Genet Program Evolvable Mach
DOI 10.1007/s10710-016-9276-6

Recurrent Cartesian Genetic Programming of Artificial Neural Networks

Andrew James Turner[1] · Julian Francis Miller[1]

Received: 5 November 2015 / Revised: 5 July 2016

Abstract Cartesian Genetic Programming of Artificial Neural Networks is a NeuroEvolutionary method based on Cartesian Genetic Programming. Cartesian Genetic Programming has recently been extended to allow recurrent connections. This work investigates applying the same recurrent extension to Cartesian Genetic Programming of Artificial Neural Networks in order to allow the evolution of recurrent neural networks. The new Recurrent Cartesian Genetic Programming of Artificial Neural Networks method is applied to the domain of series forecasting where it is shown to significantly outperform all standard forecasting techniques used for comparison including autoregressive integrated moving average and multilayer perceptrons. An ablation study is also performed isolating which specific aspects of Recurrent Cartesian Genetic Programming of Artificial Neural Networks contribute to it's effectiveness for series forecasting.

Keywords Cartesian Genetic Programming · Genetic Programming · NeuroEvolution · Forecasting

1 Introduction

NeuroEvolution (NE) is the application of Evolutionary Algorithms to the training of Artificial Neural Networks (ANNs) [15, 78]. Early work in NE evolved the connection weights of fixed topology ANNs [53, 77]; referred to as Conventional NE (CNE). This method brought many advantages over popular gradient based methods, such as standard Back Propagation [56]. These advantages include: ability to escape local

✉ Andrew James Turner
andrew.turner@york.ac.uk

Julian Francis Miller
julian.miller@york.ac.uk

1 Department of Electronics, University of York, York, UK

optima, reduced sensitivity to initial connection weights, suitability for deep ANNs and an ability to handle non-differentiable neuron transfer functions [79]. NE is also suited to supervised and reinforcement learning applications, whereas back propagation alone is suited only to supervised learning. Other ANN training methods such as restricted Boltzmann machines are also suited to unsupervised learning [60], whereas NE is typically not.

A significant advantage of many NE methods is they allow the evolution of network topology in addition to the connection weights.[1] Such methods include GNARL [2], NEAT [61], SAGA [10] and CGPANNs [35, 64]. The ability to automatically find suitable topologies is significant as topology has been shown to strongly influence the effectiveness of back propagation [40] and weight only evolving NE [65]. Indeed, evolving the topology of ANNs may even be more significant than solely evolving connection weights [65]. Although some non-evolutionary ANN training methods do adapt topology, they typically achieve this by iteratively adding or removing neurons during training. This approach is akin to a local search of topologies, and is consequently likely to become trapped in topology local optima [2].

It has previously been shown that NE produced results that are comparable with back propagation applied to hand-crafted topologies [8]. This demonstrates the benefit of topology optimising NE, the topology is self optimising and does not have to be hand-crafted by the user. Additionally, gradient descent methods struggle to train deep ANNs [17, 40], whereas the depth of the network has no impact on NE algorithms. This, coupled with the fact that deep neural networks are thought to be more efficient in terms of the number of neurons required to solve a given task [5], suggests there may be further advantages to topology optimisation via NE.

Cartesian Genetic Programming (CGP) [44, 46] is a form of Genetic Programming (GP) [38, 49] which represents computational structures as directed acyclic graphs. This brings many advantages over the more commonly used tree structure. For instance: CGP is naturally suited to multiple-input multiple-output (MIMO) tasks, it allows internally calculated values to be reused, it benefits from explicit neutral genetic drift and does not suffer from program bloat.

Cartesian Genetic Programming of Artificial Neural Networks (CGPANNs) [35, 64] is a NE method based on CGP. The CGPANNs technique is a weight and topology optimising NE method capable of evolving homogeneous and heterogeneous ANNs. Recently CGP, the algorithm on which CGPANNs is based, has been extended to be capable of evolving recurrent programs [69, 70]. This technique is called Recurrent Cartesian Genetic Programming (RCGP). This paper presents the application of the same recurrent extension to CGPANNs to allow the evolution of recurrent ANNs. The method is referred to as Recurrent Cartesian Genetic Programming of Artificial Neural Networks (RCGPANNs).

This paper investigates the suitability of RCGPANNs using the application of series forecasting (time series prediction) [12, 26]. Series forecasting is an important application of machine learning and statistical modelling techniques, including GP [16, 31, 39, 57] and ANNs [21, 83, 84], finding application in many disciplines

[1] Sometimes referred to as TWEANNs: Topology and Weight Evolving Artificial Neural Networks.

including: economics, politics and planning. Series forecasting is also a common application of NE [13, 32, 61] including CGPANNs [32, 52].

CGPANNs with an imposed Jordan type architecture[2] [30] has previously been used to create a form of recurrence with application to series forecasting [33]. However such a method is much more limited in terms of the topologies which can be produced, as the user must preselect which outputs (no internal recurrence is present) are to be fed back as inputs; that is to say it is much more restricted. The method proposed in this work allows any recurrent topology to be created, limited only by the total number of nodes and their maximum arity. Standard CGP has also recently been applied to the domain of series forecasting [71].

We examine the suitability of the proposed RCGPANNs algorithm by comparing its performance with standard CGP, RCGP and CGPANNs. This enables an evaluation of the various extensions which have been applied to CGP in order to create RCGPANNs. Firstly the benefit of the recurrent extension is evaluated by comparing CGP and RCGP as well as comparing CGPANNs with RCGPANNs. Secondly the benefit of optimising ANNs rather than using standard mathematical functions commonly used by GP is evaluated by comparing CGP and CGPANNs as well as RCGP and RCGPANNs. This ablation study[3] allows insight into which aspects of the RCGPANNs approach are beneficial.

We also evaluate the performance of RCGPANNs generally by comparing their effectiveness with two naive and three more complex standard forecasting methods: random walk, mean, exponential smoothing (EPS), autoregressive integrated moving average (ARIMA) and multilayer perceptrons (MLP) respectively. The comparison with at least two naive and two complex standard forecasting methods (including mean and ARIMA) follows the methodology recommended by Hyndman [26], an acknowledged expert in the field of forecasting, on benchmarking new forecasting methods.[4] Comparisons to MLPs are also made as they represent the current standard approach for training ANNs.

The remainder of this paper is as follows. Section 2 introduces and describes the newly proposed RCGPANNs algorithm. Section 3 describes how CGP and its derivatives can be applied to series forecasting. Section 4 introduces the standard series forecasting methods used for comparison and Sect. 5 describes the benchmarks to which they are applied. Finally, Sect. 6 presents the results of the experiments undertaken with a discussion and conclusions given in Sects. 7 and 8 respectively.

2 Recurrent Cartesian Genetic Programming of Artificial Neural Networks

The RCGPANNs algorithm introduced in this paper is a combination of two techniques: RCGP and CGPANNs. Accordingly, this section first introduces CGP, the base algorithm, followed by the two extensions utilised by RCGPANNs; the

[2] A topology where certain outputs are made available as inputs.

[3] Here the term ablation study refers to repeatedly investigating the algorithm with an individual component removed (ablated) in order to isolate it's influence on the algorithm as a whole.

[4] This particular advice is given on his personal blog http://robjhyndman.com/hyndsight/benchmarks/.

ability to create recurrent networks and the application to evolving ANNs. Once these two extensions of CGP have been introduced, their combination as RCGPANNs is then described. Interested practitioners are directed to an open source implementation of CGP, RCGP, CGPANN and RCGPANN which includes documentation and tutorials for use and [67].

2.1 Cartesian Genetic Programming

Cartesian Genetic Programming [44, 46] is a form of GP [38, 49] which typically evolves directed acyclic computational structures of nodes (graphs) indexed by their Cartesian coordinates. CGP does not suffer from program bloat [43, 66]; a recognized drawback of many GP methods [59]. CGP chromosomes contain explicitly inactive or non-functioning genes which are subject to neutral genetic drift aiding the escape from local optima and giving improved navigation of the search landscape [73, 81]. CGP typically uses point or probabilistic mutation, no crossover,[5] and a $(1 + \lambda)$-ES. Although CGP chromosomes are of static size, the number of active nodes varies during evolution enabling variable length phenotypes. The user specifies a *maximum* number of available nodes, of which only a proportion are active (used). Overestimating the number of available nodes has shown to greatly aid evolution [45, 63]; which is thought to heighten neutral genetic drift but could also be compensating for length bias [18, 19].

The reason it is thought that such a simple evolutionary strategy is so effective for CGP is twofold. Firstly, CGP does not typically utilise crossover and so there is no requirement to maintain genetic diversity. Secondly, the reason this does not not lead to CGP easily becoming trapped in local optima is due to the inactive genes creating plateaus in the search space which are navigated across via neutral genetic drift [63, 73].

CGP could be described as an indirect encoding scheme due to the fact that there is a process of decoding CGP genotypes into phenotypes. However as this process only removes the explicitly redundant/inactive genes which do not contribute to phenotype semantics, it is arguably more akin to a direct encoding scheme. Interestingly, unlike most direct encoding schemes CGP contains redundant genes which do not contribute to the phenotype but can become active later during evolution.

Each CGP chromosome comprises function genes (F_i), connection genes ($C_{i,j}$) and output genes (O_i); where i indexes each node and j indexes the inputs of each node. The function genes represent indexes in a function look-up-table and describe the functionality of each node. The connection genes describe where each node gathers its inputs. For regular acyclic CGP, connection genes may connect a given node to any previous node in the program, or any of the program inputs. The output genes can address any program input or internal node and define which are used as program outputs.

Originally CGP programs were organized with nodes arranged in rows (nodes per layer) and columns (layers); with each node indexed by its row and column.

5 Applying crossover to CGP has been previously investigated [9].

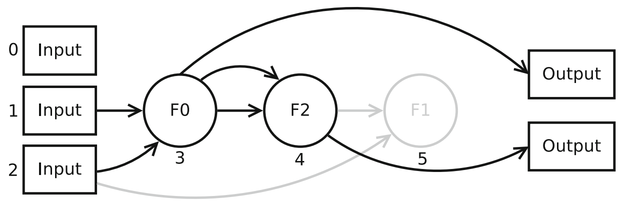

Fig. 1 Example CGP phenotype corresponding to the chromosome: 012 233 124 3 4

However, in most circumstances this is an unnecessary constraint as any configuration possible using a given number of rows and columns is also possible using one row with many columns; provided the total number of nodes remains constant. This is because CGP can evolve where each node obtains its inputs. Consequently, here the chromosomes are defined with one row and *n* columns; with each node only indexed by its column. A generic (one row) CGP chromosome is given in Eq. 1; where α is the arity of each node, *n* is the number of nodes and *m* is the number of program outputs.

$$F_0 C_{0,0} \ldots C_{0,\alpha-1} \ldots F_{n-1} C_{n-1,0} \ldots C_{n-1,\alpha-1} : O_0 \ldots O_{m-1} \tag{1}$$

An example CGP phenotype is given in Fig. 1 together with its corresponding chromosome. As can be seen, all nodes are connected to previous nodes or program inputs. Not all program inputs have to be used, enabling evolution to decide which inputs are significant. Not all available nodes have to be used giving rise to inactive genes and the ability for evolution to adapt program size. An advantage of CGP over tree-based GP, again seen in Fig. 1, is that node outputs can be reused multiple times, rather than requiring the rediscovery of the same functionality if it is needed again. In addition it can be seen that CGP is directly suited to multiple-input multiple-output problems.

2.2 Recurrent Cartesian Genetic Programming

Recurrent Cartesian Genetic Programming (RCGP) [69, 70] is a recent extension to CGP which allows recurrent or cyclic connections (i.e. feedback).

In regular CGP, connection genes are restricted to only allow connections to previous nodes in the graph; including inputs. In RCGP this restriction is lifted to allow connection genes to connect a given node to *any* node, including itself, or program input. Once the acyclic restriction is removed, RCGP solutions can contain recurrent connections. An example RCGP phenotype is given in Fig. 2 along with its corresponding chromosome.[6]

Placing no restriction on connection genes results in mutations creating as many recurrent as feed-forward connections [69]. However, it is likely that most problems

[6] RCGP chromosomes, like CGP chromosomes, can also: describe multiple-input multiple-output phenotypes, contain inactive genes and choose which inputs to utilise. These characteristics are not shown in Fig. 2 for simplicity.

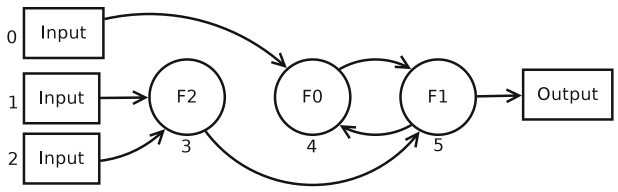

Fig. 2 Example RCGP phenotype corresponding to the chromosome: 212 005 134 5

do not require half of all the connections to be recurrent. For this reason a new parameter was introduced called *recurrent connection probability*. This parameter controls the probability that a mutation to a connection gene results in a recurrent connection. For instance a value of 5 % results in 5 % of connection gene mutations creating cyclic connections. Additionally a value of 0 % would result in only acyclic connections, thus implementing standard CGP. This parameter does not however limit the maximum or minimum number of recurrent connections (except for values of 0 and 100 %), it only places a bias on whether mutations create a recurrent connection.

RCGP chromosomes are executed identically to standard CGP chromosomes. The inputs are applied, each active node is updated in order of node index (i), and the outputs are read. The next set of inputs are then applied and the process repeated. RCGP differs from CGP in that the program output(s) can be determined by the current inputs and the current state of the internal nodes.

One important aspect of RCGP is that it makes it possible for node output values to be read before they have been calculated [69]. For this reason all nodes are initialised to output zero until they have calculated their own output value. This is akin to the initial conditions of recursive equations. Interestingly, other non-zero initial node values might be more suitable depending upon the transfer functions used [70], but this is not investigated here.

An alternative, simpler, but less flexible method of using CGP to create recurrent programs is to enforce a Jordan type architecture [34, 47]; where some predetermined program outputs are made available as program inputs. A slightly more complex and domain specific multi-chromosome version of CGP has also been adapted to be capable of creating transistor circuits which contain cyclic connections [74].

2.3 Cartesian Genetic Programming of Artificial Neural Networks

Cartesian Genetic Programming of Artificial Neural Networks (CGPANNs) [35, 36, 64] is the application of CGP to the creation and training of ANNs. CGP is adapted to evolving ANNs by the inclusion of connection weight genes ($W_{i,j}$) for each connection gene and by using transfer functions often used by ANNs; for instance logistic sigmoid functions. CGPANNs exhibits all of the benefits of CGP and is a NE training method which can evolve connection weights, topology [65] and transfer functions [68] of ANNs.

When initialising CGPANN chromosomes the same process is followed as for CGP. The additional connection weights not present in standard CGP are initialised as random floating point values taken from a user defined range i.e. ± 1. When mutating connection weight genes the new value is also randomly chosen from the same range. However recent studies suggest that less naive methods for connection weight manipulation may be more appropriate [82].

Although CGPANNs evolves topology, it is required that the user specifies a maximum network size (number of nodes). This could be considered a drawback, but overestimating the required number of nodes has been shown to be highly beneficial for CGP [45]. Additionally it has been shown for CGPANNs that the choice of the number of nodes has a far lower impact on performance than the choice of topology for non-topology evolving NE methods [65]. This is due to CGPANNs ability to optimise the topology including the number of nodes which are used up to the specified maximum.

In CGPANNs the user must also specify a maximum neuron arity. However, the effective arity used by each neuron can be lower than this maximum [64]. This occurs when the chromosome describes a pair of neurons that have two or more connections between each other. In this case, multiple connections between two neurons are equivalent to one connection; with the connection weight value equal to the sum of the individual weights.[7]

It is important to note that the types of ANNs created using CGPANNs are unconventional and often cannot be described using the standard terms of layers and nodes per layer. Figure 3 gives an example of the type of ANN which can be created using CGPANN. In Fig. 3 it can be seen that the connections between neurons are highly unconstrained; any neuron can receive it's inputs from any previous neuron in the network including input neurons. It can also be seen that any neuron in the network can be used as an output; again including input neurons. Figure 3 demonstrates that by allowing NE to adapt the topology, evolution is capable of discovering topologies which would be unlikely to be considered by a human designer.

2.4 Combining RCGP and CGPANNs

Recurrent Cartesian Genetic Programming of Artificial Neural Networks applies the same recurrent extension of RCGP to CGPANNs. This is undertaken to allow RCGPANNs to evolve recurrent ANNs. The modifications required to extend CGPANNs to RCGPANNs are the same as used to extend CGP to RCGP. The requirement of all connection genes to be acyclic is lifted and the probability of mutation creating recurrent connections is controlled via a recurrent connection probability. As with RCGP, the chromosomes are executed by applying each set of inputs, updating each active node/neuron in index order (i) and then reading the outputs. Again as with RCGP, this can result in node/neuron outputs being read before they have been calculated. As with RCGP, here each node/neuron is

[7] Other decoding strategies are also possible such as decoding only the first of multiple connections between two neurons in the phenotype [64].

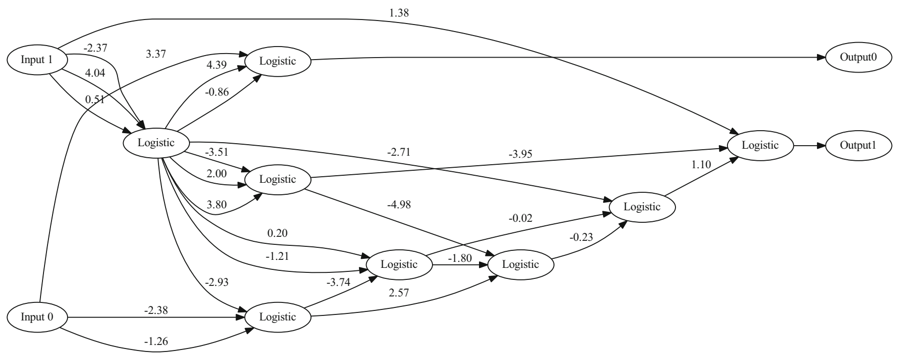

Fig. 3 Depiction of the types of ANN created using CGPANN

initialised to output zero until they have calculated their own output value; alternative initial output values may be more suitable but this is not investigated here.

Once these changes have been incorporated RCGPANNs can be used to evolve recurrent ANNs. It is important to note that RCGPANNs can create feed-forward *and* recurrent ANNs; as allowing recurrent connections does not force evolution to use them. Additionally RCGPANNs can be easily restricted to creating only feed-forward ANNs (CGPANNs) by setting the recurrent connection probability to zero. RCGPANNs is therefore a superset of CGPANNs.

3 Application to series forecasting

In this paper CGP, RCGP, CGPANNs and RCGPANNs are applied to series forecasting using a recursive forecasting method [20, 29]. This method involves the feedback of previously made forecasts as inputs to be used in the prediction of subsequent forecasts. Using this method it is possible to make forecasts to any given horizon.

A common technique used by forecasting techniques is to calculate the embedding dimension (D) and time delay (T) of the training data. This provides a suitable number of past data points, and a suitable number of time steps between these data points, in order to accurately predict the next data point. For instance if $D = 4$ and $T = 2$ then the inputs would be $[x(t), x(t-2), x(t-4), x(t-6)]$; where t indexes each sample in the series $x()$ and the task would be to predict $x(t+1)$. Here suitable embedding dimensions and time delays are calculated for each benchmark and these determine the number of past values to be used as inputs. The embedding dimensions and time delays are calculated using the pdc package [7] for the R programming language [51]; using *entropy.heuristic*.

As an example, Fig. 4 shows how recursive forecasting using multiple previous values is used. Here $D = 3$ and T is left unspecified. The buffer containing $x(t-4)$

through to $x(t)$ is initially populated with known observed values which are replaced with predicted values during the recursive forecasting process.

A disadvantage of using multiple inputs determined by D and T is that it reduces the amount of training data which can be used. For instance if $D = 2$ and $T = 2$ at time $t = 0$, $x(t - 2)$ is before the start of the training data and so $x(t + 1)$ cannot be predicted.

The fitness function used here represents how well the solutions recursively predict sections of the training data. This is achieved by recursively predicting the next fifty samples[8] from $t = 50$, $t = 100$, ..., $t = 950$. The predictions start from $t = 50$ and not $t = 0$ to compensate for the use of embedding dimensions and time delays removing the first few samples from the training data. The fitness awarded is the mean square error between the predicted and observed values.

Unlike feed-forward programs, when using RCGP and RCGPANNs the outputs are a function of the current inputs *and* the current program state (node outputs). This means the program must be 'primed' before it can be used to make forecasts. The priming process is to apply previous observed values to the program, in sequence, and execute the program in each case. The outputs are not used. This causes the internal nodes to calculate suitable values before the forecasting begins. Here, when using RCGP and RCGPANNs, the previous 50 samples from each starting point are applied to the network before making future predictions. For instance if the predictions were to be from $t = 150$ then all the values from $t = 100$ to $t = 150$ are applied in turn and the program executed in each case.

A disadvantage of many machine learning techniques is that they can easily over-fit on the training data and consequently lose their ability to generalise. CGP and its derivatives are no exception and are also likely to suffer from over-fitting when applied to series forecasting. For this reason a validation scheme is used. Here generalisation is assessed by recording how well the solutions perform beyond the forecast horizon used during training. Starting at times $t_1 = 100$, $t_2 = 200$, ..., $t_9 = 900$ the programs are used to make forecasts up to a horizon of 100 samples. The mean square error of the forecasts occurring between a time horizon of $t_i + 50$ samples and $t_i + 100$ samples are then used as a validation fitness score (where $1 \leq i \leq 9$).

The validation score could be used by any of a range of early stopping techniques [50] in order to prevent over-training. However the choice of early stopping technique is likely to influence results. For this reason, here, the chromosome which is awarded the best validation score is retained throughout evolution and is used as the final chromosome to be assessed using the testing data. For instance, if the chromosome with the best validation score is found on generation x, after the maximum number of generations have elapsed, this chromosome is used as the final chromosome to be evaluated on the testing data. Although this means the training does not stop early, in terms of the overall training time, it does help prevent over training.

[8] The number of predictions could take any value. Fifty is used here as a compromise between forecasting to a similar horizon required by the testing data and allowing for a reasonable number of separate forecasts.

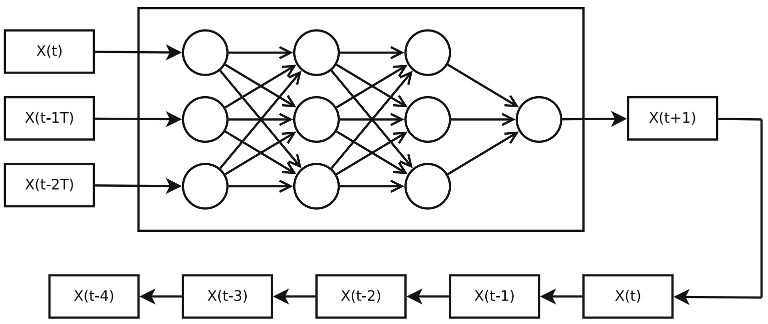

Fig. 4 Depiction of recurrent forecasting and the use of embedding dimension and time delay to determine the number of inputs; $D = 3$

In the work presented the following parameters are used: a $(1+4)$-ES, a maximum of 10,000 generations and a probabilistic mutation method.[9] In all cases the number of available nodes is set as 100. In the case of CGP and RCGP the mutation rate is set as 3 % and the function set comprises $[x_1 + x_2, x_1 - x_2, x_1 * x_2, x_1/x_2, \sin(x_1), \cos(x_1), \exp(x_1), \log(x_1)]$; where each node has an arity of two (x_1 and x_2) with some transfer functions only utilising the first input (x_1). In the case of CGPANN and RCGPANN, the mutation rate is set as 1 % and the transfer function used is unipolar logistic sigmoid with a connection weight range of ± 5. In the case of CGP and CGPANN the recurrent connection probability is set as 0 %. In the case of RCGP and RCGPANN the recurrent connection probability is set as 10 %.

The chosen parameters are relatively 'off-the-shelf' choices and have not been optimised for each benchmark. Mutation rates of 3–5 % is standard for CGP [44]. A slightly lower mutation rate is used for CGPANN, as in the authors' experience, using a lower mutation rate for CGPANN than for CGP results in better performance. Although speculative, this may be because CGPANN is more suited to gradual hill climbing through the adjustment of connection weights whereas CGP relies on larger beneficial mutations.

4 Comparative methods

A number of comparative methods are used to evaluate the performance of RCGPANNs; as well as CGP, RCGP and CGPANNs. These methods are: random walk forecasting (RWF), mean forecast (MEAN), exponential smoothing (ETS), autoregressive integrated moving average (ARIMA) and multilayer perceptrons (MLP). These methods are used to compare RCGPANNs to standard forecasting techniques and to the more common method of training ANNs.

[9] Where each gene is mutated with a given probability.

4.1 Random walk forecasting

The random walk forecasting (RWF) method is a very simple naive forecasting technique which is useful to compare new forecasting methods against; as any newly proposed forecasting method should at least be able to outperform it. RWF predicts that all future unknown values are equal to the last observed value.

4.2 Mean

The mean forecasting method is again a very simple naive forecasting technique which is also useful to compare new forecasting methods against. The mean forecasting method predicts that all future values are equal to the arithmetic mean of the observed values i.e. the training set.

4.3 Exponential smoothing

Exponential smoothing (ETS) [22] is a popular forecasting technique which, in its simplest form, bases its prediction on a weighted average of previous observations. Commonly the further ahead the prediction is from the last observation, the more previous values are used in the weighted average.

The exponential smoothing used in this paper is from the Forecast package [27] for the R programming language [51]. When creating exponential smoothing models the *ets* function is used to find suitable parameters using the methods described in [25].

4.4 Autoregressive integrated moving average

Autoregressive integrated moving average (ARIMA)[10] [6] is a popular generalised forecasting technique. ARIMA models use a collection of three forecasting techniques: autoregressive (AR), integrated (I) and moving average (MA); hence ARIMA. ARIMA models are often written in the form ARIMA(p,d,q), with the p,d and q values referring to the AR, I and MA aspects of the ARIMA model respectively. By using different p,d and q parameters ARIMA models can implement a wide range of forecasting techniques including Random-Walk, Random-trend, autoregressive and exponential smoothing models.

The ARIMA implementation used in this paper is from the Forecast package [27] for the R programming language [51]. When creating ARIMA models the *auto.arima* function [24] is used to find suitable p, d and q parameters as well as further sub parameters associated with the specific model. The *auto.arima* function uses a variation of the Hyndman and Khandakar algorithm [24] to obtain a suitable ARIMA model.

[10] Also referred to as Box–Jenkins after the original authors.

4.5 Multilayer perceptron

Multilayer perceptrons (MLPs) are a standard ANN training method which makes use of the back propagation algorithm. When applied to series forecasting it is common practice to use multiple inputs determined by the embedding dimension and time delay of the series; so this is undertaken here.

The MLP implementation used in this paper is the Fast Artificial Neural Network (FANN) library [48]. The FANN library is configured to use standard fully connected ANNs of unipolar logistic sigmoid transfer functions trained using a variant on back propagation called resilient back propagation (Rprop) [54] for 1000 epochs. As back propagation does not optimise topology a range topologies are investigated comprising one and two hidden layers of five, ten, twenty and fifty nodes per hidden layers (eight separate topologies in total).

As MLPs use a strictly supervised learning method they must be trained using input–output pairs. However this style of learning is not directly compatible with recursive forecasting. This is because future forecasts are made using previously made forecasts. When using previous forecasts as inputs the input–output pair (the current inputs and the correct outputs) do not represent a correct learning example. In this case the ANN would be trained using incorrect data.

Therefore, here, the MLPs are trained for one-step-ahead prediction; always using valid input–output pairs. The recursive forecasting performance of the ANN is then recorded after each epoch by using the ANN to recursively predict the next 100 samples starting at $t = 100$, $t = 200$, ..., $t = 900$. After the maximum number of epochs have elapsed the configuration which resulted in the best recursive forecasting performance is then returned as the final trained ANN. This method effectively trains for one-step-ahead prediction and uses the recursive forecasting performance to prevent over-training.

5 Benchmarks

In this paper three series forecasting benchmarks are utilised, one of which is a mathematical series generated from chaotic equations (Mackey–Glass) and two are real world recordings (Laser and Sunspots).

All of the benchmarks consist of a training set of 1000 data points and a testing set of 100 data points. In each case the embedding dimension and time delay are calculated using the pdc package [7].

5.1 Laser

The Laser benchmark is the recording of a “81.5-micron 14NH3 cw (FIR) laser, pumped optically by the P(13) line of an N2O laser via the vibrational aQ(8,7) NH3 transition” [23]. The benchmark was used in the Santa Fe Competition [76] and the dataset is publicly available [75].

Two versions of the dataset exist, one containing one thousand samples and another extended version with ten thousand. The one thousand sample version used by the Santa Fe Competition and the extended version is made available for further testing of methods. Here the first 1000 samples of the extended version are used as a training set and the following 100 samples are used as the testing set. This series is also normalised into a [0,1] range using Eq. 2 where: x_i is the sample to be normalised, x_i' is the normalised sample, X is the entire series and the *min* and *max* functions return the minimum and maximum sample value in the series X respectively. The Laser benchmark is plotted in Fig. 5.

The embedding dimension and time delay used for the Laser series are $D = 4$ and $T = 7$ respectively.

$$x_i' = \frac{x_i - min(X)}{max(X) - min(X)} \tag{2}$$

5.2 Mackey–Glass

The Mackey–Glass equation was originally used to model blood cell regulation [41]. However the Mackey–Glass equation has also been used as a forecasting benchmark due to its interesting chaotic properties. The Mackey–Glass equation is

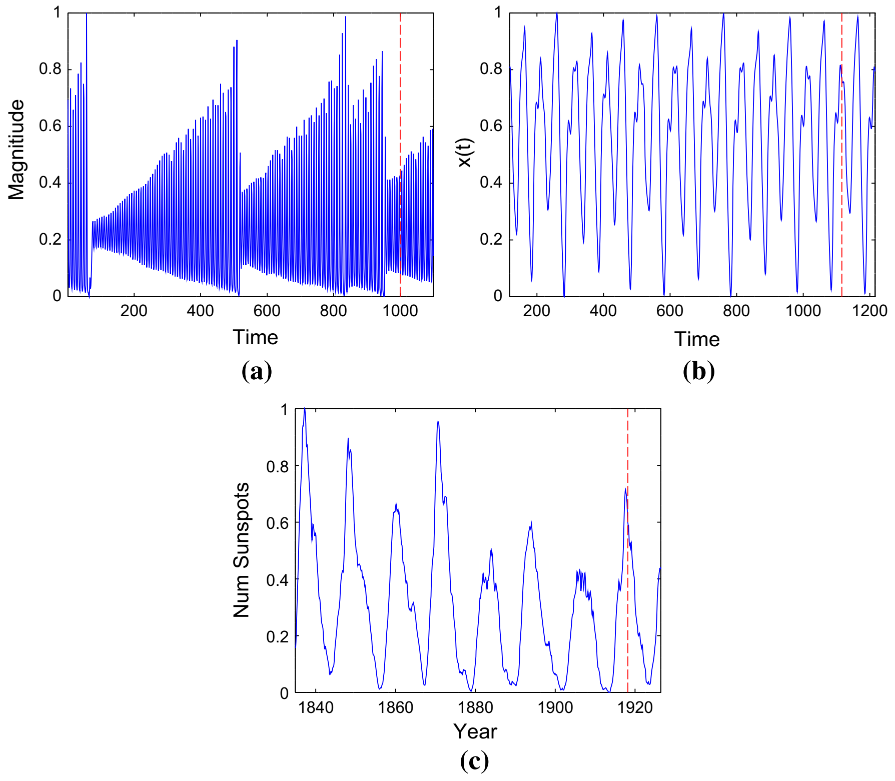

Fig. 5 Series forecasting benchmarks. **a** Laser. **b** Mackey–Glass. **c** Sunspots

given in Eq. 3. By adjusting the value of the delay parameter τ the equation produces chaotic and non-chaotic series; $\tau > 16.8$ produces chaotic behaviour.

$$\frac{dx(t)}{dt} = \frac{a \cdot x(t-\tau)}{1 + x^c(t-\tau)} - b \cdot x(t) \tag{3}$$

Here the Mackey–Glass equation parameters are set as $a = 0.2$, $b = 0.1$ and $c = 10$. The delay parameter τ is set as 17 and $x(t) = 0$ when $t \leq 0$. A series is produced using the 4th order Runge–Kutta integration method with a time step of $dt = 0.01$ s. This series is then sampled once a second to produce the series used as the benchmark. This series is also normalised using Eq. 2. The first 117 s (samples) are removed to avoid the transient response time. Then the following 1100 s (samples) are used for the training and testing sets; plotted in Fig. 5. The first 1000 s are used for training and the following 100 are used for testing.

The embedding dimension and time delay used for the Mackey–Glass series are $D = 4$ and $T = 1$ respectively.

5.3 Sunspots

Predicting the number of yearly/monthly Sunspots [58] is a commonly used [37], challenging [1], series prediction benchmark. The data is recorded by the SIDC-team, at the World Data Center for the Sunspot Index, Royal Observatory of Belgium [58] and is publicly available [55]. Here the smoothed number of monthly sunspots is used covering 1100 months (samples) of data taken from November 1834 to June 1926. The first 1000 samples are used for training with the remaining 100 used for testing. The series is once again normalised using Eq. 2. The series is plotted in Fig. 5.

The embedding dimension and time delay used for the smoothed monthly sunspots series are $D = 5$ and $T = 1$ respectively.

6 Results

The results presented investigate the suitability of RCGPANNs by isolating the benefit of the two extensions to CGP utilised by RCGPANNs; recurrence and application to ANNs. The results also investigate the suitability of RCGPANNs generally as a series forecasting method, achieved by comparing RCGPANNs with a range of standard series forecasting methods; described in Sect. 4.

There are many measurements found in the literature which are used to assess the performance of forecasting methods [4, 28]. However in the machine learning literature the most commonly used methods are the Mean Square Error (MSE), Root Mean Square Error (RMSE) and the Normalised Mean Square Error (NMSE). For this reason MSE and NMSE are used here;[11] despite other measurements possibly

[11] As RMSE is simply the root of the MSE value it is not also explicitly presented.

being more representative of forecasting accuracy [4, 28]. This provides consistency with other published machine learning methods.

The MSE and NMSE are given in Eqs. 4 and 5 respectively where: N is the number of predicted samples, p_i is the ith predicted value, o_i is the ith observed value and $\bar{o}$ is the average of all the observed values. Note that the NMSE measurement gives the MSE normalised by the MSE which would be achieved if all predictions were equal to the arithmetic mean of the observed values.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(p_i - o_i)^2 \tag{4}$$

$$\mathrm{NMSE} = \left(\frac{\sum_{i=1}^{N}(p_i - o_i)^2}{\sum_{i=1}^{N}(o_i - \bar{o})^2}\right) \tag{5}$$

The forecasts produced by the various forecasting methods are evaluated on the testing data using the two measures described. For the stochastic methods (CGP, RCGP, CGPANN, RCGPANNs and MLP) the average[12] performance of 50 runs is used for comparison; as this represents the typical performance. Additionally the testing performance of the run which scored the best training fitness is also presented. In a real scenario, this is likely the forecaster which would be used. Note this is not the solution which produced the best testing fitness, as selection should never be (and typically cannot be) based on testing performance.

In the case of MLPs, many topologies were investigated. Here the results of using the best topology are presented. The best topology is determined by the average training performance; not the testing performance which would typically not be known in advance. Specifically the recursive prediction performance on the training set is used as this more closely matches the actual final application.

Again in the case of the stochastic methods, statistical significance testing is used to assess any differences. The non-parametric Mann–Whitney U-test and the non-parametric Kolmogorov–Smirnoff test (KS) are used to test for statistical significance. Typically a value of $\rho \leq 0.05$ is used to represent statistical significance but as this work is undertaking eight pairwise comparisons Bonferroni correction [14] will be used reducing the significance level to 6.25×10^{-3}. Additionally the effect size, as defined in [72], is also used to indicate the importance of any statistical difference; with values > 0.56 indicating a small effect size, > 0.64 a medium and > 0.71 a large. The spread of results are also given graphically as box and whisker plots for visual inspection; with outliers marked as follows: '+' represents forecasts between 1.5 and 3 times the interquartile range and '$\circ$' represents forecasts greater than 3 times the interquartile range.

The forecasts produced by each method are also given in Figs. 6, 7, and 8. In the case of the stochastic methods, the best forecast as previously defined is presented.

[12] Arithmetic mean.

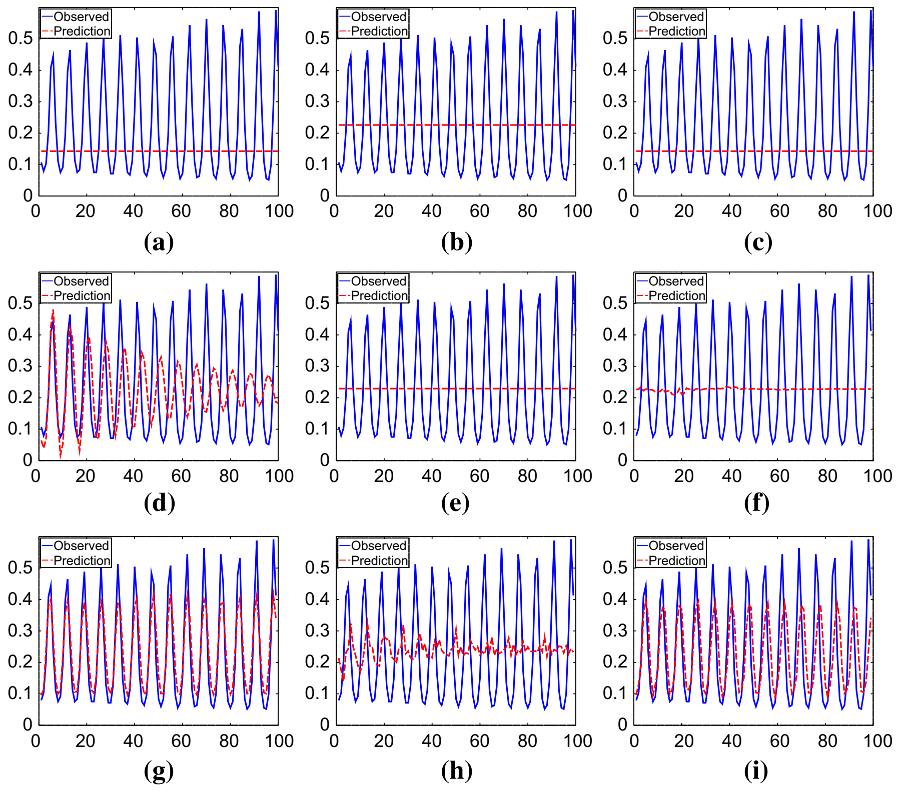

Fig. 6 Forecasts produced for the Laser benchmark. **a** RWF. **b** MEAN. **c** ETS. **d** ARIMA. **e** MLP. **f** CGP. **g** RCGP. **h** CGPANN. **i** RCGPANN

6.1 Laser

The results of applying the various series forecasting methods to the Laser benchmark are given in Table 1. In the case of the stochastic methods, Table 2 gives the statistical analysis and Fig. 9 gives the box and whisker plots. The forecasts produced are plotted in Fig. 6.

The MLP topology which produced the best recursive forecast on the training set had two hidden layers each containing five nodes. The ARIMA model produced is ARIMA(5,0,4) with non-zero mean.

Overall it can be seen in Table 1 that RCGPANNs produce the best average forecast of all the methods investigated.

When comparing CGPANNs and MLPs as training methods for feed-forward ANNs it can be seen that on average CGPANNs strongly outperformed MLPs with statistical significance and a medium effect size. This indicates that CGPANNs provide a superior training method to MLPs.

When evaluating the recurrent extension it can be seen that on average RCGP outperformed CGP but with a small effect size and no statistical significance.

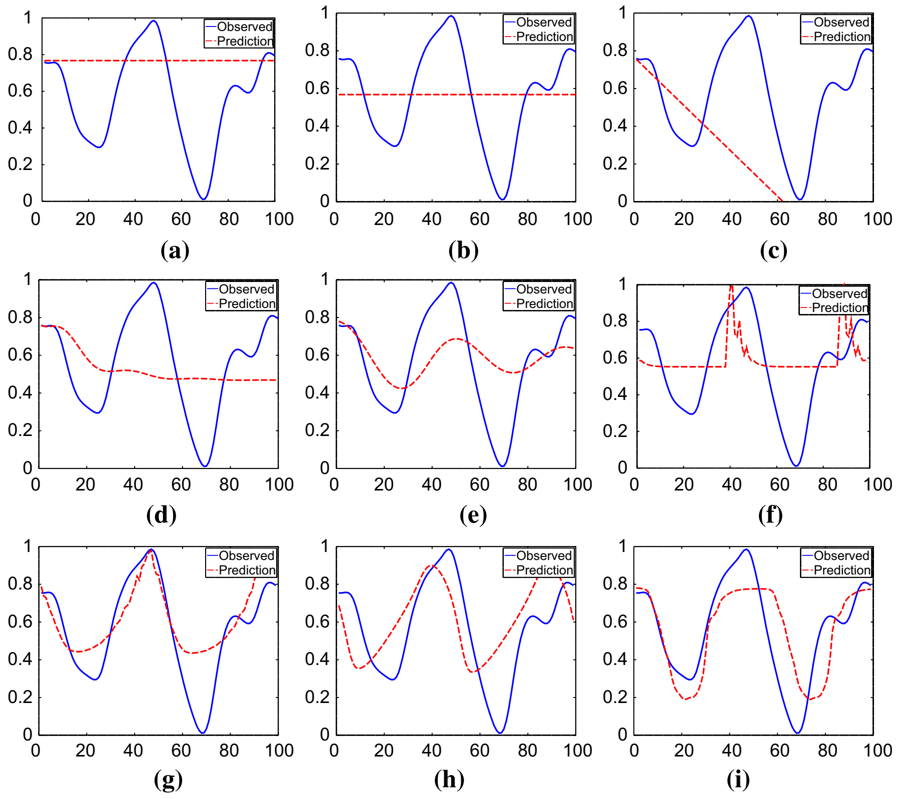

Fig. 7 Forecasts produced for the Mackey–Glass benchmark. **a** RWF. **b** MEAN. **c** ETS. **d** ARIMA. **e** MLP. **f** CGP. **g** RCGP. **h** CGPANN. **i** RCGPANN

Conversely it can be seen that on average RCGPANNs outperformed CGPANNs with statistical significance and a large effect size. This indicates than the recurrent extension is advantageous to RCGPANNs and is not detrimental to CGP.

Finally when comparing evolving ANNs to the use of standard GP mathematical functions it can be seen that CGPANNs outperformed CGP but with no statistical significance and a small effect size. Conversely RCGPANNs outperformed RCGP with a medium effect size and statistical significance. This indicates that evolving ANNs does not produce worse results than using standard GP mathematical functions and can produce superior results.

6.2 Mackey–Glass

The results of applying the various series forecasting methods to the Mackey–Glass benchmark are given in Table 3. In the case of the stochastic methods, Table 4 gives the statistical analysis and Fig. 9 gives the box and whisker plots. The forecasts produced are plotted in Fig. 7.

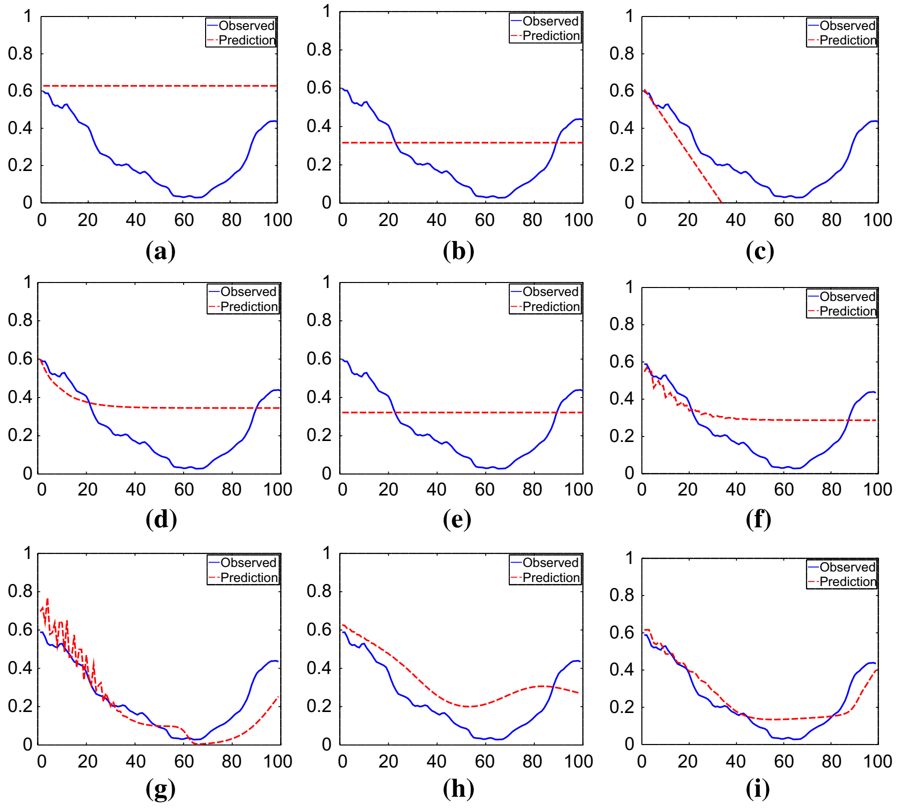

Fig. 8 Forecasts produced for the Sunspots benchmark. **a** RWF. **b** MEAN. **c** ETS. **d** ARIMA. **e** MLP. **f** CGP. **g** RCGP. **h** CGPANN. **i** RCGPANN

Table 1 Results for applying various forecasting methods to the Laser benchmark

Method	MSE		NMSE	
	Avg	Best	Avg	Best
RWF	0.034227		1.260675	
MEAN	0.027151		1.000030	
ETS	0.034223		1.260508	
ARIMA	0.034148		1.257749	
MLP	0.043985	0.035237	1.620058	1.000184
CGP	0.027946	0.027091	1.029200	0.997707
RCGP	0.025823	0.004424	0.951000	0.162913
CGPANN	0.027655	0.029380	1.018500	1.081971
RCGPANN	0.021467	0.016424	0.790580	0.604839

Table 2 Statistical significance testing between the stochastic methods applied to the the Laser benchmark

Comparison	U-test	KS-test	Effect size
CGP–MLP	3.75e−2	3.76e−8	0.66820
RCGP–MLP	5.92e−6	2.97e−9	0.76280
CGPANN–MLP	1.49e−2	7.84e−10	0.68440
RCGPANN–MLP	1.21e−10	7.84e−10	0.87340
CGP–RCGP	3.64e−2	8.90e−3	0.62160
CGPANN–RCGPANN	1.43e−5	1.08e−8	0.75200
CGP–CGPANN	4.04e−1	5.08e−1	0.54860
RCGP–RCGPANN	6.81e−3	4.43e−3	0.65720

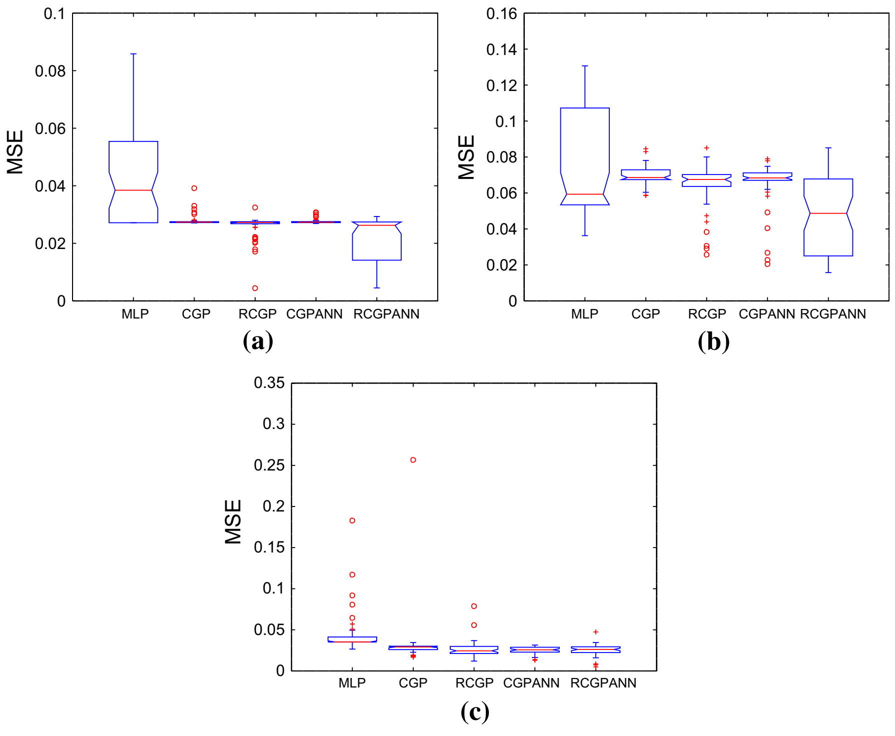

Fig. 9 Spread of the forecasts produced using stochastic methods. **a** Laser. **b** Mackey–Glass. **c** Sunspots

The MLP topology which produced the best recursive forecast on the training set had one hidden layer containing twenty nodes. The ARIMA model produced is ARIMA(3,0,5) with non-zero mean.

Overall it can be seen in Table 3 that RCGPANNs produces the best average forecast compared with all the other methods.

When comparing CGPANNs and MLPs as training methods for feed-forward ANNs it can be seen that on average CGPANNs outperformed MLPs with statistical

Table 3 Results for applying various forecasting methods to the Mackey–Glass benchmark

Method	MSE		NMSE	
	Avg	Best	Avg	Best
RWF	0.109334		1.624736	
MEAN	0.067324		1.000447	
ETS	0.357603		5.314079	
ARIMA	0.071481		1.062226	
MLP	0.075798	0.048297	1.126385	0.717701
CGP	0.069947	0.058746	1.039400	0.872979
RCGP	0.064501	0.025706	0.958500	0.381999
CGPANN	0.065563	0.049188	0.974280	0.730944
RCGPANN	0.047575	0.033219	0.706980	0.493640

Table 4 Statistical significance testing between stochastic methods on the Mackey–Glass benchmark

Comparison	U-test	KS-test	Effect size
CGP–MLP	2.34e−1	3.63e−6	0.56920
RCGP–MLP	9.42e−1	2.11e−e−2	0.50440
CGPANN–MLP	7.59e−1	1.78e−4	0.51800
RCGPANN–MLP	1.70e−4	4.23e−4	0.71840
CGP–RCGP	2.29e−2	1.71e−2	0.63220
CGPANN–RCGPANN	5.19e−6	1.02e−5	0.76460
CGP–CGPANN	2.87e−1	6.78e−1	0.56200
RCGP–RCGPANN	8.15e−5	1.78e−4	0.72880

significance but a small effect size. This indicates that MLPs and CGPANNs represent similarly suitable training methods for ANNs.

When evaluating the recurrent extension of RCGP it can be seen that on average RCGP outperformed CGP but without statistical significance and a small effect size. Additionally on average RCGPANNs outperformed CGPANNs with statistical significance and a large effect size. This indicates that the recurrent extension is advantageous to CGPANNs and is not detrimental to CGP.

Finally, when comparing evolving ANNs to the use of standard GP mathematical functions it can be seen that CGPANNs outperformed CGP but with no statistical significance and a small effect size. Conversely RCGPANNs strongly outperformed RCGP on average with statistical significance and a large effect size. This indicates that evolving ANNs may be producing better results than standard mathematical functions or at least does not produce worse results.

6.3 Sunspots

The results of applying the various series forecasting methods to the Sunspots benchmark are given in Table 5. In the case of the stochastic methods, Table 6 gives

Table 5 Results for applying various forecasting methods to the Sunspots benchmark

Method	MSE		NMSE	
	Avg	Best	Avg	Best
RWF	0.176262		6.008159	
MEAN	0.034399		1.172533	
ETS	0.546006		18.61142	
ARIMA	0.034972		1.192063	
MLP	0.043773	0.035228	1.492071	1.200790
CGP	0.031940	0.026894	1.088600	0.916592
RCGP	1.20e+30	0.011922	4.10E+31	0.406331
CGPANN	0.024991	0.018114	0.851750	0.617360
RCGPANN	0.024992	0.004925	0.851770	0.167851

Table 6 Statistical significance testing between stochastic methods on the Sunspots benchmark

Comparison	U-test	KS-test	Effect size
CGP–MLP	6.97e−13	2.13e−14	0.91680
RCGP–MLP	1.09e−11	1.09e−13	0.89440
CGPANN–MLP	6.31e−16	3.28e−18	0.96920
RCGPANN–MLP	8.10e−15	2.07e−17	0.95080
CGP–RCGP	1.31e−1	3.17e−2	0.58780
CGPANN–RCGPANN	8.12e−1	8.41e−1	0.51400
CGP–CGPANN	3.51e−3	4.43e−3	0.66960
RCGP–RCGPANN	6.52e−1	2.41e−1	0.52640

the statistical analysis and Fig. 9 gives the box and whisker plots. The forecasts produced are given in Fig. 8.

One oddity seen in the results is the extremely poor average forecast achieved using RCGP. This is due to one of the fifty runs producing multiple large spikes mid forecast, resulting in a very large error which affected the average. If this one run is removed, the average results in a MSE of 0.026701 and a NMSE of 0.910010; which does outperform CGP. This outlier is also removed from the box-plots in Fig. 9 in order for the other differences to be visible.

The MLP topology which produced the best recursive forecast on the training set had two hidden layers each containing five nodes. The ARIMA model produced is ARIMA(5,1,4).

Overall it can be seen in Table 5 that the best average result is achieved using CGPANNs and RCGPANNs with almost no difference between the two.

When comparing CGPANNs and MLPs as training methods for feed-forward ANNs it can be seen that on average CGPANNs strongly outperformed MLP with statistically significant and a large effect size. This indicates that CGPANNs are much more effective ANN training methods than MLPs.

When evaluating the recurrent extension of RCGP it can be seen that on average RCGP is strongly outperformed by CGP but with no statistical significance and a very small effect size. If the one very poor run is removed form the RCGP results, then RCGP does outperform CGP but still without statistical significance or meaningful effect size. Similarly, on average RCGPANNs produces very similar results to CGPANNs with no statistical significance and a small effect size. This indicates that the recurrent extension is not providing an advantage.

Finally when comparing evolving ANNs to the use of standard GP mathematical functions it can be seen that on average CGPANNs strongly outperformed CGP with statistical significance and a medium effect size. RCGPANNs also outperform RCGP (with or without the outlier) but this is without statistical significance and a small effect size. This indicates that evolving ANNs may be producing better results than standard mathematical functions or at least does not produce worse results.

7 Discussion

As can be seen from the results, in all cases RCGPANNs produced the best (or joint best) average forecasts compared with all the other methods used for comparison. This demonstrates RCGPANNs is a highly competitive series forecasting technique compared to a number of standard methods. RCGPANNs also outperformed all the other methods based on CGP: CGP, RCGP and CGPANN. This clearly demonstrates the suitability of the newly proposed RCGPANNs method.

When comparing CGPANNs with MLPs, CGPANNs outperformed MLPs on all three benchmarks with statistical significance. Additionally in two of the three cases CGPANNs outperformed MLPs with a medium or greater effect size. This indicates that CGPANNs is a superior training method than MLPs when applied to the domain of recursive series forecasting.

The results show that the inclusion of recurrent connections in RCGP and RCGPANNs offers an advantage when applied to series forecasting. In the case of RCGP the addition of recurrent connections always resulted in better results but without statistical significance and only with a small effect size. In the case of RCGPANNs, on two of the three benchmarks the addition of recurrent connections produced better results with statistical significance and a large effect size; with comparable results on the final benchmark. Therefore the addition of recurrent connections were often seen to be beneficial and never worse. This further demonstrates the suitability of the recurrent extension to create recurrent program structures, complementing previous research [69, 70].

In the presented work, when applying CGP and its variants to series forecasting multiple previous values from the sequence were made available to the evolved programs. These previous values were determined by the embedding dimension and time delay of the sequence. Therefore it would not have been unsurprising if the recurrent extension present in RCGP and RCGPANNs failed to outperform their non-recurrent counterparts; as a form of recurrence has effectively already been added that has been specifically designed for predicting future values. The fact that RCGP and RCGPANNs outperformed their non recurrent counterparts demonstrates that evolution has found additional recurrence which improved again on the level of recurrence already provided.

The results also show that the use of neuron transfer functions and connection weights produce better forecasts on average than the use of standard GP mathematical functions without connections weights. CGP or RCGP is never shown to outperform CGPANNs and RCGPANNs respectively. CGPANNs outperformed CGP on one of three benchmarks with statistical significance and a medium effect size; in the other cases the difference is not statistically significant. RCGPANNs outperformed RCGP on two of the three benchmarks with statistical significance and a medium or large effect size; again in the remaining case the difference is not statistically significant.

The fact that the use of neuron transfer functions and connection weights produced superior results in comparison with using standard mathematical functions may have wider implications for GP in general. It might be the case that many GP methods could be improved by using neuron transfer functions, or the addition of connection weights, or the use of both connection weights and neuron transfer functions (thus implementing NE). It is true that both ANNs [11] and GP [80] are (or can be depending on the function set used) universal approximators. However this does not necessarily indicate how trainable or evolvable the programs are. For instance the addition of connection weights to GP, previously termed weighted GP [62], might make for a more evolvable fitness landscape. Future research should investigate the use of weighted connections and neuron transfer function, independently and in union, for other GP methods. This may help indicate whether the power of ANNs is in their transfer functions, connection weights or training methods. This could even lead to interesting mixtures of GP and ANNs such as back propagation applied to a weighted form of GP.

A seemingly odd result is how well the Mean forecasting method preformed compared to the other standard forecasting methods. Mean forecasting is seen to outperform RWF, ETS, ARIMA and MLP for all of the benchmarks investigated. However, as described in [26], “some forecasting methods are very simple and surprisingly effective”. Additionally it has previously been noted that in the real-time forecasting M2-competition [42] that ARIMA (Box–Jenkins) “proved to be one of the least-accurate methods and its overall median error is 17 % greater than that for a naive forecast” [3]. Therefore it can be seen that it is not uncommon for naive methods to perform very well.

Interestingly, many of the forecasts provided using CGP and it variants either produced an output very close to a Mean forecast or exhibited behaviours which eventually settled on an output close to the Mean forecast. Examples of this can be seen in: Fig. 6f where CGP is applied to the Laser benchmark, Fig. 7o where CGP is applied to the Mackey–Glass benchmark and Fig. 8x where CGP is applied to the Sunspots benchmark. As the Mean approach is shown to produce a reasonable forecast it may be the case that this method represents a local optima in the search space. It could also be an example of evolutionary methods rediscovering a previously known technique.

Although not explored in this paper, an additional advantage of using evolutionary computation for forecasting is the ability to alter the fitness function to favour certain characteristics. For instance the forecast horizon can easily be altered. The maximum error during the forecast could be considered. Frequency information from the training data could be used to award or penalise frequencies present or not present in the produced forecasts. The fitness awarded could also represent the number of time steps predicted with an error lower than a given threshold, rather than the error up to a given forecast horizon. The solutions could also be optimised for speed, complexity or size. As the ability to set custom fitness functions also applies to NE this is another possible benefit of its use in series forecasting.

Finally the best result of the stochastic methods often represented a much superior forecast than the average. This is not surprising as early stopping methods were utilised to prevent over training. Although the average performance is used here for comparison, as it represents typical performance of the algorithms, in real applications the best of many runs could be used. In this case RCGPANNs could be argued to outperform the standard forecasting methods to an even greater extent than has been presented; although the comparison would be less rigorous.

8 Conclusion

This paper has introduced RCGPANNs, a new NE method based on CGPANNs which utilises the recurrent extension of CGP, known as RCGP. The application of series forecasting has been used to assess the performance of RCGPANNs compared to other CGP variants and a range of standard forecasting methods. The results demonstrate that RCGPANNs produce highly competitive forecasts, outperforming all of the other standard forecasting methods used for comparison. RCGPANNs is therefore shown to be a powerful NE method; at least in the domain of series forecasting.

RCGPANNs differs from standard CGP in two regards; it allows recurrent connections and uses neuron transfer function with connections weights. Both of these aspects were individually investigated revealing that they both provide benefits to standard CGP; again at least in the domain of series forecasting. This demonstrates the importance of these two previously presented CGP extensions and helps explain why the RCGPANNs approach is shown to be so effective. This result may also be significant for other GP methods which could also benefit from similar extensions.

Finally, it is important to note that RCGPANNs is a superset of CGPANNs. By setting the RCGPANNs recurrent connection probability to zero it implements standard feed-forward CGPANNs. Additionally just because RCGPANNs is capable of utilising recurrent connections does not force evolution to do so. For instance it is possible for RCGPANNs to create purely feed-forward ANNs if there were an evolutionary advantage in doing so. This, coupled with the advantageous results presented, makes RCGPANNs an important extension to CGPANNs.

References

1. L. Aguirre, C. Letellier, J. Maquet, Forecasting the time series of sunspot numbers. Sol. Phys. **249**(1), 103–120 (2008)
2. P. Angeline, G. Saunders, J. Pollack, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. **5**(1), 54–65 (1994)
3. J. Scott Armstrong (ed.), Extrapolation for time-series and cross-sectional data, in *Principles of Forecasting: A Handbook for Researchers and Practitioners* (Springer, Berlin, 2001), pp. 217–243
4. J.S. Armstrong, F. Collopy, Error measures for generalizing about forecasting methods: empirical comparisons. Int. J. Forecast. **8**(1), 69–80 (1992)
5. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks. Adv. Neural Inf. Process. Syst. **19**, 153 (2007)
6. G.E. Box, G.M. Jenkins, G.C. Reinsel, *Time Series Analysis: Forecasting and Control* (Wiley, New York, 2013)
7. A.M. Brandmaier, *pdc: Permutation Distribution Clustering* (2014). R package version 0.5. http://CRAN.R-project.org/package=pdc
8. E. Cantú-Paz, C. Kamath, An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. IEEE Trans. Syst/ Man Cybern. Part B Cybern. **35**(5), 915–927 (2005)
9. J. Clegg, J.A. Walker, J.F. Miller, A new crossover technique for Cartesian Genetic Programming, in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1580–1587. ACM (2007)
10. D. Cliff, I. Harvey, P. Husbands, Incremental evolution of neural network architectures for adaptive behaviour, in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'93)*, pp. 39–44 (1992)
11. G. Cybenko, Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
12. J.G. De Gooijer, R.J. Hyndman, 25 years of time series forecasting. Int. J. Forecast. **22**(3), 443–473 (2006)
13. J.P. Donate, G.G. Sanchez, A.S. de Miguel, Time series forecasting. A comparative study between an evolving artificial neural networks system and statistical methods. Int. J. Artif. Intell. Tools **21**(01) (2012). doi:10.1142/S0218213011000462
14. O.J. Dunn, Multiple comparisons among means. J. Am. Stat. Assoc. **56**(293), 52–64 (1961)
15. D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning. Evol. Intell. **1**(1), 47–62 (2008)
16. S. Gaur, M. Deo, Real-time wave forecasting using genetic programming. Ocean Eng. **35**(11), 1166–1172 (2008)
17. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, *in Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10).* Society for Artificial Intelligence and Statistics (2010)
18. B. Goldman, W. Punch, Analysis of Cartesian genetic programmings evolutionary mechanisms. IEEE Trans. Evol. Comput. **PP**(99), 1–1 (2014). doi:10.1109/TEVC.2014.2324539 (in press)
19. B.W. Goldman, W.F. Punch, Length bias and search limitations in Cartesian Genetic Programming, in *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, pp. 933–940. ACM (2013)
20. L.J. Herrera, H. Pomares, I. Rojas, A. Guillén, A. Prieto, O. Valenzuela, Recursive prediction for long term time series forecasting using advanced models. Neurocomputing **70**(16), 2870–2880 (2007)
21. H.S. Hippert, C.E. Pedreira, R.C. Souza, Neural networks for short-term load forecasting: a review and evaluation. IEEE Trans. Power Syst. **16**(1), 44–55 (2001)
22. C.C. Holt, Forecasting seasonals and trends by exponentially weighted moving averages. Int. J. Forecast. **20**(1), 5–10 (2004)
23. U. Huebner, N. Abraham, C. Weiss, Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH 3 laser. Phys. Rev. A **40**(11), 6354 (1989)
24. R.J. Hyndman, Y. Khandakar, Automatic time series forecasting: the forecast package for R. J. Stat. Softw. **27**(3) (2008). doi:10.18637/jss.v027.i03

25. R.J. Hyndman, M. Akram, B.C. Archibald, The admissible parameter space for exponential smoothing models. Ann. Inst. Stat. Math. **60**(2), 407–426 (2008)
26. R.J. Hyndman, G. Athanasopoulos, Forecasting: principles and practice, in *OTexts* (2014). https://www.otexts.org/fpp/
27. R.J. Hyndman, G. Athanasopoulos, S. Razbash, D. Schmidt, Z. Zhou, Y. Khan, C. Bergmeir, E. Wang, forecast: Forecasting functions for time series and linear models (2014). R package version 5.4. http://CRAN.R-project.org/package=forecast
28. R.J. Hyndman, A.B. Koehler, Another look at measures of forecast accuracy. Int. J. Forecast. **22**(4), 679–688 (2006)
29. Y. Ji, J. Hao, N. Reyhani, A. Lendasse, Direct and recursive prediction of time series using mutual information selection, in *Proceedings of the 8th International Conference on Artificial Neural Networks: Computational Intelligence and Bioinspired Systems*, pp. 1010–1017. Springer, Berlin (2005)
30. M.I. Jordan, *Serial Order: A Parallel Distributed Processing Approach* (Tech. rep, Institute for Cognitive Science, 1986)
31. M.A. Kaboudan, Genetic programming prediction of stock prices. Comput. Econ. **16**(3), 207–236 (2000)
32. G.M. Khan, S. Khan, F. Ullah, Short-term daily peak load forecasting using fast learning neural network, in *11th International Conference on Intelligent Systems Design and Applications (ISDA)*, 2011, pp. 843–848. IEEE (2011)
33. G.M. Khan, A.R. Khattak, F. Zafari, S.A. Mahmud, Electrical load forecasting using fast learning recurrent neural networks, in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6. IEEE (2013)
34. M. Khan, G. Khan, J. Miller, Efficient representation of recurrent neural networks for markovian/non-markovian non-linear control problems, in *2010 10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 615–620. IEEE (2010)
35. M.M. Khan, M.A. Ahmad, M.G. Khan, J.F. Miller, Fast learning neural networks using Cartesian Genetic Programming. Neurocomputing **121**, 274–289 (2013)
36. M.M. Khan, G.M. Khan, J.F. Miller, Evolution of neural networks using cartesian genetic programming, in *Proceedings of IEEE World Congress on Computational Intelligence CEC 2010* (2010)
37. M. Khashei, M. Bijari, An artificial neural network (p, d, q) model for timeseries forecasting. Expert Syst. Appl. **37**(1), 479–489 (2010)
38. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
39. W.B. Langdon, W. Banzhaf, Repeated sequences in linear genetic programming genomes. Complex Syst. **15**(4 (c)), 285–306 (2005)
40. H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks. J. Mach. Learn. Res. **10**, 1–40 (2009)
41. M.C. Mackey, L. Glass, Oscillation and chaos in physiological control systems. Science **197**(4300), 287–289 (1977)
42. S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, L.F. Simmons, The M2-competition: a real-time judgmentally based forecasting study. Int. J. Forecast. **9**(1), 5–22 (1993)
43. J.F. Miller, What bloat? Cartesian genetic programming on Boolean problems, in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 295–302 (2001)
44. J.F. Miller (ed.), *Cartesian Genetic Programming* (Springer, Berlin, 2011)
45. J.F. Miller, S. Smith, Redundancy and computational efficiency in Cartesian Genetic Programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
46. J.F. Miller, P. Thomson, Cartesian genetic programming, in *Proceedings of the Third European Conference on Genetic Programming (EuroGP)*, vol. 1820, pp. 121–132. Springer, Berlin (2000)
47. M. Minarik, L. Sekanina, Evolution of iterative formulas using Cartesian Genetic Programming, in *Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 11–20. Springer, Berlin (2011)
48. S. Nissen, Implementation of a fast Artificial Neural Network library (FANN). Report, Department of Computer Science, University of Copenhagen (DIKU) (2003)
49. R. Poli, W.W.B. Langdon, N.F. McPhee, J.R. Koza, A field guide to Genetic Programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008)
50. L. Prechelt, Early stopping—but when?, in *Neural Networks: Tricks of the Trade* (Springer, Berlin 2012), pp. 53–67

51. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2014). http://www.R-project.org/
52. M. Rehman, J. Ali, G.M. Khan, S.A. Mahmud, Extracting trends ensembles in solar irradiance for green energy generation using neuro-evolution, in *Artificial Intelligence Applications and Innovations* (Springer, Berlin 2014), pp. 456–465
53. R.K. Belew, J. McInerney, N.N. Schraudolph, Evolving networks: using the genetic algorithm with connectionist learning. Tech. rep., Cognitive Computer Science Research group, Computer Science and Engr. Dept (C-014), Univ. California at San Diego (1990)
54. M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591. IEEE (1993)
55. Royal Observatory of Belgium: World data center for the production, preservation and dissemination of the international sunspot number (2014). http://sidc.be/silso/home
56. D.E. Rumelhart, G.E. Hintont, R.J. Williams, Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
57. M. Santini, A. Tettamanzi, J.F. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G. Tettamanzi, W.B. Langdon, Genetic programming for financial time series, in *Genetic Programming, Proceedings of EuroGP'2001*, vol. 2038, pp. 361–370. Springer (2001)
58. SIDC-Team: The International Sunspot Number. Monthly Report on the International Sunspot Number, online catalogue (1700–1987)
59. S. Silva, E. Costa, Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genet. Program. Evol. Mach. **10**(2), 141–179 (2009)
60. P. Smolensky, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Chap. Information Processing in Dynamical Systems: Foundations of Harmony Theory* (MIT Press, Cambridge, 1986)
61. K. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
62. H.C. Tsai, Using weighted genetic programming to program squat wall strengths and tune associated formulas. Eng. Appl. Artif. Intell. **24**(3), 526–533 (2011)
63. A.J. Turner, J.F. Miller, Neutral genetic drift: an investigation using Cartesian Genetic Programming. Genet. Program. Evol. Mach. **16**(4), 531–558 (2015)
64. A.J. Turner, J.F. Miller, Cartesian Genetic Programming encoded Artificial Neural Networks: a comparison using three benchmarks, in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-13)*, pp. 1005–1012 (2013)
65. A.J. Turner, J.F. Miller, The importance of topology evolution in neuroevolution: a case study using cartesian genetic programming of artificial neural networks, in M. Bramer, M. Petridis (eds.), in *Research and Development in Intelligent Systems XXX* (Springer, Berlin 2013), pp. 213–226. doi:10.1007/978-3-319-02621-3_15. http://link.springer.com/chapter/10.1007%2F978-3-319-02621-3_15
66. A.J. Turner, J.F. Miller, Cartesian Genetic Programming: why no bloat?, in *Genetic Programming: 17th European Conference*, vol. 8599, EuroGP-2014, LNCS (Springer, Berlin, 2014), pp. 193–204
67. A.J. Turner, J.F. Miller, Introducing a cross platform open source cartesian genetic programming library. Genet. Program. Evol. Mach. **16**(1), 83–91 (2014). doi:10.1007/s10710-014-9233-1
68. A.J. Turner, J.F. Miller, NeuroEvolution: evolving heterogeneous artificial neural networks. Evol. Intell. **7**(3), 135–154 (2014). doi:10.1007/s12065-014-0115-5
69. A.J. Turner, J.F. Miller, Recurrent Cartesian Genetic Programming, in *13th International Conference on Parallel Problem Solving from Nature (PPSN 2014), LNCS*, vol. 8672, pp. 476–486 (2014)
70. A.J. Turner, J.F. Miller, Recurrent Cartesian genetic programming applied to famous mathematical sequences, in *Proceedings of the Seventh York Doctoral Symposium on Computer Science and Electronics*, pp. 37–46 (2014)
71. A.J. Turner, J.F. Miller, Recurrent Cartesian genetic programming applied to series forecasting, in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-15)*, pp. 1499–1500 (2015)
72. A. Vargha, H.D. Delaney, A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J. Edu. Behav. Stat. **25**(2), 101–132 (2000)
73. V.K. Vassilev, J.F. Miller, The advantages of landscape neutrality in digital circuit evolution, in *Proceedings of International Conference on Evolvable Systems,LNCS*, vol. 1801, pp. 252–263. Springer (2000)
74. J.A. Walker, K. Völk, S.L. Smith, J.F. Miller, Parallel evolution using multi-chromosome Cartesian Genetic Programming. Genet. Program. Evol. Mach. **10**(4), 417–445 (2009)

75. A. Weigend, Santa fe competition data sets (2014). http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html
76. A.S. Weigend, N.A. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past* (Addison-Wesley, Reading, 1994)
77. A. Wieland, Evolving neural network controllers for unstable systems, in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, 1991, vol. 2, pp. 667–673. IEEE (1991)
78. X. Yao, A review of evolutionary artificial neural networks. Int. J. Intell. Syst. **8**(4), 539–567 (1993)
79. X. Yao, Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)
80. X. Yao, Universal approximation by genetic programming, in *Foundations of Genetic Programming* (1999)
81. T. Yu, J. Miller, Neutrality and the evolvability of Boolean function landscape, in *Genetic Programming*, vol. 2038, Lecture Notes in Computer Science, ed. by J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, W. Langdon (Springer, Berlin, 2001), pp. 204–217
82. E. Z-Flores, L. Trujillo, O. Schütze, P. Legrand, A local search approach to genetic programming for binary classification, in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15*, pp. 1151–1158. ACM, New York, NY, USA (2015). doi:10.1145/2739480.2754797
83. G. Zhang, B.E. Patuwo, M.Y. Hu, Forecasting with artificial neural networks: the state of the art. Int. J. Forecast. **14**(1), 35–62 (1998)
84. G.P. Zhang, B.E. Patuwo, M.Y. Hu, A simulation study of artificial neural networks for nonlinear time-series forecasting. Comput. Oper. Res. **28**(4), 381–396 (2001)