Reinforcement Learning in Online Stock Trading Systems

Stuart Duerson
School of Electrical & Computer Eng.
Georgia Institute of Technology
sduerson@ece.gatech.edu

Farhan Saleem Khan
College of Computing
Georgia Institute of Technology
farhan@cc.gatech.edu

Victor Kovalev
College of Computing
Georgia Institute of Technology
gte190z@mail.gatech.edu

Ali Hisham Malik
College of Computing
Georgia Institute of Technology
hisham@cc.gatech.edu

Abstract

Applications of Machine Learning (ML) to stock market analysis include Portfolio Optimization, Investment Strategy Determination, and Market Risk Analysis. This paper focuses on the problem of Investment Strategy Determination through the use of reinforcement learning techniques. Four techniques, two based on Recurrent Reinforcement Learning (RLL) and two based on Q-learning, were utilized. Q-learning produced results that consistently beat Buy and Hold strategies on several technology stocks, whereas the RRL methods were often inconsistent and require further investigation.

1 Introduction

1.1 Research background

Stock market analysis has been one of the most actively pursued avenues of machine learning research and application. An extensive survey of the most recent literature in the related fields exposed Portfolio Optimization, Investment Strategy Determination, and Market Risk Analysis as three major trends in the utilization of ML approaches within the context of finance.

Portfolio Optimization focuses on the correlative properties of stock market data (stock indexes in particular - both individual and cumulative) in order to extract mutual dependency (or independency) information. Such results are then utilized to optimizing investment diversification strategies for efficient risk-management in asset allocation. The proposed solutions to this problem range over a wide range of clustering techniques, such as the commonly used random matrix theory ([5], [7]), chaotic map synchronization [8], and other less standard approaches such as Potts magnetization model ([9]) or Transfer Entropy [10].

Investment Strategy Determination addresses financial forecasting based on financial index analysis (including stocks, bonds, foreign exchange rates, interest rate, etc.) for the purposes of investment decision-making. There are two major sub-problems in this area. The first one is the challenge of creation of a Forecaster system for actual prediction of future index behavior. Various Neural Network approaches are by far the most commonly taken route in the related literature – represented by works such as [11] and [12], with a survey of earlier approaches outlines in [13]. However, a plethora of alternatives exist, some examples being Support Vector Machines [4], Genetic Algorithms [14] and statistical analysis [15]. Furthermore, some exploration has been done towards extraction of the less-obvious financial data traits – such as stock market crash prediction (although mostly theoretical for the lack of substantial real test data). Various hybrid approaches have also been introduced, such as a method combing template matching

(pattern recognition concept) with feed-forward neural network introduced by Leigh W., Paz M., and Purvis R. [3].

The second aspect of Investment Strategy Determination is the development of the Stock (or other asset) Trader system that makes intelligent decisions as to the market investment strategy based on the predictions from the Forecaster system. This problem has not been addressed nearly as thoroughly as the forecasting side at this stage of investment strategy development in practice. However, there are a few exceptions, the most interesting of which is [1], which explores the possibilities of various forecaster and trader system combinations. Interestingly, the authors of this work advocate that the most efficient way to implement a complete Trading System is through merging the Forecaster and the Stock Trader into a single Reinforcement Learner (RL). It is proclaimed that such treatment of the system as a whole eliminates the forecasting bottleneck that often hinders the performance of the Trader and an implementation of the Trading System as a single Recurrent Reinforcement Learning Neural Network (RRLNN). This approach enables them to evaluate each action of the system in terms of actual profit, allowing in turn for negligence of prediction of actual stock values. A similar notion is also exhibited in an earlier work by Ralph Neuneier [2], who utilized Q-learning in order to create an intelligent Trading System.

Finally, the Market Risk Analysis concentrates on the evaluation of the risk-factors involved in various investment options, such as expected return and volatility. An example of an overall market risk evaluation system is described in [16]. A comparison of performance of Genetic Algorithm to that of GARCH and RiskMetrics – both being commonly used risk-assessment models – applied to predict exchange rate volatility can be found in a work by Christopher Neely and Paul Weller [17].

1.2 Problem & approach

We focus our attention on the Investment Strategy Determination field of ML in stock market. In particular, we explore the relative effectiveness of various combined reinforcement learning based trading systems. Traditionally, the prediction of the stock behavior and the analysis of the optimal course of action are separated into independent Forecaster and Trader system, both implemented as a stand-alone machine learning entity. However, as discussed by Moody et al in [1], this approach limits the performance of the trader with the forecasting bottleneck. This bottleneck can be eliminated via combination of both systems into a single unit. Since the prediction process is eliminated from the environment, the system becomes a pure action based trader – which is why the majority of research in this area utilizes reinforcement learning. While providing for rather satisfactory performance, such novel approach to creation of singular trading systems is largely unexplored relative to the rest of the field. So, we explore different RL techniques as means to creating of efficient single-unit action trader:

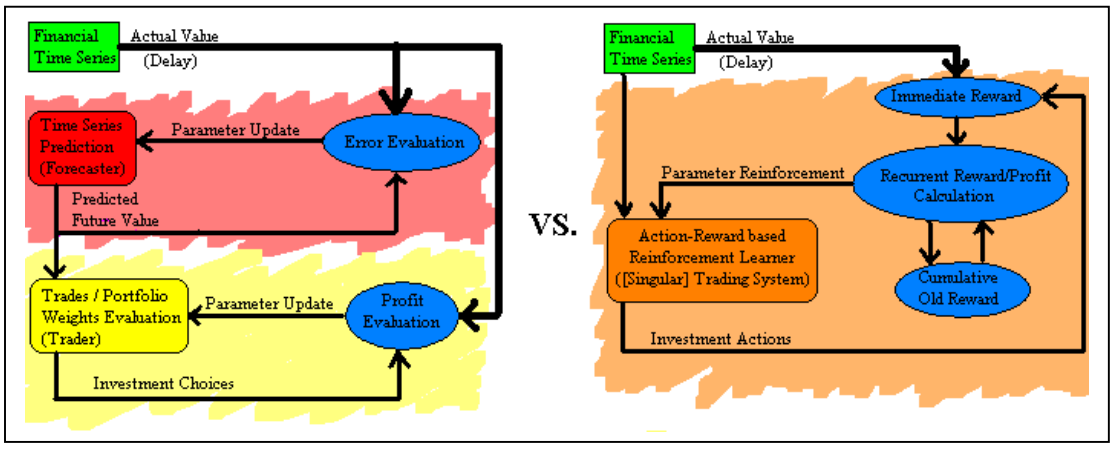

Figure 1: Comparison of conventional (left) and newly proposed (right) trading system implementation approaches.

We further explored the effectiveness of a trading system using chart analysis to generated trading signals. Comparison of trading system based on chart analysis with reinforcement learning based systems provided some interesting results.

2 Implementation

A total of four different trading systems designs were implemented and compared. The first system is a pure Q-learner implementation of the Trading System, which utilizes similar state-action paradigm with the one defined by Ralph Neuneier, with an additional incorporation of current financial time series derivative as an extra state discriminator. Such addition allows for a more robust distinction between similar stock-valued points at the bullish (going up) and bearish (going down) trends of the indexes under consideration. The second system is our own implementation of a short-term discounted history reinforcement learner. The third system is a Recurrent Reinforcement Learning Neural Network (RRLNN). It is created using the Sharpe Ratio based reward and weight update calculations proposed in [1]. Lastly, a trading system trained to predict the turning points in the market and generating trading signals based on these was implemented. The performance of the four different implementations is analyzed with each other, as well as in relation to the baseline investment strategy of buy-and-hold.

Adjusted close prices of stock indices were used as the input to the trading system both for training as well as actual trading. Experimentation with macroeconomic variables may have been useful, however the availability of macroeconomic data is a bottleneck for such experimentation. A detailed description of each system follows.

2.1 Q-Learning

The following is a description of our direct Q-Learning based trader. The approach was based on the model described in "Optimal Asset Allocation using Adaptive Dynamic Programming" (Ralph Neuneier, 1997) - applied to a single-stock trader paradigm and extended to a slightly more robust state system.

The trading environment is represented by a finite set of states and actions. Each state consists of the combination of current stock index value, current investment status, current asset value, and current index gradient. The current stock index value, C_i, as well as the current asset value A_i are both discretized in order to provide for a finite number of discrete states. The current investment status CIS_i is reflective of whether our trader is holding its assets in monetary or invested(stock) form. For the sake of optimal effectiveness, we are using a model with the assumption that if the trader chooses to invest, 100% of its current assets are used. Finally, the current index gradient provides for a distinction between otherwise two identical states based on whether the stock is rising or falling for each possible combination of other parameters. Thus the state space of the system is described as the combination of all possible permutations of the following state model.

$$\begin{aligned} \text{State } S_i = \{ & C_i \in [\text{Assets}_{MIN}, \text{Assets}_{MAX}], \\ & A_i \in [\text{StockIndex}_{MIN}, \text{StockIndex}_{MAX}], \\ & CIS_i \in \text{InvestingAssets } \mathbb{OR} \text{ StayingOutOfMarket}, \\ & s_i \in \text{Bullish } \mathbb{OR} \text{ Bearish}\} \end{aligned} \quad (1)$$

Furthermore, the action space of the system is defined as

$$a_i \in \text{InvestInStock } \mathbb{OR} \text{ WithdrawFromMarket} \quad (2)$$

At each time-step, the current state S is investigated across all possible actions a_i and the optimal evaluations of corresponding next states S` – each in turn investigated over all actions. So, we update the state table based on the following equation:

$$Q(S, a_i) = R(S, a_i) + f^*(\text{ Max}_{[\boldsymbol{s} < S`]}\{ Q(\underline{\boldsymbol{S}}, a_i) \}) \quad (3)$$

Where $R(S, a_i)$ is an immediate reward calculated as

$$R(S, a_i) = (C_{i+1} / C_i) * (A_i - w) - A_i \quad (4)$$

if a_i suggests investment – where w is the transaction cost. If a_i suggests holding [non-investment] and the CIS_i is also holding, then R(S, a_i) is 0. On the other hand, if a_i suggests holding [non-investment] and the CIS_i is investment, then R(S, a_i) = w to accommodate for the cost of exiting the market. The transaction cost can be defined as a constant or as a function of time, stock values, or asset values. However, for comparative analysis vs. other approaches, a transaction cost of 0 has been used in all approaches.

2.2 Discounted History Reinforcement Learner

The Discounted History Reinforcement Learner is similar to the Q-learning approach described above with the exception that the implementation uses a pattern recognition based approach to estimate the reward for each action. It uses the discounted differences between the adjacent values in the immediate history to calculate the reward values for each action. See equation 5.

Again, for comparative analysis, 0 transaction cost has been used. At each time step i, it estimates the reward of an action using previous $k+1$ real stock values using following equation:

$$r_{i+1} = s_i + (s_i - s_{i-1}) + \alpha(s_{i-1} - s_{i-2}) + \alpha^2(s_{i-2} - s_{i-3}) + \alpha^3(s_{i-3} - s_{i-4}) + \ldots + \alpha^k(s_{i-k} - s_{i-k-1}), \quad (5)$$

where $1/4 \le \alpha \le 1$

The equations (1), (2), (3) & (4) have been used to estimate the portfolio values, and the amount of stock to be bought and sold at any time.

2.3 Recurrent Reinforcement Learning Neural Network

The third implementation is a Real Time Recurrent Learner (RTRL) which was modified to become a reinforcement learning RRLNN. This approach similar to the one described by Moody et al [1]. A further description of the algorithm follows.

Return on investment at time t is referred to as R_t. This is the return resulting from the combination of trading decisions at time t-1 and time t. At each time step, a trading signal, $F_t \in \{-1, 0, 1\}$, is produced by the system, corresponding to the actions sell, hold, and buy.

In this context, we define an economic utility function U, which we are trying to maximize. In this case it is the profit or simply the sum of all R_t's up to the present point in time. Also, let θ be the set of the trading system parameters, which are the neural network weights. We can define the gradient of $U(\theta)$ as :

$$\frac{dU_t(\theta)}{d\theta} = \frac{dU_t}{dR_t}\left\{\frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta}\right\} \quad (6)$$

The problem then amounts to an on-line stochastic gradient ascent of the above gradient. The problem is complicated by the existence of several derivatives in the above equation. Firstly, the on-line computation of $\frac{dU_t}{dR_t}$ requires the use of the Sharpe ratio [1], which is a recurrence relation describing the moments of the distribution of returns R_t. Secondly, the on-line computation of the total derivative $\frac{dF_t}{d\theta}$ requires the use of real-time recurrent learning (Williams and Zipser 1989). The above gradient is useful in updating the system parameters through the equation:

$$\Delta\theta_t = \rho\frac{dU_t(\theta)}{d\theta_t} \quad (7)$$

Where ρ is constant defined as the learning rate**.**

For computing the utility of a trade and feeding it to the network, the differential Sharpe ratio is used. The Sharpe ratio is a risk adjusted measure of return and is used as the utility function as it is a path dependent performance function facilitating recursive updating, giving preference to recent return, and is computationally cheap [1]. It is defined as,

$$S_t = \frac{Average(R_t)}{Std\_dev(R_t)} \tag{8}$$

To speed the convergence of the learning process and adapting to the evolving market structure, it is advantageous to use exponential moving estimate of the Sharpe ratio. This differential Sharpe ratio is defined as,

$$D_t = \frac{dS_t}{d\eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_t - A_{t-1}^2)^{\frac{3}{2}}} \tag{9}$$

Where,

$$A_t = \eta R_t + (1-\eta)A_{t-1} \tag{10}$$

$$B_t = \eta R\text{^}2_t + (1-\eta)B_{t-1} \tag{11}$$

In the above equations, η is a parameter which represents the degree of influence the return values at time *t-1* have on the sharp ratio at time *t*.

For training the system, a cash account was specified at the beginning of the session, which was used to buy stock according to the trading signal F_t . Risk-free borrowing was assumed, which allowed the cash account to go negative while the borrowed money was used to buy stock to add to the stock account. At the end of training, stock account was liquidated and added to the cash account. No transaction costs were assumed.

2.4 Turning Point Indicator

This trading system was inspired by the proposal in Joone[2]. In this approach, the trading system consisted of a neural network that was trained to detect the peaks and troughs in the price of stock index. Input to the trading system is a temporal window W_1 of previous adjusted close price values along with a set of corresponding moving averages of window width of W_2 and W_3[1]. All the values were normalized between 0 and 1 before being fed to the neural network. The training of the trading system is done in off-line mode.

During actual trading period a buy signal is issued if the output of neural network is less than T_b[2], and a sell signal is issued if output is greater than T_s[3]. The overall architecture of the trading system is similar in spirit to the architecture discussed in [1].

3 Results

Real Time Recurrent Learning: The trading signals returned by this method tended to occur in "sprees" (e.g. the learner would generate buy signals for 100 time steps). This "saturation" behavior could be mitigated by altering the parameters of the learner (e.g η, etc**.**), but the combination of parameters necessary often depended on the particular stock's time series. Another possible problem was the formula for calculating the return R_t and its differential with respect to the previous forecast and the current forecast.

---

[1] W_1, W_2 and W_3 were used as 5, 15, and 15 respectively.

[2] T_b is the threshold value for generating a buy signal. This was kept to 0.3 in our experiments.

[3] Ts is the threshold value for generating a sell signal. This was kept to 0.7 in our experiments.

4 The reason for Q-learner results appearing "rounded-off" is a consequence of asset and index value discretization for the purposes of finite number of states definition via Q-learning table. The training data was also used as test data.

Since previous forecast and current forecast are not directly differentiable, some approximation functions were experimented with in two separate efforts. A well tuned RRL neural network remains a focus for future research.

Results from one implementation of RRL network design are shown below in Figure 2.

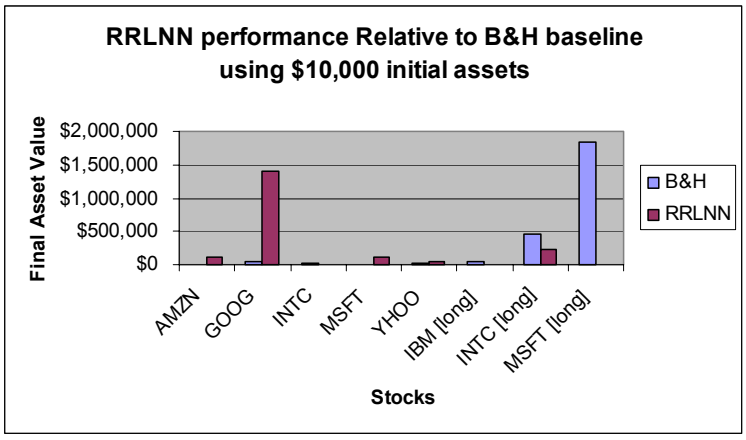

Figure 2: Comparison of returns of Buy and Hold and RTRL techniques .

The trading system tended to produce two sorts of results: 1) long sprees of shorting or buying a stock for a long time, producing a large deficit in its stock or cash accounts which would be repaid at the end resulting in a net profit, or in debts from buying or selling which could no be repaid and resulted in a negative end cash value. In either case the results were extreme as seen in the chart (negative results represented as 0).

Turning Point Indicator system was successful in beating buy and hold when training was performed on short periods of time. The performance, however, deteriorates when training is performed on longer runs. This observation suggests the shifts in market behavior. These shifts in market behavior are difficult to track for a neural network trained on only the stock price value. Yet, the successful trading simulations for short periods show the effectiveness of chart analysis. Further experimentations with the incorporation of macro-economic variables and training of neural network to detect other trends besides the turning points remains a motivation for future research

Discounted History Leaner provided for the most consistent results of all the approaches – consistently beating the baseline B&H values by a small-to-medium percentage. (Although it did require some manual adjustment of system parameters for each individual stock.)

The important parameter to consider for successful performance of PRL on financial time series is Maximum Acceptable Percentage Loss (MAPL) – the cutoff value, should the percentage loss calculated over a single transaction go below which, the trading system keeps out of the market. The chart below represents the performance of the trading system on a sample of Intel (INTC) stock from Aug 19, 2004 to November 15, 2005 as a function of MAPL. The below data was gathered with $1000 initial asset value, 0.1$ fixed transaction cost and 0.01$ variable transaction cost per stock traded.

Q-learning provided for a slightly less stable arrangement of outcomes then the pattern recognition learner. While on some data series the results obtained were extremely good – as high as 200% compared to the B&H strategy – on others the system was slightly below the baseline. This was due to the high amount of adjustable parameters – in particular index discretization and asset discretization degrees (bin sizes) – which require extended manual tweaking and customization for each given data series. For example, with Google stock ranging from approximately $100 to $300, it could be discretized into 5 or 10 dollar bins, while for Intel stock ranging from $21.11 to $25.37 a significantly lower bin size is necessary. Noteworthy, using very high precision discretization settings on high-range stocks such as Google is unfeasible due to Q-state table increasing exponentially.

In fact, the memory usage proved to be the single biggest issue with utilization of Q-learning in stock trading systems. As the stock index and investment assets range increase, one is soon forced to lower the precision of the algorithm beyond reason – limiting the ability to generate sensible results. Thus, we were unable to process some of the longer historical data series with large profit variations due to the lack of

sufficiently large memory. However, the running time of Q-learning is fairly fast in cases where the system does have the memory to initialize the state table (3-5 seconds to process 16000 time steps of data).

3.1 Performance comparison across all algorithms

The table below enumerates the performance of our algorithms on 8 different financial data series, as well as that of the baseline B&H strategy. All of the systems assumed initial investment fund of $10,000.

Table 1: Comparative algorithm performance using initial assets value of $10,000
(Results using best combinations of parameters)

Stock Name	Adjusted Closing Value for First Day	Adjusted Closing Value for Last Day	Buy & Hold Result	Q-learner based Trading System [4]	Discounted History based Trading System	Turning Point Indicator	RRLNN
AMZN	$ 38.24	$ 42.53	$ 11,122	$ 16,500	$ 16,241	$ 16,718	$1080855
GOOG	$ 100.25	$ 396.97	$ 39,598	$ 83,750	$ 44,241	$ 22,308	$14027432
INTC	$ 21.11	$ 25.37	$ 12,018	$ 10,500	$ 12,891	$ 14,090	$0
MSFT	$ 24.35	$ 27.37	$ 11,240	$ 10,250	$ 12,140	$ 12,649	$1076270
YHOO	$ 29.01	$ 38.45	$ 13,254	$ 14,750	$ 14,491	$ 14,220	$442808.2
IBM[long]	$ 20.31	$ 89.11	$ 43,874	RAM-out	$ 78,951	$ 23,102	$0
INTC[long]	$ 0.57	$ 26.86	$ 471,228	RAM-out	$ 488,874	$ 10000*	$2337842
MSFT[long]	$ 0.15	$ 27.75	$ 1,850,000	RAM-out	$ 1,972,120	$ 10000*	$0

The percentage final asset values of the algorithms in terms of the baseline strategy can be seen below.

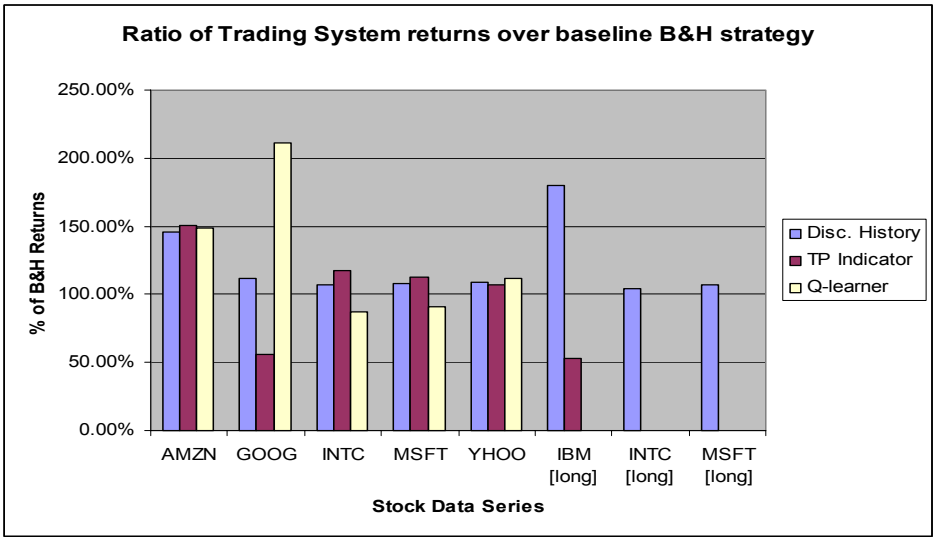

Figure 3: Composite comparison of Returns of Buy and Hold and other trading systems.

4 Conclusions

The behavior of recurrent reinforcement learner needs further analysis. The technical justification seemed most rigorous in the literature for this method, it seems deceiving that simpler methods produced more consistent results. It is observed that the performance of turning point indicator is generally better during short runs of trading which shows the validity of charting analysis techniques as used by professional stock traders. Training of neural networks with other trend predicting signals such as stochastics and relative strength index, and other filtering techniques such as using momentum values of stock price as input to the system seem interesting venues of research. In longer runs, the reinforcement based systems outperformed

turning point indicator system. This suggests that a cascaded model of a trading system may provide better performance and remains to be explored. Experimentation with additional macro-economic variables is also desirable for future.

Acknowledgements

Advisor: Dr. Alexander Gray, *College of Computing, Georgia Institute of Technology*.

Instructor: Dr. Charles Isbell, *College of Computing, Georgia Institute of Technology*.

References

[1] J. Moody, L. Wu, Y. Liao, and M. Saffell (1998): "Performance Functions and Reinforcement Learning for Trading Systems and Portfolios;" Journal of Forecasting, Vol. 17, pp 441-470.

[2] Neunier, R. (1996), "Optimal asset allocation using adaptive dynamic programming", in D. Touretzky, M. Mozer & M. Hasselmo, eds, "Advances in Neural Information Processing Systems 8", MIT Press.

[3] Leigh W., Paz M., and Purvis R. (2002): "An analysis of a hybrid neural network and pattern recognition technique for predicting short-term increases in the NYSE composite index" Omega Vol. 30, Number 2, pp 69-76(8).

[4] Wei Huang, Yoshiteru Nakamori, Shou-Yan Wang (2005) Forecasting stock market movement direction with support vector machine Source" Computers and Operations Research, Vol. 32, Issue 10, pp 2513-2522.

[5] Werbos, P. (1990), "Back-propagation through time: What it does and how to do it", IEEE Proceedings 78(10), 1550-1560.

[6] Vasiliki Plerou, Parameswaran Gopikrishnan, Bernd Rosenow, Lu´ıs A. Nunes Amaral, Thomas Guhr, H. Eugene Stanley (2001) "A Random Matrix Approach to Cross-Correlations in Financial Data"

[7] Szil´ard Pafka, Marc Potters, Imre Kondor (2004) "Exponential Weighting and Random-Matrix-Theory-Based Filtering of Financial Covariance Matrices for Portfolio Optimization"

[8] N. Basalto, R. Bellotti, F. De Carlo, P. Facchi, S. Pascazio (2004) "Clustering stock market companies via chaotic map synchronization"

[9] L. Kullmann, J. Kert´esz, R. N. Mantegna (2004) "Identification of clusters of companies in stock indices via Potts super-paramagnetic Transitions"

[10] Seung Ki Baek, Woo-Sung Jung, Okyu Kwon, Hie-Tae Moon (2005) "Transfer Entropy Analysis of the Stock Market"

[11] N. G. Pavlidis, D. K. Tasoulis, M. N. Vrahatis (2003) "Financial Forecasting Through Unsupervised Clustering and Evolutionary Trained Neural Networks"

[12] Hokky Situngkir, Yohanes Surya (2003) "Neural Network Revisited: Perception on Modified Poincare Map of Financial Time Series Data"

[13] Ramon Lawrence (1997) "Using Neural Networks to Forecast Stock Market Prices"

[14] M. B. Porecha,_P. K. Panigrahi, J. C. Parikh, C. M. Kishtawal, and Sujit Basu (2005) "Forecasting non-stationary financial time series through genetic algorithm"

[15] M.H.Jensen, A.Johansen, F.Petroni, I.Simonsen (2004) "Inverse Statistics in the Foreign Exchange Market"

[16] Mikosch T., Starica C (2004) "Stock Market Risk-Return Inference. An Unconditional Non-parametric Approach."

[17] Christopher J. Neely, Paul A.Weller (2002) "Predicting Exchange Rate Volatility: Genetic Programming Versus GARCH and RiskMetricsTM"