Forecasting High-Frequency Financial Time Series with Evolutionary Neural Trees: The Case of Hang-Seng Stock Index*

Shu-Heng Chen and Hung-Shuo Wang
AI-ECON Reserach Group
Department of Economics
Natioanl Chengchi University
Taipei, Taiwan 11623

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea

Abstract *In this paper, the evolutionary neural trees (***ENT***) are applied to forecasing the high-frequency stock returns of Heng-Sheng stock index on December, 1998. To understand what may consistute an effective implementation, six experiments are conducted. These experiments are different in data-preprocessing procedures, sample sizes, search intensity and complexity regularization. Our results shows that ENT can perform more efficiently if we can associate ENT with a linear filter so that it can concentrate on searching in the space of non-linear signals. Also, as well demonstarted in this study, the infrequent bursts (outliers) appearing in the high-frequency data can be very disturbing for the normal operation of ENT.*

Keywords: Evolutionary Artificial Neural Networks, Neural Trees, Sigma-Pi Neural Trees, Breeder Genetic Algorithm, Genetic Programming,

1 Motivation and IntroudcBtion

There has been such a rapid increase in the application of artificial neural networks to finance that ANNs have now become a standard tool for finance practitioners. However, the credibility of this application crucially depends on the style in which the network is designed. When what concerns us is not just one but many factors which may determine the performance of neural nets, it is very difficult, though not intractable, to design them manually becuase of combinatoric complexity. Therefore, it is desirable to know whether an automatic procedure, such as evolutionary computation, can help, and this motivated the many applications of *evolutionary artificial nural nets* (**EANNs**) to finance.

However, based on a survey article by [3], who distinguished three kinds of evolution in EANNs, most financial applications are only concerned with lowest level of evolution, namely, *connecting weights*. [1], to our best knowledge, is the first one to explore the potentials of EANNs at a high level of evolution, namely, evolution of *archiectures* and *learning rules*. By setting a forecasting competition on *forgein exchange rates*, they compared the performance of 8 backpropagation feedforward neural nets (**BPNNs**), 8 EANNs and the random walk model based on five chosen criteria. It is found that all nerual network models can statistically beat the RW in all criteria at the 1% significance level. In addition, among the 16 nerual netowrk models generated in different designs, the best model is the EANN with the largest search space.

In light of the earlier evidence of [1], this paper would like to pursue this line of research, while with a diffent focus. Instead of compar-

*The first author is grateful for the research support NSF No.87-2415-H-004-001.

ing the performance of EANNs with BPNNs, this paper ask a more fundamental issue: *given the possibility that evolutionary algorithms can be helpful for the design of ANNs, what may consistute an effective design of EANN?* To address this question, two setups are taken, which also distinguish this study from [1]. Firstly, the evolutionary algorithm employed in this paper, namely, *evolutionary nerual trees* ([4], [5]), is more rich in its content than the one used by [1]. This richness will enable us to ask address several important aspects of EANNs which may be difficult to be done in the framework of [1], such as complexity regulation, more flexible architectures, and hybridizing global search with local search. Secondly, the testbed chosen in this paper is a series of *stock returns* rather than foreign exchange rates.

The rest of the paper is organized as follows. Section 2 gives a brief introduction on ENT. Section 3 describes the financial time series used in this paper and, following the procedure given in [1], we justify this data as a testbed for EANNs. Section 4 gives the experiment designs followed by the analysis of the results in Section 5. We conclude this paper with a few remarks in Section 6.

2 Evolutionary Neural Trees

2.1 Neural Trees

Let $\mathcal{NT}(d,b)$ denote the set of all possible trees of maximum depth d and maixmum b branches for each node. The nonterminal nodes represent neural units and the neuron type is an element of the basis function set $\mathcal{F} = \{\text{neuron types}\}$. Each terminal node is labelled with an element from the terminal set $\mathcal{T} = \{x_1, x_2, ..., x_n\}$, where x_i is the ith component of the external input $\mathbf{x}$. Each link (j,i) represents a directed connection from node j to node i and is associated with a value w_{ij}, called the synaptic weight. The members of $\mathcal{NT}(d,b)$ are referred to as neural trees. In case of $\mathcal{F} = \{\Sigma, \Pi\}$, the trees are specifically called sigma-pi neural trees. The root node is also called the output unit and the terminal nodes are called input units. Nodes that are not input or output units are hidden units. The layer of a node is defined as the longest path length to any terminal node of its subtree. Different neuron types are distinguished in the way of computing net inputs. For the evolution of higher-order networks, we consider two types of units. Sigma units compute the sum of weighted inputs from the lower layer:

$$net_i = \sum_j w_{ij} y_j \tag{1}$$

where y_j are the inputs to the ith neuron. Pi units compute the product of weighted inputs from the lower layer:

$$net_i = \prod_j w_{ij} y_j \tag{2}$$

where y_j are the inputs to i. The output of a neuron is computed either by the threshold response function

$$y_i = \sigma(net_i) = \begin{cases} 1 & : \quad net_i \geq 0 \\ -1 & : \quad net_i < 0 \end{cases} \tag{3}$$

or the sigmoid transfer function

$$y_i = f(net_i) = \frac{1}{1 + e^{-net_i}} \tag{4}$$

where net_i is the net input to the unit computed by Eqn. 1 or 2.

A higher-order network with m output units can be represented by m sigma-pi neural trees. That is, the genotype A_i of ith individual in our evolutionary framework consists of m neural trees.

The neural tree representation does not restrict the functionality since any feedforward network can be represented with a forest of neural trees:

$$A_i = (A_{i,1}, A_{i,2}, ..., A_{i,m})$$

where $A_{i,k} \in \mathcal{NT}(d,b)\ \forall k \in \{1, ..., m\}$.

The connections between input units to arbitrary units in the network is also possible since input units can appear more than once in the neural tree representation. The output of one unit can be used as input to more than one unit. The duplication does not necessarily mean more space requirements in trees than network representations since frequently-used fit submodules can be stored and multiply reused. This leads to the construction of modular structures and reduces memory requirements for representing the population [5].

Neural trees do not require decoding for their fitness evaluation. Training and evaluation of fitness can be performed directly on the genotype since both the genotype and phenotype are equivalent. Since subtree crossover used in genetic programming may be applied without modification to this representation, we can use genetic programming as the main evolutionary engine.

2.2 Evolving Neural Trees

For the construction of neural models, we maintain a population $\mathcal{A}$ consisting of M individuals of variable size:

$$\mathcal{A}(g) = \{A_1, A_2, ..., A_M\}. \tag{5}$$

Each individual A_i is a neural network represented as neural trees. The initial population $\mathcal{A}(0)$ is created at random. In each generation g, the fitness values $F_i(g)$ of networks are evaluated and the upper $\tau\%$ are selected to be in the mating pool $\mathcal{B}(g)$. The next generation $\mathcal{A}(g+1)$ of M individuals are then created by exchanging subtrees and thereby adapting the size and shape of the network. Mutation changes the node type and the index of incoming units. The best individual is always retained in the next generation so that the population performance does not decrease as generation goes on (elitist strategy).

Between generations the network weights are adapted by a stochastic hill-climbing search. This search method is based on the breeder genetic algorithm [4], in which the step size Δw is determined with a random value $\epsilon \in [0, 1]$:

$$\Delta w = R \cdot 2^{-\epsilon \cdot K}, \tag{6}$$

where R and K are constants specifying the range and slope of the exponential curve. In the experiments, the values were $R = 2$ and $K = 4$. This method proved very robust for a wide range of parameter-optimization problems. The fitness F_i of the individuals A_i is defined as

$$F_i(g) = F(D|A_i^g) = E(D|A_i^g) + \alpha(g)C(A_i^g), \tag{7}$$

where $\alpha(g)$ is the Occam factor [5] that adaptively balances the error $E(D|A_i^g)$ and complexity $C(A_i^g)$ of the neural trees. This evaluation measure prefers simple networks to complex ones and turned out to be important for achieving good generalization.

3 Data Description and Preprocessing

Our data consists of the real-time observations of the Heng Sheng Stock Index for the period December 1 to December 31, 1998, at 1-minute frequency. This dataset (HSIX.HF) was downloaded from the Bridge company. The Hong Kong stock market opens 4 hours a day and five days a week. The specific period considered by us has 22 working days and 4586 observations. Denote the price series by $\{P_t\}_{t=1}^{4586}$. Figure 1 is the time series plot of $\{P_t\}$, and Figure 2 is its histogram. The basic statistics of this series are given at the right side of Figure 2.

Before proceeding to forecast this time series, we would like to know some basic econometric proporties of this series. The first property which we would like to study is: *is the price series stationary?* More specifically, does the price series have a unit root? The standard tool to test for the presence of a unit root is the celebrated Augmented Dickey-Fuller (**ADF**) test ([2]). The ADF test consist of running a regression of the first difference of the log price against the log price lagged once, lagged difference terms, and optionally, a constant and time trend. With four lagged difference terms, the regression is

$$\Delta p_t = \beta_1 p_{t-1} + \sum_{i=1}^{4} \beta_{i+1} \Delta p_{t-1} + \beta_6 + \beta_7 t, \tag{8}$$

Table 1: Stationarity and Linearity of the Price and the Return Series

Series	Price	Return
ADF	-2.276151	-28.24411
PSC	N.A.	(1,0)

where $p_t = \ln^{P_t}$ and $\Delta p_t = p_t - p_{t-1}$.

The null hypothesis is that β_1 is zero, i.e., $\{P_t\}$ contains a unit root. Based on the ADF test given in Table 2, the price series does not reject the null hypothesis of a unit root at both the 1% and the 5% significance level. What this suggests is that the conditional expectations (prediction) of p_t can be decomposed into two parts,

$$E(p_t \mid \Omega_{t-1}) = \hat{p}_t = p_{t-1} + E(\Delta p_t \mid \Omega_{t-1}) \quad (9)$$

Ω_{t-1} is the $\sigma-$algebra generated by the history of the log price, $\{p_{t-1}, p_{t-2}, ...\}$. However, from the machine learning perspective, the first part is not a big deal. The really interesting one is the second part $E(\Delta p_t \mid \Omega_{t-1})$.

Notice that the Δp_t is the continuous-time compound rate of return, r_t. The time series plot of $\{r_t\}$ is depicted in Figure 3. The histogram of the series is displayed in Figure 4, and the basic statistics of this return series are given at the right side of Figure 4. From these basic statistics, we can see that the value of skewness and kurtosis of the return series are much higher than what normal distribution predicts. In addition to skewing to the right, it also has an excess kutosis, normally known as *fat tails* in financial econometrics. The Jarque-Bera test further confirms that it is not normally distributed. The implication of this result is that the conditional expectations (prediction) of r_t ($\hat{r}_t$) in general may not be able to reduced to a linear predictor, and this calls for the need of nonlinear modeling technique such as *neural networks*.

However, before we come to any advanced nonlinear modeling teshnique, we have to check whether the return series is actually *nonlinearly dependent*. From the ADF test given in Table 1, we know that the return series is a stationary series. To check whether a stationary series is independent, we followed the standard econometric procedure. This procedure is composed of two steps, namely, PSC filtering and BDS testing. We first applied Rissanen′s predictive stochastic complexity (**PSC**) to filter the linear process. The third row of Table 1 gives us the ARMA(p,q) process extracted from the return series. The linear signals contained in the series follow an AR(1) process.

Table 2: The BDS Test of the PSC-filtered Return Series

ϵ / DIM	2	3	4	5
0.5	11.111	13.514	15.245	17.413
1	11.675	13.587	14.632	15.682
1.5	12.464	14.297	14.903	15.244
2	13.177	14.668	15.104	15.099

Once the linear signals are filtered out, any signals left in the residual series must be nonlinear. Therefore, one of the most frequently used statistics, the **BDS** test, is applied to the residuals filtered through the PSC filter. There are two parameters required to conduct the BDS test. One is the distance measure (ε standard deviations), and the other is the embedding dimension. The results with reference to different sets of parameters are given in Table 2. From these results, the hypothesis of the independence is rejected, i.e., the return series is nonlinearly dependent.

So far, we have evidenced that the return series is nonlinearly dependent. However, it is well known that most of the non-linearity in financial data seems to come from their second moments (σ^2). The voluminous (G)ARCH (Generalized AutoRegressive Conditional Heteroskedasticity) literature is the outcome of effort to capture by appropriate time series models the regularities in the behaviour of the second moment. In order to see whether GARCH can successfully capture all nonlinear signals,

Table 3: The LM test of the ARCH Effect in the Return Series

p	LM test	Prob.	p	LM test	Prob.
1	2.7862	0.0951	7	2.8104	0.9020
2	2.7864	0.2483	8	2.8110	0.9457
3	2.7984	0.4238	9	2.8112	0.9713
4	2.8024	0.5914	10	2.8149	0.9855
5	2.8025	0.7304	11	2.8257	0.9928
6	2.8086	0.8325	12	2.8293	0.9966

we first carried out the Lagrange Multiplier (**LM**) test for the presence of ARCH effects. The LM test for ARCH effects is a test based on the following model:

$$\sigma^2 = h(\alpha_0 + \alpha_1 \epsilon_{t-1}^2 + ... + \alpha_p \epsilon_{t-p}^2) \qquad (10)$$

where h is a differential function. The null hypothesis that the ARCH effect does not exist is:

$$\alpha_1 = ... = \alpha_p = 0 \qquad (11)$$

By taking p =1,2,...12, the LM test result is given in Table 3. ¿From Table 3, the null hypothesis was failed to reject. The sequence of procedures illustrated above is a way to construct a predictor of the stock return, which also highlights where *neural networks* can play a role. Let us briefly summarize as follows.

- **ADF procedure:**

$$p_t = p_{t-1} + r_t \qquad (12)$$

- **PSC procedure:**

$$r_t = \phi_0 + \phi_1 r_{t-1} + \epsilon_t \qquad (13)$$

- **BDS-GARCH procedure:**

$$r_t = \phi_1 r_{t-1} + g(r_{t-1}, r_{t-2}...) + e_t \qquad (14)$$

where e_t is an independent series, and $g(.)$ is a nonlinear function.

Table 4: Table of Control Parameters

Run	1	2	3	4	5	6
Series	r_t	g_t	r_t^s		r_t	r_t^l
n_1	3000				100	2000
n_2	1585				4485	2585
d,b	10, 5					
GEN	50			100	200	
POP	100				400	
O	10^{12}					10^8
τ	50				$50 - 100 \times \frac{gen-1}{Population}$	
K	4					
R	2				$2 - \frac{gen}{GEN-1}$	
LX	10				$5 \cdot gen + 5$	

Table of control parameters includes parameters of four different aspects, namely, inputs, sample size, global serach, and local search. All inputs are the observations on the last five miniutes in the series as indicated. For example, if the series is r_t, then the inputs are $\{r_{t-i}\}_{i=1}^5$.

Based on Equations (12) to (14), we can choose r_t as the input or the PSC-filtered residuals $g_t \equiv r_t - \phi_0 - \phi_1 r_{t-1}$ as the input. For the latter, ENTs is used mainly to extract nonlinear signals only, while for the former, ENTs is supposed to capture both linear and nonlinear signals. In addition, we also consider two transformations of r_t as an input. One is linear,

$$r_t^l = \frac{0.9}{0.033758} r_t, \qquad (15)$$

and the other is nonliner (sigmoid),

$$r_t^s = \frac{2}{1 + e^{-256} r_t} - 1. \qquad (16)$$

4 Experimental Designs

A sequence of 6 experiments (runs) were conducted. These 6 six experiments were designed to take into account different factors which may have influences on learning efficiency. *First, it is the choice of input series.* In addition to the raw data $\{r_t\}$, three transformed series as defined in Equations (14) to (16) were used. In this regards, what concerns us is whether the data is needed to be perprocessed before it is feed to ENTs. In particular, would it be more efficient if we can associate ENT with a linear filter so that it can concentrate on searching in the space of nonlinear signals (Experiment 2).

Second, the determination of in-sample size (n_1). Here, the main problem is a trade-off between computation intensity (search intensity) and asympototic efficiency. If the sample size is too big, it would be very time-consuming for intensice search (Experiments 1-4). Nonetheless, a sample size may be too small to achieve asympototic efficiency if the serie is stochastically nonlinear rather than chaotically (deterministically) nonlinear (Experiment 5). Once the training sample size was fixed, the post-sample sample size (n_2) was then determined by what was left.

Third, search intensity. Search intensity includes exploration (global search) intensity and exploitation (local search) intensity. The parameters driving the global search are mainly the search space of trees (d, b), the number of generations (GEN) and the population size (POP). In term of the parameters GEN and POP, Experiment 5 and 6 involved the most intensive search. In addition, there is a penalty function imposed to regularize the complexity of neural tress. The complexity function $C(A_i^g)$ used throughout all simulations was:

$$C(A_i^g) = \frac{\#\ \ of\ \ nodes + 3 \times depth}{order} \qquad (17)$$

The parameter *order* (O) was exogenously given and was adjusted to control the size of model inferred. For the first five experiments, this parameter was set to be 10^{12}. It was then decreased up to 10^8 to increase the penalty in the last experiment. Furthermore, two different τs were employed. For the first four experiment, τ was kept as a constant, while for the last two experiments, τ was decreasing in number of generation. In the latter case, eventually only the few most outstanding models would be selected to constitute the mating pool.

The parametrs controlling the local search are K, R and LX (the numbmer of iterations used in the hill-climbing search procedure). While R and LX were set as a constant in the first four experiments, they were adaptive in the last two experiments. For R, it was set to be $2 - \frac{gen}{GEN-1}$, where $gen = 1, 2, ..., 50$. By this setting, the range of parameter puturbation will shrink as evolution gets longer and longer. For LX, it was set as an increasing function of "gen" (generations). As a result, exploitation intensity would be enhanced as number of genertaions increasesd.

Table 5: Experimental Results

	MSE	MAPE	HIT
Experiments 1-4			
RW	8.95	100	0.47
AR1	9.46	196	0.51
ENT(1)	9.49	199	0.507
ENT(2)	9.17	142	0.512
ENT(3)	9.21	221	0.50
ENT(4)	9.24	230	0.50
Experiment 5			
RW	9.8	100	0.496
AR1	10.13	240	0.55
ENT(5)	59.83	272	0.56
Experiment 6			
RW	7.56	100	0.488
AR1	7.82	193	0.536
ENT(6)	7.87	231	0.537

5 Experimental Results

The evaluation is made by using the randomw walk (**RW**) and AR(1) as the benckmarks and by choosing the *mean sequared error* (**MSE**), *mean absoulte percentage error* (**MAPE**) and hit ratio (**HIT**) as the performance criteria. AR(1) was used as a benchmark because the time series on question was identified as an AR(1) series by PSC. The results are exhibited in Table 5.

From Table 5, we can see that RW beat both AR(1) and ENT in all experiments in the criteria MSE and MAPE, while, in the criterion HIT, it was defeated by AR(1) and ENT. We feel a little puzzle about this behaviour RW. Theoretically speaking, if a series has a AR(1) as a part, it is unlikely that RW cannot possibly beat AR(1) in the large sample (n_2 =

1585). Nevertheless, in Figure 3, there are many *bursts* irregularly observed in the return series. We are not sure whether these burst may be in favour of RW relative to AR(1). A careful analysis, such as an analysis based on the Monte Carlo simulation, is needed in the future analysis.

Leaving this puzzle aside, let us examine how the peformance of ENT was affected by our different designs. *First, input series.* Six experiments covered four different input series, namely, r_t, g_t, r_t^l and r_t^s. Only in the case when the input series was taken as the filtered residual g_t, ENT outperofrmed AR(1) in all three criteria. This result is somewhat interesting because, first, it confirms the significance of the seperation of nonlinear signals from linear signals, and second, it says that ENT can be more effective in extracting nonlinear signals than linear ones. Since the return series had linear signals as a part, ENT had difficulties to beat AR(1) directly. However, once the linear signals were filtered out, it is not surprising that ENT, as a nonlinear signal processing, was the winner. Other data preprocessing may also work. For example, in the case when the input series was taken as the nonlinear (sigmoid) transformation of returns, ENT also beat AR(1) in the criterion MSE, while not others.

Second, the sample size. When the data is stochastic rather than chaotic, reducing training sample size may hurt the post-sample perfromance. It is quite clear from Experiment 5, where only 100 observations were used as the trainging samples. Third, the sample size and the serarch intensity. Enhancing search intensity may not help very much when the data is too nosiy in the sense that nonlinear signals are weak. It would be awefully bad if the sample size is not large enough. In other words, intensive learning on a small noisy sample can easily lead to a overfiting result. This is probably what we learned from experiments 5 and 6.

A related issus is complexity regularization. The paramter O had a very signiifanct impact on the size of the best tree. To see this, Box 1 presents the best tree discovered in Experiment 4 and 6. The difference in their size is noticeable.

6 Conclusions

Since highly non-linear signals has been shown to exist in many financial time series, there is always a potential reward for search, in particular, smart search. The research area EANNs is about the smart search over the space of ANNs. This paper provides an initial attemp to examine what factors may define a smart search. Clearly, lots of efforts needed to be done in the future to see how this way to search may be relevant to financial time series.

References

[1] Chen, S.-H. and C.-F. Lu. Would Evolutionary Computation Help in Designs of ANNs in Forecasting Foreign Exchange Rates? forthcoming in *Proceeding of 1999 Congress on Evolutionary Computation*, IEEE Press.

[2] Dickey, D. A. and W. A. Fuller. Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root *Econometrica* 49, pp. 1057-1072, 1981.

[3] Yao, X. A Review of Evolutionary Artificial Neural Networks. International Journal of Intelligent Systems, Vol. 8, No. 4, pp. 539-567. 1999.

[4] Zhang B.-T. and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, Vol. 3, pp. 17-38, 1995.

[5] Zhang, B.-T., Ohm, P. and Müehlenbein, H. Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5(2): 213-236. 1997.