Decision Support Systems 64 (2014) 100–108

Contents lists available at ScienceDirect

Decision Support Systems

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

Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning

Dennis Eilers [a], Christian L. Dunis [b], Hans-Jörg von Mettenheim [a,*], Michael H. Breitner [a]

[a] Leibniz Universität Hannover, Institut für Wirtschaftsinformatik, Königsworther Platz 1, 30167 Hannover, Germany
[b] Horus Partners Wealth Management Group SA, 1, rue de la Rôtisserie, CH-1204 Geneva, Switzerland

ARTICLE INFO

Article history:
Received 13 December 2013
Received in revised form 11 April 2014
Accepted 28 April 2014
Available online 29 May 2014

JEL classification:
G17
C45
C88

Keywords:
Reinforcement learning
Seasonalities
Trading system
Neural networks

ABSTRACT

Seasonalities and empirical regularities on financial markets have been well documented in the literature for three decades. While one should suppose that documenting an arbitrage opportunity makes it vanish there are several regularities that have persisted over the years. These include, for example, upward biases at the turn-of-the-month, during exchange holidays and the pre-FOMC announcement drift. Trading regularities is already in and of itself an interesting strategy. However, unfiltered trading leads to potential large drawdowns. In the paper we present a decision support algorithm which uses the powerful ideas of reinforcement learning in order to improve the economic benefits of the basic seasonality strategy. We document the performance on two major stock indices.

1. Introduction

Seasonalities and empirical regularities on financial markets are one of the most frequently studied phenomena in the scientific literature. This is due to simple but promising assumptions, which can easily be translated into a trading strategy. Hence, only investigating this phenomenon has lost its appeal. Therefore, it is our motivation, to not simply create a trading system based on seasonalities, but to verify the signals of this strategy with an intelligent filter in order to provide a robust decision support.

The procedure for filtering is the focus of our work. We use the promising approach of reinforcement learning (RL) to realize an effective filtering. This heuristic method is often used in unstructured and complex situations and provides very good results in the field of robotics, but also increasingly in economic decision-making. Our goal is to find a policy with RL in order to filter the output signals of the basic strategy to improve the reward to risk ratios.

A novel approach is used. To link the trading decision with a reward, we use an artificial neural network (ANN). The same ANN (a simple three layer feedforward network) acts as decision support to determine the optimal parameters for future trades. We use a combination of three major research areas (RL, ANN, seasonalities). We only introduce the basics of each topic. For an in-depth insight many suggestions can be found in the corresponding section. This paper aims to demonstrate the strength of a combination of several economic and interdisciplinary methods.

Our paper is divided into the following parts. Section 2 presents the ideas and methods. Section 2.1 describes the results of a brief analysis on seasonalities. It shows the promising approach of this surprisingly simple strategy. We investigate two major indices (DAX and S&P 500) on detectable trading regularities like upward biases at the turn-of-the-month, exchange holidays and the pre-FOMC (Federal Open Market Committee) announcement drift. Section 2.2 deals with RL and the detailed description of our modified version. Section 2.3 offers a small introduction to the world of artificial neural networks. Section 3 shows a merger of the mentioned disciplines in a fully automated and self-learning trading system. In Section 4 the results are presented and discussed. Section 5 discusses possible limitations of the strategy and provides an outlook for further research. Appendix A shows the complete algorithm in pseudocode.

2. Ideas and methods

In this section we present an overview of the methods and ideas. First, previous studies about seasonalities on financial markets are

* Corresponding author. Tel.: +49 511 762 4982.
E-mail addresses: eilers@iwi.uni-hannover.de (D. Eilers), christian.dunis@hpwmg.com (C.L. Dunis), mettenheim@iwi.uni-hannover.de (H.-J. von Mettenheim), breitner@iwi.uni-hannover.de (M.H. Breitner).
URL: http://www.iwi.uni-hannover.de (H.-J. von Mettenheim).

http://dx.doi.org/10.1016/j.dss.2014.04.011
0167-9236/

introduced in Section 2.1. After that we test whether the basic approach used for our subsequent programming is performing significantly better than a random strategy. Furthermore, in Section 2.2 reinforcement learning and our modifications are described. Section 2.3 gives an insight into the field of artificial neural networks.

2.1. Seasonalities and empirical regularities

> *October. This is one of the peculiarly dangerous months to speculate in stocks in. The others are July, January, September, April, November, May, March, June, December, August, and February.*
>
> [Mark Twain (1894).]

We give a very brief overview on the existing seasonal effects literature and provide notes for an in-depth exploration of this topic. For our trading system, we focus on three interesting assumptions. Upward biases at the turn-of-the-month described by Ariel [2], during exchange holidays examined by French [10] and the pre-FOMC announcement drift investigated in a highly topical paper by Lucca et al. [20].

In the literature several other seasonalities are described, nevertheless as an illustration we only deal with the above three. A detailed list of various calendar effects with corresponding tests is offered by Hansen et al. [13]. They found significant calendar effects in most major world indices. Some of the strongest evidence has been demonstrated for small- and mid-cap indices. However, they come to the conclusion that these effects are less important since the 1990s. Sullivan et al. [30] take a more critical position on this phenomenon. It is an interesting study how methods of data mining affect the significance of seasonalities. Further references are [7] with an analysis of turn-of-the-month and pre-holiday effects, [9] with effects on the options market, [14] with an extended investigation of [2], as well as [3,16,25,27].

The question is, which assumptions are necessary to apply seasonalities in a trading system? Our approach assumes that the three aforementioned events reduce uncertainty in the market after their occurrence. E.g. fund managers often adjust their portfolios at the beginning of the month. Exchange holidays are associated with a trading break, leading to more uncertainty beforehand, which is subsequently dissolved. The FOMC results lead (regardless of their actual decision) to more planning reliability, and thus to more investments. These considerations allow the conclusion that it could be possible to benefit from these trends with a simple trading strategy.

Therefore, we wanted to test the hypothesis that these events lead to an average increase in the share indices. As a basis we used the data of the German stock index (DAX) and the S&P 500 since the year 2000 to 2012. For a test run we used a simple strategy which is easy to analyze and program. At the day before a certain event the close price of the index will be stored. This will be compared with the close price of the second day after the event. The strategy has the advantage that, with these relatively short holding periods, the risk of open trades during stock market crashes will be reduced. In practice this can be done by an index certificate. The transaction costs are neglected here. The evaluation (Table 1) showed that, in fact, a trend toward rising prices can be seen.

This approach is based on the investigations of Lakonishok et al. [16] (turn of the month effect). With their study using data from the Dow Jones Industrial Average, they have shown that significantly higher returns are possible around the turn of a month. The trading days of t_{-1} (t_0 month changes) to t_3 resulted in an increase of 0.473%, while in the average score (with the same period length), a value of 0.0612% was achieved. For an easy implementation in a decision algorithm, we apply this approach to all three event types. According to Lucca et al. [20] we shift this window one day backward (event is set one day before the actual statement) for the pre-FOMC event. They demonstrated on the basis of intraday data (1994–2011) that there is a significant upward trend in the S&P 500 on the day of the statement. Regarding the holiday effects, the literature is ambiguous. Distinction can be made between pre- and post-holiday effects [13]. We treat the holiday effects, like a turn-of-the-month (post-holiday effect).

Table 1
Price reaction to events (values in percent).

Event	Upturns DAX	Upturns S&P 500
Turn-of-the month	57.96	61.15
Pre-FOMC announcement drift	68.75	60.94
Exchange holidays	75.00	55.08
Overall	63.11	58.60
Each stock exchange day	52.32	52.94

As mentioned, we want to benefit from short holding periods to reduce the risk of being invested during stock crashes. Therefore, we limit the holding period to a maximum of two days (in Section 3, the decision algorithm is free to select one or two days). A reason for this is that the studies of Lakonishok et al. do not consider the developments of fast and volatile computer trading as a source of risk. In addition, Lucca et al. have shown that the main effect is only significant during a few hours before the FOMC-statement.

The comparison between a holding period of one or two days shows that the cumulative returns (at each event go long at close price and quit at close price on the first/second trading day after the event) are larger for a holding period of two days. For the DAX, the cumulative return of this strategy (13 years) is 75.98% (one day) or 106.64% (two days). For the S&P 500 62.42% (one day) or 63.11% (two days). In the following the unfiltered strategy is defined as a static strategy with two days holding period.

Moreover, we examined whether performance differences exist in certain months over the 13 years to see if it makes sense to add the month as a filter criterion. The result includes all events which are associated with their corresponding months (gains or losses are allocated to the month that includes the beginning of the event). It turns out that there are significant differences. Mainly at the beginning and at the end of the year the performance of this simple strategy is already very promising. Such a January effect was first described by Rozeff et al. in 1976 [27]. In the middle two quarters the strategy acts rather unprofitably. This "Sell-in-May" effect was recently investigated by Andrade et al. [1]. In fact, for a high-performance filtering it makes sense to add the month as a criterion. Nevertheless, the following chart (Fig. 1) shows that it is almost impossible to find clear rules for certain months.

As we pursue such a naive strategy, a significance test of this method seems useful to review the basic practicality of this approach. For this, one needs a comparison variable to empirically distinguish the phenomenon from a pure coincidence. In our study, we assume that not only on event days a trade is executed, but at every possible trading day. The average profit of this calculation is used as a benchmark for the actual event trades. With a simple one-sample t-test we check whether the gains for each event are significantly different from this average. The Tables 2–4 show the results. *, ** and *** indicate statistical significance at the 10%, 5%, and 1% levels, respectively.

Of course, it would be possible to filter the seasonalities manually and eliminate unprofitable events with these testing results. However, manual filtering also requires a continuous adaptation and verification. A much more elegant solution would be if a self-learning and intelligent algorithm could make these decisions by itself, a software which adapts automatically to new situations without any human influence. The research field of machine learning provides a large number of possible approaches. Our algorithm is based on the ideas of reinforcement learning.

2.2. Reinforcement learning

> *A man who has committed a mistake and doesn't correct it, is committing another mistake.*
>
> Confucius

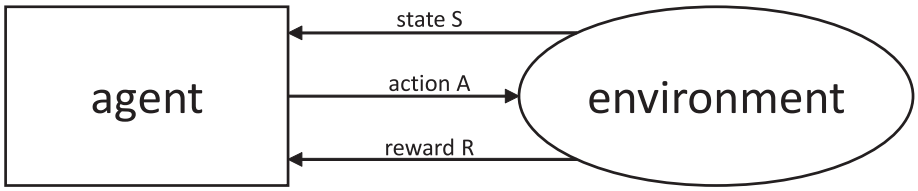

Fig. 1. Aggregate profit in each month (2000–2012).

Historical and recent financial crises have shown all too often that an attempt to press the financial markets into everlasting mathematical formulas is not profitable in the long term. Therefore the question of alternative solutions which do not blindly follow the alleged regularities arises. The aim must be to develop a dynamic system that adapts to the environment in order to deal with the increasingly complex and ever-changing modern financial markets. One approach is the technique of RL [31]. Inspired by the research field of robotics [21], where RL is already successfully applied to the most complex problems in unknown environments, the concept seems suitable for a trading strategy (Fig. 2).

It is interesting to see how RL has been used in previous studies. The most commonly used algorithm in financial applications is recurrent reinforcement learning (RRL), first described by Moody et al. [23]. The idea is to maximize a utility function (average excess return in relation to the volatility of the excess return, called Sharpe ratio), to learn automatically how to trade assets or manage a portfolio. This is made possible by a neural network with the recent price developments and the previous position as inputs. The output of the network is the position to be taken at the current time. Depending on the implementation, for example with a two-layer perceptron, long (output $= 1$), short (output $= -1$) or nothing (output $= 0$) are possible. Among others Bertoluzzo et al. [5] show the promising performance of a reinforcement learning approach with a large-scale study on the major world indices. Other interesting results can be found in the study of Nevmyvaka et al. [24] with high-frequency data for several NASDAQ stocks. A comprehensive study on FX markets was published by Gold [11]. Here, neural networks with a hidden layer were tested compared to two-layer perceptrons. Another promising approach is the fully automated trading system for FX markets by Dempster et al. [8], referred to as adaptive reinforcement learning. Here, the RRL is embedded in a three-layer structure. A risk management layer and a dynamic optimization layer complement the well-known machine learning algorithm. The idea is that a combination of different methods is more successful. As justification for the approach of reinforcement learning Dempster et al. [8] called the inability of conventional trading systems to adapt to changing market situations. This is exactly our motivation to develop a self learning trading system (Fig. 3).

To understand our approach, it is useful to look at the learning process of a human being. Human action is based on an analysis of the current environment and an impact assessment of a certain behavior. For successful learning, the feedback that is related to a previous action in a given situation must be analyzed. Humans try to improve their behavior on the basis of feedback from the environment. The best way for a human trader to learn something about the market is to analyze the gains and losses. A strategy which is obviously unprofitable will be removed from the market by every rational person. Instead, a new approach must be found and implemented. This in turn must be tested for profitability and should be adapted or discarded. Compared to normal everyday situations the feedback of this environment is very easy to interpret. The greater the profit, the better was the decision given the environmental characteristics. Here, the profit is the direct reward for an executed trade. The procedure in this particular situation will be linked with a positive experience and high profit, respectively. In long-term this leads to a preference of possible alternatives with positive linkages. In similar situations actions with bad experiences will be avoided.

So, unlike normal RL applications which try to maximize a reward function over time, we always want to maximize only the immediate reward for each individual order in this trading example. The reason for this is that, compared to applications in robotics, an executed action has no influence on future market conditions. You only need a method to assign the best action to a specific situation. For that there must be a certain scope in which an (artificial) agent can act more or less freely. Here, the tradable products or potential levers etc. are defined. At the beginning the agent starts with no prior knowledge and randomly selects a combination of parameters within the room of maneuver, to test it in the market. The gain or loss is now associated with the state-action pair (SAP). In our case, the SAP consists of the current market situation (state) and the order parameters (action). The conditions in the market are described by various indicators or historical price data. Because of that, an infinite number of states are possible. Hence, the link between the SAP (input) on the one hand and the gain/loss (output) on the other hand, must be made via an ANN using just a simple three layer feedforward network (subsection 2.3). This allows us to estimate the possible profit with nonlinear function approximation. After the first randomly executed trade the network must be trained with this first set of training data. The next trade will no longer be selected at random, but the ANN will question what kind of action for a given market situation provides the best output. All possible alternatives will be applied to the input neurons and the resulting outputs will be compared.

Table 2
Exchange holidays.

S&P 500			Dax		
Event	Profit	p-Value	Event	Profit	p-Value
M.L.K. day	−0.0120	0.3564	Easter	0.1222	0.0342**
G.W. day	−0.0620	0.1943	Labor day	−0.0330	0.2698
Good Friday	0.0399	0.2691	Christmas	0.0570	0.2104
Memorial day	0.0326	0.2943	New year	0.2336	0.0054***
Fourth of July	0.0091	0.4634			
Labor day	0.0053	0.4796			
Thanksgiving	−0.0540	0.2841			
Christmas	0.0001	0.4762			
New year	0.1586	0.0079***			
Overall	0.1176	0.3085	Overall	0.3841	0.0043***

Table 3
Turn-of-the-month.

S&P 500			Dax	
Event	Profit	p-Value	Profit	p-Value
Jan	0.0390	0.2588	0.0555	0.2937
Feb	−0.0263	0.3611	0.0248	0.4158
Mar	0.1147	0.1060	0.2000	0.0662*
Apr	0.1044	0.0246**	−0.0121	0.4060
May	0.0381	0.3173	0.0058	0.4947
June	−0.0326	0.2973	−0.0433	0.2409
July	−0.0467	0.2515	−0.1493	0.0389
Aug	−0.0300	0.3276	−0.0270	0.3947
Sept	−0.0202	0.3757	0.0090	0.4831
Oct	0.0714	0.1417	0.1226	0.1144
Nov	0.0095	0.4643	0.0904	0.1123
Dec	0.1586	0.0079***	0.2336	0.0054***
Overall	0.3798	0.0713*	0.5098	0.0786*

Table 4
FOMC and a combination of all.

S&P 500			Dax	
Event	Profit	p-Value	Profit	p-Value
FOMC	0.2815	0.0593*	0.3135	0.0639*
Combination	0.6311	0.0465**	1.0664	0.0060***

The alternative with the largest projected gain will be chosen and carried out in the market. This would result in an additional gain/loss which is also associated with the SAP within a new set of training data. This procedure is repeated continuously, so that the ANN gets more and more experience. From time to time, not the alternative with the largest output, but again random parameters are chosen. This is the exploration to test new and potentially more profitable approaches (avoiding a local-minimum-problem). Over time this gives rise to a self-learning, fully automatic, flexible trading system.

How can the idea of RL be formally described? First, these are just the basics for a general application of RL before we consider the special case of a trading system with all adjustments. In general, the task of an artificial agent is to perform an action (a_t) in a certain state (s_t) in order to move to a new state (s_{t+1}).

$$\text{State } s_t \in S \quad s_t \xrightarrow{a_t} s_{t+1}. \tag{1}$$

For this, a transition function δ is used:

$$s_{t+1} = \delta(s_t, a_t). \tag{2}$$

Furthermore, the reward must be included in the model. For this purpose one uses a function that is determined by the state and the selected action:

$$r_t = r(s_t, a_t). \tag{3}$$

This function calculates the immediate reward for an action in a numeric value. The following applies:

$$r_t > 0 : \text{positive reward} \tag{4}$$

$$r_t = 0 : \text{no reward} \tag{5}$$

$$r_t < 0 : \text{negative reward}. \tag{6}$$

The goal should be to find a policy that maximizes the long term reward by projecting an action to each state. This relationship represents the learning task of RL, which is the core of the problem.

$$\text{policy } \pi : S \rightarrow A. \tag{7}$$

This modeling results in a maximization problem. If you want to find the best policy, the weighted sum of the single rewards of a policy must be maximal. A discount factor γ is used to weight future rewards and to ensure that the sum converges. The sum converges under the condition that r is limited ($|r| \leq$ a constant B where $B < \infty$) and $0 < \gamma < 1$. A policy π^* is optimal if the following applies to all states:

$$V^{\pi^*}(s) \geq V^{\pi}(s). \tag{8}$$

The value function:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}. \tag{9}$$

Convergence:

$$\left| \sum_{t=0}^{\infty} \gamma^j r_t \right| = \sum_{t=0}^{\infty} \gamma^i |r_t| < \sum_{t=0}^{\infty} \gamma^j B = B \sum_{t=0}^{\infty} \gamma^j = B \frac{1}{1-\gamma}. \tag{10}$$

Furthermore, we assume that the problem is a Markov decision process. This means that the reward of an executed action depends only on the current state and the executed action. It doesn't matter how this situation has come about. To optimize the infinite reward under the assumption of a Markov decision process, algorithms can be used, which will not be explained in detail here. In the literature there are good analyses of dynamic programming by Bellman (1957). Here, only the basic idea of the RL should be illustrated formally in order to lay the foundation for our strategy. In the following, we can make significant simplifications of the standard model, if we transfer the technology to a trading software.

What is the difference between our method and the situation described above? We have seen that the goal of RL is to optimize long term reward. However, in trading we want to maximize the profit of each order, i.e. only the immediate reward must be considered. Thus, we avoid the typical credit assignment problem. This occurs, for example if a robot must execute many steps which provide almost no immediate reward, but which are essential for the cumulative reward. This leads to the question which of the actions were critical to the eventual reward. Avoiding this problem is a significant simplification. It is also

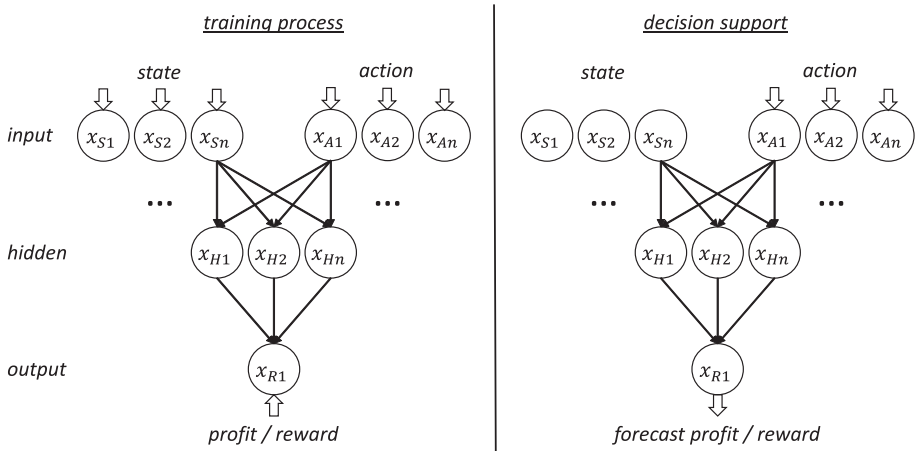

Fig. 2. Idea of reinforcement learning.

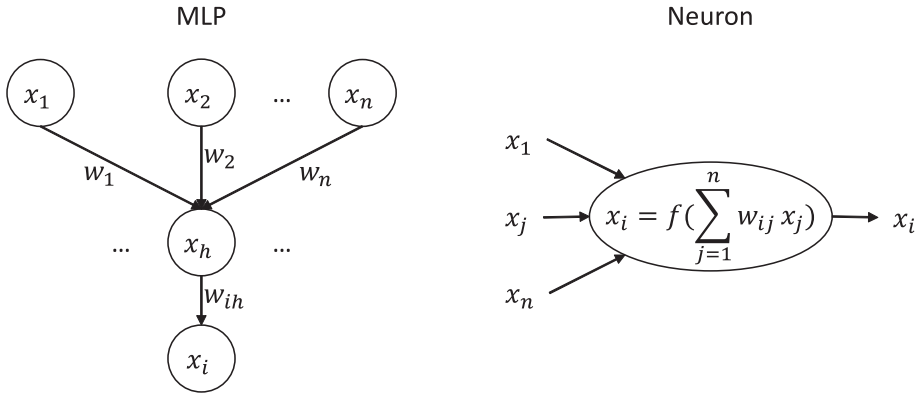

Fig. 3. The multilayer perceptron (MLP).

clear that we cannot influence the next state with our trade, because of the assumption that the trade has no effect on a competitive market situation (state). What remains is a simple function:

$$V^{\pi}(s_t, a_t) = r_t = r(s_t, a_t). \tag{11}$$

It remains only the core idea of RL to map a specific action to each state. In our case only the immediate reward is maximized. While the states already (following specification in Section 3, Trading System) defined as the market situation, the action is a combination of different order parameters. An optimal policy for our purposes looks like this.

$$\pi^*(s_t) = \operatorname{argmax}_{a_t} r(s_t, a_t). \tag{12}$$

I.e. an optimal (one-step) policy in a particular state is the argument which maximizes a reward function that depends on the state and the action, while the state is fixed and the action is optimized. Since we are dealing with financial markets with infinite state spaces, we use an artificial neural network in our simulation that allows to approximate the optimal policy π^* in a particular market situation.

2.3. Neural networks

Prediction is very difficult, especially if it's about the future.

Niels Bohr

ANNs have been used for many years in the financial world to forecast time series of stock prices [17,18]. Schocken et al. analyze their use in the context of decision support [28]. They say that in contrast to traditional DSS resources ANNs are robust against low structurability and noisy or missing data. Hence, ANNs are well suited for implementing our idea. A detailed explanation of this large area of ANN can be read in standard works such as [6]. Zhang et al. [36] (generally) and Li et al. [19] (finance) give also good overviews of the state of the art. Some interesting studies with advanced methods of forecasting using neural networks in the field of financial timeseries are provided by Sermpinis et al. [29] (e.g. Psi Sigma Neural Networks) and von Mettenheim et al. [35] (shared layer perceptrons).

This subsection briefly describes simple feedforward networks and their use in our application. Multilayer perceptrons (MLP) can be viewed in two ways. Like a decision tree, through which you can link data (input/output), or as a method for nonlinear function approximation.

In a neuron, we use the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{13}$$

With this example of function approximation, our approach can be illustrated. Our goal is to find an action, for a given state, which promises the maximum reward. This reward is a mathematical function that depends on a state vector and an action vector as described in subsection 2.2. To perform a function approximation, state and action as independent variables must be included in the input of the ANN. The dependent variable is the reward. If you train the network with enough samples, this produces an approximation of the reward function. Since there are theoretically an infinite number of states in the financial markets, the generalization of ANN approximation is well-suited for our purposes. One can achieve already good approximations of unknown patterns with a manageable number of training patterns. Fig. 4 shows the structure of such an ANN.

The input layer consists of the vector of states and actions, which should be linked to the reward/output. Every input neuron is connected to each hidden neuron (in our case three neurons) via respective weights (w). Each hidden neuron is linked to the output neuron. The forecast of the ANN is the expected profit for a given SAP. To train the ANN the mean square error of the prediction compared to the actual profit has to be minimized over all training patterns. This is done by adjusting the weights between the neurons. In our example, we use a simple backpropagation algorithm. For this, the input is applied to the neurons and passed through the ANN according to the weights of the network. At the beginning the weights are chosen randomly. The difference between the output and the actual value is the error of the ANN. The error is then propagated backwards through the ANN from the output to the input. The weights are adjusted depending on their influence on the error. The exact description is beyond the scope of this paper and can be studied in standard works as mentioned before. A formal description of the training:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}. \tag{14}$$

Δw_{ij}	the change of the weight
η	the learning rate which determines the amount of the weight changes
E	error function

Setting the learning rate is an important part of the learning process. It consists a trade-off between accuracy and speed. A high learning rate leads to larger changes, but some matching local minima may be missed. A low learning rate increases the accuracy, but slows down the learning process. We use a standard learning rate of 0.05 with good results. Furthermore, the question of the number of iterations in the learning process arises. Too many iterations lead to overfitting. Too many iterations worsen the approximation. Due to the structure

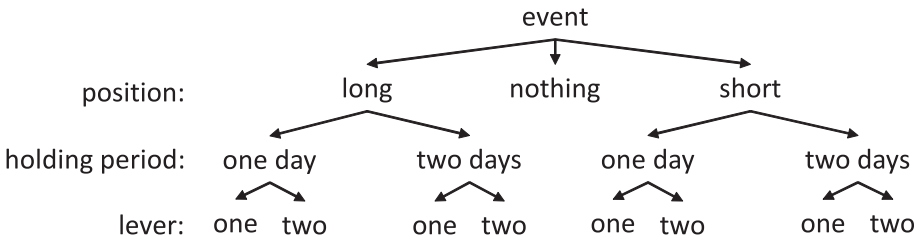

Fig. 4. Use of the multilayer perceptron in decision support.

of our model, we use an unconventional method for the determination of iterations. At each decision the number of iterations depends on the number of records already stored during training. Good approximations can be achieved by multiplying the number of stored experiences by the factor of ten. At 100 data sets this results in 1000 iterations for the learning process before the decision process.

Although one can develop their own ANNs using e.g. Matlab dedicated libraries, it is not recommended to implement these algorithms by yourself. Some suggestions for powerful software packages:

- The Neurosimulator FAUN developed by Breitner et al. [32,33], with the possibility of multi-core computing for large networks.
- The WEKA (Waikato Environment for Knowledge Analysis) open source library for Java, which includes many other machine learning algorithms alongside ANN.
- MemBrain [15] (version 05.00.03.00) which brings the possibility of a GUI for graphical modeling even complex ANN.

After training, the decision is made by applying all possible action combinations to the action neurons (while the state remains the same) and the expected profits are compared. The action which results in the highest anticipated output for a given state is carried out on the market. The result of this action is saved after closing the trade as a new data set and is part of the training process in the next round. Exceptions to this procedure exist only because of the exploration, when the action is chosen randomly from time to time in order to avoid a local minimum problem and explore possibly better strategies. The configuration of the exploration will be discussed in Section 4 on the basis of the results, and can be traced in the pseudocode Appendix A. Section 3 now describes the components of the ANN for an automated trading system.

3. Trading system

The basic idea is to use the detectable seasonalities and empirical regularities for an automated trading system. But the question is how this assumption of rising prices after a trade event can be transformed into a profitable behavior. Our research shows that it is difficult to develop firm rules. Obeying fixed rules may lead to large drawdowns. Depending on market conditions, e.g. a different leverage or a customized holding period could be useful. Inactivity or even betting on a price decline can protect against major drawdowns, too. Therefore, we tried to find a dynamic strategy with RL that can adapt automatically to changes in the market situation. As described in the subsection 2.2 about RL, we defined a room of maneuver from which the agent can autonomously choose the best settings for the trade execution. In our example, the agent can decide between long, short, and no trade. Longs and shorts can be combined with a leverage [one, two] and a holding period [one day, two days]. This leads to the parameter combinations in Fig. 5.

For a good decision, the agent has to be aware of the state of the environment and the market situation. We use eleven values to describe the market as effectively as possible. Three neurons for the actions and one neuron for the output complete the ANN, which is the core of the agent. Table 5 shows the corresponding layout.

First, three trade events are distinguished. The turn-of-the month, exchange holidays and the pre-FOMC announcement drift. Furthermore, the current OHLC values of the day, as well as the close-data of the past three trading days are added. As we trade at the close, OHLC data for the present day is available. A simple moving average (5, close) and a relative strength index (6, close) as well as the number for the current month round off the state description. This specific condition, together with the action, are the input of an ANN. This results in 14 input neurons. As feedback, this input data set is associated with the profit (output). To find a profitable strategy, traded combinations will be associated with the resulting profit and stored in training datasets (experience).

A suitable selection of the inputs of an ANN is a critical point. The investigations of Martinez et al. [22] show that an extension of the inputs does not necessarily lead to better results. In their study, the best results were achieved with 15 inputs. More inputs worsened the reward

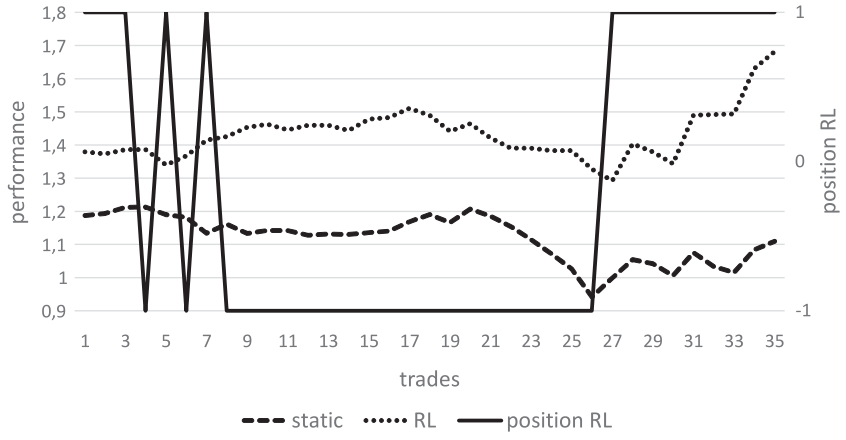

Fig. 5. Order options for the agent.

Table 5
ANN layout.

	Description	Neurons
State	Event: turn-of-the month, FOMC or exchange holidays	1
	OHLC of the present day	4
	Close-values from the past three days	3
	SMA of the last five close-values	1
	RSI of the last six close-values	1
	The number for the current month	1
Action	Position	1
	Holding period	1
	Leverage	1
Output	Immediate reward (the profit of each order)	1

to risk ratios. In our simulations we adhere to this specification (14 inputs including action variables). In the literature it is common to add the OHLC values to the models, which has proven itself in various studies [34,37]. According to our model it is necessary to use the type of the event and the current month as inputs. The close values of the last three days describe the short-term development at the exit points of the strategy. In order to include technical analysis aspects we have chosen a trend follower and an oscillator (see an example of a similar composition by Gunasekaran et al. [12]). The settings of the technical indicators are the result of a trial and error preselection process based on the DAX values. The decisive factor for the selection is the ability of the agent to find a policy as fast and reliable as possible. Because of the random exploration every simulation run is different, despite the same inputs. Therefore, we define the robustness of an agent according to its ability to achieve at least the return of the unfiltered strategy in several simulation runs. The agent with the presented inputs reached a higher return than the unfiltered version in 9 out of 10 simulation runs. The results are presented in Section 4 on the basis of best, average and worst case scenarios. All other input variants of the agent reached less than 9 out of 10 reliable runs. The tested variants include the adjustments of the SMA by 5 up to 50 included periods (in steps of five units) and the RSI by 4 up to 14 periods (in steps of two units). Variants with a stochastic oscillator, instead of the RSI, had no effect in decision making.

In the presented variant, the time required for a simulation run over 13 years is about 10 min with an eight-core CPU. This depends essentially on the number of iterations in the training process. The type of the implementation (standard packages, learning algorithms, learning rate) is also a critical time factor. It should be noted, that the training can be carried out directly after the end of an event trade, so this will have no adverse effects in practice. The decision process of an agent is made in a split second, so a decision can be executed on the market without significant time delay.

Our aim in this paper is to provide instructions for building a self-learning trading system. So far, the ideas and basic structures were explained. A detailed guide for reverse engineering of the algorithm provides the truncated pseudocode in Appendix A. This makes it possible to implement the algorithm step by step in your own applications or projects.

4. Results and discussion

The results of the RL strategy were generated by using a self-developed simulation program. The software written in Java mimics a market with real prices of the past (2000–2012) and is also able to create and train neural networks. Thus the agent can be tested realistically over several years and with different parameters (e.g. with different probabilities for exploration). For a detailed analysis learning logs were stored, which give information on how the agent works. The benchmark is a static strategy (static) that buys one day before an event at the close price and sells two days later at close price.

The following diagram (Fig. 6) provides a graphical illustration of the operation of an artificial agent.

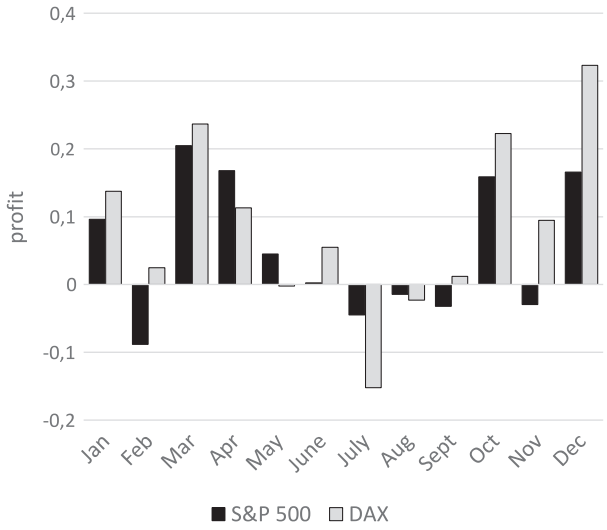

Fig. 6. Comparison of Strategies from 10-31-2000 to 04-17-2003.

You can see a detailed performance history of the strategy for the DAX. The excerpt covers the period from 10-31-2000 to 04-17-2003 (35 trades), thus including one of the largest stock crashes in recent history. In this time span, the index fell by 59%. The unfiltered strategy of seasonalities could limit the decline to 6.6%. The variant filtered with RL even achieved a rise of 21.9%. As seen in the graph, the agent adapts very quickly to the new situation and bets against the market for a certain time. The solid line shows in which direction the agent acts. 1 is a long position, a short position is a −1 and 0 means no trade. After the situation had reassured, the agent switched back to a long strategy. At the beginning of this extract, the agent had only learnt and traded the patterns from 01-01-2000 to 10-31-2000.

After this example we show a performance analysis of the entire 13 years. As described in Section 3, the agent has the possibility to choose between nine parameter combinations. Because of the exploration (at random times during the training process, the agent does not perform the action with the highest predicted reward), the results of each simulation run are different. Therefore, it makes sense to distinguish between best, worst and average cases. The case distinction is based on the realized returns. The average case is the median of the simulation runs.

Table 6 shows the performance differences of the two strategies for the DAX with a total of 243 trading events. The statement shows the total profit/loss over 13 years (return) and the annualized return, respectively. You can also see the maximum drawdown of each simulation run.

Here, for comparison, a buy-and-hold strategy (b&h) for this period is specified. This clearly shows the superiority of the seasonality strategy, which significantly reduces the risk of great crashes due to the few but effective trades.

As mentioned in Section 3, the rate of exploration in RL can be changed easily. For our results, we have used a simple random function. The program generates a random integer between 0 and the number of training runs. If this random number is less than a defined threshold, the order options are set randomly (exploration). Otherwise, the parameters are set by the agent. This is called exploitation, because the

Table 6
DAX and benchmark (values in percent).

	b&h	Static	Best	Average	Worst
Return	9.35	106.64	232.83	173.14	−33.92
Ann. ret.	0.69	5.74	9.69	8.04	−3.14
Max. dd.	72.68	22.35	14.50	14.32	61.94

Table 7
S&P 500 and benchmark (values in percent).

	b&h	Static	Best	Average	Worst
Return	−4.55	63.11	133.99	80.78	−4.80
Ann. ret.	−0.36	3.84	6.76	4.66	−0.38
Max. dd.	56.78	14.93	9.52	16.00	30.93

parameters with the highest projected output will be chosen. This approach means that as long as the number of training runs is smaller than the defined limit, only exploration is performed. If the limit is reached, the probability of exploitation increases with each new training run.

If the share of exploration is set very low, relative to the total running time, it may happen in exceptional cases, that the agent does not find a proper policy. This can lead to unpredictable losses in the worst case. This can be prevented by a high rate of exploration (a great threshold) at the beginning. Nevertheless, our investigations have shown that it is useful to start with 100% of exploration but only for the first three or four training runs so that the probability decreases rapidly. An optimization of the RL parameters is left open for future research.

Table 7 shows the performance difference for the S&P 500 from 2000 to 2012 with a total of 314 trading events.

It is obvious that the results are worse in the average case, compared to the DAX. This could be due to the fact that the basic strategy already works less well. Nevertheless, in the best case, a marked improvement of the reward to risk ratios was achieved. So it is not impossible to find a well acting agent for the S&P 500, but one can see how important a properly working basic strategy for RL is. The results show that a significant performance improvement is possible. A greater profit and a smaller maximum drawdown can be achieved by filtering with RL, so the reward to risk ratios becomes more attractive.

5. Conclusion, limitations, and future research

In this paper three major research fields are combined in an intelligent, self-learning and fully automated trading system. The basis is the promising strategy of seasonalities and empirical regularities in financial markets which has been described and discussed in the literature for three decades. For the period from 2000 to 2012, the significance of upward biases at the turn-of-the-month, during exchange holidays and the pre-FOMC announcement drift could be confirmed in many cases. Based on this study, a self-learning decision support algorithm was developed which filters this strategy by using reinforcement learning and artificial neural networks, in order to achieve even better results. The software is able to act autonomously as an artificial agent on the financial markets.

Despite the performance of our artificial agents, there is much room for improvement. We have shown that a small number of possible order options for the agent are sufficient to achieve a significant improvement of the reward to risk ratios. An expansion of the possible options might improve the outcome. For example, the freedom of action could be expanded to the product choice, so that the best stock index is automatically selected. A significantly greater exploration will then become necessary. To realize this, other strategies could be used, which have a higher trading frequency. This would accumulate more experience in a shorter time, which leads to a faster adjustment. A study in the high-frequency area might produce interesting results. But this necessitates tick-data, which leads to a much larger simulation effort. Nevertheless, further optimization problems remain which are discussed in the literature for many years.

- What ratio between exploration and exploitation is optimal? [23,31]
- Which inputs are best to describe the current state of the market? [12,17,37]
- How should the ANN be configured? [18,28,32]
- Should older training patterns be replaced after a certain time? [11,34]
- How much patterns are necessary to reduce the risk of underfitting and overfitting respectively? [4,6]

Consideration should be given whether agents with different experience can be used simultaneously. Agents act very differently in similar situations. It seems sensible to use several agents for a decision process to make the system more robust. A majority decision may improve the behavior.

Another idea concerns the determination of the leverage. In our example, it is chosen as a discrete action (one or two) by the agent. You might consider using the projected profit of the ANN to set a continuous leverage, i.e. the higher the predicted profit, the higher the leverage for this trade. These are optimization problems with a large room for improvement by future research.

It would also be interesting to test the predictive power of the algorithm in forecasts of the direction of change in financial time series and beyond [26].

Acknowledgment

We would like to thank the anonymous reviewers for their interesting and valuable comments to improve the quality of this paper.

Appendix A

Listing 1: Self-Learning Trading System

```
//Definition
double state[11]; int action[3]; double output;
double ann[15][#]; // # training data sets for ANN
//Settings
Create a feed-forward network:
14 inputs, 1 output, # hidden layer;
Randomize neural network;
//Event loop
if (date is equal trade event){
  //set state
  state[0 to 10] = [Event_numerical, open, high, low,
  close_current, close(t-1), close(t-2), close(t-3),
  SMA(5,close), RSI(6,close), month_numerical];
  //exploration or exploitation
  if (random integer between 0 and # training data sets
  < defined threshold){
    action[0] = random(position_numerical);
    action[1] = random(holding period);
    action[2] = random(lever);}
  else{
    put state[] at ANN-Input;
    for (queries < # possible combinations){
      put action[#,#,#] at ANN-Input;
      ANN perform think step;
      save action with the greatest output;}}
  //trade
  perform trade with the stored action;
  ann[][#] = state[], action[], profit;
  ANN training with ann[][];}
else{perform no action;}
```

References

[1] Sandro C. Andrade, Vidhi Chhaochharia, Michael E. Fuerst, Sell in May and go away just won't go away, Financial Analysts Journal 69.4 (July/August 2013) 94–105.
[2] Robert A. Ariel, A monthly effect in stock returns, Journal of Financial Economics (March 1987) 161–174.
[3] Christina V. Atanasova, Robert S. Hudson, Technical trading rules and calendar anomalies — are they the same phenomena? Economics Letters 106 (February 2010) 128–130.
[4] Eric B. Baum, David Haussler, What size net gives valid generalization? Neural Computation 1 (Spring 1989) 151–160.
[5] Francesco Bertoluzzo, Marco Corazza, Making financial trading by recurrent reinforcement learning, Knowledge-Based Intelligent Information and Engineering Systems, September 12–14 2007. 619–626.
[6] Christopher M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[7] Charles Bram Cadsby, Mitchell Ratner, Turn-of-month and pre-holiday effects on stock returns: some international evidence, Journal of Banking & Finance (June 1992) 497–509.
[8] M.A.H. Dempster, V. Leemans, An automated FX trading system using adaptive reinforcement learning, Expert Systems with Applications 30 (3) (April 2006) 543–552.
[9] Amy Dickinson, David R. Peterson, Expectations of weekend and turn-of-the-month mean return shifts implicit in index call option prices, Journal Of Financial And Strategic Decisions 8.3 (1995) 69–76.
[10] Kenneth R. French, Stock returns and the weekend effect, Journal of Financial Economics (March 1980) 55–69.
[11] Carl Gold, FX trading via recurrent reinforcement learning, Proceedings of the 2003 IEEE International Conference on Computational Intelligence for Financial Engineering, March 20–23 2003, pp. 363–370.
[12] M. Gunasekaran, S. Anitha, S. Kavipriya, Neuro fuzzy based stock market prediction system, Second National Conference on Signal Processing, Communications and VLSI Design — NCSCV'10 ANNA UNIVERSITY COIMBATORE, May 2010, pp. 354–359.
[13] Peter Reinhard Hansen, Asger Lunde, James M. Nason, Testing the Significance of Calendar Effects, Federal Reserve Bank, Atlanta, January 2005.
[14] Jeffrey Jaffe, Randolph Westerfield, Is there a monthly effect in stock market returns?: evidence from foreign countries, Journal of Banking & Finance (May 1989) 237–244.
[15] Thomas Jetter, MemBrain, (Website) http://www.membrain-nn.de/ (visited on August 25th 2013; version 05.00.03.00; DLL version 04.00.01.00).
[16] Josef Lakonishok, Seymour Smidt, Are seasonal anomalies real? A ninety-year perspective, The Review of Financial Studies (Winter 1988) 403–425.
[17] Monica Lam, Neural network techniques for financial performance prediction: integrating fundamental and technical analysis, Decision Support Systems (September 2004) 567–581.
[18] William Leigh, Russell Purvis, James M. Ragusa, Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: a case study in romantic decision support, Decision Support Systems (March 2002) 361–377.
[19] Yuhong Li, Weihua Ma, Applications of Artificial Neural Networks in Financial Economics: A Survey, International Symposium on Computational Intelligence and Design (ISCID), 1, October 2010, pp. 211–214.
[20] Staff Report, Federal Reserve Bank of New York, No. 512 (2011), Journal of Finance (2014) (forthcoming).
[21] Sridhar Mahadevan, Jonathan Connell, Automatic programming of behavior-based robots using reinforcement learning, Artificial Intelligence (June 1992) 311–365.
[22] Leonardo C. Martinez, Diego N. da Hora, Joao R. de M. Palotti, Wagner Meira Jr., Gisele L. Pappa, From an artificial neural network to a stock market day-trading system: a case study on the BM&F BOVESPA, Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA, June 2009, pp. 2006–2013.
[23] John Moody, Matthew Saffell, Learning to trade via direct reinforcement, IEEE Transactions on Neural Networks 12 (4) (July 2001) 875–889.
[24] Yuriy Nevmyvaka, Yi Feng, Michael Kearns, Reinforcement learning for optimized trade execution, Proceedings of the 23rd International Conference on Machine Learning (ICML 2006), 2006, pp. 673–680.
[25] Joseph P. Ogden, Turn-of-month evaluations of liquid profits and stock returns: a common explanation for the monthly and January effects, The Journal of Finance 45 (September 1990) 1259–1272.
[26] M. Hashem Pesaran, Allan Timmermann, A simple nonparametric test of predictive performance, Journal of Business & Economic Statistics (October 1992) 461–465.
[27] Michael S. Rozeff, William R. Kinney Jr., Capital market seasonality: the case of stock returns, Journal of Financial Economics (October 1976) 379–402.
[28] Shimon Schocken, Gad Ariav, Neural networks for decision support: problems and opportunities, Decision Support Systems (June 1994) 393–414.
[29] Georgios Sermpinis, Christian Dunis, Jason Laws, Charalampos Stasinakis, Forecasting and trading the EUR/USD exchange rate with stochastic Neural Network combination and time-varying leverage, Decision Support Systems (December 2012) 316–329.
[30] Ryan Sullivan, Allan Timmermann, Halbert White, Dangers of data mining: the case of calendar effects in stock returns, Journal of Econometrics (November 2001) 249–286.
[31] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 1998.
[32] Hans-Jörg von Mettenheim, Advanced Neural Networks: Finance, Forecast, and Other Applications, Leibniz Universität Hannover, Germany, 2010.
[33] Hans-Jörg von Mettenheim, Michael H. Breitner, Robust decision support systems with matrix forecasts and shared layer perceptrons for finance and other applications, ICIS 2010 Proceedings, 2010, p. 83.
[34] Hans-Jörg von Mettenheim, Michael H. Breitner, Forecasting and trading the high-low range of stocks and ETFs with Neural Networks, Engineering Applications of Neural Networks 311 (2012) 423–432.
[35] Hans-Jörg von Mettenheim, Michael H. Breitner, Robust forecasts with shared layer perceptrons, Proceedings of the 17th International Conference on Forecasting Financial Markets, May 26–28 2010.
[36] Guoqiang Zhang, B. Eddy Patuwo, Michael Y. Hu, Forecasting with artificial neural networks: the state of the art, International Journal of Forecasting (March 1998) 35–62.
[37] Y.-Q. Zhang, S. Akkaladevi, G. Vachtsevanos, T.Y. Lin, Granular neural web agents for stock prediction, Soft Computing (August 2002) 406–413.

Dennis Eilers is a Bachelor Student at Leibniz University of Hanover, Germany. His research interests include machine learning algorithms for intelligent trading strategies.

Christian Dunis is an emeritus professor of Banking and Finance from Liverpool John Moores University. He currently works as Risk Manager at a Swiss private bank. His research interests include quantitative trading strategies and artificial intelligence.

Hans-Jörg von Mettenheim is a professor for Decision Support Systems at Leibniz University of Hanover, Germany. His research interests include complex systems and forecasting.

Michael H. Breitner is a professor for Information Systems Research at Leibniz University of Hanover, Germany. His research interests include artificial intelligence, especially artificial neural networks.