Neural Computing and Applications (2023) 35:11403–11416
https://doi.org/10.1007/s00521-023-08305-6

ORIGINAL ARTICLE

Using neural-genetic hybrid systems for complex decision support

Pi-Sheng Deng[1] · Tzu-Man Huang[2]

Received: 9 May 2022 / Accepted: 16 January 2023 / Published online: 3 February 2023

Abstract
We propose a hybrid system for supporting complex decisions through integrating neural networks (NNs) and genetic algorithms (GAs). We investigate the feasibility of leveraging the synergistic effect of integrating NNs and GAs to support stock market investment decisions. Utilizing 10-year daily US stock market data, we identified a set of effective attributes to predict stock market. The results suggest that our system is capable of exhibiting learning behavior and is a promising tool for stock market prediction. Though NNs have been successfully applied to a variety of pattern recognition applications, the connection weights generation process is highly computationally demanding. We apply GAs to search stochastically for connection weights. This alleviates an NN's lengthy training problem so that our hybrid system is more applicable to support complex decision making. Another contribution of our research is parameter setting for GAs. Parameter setting is a long-time thorny issue for GA implementation. We focus on one of the most difficult issues—the setting for the mutation rate. Using the stock market prediction as an application area, we have helped shed light on the role and importance of the mutation rate, as well as the complementary effect of mutation and crossover functions. Our findings favor "low-mutation-rates."

Keywords Exploitation-and-exploration · Genetic operators · NN-GA Hybrid system · Stochastic search · Fundamental analysis · Technical analysis

1 Introduction

One of the recent developments in artificial intelligence (AI) is the innovative employment of different AI techniques in a hybrid manner. It gave rise to an emerging computational approach called soft computing, including fuzzy logic, neural networks (NNs), genetic algorithms (GAs), simulated annealing, and chaotic systems [1]. These techniques are resilient to errors, imprecision, fuzziness, and uncertainty, while traditional techniques find accurate and optimal solutions to problems. The synergistic integration of these computing paradigms empowers soft computing-based hybrid systems to adapt to changing environment for better performance.

For most of the multi-dimensional decisions, we can only approximate the optimal solutions. Optimal solutions are usually unachievable due to multiple local optima in a complex solution space [2]. Researchers have been experimenting with soft computing as a promising tool for this problem by leveraging the exploration-and-exploitation capability of soft computing [2].

In this research we propose a system for supporting complex decisions through the synergistic integration of neural networks and genetic algorithms. Neural networks have been successfully applied to a variety of pattern recognition-related applications. In many practical applications, the most difficult part of neural networks is the highly computation-demanding process for computing connection weights. We apply GAs to stochastically search for connection weights. This would alleviate a neural network's lengthy training problem so that the NN-GA hybrid system would be more applicable to support complex decision making.

✉ Pi-Sheng Deng
pdeng@csustan.edu

Tzu-Man Huang
thuang@csustan.edu

[1] Department of Management Information Systems, College of Business Administration, California State University, Stanislaus, Turlock, CA 95382, USA

[2] Department of Accounting and Finance, College of Business Administration, California State University, Stanislaus, Turlock, CA 95382, USA

We investigate the feasibility of leveraging the synergistic effect of a NN-GA system to support stock market investment decisions. Stock market is such a dynamic and volatile system that no model can predict its highly complicated behavior perfectly. Utilizing 10-year daily US stock market data, we identified a set of effective attributes for our model to predict stock market. Our empirical results suggest that our hybrid system serves as a promising tool for stock market prediction. In addition, our research results show that our system is capable of exhibiting learning behavior, i.e., improving performance as the solution searching process develops. Another major contribution of our research is that we help shed light on an open issue about GA parameter settings, especially for the mutation rate.

Next section describes the concept, operations, and characteristics of GAs and NNs. We present our hybrid system's conceptual model in section three and the structural design of our system in section four. Design implementation is discussed in section five. Section six is the application of our system for predicting stock market movement. We also present the result of ANOVA and performance analysis in this section. Section seven concludes this research.

2 Methodology

Soft computing is characterized by its synergistic effect through innovative integration of two or more techniques together. Many studies have shown that synergistic integration is a promising approach to effective decision support [11].

2.1 Genetic algorithms

Genetic algorithms (GAs) are population genetics-inspired search algorithms which improves solutions gradually based on previous solutions [3–5]. Unlike traditional optimization tools, which strive to find the optimal solutions, GAs are most suitable for finding near-optimal solutions. The solution space consists of syntactic patterns of solutions [5]. A solution space consists of a set of patterns of genes, and each patter represents an area in the solution space [10, 11]. Genetic operators are applied to the most promising solutions to exploit the solution space and explore new regions in the solution space [10, 11]. According to Goldberg [3], the operational cycle of GAs consists of the following stages:

- Generating solution population;
- Defining objective function;
- Evaluating each solution;
- Selecting two solutions for regeneration by applying crossover operator and mutation operator;
- New solutions will be formed and consequently replace existing solutions.

This process will repeat until the termination conditions are met or until the decision maker terminates the search process. When searching for solutions in the solution space, GA operators interact with each other and enable GAs to explore and exploit different regions in the solution space concurrently [6, 7]. Through the interaction of GA operators, GAs are capable of exploring and exploiting the solution space for better solutions [6, 7]. This capability allows GAs to avoid the local optima problem through mutating gene values stochastically [6, 7].

Some studies asserted that mutation was not as effective as crossover in the solution search process [3, 8]. However, Muhlenbein [9] experimented a set of non-derivative functions and achieved an opposite conclusion about the effectiveness of mutation. Deng [10] investigated a different application domain than Muhlenbein [9] by implementing a GA-based flexible manufacturing system for the batch selection problem and achieved the similar result regarding the importance of mutation. To investigate whether the similar conclusion about GA parameter settings also holds for different computational paradigms, Deng & Tsacle [11] coupled a rule-based system and a GA for the same batch selection problem of Deng [10]. Their result showed the same conclusion regarding the role of mutation in GA implementation [11]. The result was also similar to that of Muhlenbein [9].

In this study, we changed the computational paradigm further by implementing a GA-driven NN hybrid system for stock market prediction, a more challenging application domain than those of Deng, Deng & Tsacle [10, 11]. Following the parameter settings suggested by Muhlenbein, Deng, Deng & Tsacle [9–11] for a new hybrid system, our study reached the similar conclusion to the aforementioned studies and showed that the power of mutation seemed to have been underestimated. Thus, the parameter settings, especially for the mutation rate, as proposed by Muhlenbein [9], seem to hold for different application domains and different computational paradigms as well.

2.2 Neural networks

A neural network (NN), a different computational paradigm than that of GAs, is a gradient-based optimization technique, which is good at "digging into" an applied solution region for the best solution. NNs are one of the most effective learning algorithms for approximating complex functions. In addition, NNs can explore multiple promising areas in the solution space concurrently. NNs

consist of a large number of interconnected neurons. Each neuron consists of a summation function and a nonlinear transformation function. A neuron is connected to other neurons by links with adjustable strengths. The distribution of weights on the entire network represents a knowledge structure.

Structurally, an NN is a composite function consisting of layers of computational nodes, including one input layer, several hidden or intermediate layers, and one output layer. The major function of the hidden layers is to develop useful intermediate representation for capturing major characteristics of the input instances. The output layer would use this internal representation to compute the target output. This ability to automatically discover useful representations at the hidden layers enables multilayer NNs to make generalization or draw inferences based on incomplete information.

The development of intermediate representation by multilayer NNs is achieved through the application of learning algorithms to search for optimal weights in a huge multi-dimensional hypothesis space. Learning algorithms equip NNs with adaptability through adjusting neural interaction weights by repeatedly applying gradient descent to search the solution space of weights in finding the weights that would yield minimum prediction error. However, the gradient descent-based learning process is usually highly computationally demanding which results in slow process in converging asymptotically to a final solution. In addition, for a complex solution space with multiple local minima, this gradient descent-based learning process is usually trapped in some local minimum and cannot converge to the global minimum [12].

In order to alleviate the problem of local minima, we apply GAs to learn the near-optimal weights for NNs. Since GAs are stochastic explore-and-exploit search algorithms, they are more likely to escape from local minima [5, 7]. In this research, we harness the stochastic search power of GAs in searching for a near-optimal set of connection weights with the purpose to both speed up the learning process and increase the classification effectiveness for our system. Through the integration of GAs and NNs techniques, our genetic-neural hybrid system is able to support decision makers with the adaptation and learning capabilities for effective decision making.

3 A NN-GA hybrid system for stock market prediction

To demonstrate the applicability of our system, we design a neural-genetic hybrid model for stock market prediction. Though efficient market hypothesis states that stock prices behave as random walks [13] and are unpredictable, there have been many empirical studies showing the opposite [14].

In fact, stock prices are influenced by numerous factors. Traditionally, financial professionals use fundamental and technical analysis to predict stock returns. Fundamental analysis involves the utilization of firm financial information, as well as industry outlook and broad economic conditions. Technical analysis involves the utilization of historical trading statistics, such as charting techniques [15], the expected return factor model [16], and even artificial intelligence techniques [17].

In this research, we utilize a neural-genetic hybrid model to predict stock market movement. Generally, the prediction period could range from daily [18–24], monthly [25, 26], to annual [27] predictions. As the stock market changes rapidly, we predict daily stock market movement in this study.

The predicting variables for stock market could include macroeconomic factors, financial information, and technical indicators. The macroeconomic factors usually include interest rates, inflation, unemployment rates, money supply, oil and commodity prices, exchange rates, foreign stock markets, and many more [24, 26–28]. Financial information includes firm specific financial statements and ratios, as well as the overall industry outlook [25, 28]. Technical indicators include various trading statistics, such as open, close, the highest, the lowest, price change, trading volume, simple moving average, exponential moving average, momentum, relative-strength index, on-balance-volume, moving average convergence/divergence, the band system, and many more [18, 20–23, 26, 29].

As we predict daily stock market movement, we have to use daily variables. Most financial information under fundamental analysis is not daily variables and thus not included in this study. We focus on technical indicators and available daily macroeconomic variables. We use the S&P 500 Index as the proxy for the US stock market. The daily variables we generated include open price (Open), close price (Close), net change of the price (Net), percentage change (Chang), the lowest price (Low), the highest price (High), trading volume (Volume), five-day simple moving average (MA5D), 20-day simple moving average (MA20D), 21-day exponential moving average (EMA21D), 14-day relative-strength index (RSI14D), moving average convergence/divergence (MACD), as well as the one-year Treasury Security interest rate (INT1YR). We also include European Stock Market Index (STXE600) and Euro currency exchange rate (EUR) to incorporate international considerations.

We collected a data set of 10 years daily data from January 2010 to December 2019, consisting of 2,497 records, from Datastream. The collected data set consists of the following daily attributes for the S&P 500 Index: Open,

Close, Net, Chang, Low, High, Volume, MA5D, MA20D, EMA21D, RSI14D, MACD, as well as the one-year interest rate (INT1YR), European Stock Market Index (STXE600), and Euro currency exchange rate (EUR).

In many cases, the accuracy of prediction can be improved by inventing more appropriate features to describe the available data [30]. Thus, we also derive attributes from charting techniques and meaningful combinations to form new input factors. We found the range of the highest index values in the past one month (HH), the range of the lowest index values in the past one month (LL), the five-day exponential moving average (EMA5D), European Stock Market Index daily return (STXE600R), S&P 500 Index daily return (SP500R), and one-day lag S&P 500 Index daily return (SP500RL1) relatively predictive and included them in the analysis. On the other hand, the one-year interest rate (INT1YR), European Stock Market Index (STXE600), and the Euro currency exchange rate (EUR) did not have predicting power and therefore were dropped from the model. In addition, MA20D, EMA21D, and MACD were also dropped for the same reason.

Finally, we established our model based on the following attributes: Open, Close, Net, Chang, Low, High, Volume, MA5D, EMA5D, RSI14D, HH, LL, STXE600R, SP500R, and SP500RL1. Open next day is our target variable, while the rest are predicting variables.

4 Structural design of an NN-GA hybrid model

When designing a hybrid system, we need a technique for integrating its subsystems. There are three techniques we can consider: loose coupling, tight coupling, and fully integrated [17, 31]. Among these three techniques, loose coupling is the easiest one to implement. Through a common data base, loose coupling connects different modules through sharing the content of the common data base [31]. This allows easy and flexible maintenance of the integrity of module structure. In addition, loosely coupled systems are more reliable than tightly coupled systems [17].

Tight coupling model requires structural compatibility between the interacting modules [31]. This restriction leads to structural inflexibility and functional rigidity for the interacting modules. In a full integration environment, each module interacts and affects the working of any other module potentially [31]. The complexity of interaction grows exponentially as the number of modules grows.

With the consideration of flexibility and simplicity, we choose loose coupling for our GA-based NN system. Our NN-GA hybrid model is a three-layer feedforward network structure with one input layer, one hidden layer, and one output layer. Each node in each layer is connected to every node in the next layer. Each input record is passed through this three-layer network in generating the corresponding output. Structurally, an NN with two or more hidden layers is called a deep learning NN [32] or deep neural network [33], and it has better function approximation capability, if given long enough training time. The additional hidden layers have the effect of causing an NN to move a smaller step each time in the solution space. In other words, the system becomes more "cautious" moving in the solution space, and thus causes the system to converge very slowly.

Instead of using the traditional backpropagation algorithm for fine-tuning the connection weights, we apply genetic algorithms to identify the near-optimal connection weights. GAs are well known by their parallel search capability [7]. This characteristic increases the chance for GAs to discover optimal solutions. However, due to the stochastic search process, the solutions-finding process tends to be very slow, and solutions found are usually near-optimal, especially for highly complex decisions [2, 7]. The stochasticism also leads to oscillation of the search process and makes it difficult to fine-tune genetic operators during the solution search process [11].

5 Design implementation

We collected a 10-years daily data set, with 12 attributes associated with the S&P 500 Index, one macroeconomic variable, and two international considerations. We found new attributes within the data with predicting power; we also removed some attributes that do not have predicting power. At the end, we established our model based on 14 predictive attributes.

The structure of our NN-GA hybrid model includes a three-layer NN module: one input layer, one hidden layer, and one output layer. Each neuron in the hidden layer and the output layer consists of one adder operator and one Tanh activation function. Output from the output layer is entered into the Trend Detector module to be compared with the output of the day before. The Adder module would calculate the total number of correct predictions which would be sent to the GA module for maximization.

(1) The input layer is a large database $\mathbf{D} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \ldots \quad \mathbf{x}_{|\mathbf{D}|}]^T$, where each record $\mathbf{x}_d$ consists of $|\mathbf{K}|$ attributes:$\mathbf{x}_d = [x_{d1} \quad x_{d2} \quad \ldots \quad x_{d|\mathbf{K}|}]$, where $d = 1, 2, \ldots, |\mathbf{D}|$. Let $\mathbf{K}$ be the set of attributes, and $|\mathbf{K}|$ is the number of attributes. Since our input variables are not based on the same numerical scale and extreme values might introduce biases, we normalize or standardize our input data. The input value x_{dk} of attribute k of

record **d** is transformed into a z-value, z_{dk}, of range [− 1,1] by

$$z_k = 2\left(\frac{x_k - \text{Min}_k}{\text{Max}_k - \text{Min}_k}\right) - 1, \text{ where } k = 1, 2, ..., |\mathbf{K}|.$$

After the z-score standardization, each record $\mathbf{x}_d$ is converted into a set of standardized values $\mathbf{z}_d = [z_{d1} \quad z_{d2} \quad \ldots \quad z_{d|\mathbf{K}|}]$. In an NN, each neuron is characterized by two functions: one is a summation function, and the other is a nonlinear activation function as shown in Fig. 1.

(2) For each neuron h on the hidden layer, there is a connection weight between attribute k and neuron h, $w_{kh} = [w_{1h} \quad w_{2h} \quad \ldots \quad w_{|\mathbf{K}|h}]$, for h = 1, 2, …, |**H**|. For the dth record, the summation function of neuron h multiplies normalized inputs z_k by weights w_{kh} and summed up together with a constant bias $w_{h,}$ as:

$$\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h$$

To allow our system to develop estimation for nonlinear functions, each neuron is equipped with a nonlinear parameterized function Θ as the activation function of a neuron. Usually, Θ is chosen to be a sigmoidal function or hyperbolic tangent (tanh) to transform the weighted sum of input data to the hidden layer nonlinearly into the range [–1,1]. Sigmoidal or hyperbolic tangent normalization has been shown to be especially appropriate for including very large outliers without overly compressing the real range of data values [37]. We let Θ be a hyperbolic tangent as

$$\tanh(\alpha) = \frac{1 - e^{-\alpha}}{1 + e^{-\alpha}},$$

where α is the parameter representing the weighted sum of input.

Thus, the output of neuron h in the hidden layer is:

$$y_h = \Theta\left(\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h\right) = \tanh\left(\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h\right) = 1 - e^{-\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h} \Big/ 1 + e^{-\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h}$$

For each neuron o in the output layer, where o = 1, 2, …, |**O**|, its summation function computes weighted sum $\sum_{h=1}^{|\mathbf{H}|} w_{ho} y_h + w_o$ as input to the activation function Θ which will, in turn, transform this weighted sum into the output as generated from this output node o:

$$y_o = \Theta\left(\sum_{h=1}^{|\mathbf{H}|} w_{ho} y_h + w_o\right) = \Theta\left(\sum_{h=1}^{|\mathbf{H}|} w_{ho}\left(\Theta\left(\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h\right)\right) + w_o\right)$$

This output y_o is the prediction of the target variable, which is the stock market price in our case.

(3) Another design consideration is the number of layers in the network. Cybenko [34] and Hornik et al. [35] have shown that every bounded continuous function can be approximated with arbitrarily small error by an NN with three layers of neurons. As mentioned earlier, number of hidden layers not only affects the function approximation accuracy of an NN, but also causes an NN to converge very slowly in the solution space [32, 33]. We balance the considerations of training efficiency and prediction effectiveness for our network design. As a result, our NN-GA model is

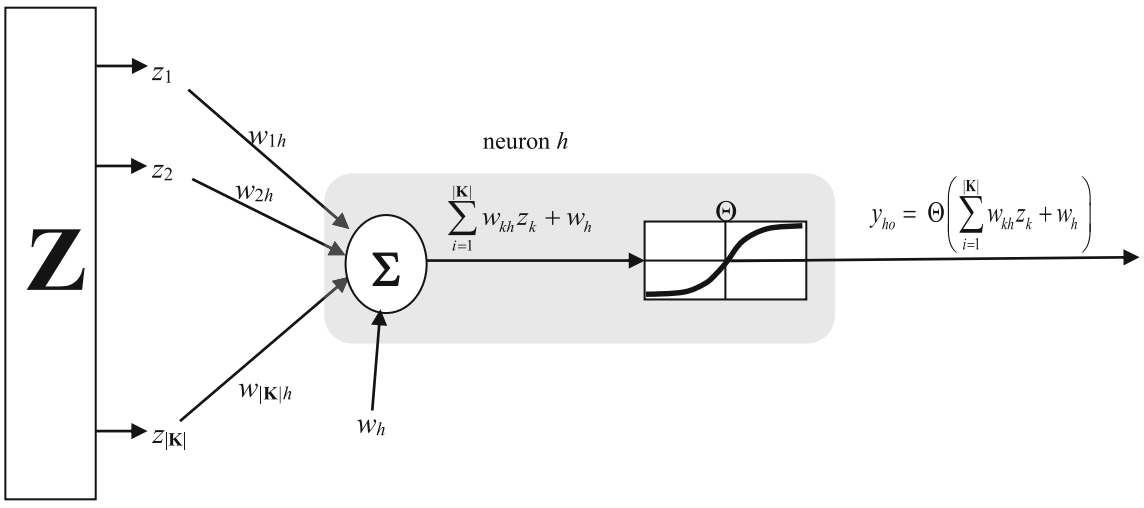

Fig. 1 Structure of a neuron

a three-layer feedforward network structure as shown in Fig. 2.

According to Mitchell [12] and Brownlee [36], using too many nodes on the hidden layer often increases the tendency of the network to overfit the training data, and thus reduces generalization accuracy. A good rule of thumb is to use a number of hidden nodes on the order of the square root of the number of training examples or lower [37].

(4) The task for our hybrid system is to predict stock market one day ahead. The training data set **D** consists of |**D**| days of stock market movement. For the dth day, the predicted output generated by the activation function of the oth neuron in the output layer is $y_{do}^{predict}$. Since our model has only one neuron in the output layer, we represent this prediction as $y_d^{predict}$. The Comparator function compares prediction for the dth day with the prediction for the day before, $y_{d-1}^{predict}$. The stock market movement direction from one day to the next is denoted by the result of applying the difference operator (Δ) to y_d and y_{d-1}. Note that we are predicting the market movement direction, $y_d - y_{d-1}$, instead of the stock price. When the predicted market movement direction is the same for the actual market movement direction, this is a correct prediction. On the other hand, if the predicted market movement trend is not the same as the actual market movement trend, it is a wrong prediction. We apply the multiplier operator (Π) to $y_d^{target} - y_{d-1}^{target}$ and $y_d^{predict} - y_{d-1}^{predict}$, and let the result $(y_d^{target} - y_{d-1}^{target})(y_d^{predict} - y_{d-1}^{predict})$ be the input to the Heaviside step function (H):

$$H = \begin{cases} 1, & (y_d^{target} - y_{d-1}^{target})(y_d^{predict} - y_{d-1}^{predict}) \geq 0 \\ 0, & (y_d^{target} - y_{d-1}^{target})(y_d^{predict} - y_{d-1}^{predict}) < 0 \end{cases}$$

Through summation of the H function output for the entire training set **D**, we obtain the frequency of accuracy (i.e., rate of accuracy) of our model in predicting the stock market movement trend. Our objective is to maximize

$$\sum_{d=1}^{|\mathbf{D}|} \mathrm{H} = \sum_{d=1}^{|\mathbf{D}|} (y_d^{target} - y_{d-1}^{target})(y_d^{predict} - y_{d-1}^{predict}).$$

Note that y_d^{target}, y_{d-1}^{target}, and $y_{d-1}^{predict}$ are not variables, for they are obtainable from the training set. Thus, Max $\sum_{d=1}^{|\mathbf{D}|} H$ can be approached, without loss of generality, by

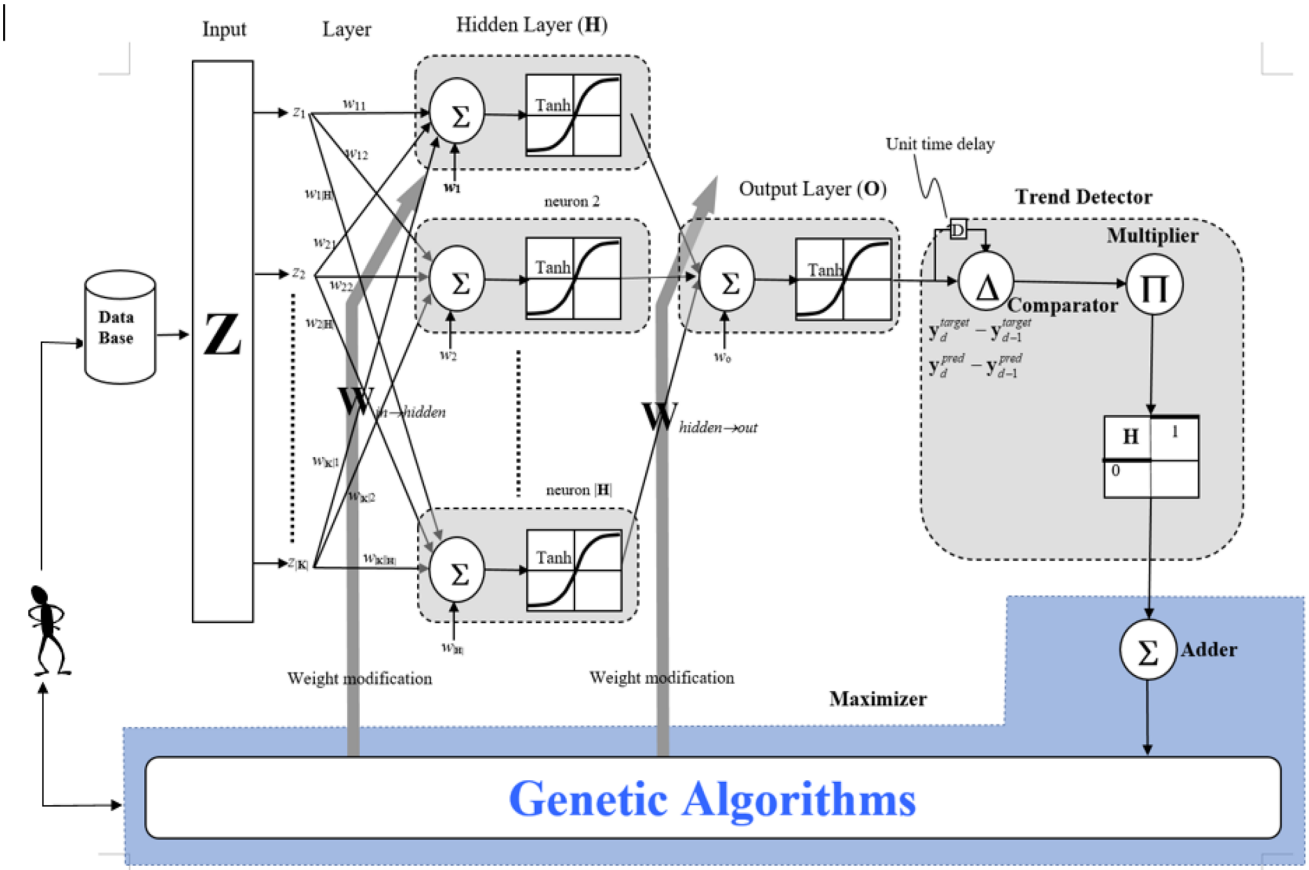

Fig. 2 Structure of our genetic-neural hybrid system

Max $\sum_{d=1}^{|\mathbf{D}|} y_d^{\text{predict}}$. Note that y_d, when the number of neurons $|\mathbf{O}| \geq 2$, is denoted as y_{do}. Thus,

$$\begin{aligned}\text{Max} \sum_{d=1}^{|\mathbf{D}|} \text{H} &= \text{Max} \sum_{d=1}^{|\mathbf{D}|} y_d^{\text{predict}} \\ &= \text{Max} \sum_{d=1}^{|\mathbf{D}|} \Theta\left(\sum_{h=1}^{|\mathbf{H}|} w_{ho}\left(\Theta\left(\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h\right)\right) + w_o\right) \\ &= \text{Max} \sum_{d=1}^{|\mathbf{D}|} \text{Tanh}\left(\sum_{h=1}^{|\mathbf{H}|} w_{ho}\left(\text{Tanh}\left(\sum_{k=1}^{|\mathbf{K}|} w_{kh} z_k + w_h\right)\right) + w_o\right).\end{aligned}$$

This maximization is based on the adjustment of the entire set of weight of connections, W, for the neural net. This weight adjustment would be performed by the GA module for learning the most effective connection weights to maximize the total prediction accuracy for movement direction. The learning problem faced by the GA module is to search a huge hypothesis space, with dimension $|\mathbf{K}| \times |\mathbf{H}| \times |\mathbf{O}|$, defined by all possible connection weight values for all the nodes in the network. The objective function for our GA module is to maximize the total sum of H values. This whole process is conceptually shown in Fig. 2.

(5) Performance of a GA depends upon many factors. In our hybrid system, the GA module maximizes the objective function through modifying the connection weights between layers, including the input layer to hidden layer, hidden layer to next hidden layer, and hidden layer to output layer.

In this study we treat the GA module as the maximizer or minimizer of a constrained optimization problem. Searching for optimal solutions for a complex decision is an NP-complete task, which also applies to parameter settings for genetic operators [6]. Research has not found agreed conclusions on population size yet. While many studies [3, 10, 11, 38, 39] advocated using population of medium- to large-sized population, Grefenstette, Schaffer et al. [4, 40] suggested the use of a very small-sized population. A large population suffers from a lengthy search process. In the meanwhile, a small population might cause insufficient evaluations of sampled solutions before moving onto other promising solution areas. For our study, we adopt the suggestions by Mitchell, Deng, Deng & Tsacle [6, 10, 11] and set the population size at 100.

Another operator is crossover. This operator is applied to a pair of good solutions in the current generation of population for generating new solutions to replace two low performance solutions. As generations passed by, the average performance of the population of solutions gradually improves, suggesting that GAs are capable of learning. This can be attributed to the crossover operator which exchanges part of the good solution strings with another good solution string. According to Mitchell [6], crossover is one of the most effective operators of GAs, and it usually interacts with other operators, selection strategies, difficulty level of applications, etc. Former research findings suggest the use of a medium to high value for the crossover rate [4, 10, 38–40]. In this research, we follow Deng, Deng & Tsacle [10, 11] and set the crossover rate at 0.9.

For new solution strings generated by the crossover operator in a new generation, the mutation operator will modify each gene probabilistically. So far, there is still no definitive conclusion for the power of mutation yet. When the crossover rate is almost one, there will be a possibility of degenerating into a population consisting of identical solutions. Though there is no consensus regarding how effective mutation is during the solution-finding process, it is certain that mutation helps alleviate this problem [6, 7] via altering the gene value of a solution string only occasionally. Many studies show that mutation alone does not improve the search for a solution [3].

De Jong [39] experimented with five problems on function minimization and found the best mutation rate to be 0.001. However, the study of parameter optimization for GAs by Grefenstette [4] suggested a mutation rate of 0.01. Contrary to common practices, Croitoru [41] conducted experiments on numerical functions and bit-block functions by using very high mutation rates and found high mutation rates performing well on most test functions, even outperforming low-mutation rates on some of them. Still, according to Deng, Deng & Tsacle [10, 11], high mutation rate would result in much worse performance than medium rate or low rate would.

In this study, we followed the suggestion of Mitchell, Deng, Deng & Tsacle [6, 10, 11] and experimented with the mutation rate at three levels: 0.001 (Low), 0.01 (Medium), and 0.5 (High), while holding the crossover rate, population size, selection strategy, and termination condition constant, to compare their stock market prediction performance. We investigated the aforementioned three mutation rates, the learning rates, prediction accuracy, and the number of generations past before the prediction rate stabilized in one run of simulation. We also tested the significant difference among those three mutation rates. We showed that our hybrid system was capable of exhibiting learning behavior as it moved from generation to generation. When the best solution found did not improve for more than 100 generations, our simulation run would stop. With a population size of 100 per generation, there were 10,000 evaluations performed before the solution search process stopped. We performed the search process 40 times, due to time constraint, by setting the mutation rate at 0.001, 0.01, and 0.5, in order to obtain average performance.

6 Application to stock market prediction and performance comparisons

In order to identify a set of effective predictors for stock market movement, we conducted a sequence of experiments.

Stage One: preliminary analysis on original dataset

In Stage One, our input set was based on the attributes of the original data file, including Close, Net, Chang, Low, High, Volume, MA5D, MA20D, EMA21D, RSI14D, MACD, INT1YR, STXE600, and EUR. The purpose of this stage was to understand whether these attributes were predictive or not.

We conducted a preliminary analysis in order to identify a set of seed input variables. We perform simulation runs for each set of input variable combinations. We set the threshold at 80 percent accuracy and removed one at a time if the presence or absence of the variable did not affect much of the performance. We ended up removing MA20D, EMA21D, MACD, INT1YR, STXE600, and EUR from our input variables.

Stage Two: further analysis on input variables

Based on our preliminary analysis, the set of input variables that yields prediction accuracy rate above 80 percent includes Close, Net, Chang, Low, High, Volume, MA5D, and RSI14D. We tried to improve the predicting accuracy by deriving attributes from charting techniques and meaningful combinations to form new input factors. We found that the range of the highest index values in the past one month (HH), the range of the lowest index values in the past one month (LL), the 5-day exponential moving average (EMA5D), European Stock Market Index daily return (STXE600R), S&P 500 Index daily return (SP500R), and one-day lag S&P 500 Index daily return (SP500RL1) were relatively predictive factors and thus were included in the analysis.

Stage Three: identification of the most effective indicators for stock market prediction

We conducted analysis to identify the most effective predicting input variables for stock market prediction. Based on the simulation results, the best set of input variables included: Close, Net, Chang, Low, High, Volume, MA5D, RSI14D, HH, LL, EMA5D, STXE600R, SP500R, and SP500RL1. These were the input predictors to the NN module. The final data set of 2,497 records was split into two data sets, training dataset (**D**) and validation dataset (**TD**), by using the ratio roughly 70:30. Based on the training dataset, we conducted training for our connectionist hybrid system. We then applied the trained system to measure the prediction performance for both the training dataset and the validation dataset.

6.1 Stock market prediction

We investigated the prediction performance of our model for the training set at three levels of the mutation rate: 0.001, 0.01, and 0.5, while holding crossover rate at 0.9, population size at 100, and the termination condition as 100 generations of no improvement. Our experiment showed that our model was capable of exhibiting learning behavior for each of the mutation rates. We compared the predicted value with the actual target value. When the predicted value and the actual value move in the same direction, the prediction is considered accurate, and vice versa. The average prediction rate over generations for each of the mutation rates is shown in Fig. 3. Our experiment results suggested that mutation rate of 0.5 had the worst learning performance, which was the first one to slow down its pace of improvement, and its highest accuracy rate never exceeded 85 percent. On the other hand, the other two curves were quite similar to each other. It showed that mutation rate of 0.001 started to slow down its learning rate slightly earlier than the mutation rate of 0.01 and performed not as well as the mutation rate of 0.01 did in early generations (roughly the tenth generation to the fifty-fifth generation). However, it passed the curve of 0.01 mutation rate later and kept being the best thereafter. Our findings favor small mutation rates when all other parameters are held constant.

6.2 Performance comparisons for mutation rates

To gain deeper insight among different levels of mutation rates, we conducted simulation of 40 runs for each level of mutation rate investigated in this research. Each run of simulation usually required several hundred generations before a stable solution was found. Figure 4 documents the result of 40 simulation runs on training set for mutation rates of 0.001, 0.01, and 0.5, with the rest of parameters held constant.

According to Fig. 4, the mutation rate of 0.001 was the best setting. There were only two instances when the mutation rate of 0.01 was slightly better than the mutation rate of 0.001, and the mutation rate of 0.5 was worse than the other two cases, except only one time almost tied with the mutation rate of 0.01. Thus, in terms of the 40 runs of average prediction performance, we concluded that the mutation rate of 0.001 was the best setting for mutation rate, and the mutation rate of 0.01 the runner up, based on the training set.

Similarly, we applied our model to the validation set for prediction at the mutation rates of 0.001, 0.01, and 0.5. The

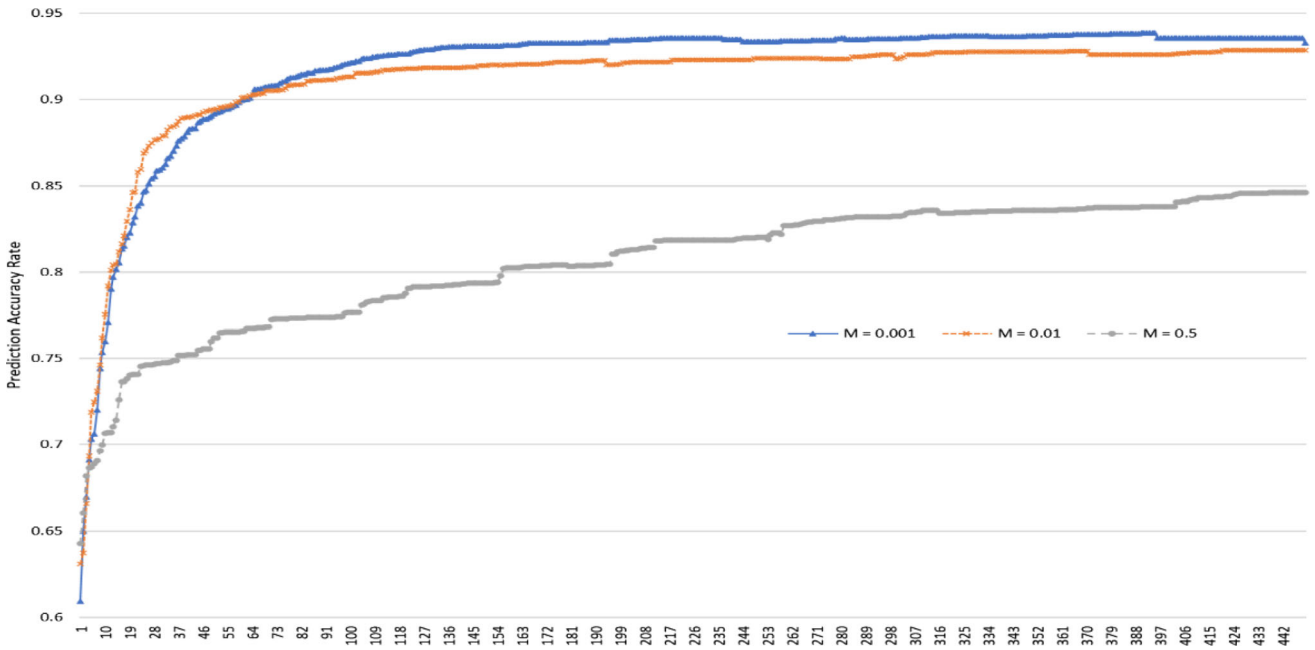

Fig. 3 Comparing the learning behavior at three levels of mutation

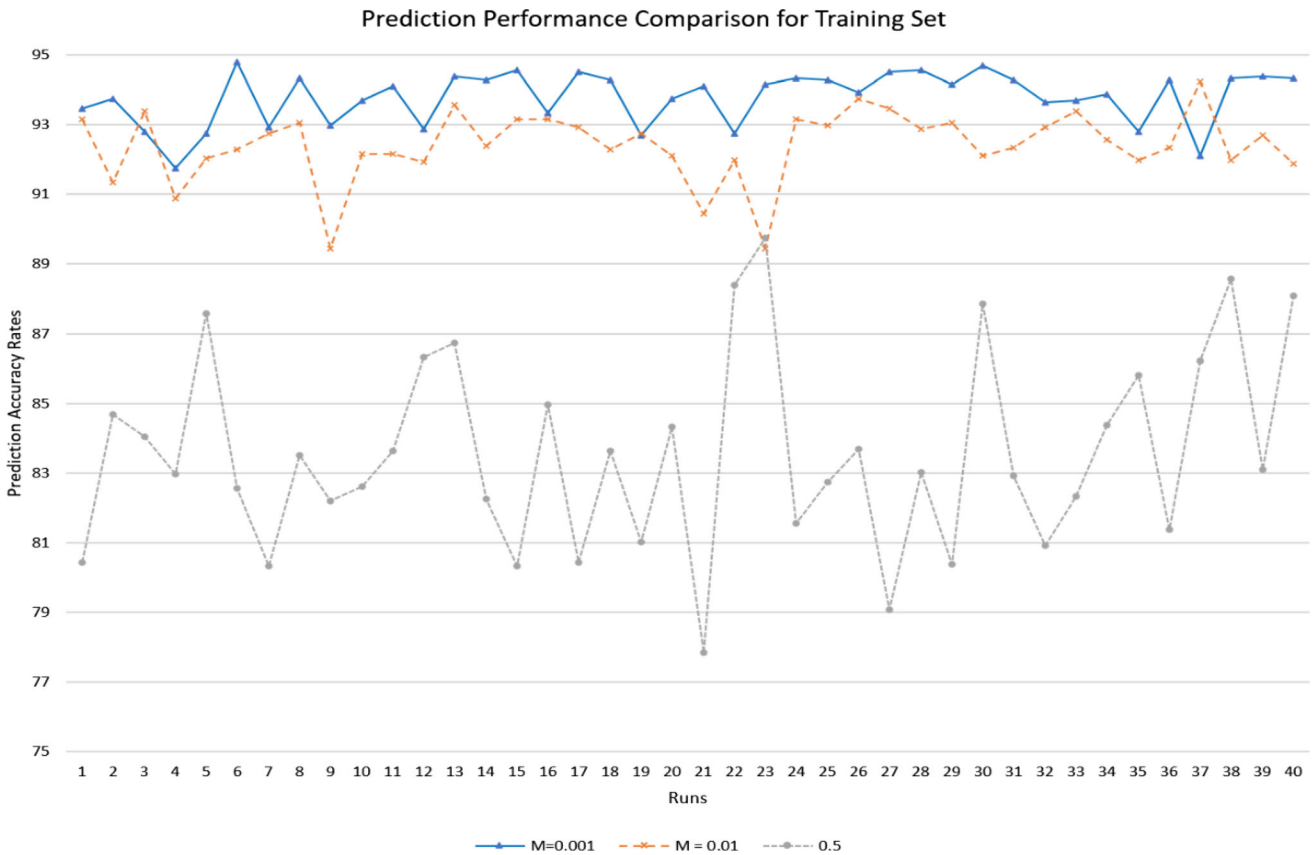

Fig. 4 Market prediction performance for the training set with three mutation rates of 0.001, 0.01, and 0.5

result is shown in Fig. 5. Since the validation set was not used for training our system, we expect lower prediction rates, compared to the results from the training set. The prediction accuracy for the three mutation rates was very close on the validation set. However, we could still recognize 0.001 being the best among the three rates investigated.

The data we collected were time dependent and had the characteristics of time sequentiality and continuity. Thus, we could not conduct random sampling to form training or validation datasets. Our training set consists of the first 1,697 records in their time sequence order, while our validation dataset has the most recent 800 records also in time sequence order. We trained our system by using the training data (**D**), and then applied it back to the training data (**D**) and the validation data (**TD**).

Table 1 shows the prediction performance results of 40 simulation runs for the training set at the mutation rates of 0.001, 0.01, and 0.5, with the same population size of 100 and the crossover rate of 0.9. We included in Table 1 the maximum, the minimum, the standard deviation, and the average prediction rate obtained based on the 40 runs of computation. From Table 1, it seemed mutation rate of 0.001 was the best choice.

Table 1 Stock market prediction for the training set (**D**) based on 40 runs of simulation (pop size = 100, crossover rate = 0.9)

Mutation rate	Accuracy rate (for training set).			
	Average (%)	Stdev (%)	Max. (%)	Min. (%)
0.001	93.78	0.76	94.81	91.75
0.01	92.41	1.01	94.23	89.45
0.5	83.28	2.88	89.75	77.31

Similarly, the prediction performance for the 40 runs of experiment for the validation set (**TD**) at the same three levels of mutation rate is presented in Table 2, with the same population size of 100 and the crossover rate of 0.9. From Table 2, difference among the three levels was smaller. However, the mutation rates of 0.001 and 0.01 seem to be slightly better than the mutation rate of 0.5.

Each simulation run usually required a large number of generations in finding the best solution before the termination condition was reached. We also showed the maximum, the minimum, and the average number of generations elapsed before a final solution was obtained. Our 40 runs of result for generations past is shown in Table 3. From Table 3, the mutation rate of 0.01 seemed to

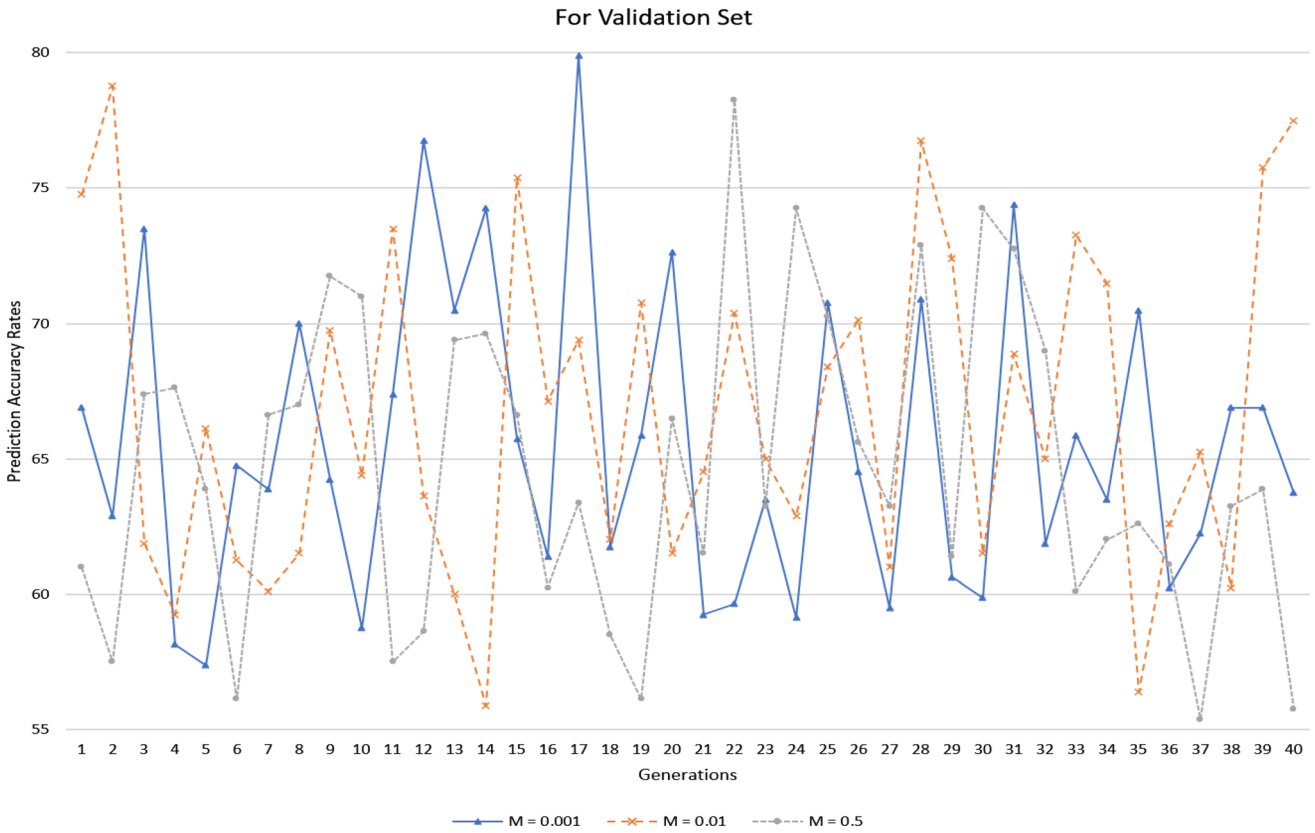

Fig. 5 Market prediction performance for the validation set with mutation rates of 0.001, 0.01, and 0.5

Table 2 Stock market prediction for the validation set (**TD**) based on 40 runs of simulation (pop size = 100, crossover rate = 0.9)

Mutation rate	Accuracy rate (for validation set)			
	Average (%)	Stdev (%)	Max. (%)	Min. (%)
0.001	65.50	5.58	79.88	57.38
0.01	66.65	6.10	78.75	55.88
0.5	64.09	5.57	74.25	55.00

be the best among the three levels. On average, the mutation rate of 0.01 took the smallest amount of generations in reaching a stable solution.

Figure 6 also shows that in most of the simulation runs, the mutation rate of 0.5 took much longer in finding a final solution than the other two cases which were quite competitive with each other.

When comparing the training (**D**) data accuracy rates and the validation (**TD**) data accuracy rates, though we expected inferior results on the validation data, a large difference in training and validation performance might be indicative of overfitting. To investigate whether there is any significant difference between training accuracy and validation accuracy, we conducted the t test to compare the difference of averages for these two data sets, at mutation rates of 0.001, 0.01, and 0.5, as follows.

$$t = \frac{\frac{\sum x_1}{N_1} - \frac{\sum x_2}{N_2}}{\sqrt{s_p^2\left(\frac{1}{N_1} + \frac{1}{N_2}\right)}}, \text{ where}$$

$$\text{Pooled Sample Variance} = s_p^2 = \frac{\sum x_1^2 - \frac{\left(\sum x_1\right)^2}{N_1} + \sum x_2^2 - \frac{\left(\sum x_2\right)^2}{N_2}}{(N_1 - 1) + (N_2 - 1)}$$

Based on the model, we had t value of 31.71 for the mutation rate of 0.001, t value of 28.91 for the mutation rate of 0.01, and t value of 25.95 for the mutation rate of 0.5. Since each t is higher than $t_{0.9995(40)}$ of 3.55, we concluded that the difference of average accuracy rates of the training set and those of validation set reached the 99.5 percent significance level. We had enough evidence to reject the null hypothesis that the accuracy rates are the same between the training and validating data sets (H_0: $\mu_{\text{train}} - \mu_{\text{valid}} = 0$). This implies that there is overfitting in training data set at each mutation rate we studied.

Table 3 Number of generations past before a stable solution is found based on 40 runs of simulation (pop size = 100, crossover rate = 0.9)

Mutation rate	Number of generations			
	Average	Stdev	Max	Min
0.001	322	80	585	194
0.01	295	110	560	101
0.5	378	181	808	157

We also performed ANOVA hypothesis testing on the prediction accuracy among the three mutation rates from the training set (H_0: $\mu_{0.001} = \mu_{0.01} = \mu_{0.5}$). Shown in Table 4, we have F value of 393.98, which is higher than $F_{0.995(2,120)}$ of 5.54. Thus, we cannot accept H_0. This implies that at least one pair of mutation rates has significant difference.

We also prepared Table 5 for testing the prediction accuracy on the validation set (H_0: $\mu_{0.001} = \mu_{0.01} = \mu_{0.5}$). From Table 5, F value was 1.97 and it is lower than $F_{0.95(2,120)}$ of 3.07. Thus, we did not have enough evidence to reject H_0. For the validation set, different mutation rates had no significantly different influences on the prediction accuracy.

In addition, we tested the hypothesis about the equivalence of the time taken before a stabilized final solution was found for the three levels of mutation rates. The result is shown in Table 6. The F-value achieved the 97.5 percent significance level, and thus H_0 should be rejected.

We investigated the mutation rates of 0.01, 0.1, and 0.5. Our experiment results suggested that the lowest mutation rate had the best predicting result on the training dataset and the difference among different mutation rates is statistically significant. However, such superior performance is not statistically significant on the validating dataset. Our results also showed that the predicting accuracy is different from the training dataset to the validating dataset. We also investigated the number of generations needed in finding the best solution before the termination condition was reached. The lowest mutation rate did not necessarily have the highest number of generations needed. In fact, the highest mutation rate of 0.5 had the highest number of generations needed in our simulation. When we consider the predicting accuracy and simulation efficiency, our experiment results favor low-mutation rates.

6.3 Prediction performance

After identifying the most effective mutation rate, we predicted the stock market value utilizing the 14 predictive variables in our model. The training dataset and the validating dataset remained the same ratio of roughly 70:30. We compared the predicted value to the actual value in the validating dataset by examining the difference of the predicted and the actual value. We then divided the difference by the actual value to get to a percentage deviation. If the predicted value is higher than the actual value, the

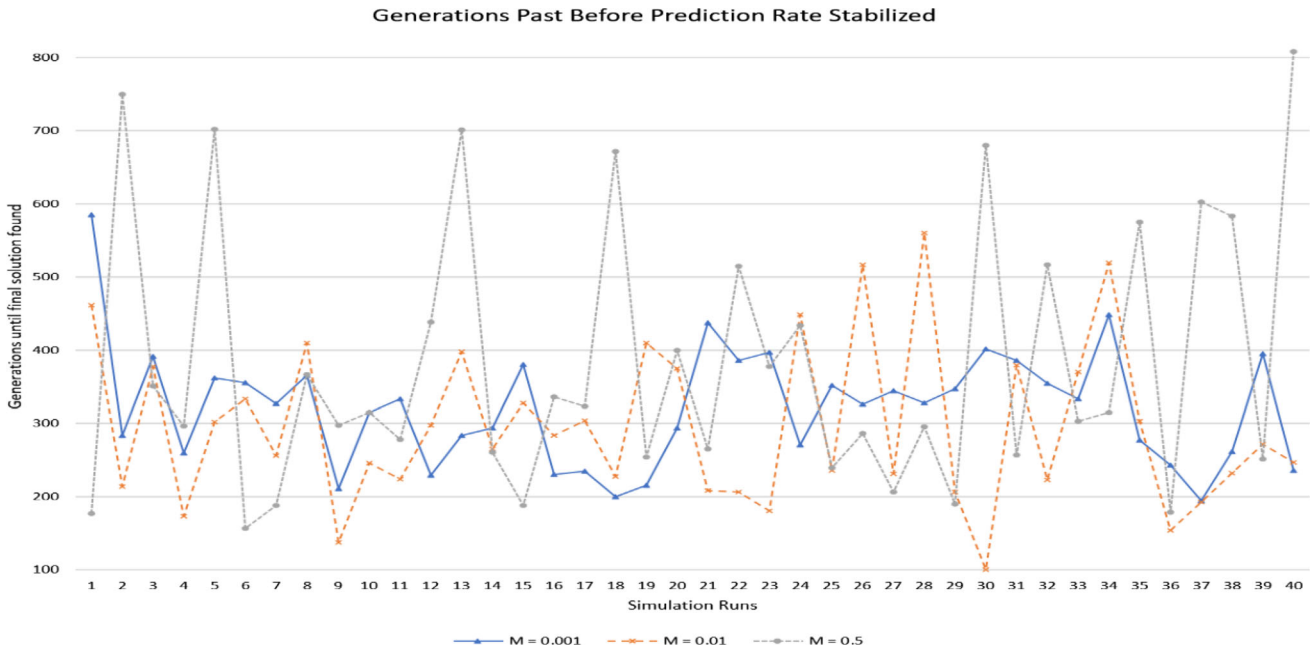

Fig. 6 Number of generations past before final solutions are found

Table 4 ANOVA for testing the equivalence of average accuracy rate for the training set (mutation rates of 0.001, 0.01, and 0.5)

Source	SS	df	MS	*F*
Between	2605.82	2	1302.91	393.98
Within	386.91	117	3.30	
Total		2992.74	119	
			$F0.995(2,120) = 5.54$	

Table 5 ANOVA for testing the equivalence of average accuracy rate for the validation set (mutation rates of 0.001, 0.01, and 0.5)

Source	SS	df	MS	*F*
Between	131.12	2	65.56	1.97
Within	3882.15	117	33.18	
Total		4013.27	119	
			$F0.95(2,120) = 3.07$	

Table 6 ANOVA for testing the equivalence of average number of generations past before a stable final prediction rate was found

Source	SS	df	MS	*F*
Between	143,441	2	71,720.72	4.17
Within	2,010,622	117	17,184.80	
Total		2,154,063	119	
	$F_{0.975(2,120)} = 3.80$		$F_{0.99(2,120)} = 4.79$	

deviation would be positive. On the other hand, if the predicted value is lower than the actual value, the deviation would be negative. To evaluate the predicting result, we took the absolute values of the deviations and presented the average, the standard deviation, the maximum, and the minimum of the predicting deviations, shown in Table 7. We utilized our model to predict the next day open value in the validating dataset. With 800 observations in the validating dataset, the predicted value is on average 0.5094 percent away from the actual value. Our model exhibited superior predicting performance.

In our model, we predicted the next day open value. With the latest trading information, the predicted value can be transformed into return. With the predicted value and return, it helps investors incorporate transaction costs and other trading considerations to make informed investment decisions. In addition, we utilized the S&P500 Index as our example, as the S&P 500 Index is the proxy for the US stock market. The model can be applied to any stocks or other investment vehicles. Stocks are usually more volatile than the overall market, so the model is even more valuable when applied to stock prediction. In addition, we are able to generate more predicting variables to increase the predicted accuracy in the process. We argue that our NN-GA model is more flexible and applicable. Our experiment evidence suggested that the NN-GA model adds value to the stock market prediction and is a promising tool to help make complex investment decisions.

Table 7 Stock market prediction with NN-GA model

	Average (%)	Stdev (%)	Max. (%)	Min. (%)
Predicting deviation	0.5094	0.5466	4.8289	0.0001

7 Conclusion

In this research, we propose a three-layer genetic-neural hybrid system with a trend detector and a maximizer for stock market prediction. We collected 10 years US stock market daily data. We identify a set of useful predictors through a sequence of experiments in refining and expanding the original predictors. The data set consists of training and validation datasets. Our hybrid system's average predicting accuracy reaches as high as 95 percent, with the predicted value very close to the target value. With the predicted value, investors can further incorporate transaction costs and other trading considerations to make informed investment decisions. Our research results suggest that our model serve as a promising tool for stock market prediction. In addition, our system is capable of exhibiting learning behavior and gradual performance improvement in the basis of experience. We argue that our NN-GA model is more flexible and adds value to the stock market prediction.

This set of our predictors is based on the time period studied in this research. The most effective set of predictors might vary at different time period, for important political or economic events might happen during different period of time. In addition, the data we used to train our system represent just a small sample in the entire data universe. It is worth further investigation to include more economic, political, business, and other environmental factors in the future research.

Parameter setting has been a long-time thorny issue for GA implementation. In this research we have focused on one of the most difficult issues, i.e., the setting for the mutation rate. Our research lends one more support to the "low-mutation-rate" school of researchers and practitioners. Using the stock market prediction as an application domain for our hybrid model, we have helped shed light on the role and importance of the mutation, and the possible complementary effect of mutation and crossover functions. We plan to investigate how the crossover rate might interact with the decision task complexity level, selection strategies, population size, and mutation rates with the ultimate goal to make contributions to establishing a guideline for the design of GA parameters.

Data availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests that are directly or indirectly related to the work submitted for publication.

References

1. Zadeh LA (1994) Fuzzy logic, neural networks, and soft computing. Commun ACM 37(3):77–84
2. Baker R (1998) Genetic algorithms in search and optimization. Financ Eng News 2(3)
3. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, San Francisco
4. Grefenstette JJ (1993) Introduction to the special track on genetic algorithms. IEEE Expert Intell Syst Appl 8(5):5–8
5. Back T, Hammel U, Schwefel H (1997) Evolutionary computation: comments on the history and current state. IEEE Trans Evol Comput 1(1):3–17
6. Mitchell M (1996) An introduction to genetic algorithms. MIT Press, Cambridge
7. Holland JH (1992) Genetic algorithms. Sci Am 267:66–72
8. Spears WM (1993) Crossover or mutation? In: LD Whitley (ed) Foundation of genetic algorithms, 2. Morgan Kaufmann, San Francisco
9. Muhlenbein H (1992) How genetic algorithms really work: mutation and hill climbing. In: R Manner, B Manderick (eds) Parallel problem solving from nature, vol. 2. North-Holland
10. Deng PS (1999) Using genetic algorithms for batch selection decisions. Expert Syst Appl 17:183–194
11. Deng PS, Tsacle EG (2000) Coupling genetic algorithms and rule-based systems for complex decisions. Expert Syst Appl 19(3):209–218
12. Mitchell TM (1997) Machine learning. McGraw-Hill, New York
13. Fama EF (1970) Efficient capital markets: a review of theory and empirical work. J Financ 5(2):383–417
14. Haugen RA (1998) The inefficient stock market. Prentice Hall, Upper Saddle River
15. Bodie Z, Kane A, Marcus AJ (1995) Investments, 3rd edn. McGraw-Hill, New York
16. Haugen RA (1998) Beast on wall street. Pearson
17. Chorafas DN (1994) Chaos theory in the financial markets. McGraw-Hill, New York
18. Chen CC, Liu YS, Hsu TH (2019) An analysis on investment performance of machine learning: an empirical examination on Taiwan stock market. Int J Econ Financ Issues 9(4):1–10
19. Mallikarjuna M, Rao RP (2019) Evaluation of forecasting methods from selected stock market returns. Financ Innov 5(1):1–16
20. Zhong X, Enke D (2019) Predicting the daily return direction of the stock market using hybrid machine learning algorithms. Financ Innov 5(1):1–20
21. Liew JKS, Mayster B (2018) Forecasting ETFs with machine learning algorithms. J Altern Invest 20(3):58–78
22. Pimenta A, Nametala CAL, Guimaraes FG, Carrano EG (2018) An automated investing method for stock market based on multi-objective genetic programming. Comput Econ 52(1):125–144

23. Singh R, Srivastava S (2017) Stock prediction using deep learning. Multimed Tools Appl 76(18):18569–18584
24. Siddiqui TA, Abdullah Y (2015) Developing a nonlinear model to predict stock prices in India: an artificial neural networks approach. IUP J Appl Financ 21(3):36–49
25. Jan MN, Ayub U (2019) Do the Fama and French five-factor model forecast well using ANN. J Bus Econ Manag 20(1):168–191
26. Macchiarulo A (2018) Predicting and besting the stock market with machine learning and technical analysis. J Internet Bank Commer 23(1):1–22
27. Kyriakou I, Mousavi P, Nielsen JP, Scholz M (2021) Forecasting benchmarks of long-term stock returns via machine learning. Ann Oper Res 297(1):221–240
28. Safa M, Panahian H (2018) P/E model and prediction of firms listed on the Tehran stock exchange: a new approach to harmony search algorithm and neural network hybridization. Iran J Manag Stud 11(4):769–793
29. Nti IK, Adekoya AF, Weyori BA (2020) A comprehensive evaluation of ensemble learning for stock-market prediction. J Big Data 7(1):1–40
30. Mitchell TM (1999) Machine learning and data mining. Commun ACM 42(11):30–36
31. Liebowitz J (1993) Roll your own hybrids. Byte 18(9):113–115
32. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
33. Knox SW (2018) Machine learning: a concise introduction. Wiley & Sons, Hoboken
34. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2:303–314
35. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2:359–366
36. Brownlee J (2021) Better deep learning: Train faster, reduce overfitting, and make better predictions. Machine Learning Mastery
37. Kennedy RL, Lee Y, van Roy B, Reed C, Lippmann RP (1997) Solving data mining problems through pattern recognition. Prentice Hall, Upper Saddle River
38. Spears WM, DeJong KA (1995) On the virtues of parameterized uniform crossover. In: RK Belew, LB Booker (eds.) Proceedings of the fourth international conference on genetic algorithms. Morgan Kaufmann, San Francisco
39. DeJong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan, MI
40. Schaffer JD, Caruana RA, Eshelman LJ, Das R (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization. In: JD Schaffer (ed.) Proceedings of the third international conference on genetic algorithm
41. Croitoru NE (2014) High-probability mutation in basic genetic algorithm. In: 16th international symposium on symbolic and numeric algorithms for scientific computing