}\ \left(x_t, \overleftarrow{h}_{t+1}\right)$$

$$y_t = \delta\left(W_{\vec{h}_y}\vec{h}_t + W_{\overleftarrow{h}_t}\overleftarrow{h}_t + b_y\right)$$

where LSTM(.), represents LSTM network, $W_{\vec{h}_y}$ and $W_{\overleftarrow{h}_t}$ represent the weight of the forward and backward LSTM layer at time t, respectively, b_y denotes the bias of the output layer, δ (.) represents the activation function.

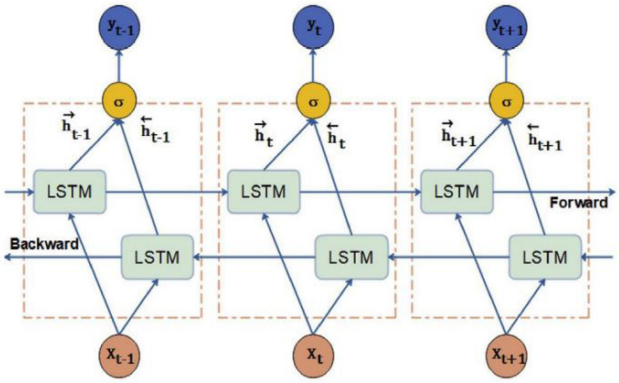

Fig. 3. Architecture of the Bi-LSTM

3.4 Autoencoder (AE)

Autoencoder (AE) was first introduced by Rumelhart *et al.* (1985) as a neural network for self-supervised learning. AE consists of an input layer, output layer, and hidden layers, and its goal is to derive a representation for an input dataset, such as dimensionality reduction, while keeping the reorganised data as close as possible to the input data. In Fig. 4, the encoder stage learns important characteristics of the inputs, while the decoder generates outputs similar to the inputs. The output represents a state where the noise of the inputs is removed, resulting in more distinct characteristics.

$$Y = f\ (W_1\ .\ X + b)$$

$$\tilde{X} = \tilde{f}\ (W_2\ .\ Y + b)$$

where W_1 is the weight between input X and hidden representation Y, W_2 is the weight between a hidden representation Y and the output $\tilde{X}$, and b is the bias, f and $\tilde{f}$ represent the encoder and decoder, respectively, f accepts and compresses the input data (X) into a latent space (Y), and $\tilde{f}$ is responsible for accepting latent space (Y) representations and reconstructing original inputs ($\tilde{X}$).

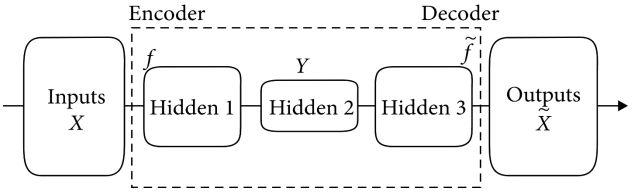

Fig. 4. Structure of an autoencoder

3.5 Stacked Autoencoder (SAE):

To improve performance by manipulating hidden layers, Stacked Autoencoder (SAE) is used in several methods. When a neural network is deep, a stacked autoencoder is utilised to solve the vanishing gradient problem by stacking hidden layers. Fig. 5 illustrates a simple example of a stacked autoencoder, where hidden nodes are increased by stacking autoencoders hierarchically.

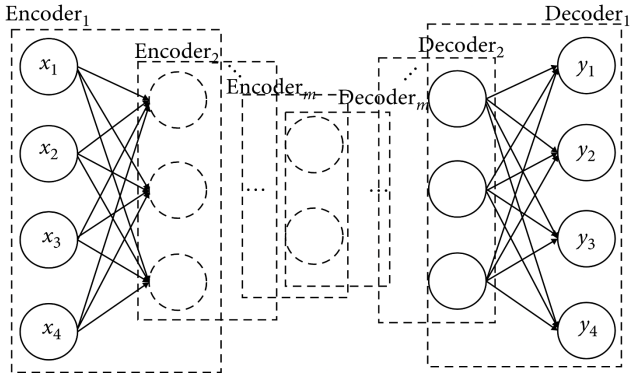

Fig. 5. Structure of a Stacked Autoencoder (SAE)

3.6 LSTM Autoencoder (LSTM-AE)

The LSTM Autoencoder has a similar structure to an autoencoder but is built using LSTM layers, as illustrated in Fig. 6. The LSTM-AE model has the ability to learn complex and dynamic input sequence data from adjacent periods by utilising memory cells that can remember long input sequence data.

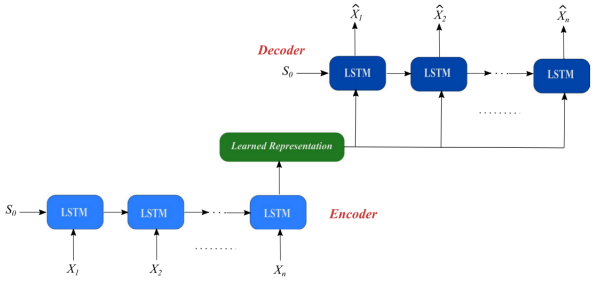

Fig. 6. Structure of a LSTM Autoencoder (LSTM-AE)

3.7 LSTM based Stacked Autoencoder (LSTM-SAE)

The LSTM-based Stacked Autoencoder (LSTM-SAE) has the same structure as LSTM-AE, but with stacked LSTM cells to solve the vanishing gradient problem (Fig. 7).

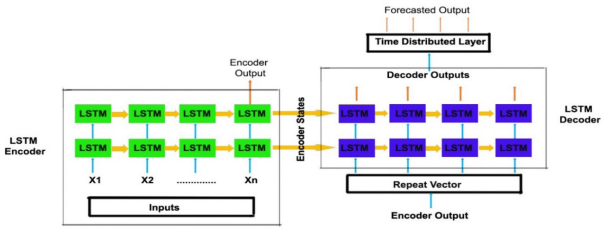

Fig. 7. Structure of a LSTM based Stacked Autoencoder (LSTM-SAE)

Adding the "Repeat Vector" to the layer is nothing but repeating the input n number of times. Whereas, the "Time Distributed layer" takes the information from the previous layer and creates a vector with a length of the output layers.

4. RESULTS AND DISCUSSION

The experimental dataset for this study consists of the daily price of Ruchi soya stock (in INR), obtained from the Yahoo finance website (https://finance.yahoo.

com/quote/RUCHI.NS?p=RUCHI.NS&.tsrc=fin-srch), covering the period from January 1, 2015, to February 24, 2023. The descriptive statistics of the soya stock series used in the study are presented in Table 2 and depicted in Fig. 8. Table 3 includes statistical tests to confirm the stationarity and linearity of the considered data.

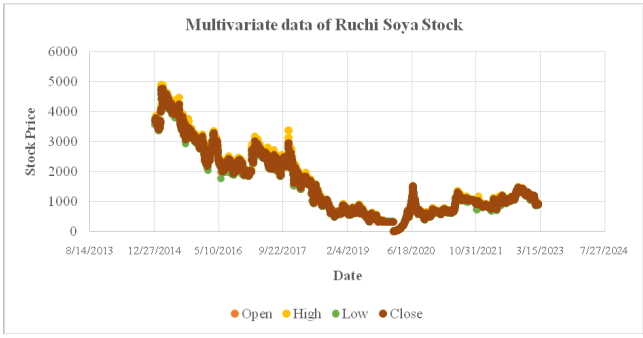

Fig. 8. Ruchi Soya stock price daily data (in INR) from Jan 01, 2015 through Feb 24, 2023

Table 2. Descriptive statistics of Soya stock price series (in INR)

Descriptive statistics	Closing price (in INR)
Minimum	17.00
Mean	1571.21
Maximum	4765.00
Standard deviation	1080.63
Coefficient of variation (%)	68.77
Skewness	0.88
Kurtosis	-0.08
Jarque-Bera	264.26**

A stationary series has a mean/variance, or both remain constant across time. The Augmented Dickey-Fuller (ADF), Phillip-Perron (PP), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests are used to determine whether the data series is stationary or not. The null hypothesis of the ADF test is that the time series contains a unit root, indicating that it is nonstationary. Soya price series failed to reject the null hypothesis in this case study. Meanwhile, PP test defines the null hypothesis that the time series is integrated of order 1. Table 4 shows the results of these tests, which substantiate the non-stationarity.

Table 3. ADF, PP and KPSS test results of daily price series of soya stock

Price series	ADF test		PP test		KPSS test	
	Statistic	p-value	Statistic	value	Statistic	p-value
Soya stock price series	-1.84	0.35	-2.42	0.28	4.95	0.01

Brock-Dechert-Scheinkman (BDS) test has been employed for testing of nonlinearity. The null hypothesis states that the series is independent and identically distributed. The results of the test are shown in Table 5, where it can be seen that the probability values computed at points 0.50 to 2.00 confirm the nonlinearity in soya stock market price data for the embedding dimensions (number of lags) 2 and 3.

Table 4. BDS test for nonlinearity

Epsilon for close points	Embedding dimensions		p- value
	2	3	
0.5 σ	208.08	324.99	<0.0001
1.0 σ	144.16	176.64	<0.0001
1.5 σ	136.38	150.76	<0.0001
2.0 σ	121.97	122.33	<0.0001

The normalisation technique changes the form of a data series by rescaling its values between 0 and 1.

$$X_t^{'} = \left(X_t - X_{min}\right) / \left(X_{max} - X_{min}\right)$$

where X_{min}, X_{max} and X_t are the minimum, maximum and observation at time t, respectively and $X_t^{'}$ is the rescaled value. In python software, we have used MinMaxScaler function of the Scikitlearn package for this purpose.

All the above models are fitted by using Python software. The model implementation on the dataset (2017 data points) begins with partitioning of the data series in to 70, 20 and 10 percent [*i.e.*, training (1412 data points), validation (404 data points) and testing (201 data points)]. The use of hyperparameters is an important aspect of deep learning, as they can greatly affect the performance of a model. In this case, the hyperparameters selected were the number of lags (1, 5, 10, 30), batch size (8, 16, 32, 64,128, 256), the number of epochs (50, 100, 150, 200), the number of hidden layers (1,2), the number of hidden units (8, 16, 32, 64, 128, 256) and dropout rate (0.0, 0.1, 0.2). These were chosen based on their potential impact on the model's performance and were varied across a range of values in order to find the optimal combination through grid search cross validation technique. After obtaining the best hyperparameters, training has to be done. The optimal hyperparameter of the models has been given in Table 5.

The batch size determines the number of samples that are processed before the model's weights are updated. Larger batch sizes can lead to more stable

gradients but can also require more computational resources. The number of epochs determines the number of times the model is trained on the entire dataset. More epochs can lead to better model performance, but can also increase the risk of overfitting. The number of input units determines the number of variables that the model takes as input. A larger number of input units can allow the model to capture more complex relationships in the data, but can also lead to higher computational costs.

Table 5. Optimal hyperparameter selection for several forecasting models

Hyperparameters	GRU	LSTM	Bi-LSTM	LSTM-AE	LSTM-SAE
Batch size	32	128	64	128	64
No. of epochs (Stopping criteria)	76	57	78	27	65
No. of hidden layers	1, 2	2	1,2	1	2
No. of hidden units	32	32, 64	64	128	128, 64
Drop rate	0.1	0.1	0.2	0.3	0.2
No. of lags	10	5	10	30	10

Once the optimal combinations of hyperparameters were found, the model was used for prediction. The use of the early stopping method ensured that the model was not trained for too long, which can lead to overfitting and poor generalisation of new data. Henceforth, the selection and tuning of hyperparameters is an important step in building a machine-learning or deep leaning model, and can greatly affect its performance. It is important to note that the optimal hyperparameters may not be the same for all datasets or tasks, and may need to be determined on a case-by-case basis.

After obtaining the best parameters, all above stated models were trained by using early stopping criteria technique to avoid overfitting of the model. Then, to know the impact of several deep trained models were forecasted for 1 day, 5 day, 10 day and 30 days (Fig.s 9, 10,11 and 12). The result based on RMSE and MAPE revealed that all the above stated models perform well for a-head forecast but as time period increases GRU, Bi-LSTM shows poor performance but LSTM, LSTM-AE and LSTM-SAE performs well (Table 6). During 10 day and 30-day head forecast, LSTM lose its performance but LSTM-AE and LSTM-SAE captures long memory property and also forecast at longer horizon.

Table 6. Optimised results obtained by different models for the Soya stock price series

Model	1 day Forecast		5-day Forecast		10-day Forecast		30-day Forecast	
	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
GRU	101.70	6.65	166.69	11.68	341.59	24.29	350.31	25.36
LSTM	**30.29**	**1.27**	137.129	9.30	206.93	14.56	349.47	25.05
Bi-LSTM	114.28	6.11	153.94	8.01	342.91	23.48	437.81	31.11
LSTM-AE	51.18	3.18	163.43	10.88	108.25	7.01	105.60	7.43
LSTM-SAE	39.34	2.40	**113.92**	**6.82**	**104.85**	**6.71**	**84.27**	**5.64**

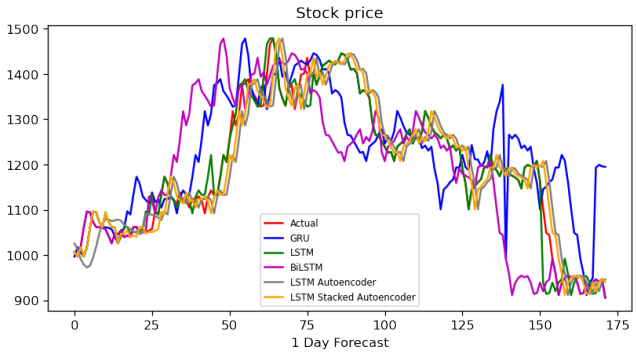

Fig. 9. One day ahead forecast using five different models for test data

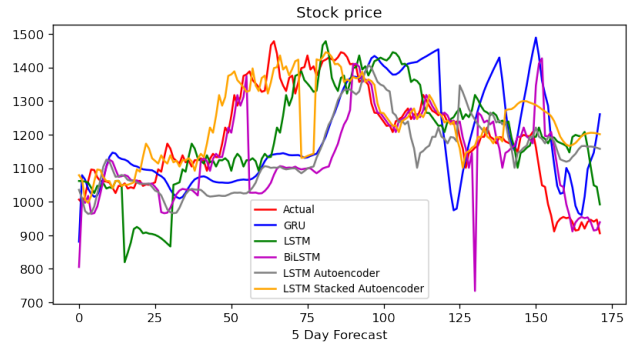

Fig. 10. Five days ahead forecast by using five different models for test data

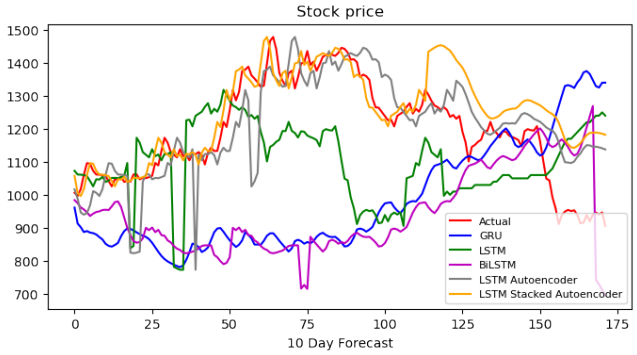

Fig. 11. Ten days ahead forecast by using five different models for test data

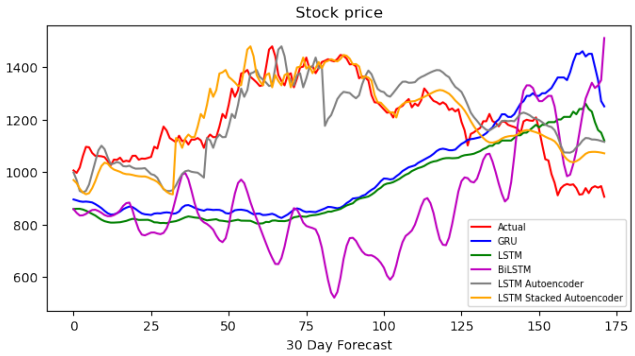

Fig. 12. Thirty days ahead forecast by using five different models for test data

LSTM Stacked Autoencoders (LSTM-SAEs) are a powerful tool for time series forecasting, as they can capture long-term dependencies using RNNs, especially LSTM cells. Moreover, SAEs employ an encoder that compresses multivariate time series into a smaller latent space, identifying critical features like trend, seasonality, and chaotic patterns while filtering out irrelevant or noisy data. By reducing the dimensionality of the input through compression into a smaller representation in the latent space, autoencoders make it easier to learn and model complex patterns in the time series data. As a result, the model can produce more precise forecasts and achieve better overall performance by capturing both temporal dependencies and chaotic patterns in the time series data.

5. CONCLUSION

The goal of this study was to forecast Ruchi soya stock price series based on deep learning models. From our study it can be inferred that the proposed model LSTM-SAE model outperformed the other competitive model like LSTM-AE, LSTM, Bi-LSTM, and GRU. In practice, our findings can be helpful to researchers and policy makers who determine national economic policies related to edible oil marketing because Ruchi soya stock reveal important trends for the Indian economy. Moreover in this study, hyperparameter optimisation was performed using only a grid search cross validation technique, which is a commonly used machine learning/ deep learning algorithm, but there is also a research gap to increase the reliability of prediction by considering additional optimisation algorithms.

Code availability: Code will be available on request to the corresponding author.

REFERENCES

Araujo, R.D.A., Oliveira, A.L. and Meira, S. (2015). A hybrid model for high-frequency stock market forecasting. *Expert Systems with Applications*, **42(8)**, 4081-4096.

Bao, W., Yue, J. and Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, **12(7),** e0180944.

Cho, K., Van Merrienboer, B., Bahdanau, D. and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:*1409.1259. (https://arxiv.org/abs/1409.1259 was available for access on May 31, 2022)

Chong, E., Han, C. and Park, F.C. (2017). Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, **83**, 187-205.

Gianniotis, N., Kugler, S.D., Tino, P. and Polsterer, K.L. (2016). Model-coupled Autoencoder for time series visualisation. *Neurocomputing*, **192,** 139-146.

Guo, Y., Liao, W., Wang, Q., Yu, L., Ji, T. and Li, P. (2018, November). Multidimensional time series anomaly detection: A gru-based gaussian mixture variational autoencoder approach. In *Asian Conference on Machine Learning* (97-112).

Haykin, S. (2009). *Neural networks and learning machines, 3/E*. Pearson Education India.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9(8)**, 1735-1780.

Homayouni, H., Ghosh, S., Ray, I., Gondalia, S., Duggan, J. and Kahn, M.G. (2020). An autocorrelation-based lstm-autoencoder for anomaly detection on time-series data. In *2020 IEEE International Conference on Big Data (Big Data)* (5068-5077).

Jaiswal, R., Jha, G.K., Kumar, R.R. and Choudhary, K. (2021). Deep long short-term memory-based model for agricultural price forecasting. *Neural Computing and Applications*, **34,** 4661–4676. https://doi.org/10.1007/s00521-021-06621-3

Jiang, S., and Durlofsky, L.J. (2021). Data-space inversion using a recurrent autoencoder for time-series parameterisation. *Computational Geosciences*, **25,** 411-432.

Jung, G., and Choi, S.Y. (2021). Forecasting foreign exchange volatility using deep learning autoencoder-LSTM techniques. *Complexity*, https://doi.org/10.1155/2021/6647534

Kieu, T., Yang, B., Guo, C., and Jensen, C.S. (2019, August). Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *IJCAI* (2725-2732).

Li, L., Yan, J., Wang, H., and Jin, Y. (2020). Anomaly detection of time series with smoothness-inducing sequential variational auto-encoder. *IEEE transactions on neural networks and learning systems*, **32(3),** 1177-1191.

Liguori, A., Markovic, R., Dam, T.T.H., Frisch, J., van Treeck, C., and Causone, F. (2021). Indoor environment data time-series reconstruction using autoencoder neural networks. *Building and Environment*, **191**, 107623.

Liu, P., Sun, X., Han, Y., He, Z., Zhang, W., and Wu, C. (2022). Arrhythmia classification of LSTM autoencoder based on time series anomaly detection. *Biomedical Signal Processing and Control*, **71,** 103228.

Mohanty, D.K., Parida, A.K., and Khuntia, S.S. (2021). Financial market prediction under deep learning framework using autoencoder and

kernel extreme learning machine. *Applied Soft Computing*, **99,** 106898.

Nguyen, N., and Quanz, B. (2021, May). Temporal latent auto-encoder: A method for probabilistic multivariate time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, **35(10)**, 9117-9125.

Noering, F.K.D., Schroeder, Y., Jonas, K. and Klawonn, F. (2021). Pattern discovery in time series using Autoencoder in comparison to nonlearning approaches. *Integrated Computer-Aided Engineering*, **28(3),** 237-256.

Pan, Z., Wang, Y., Wang, K., Chen, H., Yang, C. and Gui, W. (2022). Imputation of missing values in time series using an adaptive-learned median filled deep autoencoder. *IEEE Transactions on Cybernetics*, **53**(2), 695-706. DOI: 10.1109/TCYB.2022.3167995.

Provotar, O.I., Linder, Y.M. and Veres, M.M. (2019, December). Unsupervised anomaly detection in time series using lstm-based autoencoders. *International Conference on Advanced Trends in Information Theory (ATIT)*, 513-517.

Richard, G., Grossin, B., Germaine, G., Hebrail, G. and de Moliner, A. (2020). Autoencoder-based time series clustering with energy applications. *arXiv preprint arXiv:2002.03624*.

Sagheer, A., and Kotb, M. (2019). Unsupervised pre-training of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, **9(1)**, 1-16.

Shumway, R.H., Stoffer, D.S. and Stoffer, D.S. (2000). *Time series analysis and its applications* **(3)**.New York: springer.

Terefe, T., Devanne, M., Weber, J., Hailemariam, D. and Forestier, G. (2020, November). Time series averaging using multi-tasking autoencoder. *32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 1065-1072

Von Schleinitz, J., Graf, M., Trutschnig, W. and Schröder, A. (2021). VASP: An autoencoder-based approach for multivariate anomaly detection and robust time series prediction with application in motorsport. *Engineering Applications of Artificial Intelligence*, **104**, 104354.

Wei, Y., Jang-Jaccard, J., Xu, W., Sabrina, F., Camtepe, S. and Boulic, M. (2023). Lstm-autoencoder based anomaly detection for indoor air quality time series data. *Sensors*.DOI: https://doi.org/10.1109/JSEN.2022.3230361

Xie, L., and Yu, S. (2021). Unsupervised feature extraction with convolutional Autoencoder with application to daily stock market prediction. *Concurrency and Computation: Practice and Experience*, **33(16)**, 6282.

Wu, K., Liu, J., Liu, P., and Yang, S. (2019). Time series prediction using sparse Autoencoder and high-order fuzzy cognitive maps. *IEEE transactions on fuzzy systems*, **28(12)**, 3110-3121.

Xu, X., and Ren, W. (2022). A hybrid model of stacked Autoencoder and modified particle swarm optimisation for multivariate chaotic time series forecasting. *Applied Soft Computing*, **116**, 108321.

Yin, C., Zhang, S., Wang, J., and Xiong, N.N. (2020). Anomaly detection based on convolutional recurrent Autoencoder for IoT time series. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, **52(1)**, 112-122.

Yu, W., Kim, I.Y., and Mechefske, C. (2021). Analysis of different RNN autoencoder variants for time series classification and machine prognostics. *Mechanical Systems and Signal Processing*, **149,** 107322.

Zha, M. (2022). Time series generation with masked Autoencoder. *arXiv preprint arXiv:2201.07006*.

Zhang, C., Li, S., Zhang, H., and Chen, Y. (2019). Velc: A new variational autoencoder based model for time series anomaly detection. *arXiv preprint arXiv:1907.01702.*

Zhang, J., and Dai, Q. (2022). Latent adversarial regularised Autoencoder for high-dimensional probabilistic time series prediction. *Neural Networks*, **155,** 383-397.

Zhao, X., Han, X., Su, W., and Yan, Z. (2019, November). Time series prediction method based on convolutional Autoencoder and LSTM. In *2019 Chinese Automation Congress (CAC)*, 5790-5793.

Zhou, H., Yu, K., Zhang, X., Wu, G., and Yazidi, A. (2022). Contrastive Autoencoder for anomaly detection in multivariate time series. *Information Sciences*, **610,** 266-280.

APPENDIX

Python code for the LSTM stacked autoencoders (LSTM-SAEs) for multivariate time series forecasting

```
X_train, y_train = split_series(train.values,n_past, n_future)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1],n_features))
y_train = y_train.reshape((y_train.shape[0], y_train.shape[1], n_features))
X_test, y_test = split_series(test.values,n_past, n_future)
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1],n_features))
y_test = y_test.reshape((y_test.shape[0], y_test.shape[1], n_features))
encoder_inputs = tf.keras.layers.Input(shape=(n_past, n_features))
encoder_l1 = tf.keras.layers.LSTM(128, return_sequences = True, return_state=True)
encoder_outputs1 = encoder_l1(encoder_inputs)
encoder_states1 = encoder_outputs1[1:]
encoder_l2 = tf.keras.layers.LSTM(128, return_state=True)
encoder_outputs2 = encoder_l2(encoder_outputs1[0])
encoder_states2 = encoder_outputs2[1:]
decoder_inputs = tf.keras.layers.RepeatVector(n_future)(encoder_outputs2[0])
decoder_l1=tf.keras.layers.LSTM(128, return_sequences=True)(decoder_inputs,initial_state = encoder_states1)
decoder_l2 = tf.keras.layers.LSTM(128, return_sequences=True)(decoder_l1,initial_state = encoder_states2)
decoder_outputs2 = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(n_features))(decoder_l2)
model_e2d2 = tf.keras.models.Model(encoder_inputs,decoder_outputs2)
model_e2d2.summary()
```