Evolutionary Artificial Neural Networks and Genetic Programming: A Comparative Study Based on Financial Data

S.-H. Chen and C.-C. Ni
Department of Economics, National Chengchi University,
Taipei, Taiwan 11623
E-mail: chchen@cc.nccu.edu.tw, g2258503@grad.cc.nccu.edu.tw

Abstract

In this paper, the stock index $S\&P\,500$ is used to test the predicting performance of genetic programming (GP) and genetic programming neural networks (GPNN). While both GP and GPNN are considered *universal approximators*, in this practical financial application, they perform differently. GPNN seemed to suffer the *overlearning* problem more seriously than GP; the latter outdid the former in all the simulations.

1 Introduction and Motivation

In this paper, we compare the prediction performance between *evolutionary artificial neural networks* (EANNs) and *genetic programming* (GP). EANNs can be regarded as a subset of the function space defined by GP, i.e., $Space_{EANN} \subseteq Space_{GP}$. To exemplify this *set relation*, an artificial neural network (ANN) and its corresponding LISP tree representation are depicted in Figures 1 and 2.

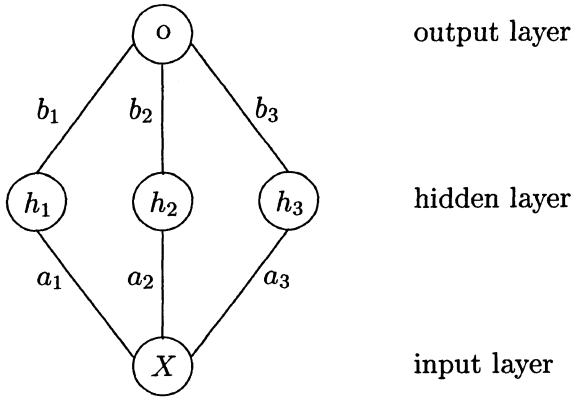

Figure 1: A 1-3-1 ANN Architecture.

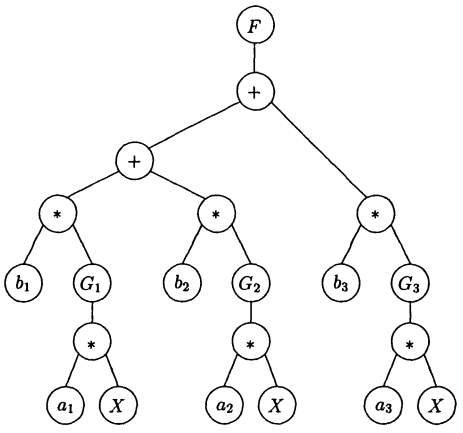

Figure 2: The LISP Tree of ANN(1-3-1).

While this paper only considers *feedforward* ANNs, this illustration can be easily extended to the case of *recurrent* ANNs. The interested reader is referred to [3].

The ANN in Figure 1 is a 1-3-1 architecture, i.e., a single input-output node and one hidden layer with three hidden nodes. The input for the ANN is denoted by X. The transfer function of the hidden nodes and output node are given in Equations (1) and (2),

$$h_i = G_i(a_i \times X), \quad i = 1, 2, 3 \tag{1}$$

$$o = F(\sum_{j=1}^{3}(b_j \times h_j)), \tag{2}$$

where a_i and b_j are weights. In a typical application, G_i can be the identity function and F the

G. D. Smith et al., *Artificial Neural Nets and Genetic Algorithms*

sigmoid function, the *Gaussian basis function* or the *Gaussian kernel function*. Generally speaking, in the EANN literature, choice of these functions should be *automatically* determined [3].

Since each ANN can be coded as a LISP tree, EANNs can also be implemented with GP. In fact, there has been a growing interest in using GP to evolve the ANN architecture over the last two years [3]. The difference between GPNN and GP lies in the *search space*. The search space defined by GPNN is a network architecture space with additional syntactic restrictions imposed upon the original GP search space. From the perspective of *approximation theory*, these two search spaces are *asymptotically equivalent*, i.e.,

$$\lim_{d\to\infty} Space_{GP}(d) \approx \lim_{d\to\infty} Space_{GPNN}(d), \qquad (3)$$

where d denotes the maximum depth of the LISP tree. In other words, if there is no limit set upon *the depth of the tree*, then these two search spaces are approximately equivalent in the sense that both GP and GPNN are *universal approximators*. However, in practice, the upper limit of d always exists. Take Koza [4] as an example; d was set to be 17 in most of the applications. Therefore, in application, it is very likely that $Space_{GP}(d)$ is not the same as $Space_{GPNN}(d)$, and neither is a subset of the other. Even if they are the same, the difference in representation (syntax) may cause these two learning schemes to have such different dynamics that their performance will differ. Therefore, while GPNN provides EANN research with a promising technique, many genetic programmers are still quite reserved about GPNN.

Since the GP-GPNN equivalence issue is difficult to solve analytically, and since ANNs have been extensively used in *financial engineering*, it is desirable to have an empirical exploration of this issue based on financial applications. Motivated by this equivalence question, we conducted two series of experiments of predicting stock returns. Each series is composed of ten simulations. The first series of experiments were carried out by using GP, and the second by GPNNs.

2 Simulation Design

The data used in this study is *inherently difficult to predict*. Such 'inherent difficulty' is made precise via the *minimum description length (MDL) principle*. The MDL principle is applied to daily returns of the *S&P* 500 index to identify highly unpredictable subsets of samples with size 200. Details of this procedure are well documented in Chen and Tan [1]. Daily observations of the S&P 500 index from 1/2/1953 to 9/9/1994 are used to create percentage returns. The MDL methodology is applied to this whole dataset of percentage returns and the subset period 1/3/92 to 10/16/92 is chosen. This subset is further decomposed into the in-sample set and post-sample set in the ratio of 10 to 1.

To implement genetic programming, the program GP-Pascal is written in Pascal 4.0 by following the instructions given in [3]. The chosen parameters to run GP-Pascal are given in Table 1. % and *RLOG* appearing in the function set are the protected division function and the protected natural logarithm function respectively [3]. In this paper, all simulations conducted are based on the terminal set, which includes the ephemeral random floating-point constant R ranging over the interval [-9.99, 9.99] and the rate of return lagging up 10 periods, i.e., R_{t-1},...R_{t-10}. To escape local optima, the mutation rate is set to be 0.2. In addition, *elite operator* is "on" and is to keep the best-so-far program to the next generation.

The fitness criterion *Mean Absolute Percentage*

Table 1: Tableau for GP Parameters.

Population size	500
Number of trees created by complete growth	50
Number of trees created by partial growth	50
Function set	$\{+, -, \times, \%, EXP, RLOG, Sin, Cos\}$
Terminal set	$\{R_{t-1}, R_{t-2}, ..., R_{t-10}, R\}$
Number of trees generated by reproduction	50
Number of immigrants	50
Number of trees created by crossover	300
Number of trees created by mutation	100
Elite Operator	on
Probability of mutation	0.2
Maximum depth of the tree	17
Probability of leaf selection under crossover	0.5
Number of generations	200
Maximum number in the domain of Exp	1700
Criterion of fitness	MAPE

	GP	GPNN
Gen	MAPE	MAPE
50	1.008918	1.292107
	1	2.558425
	0.99877	1.984527
	1.002416	1.424301
	1.007977	1.23407
	1.019065	1.774092
	1.202154	1.945665
	1.001401	1.980485
	0.999696	1.201444
	1.005213	1.241057
100	1.008026	1.221301
	1	2.291675
	0.99877	1.571851
	1.002416	1.424301
	1.00608	1.184122
	1.019395	1.894853
	1.039181	1.225833
	1.001633	1.812707
	1.000016	1.356968
	1.000816	1.815196
150	1.008748	1.369779
	1	3.6597
	1.00191	4.02578
	1.01299	2.644374
	1.001158	2.645544
	1.020601	2.495246
	1.038312	1.488896
	1.000334	1.429968
	1.000016	2.828882
	1.001625	2.755031
200	1.008878	3.005329
	1	3.695376
	0.99197	4.018836
	1.017207	0.966569
	0.998223	2.181985
	1.02005	2.147918
	1.038935	3.499633
	1.000271	4.837395
	1.000018	2.31551
	1.022788	3.575092

Figure 3. The Predicting Performance Comparison between GP and GPNN : Gen 50

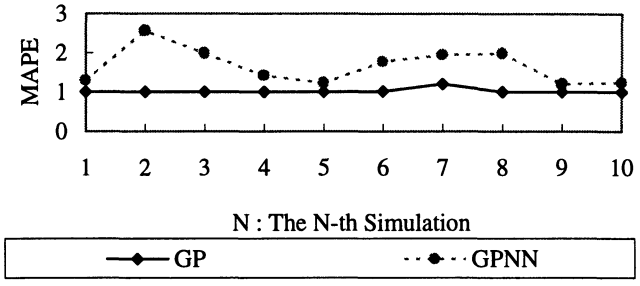

Figure 4. The Predicting Performance Comparison between GP and GPNN : Gen 100

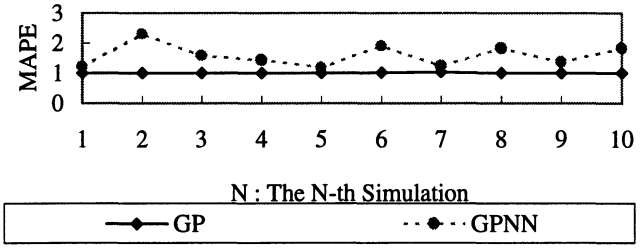

Figure 5. The Predicting Performance Comparison between GP and GPNN : Gen 150

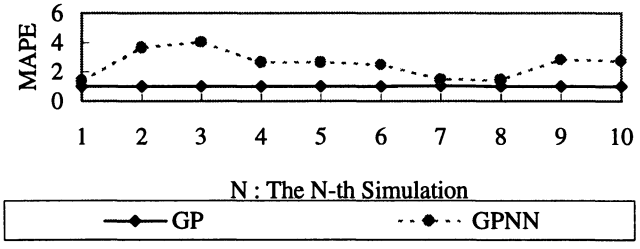

Figure 6. The Predicting Performance Comparison between GP and GPNN : Gen 200

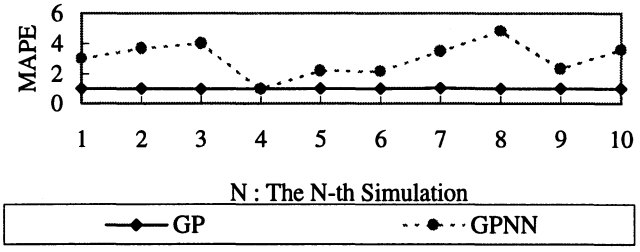

Table 2: Tableau for EANN Parameters.

Population Size	196
Number of input nodes	11
Window size	3
Learning rate	self adapting
Error tolerance	self adapting
Momentum rate	self adapting
Output processing	non-linear but fixed
Network model	genetica

Error (MAPE) is defined as follows:

$$\text{MAPE} = \sum_{i=1}^{m} \frac{\mid \hat{R}_i - R_i \mid}{m \mid R_i \mid}, \tag{4}$$

where m is the sample size and $\hat{R}_i$ is the prediction value of R_i. The choice of the *mean absolute percentage error* as the fitness function is attributed to [5], who suggested a modified form of MAPEs as the most appropriate measure satisfying both theoretical and practical concerns while allowing meaningful relative comparisons.

Based on those control parameters, multiple runs of simulations were executed. For each of the simulations, the MAPE is calculated for the in-sample period and the post-sample period. The results of the post-sample MAPEs of generations 50, 100, 150 and 200 under each simulation are exhibited in Figures 3-6.

As to the implementation of EANNs, there are lots of different encoding strategies, including genetic algorithms, evolutionary programming, and genetic programming. In this paper, we follow Wong [6] to encode ANNs and use *NeuroForecaster 4.2* to conduct the experiments. The controlled parameters to run EANNs are given in Table 2. Notice that the node transfer function is exogenously given and is fixed. As a comparison, the results of the post-sample MAPEs of generations 50, 100, 150 and 200 under each simulation are also depicted in Figures 3-6.

3 Simulation Results and Conclusions

From Figures 3-6, we can see that GP's performance is *uniformly superior* to GPNN's for all generations. Moreover, by comparing Figures 3-4 with Figures 5-6, it is interesting to note that GPNN suffers the *overfitting* problem more seriously than GP. These results indicate that presence or absence of the ANN architecture can make a difference in implementing GP driven search. In this typical financial engineering application, imposing the ANN architecture did not bring anything good. It remains to be investigated whether it is helpful to make automatic the determination of *node transfer functions.*

4 Acknowledgements

Research support from NSC grant No.85-2415-H-004-001 is gratefully acknowledged. The authors are grateful to two anonymous referees for helpful comments. This is a short version of the full paper with the same title.

References

[1] S.-H. Chen and C.-W. Tan. *Measuring Randomness by Rissanen's Stochastic Complexity: Applications to the Financial Data*, pages 200–211. World Scientific, 1996.

[2] S.-H. Chen and C.-H. Yeh. Bridging the gap between nonlinearity tests and the efficient market hypothesis by genetic programming. In *Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering*, pages 34–39. IEEE Press, 1996.

[3] A. I. Esparcia-Alcazar and K. C. Sharman. Evolving recurrent neural network architectures by genetic programming. In *Proc. Genetic Programming 1996 Conference.* Stanford, CA, U.S.A., July 28-31 1996.

[4] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, 1992.

[5] S. Makridakis. Accuracy measure: Theoretical and practical concerns. *International Journal of Forecasting*, 9:527–529, 1993.

[6] F. Wong. Neurogenetic computing technology. *NeuroVe$t Journal*, 2(4):12–15, 1996.