International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Proposed Hybrid GA-TDDPL-CNN-LSTM Architecture for Stock Trend Prediction

Wei Chuan Loo[1], Esraa Faisal Malik[2], XinYing Chew[3*], Khai Wah Khaw[4], Sajal Saha[5] , Ming Ha Lee[6], Mariam Al Akasheh[7]

Submitted: 26/04/2023 **Revised**: 28/06/2023 **Accepted**: 06/07/2023

Abstract: This study constructs a hybrid deep learning model to predict the price trend movement of Standard & Poor's 500 index. Predicting stock market price trends is challenging because stock market data are non-linear and complex. Additionally, various factors, such as investor sentiment and news events, exert influence on stock price trends, leading to fluctuations in price trends. Researchers have implemented a variety of machine learning methods to predict stock price movements. The present study develops a hybrid deep learning network model consisting of a feature learning model, which is a long short-term memory model, and a feature selection model. Different types of data, including stock price, smoothing indicators, trend indicators, and oscillator indicators, are used as inputs to improve the model's performance. Furthermore, to optimize the hyperparameter of each feature extraction model and feature selection model, a Genetic algorithm is utilized. An expert rule trend deterministic layer is also implemented to pre-process the data to further improve the model's performance. The results indicate that the proposed model has superior testing performance compared to restricted Boltzmann machine, convolutional neural network, and autoencoder models.

Keywords: *CNN, expert rule, genetic algorithm, LSTM, RBM, stock market*

1. Introduction

The trend of a stock market index is defined as the tendency of the price series for the financial securities to go downward or upward. The ability to predict stock trends can help investors to obtain higher earnings. Stock market price movement can be predicted by macroeconomic variables such as dividend price ratio, inflation rate and future profit growth, and by technical indicators that rely on the previous price movement and volume pattern. Using technical indicators to predict stock price movements is simple and thus is popular among traders. According to the findings of the authors in [1], the predictive performance of technical indicators is comparable to that of economic variables. Similarly, the authors in [2] used technical indicators as input data to predict price trends. Additionally, applying a trend deterministic data preparation layer (TDDPL) to pre-processing technical data to discretize the data to 0 and 1 has been implemented by [3]. Stock market prediction machine learning methods can be divided into two types: traditional statistical machine learning methods and deep learning methods. Traditional machine learning methods, such as the support vector machine (SVM), logistic regression (LR) and decision trees, are often used in stock trend prediction as they have low complexity and often perform better than other methods when the data sample size is small. In contrast, deep learning methods perform better when the training dataset is very large, and the data are highly non-linear. Several deep learning models have been developed in various studies to forecast stock market movements. The authors in [4] compared the prediction performance of deep learning methods such as recurrent neural network (RNN), multilayer perceptron, long short-term memory (LSTM), convolutional neural network (CNN) and auto regressive integrated moving average statistical methods. Their results indicate that deep learning methods outperform traditional machine learning methods. In the deep learning domain, CNN has received attention due to its ability to extract complex features. In addition, LSTM is popular for predicting time series data due to its ability to solve the long-term dependency problem in RNN. The strength of feature extraction of CNN and the predictive power of LSTM have inspired the researchers to combine both models to form a hybrid neural network [5], [6]. The authors in [7] propose a novel CNN-LSTM hybrid deep learning model based on short-, medium- and long-term features of closing price

[1]*School of Computer Sciences, Universiti Sains Malaysia, Penang, Malaysia*
E-mail: weichuanloo@student.usm.my
[2] *School of Management, Universiti Sains Malaysia, Penang, Malaysia*
E-mail: esraa.f@student.usm.my
[3]*School of Computer Sciences, Universiti Sains Malaysia, Penang, Malaysia*
E-mail: xinying@usm.my (Corresponding Author)
[4] *School of Management, Universiti Sains Malaysia, Penang, Malaysia*
E-mail: khaiwah@usm.my
[5]*Department of Mathematics, IUBAT–International University of Business Agriculture and Technology, Dhaka, Bangladesh*
E-mail: sajal.saha@iubat.edu
[6]*Faculty of Engineering, Computing and Science, Swinburne University of Technology, Sarawak Campus, 93350 Kuching, Sarawak, Malaysia*
E-mail: mhlee@swinburne.edu.my
[7]*Department of Analytics in the Digital Era, College of Business and Economics, United Arab Emirates University, Al Ain 15551, United Arab Emirates*
E-mail: mariam.alakasheh@uaeu.ac.ae

sequences, and they also suggest that different time frames provide unique features for the machine learning model to learn; that study uses two layers of CNN to extract multiple time scale data.

The present paper proposes a hybrid deep learning model called GA(genetic algorithm)-TDDPL-CNN-LSTM to predict the price trend of Standard & Poor's 500 index (S&P 500). The inspiration for this model comes from the TDDPL layer proposed by [3] and the multiple time frame model introduced by [7]. In addition, the hyperparameters of the deep learning model are optimized using GA. The aim of this research is to enhance the performance of the multiple time scale CNN-LSTM model proposed by [7].

To the best of the authors' knowledge, there is currently no deep learning model that utilizes short-, medium- and long-term time frames to predict stock prices. This constitutes the first novel contribution of this paper. Furthermore, instead of solely relying on closing prices, a different type of stock information data is utilized for multiple time scale prediction, marking the second novel contribution. Lastly, expert rules are employed in the data pre-processing stage to train the multiple time scale prediction model, which constitutes the third novel contribution. The structure of this paper is as follows: Section 2 provides a brief overview of the existing literature on deep learning techniques for stock price trend prediction. In Section 3, a novel methodology known as GA-TDDPL-CNN-LSTM is described. The performance of the proposed model is examined and discussed in Section 4. Finally, the paper concludes with our findings and suggestions for future research in Section 5.

2. Related Work

There is growing interest in using machine learning to predict the trend of the stock market [8]–[10]. Deep learning methods have gained a lot of attention due to their performance in predicting highly complex stock market data, as they have outperformed traditional machine learning methods [11]–[13]. Combinations of different types of deep learning layers have also implemented by researchers due to their ability to outperform a simple model. An autoencoder LSTM network model called AE-LSTM was proposed in [14] to predict stock market price trends. The autoencoder consisted of LSTM layers in the decoder and encoder parts. After each LSTM layer, dropout was used as a regularization to prevent overfitting. During training, the encoder part was used as the feature generator to feed the LSTM based forecaster to predict the next day's price. The model trained by the authors used S&P 500 index stock data from 2019 to 2020. Their data consisted of six features: adjusted closing price, open, high, low, and closing prices and trading volume. Nine technical indicators were used, including moving average, convergence/divergence, relative strength index (RSI), Williams %R, stochastic oscillator, price rate of change (PROS), average directional index(ADI), Bollinger bands and logarithmic return. Their results indicated a lower mean absolute error and root mean square error than generative adversarial network.

In the research performed by [15], wavelet transform (WT), stacked autoencoders and LSTM were used to predict stock prices. First, WT was used to decompose the time series data for noise elimination before it was fed into the model. The model consisted of stacked autoencoders to train in an unsupervised manner and LSTM with delays to generate output at the specified future step. The six stock indices used to evaluate the performance of the deep learning model were the DJIA index, S&P 500, Nikkei 225, Hang Seng index, Nifty 50 index and CSI 50. The input variables consisted of daily trading data, technical indicators and macroeconomic variables. The stock price prediction results were compared with LSTM and RNN, and the proposed model had higher accuracy and less volatility than LSTM and RNN. The authors suggested that the autoencoder reduced the noise of the input and therefore produced less variance prediction.

Similar research was performed by [16] to construct a stacked denoising autoencoder (SDAE) to forecast the financial market direction in order to predict the daily CSI 300 index from Shanghai and Shenzhen Stock Exchanges in China. SDAE was the extension of the stacked autoencoder. The training of the SDAE started with the training of the first layer as an autoassociator to minimize errors in the reconstruction of the output. The output of the autoassociator was the input for another layer to generate another autoassociator. This step was iterated to generate a specific number of layers. The last hidden layer output was taken as the input to the supervised layer with specific parameters. Finally, the hyperparameters of this model were fined tuned based on the supervised criterion. The SDAE consisted of two key facades which were a multilayer perceptron and autoencoders. During the first stage of training, each autoencoder was trained separately. In the second stage, the multilayer perceptron was trained. The authors suggested that this layer-wise training procedure has better performance and achieves better generalization than random initialization of deep networks. To train the model, 28 technical indicators from moving averages, trend detection, oscillators, volume, momentum, and price volume were used as input features. The output of the model is "-1" if the price of the index tomorrow is lower than today and "1" if it is higher. The authors discovered that SDAE had higher accuracy than SVM of radial kernel function and three-layer backpropagation network. The authors concluded that SAED can remove large amounts of noise in stock market data and was able to find an underlying common pattern among stock data, therefore performing better than the non-deep learning method and swallow layer deep learning method.

CNN is one of the most popular methods in stock market prediction due to its ability to extract sophisticated features without the need for complex pre-processing. CNNs are normally combined with pooling and fully connected layers to form a neural network architecture. The authors in [2] implemented a CNN model to forecast the hourly stock price trend of 100 stocks in the Turkish stock market. A novel graph convolutional feature neural network to simultaneously learn the feature from stock market movement and individual stock movement to predict the trend of the stock price movement was proposed by [17]. Furthermore, the authors in [18] implemented a CNN to predict the stock price movement in China's stock market with feature normalization, in which they utilized reinforcement learning to simulate the expert decision and enhanced the reinforcement learning with CNN. Additionally, RNN is needed to learn the time dependency feature of stock price movement. However, conventional RNN suffers from gradient exploding and vanishing errors if the time sequence is too large; as a result, LSTM was introduced by [19]

Different deep learning models can be combined to form a hybrid model that can utilize the advantages of different models. A popular example is the combination of CNN and LSTM network. The authors in [18] proposed a hybrid CNN-LSTM deep neural network model to predict stock trends. The CNN model is used to extract the feature from buying volume and transaction number. bidirectional long short-term memory (BiLSTM) was implemented to learn the data extracted from the CNN model. The authors in [7] combined a CNN and LSTM network to predict the stock price trend with short-, medium- and long-term features of the stock price. In their paper, the authors employ different numbers of CNN layers to extract multiple time frame features and use one LSTM model to learn the time dependency feature of a time frame. Finally, the output of each LSTM model is fed into fully connected layer for final trend prediction [7].

To improve the performance of the model, data are often pre-processed using a denoising signal processing algorithm or rule defined with domain knowledge. A WT and autoencoder to eliminate the noise in the model were utilized by [15]. Moreover, the authors in [7] implemented a stacked denoised autoencoder to pre-process the input of their stock trend prediction model. The authors in [5] utilized a restricted Boltzmann machine (RBM) to extract useful features from the data while eliminating noise. They also implemented TDDPL to convert the technical indicators' input to binary values based on a set of expert rules. The authors conclude that RBM and TDDPL pre-processing methods improve on the performance of the base model in stock trend prediction [5].

3. Methodology

This section describes data science project lifecycle, including the analysis of different deep learning models and the construction of the proposed hybrid deep learning model, GA-TDDPL-CNN-LSTM. Furthermore, it provides a detailed description of the ensemble learning method. The target of the prediction, upward or downward price trend, is defined by using the following rule as shown in equation (1), where *n* is the number of days and *xi* is the closing price for the day. Y = 1 indicates the price trend of the day is an uptrend; Y = 0 indicates the price trend of the day is a downtrend. *n* is set to 20 in our experiment.

$$Y = \begin{cases} 1 \; x_i < x_{i+n} \\ 0 \; x_i > x_{i+n} \end{cases} \tag{1}$$

3.1. Data Collection

The dataset utilized in this research consists of daily S&P 500 index data, spanning from January 30, 1999, to January 30, 2019. The selection of this specific index was motivated by its frequent usage in tracking the performance of the stock market in the United States, as well as its common application in stock trend prediction studies [8], [17]. The dataset was partitioned into training, validation, and testing sets. The data can be obtained from Yahoo Finance in Excel format, including the opening, high, low and closing points of the index, as well as volume information.

3.2. Data Pre-processing

Three types of technical indicators are calculated from the dataset. The first technical indicator is the exponential moving average (EMA) of 10 days, which is a type of smoothing indicator. EMA is calculated using equation (2):

$$EMA_{t,n} = P * \left(\frac{2}{1+n}\right) + EMA_{t-1,n} * \left(\frac{2}{1+n}\right) \tag{2}$$

where *t* is the trading period, *n* is the number of days to average, and *P* is the closing price of the index.

The second type of technical indicator is the RSI for 14 days, which is a type of oscillator indicator. The RSI calculation is shown in equations (3) and (4).

$$RSI = 100 - \frac{100}{1+RS} \tag{3}$$

$$RS = \frac{AvgU}{AvgD} \tag{4}$$

where *RS* is the relative strength, *AvgU* is the average of all up moves in the last 14 days and *AvgD* is the average of all down moves in the last 14 days.

The third type of data is a ROS percentage indicator for 1 month. The ROS percentage indicator is calculated using

equation (5):

$$\text{ROS} = \left(\frac{current\ value}{previous\ value} - 1\right) * 100 \quad (5)$$

where the current value is set as the closing price for the day and the previous value is set as the closing price 30 days before. All the technical indicators shown above are combined with the closing price of the S&P 500 index to form the input data. The training set, validation set and testing set are determined by random shuffle, split in the ratio 0.8:0.1:0.1. After the input data are formed, a TDDPL is constructed to convert the continuous technical indicator data value to discrete 0 or 1 values according to specific rules. This method was proposed by [3] to improve the performance of machine learning methods such as artificial neural networks, SVM and RF methods. The input technical indicator data consists of closing price, RSI, ROS and EMA for 10 days. The discretization of the values is shown in Table 1.

Table 1. Discretization of input data

Technical indicator	Strategy for discretization
Closing price	value = 1, if the closing price at day t is higher than day t-20.
	value = 0, if the closing price at day t is lower than day t – 20
RSI	When RSI not in [30, 70],
	value = 1 if RSIt less than 30
	value = 0 if RSIt more than 70
	When RSI in [30, 70],
	value = 1 if RSIt >= RSIt-1,
	value = 0 if RSIt < RSIt-1
	where RSIt is the RSI value at day t
ROS	value = 0 if ROCt is < 0,
	value = 1 if ROCt is >= 0
	where ROCt is ROC at time t
EMA 10	value = 1 if closing price is more than EMA10
	value = 0 if closing price is less than EMA10

3.3. The Development of the Proposed Model

3.3.1. Deep Learning Model Construction

The description of the deep learning model is split into two components, which are the base deep learning layer for output prediction and the variety of deep learning layers that are combined with the base layer as shown in Fig.1. LSTM is utilized as the base layer for output prediction based on the research by [8] because it has demonstrated incredible performance, outperforming memory free classification methods such as Random Forest (RF) and LR. Additionally, LSTM solves the gradient exploding/vanishing error that occurs in conventional RNN prediction. It is normally combined with max pooling to extract the most valuable feature in a patch. The layer usually utilized is autoencoder, and can be merged with LSTM, as done in [19], and RBM, as done in [5].

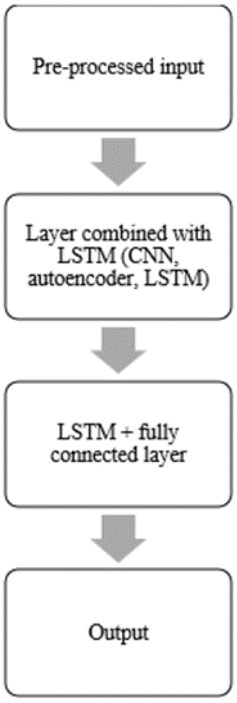

Fig.1 Component of improvised deep learning model

However, the authors in [2] relied solely on CNN in their study. Therefore, we analyze different deep learning models and construct a CNN-LSTM deep learning model. The following points summarize the deep learning experiment model:

- Autoencoder +LSTM
- TDDPL + CNN + LSTM
- CNN + LSTM
- CNN
- TDDPL + RBM + LSTM
- RBM + LSTM

Moreover, LSTM units, type of activation function, and the number of units in the fully connected layer along with the number of network layers, hidden units, activation function, and the pooling patch size of CNN, will be optimized by

using a GA. Autoencoder is an unsupervised artificial neural network that encodes data and learns how to reconstruct the data and reduce noise. It consists of an input layer, a hidden layer and an output layer. In this research, the activation function of each layer is tuned using a GA. Binary RBM is used in this research because it has higher performance than the Gaussian restricted Boltzmann machine and is not limited to underlying data that follows a Gaussian distribution [5]. We also utilize a binary cross-entropy loss function [7].

Binary cross entropy penalizes if the probability predicted is far from the expected value and is famously used in binary class classification. A stochastic gradient descent optimizer is utilized to train the neural network with an initialized learning rate provided by the GA. The value of batch size is 80. Early stopping and dropout techniques are implemented to prevent overfitting. The patience of early stopping is set to 70, indicating that the training will be stopped if the validation accuracy does not improve in 70 epochs. The dropout rate is set to 0.2. Batch normalization is implemented after convolution operation to solve internal covariate shifts that occur in the training of deep learning networks.

3.3.2. Ensemble Learning Model

Subsequently, three similar best deep learning models with different time series length inputs are constructed. The first, second and third models have time-series length inputs of 20 days, 40 days and 80 days. The predicted output of the three models will be combined through bagging to generate a positive return in the stock market. We developed the methodology from [20], in which the authors implemented the bagging method to combine linear forecasting models with non-parametric forecasting models to reduce the mean squared forecast error. The bagging model can generate a positive return in the stock market. The ensemble learning model can be constructed by averaging the uptrend probability predicted by each model. The prediction probability of each input model of input time lengths 20, 40 and 80 are combined to form the final prediction output using equation (6):

$$Prob_{comb} = \frac{Prob_{20} + Prob_{40} + Prob_{80}}{3} \quad (6)$$

where *Prob20*, *Prob40* and *Prob80* are the prediction probability of input time series length of 20, 40 and 80. The prediction output is discretized based on equation (7):

$$Pred = \begin{cases} 1 \; if \; prob_{comb} \geq 0.5 \\ 0 \; if \; prob_{comb} < 0.5 \end{cases} \quad (1) \quad (7)$$

3.3.3. GA

GA is a search heuristic used to find out the optimum value by mimicking the evolutionary process of Darwinian evolution. it is applied to the hyperparameter tuning of all deep learning models. The individuals in GA can be represented in a fixed length of the vector and the value in each vector can be a binary or floating point value. The GA starts with 10 individuals with random initial values. First, the fitness score of individuals is evaluated using an accuracy score. Second, the accuracy score is obtained by the accuracy of the trend predictions of the deep learning model trained with the hyperparameter value in the validation dataset. Third, parent selection to select the five best individuals is defined. The parent undergoes a crossover operation to produce offspring. The offspring will be mutated to produce variation in the solution representation. The process will be repeated until a certain generation number is met, or a termination criterion is hit. In our case, the algorithm stops when 100 individuals are produced in each experiment of the deep learning model. The solution with the best validation accuracy score is then picked.

Chromosome solution encoding in an individual

The solution representation of RBM, autoencoder, CNN is represented in this subsection. The chromosome encoding of the autoencoder LSTM model, CNN, CNN-LSTM and RBM are shown in Table 2.

Table 2. Chromosome encoding of the autoencoder LSTM model, CNN, CNN-LSTM and RBM

Name	Gene number	Value range	Binary/Arithmetic
Chromosome encoding of the autoencoder LSTM model			
Learning rate	0	0.001-0.5	Arithmetic
Number of hidden layers	1	1-5	Arithmetic
Number of hidden layers 1	2	5-25	Arithmetic
Number of hidden layers 2	3	5-25	Arithmetic
Number of hidden layers 3	4	5-25	Arithmetic
Number of hidden layers 4	5	5-25	Arithmetic
Number of hidden layers 5	6	5-25	Arithmetic
Fully connected layer number of nodes	1	1-10	Arithmetic
LSTM number of hidden units	2	10-50	Arithmetic
Chromosome encoding CNN			
Learning rate	0	0.001-0.5	Arithmetic
Convolutional kernel size	1	1-7	Arithmetic
Number of convolutional kernels	2	10-50	Arithmetic
Fully connected layer number of nodes	3	10-50	Arithmetic
LSTM number of hidden units	4	10-50	Arithmetic
Chromosome encoding of CNN-LSTM			
Learning rate	0	0.001-0.5	Arithmetic
Convolutional kernel size	1	1-7	Arithmetic
Number of convolutional kernels	2	10-70	Arithmetic
Fully connected layer number of nodes	3	10-50	Arithmetic
LSTM number of hidden units	4	10-50	Arithmetic
Chromosome encoding of RBM			
Number of components	0	0-512	Arithmetic
Learning rate	1	0.001-0.25	Arithmetic
Random state	2	0 (None) or 1	Binary

Individual and population initialization

Initialization of the value of the solution for an individual is performed randomly to ensure that the solution is spread evenly in the search space. There is no general rule to identify the suitable number of individuals in the population, which is dependent on the problem search space size, the problem difficulty, the length of the solution, and other factors.

In this research, 10 individuals are used in a generation and, subsequently, 10 generations are produced in the evolutionary algorithm. Therefore, a total of 100 individuals will be produced in the evolutionary algorithm.

Fitness evaluation

The fitness of an individual is determined by the accuracy of trend prediction in the validation set by using the hyperparameter in the solution represented by the individual. The validation accuracy is equal to the fitness score of the individual. The higher the fitness score, the better the solution represented by the individual.

Parent selection

The purpose of parent selection is to pick the best individual

to generate offspring, with reasonably diversified individuals, from a pool of parents and offspring. There are many types of parent selection methods such as tournament selection, elitism, fitness proportional selection, and linear ranking. Elitism will result in selection of the individuals with the top fitness function. Standard tournament selection will randomly sample *k* individuals with replacements into a tournament size of *k* from the current population. The individual of top n fitness will be selected from the tournament. Compared with linear ranking and exponential ranking, tournament selection is simple to code and not computationally expensive. It also does not require sorting of the entire population first. Fitness proportional selection assigns selection probability based on the fitness value of the individual. Meanwhile, linear or exponential ranking assigns selection probability to an individual based on the rank of the fitness of the individual instead of the fitness value itself. By doing so, linear ranking prevents the entire population from being overwhelmed by the individual whose fitness value is too large at the initial stage, and therefore preserves the diversity of the population. Furthermore, the authors in [21] aimed to identify the performance of each selection method, and their results show that linear ranking outperforms other methods such as tournament selection. Therefore, linear ranking is used in this research. The selection probability of linear ranking is calculated based on equation (8).

$$P_{lin-rank}(i) = ((2-s)/u) + (2i(s-1)/u(u-1)) \quad (8)$$

Table 3 illustrates the selection probabilities of each individual sorted in ascending order when the population size (u) ranges from 6 to 10, with s = 0.05. Elitism is also implemented to ensure that the best individual is preserved to the next generation without being mutated.

Table 3. Selection probability of everyone in a linear ranking

The number of individuals (u)	Selection probability of individually sorted according to ascending order
10	0.05, 0.061, 0.072, 0.083, 0.094, 0.105, 0.117, 0127, 0.139, 0.152
9	0.056, 0.069, 0.083, 0.097, 0.111, 0.125, 0.139, 0.153, 0.167
8	0.0625, 0.08, 0.098, 0.116, 0.134, 0.151, 0.17, 0.1885
7	0.0714, 0.0956, 0.119, 0.143, 0.167, 0.19, 0.214
6	0.083, 0.117, 0.15, 0.18, 0.22, 0.25

Crossover

Crossover occurs between N number of selected parents, to produce *N* number of offspring. There are many methods of crossover, such as uniform crossover, *M* point crossover, and cycle crossover. The crossover method is divided into binary and arithmetic types. For binary values, uniform crossover will be used in this research. Each gene will have a 50% probability of being inherited from one parent. Fig.2 illustrates how uniform crossover works.

For arithmetic values, whole arithmetic crossover is used to produce offspring in this research. Whole arithmetic crossover works by averaging out the value for two parents to produce two offspring. Fig.3 illustrates how whole arithmetic crossover works in this research.

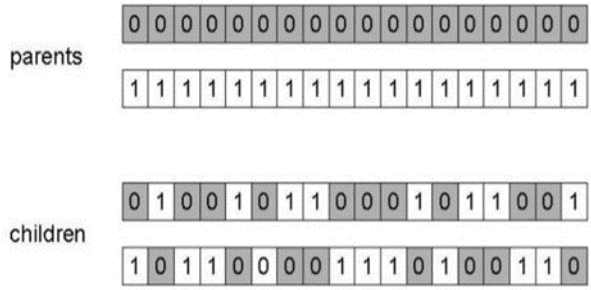

Fig.2 Example of uniform crossover

Fig.3 Example of arithmetic crossover

Mutation

For binary values, bit flip mutation is implemented, which involves flipping the binary value in a gene with a specified probability. The mutation probability is 0.1. Bit flip mutation is illustrated in Fig.4.

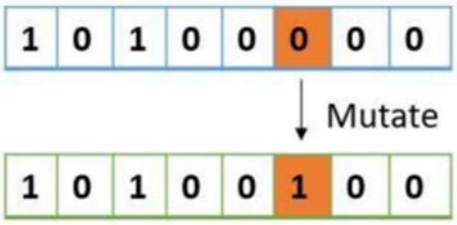

Fig.4 Bit flip mutation

Meanwhile, for arithmetic values, a small increment or decrement value, x, is added to the gene with a specified probability. The mutated value x is different in each gene

depending on the search space of the solution. Table 4 shows the mutations that have been made to each search hyperparameter in autoencoder, CNN, LSTM + fully connected layer, and RBM, respectively.

Table 4. Mutation in autoencoder, CNN, LSTM + fully connected layer, and RBM

Gene number	Value range	Mutation (+/-)
Mutation in autoencoder		
0	0.001-0.5	0.05
1	1-5	2
2	5-25	4
3	5-25	4
4	5-25	4
5	5-25	4
6	5-25	4
Mutation in CNN		
0	0.001-0.5	0.05
1	1-10	2
2	10-70	4
Mutation in LSTM		
0	0.001-0.25	0.05
1	1-10	2
2	10-50	4
Mutation in RBM		
0	1-512	20
1	0.001-0.25	0.05
2	0 or 1	Bit flip

3.4. Model Evaluation

The target of the deep learning model is increasing prices (representing uptrend) and decreasing prices (representing downtrend) after 20 days. If the price increases after 20 days, the target value is 1. If the price decreases after 20 days, the target value is 0. The performance of different feature extractions is compared. The evaluation metrics used are accuracy, precision, and recall. True positive (TP) refers to the total number of correctly predicted uptrends. True negative (TN) is the total number of a correctly predicted downtrends. False positive (FP) is the total number of wrongly predicted uptrends. False negative (FN) is the total number of wrongly predicted downtrends. Accuracy, which represents the proportion of samples that can be correctly predicted among the total number of samples, is calculated by equation (9):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

Precision refers to the proportion of true positive samples compared with the total number of positive identifications. The precision is calculated by equation (10):

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

Recall indicates the proportion of actual positives predicted compared with the number of positives. The recall is calculated by equation (11):

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

4. Results and Discussion

The implementation of GA and the hyperparameter value in each model, as well as the optimum parameter acquired using GA on each run, will be discussed further below. The summary and description of the results of GA are illustrated in Table 5.

Table 5. Summary and description of GA

Symbolic parameters	
Representation	Bit-string
Parent and Survivor selection	Linear ranking selection
Mutation	Bit flip mutation for binary value and uniform mutation for arithmetic value
Crossover	Uniform crossover
Numeric parameter	
Population size	10
Mutation probability	0.2
Crossover probability	0.5

The methods and values shown in Table 5 are used to fine-tune the optimum hyperparameter for each gene shown in Table 6. A total of 100 individuals is generated in the evolutionary algorithm. The individual with the best fitness score will be picked. The optimum hyperparameter obtained from GA runs for autoencoder LSTM, CNN, CNN-LSTM,

and RBM model is shown in Table 6.

Name	Gene number	Value range
The optimum hyperparameter for the autoencoder LSTM		
Learning rate	0	0.4758
Number of autoencoder layer	1	5
Number of hidden layers 1	2	24
Number of hidden layers 2	3	18
Number of hidden layers 3	4	15
Number of hidden layers 4	5	13
Number of hidden layers 5	6	8
Fully connected layer number of nodes	7	8
LSTM number of hidden units	8	20
The optimum hyperparameter for CNN		
Learning rate	0	0.2593
Convolutional kernel size	1	3
Number of convolutional kernels	2	66
Fully connected layer number of nodes	3	31
LSTM number of hidden units	4	20
The optimum hyperparameter for RBM		
Learning rate	0	0.1187
Number of components	1	43
Random state	2	1
LSTM number for hidden units	3	6
Fully connected layer node number	4	5

Table 6. The optimum hyperparameter for the autoencoder LSTM , CNN, CNN-LSTM, and RBM model

The hyperparameter values obtained in the previous table (Table 6) are used to construct the deep learning model. The CNN deep learning model consists of a one-dimensional convolutional layer, one max-pooling layer, and two fully connected layers with a sigmoid activation function. The CNN-LSTM deep learning model consists of a 1D CNN layer connected with a pooling layer, batch normalization layer, dropout layer, LSTM layer, and two fully connected layers. The LSTM deep learning model only consists of an LSTM layer connected with two fully connected layers. The autoencoder model consists of an autoencoder layer connected with LSTM and a fully connected layer. The autoencoder layer consists of an encoder and decoder with five RELU activation functions and fully connected dense layers, respectively. The constructed model receives the pre-processed input to output the stock trend prediction.

The training, validation, and testing accuracy of each model are shown in Tables 7, 8, and 9, respectively. As shown in Table 7 below, using the training dataset, the GA-TDDPL-CNN-LSTM model has the highest accuracy, followed by the TDDPL autoencoder-LSTM, RBM-LSTM, CNN-LSTM, CNN, autoencoder – LSTM, and TDDPL RBM-LSTM. It is observed that the TDDPL layer improves the training performance of the CNN-LSTM model and autoencoder-LSTM model. Furthermore, an improvised deep learning model has better performance than a simple deep learning model. For example, improvised deep learning models such as the CNN-LSTM have slightly better performance than the CNN model.

Table 7. Training accuracy, recall, and precision of each deep learning model

Experiment model	Accuracy	Recall	Precision
GA-TDDPL-CNN-LSTM (proposed model)	0.8832	0.9301	0.8726
CNN-LSTM	0.5493	0.6608	0.5135
CNN	0.5459	0.6028	0.3287
Autoencoder-LSTM	0.5480	0.8615	0.1362
TDDPL autoencoderLSTM	0.6414	0.9469	0.3145
RBM-LSTM	0.5872	0.6634	0.4014
TDDPL RBM-LSTM	0.5217	0.5451	0.3437

Meanwhile, in the validation dataset, the GA-TDDPL-CNN-LSTM has the highest accuracy, followed by TDDPL autoencoder-LSTM, RBM-LSTM, autoencoder LSTM, CNNLSTM, CNN, and TDDPL discretized RBM-LSTM. There was no notable difference in performance between the validation and testing datasets. Therefore, the model trained is not overfitted. Table 8 shows the validation accuracy, recall, and precision of each model.

Table 8. Validation accuracy, recall and precision of each deep learning model

Experiment model	Accuracy	Recall	Precision
GA-TDDPL-CNN-LSTM (proposed model)	0.8508	0.8984	0.8487
CNN-LSTM	0.5693	0.7216	0.5384
CNN	0.5153	0.5322	0.3626
Autoencoder-LSTM	0.5710	0.8285	0.1638
TDDPL autoencoderLSTM	0.6407	0.8615	0.3181
RBM-LSTM	0.6128	0.6283	0.4226
TDDPL RBM-LSTM	0.5125	0.5547	0.3757

As shown in Table 9, utilizing the test data dataset, the GA-TDDPL-CNN-LSTM exhibited the most superior performance, followed by CNN-LSTM, CNN, autoencoder-LSTM, RBM-LSTM, TDDPL-LSTM, autoencoder-LSTM and TDDPL autoencoder-LSTM. Compared with RBM and autoencoder, CNN has the best performance. CNN can extract features of highly complex and noisy data better compared with RBM and autoencoder. When CNN is paired with LSTM, the performance of the deep learning model is slightly superior, which shows that LSTM can learn time dependency data from the feature output by the CNN model. It also indicates that the hybrid model yields better performance than the single CNN model alone. Without data pre-processing, the CNN-LSTM model has a 1% higher accuracy score than RBM-LSTM and a 5% higher score than the autoencoder LSTM model. When combined with the TDDPL layer, in all the testing, training, and validation datasets, GA-TDDPL-CNN-LSTM has the highest performance after using the parameter finetuning by the GA. The TDDPL layer uses the expert rule to help the deep learning model to learn by discretizing the data into 1 (uptrend) and 0 (downtrend). It informs the deep learning model in which condition the technical indicators are bullish or bearish. It is interesting to note that TDDPL has the highest accuracy score percentage increment (32%) on the CNN-LSTM model compared with RBM-LSTM (-2.3%) and autoencoder LSTM (-13.74%). Lastly, the TDDPL CNN-LSTM model with input shapes of (20,4), (40,4), and (80,4) are constructed and trained as illustrated in Table 10. It is noted that the higher the input time length, the higher the performance of the model. The testing accuracy, recall, and precision of the ensemble model using equation (2) are shown in Table 11.

Table 9. Testing accuracy, recall and precision of each deep learning model

Experiment model	Accuracy	Recall	Precision
GA-TDDPL-CNN-LSTM (proposed model)	0.8036	0.8961	0.7920
CNN-LSTM	0.5471	0.7148	0.5138
CNN	0.5313	0.5945	0.3173
Autoencoder-LSTM	0.5188	0.9032	0.1290
TDDPL autoencoderLSTM	0.4561	0.5116	0.2010
RBM-LSTM	0.5414	0.6124	0.3726
TDDPL RBM-LSTM	0.5288	0.6031	0.3674

Table 10. Accuracy, recall and precision of CNN-LSTM of different input time lengths

Input time length of CNN-LSTM	Accuracy	Recall	Precision
20	0.8036	0.8961	0.7920
40	0.8672	0.9049	0.8682
80	0.8844	0.9052	0.8958

Table 11. Accuracy, recall and precision of combination of input length 20, 40 and 80

Model	Accuracy	Recall	Precision
Combination of input length 20, 40 and 80	0.9029	0.9201	0.9071

The results indicate that the testing accuracy, recall, and precision of the proposed GA-TDDPL-CNN-LSTM model are higher than those of the prediction model of the input time length of 20, 40, and 80, respectively. Therefore, the use of ensemble learning can boost the accuracy of the prediction model. Accurate trend prediction is crucial to achieve profitability in stock trading. For profitable trades, the trader should buy when the model predicts an uptrend and sell when the model predicts a downtrend. To ensure profitability, the model must attain an accuracy higher than 50%. In this research, GA-TDDPL-CNN-LSTM, CNN-LSTM, CNN autoencoder-LSTM, RBM-LSTM, and TDDPL RBM-LSTM models can be applied in real-life trading. It is important to consider that in trading, a 1% increase in accuracy corresponds to a 2% improvement in profitability. When the prediction is correct, the individual will both gain profit and avoid a loss of the same magnitude. Therefore, slight accuracy improvement can significantly improve the profitability of trading. It is worth implementing data pre-processing and hyperparameter optimization to improve the accuracy of the model. The results show that the proposed GA-TDDPL-CNN-LSTM deep learning model has the highest testing performance compared to CNN-LSTM, CNN, autoencoder-LSTM, TDDPL autoencoder-LSTM, RBM-LSTM, and TDDPL RBM-LSTM with GA optimization technique. Furthermore, the present findings indicate that building a deep learning model that takes into account multiple time frame features improves the performance of the model.

5. Conclusion

In this research, a proposed deep learning model called GA-TDDPLCNN-LSTM is developed to predict stock market trends. The proposed architecture intends to fill the research gap in regard to the multiple time frame CNN-LSTM model. This research makes three novel contributions: the first is the use of the different types of technical analysis data and expert rule TDDPL layer to improve the performance of the multiple time frame model in stock market trend prediction. The second contribution is the use of a GA to search the hyperparameter for the deep learning model for stock market trend prediction. The search space of the hyperparameter value proposed by the authors is expanded. In the final stage, different input time lengths of GA-TDDPL-CNN-LSTM model are constructed, and the prediction probability of uptrend from each model is combined through voting. The proposed GA-TDDPL-CNN-LSTM model has the best accuracy, recall, and precision. The result shows that TDDPL layer data pre-processing, GA hyperparameter searching, the use of CNN-LSTM as a deep learning model, and multiple time frame training can improve the performance of the model. We hope that further studies can confirm our findings by employing different techniques, e.g., more technical analysis data can be added as an input feature to improve the performance of the model. Furthermore, fundamental financial data such as S&P 500 earnings growth and price to earnings ratio can be incorporated into the feature to improve the result. Correlation between the difference index and news event can also be identified and used as a training feature to boost the performance of the model. In the combination of multiple time frame prediction, instead of using voting ensemble learning, the three different input time lengths GA-TDDPL-CNN-LSTM model can be connected to a dense layer. The dense layer can be trained to assign weightage on each model to produce the output. Finally, various denoising methods such as WT can be used to reduce the noise in the time series stock data before it is input into the TDDPL layer of the model.

Acknowledgements

This work is supported by Ministry of Higher Education Malaysia, Fundamental Research Grant Scheme Grant No. FRGS/1/2022/STG06/USM/02/4, for the project entitled "Efficient Joint Process Monitoring Using a New Robust Variable Sample Size and Sampling Interval Run Sum Scheme".

References

[1] C. J. Neely, D. Rapach, J. Tu, and G. Zhou, "Forecasting the Equity Risk Premium: The Role of Technical Indicators," *SSRN Electron. J.*, Jan. 2013, doi: 10.2139/SSRN.1787554.

[2] H. Gunduz, Y. Yaslan, and Z. Cataltepe, "Intraday prediction of Borsa Istanbul using convolutional neural networks and feature correlations," *Knowledge-Based Syst.*, vol. 137, pp. 138–148, Dec. 2017, doi: 10.1016/J.KNOSYS.2017.09.023.

[3] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques," *Expert Syst. Appl.*, vol. 42, no. 1, pp. 259–268, Ja[21n. 2015, doi: 10.1016/J.ESWA.2014.07.040.

[4] M. Hiransha, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "NSE Stock Market Prediction Using Deep-Learning Models," *Procedia Comput. Sci.*, vol. 132, pp. 1351–1362, Jan. 2018, doi: 10.1016/J.PROCS.2018.05.050.

[5] Q. Liang, W. Rong, J. Zhang, J. Liu, and Z. Xiong, "Restricted Boltzmann machine based stock market trend prediction," *Proc. Int. Jt. Conf. Neural Networks*, vol. 2017-May, pp. 1380–1387, Jun. 2017, doi: 10.1109/IJCNN.2017.7966014.

[6] W. Lu, J. Li, Y. Li, A. Sun, and J. Wang, "A CNN-LSTM-based model to forecast stock prices," *Complexity*, vol. 2020, 2020, doi: 10.1155/2020/6622927.

[7] Y. Hao and Q. Gao, "Predicting the Trend of Stock

Market Index Using the Hybrid Neural Network Based on Multiple Time Scale Feature Learning," *Appl. Sci. 2020, Vol. 10, Page 3961*, vol. 10, no. 11, p. 3961, Jun. 2020, doi: 10.3390/APP10113961.

[8] Y. Huang, "Predicting home value in California, United States via machine learning modeling," *Stat. Optim. Inf. Comput.*, vol. 7, no. 1, pp. 66–74, 2019, doi: 10.19139/soic.v7i1.435.

[9] S. Pourmand, A. Shabbak, and M. Ganjali, "Feature Selection Based on Divergence Functions: A Comparative Classification Study," *Stat. Optim. Inf. Comput.*, vol. 9, no. 3, pp. 587–606, 2021, doi: 10.19139/soic-2310-5070-1092.

[10] M. Kheirkhahzadeh and M. Analoui, "Community detection in social networks using consensus clustering," *Stat. Optim. Inf. Comput.*, vol. 7, no. 4, pp. 864–884, 2019, doi: 10.19139/soic-2310-5070-801.

[11] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *Eur. J. Oper. Res.*, 2017, Accessed: Oct. 14, 2021. [Online]. Available: https://ideas.repec.org/p/zbw/iwqwdp/112017.html.

[12] P. Zhong and Z. Gong, "A hybrid DBN and CRF model for spectral-spatial classification of hyperspectral images," *Stat. Optim. Inf. Comput.*, vol. 5, no. 2, pp. 75–98, 2017, doi: 10.19139/soic.v5i2.309.

[13] F. Abdullayeva and Y. Imamverdiyev, "Development of oil production forecasting method based on deep learning," *Stat. Optim. Inf. Comput.*, vol. 7, no. 4, pp. 826–839, 2019, doi: 10.19139/soic-2310-5070-651.

[14] M. Faraz, H. Khaloozadeh, and M. Abbasi, "Stock Market Prediction-by-Prediction Based on Autoencoder Long Short-Term Memory Networks," *2020 28th Iran. Conf. Electr. Eng. ICEE 2020*, Aug. 2020, doi: 10.1109/ICEE50131.2020.9261055.

[15] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLoS One*, vol. 12, no. 7, p. e0180944, Jul. 2017, doi: 10.1371/JOURNAL.PONE.0180944.

[16] S. Lv, Y. Hou, and H. Zhou, "Financial Market Directional Forecasting With Stacked Denoising Autoencoder," Dec. 2019, Accessed: Oct. 14, 2021. [Online]. Available: https://arxiv.org/abs/1912.00712v1.

[17] W. Chen, M. Jiang, W. G. Zhang, and Z. Chen, "A novel graph convolutional feature based convolutional neural network for stock trend prediction," *Inf. Sci. (Ny).*, vol. 556, pp. 67–94, May 2021, doi: 10.1016/J.INS.2020.12.068.

[18] J. Long, Z. Chen, W. He, T. Wu, and J. Ren, "An integrated framework of deep learning and knowledge graph for prediction of stock price trend: An application in Chinese stock exchange market," *Appl. Soft Comput. J.*, vol. 91, Jun. 2020, doi: 10.1016/J.ASOC.2020.106205.

[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/NECO.1997.9.8.1735.

[20] S. JIN, L. SU, and A. ULLAH, "Robustify Financial Time Series Forecasting with Bagging," *Econom. Rev.*, vol. 33, no. 5–6, p. 575, Aug. 2014, doi: 10.1080/07474938.2013.825142.

[21] H. M. Pandey, "Performance Evaluation of Selection Methods of Genetic Algorithm and Network Security Concerns," Procedia Comput. Sci., vol. 78, pp. 13–18, Jan. 2016, doi: 10.1016/J.PROCS.2016.02.004

[22] Dhabliya, D., & Dhabliya, R. (2019). Key characteristics and components of cloud computing. International Journal of Control and Automation,12(6 Special Issue), 12-18. Retrieved from www.scopus.com

[23] Pekka Koskinen, Pieter van der Meer, Michael Steiner, Thomas Keller, Marco Bianchi. Automated Feedback Systems for Programming Assignments using Machine Learning. Kuwait Journal of Machine Learning, 2(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/190

[24] Carlos Silva, David Cohen, Takashi Yamamoto, Maria Petrova, Ana Costa. Ethical Considerations in Machine Learning Applications for Education. Kuwait Journal of Machine Learning, 2(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/192