Sequential optimizing investing strategy with neural networks

Ryo Adachi*†
Akimichi Takemura*

February 2010

Abstract

In this paper we propose an investing strategy based on neural network models combined with ideas from game-theoretic probability of Shafer and Vovk. Our proposed strategy uses parameter values of a neural network with the best performance until the previous round (trading day) for deciding the investment in the current round. We compare performance of our proposed strategy with various strategies including a strategy based on supervised neural network models and show that our procedure is competitive with other strategies.

1 Introduction

A number of researches have been conducted on prediction of financial time series with neural networks since Rumelhart [10] developed back propagation algorithm in 1986, which is the most commonly used algorithm for supervised neural network. With this algorithm the network learns its internal structure by updating the parameter values when we give it training data containing inputs and outputs. We can then use the network with updated parameters to predict future events containing inputs the network has never encountered. The algorithm is applied in many fields such as robotics and image processing and it shows a good performance in prediction of financial time series.

*Graduate School of Information Science and Technology, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, JAPAN

†Institute of Industrial Science, University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, JAPAN

Relevant papers on the use of neural network to financial time series include [5], [6], [8] and [14].

In these papers authors are concerned with the prediction of time series and they to not pay much attention to actual investing strategies, although the prediction is obviously important in designing practical investing strategies. A forecast of tomorrow's price does not immediately tell us how much to invest today. In contrast to these works, in this paper we directly consider investing strategies for financial time series based on neural network models and ideas from game-theoretic probability of Shafer and Vovk (2001) [11]. In the game-theoretic probability established by Shafer and Vovk, various theorems of probability theory, such as the strong law of large numbers and the central limit theorem, are proved by consideration of capital processes of betting strategies in various games such as the coin-tossing game and the bounded forecasting game. In game-theoretic probability a player "Investor" is regarded as playing against another player "Market". In this framework investing strategies of Investor play a prominent role. Prediction is then derived based on strong investing strategies (cf. defensive forecasting in [13]).

Recently in [9] we proposed sequential optimization of parameter values of a simple investing strategy in multi-dimensional bounded forecasting games and showed that the resulting strategy is easy to implement and shows a good performance in comparison to well-known strategies such as the universal portfolio [3] developed by Thomas Cover and his collaborators. In this paper we propose sequential optimization of parameter values of investing strategies based on neural networks. Neural network models give a very flexible framework for designing investing strategies. With simulation and with some data from Tokyo Stock Exchange we show that the proposed strategy shows a good performance.

The organization of this paper is as follows. In Section 2 we propose sequential optimizing strategy with neural networks. In Section 3 we present some alternative strategies for the purpose of comparison. In Section 3.1 we consider an investing strategy using supervised neural network with back propagation algorithm. The strategy is closely related to and reflects existing researches on stock price prediction with neural networks. In Section 3.2 we consider Markovian proportional betting strategies, which are much simpler than the strategies based on neural networks. In Section 4 we evaluate performances of these strategies by Monte Carlo simulation. In Section 5 we apply these strategies to stock price data from Tokyo Stock Exchange. Finally we give some concluding remarks in Section 6.

2 Sequential optimizing strategy with neural networks

Here we introduce the bounded forecasting game of Shafer and Vovk [11] in Section 2.1 and network models we use in Section 2.2. In Section 2.3 we specify the investing ratio by an unsupervised neural network and we propose sequential optimization of parameter values of the network.

2.1 Bounded forecasting game

We present the bounded forecasting game formulated by Shafer and Vovk in 2001 [11]. In the bounded forecasting game, Investor's capital at the end of round n is written as $\mathcal{K}_n$ $(n = 1, 2, \ldots)$ and initial capital $\mathcal{K}_0$ is set to be 1. In each round Investor first announces the amount of money M_n he bets ($|M_n| < \mathcal{K}_{n-1}$) and then Market announces her move $x_n \in [-1, 1]$. x_n represents the change of the price of a unit financial asset in round n. The bounded forecasting game can be considered as an extension of the classical coin-tossing game since the bounded forecasting game results in the classical coin-tossing game if $x_n \in \{-1, 1\}$. With $\mathcal{K}_n, M_n$ and x_n, Investor's capital after round n is written as $\mathcal{K}_n = \mathcal{K}_{n-1} + M_n x_n$.

The protocol of the bounded forecasting game is written as follows.

Protocol:

$\mathcal{K}_0 = 1$.
FOR $n = 1, 2, \ldots$:
    Investor announces $M_n \in \mathbb{R}$.
    Market announces $x_n \in [-1, 1]$.
    $\mathcal{K}_n = \mathcal{K}_{n-1} + M_n x_n$
END FOR

We can rewrite Investor's capital as $\mathcal{K}_n = \mathcal{K}_{n-1} \times (1 + \alpha_n x_n)$, where $\alpha_n = M_n / \mathcal{K}_{n-1}$ is the ratio of Investor's investment M_n to his capital $\mathcal{K}_{n-1}$ after round $n-1$. We call α_n the investing ratio at round n. We restrict α_n as $-1 < \alpha_n < 1$ in order to prevent Investor becoming bankrupt. Furthermore we can write $\mathcal{K}_n$ as

$$\mathcal{K}_n = \mathcal{K}_{n-1}(1 + \alpha_n x_n) = \cdots = \Pi_{k=1}^{n}(1 + \alpha_k x_k).$$

Taking the logarithm of $\mathcal{K}_n$ we have

$$\log \mathcal{K}_n = \sum_{k=1}^{n} \log(1 + \alpha_k x_k). \tag{1}$$

The behavior of Investor's capital in (1) depends on the choice of α_k. Specifying a functional form of α_k is regarded as an investing strategy. For example, setting $\alpha_k \equiv \epsilon$ to be a constant ϵ for all k is called the ϵ-strategy which is presented in [11]. In this paper we consider various ways to determine α_k in terms of past values $x_{k-1}, x_{k-2}, \ldots$, of x and and seek better α_k in trying to maximize the future capital $\mathcal{K}_n$, $n > k$.

Let $\boldsymbol{u}_{k-1} = (x_{k-1}, \ldots, x_{k-L})$ denote past L values of x and let α_k depend on $\boldsymbol{u}_{k-1}$ and a parameter ω: $\alpha_k = f(\boldsymbol{u}_{k-1}, \omega)$. Then

$$\omega^*_{k-1} = \text{argmax} \sum_{t=1}^{k-1} \log(1 + f(\boldsymbol{u}_{t-1}, \omega) x_t)$$

is the best parameter value until the previous round. In our sequential optimizing investing strategy, we use ω^*_{n-1} to determine the investment M_n at round n:

$$M_n = \mathcal{K}_{n-1} \times f(\boldsymbol{u}_{n-1}, \omega^*_{n-1}).$$

For the function f we employ neural network models for their flexibility, which we describe in the next section.

2.2 Design of the network

We construct a three-layered neural network shown in Figure 1. The input layer has L neurons and they just distribute the input u_j $(j = 1, \ldots, L)$ to every neuron in the hidden layer. Also the hidden layer has M neurons and we write the input to each neurons as I^2_i which is a weighted sum of u_j's.

As seen from Figure 1, I^2_i is obtained as

$$I^2_i = \sum_{j=1}^{L} \omega^{1,2}_{ij} u_j,$$

where $\omega^{1,2}_{ij}$ is called the weight representing the synaptic connectivity between the jth neuron in the input layer and the ith neuron in the hidden layer. Then the output of the ith neuron in the hidden layer is described as

$$y^2_i = \tanh(I^2_i).$$

As for activation function we employ hyperbolic tangent function. In a similar way, the input to the neuron in the output layer, which we write I^3, is obtained as

$$I^3 = \sum_{i=1}^{M} \omega^{2,3}_i y^2_i,$$

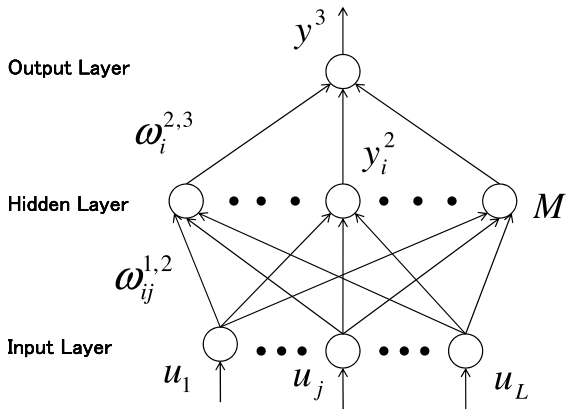

Figure 1: Three-layered network

where ω_i^1 is the weight between the ith neuron in the hidden layer and the neuron in the output layer. Finally, we have

$$y^3 = \tanh(I^3),$$

which is the output of the network. In the following argument we use y^3 as an investment strategy. Thus we can write

$$\alpha_k = y^3 = f(\boldsymbol{u}_{k-1}, \boldsymbol{\omega}),$$

where

$$\boldsymbol{\omega} = (\boldsymbol{\omega}^{1,2}, \boldsymbol{\omega}^{2,3}) = \big((\omega_{ij}^{1,2})_{i=1,\ldots,M,\, j=1,\ldots,L}, (\omega_i^{2,3})_{i=1,\ldots,M}\big).$$

Investor's capital is written as

$$\mathcal{K}_n = \mathcal{K}_{n-1}(1 + f(\boldsymbol{u}_{n-1}, \boldsymbol{\omega})x_n).$$

We need to specify the number of inputs L and the number of neurons M in the hidden layer. It is difficult to specify them in advance. We compare various choices of L and M in Section 4 and Section 5. Also in $\boldsymbol{u}_{n-1}$ we can include any input which is available before the start of round n, such as moving averages of past prices, seasonal indicators or past values of other economic time series data. We give further discussion on the choice of $\boldsymbol{u}_{n-1}$ in Section 6.

2.3 Sequential optimizing strategy with neural networks

In this section we propose a strategy which we call Sequential Optimizing Strategy with Neural Networks (SOSNN).

We first calculate $\boldsymbol{\omega}^* = \boldsymbol{\omega}^*_{n-1}$ that maximizes

$$\phi = \sum_{k=1}^{n-1} \log(1 + f(\boldsymbol{u}_{k-1}, \boldsymbol{\omega}) x_k). \tag{2}$$

This is the best parameter values until the previous round. If Investor uses $\alpha_n = f(\boldsymbol{u}_{n-1}, \boldsymbol{\omega}^*_{n-1})$ as the investing ratio, Investor's capital after round n is written as

$$\mathcal{K}_n = \mathcal{K}_{n-1}(1 + f(\boldsymbol{u}_{n-1}, \boldsymbol{\omega}^*_{n-1}) x_n).$$

For maximization of (2), we employ the gradient descent method. With this method, the weight updating algorithm of $\omega_i^{2,3}$ with the parameter β (called the learning constant) is written as

$$\omega_i^{2,3} = \omega_i^{2,3} + \Delta\omega_i^{2,3} = \omega_i^{2,3} + \beta \frac{\partial \phi}{\partial \omega_i^{2,3}},$$

where

$$\begin{aligned} \frac{\partial \phi}{\partial \omega_i^{2,3}} &= \frac{\partial \phi}{\partial f} \frac{\partial f}{\partial \omega_i^{2,3}} = \sum_{k=1}^{n-1} \frac{\partial \phi}{\partial f} \frac{\partial f}{\partial\, {}^k I^3} \frac{\partial\, {}^k I^3}{\partial \omega_i^{2,3}} \\ &= \sum_{k=1}^{n-1} \frac{x_k}{1 + f(\boldsymbol{u}_{k-1}, \boldsymbol{\omega}) x_k} (1 - \tanh^2({}^k I^3))\, {}^k y_i^2 \\ &= \sum_{k=1}^{n-1} {}^k\delta_1\, {}^k y_i^2, \end{aligned}$$

and the left superscript k to I^3, y_i^2 indexes the round. Thus we obtain

$$\Delta\omega_i^{2,3} = \beta \frac{\partial \phi}{\partial \omega_i^{2,3}} = \beta \sum_{k=1}^{n-1} {}^k\delta_1\, {}^k y_i^2.$$

Similarly, the weight updating algorithm of $\omega_{ij}^{1,2}$ is expressed as

$$\omega_{ij}^{1,2} = \omega_{ij}^{1,2} + \Delta\omega_{ij}^{1,2} = \omega_{ij}^{1,2} + \beta \frac{\partial \phi}{\partial \omega_{ij}^{1,2}},$$

where

$$\begin{aligned}\frac{\partial\phi}{\partial\omega_{ij}^{1,2}} &= \sum_{k=1}^{n-1}\frac{\partial\phi}{\partial\,{}^kI_i^2}\frac{\partial\,{}^kI_i^2}{\partial\omega_{ij}^{1,2}} = \sum_{k=1}^{n-1}\frac{\partial\phi}{\partial f}\frac{\partial f}{\partial\,{}^kI^3}\frac{\partial\,{}^kI^3}{\partial\,{}^ky_i^2}\frac{\partial\,{}^ky_i^2}{\partial\,{}^kI_i^2}\frac{\partial\,{}^kI_i^2}{\partial\omega_{ij}^{1,2}}\\ &= \sum_{k=1}^{n-1}{}^k\delta_1\omega_i^{2,3}(1-\tanh^2({}^kI_i^2))\,(\boldsymbol{u}_{k-1})_j\\ &= \sum_{k=1}^{n-1}{}^k\delta_2\,(\boldsymbol{u}_{k-1})_j.\end{aligned}$$

Thus we obtain

$$\Delta\omega_{ij}^{1,2} = \beta\frac{\partial\phi}{\partial\omega_{ij}^{1,2}} = \beta\sum_{k=1}^{n-1}{}^k\delta_2(\boldsymbol{u}_{k-1})_j.$$

Here we summarize the algorithm of SOSNN at round n.

1. Given the input vector $\boldsymbol{u}_{k-1} = (x_{k-1},\ldots,x_{k-L})$ $(k=1,\ldots,n-1)$ and the value of $\boldsymbol{\omega}_{n-1}$, we first evaluate ${}^kI_i^2 = \sum_{j=1}^{L}\omega_{ij}^{1,2}(\boldsymbol{u}_{k-1})_j$ and then ${}^ky_i^2 = \tanh({}^kI_i^2)$. Also we set the learning constant β.

2. We calculate ${}^kI^3 = \sum_{i=1}^{M}\omega_i^{2,3}\,{}^ky_i^2$ and then ${}^ky^3 = \tanh({}^kI^3)$ with ${}^kI_i^2$ and ${}^ky_i^2$ of the previous step. Then we update weight $\boldsymbol{\omega}$ with the weight updating formula $\omega_i^{2,3} = \omega_i^{2,3} + \beta\sum_{k=1}^{n-1}{}^k\delta_1\,{}^ky_i^2$ and $\omega_{ij}^{1,2} = \omega_{ij}^{1,2} + \beta\sum_{k=1}^{n-1}{}^k\delta_2(\boldsymbol{u}_{k-1})_j$.

3. Go back to step 1 replacing the weight $\boldsymbol{\omega}_{n-1}$ with updated values.

After sufficient times of iteration, ϕ in (2) converges to a local maximum with respect to $\omega_{ij}^{1,2}$ and $\omega_i^{2,3}$ and we set $\omega_{ij}^{1,2} = \omega_{ij}^{1,2*}$ and $\omega_i^{2,3} = \omega_i^{2,3*}$, which are elements of $\boldsymbol{\omega}_{n-1}^*$. Then we evaluate Investor's capital after round n as $\mathcal{K}_n = \mathcal{K}_{n-1}(1 + f(\boldsymbol{u}_{n-1}, \boldsymbol{\omega}_{n-1}^*)x_n)$.

3 Alternative investment strategies

Here we present some strategies that are designed to be compared with SOSNN. In Section 3.1 we present a strategy with back-propagating neural network. The advantage of back-propagating neural network is its predictive ability due to "learning" as previous researches show. In Section 3.2 we show some sequential optimizing strategies that use rather simple function for f than SOSNN does.

3.1 Optimizing strategy with back propagation

In this section we consider a supervised neural network and its optimization by back propagation. We call the strategy NNBP. It decides the betting ratio by predicting actual up-and-downs of stock prices and can be regarded as incorporating existing researches on stock price prediction. Thus it is suitable as an alternative to SOSNN.

For supervised network, we train the network with the data from a training period, obtain the best value of the parameters for the training period and then use it for the investing period. These two periods are distinct. For the training period we need to specify the desired output (target) T_k of the network for each day k. We propose to specify the target by the direction of Market's current price movement x_k. Thus we set

$$T_k = \begin{cases} +1 & x_k > 0 \\ 0 & x_k = 0 \\ -1 & x_k < 0 \end{cases}.$$

Note that this T_k is the best investing ratio if Investor could use the current movement x_k of Market for his investment. Therefore it is natural to use T_k as the target value for investing strategies. We keep on updating $\boldsymbol{\omega}_k$ by cycling through the input-output pairs of the days of the training period and finally obtain $\boldsymbol{\omega}^*$ after sufficient times of iteration.

Throughout the investing period we use $\boldsymbol{\omega}^*$ and Investor's capital after round n in the investing period is expressed as

$$\mathcal{K}_n = \mathcal{K}_{n-1}(1 + f(\boldsymbol{u}_{n-1}, \boldsymbol{\omega}^*)x_n).$$

Back propagation is an algorithm which updates weights ${}^k\omega_{ij}^{1,2}$ and ${}^k\omega_i^{2,3}$ so that the error function

$$E_k = \frac{1}{2}(T_k - {}^k y^3)^2$$

decreases, where T_k is the desired output of the network and ${}^k y^3$ is the actual output of the network. The weight ${}^k\omega_i^{2,3}$ of day k is renewed to the weight ${}^{k+1}\omega_i^{2,3}$ of day $k+1$ as

$$\begin{aligned} {}^{k+1}\omega_i^{2,3} &= {}^k\omega_i^{2,3} + \Delta {}^k\omega_i^{2,3} = {}^k\omega_i^{2,3} - \beta \frac{\partial E_k}{\partial {}^k\omega_i^{2,3}} = {}^k\omega_i^{2,3} - \beta \frac{\partial E_k}{\partial {}^k I^3} \frac{\partial {}^k I^3}{\partial {}^k\omega_i^{2,3}} \\ &= {}^k\omega_i^{2,3} - \beta \, {}^k\epsilon^1 \, {}^k y_i^2, \end{aligned}$$

where

$${}^k\epsilon^1 = \frac{\partial E_k}{\partial {}^k I^3} = \frac{\partial E_k}{\partial {}^k y^3} \frac{\partial {}^k y^3}{\partial {}^k I^3} = -(T_k - {}^k y^3)(1 - \tanh^2({}^k I^3)).$$

Also weight ${}^k\omega_{ij}^{1,2}$ is renewed as

$$\begin{aligned} {}^{k+1}\omega_{ij}^{1,2} &= {}^k\omega_{ij}^{1,2} + \Delta {}^k\omega_{ij}^{1,2} = {}^k\omega_{ij}^{1,2} - \beta \frac{\partial E_k}{{}_k\omega_{ij}^{1,2}} = {}^k\omega_{ij}^{1,2} - \beta \frac{\partial E_k}{\partial\, {}^kI^3} \frac{\partial\, {}^kI^3}{\partial\, {}^ky_i^2} \frac{\partial\, {}^ky_i^2}{\partial\, {}^kI_i^2} \frac{\partial\, {}^kI_i^2}{\partial {}^k\omega_{ij}^{1,2}} \\ &= {}^k\omega_{ij}^{1,2} - \beta\, {}^k\epsilon_i^2\, (\tilde{\boldsymbol{u}}_{k-1})_j, \end{aligned}$$

where

$${}^k\epsilon_i^2 = \frac{\partial E_k}{\partial\, {}^kI^3} \frac{\partial\, {}^kI^3}{\partial\, {}^ky_i^2} \frac{\partial\, {}^ky_i^2}{\partial\, {}^kI_i^2} = {}^k\epsilon^1 \omega_i^{2,3} (1 - \tanh^2({}^kI_i^2)).$$

At the end of each step we calculate the training error defined as

$$\text{training error} = \frac{1}{2m} \sum_{k=1}^{m} (T_k - {}^ky^3)^2 = \frac{1}{m} \sum_{k=1}^{m} E_k, \tag{3}$$

where m is the length of the training period. We end the iteration when the the training error becomes smaller than the threshold μ, which is set sufficiently small.

Here let us summarize the algorithm of NNBP in the training period.

1. We set $k = 1$.
2. Given the input vector $\boldsymbol{u}_{k-1} = (x_{k-1}, \ldots, x_{k-L})$ and the value of $\boldsymbol{\omega}_k$, we first evaluate ${}^kI_i^2 = \sum_{j=1}^{L} {}^k\omega_{ij}^{1,2} (\boldsymbol{u}_{k-1})_j$ and then ${}^ky_i^2 = \tanh({}^kI_i^2)$. Also we set the learning constant β.
3. We calculate ${}^kI^3 = \sum_{i=1}^{M} {}^k\omega_i^{2,3}\, {}^ky_i^2$ and then ${}^ky^3 = \tanh({}^kI^3)$ with ${}^kI_i^2$ and ${}^ky_i^2$ of the previous step. Then we update weight $\boldsymbol{\omega}_k$ with the weight updating formula ${}^{k+1}\omega_i^{2,3} = {}^k\omega_i^{2,3} - \beta\, {}^k\epsilon^1\, {}^ky_i^2$ and ${}^{k+1}\omega_{ij}^{1,2} = {}^k\omega_{ij}^{1,2} - \beta\, {}^k\epsilon_i^2 (\boldsymbol{u}_{k-1})_j$.
4. Go back to step 2 setting $k + 1 \leftarrow k$ and $\boldsymbol{\omega}_{k+1} \leftarrow \boldsymbol{\omega}_k$ while $1 \le k \le m$. When $k = m$ we set $k = 1$ and $\boldsymbol{\omega}_1 \leftarrow \boldsymbol{\omega}_{m+1}$ and continue the algorithm until the training error becomes less than μ.

3.2 Markovian proportional betting strategies

In this section we present some sequential optimizing strategies that are rather simple compared to strategies with neural network in Section 2 and Section 3.1. The strategies of this section are generalizations of Markovian strategy in [12] for coin-tossing games to bounded forecasting games. We

present these simple strategies for comparison with SOSNN and observe how complexity in function f increases or decreases Investor's capital processes in numerical examples in later sections.

Consider maximizing the logarithm of Investor's capital in (1):

$$\log \mathcal{K}_n = \sum_{k=1}^{n} \log(1 + \alpha_k x_k).$$

We first consider the following simple strategy of [9] in which we use $\alpha_n = \alpha^*_{n-1}$, where

$$\alpha^*_{n-1} = \text{argmax}\, \Pi_{k=1}^{n-1}(1 + \alpha x_k).$$

In this paper we denote this strategy by MKV0.

As a generalization of MKV0 consider using different investing ratios depending on whether the price went up or down on the previous day. Let $\alpha_k = \alpha^+_k$ when x_{k-1} was positive and $\alpha_k = \alpha^-_k$ when it was negative. We denote this strategy by MKV1. In the betting on the nth day we use $\alpha^+_n = {\alpha^+_{n-1}}^*$ and $\alpha^-_n = {\alpha^-_{n-1}}^*$, where

$$({\alpha^+_{n-1}}^*, {\alpha^-_{n-1}}^*) = \text{argmax}\, \Pi_{k=1}^{n-1}(1 + f(\boldsymbol{u}_{k-1}, \alpha^+, \alpha^-)x_k),$$

$\boldsymbol{u}_{k-1} = (x_{k-1})$ and

$$f(\boldsymbol{u}_{k-1}, \alpha^+, \alpha^-) = \alpha^+ \mathrm{I}_{\{x_{k-1} \geq 0\}} + \alpha^- \mathrm{I}_{\{x_{k-1} < 0\}}.$$

Here $\mathrm{I}_{\{\cdot\}}$ denotes the indicator function of the event in $\{\cdot\}$. The capital process of MKV1 is written in the form of (1) as

$$\log \mathcal{K}_n = \sum_{k=1}^{n} \log(1 + f(\boldsymbol{u}_{k-1}, {\alpha^+_{k-1}}^*, {\alpha^-_{k-1}}^*)x_k).$$

We can further generalize this strategy considering price movements of past two days. Let $\boldsymbol{u}_{k-1} = (x_{k-1}, x_{k-2})$ and let

$$\begin{aligned} f(\boldsymbol{u}_{k-1}, \alpha^{++}, \alpha^{+-}, \alpha^{-+}, \alpha^{--}) = \alpha^{++} \mathrm{I}_{\{x_{k-2} \geq 0,\, x_{k-1} \geq 0\}} + \alpha^{+-} \mathrm{I}_{\{x_{k-2} \geq 0,\, x_{k-1} < 0\}} \\ + \alpha^{-+} \mathrm{I}_{\{x_{k-2} < 0,\, x_{k-1} \geq 0\}} + \alpha^{--} \mathrm{I}_{\{x_{k-2} <, x_{k-1} < 0\}}. \end{aligned}$$

We denote this strategy by MKV2.

We will compare performances of the above Markovian proportional betting strategies with strategies based on neural networks in the following sections.

4 Simulation with linear models

In this section we give some simulation results for strategies shown in Section 2 and Section 3. We use two linear time series models to confirm the behavior of presented strategies. Linear time series data are generated from the Box-Jenkins family [2], autoregressive model of order 1 (AR(1)) and autoregressive moving average model of order 2 and 1 (ARMA(2,1)) having the same parameter values as in [15]. AR(1) data are generated as

$$x_n = 0.6x_{n-1} + \epsilon_n \tag{4}$$

and ARMA(2,1) data are generated as

$$x_n = 0.6x_{n-1} + 0.3x_{n-2} + \epsilon_n - 0.5\epsilon_{n-1}, \tag{5}$$

where we set $\epsilon_n \sim N(0, 1)$. After the series is generated, we divide each value by the maximum absolute value to normalize the data to the admissible range $[-1, 1]$.

Here we discuss some details on calculation of each strategy. First we set the initial values of elements of $\boldsymbol{\omega}$ as random numbers in $[-0.1, 0.1]$. In SOSNN, we use the first 20 values of x_n as initial values and the iteration process in gradient descent method is proceeded until $|\Delta\omega_{ij}^{1,2}| < 10^{-4}$ and $|\Delta\omega_i^{2,3}| < 10^{-4}$ with the upper bound of 10^4 steps. As for the learning constant β, we use learning-rate annealing schedules which appear in Section 3.13 of [7]. With annealing schedule called the search-then-converge schedule [4] we put β at the nth step of iteration as

$$\beta(n) = \frac{\beta_0}{1 + (n/\tau)},$$

where β_0 and τ are constants and we set $\beta_0 = 1.0$ and $\tau = 5.0$. In NNBP, we train the network with five different training sets of 300 observations generated by (4) and (5). We continue cycling through the training set until the training error becomes less than μ and we set $\mu = 10^{-2}$ with the upper bound of 6×10^5 steps. Also we check the fit of the network to the data by means of the training error for some different values of β, L and M. In Markovian strategies, we again use the first 20 values of x_n as initial values. We also adjust the data so that the betting is conducted on the same data regardless of L in SOSNN and NNBP or different number of inputs among Markovian strategies.

In Table 1 we summarize the results of SOSNN, NNBP, MKV0, MKV1 and MKV2 under AR(1) and ARMA(2,1). The values presented are averages

of results for five different simulation runs of (4) and (5). As for SOSNN, we simulate fifty cases (combinations of $L = 1, \ldots, 5$ and $M = 1, \ldots, 10$), but only report the cases of $L = 1, 2, 3$ and some choices of M because the purpose of the simulation is to test whether $L = 1$ works better than other choices of L under AR(1) and $L = 2$ works better under ARMA(2,1). For NNBP we only report the result for one case since fitting the parameters to the data is quite a difficult task due to the characteristic of desired output (target). Also once we obtain the value of β, L and M with training error less than the threshold $\mu = 10^{-2}$, we find that the network has successfully learned the input-output relationship and we do not test other choices of the above parameters. (See Appendix for more detail.) We set $\beta = 0.07$, $L = 12$ and $M = 30$ in simulation with AR(1) model and $\beta = 0.08$, $L = 15$ and $M = 40$ in simulation with ARMA(2,1) model.

Investor's capital process for each choice of L and M in SOSNN, NNBP and each Markovian strategy is shown in three rows, corresponding to rounds 100, 200, 300 of the betting (without the initial 20 rounds in SOSNN and Markovian strategies). The fourth row of each result for NNBP shows the training error after learning in the training period. The best value among the choices of L and M in SOSNN or among each Markovian strategy is written in bold and marked with an asterisk and the second best value is also written in bold and marked with two asterisks. Also calculation results written with "—" are cases in that simulation did not give proper values for some reasons.

Notice that SOSNN whose betting ratio f is specified by a complex function gives better performance than rather simple Markovian strategies both under AR(1) and ARMA(2,1). Also the result that NNBP gives capital processes which are competitive with other strategies shows that the network has successfully learned the input-output relationship in the training period.

5 Comparison of performances with some stock price data

In this section we present numerical examples calculated with the stock price data of three Japanese companies SONY, Nomura Holdings and NTT listed on the first section of the Tokyo Stock Exchange for comparing betting strategies presented in Section 2 and Section 3. The investing period (without days used for initial values) is 300 days from March 1st in 2007 to June 19th in 2008 for all strategies and the training period in NNBP is 300 days from December 1st in 2005 to February 20th in 2007. We use a shorter training period than those in previous researches, because longer periods resulted in

Table 1: Log capital processes of SOSNN, NNBP, MKV0, MKV1, MKV2 under AR(1) and ARMA(2,1)

AR(1) model

SOSNN

$L\backslash M$	1	2	3	4	5	6	7	8
	9.890	9.168	**10.021***	10.007	9.982	9.559	**10.009****	9.821
1	18.285	17.436	17.241	17.709	**18.480***	17.542	**18.260****	16.991
	30.151	23.574	29.465	29.949	**32.483***	28.941	—	—
	7.476	7.132	7.864	8.004	7.267	8.738	6.009	7.771
2	14.983	12.710	15.518	17.350	13.914	13.614	—	12.016
	26.211	25.785	26.090	**32.144****	21.981	23.137	—	19.492
	8.128	9.084	5.427	—	2.922	7.291	5.209	5.989
3	13.934	12.242	11.166	—	8.607	11.539	8.311	10.994
	23.232	19.889	20.013	—	16.330	18.807	16.055	20.388

NNBP	**MKV0**	**MKV1**	**MKV2**
10.118	−1.175	**7.831***	6.517
11.921	−0.800	**16.974***	15.452
24.323	−1.647	**32.392***	30.875
(5.28×10^{-3})			

ARMA(2,1) model

SOSNN

$L\backslash M$	1	2	3	4	5	6	7	8
	4.985	5.292	4.350	4.532	4.052	3.563	3.408	1.522
1	10.234	11.108	9.905	8.793	10.012	8.411	8.029	4.746
	11.666	11.460	10.024	9.924	11.781	8.016	—	5.882
	8.518	10.287	**11.052****	**11.567***	9.147	8.483	6.915	5.579
2	18.474	20.177	**21.458***	**20.768****	13.030	12.042	—	13.573
	16.818	**25.167***	24.979	**25.114****	17.538	15.074	—	16.846
	8.511	10.490	7.344	7.883	7.409	7.445	7.011	5.772
3	15.401	18.241	18.120	15.697	—	12.395	—	11.961
	18.047	24.904	23.362	21.280	—	15.930	—	14.185

NNBP	**MKV0**	**MKV1**	**MKV2**
7.813	−0.729	3.160	**7.566***
15.420	−0.132	10.483	**17.619***
25.819	−2.422	13.375	**22.911***
(2.04×10^{-2})			

poor fitting.

The data for 300 days from December 1st in 2005 to February 20th in 2007 is used for input normalization of x_n to $[-1, 1]$, which is conducted according to the method shown in [1] and the procedure is as follows. For the data of daily closing prices in the above period, we first obtain the maximum value of absolute daily movements and then divide daily movements in the investing period by that maximum value. In case $x_n \leq -1.0$ or $x_n \geq 1.0$ we put $x_n = -1.0$ or $x_n = 1.0$. Thus we obtain x_n in $[-1, 1]$ and we use them for inputs of the neural network. We tried periods of various lengths for normalization and decided to choose a relatively short period to avoid Investor's capital processes staying almost constant. Also in NNBP we used $\beta = 0.07$, $L = 12$ and $M = 90$ for SONY, $\beta = 0.07$, $L = 15$ and $M = 100$ for

Nomura and $\beta = 0.03$, $L = 15$ and $M = 120$ for NTT with upper bound of 10^5 iteration steps. Other details of calculation are the same as in Section 4. We report the results in Table 2.

In Figure 2 we show the movements of closing prices of each company during the investing period. In Figures 3-5 we show the log capital processes of the results shown in Table 2 to compare the performance of each strategy. Figure 3 is for SONY, Figure 4 is for Nomura Holdings and Figure 5 is for NTT. For SOSNN we plotted the result of L and M that gave the best performance at $n = 300$ (the bottom row of the three rows) in Table 2.

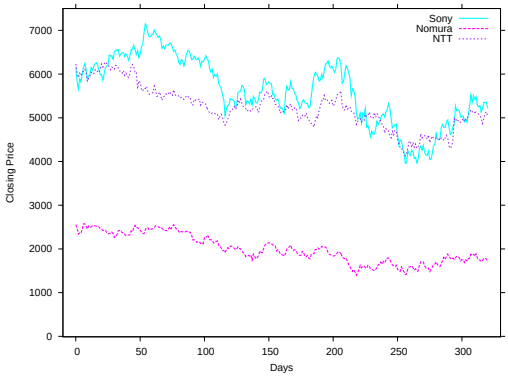

Figure 2: Closing prices

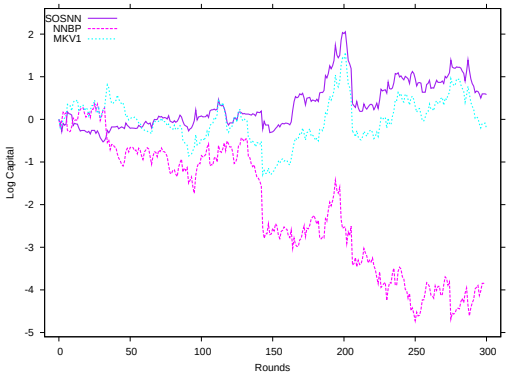

Figure 3: SONY

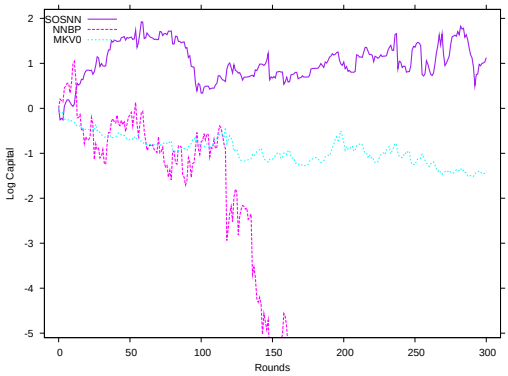

Figure 4: Nomura

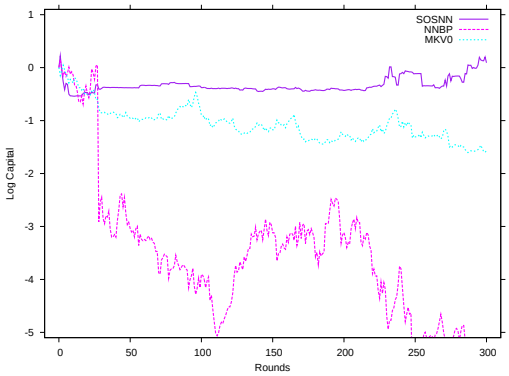

Figure 5: NTT

As we see from above figures, NNBP which shows competitive performance for two linear models in Section 4 gives the worst result. Thus it is obvious that the network has failed to capture trend in the betting period even if it fits in the training period. Also the results are favorable to SOSNN if we adopt appropriate numbers for L and M.

Table 2: Log capital process of SOSNN, NNBP, MKV0, MKV1 and MKV2 for TSE stocks

SONY

	SOSNN						
$L\backslash M$	1	2	4	5	7	8	9
	−0.339	−0.292	**0.144****	−0.832	−0.964	−0.896	0.082
1	0.572	0.520	**1.438****	0.401	0.383	−0.433	**2.012***
	−0.153	−0.220	**0.461****	−0.633	−0.762	−1.283	**0.582***
	−0.329	0.163	−1.273	−0.675	−0.349	−2.056	−2.541
2	0.000	−0.260	−1.825	−1.420	−0.072	−1.100	−1.256
	−0.565	−0.412	−2.274	−2.006	−0.980	−2.490	−1.981
	−0.230	−1.448	−1.402	−1.232	**0.397***	−0.627	−1.404
3	−0.309	−1.823	−0.438	−0.695	0.970	−0.599	0.281
	−0.307	−2.237	−1.333	−1.843	−0.212	−1.827	−2.749

NNBP	MKV0	MKV1	MKV2
−1.039	−0.578	**−0.459***	−1.285
−2.557	−0.979	**1.414***	−0.482
−3.837	−1.260	**−0.212***	−2.297
(3.67×10^{-2})			

Nomura

	SOSNN						
$L\backslash M$	1	2	4	5	7	8	9
	0.200	−0.212	−1.309	−0.479	**0.839***	**0.679****	−0.726
1	**1.193***	0.754	−2.417	−0.370	−2.650	0.005	−1.212
	−3.326	−2.888	−2.333	**0.581****	−6.938	−4.979	−0.504
	−0.819	0.338	−4.148	0.229	−4.787	−1.662	−0.754
2	−0.969	**1.030****	−4.920	−0.007	−11.478	−3.385	−10.046
	−0.202	**1.127***	−4.941	−7.003	−18.795	−10.897	−22.861
	−1.066	−2.076	−1.458	−0.389	−2.926	−1.783	−2.451
3	−1.111	1.002	−5.307	0.198	−2.897	−5.254	−9.595
	−3.672	−1.599	−10.570	−3.885	−0.621	−8.420	−14.021

NNBP	MKV0	MKV1	MKV2
−0.743	**−0.883***	−1.911	−3.789
−8.087	**−0.753***	−1.354	−4.970
−16.051	**−1.390***	−1.952	−6.410
(2.03×10^{-2})			

NTT

	SOSNN						
$L\backslash M$	1	2	4	5	7	8	9
	−0.674	−0.365	−1.824	**0.880****	−5.673	**0.888***	−4.757
1	−0.825	−0.411	**−0.246***	−4.248	−4.472	−7.931	−4.386
	−1.115	—	—	—	—	—	—
	−1.269	−6.049	−5.476	−2.769	−7.899	−6.606	−4.134
2	−1.175	−8.333	−9.581	−5.377	−11.071	−9.206	−5.250
	−0.498**	−10.660	—	−7.137	—	−13.900	—
	−0.377	−3.131	−2.192	−4.449	−1.670	−10.896	−8.633
3	**−0.431****	−5.861	−1.366	−9.349	−16.990	−10.811	−12.315
	0.092*	−5.463	—	−14.584	—	—	−15.456

NNBP	MKV0	MKV1	MKV2
−4.155	**−0.902***	−2.048	−4.788
−3.161	**−1.272***	−2.642	−6.807
−5.669	**−1.566***	−3.391	−8.687
(3.73×10^{-2})			

6 Concluding remarks

We proposed investing strategies based on neural networks which directly consider Investor's capital process and are easy to implement in practical applications. We also presented numerical examples for simulated and actual stock price data to show advantages of our method.

In this paper we only adopted normalized values of past Market's movements for the input $\boldsymbol{u}_{n-1}$ while we can use any data available before the start of round n as a part of the input as we mentioned in Section 2.2. Let us summarize other possibilities considered in existing researches on financial prediction with neural networks. The simplest choice is to use raw data without any normalization as in [6], in which they analyze time series of Athens Stock index to predict future daily index. In [5] they adopt price of FAZ-index (one of the German equivalents of the American Dow-Jones-Index), moving averages for 5, 10 and 90 days, bond market index, order index, US-Dollar and 10 successive FAZ-index prices as inputs to predict the weekly closing price of the FAZ-index. Also in [8] they use 12 technical indicators to predict the S&P 500 stock index one month in the future. From these researches we see that for longer prediction terms (such as monthly or yearly), longer moving averages or seasonal indexes become more effective. Thus those long term indicators may not have much effect in daily price prediction which we presented in this paper. On the other hand, adopting data which seems to have a strong correlation with closing prices of Tokyo Stock Exchange such as closing prices of New York Stock Exchange of the previous day may increase Investor's capital processes presented in this paper. Since there are numerical difficulties in optimizing neural networks, it is better to use small number of effective inputs for a good performance.

Another important generalization of the method of this paper is to consider portfolio optimization. We can easily extend the method in this paper to the betting on multiple assets. Let the output layer of the network have P neurons as shown in Figure 7 and the output of each neuron is expressed as y_h^3, $h = 1, \ldots, P$. Then we obtain a vector $\boldsymbol{y}^3 = (y_1^3, \ldots, y_P^3)$ of outputs. The number of neurons P refers to the number of different stocks Investor invests.

Investor's capital after round n is written as

$$\mathcal{K}_n = \mathcal{K}_{n-1}(1 + \sum_{h=1}^{P} f_h(\boldsymbol{u}_{n-1}, \boldsymbol{\omega}_h)x_{n,h}),$$

where

$$\boldsymbol{\omega}_h = \big((\omega_{ij}^{1,2})_{i=1,\ldots,M,\, j=1,\ldots,L}, (\omega_{hi}^{2,3})_{i=1,\ldots,M}\big).$$

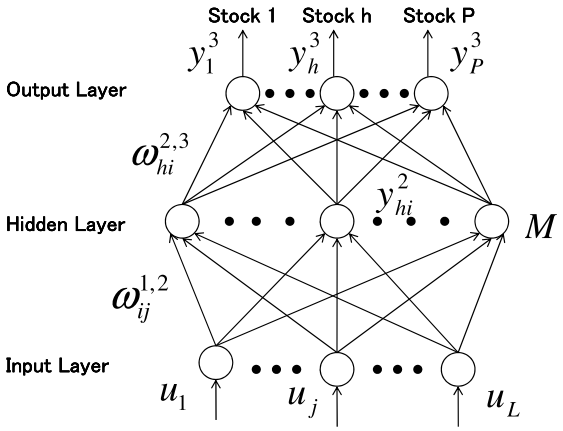

Figure 6: Three-layered network for portfolio cases

Thus also in portfolio cases we see that our method is easy to implement and we can evaluate Investor's capital process in practical applications.

Appendix

Here we discuss training error in the training period of NNBP. In this paper we set the threshold μ for ending the iteration to 10^{-2}, while the value commonly adopted in many previous researches is smaller, for instance, $\mu = 10^{-4}$. We give some details on our choice of μ.

Let us examine the case of Nomura Holdings in Section 5. In Figure 7 we show the training error after each step of iteration in the training period calculated with (3). While the plotted curve has a typical shape as those of previous researches, it is unlikely that the training error becomes less than 10^{-2}. Also in Figure 8 we plot $E_k = \frac{1}{2}(T_k - {}^k y^3)^2$ for each k calculated with parameter values $\boldsymbol{\omega}$ after learning. We observe that the network fails to fit for some points (actually 9 days out of 300 days) but perfectly fits for all other days. It can be interpreted that the network ignores some outliers and adjust to capture the trend of the whole data.

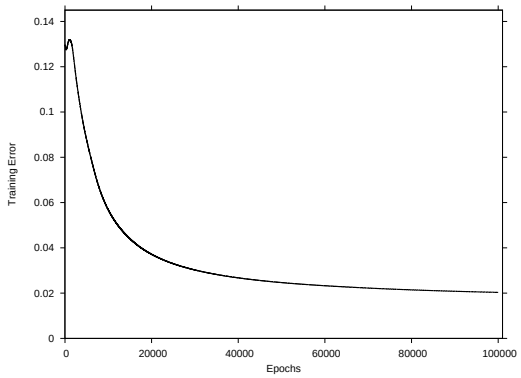

Figure 7: training error (epochs)

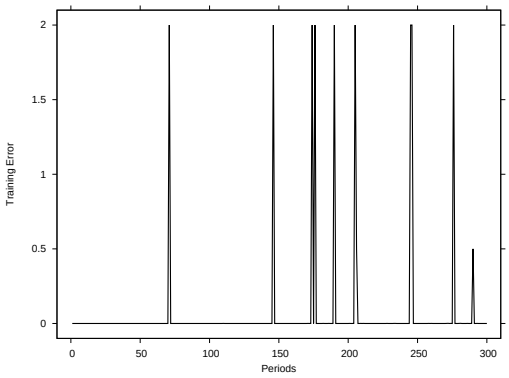

Figure 8: training error (periods)

References

[1] E. M. Azoff. *Neural Network Time Series Forecasting of Financial Markets.* Wiley, Chichester, 1994.

[2] G. P. E. Box and G. M. Jenkins. *Time Series: Analysis Forecasting and Control.* Holden-Day, San Francisco, 1970.

[3] T. M. Cover. Universal portfolios. *Mathematical Finance*, **1**, No.1, 1–29, 1991.

[4] C. Darken, J. Chang and J. Moody. Learning rate schedules for faster stochastic gradient search. *IEEE Second Workshop on Neural Networks for Signal Processing*, 3–12, 1992.

[5] B. Freisleben. Stock market prediction with backpropagation networks. *Industrial and Engineering Applications of Artificial Intelligence and Expert System 5th International Conference*, 451–460, 1992.

[6] M. Hanias, P. Curtis and J. Thalassinos. Prediction with neural networks: The Athens stock exchange price indicator. *European Journal of Economics, Finance and Administrative Sciences*, **9**, 21–27, 2007.

[7] S. S. Haykin. *Neural Networks and Learning Machines.* 3rd ed., Prentice Hall, New York, 2008.

[8] N. L. D. Khoa, K. Sakakibara and I. Nishikawa. Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors. *SICE-ICASE International Joint Conference*, 5484–5488, 2006.

[9] M. Kumon, A. Takemura and K. Takeuchi. Sequential optimizing strategy in multi-dimensional bounded forecasting games. `arXiv:0911.3933v1`, 2009.

[10] D. E. Rumelhart, G. E. Hinton and R. J. Williams. Learning internal representation by backpropagating errors. *Nature*, **323**, 533–536, 1986.

[11] G. Shafer and V. Vovk. *Probability and Finance: It's Only a Game!*. Wiley, New York, 2001.

[12] K. Takeuchi, M. Kumon and A. Takemura. Multistep Bayesian strategy in coin-tossing games and its application to asset trading games in continuous time. `arXiv:0802.4311v2`, 2008. Conditionally accepted to *Stochastic Analysis and Applications*.

[13] V. Vovk, A. Takemura and G. Shafer. Defensive Forecasting. *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics* (R. G. Cowell and Z. Ghahramani editors), 365–372, 2005.

[14] Y. Yoon and G. Swales. Predicting stock price performance: A neural network approach. *Proceedings of the 24th Annual Hawaii International Conference on System*, **4**, 156–162, 1991.

[15] G. P. Zhang. An investigation of neural networks for linear time-series forecasting. *Computers & Operations Research*, **28**, No.12, 1183–1202, 2001.