ELSEVIER

Physica A 289 (2001) 574–594

PHYSICA A

www.elsevier.com/locate/physa

Using genetic algorithms to select architecture of a feedforward artificial neural network

Jasmina Arifovic[a], Ramazan Gençay[b,c,*]

[a] *Department of Economics, Simon Fraser University, Burnaby, BC, Canada V5A 1N6*

[b] *Department of Economics, University of Windsor, 401 Sunset Avenue, Windsor, Ont. Canada N9B 3P4*

[c] *Department of Economics, Bilkent University, Bilkent, Ankara 06533, Turkey*

Received 12 June 2000; received in revised form 14 August 2000

Abstract

This paper proposes a model selection methodology for feedforward network models based on the genetic algorithms and makes a number of distinct but inter-related contributions to the model selection literature for the feedforward networks. First, we construct a genetic algorithm which can search for the global optimum of an arbitrary function as the output of a feedforward network model. Second, we allow the genetic algorithm to evolve the type of inputs, the number of hidden units and the connection structure between the inputs and the output layers. Third, we study how introduction of a local elitist procedure which we call the *election* operator affects the algorithm's performance. We conduct a Monte Carlo simulation to study the sensitiveness of the global approximation properties of the studied genetic algorithm. Finally, we apply the proposed methodology to the daily foreign exchange returns. © 2001 Published by Elsevier Science B.V. All rights reserved.

PACS: 84.35; 02.60

Keywords: Genetic algorithms; Neural networks; Model selection

1. Introduction

The design of an artificial network architecture capable of learning from a set of examples with the property that the knowledge will generalize successfully to other patterns from the same domain has been widely recognized as an important issue in the literature. This paper proposes a model selection methodology for feedforward network models based on the genetic algorithms. At the outset, we would like to point

* Corresponding author. Fax: +1-5199737096.

E-mail address: gencay@uwindsor.ca (R. Gençay).

PII: S0378-4371(00)00479-9

out that our framework is entirely unrelated to biological networks and our attempt is not to emulate an actual neural network.

Artificial neural networks provide a rich, powerful and robust nonparametric modelling framework with proven and potential applications across sciences. Examples of such applications include Elman [1] for learning and representing temporal structure in linguistics; Jordan [2] for controlling and learning smooth robot movements; Gençay and Dechert [3], Gençay [4,5] and Dechert and Gençay [6,7] to decode noisy chaos and Lyapunov exponent estimations and Kuan and Liu [8] for exchange rate prediction. Kuan and Liu [8] use the feedforward and recurrent network models to investigate the out-of-sample predictability of foreign exchange rates. Their results indicate that neural network models provide significantly lower out-of-sample mean squared prediction errors relative to the random walk model. Swanson and White [9] study the term structure of the interest rates with feedforward neural networks together with the linear models. Their results indicate that the premium of the forward rate over the spot rate helps to predict the sign of the future changes in the interest rate when the conditional mean is modelled by the feedforward network estimator. Hutchinson et al. [10] employ feedforward networks along with other nonparametric networks for estimating the pricing formula of derivative assets. Their results indicate that although parametric derivative pricing formulas are preferred when they are available, nonparametric networks can be useful substitutes when parametric methods fail. Garcia and Gençay [11] utilize feedforward networks in modelling option prices by imposing hints originating from the economic theory. Their results indicate that feedforward networks provide more accurate pricing and hedging performances. They point out that network selection needs to be done in accordance with the objective function of the problem at hand.

The specification of a typical neural network model requires the choice of the type of inputs, the number of hidden units, the number of hidden layers and the connection structure between the inputs and the output layers. The common choice for this specification design is to adopt the model-selection approach. In the recent literature, information based criteria such as the Schwarz information criterion (SIC) and the Akaike information criterion (AIC) are used widely. Swanson and White [9] report that the SIC fails to select sufficiently parsimonious models in terms of being a reliable guide to the out-of-sample performance. Since the SIC imposes the most severe penalty among the AIC and the Hannan–Quinn, the results with the two other criteria would give even worse results for the out-of-sample prediction. Hutchinson et al. [10] indicate the need for proper statistical inference in the specification of nonparametric networks. This involves the choices for additional inputs and the number of hidden units in a given network.

The purpose of this paper is to introduce an alternative model selection methodology for feedforward network models based on the genetic algorithm [12] which can search for the global optimum of an arbitrary function as the output of a feedforward network model.[1] There have been a large number of applications of the genetic algorithm

[1] For a discussion of the advantages of the genetic algorithm over hill-climbing and simulated annealing in a simple optimization problem, see Ref. [13].

for the artificial neural networks. The purpose of using the genetic algorithm has been twofold. The first one is to use it as a means to learn artificial neural network connection weights that are coded, as binary or real numbers, in a genetic algorithm string (see, for example, Refs. [14–17]). The second one is to use the genetic algorithm to evolve and select the artifical neural network architecture, together or independently from the evolution of weights. Miller et al. [18] identified two approaches to code the artifical neural network architecture in a genetic algorithm string. One is the strong specification scheme (or direct encoding scheme) where a network's architecture is explicitly coded. The other is a weak specification scheme (or indirect encoding scheme) where the exact connectivity pattern is not explicitly represented. Instead it is computed on the basis of the information encoded in the string by a suitable developmental rule. The examples of the applications of the strong specification scheme include Miller et al. [18], Whitley et al. [19], Schaffer et al. [20], Menczer and Parisi [15]. The applications of the weak specification scheme include Harp et al. [21] and Kitano [22,23].[2]

Our approach to encoding the neural network architecture is similar to the approach taken by Schaffer et al. [20] They use the genetic algorithm to evolve the range of parameter values of the backpropagation algorithm used for neural network training (learning rate and momentum), the number of hidden units and the range of initial weights values. The neural network is trained on a standard XOR problem frequently used in the studies of neural networks' performance.

In our approach, we use the genetic algorithm to evolve the range of initial neural network weights, the number of hidden units and the number and the type of inputs. The neural networks constructed from the information encoded in the genetic algorithm strings are trained on simulated as well as actual financial time series data. The simulated series are generated from the Henon map as it is a well-known benchmark and used widely in many studies. The financial time series is the daily foreign exchange rate on French franc denominated in US dollars.

We employ a local elitist operator, the election operator [25]. The application of this operator results in the endogenous control of the realized rates of crossover and mutation. Over the course of a simulation, there is less and less improvement in the performance of new genetic algorithm strings generated through crossover and mutation. New strings that encode architectures with inferior performance are prevented from becoming the members of the actual genetic algorithm populations. Over time, the use of this operator results in the convergence of the genetic algorithm population to a single string (architecture).

We conduct a Monte Carlo simulation to study the sensitiveness of the global approximation properties of our genetic algorithm. The comparison of the effects of using the genetic algorithm (GA) as a model selection methodology to the other standardly used criteria, AIC and SIC, has not been done in the literature. We find that the genetic algorithm selects networks with the out-of-sample mean squared prediction error lower

[2] For a survey of the encoding methods in the use of genetic and evolutionary algorithms in neural network applications, see Ref. [24].

than the networks selected by SIC and AIC although the GA selected networks have larger number of hidden units relative to the SIC (AIC) ones.

We also find that allowing the initial weight range to evolve again substantially reduces the out-of-sample mean squared prediction error. The optimization problems where neural networks are used are frequently characterized by the ruggedness of the surface. In these cases, the choice of initial weights becomes extremely important. As our study shows, letting the genetic algorithm choose the initial weight range greatly improves the neural network performance.

Moreover, we investigate the impact of the evolvable number and type of inputs and compare the results of simulations in which the number of inputs was fixed and the one where it was allowed to vary. The results of our simulations show that in cases where the number and type of inputs was allowed to evolve, the neural networks had lower out-of-sample mean-squared prediction error (MSPE).

We also compare the performance of the neural network architectures that were evolved using the genetic algorithm with the election operator to those that were evolved using the genetic algorithm without the election operator. Simulations with the election operator result in much faster convergence and in the selection of networks with lower values of the out-of-sample mean squared prediction error.

The rest of the paper is organized as follows. Feedforward neural networks are described in Section 2. The hybrid genetic algorithm is described in Section 3. The results of simulations are presented in Section 4. The financial time series application is presented in Section 5. We conclude thereafter.

2. Feedforward neural network

A typical regression function is written as, $f(x, \theta)$, where x stands for the *explanatory variables*, θ is a vector of parameters and the function f determines how x and θ interact. This representation is identical to the output function of a feedforward network such that the network inputs are interpreted as the explanatory variables and the weights in the network are interpreted as the parameters, θ. In a typical feedforward network, the input units send signals x_j across weighted connections to intermediate or *hidden* units. Any given hidden unit j sees the sum of all the weighted inputs, $\gamma_{j0} + \sum_{i=1}^{p} \gamma_{ji} x_i = \gamma_{j0} + \gamma_{j1} x_1 + \cdots + \gamma_{j1} x_p$. The first term γ_{j0} is an intercept or a *bias* term. The weights γ_{ji} are the weights to the jth hidden unit from the ith input. The hidden unit j outputs a signal $h_j = G(\gamma_{j0} + \sum_{i=1}^{p} \gamma_{ji} x_i)$ where the *activation function* G is

$$G(x) = \frac{1}{1 + \mathrm{e}^{-\alpha x}} ,$$

a logistic function and it has the property of being a sigmoidal[3] function. The signals from the hidden units $j = 1, \ldots, d$ are sent to the output unit across weighted connections

[3] G is a sigmodial function if $G : R \to [0, 1]$, $G(a) \to 0$ as $a \to -\infty$, $G(a) \to 1$ as $a \to \infty$ and G is monotonic.

in a manner similar to what happens between the input and hidden layers. The output unit sees the sum of the weighted hidden units, $\beta_0 + \sum_{j=1}^{d} \beta_j h_j$; the hidden to output weights are $\beta_0, \ldots, \beta_d$. The output unit then produces a signal $\beta_0 + \sum_{j=1}^{d} \beta_j h_j$. If the expression for h_j is substituted into the expression $\beta_0 + \sum_{j=1}^{d} \beta_j h_j$, it yields the output of a single layer feedforward network

$$f(x, \theta) = \Phi\left(\beta_0 + \sum_{j=1}^{d} \beta_j G\left(\gamma_{j0} + \sum_{i=1}^{p} \gamma_{ji} x_i\right)\right)$$

as a function of inputs and weights. The expression $f(x, \theta)$ is convenient short-hand for network output since this depends only on inputs and weights. In general, Φ is an identity function for the regression function estimation. The symbol x represents a vector of all the input values, and the symbol θ represents a vector of all the weights (β's and γ's). We call f the *network output function*.

Many authors have investigated the universal approximation properties of neural networks [26–31]. Using a wide variety of proof strategies, all have demonstrated that under general regularity conditions, a sufficiently complex single hidden layer feedforward network can approximate any member of a class of functions to any desired degree of accuracy where the complexity of a single hidden layer feedforward network is measured by the number of hidden units in the hidden layer. One of the requirements for this universal approximation property is that the activation function has to be a sigmoidal, such as the logistic function presented above. Because of this universal approximation property, the feedforward networks are useful for applications in pattern recognition, classification, forecasting, process control, image compression and enhancement and many other related tasks. For an excellent survey of the feedforward and recurrent network models, the reader may refer to Refs. [32,33].

Given a network structure and the chosen functional forms for G and Φ, a major empirical issue in the neural networks is to estimate the unknown parameters θ with a sample of data values of targets and inputs. The following learning algorithm[4] is commonly used:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + \eta \nabla f(x_t, \hat{\theta}_t)[y_t - f(x_t, \hat{\theta}_t)]\,,$$

where $\nabla f(x_t, \theta)$ is the (column) gradient vector of f with respect to θ and η is a learning rate. Here, $\nabla f(x_t, \theta)[y_t - f(x_t, \theta)]$ is the vector of the first-order derivatives of the squared-error loss: $[y_t - f(x_t, \theta)]^2$. This estimation procedure is characterized by the recursive updating or the learning of estimated parameters. This algorithm is called the method of *backpropagation*. By imposing appropriate conditions on the learning rate and functional forms of G and Φ, White [36] derives the statistical properties for this estimator. He shows that the backpropagation estimator asymptotically converges to the estimator which locally minimizes the expected squared error loss.

[4] The learning rule that we study here is not in biological nature. Heerema and van Leeuwen [34] study biologically realizable learning rules which comply with Hebb's [35] neuro-physiological postulate and they show that these learning rules are not the types proposed in the literature.

A modified version of the backpropagation is the inclusion of the Newton direction in recursively updating $\hat{\theta}_t$ [32]. The form of this recursive Newton algorithm is

$$\hat{\theta}_{t+1} = \hat{\theta}_t + \eta_t \hat{G}_t^{-1} \nabla f(x_t, \hat{\theta}_t)[y_t - f(x_t, \hat{\theta}_t)] ,$$
$$\hat{G}_{t+1} = \hat{G}_t + \eta_t [\nabla f(x_t, \hat{\theta}_t) \nabla f(x_t, \hat{\theta}_t)' - \hat{G}_t] , \tag{1}$$

where $\hat{G}_t$ is an estimated, approximate Newton direction matrix and $\{\eta_t\}$ is a sequence of learning rates of order $1/t$. The inclusion of Newton direction induces the recursively updating of $\hat{G}_t$, which is obtained by considering the outer product of $\nabla f(x_t, \hat{\theta}_t)$. In practice, an algebraically equivalent form of this algorithm can be employed to avoid matrix inversion.

These recursive estimation (or on-line) techniques are important for large samples and real-time applications since they allow for adaptive learning or on-line signal processing. However, recursive estimation techniques do not fully utilize the information in the data sample. White [36] further shows that the recursive estimator is not as efficient as the nonlinear least-squares (NLS) estimator. We, therefore, use the NLS estimator by minimizing

$$L(\theta) = \sum_{t=1}^{n} (y_t - f(x_t, \theta_t))^2 . \tag{2}$$

In Gallant and White [27], it is shown that feedforward networks can be used to consistently estimate both a function and its derivatives. They show that the least-squares estimates are consistent in Sobolev norm, provided that the number of hidden units increases with the size of the data set. This would mean that a larger number of data points would require a larger number of hidden units to avoid overfitting in noisy environments.

3. Genetic algorithm

The genetic algorithm is a global search algorithm which operates on a population of rules. Based on the mechanics of selection and natural genetics, it promotes over time the rules that perform well in a given environment and introduces into the population new rules to be tried. Rules are coded as binary strings of finite length. The measure of the rules' performance is defined by their fitness function.

We use the genetic algorithm to develop an alternative model selection methodology for feedforward network models. A genetic algorithm population consists of N binary strings. Each binary string i, $i \in [1, N]$, encodes a neural network architecture i, $i \in [1, N]$. The binary string consists of *lchrom* bits. The *lchrom* bits are divided into three parts. The first part of length *lw* is used to encode the initial weight range. The second part of length *li* is used to encode what inputs will be used and the third part of length *lh* is used to encode the number of hidden units.

Given the number of bits *lw* in the first part of the string, the number of different intervals that can be represented is 2^{lw}. Each integer j, $j \in [0, 2^{lw}]$ is interpreted as the

jth interval. The real value range of each interval is exogenously given. Here is an example of $lw = 2$ and the interpretation of combinations of bit values. Since $lw = 2$, four different intervals for initial weights can be encoded:

Encoding of initial weights' range

bits	*weight range*
00	$[-0.125, 0.125]$
01	$[-0.25, 0.25]$
10	$[-0.5, 0.5]$
11	$[-1, 1]$

Given the number of bits li in the second part of the string, the number of inputs that can be encoded is li. If bit j, $j \in [1, li]$, is equal to 1 then jth input, $j \in [1, li]$, is used in training. If bit j is equal to 0, input j is not used in training.[5]

Given the number of bits, lh, in the third part of the string, the maximum number of hidden units, nh, that a network can have is given by 2^{lh}. Here is an example with $lh = 3$ with the maximum number of hidden units $nh = 8$.

Encoding of hidden units

bits	*# of hidden units*	*bits*	*# of hidden units*
000	1	100	5
001	2	101	6
010	3	110	7
011	4	111	8

The following is an example of a string with $lchrom = 7$, $lw = 2$, $li = 5$, and $lh = 3$ and how it is decoded:

10 10100 010 .

This string will decode into a neural network whose initial range of weights is between -0.25 and 0.25, that uses first and third input in its training pattern and has three hidden units.

Each data set consists of three parts, called the training, test, and prediction samples, respectively. The training sample is utilized during the local minimization stage, while the test sample is used to evaluate a fitness value of a given network. Finally, the prediction sample of a data set is used only for evaluating networks' predictive power and is not utilized at any stage of the estimation of a network.

Information decoded from a binary string i, $i \in [1, N]$, is used to construct a neural network architecture i. Then 500 different sets of initial weights are generated within the initial weight range given by the architecture. These 500 sets of weights are used to construct 500 neural networks with the architecture i. These networks are then

[5] If only the number of inputs need to be encoded, a binary number encoding scheme can be adopted.

trained using the conjugate gradient method on a set of given input/output patterns constructed using the training sample of a data set. The network that results in the lowest mean-squared error in the test sample is used as a starting point in computation of a fitness value architecture of i.

The fitness value of a binary string i is calculated using the mean squared error[6] for the test sample, MSE_i, of a feedforward network architecture i. A fitness value μ_i of the binary string i is then given by

$$\mu_i = \frac{1}{(MSE_i + 1)} ,$$

where MSE_i is the mean squared error of network i from the test sample. Thus, the smaller the network's MSE, the closer a fitness value to 1. Once fitness values of N strings are evaluated, a population of binary strings is updated using four genetic operators: reproduction, crossover, mutation and election.

Reproduction makes copies of individual strings. The criterion used for copying is the value of the fitness function. In this paper, the tournament selection method is used as a reproduction operator. Two binary strings are randomly selected and their fitnesses are compared. The binary string with a higher fitness is copied and placed into the mating pool. Again, tournament selection is repeated N times in order to obtain N copies of chromosomes.

Crossover exchanges parts of randomly selected binary strings. First, two binary strings are selected from the mating pool at random, without replacement. Secondly, a number k, $k \in [1, l-1]$, is randomly selected and two new binary strings are obtained by swapping the bit values to the right of the position k. Thus, one offspring takes the first part of parent 1, up to k, and the second part of parent 2, from $k+1$ to $lchrom$, and the other offspring takes the first part of parent 2, up to k, and the second part of parent 1, from $k+1$ to $lchrom$. Here is an example with $lchrom = 7$ and $k = 3$:

100|1101 parent 1 ,

011|1000 parent 2 .

The resulting offspring are

1001000 offspring 1 ,

0111101 offspring 2.

A total of $N/2$ pairs (where N is an even integer) are selected. A probability that crossover takes place on a given selected pair i, $i \in [1, N/2]$ is given by $pcross$.

If a two-point crossover is used, two integer numbers l and m in the interval $[1, lchrom - 1]$, $l\langle\ \rangle m$ are randomly selected. Two offspring are created by swapping the bits in the interval $[l+1, m]$. One offspring takes the first part of parent 1, up to l,

[6] MSE's are calculated with one-folded cross-validation (i.e., squared error is calculated on one pattern when the parameters are chosen by training on the other patterns). For brevity, we simply refer to it as mean squared error in the text rather than cross-validated mean squared error.

the second part of parent 2, from $l+1$ to m, and the third part from parent 1, from $m+1$ to $lchrom$. The other offspring takes the first part of parent 2, up to l, and the second part of parent 1, from $l+1$ to m, and the third part from parent 2, from $m+1$ to $lchrom$. Here is an example with $lchrom = 10$ and $l = 3$ and $m = 7$:

100|1101|001 parent 1 ,

011|1000|100 parent 2.

The resulting offspring are

1001000001 offspring 1 ,

0111101100 offspring 2.

Mutation randomly changes the value of a position within a binary string. Each position has a probability of *pmut* of being altered by mutation, independent of other positions.

During the crossover stage, the pair of strings that are selected to participate in the recombination of genetic material are recorded as parent strings. Once crossover is applied, two offspring are recorded for each parent pair. If crossover takes place, the resulting offspring consist of recombined genetic material. If crossover does not take place, copies of two parents are made and they are recorded as two offspring. In either case, offspring may undergo further alterations via mutation. Each new offspring that did not appear in any previous generation is used to construct a network architecture in the way described above. The local minimization procedure is applied to select a network that is used for the fitness evaluation of a newly created offspring. The fitness of new offspring can be lower or higher than their parents'.

As long as there is diversity in the population of strings, both crossover and mutation will continue introducing new, different offspring which may be less fit than their parents. Over time, the effect of crossover is reduced due to reproduction, but mutation will keep introducing diversity into the population. While the effects of mutation are beneficial in the initial stages of a simulation, they become disruptive in the later stages, preventing the convergence of the population.

Some of the applications of evolutionary algorithms deal with this problem by reducing the rate of mutation exogenously after a given number of iterations. Others employ some sort of the *elitist procedure* designed to discard the offspring that are less fit than their parents. We use the *election operator* to determine the offspring that will replace their parent in the population of neural networks' architectures. It is applied in the following way. There are $N/2$ parent pairs in the population and $N/2$ offspring pairs associated with each parent pair. Fitness values of a pair of parents and a pair of their offspring are ranked, and two strings with the highest fitness values are selected. In case of a tie, a string (two strings) is (are) selected randomly.

A new population of strings consists of selected parents and offspring. Since their fitness values have already been evaluated, they undergo a new application of reproduction, crossover, and mutation. Once crossover and mutation have taken place, parents

and offspring are again subjected to the election operator. The initial population of binary strings is randomly generated. A simulation is terminated once all the population converges to a single architecture.

4. Simulations

The long-term behavior of dissipative systems can be expected to settle into simple patterns of motion such as a fixed point or a limit cycle. In contrast, the long-term dynamics of some dissipative systems display highly complex, chaotic dynamics in a strange attactor. Strange attactors has drawn attention from a wide spectrum of disciplines inclusive of both natural and social sciences. The interest originates from the an inter-disciplinary interest such as the understanding of climate, brain activity, economic activity, dynamics behind financial markets, turbulence are only a few to list here. Here, we use the Hénon map [37]), a two-dimensional mapping with a strange attactor, as a model of our simulations. The Hénon map is given by

$$\begin{aligned} x_{t+1} &= 1 - 1.4x_t^2 + z_t \,, \\ z_{t+1} &= 0.3x_t \,. \end{aligned} \tag{3}$$

The matrix of derivatives of the Hénon map is

$$\begin{bmatrix} -2.8x_t & 1 \\ 0.3 & 0 \end{bmatrix}. \tag{4}$$

Since the determinant of this matrix is constant, the Lyapunov exponents [7] for this map satisfy $\lambda_1 + \lambda_2 = ln(0.3) \approx -1.2$. The two largest Lyapunov exponents of the Hénon map are 0.408 and -1.620 so that this map exhibits chaotic behavior.

The observations are generated by

$$y_t = x_t + \gamma\varepsilon_t, \quad \varepsilon_t \sim U(0,1). \tag{5}$$

The degree of the measurement noise γ is set to 0, 0.05 and 0.1 and generated from a uniform random number generator. Data sets consist of 1100 observations where the last 10% of the data is used as a prediction sample.

[7] Let $f : R^n \to R^n$ define a discrete dynamical system and select a point $x \in R^n$. Let $(Df)_x$ be the matrix of partial derivatives of f evaluated at the point x. Suppose that there are subspaces $R^n = V_t^1 \supset V_t^2 \cdots \supset V_t^{n+1} = \{0\}$ in the tangent space of R^n at $f^t(x)$ and $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$ such that $(Df^t)_x(V_t^j) \subseteq V_{t+1}^j$, $\dim V_t^j = n+1-j$ and $\lambda_j = \lim_{t\to\infty} t^{-1} \ln||(Df^t)_x v||$ for all $v \in V_0^j \setminus V_0^{j+1}$. Then the λ_j are called the Lyapunov exponents of f. For an $n-$dimensional system as above, there are n exponents which are customarily ranked from largest to smallest: $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$. It is a consequence of Oseledec's Theorem [38], that the Lyapunov exponents exist for a broad class of functions. Also see Raghunathan [39], Ruelle [40] and Cohen et al. [41] for precise conditions and proofs of the theorem.

Lyapunov exponents measure the average exponential divergence or convergence of nearby initial points in the phase space of a dynamical system. A positive Lyapunov exponent is a measure of the average exponential divergence of two nearby trajectories whereas a negative Lyapunov exponent is a measure of the average exponential convergence of two nearby trajectories. If a discrete nonlinear system is dissipative, a positive Lyapunov exponent is an indication that the system is chaotic.

In order to examine the performance of our algorithm we conducted a number of simulations with the following parameter settings. The population size was equal to 50. The number and type of inputs were evolved such that the maximum number of inputs was set to $li = 2$ or $li = 5$. In the case of the Hénon map, the interpretation of $li = 2$ is that the values of x_t and x_{t-1} can be used as input values in networks' training and the interpretation of $li = 5$ is that the values of x_t, x_{t-1}, x_{t-2}, x_{t-3}, and x_{t-4} can be used in networks' training. The number of intervals for the initial weight range was set to $lw = 4$. The four different ranges for the initial weights were: $[-0.125, 0.125]$, $[-0.25, 0.25]$, $[-0.5, 0.5]$, and $[-1, 1]$. The number of bits used to encode the number of hidden units was set to $lh = 4$. This means that a network could have a maximum of 16 hidden units. We used the tournament selection and one-point crossover for the set of simulations reported in this paper. The rate of crossover, *pcross*, was set to 0.6 and the rate of mutation, *pmut*, was set to 0.0033.[8] The election operator was used in all of the above simulations. In addition, we conducted three simulations without the election operator.

Simulations are terminated when a genetic algorithm population converges to a single string. In each generation, only the newly created strings that were not members of previous generations are decoded and the resulting networks are trained using the local minimization technique. The performance measurements for strings that were members of the previous generation are kept and carried over. Over time, as the population starts convergence towards a single string due to the effects of reproduction and election, a smaller and smaller number of strings is evaluated. Thus, during the course of evolution, as the diversity of the genetic algorithm population decreases, the computational time required for training of the networks substantially decreases as well.

The Schwarz information criteria (SIC) is calculated by

$$SIC = \log(MSE) + q\frac{\log(n)}{n}\,, \tag{6}$$

where MSE is the mean squared error from the training set, q is the total number of parameters in the network and n is the number of observations in the training sample.

In order to evaluate the prediction performance of each network, we report the percentage *sign* predictions and the mean squared prediction error (MSPE) for the prediction sample. We also report the values of AIC and SIC. The Akaike information criteria (AIC) is calculated by

$$AIC = \log(MSE) + \frac{2q}{n}\,, \tag{7}$$

where MSE, q and n are as in (6).

We examine the following questions using the results of our simulations. First, how does the performance of the networks selected by the genetic algorithm compare to the performance of the networks selected by the standard model selection criteria, such as

[8] Three simulations were conducted with the mutation rate of 0.033. These simulations did not converge to a single network in 30 generations. Populations were characterized by a high degree of population variability at the end of each of these simulations.

SIC and AIC? Second, what is the impact of the evolution of initial weight range and inputs on the algorithm's performance of selected networks as measured by MSPE? Third, how does the use of the election operator affect the algorithm's performance and its speed of convergence?

4.1. Model selection methodology: Genetic algorithm versus SIC and AIC

The initial genetic algorithm population consisting of 50 strings is randomly generated. Then, information encoded in each string is used to construct 50 neural network architechtures. At the stage of the local minimization, 500 sets of starting values are used to choose the best starting point for each of the 50 architectures. After the local minimization, the SIC and AIC for each architecture is calculated from these initial 50 networks. The network architectures corresponding to the smallest SIC and AIC values are chosen as the SIC and AIC selection based network architectures.

The results of the comparison of the networks selected by the genetic algorithm and the networks selected by the SIC and AIC indicate that the network complexity selected by the genetic algorithm is larger than the network complexity selected by the SIC and AIC. At the same time, the genetic algorithm selects the networks with the value of the MSPE equal to or lower than the value of the MSPE of the networks selected by the SIC and AIC.

Tables 1 and 2 contain comparison between the networks chosen by the genetic algorithm and the networks chosen by the SIC and AIC selection criteria. Table 1 shows results with 2 ($li = 2$) and Table 3 with 5 ($li = 5$) inputs. Each table consists of three panels, showing results for three different levels of noise, $\gamma = 0.00$ (panel a), $\gamma = 0.05$ (panel b), and $\gamma = 0.1$ (panel c). For $\gamma = 0.0$ and for $\gamma = 0.05$, GA converges in 6, and for $\gamma = 0.1$, it converges in 7 generations. There are two common features of the genetic algorithm selected architectures. The first one is that the genetic algorithm selects network complexity with a larger number of hidden units than SIC and AIC. The second is that the genetic algorithm selected networks that have lower MSPE compared to the networks selected by SIC and AIC. For example, in Table 1(a), the genetic algorithm selects a network with seven hidden units while the network selected by SIC and AIC has five hidden units. At the same time, the MSPE ratio shows that the genetic algorithm improves on MSPE of the SIC and AIC model by 42%. In Table 1(b) and (c), there is a measurement noise added to the Hénon map which are $\gamma = 0.05$ and 0.1, respectively. In both tables, the genetic algorithm chooses larger number of hidden units but smaller MSPEs. In Table 1(b), the SIC- and AIC-based network complexities are eight hidden units whereas the GA-based network complexity is 12. On the other hand, the MSPE of the GA-based network is 16% smaller than that of the SIC and AIC architectures. In Table 1(c), the difference between the number of hidden units indicated by AIC (SIC) versus GA are substantially different. AIC and SIC indicate rather a small and parsimonious network with three hidden units whereas the GA indicates a network with 14 hidden units. However, the MSPE Ratio is in the favor of the GA-based network architecture based on the MSPE performance. In all

Table 1
Flexible number of inputs[a]

(a) $\gamma = 0$, $li = 2$, selected inputs $= x_t, x_{t-1}$, convergence in generation 6

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	5	1.00	1.516e-06	−13.29	−13.40
AIC	5	1.00	1.516e-06	−13.29	−13.40
GA	7	1.00	1.067e-06	−13.54	−13.69

Criteria	MSPE Ratio
SIC/GA	1.42
AIC/GA	1.42

(b) $\gamma = 0.05$, $li = 2$, selected inputs $= x_t, x_{t-1}$, convergence in generation 6

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	8	0.99	1.120e-03	−6.81	−7.02
AIC	8	0.99	1.120e-03	−6.81	−7.02
GA	12	0.99	9.655e-04	−6.56	−7.01

Criteria	MSPE Ratio
SIC/GA	1.16
AIC/GA	1.16

(c) $\gamma = 0.1$, $li = 2$, selected inputs $= x_t, x_{t-1}$, convergence in generation 7

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	3	0.99	4.938e-03	−4.67	−5.04
AIC	3	0.99	4.938e-03	−4.67	−5.04
GA	14	0.99	4.109e-03	−2.97	−4.42

Criteria	MSPE Ratio
SIC/GA	1.20
AIC/GA	1.20

[a]H.U. refers to the number of hidden units in a feedforward network. *Sign* is the sign predictions. Sign predictions are expressed in percentage and 1.00 refers to 100%. *MSPE* is the mean squared prediction error. *SIC* and *AIC* refer to the Schwarz and Akaike's information criteria. *GA* refers to genetic algorithm. γ is the level of measurement noise and li is the number of inputs in a feedforward network.

three panels, the GA-based model selection criteria settles for two inputs (x_t, x_{t-1}) as expected.

In Table 2, the results of simulations with 5 inputs ($li = 5$) are reported for $\gamma = 0.00$, 0.05, and 0.1. For $\gamma = 0.00$, GA converges in eight generations, for $\gamma = 0.05$, it converges in 10 generations, and for $\gamma = 0.1$, it converges in eight generations. The results display the same features as those described for the case with two inputs in Table 1. In Table 2(a), all three methods indicate the same number of hidden units although the network selected by the GA provides 44% reduction in the MSPE relative to the one selected by SIC and AIC. The interpretation of this result is that SIC- and AIC-based network gets stuck in a local optimum as it is directly obtained from the optimization of the initial 50 network architectures from the starting population. One important point to make here is that 500 sets of starting values are used to choose the best starting point

Table 2
Flexible number of inputs[a]
(a) $\gamma = 0$, $li = 5$, selected inputs $= x_t, x_{t-2}, x_{t-3}$, convergence in generation 8

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	8	1.00	2.573-e06	−1.09	−1.23
AIC	8	1.00	2.573-e06	−1.09	−1.23
GA	8	1.00	1.785-e06	−1.11	−1.23

Criteria	MSPE Ratio
SIC/GA	1.44
AIC/GA	1.44

(b) $\gamma = 0.05$, $li = 5$, selected inputs $= x_t, x_{t-1}, x_{t-2}$, convergence in generation 10

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	2	0.991	1.700-e03	−5.52	−6.01
AIC	6	0.982	1.298-e03	−5.19	−6.02
GA	12	0.991	1.027-e03	−5.52	−5.71

Criteria	MSPE Ratio
SIC/GA	1.66
AIC/GA	1.26

(c) $\gamma = 0.1$, $li = 5$, selected inputs $= x_t, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}$, convergence in generation 8

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	5	0.982	5.121-e03	−4.03	−4.74
AIC	5	0.973	3.986-e03	−4.02	−4.88
GA	12	0.973	3.465-e03	−1.81	−4.02

Criteria	MSPE Ratio
SIC/GA	1.48
AIC/GA	1.15

[a]H.U. refers to the number of hidden units in a feedforward network. *Sign* is the sign predictions. Sign predictions are expressed in percentage and 1.00 refers to 100%. *MSPE* is the mean squared prediction error. *SIC* and *AIC* refer to the Schwarz and Akaike's information criteria. *GA* refers to genetic algorithm. γ is the level of measurement noise and li is the number of inputs in a feedforward network.

for the optimization of each of the 50 networks for the initial generation. The SIC- and AIC-based networks are determined from the optimization of these initial 50 networks. Given the results in Table 2(c), it can be argued that even a large number of starting points (500×50 in our case) may not be enough to reach a global optimum. Hence, a genetic algorithm may serve as a more robust global search method.

In Table 2(b), the number of hidden units for the GA based network is again substantially larger than that of the AIC- or SIC-based networks. The MSPEs, though, is in favor of the GA network which are 66% and 26% gains relative to the SIC and AIC networks. In Table 2(c), a similar pattern emerges such that the GA chooses a larger network with a smaller MSPE relative to SIC- and AIC-based model selection methods.

Table 3
Fixed number of inputs[a]
(a) $\gamma = 0$, $li = 5$, convergence in generation 10[a]

Criteria	H.U.	Sign	MSPE	SIC	AIC
GA fixed	9	1.000	3.207-06	−9.07	−11.40
Criteria	MSPE Ratio				
GA fixed/ GA flexible	1.8				

(b) $\gamma = 0.05$, $li = 5$, convergence in generation 12

Criteria	H.U.	Sign	MSPE	SIC	AIC
GA fixed	6	0.99	1.028e-03	−4.82	−6.00
Criteria	MSPE Ratio				
GA fixed/ GA flexible	1.00				

(c) $\gamma = 0.1$, $li = 5$, convergence in generation 23

Criteria	H.U.	Sign	MSPE	SIC	AIC
GA fixed	6	0.972	4.108e-03	−3.44	−4.62
Criteria	MSPE Ratio				
GA fixed/ GA flexible	1.19				

[a]H.U. refers to the number of hidden units in a feedforward network. *Sign* is the sign predictions. Sign predictions are expressed in percentage and 1.00 refers to 100%. *MSPE* is the mean squared prediction error. *SIC* and *AIC* refer to the Schwarz and Akaike's information criteria. *GA fixed* refers to genetic algorithm with fixed number inputs (Table 2). *GA flexible* refers to genetic algorithm with flexible number of inputs. γ is the level of measurement noise and li is the number of inputs in a feedforward network.

In particular, the GA-based network complexity in Tables 1(b) and (c), and 2(b) and (c) are worth noticing. In all of these four tables, the GA-based networks have substantially larger number of hidden units and have smaller MSPEs relative to the networks indicated by SIC and AIC. It is also noticable that GA based networks have higher SIC and AIC values than the SIC (AIC)-based networks. This is mostly due to a much larger number of parameters in larger networks in the GA-based networks. The penalty factor from the increase in the number of parameters outweigh the reduction in the mean squared error in the training set. All sign predictions in Tables 1 and 2 are comparable and no model selection method has significant advantage over another in terms of sign predictions. Overall, the results indicate that SIC- and AIC-based network selection criteria over-penalize larger networks, settle for parsimonious but inferior networks in terms of MSPE performance. If the out of sample predictability is an important factor from the modelling perspective, then GA-based model selection methodology provides better forecast accuracy here.

4.2. Impact of the evolvable number and type of inputs

In Table 3(a)–(c), the results of simulations with a fixed number of inputs are presented. The number of inputs (li) is set to 5. In the dynamics of the Hénon map, there are only two lags and working with a fixed number of five lags as inputs leads to overparametrization. This overparametrized design is compared to the flexible case with five inputs from Table 2.

In Table 3(a), the case with no measurement noise is studied with $\gamma = 0.00$. The genetic algorithm with fixed number of inputs selects a network that has a mean squared prediction error that is 1.8 times larger than the mean squared prediction error of the network selected by the genetic algorithm with flexible number of inputs for the same level of noise from Table 2(a). The number of hidden units between fixed and the flexible designs are not significantly different with the fixed design having nine hidden units relative eight hidden units for the flexible design.

In case of γ=0.05, the mean squared error of the genetic algorithm with fixed number of inputs is equal to the one of the network chosen by the genetic algorithm with flexible number of inputs. The number of hidden units in the fixed design is substantially smaller with 6 hidden units relative to 12 hidden units in the flexible design case.

Finally, for $\gamma = 0.1$, the network selected by the genetic algorithm with fixed number of inputs, has a mean squared error 1.18 times larger than the network chosen by the genetic algorithm with fixed number of inputs. The fixed design case has six hidden units whereas the flexible design case has 12 hidden units. Although the GA with fixed number of inputs invariably chooses networks with smaller number of hidden units, it has a larger number of input units compared with the flexible design case. As reported in Table 2, the flexible design networks settle for three inputs rather than opting for the full set of five inputs. One noticable comparison is the Tables 2(c) and 3(c) where flexible and fixed design networks both settle for five inputs with a noise level of $\gamma = 0.1$. The flexible design selects a network with 12 hidden units whereas, a fixed design selects a six hidden unit network. Since the MSPE ratio is in favor of the flexible design model, a less parsimonious model is preferred based on its forecast accuracy. The sign predictions between the fixed and the flexible design do not exhibit significant difference.

Finally, simulations with fixed number of inputs took longer to converge, 10 generations for $\gamma = 0.0$, 12 generations for $\gamma = 0.05$, and 23 generations for $\gamma = 0.1$, compared to the speed of convergence of simulations with flexible inputs.

4.3. Impact of the evolvable initial weight range

Table 4illustrates the impact of the choice of the initial weight range in a simulation where the studied example is the Hénon map without the measurement noise ($\gamma = 0$) and with two inputs.[9] In Table 4, the results of two networks are presented. The first

[9] Similar findings prevail with measurement noise and larger number of inputs.

Table 4
Impact of evolving initial weight range[a] ($\gamma = 0$, $li = 2$)[a]

Criteria	H.U.	Sign	MSPE	SIC	AIC
GA(init)	7	1.00	2.270e-05	−10.53	10.68
GA(fin)	7	1.00	1.067e-06	−13.45	−13.69
Criteria	MSPE Ratio				
GA(init)/GA(fin)	22.7				
		Weights	Inputs	H.U.	
GA(init)	11 11 0110	4	1,2	7	
GA(fin)	10 11 0110	3	1,2	7	

[a]H.U. refers to the number of hidden units in a feedforward network. *Sign* is the sign predictions. Sign predictions are expressed in percentage and 1.00 refers to 100 percent. *MSPE* is the mean squared prediction error. *SIC* and *AIC* refer to the Schwarz and Akaike's information criteria. *GA* (*init*) refers to the network that had the same architecture as the network selected by the genetic algorithm except for the initial weight range; *GA* (*fin*) refers to the network selected by the genetic algorithm.

one, GA(init), is the member of the initial genetic algorithm population. The second one, GA(fin), is the network architecture to which the genetic algorithm converged. The two networks are equal in the number of inputs, type of inputs and in the number of hidden units. They differ in the initial weight range only. The initial weight range of the first network is equal to 4 while the initial weight range of the second network is equal to 3. Table 4 indicates that the MSPE of the first network is 21.3 times larger than the MSPE of the second network. This again indicates the importance of the global search for the parameter surface in appropriate directions. As the example demonstrates, the genetic algorithm improves substantially in terms of the MSPE of the selected network by searching starting parameter regions for the local optimizer.

4.4. The election operator

The role of the election operator is to speed up the genetic algorithm's convergence. It prevents offspring whose fitness value is lower than their parents' to enter into the genetic algorithm population. On the other hand, if the fitness value of an offspring is higher than the parents' fitness values, the offspring is admitted into the population. Thus, if the evolution finds a superior network architecture, the election operator will accept it as a new member of the genetic algorithm population. The operator leaves room for improvements while at the same time it lowers the realized rate of mutation over time and reduces the amount of noise introduced into the population. Table 5 presents the distribution of a final population in a simulation which was conducted without the election operator and which was terminated at generation 25. The simulation was conducted with no measurement noise ($\gamma = 0$) and with 2 inputs ($li = 2$).[10] At generation 25, there is significant diversity in the population. The simulation that was conducted with the same parameter settings, but with the addition of the election

[10] Similar findings prevail with measurement noise and larger number of inputs.

Table 5
Final population without election operator ($\gamma = 0$, $li = 2$)[a]

H.U.	Total	Weights				Inputs	
		1	2	3	4	1	2
9	2	0	0	2	0	0	2
11	20	0	0	20	0	0	20
13	13	0	0	13	0	0	13
14	1	0	0	1	0	0	1
15	14	0	0	13	1	0	13

[a] γ is the level of measurement noise and li is the number of inputs in a feedforward network.

operator, converged after 5 generations. In addition, the values of MSPE of the networks generated in simulations without the election operator at the time when these simulations were terminated (generation 25) were higher than the values of MSPE of the networks selected in the genetic algorithm with the election operator. Overall, simulations with the election operator converged much faster and resulted in the selected networks with lower values of MSPE. As can be seen from Tables 1 and 2, convergence was achieved in 10 generations or less in simulations with flexible inputs.

5. An empirical example

In this section, the daily spot rates French franc are studied. The data set is from the EHRA macro tape of the Federal Reserve Bank for the period of January 3, 1985 to July 7, 1992, for a total of 1886 observations. The daily returns are calculated as the log differences of the levels. All five series exhibit slight skewness and high kurtosis which is common in high frequency financial time-series data. The first 10 autocorrelations ($\rho_1, \ldots, \rho_{10}$) and the Bartlett standard errors from these series exhibit evidence of autocorrelation. The Ljung–Box–Pierce statistics reject the null hypothesis of identical and independent observations. The last 10% of a data set is used as the prediction sample.

The population size was equal to 50. The number and type of inputs were evolved such that the maximum number of inputs was set to $li = 5$. The number of intervals for the initial weight range was set to $lw = 4$. The four different ranges for the initial weights were: $[-0.125, 0.125]$, $[-0.25, 0.25]$, $[-0.5, 0.5]$, and $[-1, 1]$. The number of bits used to encode the number of hidden units was set to $lh = 4$. This means that a network could have a maximum of 16 hidden units. The tournament selection and one-point crossover are used in the genetic algorithm design. The rate of crossover, *pcross*, was set to 0.6 and the rate of mutation, *pmut*, was set to 0.0033. The election operator is used in the calculations.

In the implementation of the genetic algorithm, a set of 50 initial strings is generated. Each string is decoded to obtain the corresponding network structure with an initial weight range. At the stage of the local minimization, 500 sets of starting values are used to choose the best starting point for each of the 50 networks. After the local

Table 6
French franc ($li = 5$, *Selected inputs* $= x_t, x_{t-1}, x_{t-2}, x_{t-4}$, convergence in generation 19)[a]

Criteria	H.U.	Sign	MSPE	SIC	AIC
SIC	1	0.55	6.94-05	−9.879	−9.895
AIC	1	0.55	6.94-05	−9.879	−9.895
GA	15	0.492	5.875-05	−9.472	−9.777
Criteria	MSPE Ratio				
SIC/GA	1.18				
AIC/GA	1.18				

[a]H.U. refers to the number of hidden units in a feedforward network. *Sign* is the sign predictions. Sign predictions are expressed in percentage and 1.00 refers to 100%. *MSPE* is the mean squared prediction error. *SIC* and *AIC* refer to the Schwarz and Akaike's information criteria. *GA* refers to genetic algorithm. γ is the level of measurement noise and li is the number of inputs in a feedforward network.

minimization, the fitness function for each network is calculated and the genetic operators are used to update the current population network architectures. Finally, the members of the new population are determined and the local minimization is performed on the members of this population. The calculations are terminated when a genetic algorithm population converges to a single string.

The results in Table 6 indicate that the GA model performs 18% higher forecast accuracy relative to the SIC- and AIC-based model selection methods. Although, five lags are allowed as inputs, the GA converges to a network with four most recent lags. The convergence is reached in generation 19. The GA model produces a sign prediction of 49% whereas, the sign predictions of the SIC (AIC)-based models are 55%. One remarkable observation is the complexities of the networks chosen by the GA versus SIC (AIC). The GA method settles for a network with 15 hidden units whereas the SIC (AIC) method chooses a much simpler network with one hidden unit. Although the forecast accuracy (when measured in terms of the mean squared prediction error) is higher in the GA-based methodology, the GA-based model is much less parsimonious. Overall, the results with the foreign exchange returns confirm the simulation findings that GA models perform better in terms of the forecast performance but it is less parsimonious.

6. Conclusions

This paper proposes a model selection methodology for choosing optimal feedforward network complexity from data. The proposed methodology is completely data driven. The methodology uses the genetic algorithm to search for optimal feedforward network architectures. The genetic algorithm consists of binary strings such that each binary string encodes the information about the range of network initial weights, the number and type of inputs, and the number of hidden units of a feedforward network. Feedforward networks which are constructed from the decoded information are trained using a local search technique. The mean squared error of a network is used as the

measure of performance of a binary string. In general, other types of fitness functions can also be used and this choice depends on the nature of the problem. For instance, the fitness function can be chosen such that it corresponds to maximum expected profit or maximum expected returns in financial applications.

The results of this paper indicate that the genetic algorithm as a model selection criterion selects networks with lower values of MSPE but a larger number of hidden units compared to the more traditional model selection methods such as the SIC and the AIC. In addition, allowing the number and type of inputs to evolve results in networks with lower MSPE compared to the networks with a fixed number of inputs. Evolution of the range of initial weights results in a decrease in the values of MSPE of the network architectures selected by the genetic algorithm. Finally, the election operator greatly reduces the amount of time required for the genetic algorithm's convergence. Simulations in which the election operator was used also resulted in the selection of networks with lower MSPE than the networks generated in simulations in which the election operator was not used.

Acknowledgements

Jasmina Arifovic gratefully acknowledges financial support from the Social Sciences and Humanities Research Council of Canada. Ramazan Gençay gratefully acknowledges financial support from the Natural Sciences and Engineering Research Council of Canada and the Social Sciences and Humanities Research Council of Canada.

References

[1] J.L. Elman, Finding structure in time, Cognitive Sci. 14 (1990) 179–211.
[2] M.I. Jordan, Serial order: a parallel distributed processing approach, UC San Diego, Institute for Cognitive Science Report 8604, 1980.
[3] R. Gençay, W.D. Dechert, An algorithm for the n Lyapunov exponents of an n-dimensional unknown dynamical system, Physica D 59 (1992) 142–157.
[4] R. Gençay, Nonlinear prediction of noisy time series with feedforward networks, Physics Lett. A 187 (1994) 397–403.
[5] R. Gençay, A statistical framework for testing chaotic dynamics via Lyapunov exponents, Physica D 89 (1996) 261–266.
[6] W.D. Dechert, R. Gençay, Lyapunov exponents as a nonparametric diagnostic for stability analysis, J. Appl. Econometrics 7 (1992) 41–60.
[7] W.D. Dechert, R. Gençay, The topological invariance of Lyapunov exponents in embedded dynamics, Physica D 90 (1996) 40–55.
[8] C. Kuan, T. Liu, Forecasting exchange rates using feedforward and recurrent neural networks, J. Appl. Econometrics 10 (1995) 347–364.
[9] N. Swanson, H. White, A model-selection approach to assessing the information in the term structure using linear models and artificial neural networks, J. Busi. Econ. Statist. 13 (1995) 265–275.
[10] J.M. Hutchinson, A.W. Lo, T. Poggio, A nonparametric approach to pricing and hedging derivative securities via learning network, J. Finance 3 (1994) 851–889.
[11] R. Garcia, R. Gençay, Pricing and hedging derivative securities with neural networks and a homogeneity hint, J. Econometrics 94 (2000) 93–115.

[12] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.
[13] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd Edition, Springer, New York, 1996.
[14] D. Fogel, L. Fogel, V. Porto, Evolving neural networks, Biol. Cybernet. 63 (1990) 487–493.
[15] F. Menczer, D. Parisi, Evidence of hyperplanes in the genetic learning of neural networks, Biol. Cybernet. 66 (1992) 283–289.
[16] D. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, in: Proceedings of Eleventh International Joint Conference on Artifical Intelligence, N.S. Sridharan (Ed.), Morgan Kaufman Publishers, 1989.
[17] S. Saha, J. Christensen, Genetic design of sparse feedforward neural networks, Inform. Sci. 79 (1994) 191–200.
[18] G. Miller, P. Todd Hedge, Designing neural networks, Neural Networks 4 (1991) 53–60.
[19] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithm and neural networks: optimizing connections and connectivity, Computing 14 (1989) 347–361.
[20] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, Using genetic search to exploit the emergent behavior of neural networks, Physica D 42 (1990) 244–248.
[21] S. Harp, T. Samad, A. Guha, Toward the genetic synthesis of neural networks. In: Proceedings of the Third International Conference on Genetic Algorithms, J.D. Schaffer (Ed.), San Mateo, CA, Morgan Kaufman, 1989, pp. 762–767.
[22] H. Kitano, Designing neural networks using genetic algorithms with graph generation system, Complex Systems 4 (1990) 461–476.
[23] H. Kitano, Evolution, complexity, entropy and artificial reality, Physica D 75 (1994) 239–263.
[24] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1995.
[25] J. Arifovic, Genetic algorithm and the Cobweb model, J. Econ. Dyn. Control 18 (1994) 3–28.
[26] A.R. Gallant, H. White, There exists a neural network that does not make avoidable mistakes, Proceedings of the Second Annual IEEE Conference on Neural Networks, San Diego, CA, IEEE Press, New York, 1998, pp. I.657–I.664.
[27] A.R. Gallant, H. White, On learning the derivatives of an unknown mapping with multilayer feedforward networks, Neural Networks 5 (1992) 129–138.
[28] G. Cybenko, Approximation by superposition of a sigmoidal function, Math. Control, Signals Systems 2 (1989) 303–314.
[29] K.-I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183–192.
[30] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[31] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Networks 3 (1990) 551–560.
[32] C.-M. Kuan, H. White, Artificial neural networks: an econometric perspective, Econometric Rev. 13 (1994) 1–91.
[33] H. White, Artificial Neural Networks: Approximation & Learning, Blackwell, Cambridge, 1992.
[34] M. Heerema, W.A. van Leeuven, Derivation of Hebb's rule, J. Phys. A 32 (1999) 263–286.
[35] D.O. Hebb, The Organization of Behavior, New York, Wiley, 1949.
[36] H. White, Some asymptotic results for learning in single hidden layer feedforward network models, J. Amer. Statist. Assoc. 94 (1989) 1003–1013.
[37] M. Hénon, A two-dimensional mapping with a strange attactor, Commun. Math. Phys. 50 (1976) 69–77.
[38] V.I. Oseledec, A multiplicative ergodic theorem. Liapunov characteristic numbers for dynamical system, Trans. Moscow Math. Soc. 19 (1968) 197–221.
[39] M.S. Raghunathan, A proof of Oseledec's multiplicative ergodic theorem, Israel J. Math. 32 (1979) 356–362.
[40] D. Ruelle, Ergodic Theory of differentiable dynamical systems, Publ. Math. Inst. Hautes Etudes Scientifiques 50 (1979) 27–58.
[41] J.E. Cohen, J. Kesten, C.M. Newman (Eds.), Random Matrices and Their Application. Contemporary Mathematics, Vol. 50, American Mathematical Society, Providence, RI, 1986.