Noname manuscript No.
(will be inserted by the editor)

NeuroEvolution: Evolving Heterogeneous Artificial Neural Networks

Andrew James Turner · Julian Francis Miller

the date of receipt and acceptance should be inserted later

Abstract NeuroEvolution is the application of Evolutionary Algorithms to the training of Artificial Neural Networks. Currently the vast majority of NeuroEvolutionary methods create homogeneous networks of user defined transfer functions. This is despite NeuroEvolution being capable of creating heterogeneous networks where each neuron's transfer function is not chosen by the user, but selected or optimised during evolution. This paper demonstrates how NeuroEvolution can be used to select or optimise each neuron's transfer function and empirically shows that doing so significantly aids training. This result is important as the majority of NeuroEvolutionary methods are capable of creating heterogeneous networks using the methods described.

Keywords Heterogeneous Artificial Neural Networks · NeuroEvolution · Evolutionary Algorithms · Artificial Neural Networks · Computational intelligence · Cartesian Genetic Programming

1 Introduction

NeuroEvolution (NE) is the application of Evolutionary Algorithms (EA) to the training of Artificial Neural Networks (ANN) [11,47]. NE's history began by evolving the connection weights of fixed topology ANNs [30,45]. This method brought many advantages over the still popular gradient based methods; such as simple

Andrew James Turner
The University of York
Electronics Department
Intelligent Systems Group
York, UK
E-mail: andrew.turner@york.ac.uk

Julian Francis Miller
The University of York
Electronics Department
Intelligent Systems Group
York, UK
E-mail: julian.miller@york.ac.uk

back propagation [31]. These advantages include: being able to natively escape local optima, being less sensitive to the initial connection weights, being suited to deep ANNs and not requiring that each neuron's Transfer Function (TF) be differentiable [48]. NE is also suited to reinforcement learning as well as supervised learning; whereas back propagation is only suited to supervised learning. Other ANN training methods such as restricted Bolzmann machines are also suited to unsupervised learning [34].

A significant advantage of NE is its ability to evolve the topology of ANNs; as well as the connection weights. Topology evolving NE methods include: GNARL [1], NEAT [35], SAGA [7] and CGPANN [14] [38]. This ability to automatically create suitable topologies is significant as topology has been shown to strongly influence the effectiveness of back propagation [16] and weight only evolving NE [39]. Evolving the topology of ANNs has even been shown to be more important to training than evolving connection weights [39]. 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 locally sub-optimal topologies [1]. It has been shown that using simple back propagation with hand crafted topologies produce results as good as NE [4]. This result demonstrates the benefit of topology optimising NE; the topology is also optimised and does not have to hand crafted by trial and error. Finally gradient descent methods struggle to train deep ANNs [12] [16] 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 task [3] is another benefit of topology optimising NE.

Interestingly, NE can also be used to optimise the TF of each neuron within heterogeneous ANNs[1]. However, this capability of NE has been widely overlooked in recent research. Indeed, at the turn of the 21st century many ANN publications stated that more research was required concerning the optimisation of TFs: "Relatively little has been done on the evolution of node transfer functions, let alone the simultaneous evolution of both topological structure and node transfer functions" [48], "The current emphasis in neural network research is on learning algorithms and architectures, neglecting the importance of transfer functions" [8] and "Selection and/or optimisation of transfer functions performed by artificial neurons have been so far little explored ways to improve performance of neural networks in complex problems" [9]. However, a search of the literature reveals that there has been little active research in this area. This paper intends to help fill this gap by showing how NE can easily optimise neuron TFs during evolution and that doing so produces strongly beneficial results.

The remainder of this paper is structured as follows. Section 2 discusses research literature relating to the application NE to the evolution of the TFs of ANNs. Section 3 describes the investigations which were undertaken using NE to evolve heterogeneous ANNs, leading to the results given in Section 4. Finally Section 5 discusses the overall findings with closing conclusions given in Section 6.

[1] Homogeneous ANNs are ANNs where each neuron's TF is identical. Heterogeneous ANNs are ANNs where each neuron's TF is not identical.

2 Background

There are many ANN TFs found in the literature [9]. However, the majority of NE implementations only evolve homogeneous ANNs of logistic or Gaussian functions, which have both been shown capable of universal approximation; [13] and [26] respectively. Of those which do evolve heterogeneous ANNs, there are two main methods.

The first method selects the TF of each neuron from a predetermined list of TFs. Training methods which use this method include General Neural Networks (GNN) [17]; which randomly adds or removes logistic or Gaussian TFs using an evolutionary programming method. GNN is also a hybrid approach which makes use of back propagation during training. Other NE methods which select specific TFs for each neuron include Parallel Distributed Genetic Programming (PDGP) [28], a modified Hierarchical Co-evolutionary Genetic Algorithm ($HCGA_2$) [44] and Cartesian Genetic Programming of Artificial Neural Networks (CGPANN) [14] [38]. These methods use genes to encode which TF is used by each neuron. These genes are then subject to mutation and/or crossover during evolution.

The second method by which NE can optimise neuron TFs is to use TFs which are described by a number of parameters [9]. The training methods then optimise these parameters for each individual neuron. A simple version of this technique has been used by CGPANN [19]; where the widths of Gaussian functions were optimised for each neuron. Again the parameter(s) associated with each neuron's TF were encoded in the chromosome by the inclusion of additional gene(s). A more complex version of this method was used in [2] where each neuron's TF was itself an evolved Genetic Program. This method allowed for an almost limitless variation of TFs. Another example where each neuron is described by a number of genes, is state-enhanced neural networks [25], where the dynamics of each neuron are evolved. These state-enhanced neural network exhibit memory which can be utilised on partially observable Markov decision tasks.

Until now however, there has been little research which empirically and rigorously investigates if the ability for NE to evolve heterogeneous ANNs actually provides any benefit. This is important research as if it is shown to be beneficial it could easily be adopted by other NE methods; as the described methods just require additional genes for each neuron. As discussed there are two ways in which NE can evolve TFs: 1) by choosing the TF of each neuron from a predetermined list or 2) by optimising parameters associated with each individual neuron. Additionally these two methods can be combined by allowing evolution to both select the TF for each neuron and optimises the parameters associated with the TF. Here both of these methods are investigated along with a combination of the two. The investigation uses two NE strategies and compares the results to evolving regular homogeneous ANNs.

3 Investigation

The investigation presented on evolving heterogeneous ANN using NE takes four parts. The first investigation is to identify if the choice of TF impacts on the effectiveness of NE for evolving homogeneous ANNs. The second investigates if evolving heterogeneous ANNs, by allowing evolution to select each neuron's TF

from a predetermined list, outperforms evolving homogeneous ANNs. The third investigates if using NE to optimise parameters associated with each neuron's TF outperforms evolving homogeneous ANNs. The fourth investigates using NE to both select each neuron's TF from a predetermined list and optimise parameters associated with that TF.

The remainder of this section introduces the NE methods employed by the investigation, the TFs made available and the benchmarks used.

3.1 NeuroEvolutionary Strategies

In order to undertake the described experiments, two NE methods were used; this is to ensure that any conclusions are not specific to a particular type of NE. The chosen NE methods are Conventional NeuroEvolution (CNE) and Cartesian Genetic Programming of Artificial Neural Networks (CGPANN). CNE is the simplest (and oldest) form of NE and evolves the connection weights of fixed topology ANNs. CGPANN is a more complex NE method which evolves both the connection weights and topology of ANNs. These two NE methods represent the two main types of NE; those which evolve only connection weights and those which evolve connection weights and topology[2].

3.1.1 Conventional NeuroEvolution

CNE [30] operates by storing the connection weights of a fixed topology ANNs as an array of floating point numbers. Each of these arrays represents a chromosome. Mutation is implemented by selecting a new random weight value for each gene (weight) with a given probability. CNE is extended here to be capable of evolving each neuron's TF by the inclusion of an additional gene per neuron. These additional TF genes can either be used as an index in a look-up-table of TFs, or as a parameter value to be used by each neuron's TF. As CNE uses fixed topologies, this topology must be selected in advanced by the user.

3.1.2 Cartesian Genetic Programming of Artificial Neural Networks

CGPANN [14,38] is the application of Cartesian Genetic Programming (CGP) to the evolution of ANNs. CGP [22,24] is a form of Genetic Programming (GP) which represents computational structures as directed graphs of nodes indexed by their Cartesian coordinates. CGP does not suffer from bloat [21,40]. Bloat refers to the condition where during evolutionary time the size of evolved computational structures grow without limit. Many other forms of GP do suffer from bloat and there has been intensive research to address the issue [33]; interestingly other graph based encoding schemes have also been shown to suffer less from bloat that standard GP [32]. CGP chromosomes also contain non-functioning genes enabling

[2] Sometimes refereed to as TWEANNs - Topology and Weight Evolving Artificial Neural Networks.

neutral genetic drift during evolution [43,49]. CGP typically evolves acyclic networks but can also be easily adapted to evolve cyclic or recurrent networks [41]. CGP typically uses point or probabilistic mutation and no crossover.[3]

Other forms of graph based GP have also been proposed including Parallel Distributed Genetic Programming (PDGP) [27]. PDGP has also been applied to NE [27]. PDGP is similar in structure to CGP, graph based, but uses different mutation and crossover operations. CGP also placed fewer restraints on the structure of the generated graphs [22]. Research surrounding PDGP appears to be abandoned.

Each CGP chromosome is comprised of a number of gene types: function genes (F_i), connection genes ($C_{i,j}$) and output genes (O_i). The function genes represent indexes in a function look-up-table and describe the functionality of each node. The connection genes define from 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 address any program input or internal node and define which nodes 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 a column. However, 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 due to CGP being capable of evolving where each node connects 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 Equation 1; where α is the arity of each node, n is the number of nodes and m is the number of program outputs.

An example CGP program is given in Figure 1 along 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. An advantage of CGP over tree-based GP, again seen in Figure 1, is that node outputs can be reused multiple times, rather than requiring the same value to be recalculated if it is needed again. Finally, not all nodes contribute to the final program output, these represent the inactive nodes which enable neutral genetic drift and make variable length phenotypes possible.

$$F_0C_{0,0}...C_{0,\alpha}F_1C_{1,0}...C_{1,\alpha} \text{ ..... } F_nC_{n,0}...C_{n,\alpha}O_0...O_m \tag{1}$$

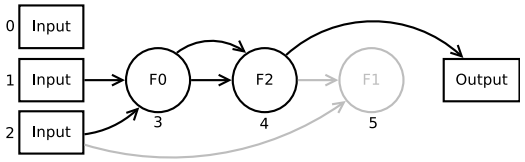

Fig. 1 Example CGP program corresponding to the chromosome: 012 233 124 4

CGP is easily applied to ANNs [14,38] by the inclusion of connection weight genes ($W_{i,j}$) for each node input and by using TFs suited to ANNs. CGPANN

[3] An open source cross platform CGP/CGPANN library has also been recently released for those interested in using CGP or CGPANN [37].

exhibits all of the benefits of CGP and is a NE training method which can evolve the weights, topology [39] and TFs of ANNs. Although CGP evolves topology, it is required that the user specifies a maximum network size. This could be considered a drawback, but overestimating the required number of nodes has been shown to be highly beneficial for CGP [23]. Similarly, a maximum neuron arity must be specified, however, the arity of each neuron can be lower than this maximum [38]. This occurs when the chromosome describes two neurons being connected by two or more connections. In this case, multiple connections between two neurons are equivalent to one connection; with the connection weight value being the sum of the individual weights.

It is important to note that the types of ANN created using CGPANN are unconventional and often cannot be described in terms of layers and nodes per layer. Figure 2 gives an example of the type of ANN which can be created using CGPANN. It can be seen that each neuron's input is highly unconstrained; they can connect to any previous neuron in the network including input neurons. It can also be seen that the arity of each neuron can vary. Additionally any neuron can be used as an output; including the input neurons. Figure 2 demonstrates that by allowing NE to optimise topology, evolution is capable of discovering topologies which would be unlikely to be considered by a human designer.

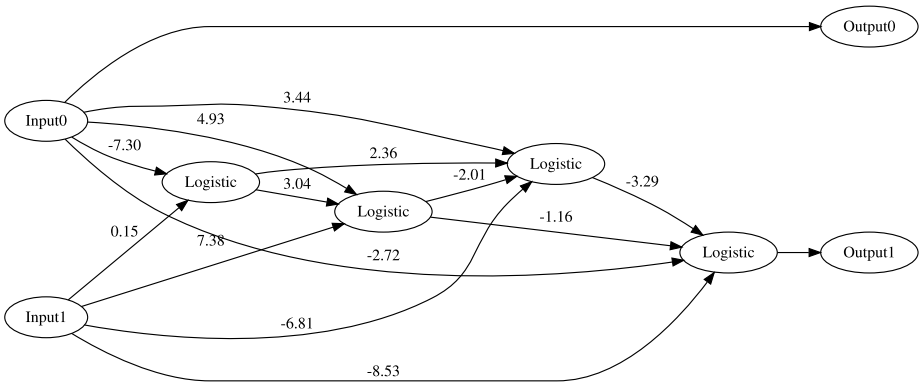

Fig. 2 Depiction of the types of ANN created using CGPANN.

3.2 Transfer Functions

The TFs used in this investigation are the Heaviside step function, Equation 2, the Gaussian function, Equation 3, and the logistic sigmoid function[4], Equation 4. Each of these TFs is shown graphically in Figure 3. These particular TFs were selected as they are the most commonly used by ANNs.

[4] The logistic sigmoid function is often simply referred to as the sigmoid function in the ANN literature. In fact the term sigmoid function refers to any function which is 'S' shaped. The logistic sigmoid function is therefore a specific type of sigmoid function along with other functions including the Gompertz function.

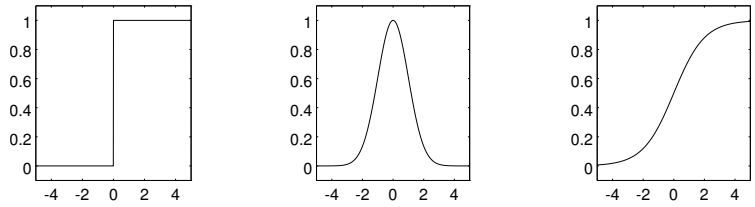

Fig. 3 Form left to right: Heaviside step function, Gaussian function and the logistic function. With $\sigma = 1$ for the Gaussian and logistic functions.

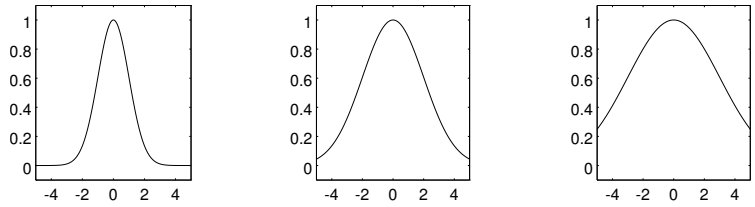

Fig. 4 Variable Gaussian function. From left to right σ = 1,2 and 3.

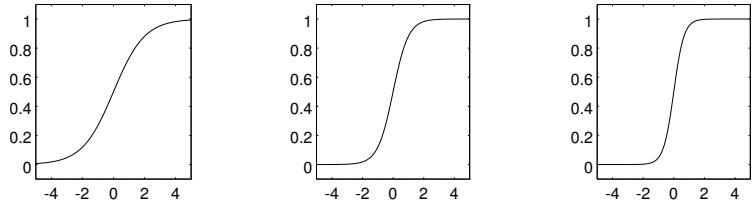

Fig. 5 Variable logistic function. From left to right σ = 1,2 and 3.

As can be seen in Equations 3 and 4, the Gaussian and logistic functions have been given in a form which contains a σ variable. Where $\sigma = 1$ gives the typical form of these TFs. When using NE to evolve parameters associated with each neuron's TF, the σ value can be evolved or optimised. Figures 4 and 5 show the Gaussian and logistic function respectively for a range of σ values.

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$f(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{3}$$

$$f(x) = \frac{1}{1 + \exp(-\sigma x)} \tag{4}$$

3.3 Benchmarks

In order to draw strong conclusions regarding whether it is beneficial to evolve TFs, it is necessary to examine its effectiveness on a wide range of benchmarks. In this paper five benchmarks are employed. The chosen benchmarks mainly include

supervised learning classification tasks, a common application of ANNs, but also include a reinforcement learning control task.

Despite many of the described benchmarks being classification tasks, they each use their own type of fitness function. Although this adds complexity, the fitness functions used here are those typically used with these benchmarks. This is done to make standard the use of these benchmarks; which is important when comparing machine learning methods.

3.3.1 Ball Throwing

The ball throwing benchmark [15] is a reinforcement learning control task. The task is to design a controller for a driven arm so as to throw a ball a distance of ≥ 9.5 m. A depiction of the task is given in Figure 6, with the equations describing the dynamics of the arm given in Equations 5 and 6; symbol definitions given in Table 1. The model is simulated using Euler integration with a time step of 0.01 s for 3000 time steps. The control system has two inputs θ and ω and outputs two values T and whether or not to release the ball. The inputs to the controller are linearly scaled from $\pm\pi/2$ and ± 5 rad/s to a [0,1] range for θ and ω respectively. The first output of the controller sets the torque applied to the arm and is linearly mapped to a $[-5, 5]$ N range. The ball is released if the second output exceeds a threshold of 0.5. Once the ball is released, Newtonian mechanics are used to calculate the distance the ball is thrown (d) which is then used as the fitness value.

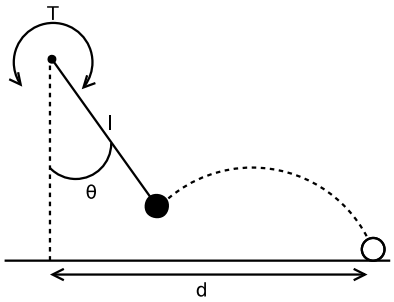

Fig. 6 Depiction of the ball throwing benchmark.

$$\left(\dot{\theta}, \dot{\omega}\right) = \left(\omega, -c \cdot w + \frac{g \cdot \sin(\theta)}{l} + \frac{T}{m \cdot l^2}\right) \tag{5}$$

$$\omega = 0 \text{ if } |\theta| \geq \pi/2 \tag{6}$$

3.3.2 Full Adder

The full adder benchmark is the task of implementing a full adder circuit using an ANN. The ANN has three inputs (two input bits and a carry bit) and two outputs (one for the sum bit and the other for the carry out). Each output is decoded as

Table 1 Ball throwing symbol definitions with commonly used values.

Symbol	Description	Value
θ	The arm angle	$[-\frac{\pi}{2}, \frac{\pi}{2}]rad$
ω	The arms angular Velocity	
c	Friction constant	$2.5s^{-1}$
l	Arm length	$2m$
g	Gravity	$9.81ms^{-2}$
m	Ball mass	$0.1kg$
T	Torque applied to arm	$[-5, 5]Nm$

a '1' if ≥ 0.5, otherwise it is decoded as a '0'. The fitness value assigned to each chromosome is the number of correct output bits generated after every possible input pattern has been applied; see Table 2. This results in a maximum fitness of sixteen.

Table 2 Full Adder truth table.

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3.3.3 Monks Problem 1

The Monks Problems [36] are a set of three classification benchmarks intended for comparing learning algorithms. The classification tasks are based on the appearance of robots which are described by six attributes, each with a range of values; see Table 3. Only the first classification task is used here, where a robot belongs to a class if head_shape = body_shape OR jacket_color = red. This task is one of the recommended non-trivial benchmarks given in [10]. The task uses 124 of the possible 432 combinations for the training set and the remainder for the testing set. The implementation commonly used by ANN is to assign each value of each attribute its own input to the network; totaling seventeen inputs. Each of these inputs is set as '1' if the particular attributes value is present and as '0' otherwise. The ANN classifies each sample as belonging to the class if the single ANN output is ≥ 0.5. The target fitness is zero percent classification error.

3.3.4 Two Spirals

The two spirals classification benchmarks was created in the 1980s and was originally posted on a connectionist mailing list by Alexis Wieland [5]. The task is considered highly challenging for ANN [6]. The benchmark consists of 194 data points describing samples taken from two spirals in Cartesian space; see Figure 7.

Table 3 Monks Problem Robot Descriptions.

Description	Attributes
head_shape	round, square, octagon
body_shape	round, square, octagon
is_smiling	yes, no
holding	sword, balloon, flag
jacket_color	red, yellow, green, blue
has_tie	yes, no

The task is to classify to which spiral each sample belongs using only the (x, y) Cartesian coordinates. The target fitness is zero, meaning that there are no incorrect classifications. The ANN comprises of two inputs for the (x, y) Cartesian coordinates of each sample, and one output. The two inputs are linearly scaled into a $[0, 1]$ range by assuming the maximum values are ± 6.5 and ± 6 for the x and y axis respectively. When the output value is < 0.5 it is interpreted as one spiral and ≥ 0.5 as the other.

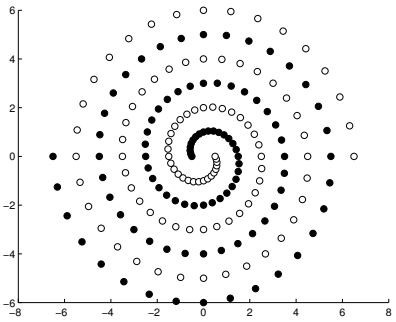

Fig. 7 Depiction of the Two Spiral Classification Benchmark.

3.3.5 Proben1: Cancer1

The Cancer1 dataset[5] is a classification task described in the Proben1 document [29]. The dataset was originally constructed at the University of Wisconsin Hospital [18]. Each sample in the dataset describes nine values, recorded by a surgeon using fine needle aspiration, of a tumour located in the breast of patients. Each sample is labelled with two mutually exclusive flags, benign and malignant, indicating the tumour type. All of the values are scaled into a $[0, 1]$ range. The dataset contains 699 samples, 65.5% of which represent benign tumours. The first 525 samples are used as the training set with the remainder used for the testing set. The fitness assigned to each chromosome is the squared error percentage, Equation 7. Where o_{min} and o_{max} are the minimum and maximum output values form the ANN, N is the number of outputs from the ANN, P is the number of training examples, o_{pi} are the actual output values from the ANN and t_{pi} are the target

[5] The '1' in 'Cancer1' refers to the permutation of the dataset; see [29].

outputs. Therefore the optimum corresponds to a squared percentage error equal to zero.

$$E = 100 \cdot \frac{o_{max} - o_{min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2 \tag{7}$$

4 Results

Four experiments are presented here which investigate the influence of TFs when using NE to train ANNs. All of the results presented are the average fitness of the best solutions found from fifty repeated runs. Each run was terminated after 100,000 generations, all used a $(1 + 4)$-ES[6], 3% probabilistic mutation[7] and connection weights in the range ± 5. When using CNE, three hidden layers were used, each containing ten neurons; plus one input layer and one output layer. The arity of each neuron was such that the ANN was fully connected between layers. When using CGPANN the maximum number of nodes was set as thirty each with a maximum arity of ten.

Where appropriate, the results are compared using the non-parametric two sided Mann-Whitney U-test and effect size [42] statistics. A U-test value of < 0.05 indicates that the difference between two datasets is statistically significant. The effect size value shows the important of this difference considering the spread of the data; with values > 0.56 showing small importance, > 0.64 medium importance and > 0.71 large importance. Therefore if a comparison between results is shown to be statically significant with a medium or large effect size, then we can be reasonably sure that any difference is not due to under sampling and that the difference is significantly large.

4.1 Experiment 1 - Homogeneous Networks

The first experiment identifies whether, and to what extent, the choice of TF impacts on the effectiveness of training homogeneous ANNs using NE. As previously discussed, the three TFs used for this investigation are the Heaviside step, Gaussian and logistic functions; see Section 3.2.

The average fitness achieved when using each TF is given for the five benchmarks in Tables 4 and 6; when using CNE and CGPANN respectively. The average fitness value is given in bold if it represents the best fitness for that benchmark; indicating the most suitable TF for that benchmark. When appropriate, the fitness is given for the training and testing sets. Where the testing fitness is the average fitness achieved by each of the fifty runs on the testing set after training on the training set is complete. The statistical significance between the fitnesses achieved using each TF are given in Tables 8 and 10; when using CNE and CGPANN respectively. When the difference is statistically significant, $p \leq 0.05$, the value is

[6] $(1 + 4)$-ES has been used previously for CGPANN and is also used here for CNE for simplicity. Additionally as a greedy strategy is used with no crossover maintaining genetic divinity with large populations becomes less significant.

[7] Where probabilistic mutation changes each gene to a new valid value with a given probability.

given in bold. The effect size of the differences between the fitnesses are also given in Tables 12 and 14; when using CNE and CGPANN respectively. When the effect size is of medium or greater importance the value is given in bold.

In all cases a perfect solutions was found for the Full Adder benchmark and similarly in many cases for the Ball Throwing. When perfect solutions are found no comparisons can be made in terms of the fitnesses achieved. For this reason the number of required generations to find perfect solutions are also presented in for these cases. Tables 5 and 7 give the average number of generations required by CNE and CGPANN for the cases where perfect solutions were found. These generation results are analysed using the same significant methods as before in Tables 9 and 13 for CNE and Tables 11 and 15 for CGPANN.

In the case of CGPANN, all TFs found a solution to the Ball Throwing benchmark; to throw the ball a distance of ≥ 9.5 m. Interestingly some TFs managed, on average, to throw the ball much further than the 9.5 m target. This could be framed as greater generalisation. However this aspect of the ball throwing results are not analysed further in this paper.

From the results given in Tables 4 and 6 it can be seen, for both CNE and CGPANN, that the choice of TF has a large impact on the effectiveness of NE. Additionally, in the majority of cases these differences are shown to be statistically significant and with a medium or large effect size. This confirms that the choice of TF has a large impact on the effectiveness of evolving homogeneous ANN using NE.

A further interesting result, also seen in Tables 4 and 6, is that despite being the least commonly used of the three TFs, the Heaviside step function produced the best results in many cases. It can also be seen that the best TF was often also dependent on the NE method used.

Table 4 Average fitness achieved using homogeneous ANNs of different TFs trained using CNE.

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	**9.71**	9.30	5.89	8.30
Full Adder	**16.00**	**16.00**	**16.00**	16.00
Monks Problem 1 Train	**0.065**	14.016	0.258	4.80
Monks Problem 1 Test	18.991	39.116	**14.981**	24.363
Two Spirals	54.64	**34.04**	74.58	54.42
Proben1: Cancer Train	5.185	2.389	**1.798**	3.124
Proben1: Cancer Test	12.816	6.736	**3.034**	7.529

Table 5 Average number of generations required using homogeneous ANN of different TFs trained using CNE.

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	20365.44	-	-	-
Full Adder	**132.94**	317.04	476.54	308.84

Table 6 Average fitness achieved using homogeneous ANNs of different TFs trained using CGPANN.

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	**9.72**	**9.58**	**9.65**	9.65
Full Adder	**16.00**	**16.00**	**16.00**	16.00
Monks Problem 1 Train	**0.210**	15.161	0.952	5.44
Monks Problem 1 Test	4.398	19.532	**4.352**	9.43
Two Spirals	**39.56**	49.28	71.28	53.37
Proben1: Cancer Train	**0.457**	1.364	1.429	1.083
Proben1: Cancer Test	3.678	2.989	**2.218**	2.962

Table 7 Average number of generations required using homogeneous ANN of different TFs trained using CGPANN.

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	**487.76**	9850.46	20401.14	10246.45
Full Adder	**386.60**	729.20	1092.50	736.10

Table 8 Statistical significance between the homogeneous CNE fitness results given in Table 4.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**2.65E-1**	**8.19E-19**	**1.13E-14**
Full Adder	-	-	-
Monks Problem 1 Train	**1.16E-19**	1.81E-1	**3.30E-19**
Monks Problem 1 Test	**6.97E-18**	**1.49E-04**	**6.96E-18**
Two Spirals	**4.42E-18**	**4.30E-18**	**6.55E-18**
Proben1: Cancer Train	**1.15E-14**	**5.59E-18**	**6.76E-3**
Proben1: Cancer Test	**4.90E-14**	**6.29E-18**	**5.98E-14**

Table 9 Statistical significance between the homogeneous CNE generation results given in Table 5.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Full Adder	**1.19E-07**	**1.26E-10**	**4.63E-2**

Table 10 Statistical significance between the homogeneous CGPANN fitness results given in Table 6.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**7.25E-5**	7.36E-2	**8.20E-3**
Full Adder	-	-	-
Monks Problem 1 Train	**1.42E-13**	9.90E-2	**1.35E-11**
Monks Problem 1 Test	**1.96E-9**	9.17E-1	**2.94E-9**
Two Spirals	**6.36E-10**	**6.61E-18**	**4.39E-17**
Proben1: Cancer Train	**1.36E-16**	**1.11E-17**	7.16E-2
Proben1: Cancer Test	**9.10E-3**	**2.48E-9**	**1.78E-3**

Table 11 Statistical significance between the homogeneous CGPANN generation results given in Table 7.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**5.69E-15**	**4.06E-14**	**2.04E-3**
Full Adder	8.25E-1	**3.03E-4**	**8.89E-3**

Table 12 Effect Size between the homogeneous CNE fitness results given in Table 4.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**0.90600**	**0.98840**	**0.92600**
Full Adder	0.50000	0.50000	0.50000
Monks Problem 1 Train	**1.00000**	0.54400	**1.00000**
Monks Problem 1 Test	**1.00000**	**0.72020**	**1.00000**
Two Spirals	**1.00000**	**1.00000**	**1.00000**
Proben1: Cancer Train	**0.94760**	**1.00000**	**0.65560**
Proben1: Cancer Test	**0.93680**	**0.99980**	**0.93420**

Table 13 Effect Size between the homogeneous CNE generation results given in Table 5.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Full Adder	**0.80740**	**0.87340**	**0.61580**

Table 14 Effect Size between the homogeneous CGPANN fitness results given in Table 6.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**0.73040**	**0.60400**	**0.65360**
Full Adder	0.50000	0.50000	0.50000
Monks Problem 1 Train	**0.87360**	0.53960	**0.85060**
Monks Problem 1 Test	**0.84540**	0.50620	**0.84120**
Two Spirals	**0.85840**	**1**	**0.98740**
Proben1: Cancer Train	**0.97580**	**0.99260**	**0.60320**
Proben1: Cancer Test	**0.65000**	**0.84240**	**0.67920**

Table 15 Effect Size between the homogeneous CGPANN generation results given in Table 7.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**0.95340**	**0.93880**	**0.67920**
Full Adder	0.51300	**0.70980**	**0.65200**

4.2 Experiment 2 - Heterogeneous Networks

The second experiment identifies if allowing NE to evolve heterogeneous ANNs, by selecting each neuron's TF from a predetermined list, produces better results than evolving homogeneous ANNs. Evolving the TF used by each neuron is considered beneficial if the result is better than the average of using each TF individually. This measure is chosen because when approaching a new task it not generally known which TF would be most suited, therefore a TF would have to be selected arbitrarily. When evolving heterogeneous ANNs the need to make this choice is removed, and hence it should be considered beneficial if it beats the average random choice of TF. The average fitness for evolving homogeneous ANNs using each TF individually for the five benchmarks are given in Tables 4 and 6 for CNE and CGPANN respectively. In the cases where a perfect solution is always found the required number of generations is also given and used for the comparison.

The results achieved when evolving heterogeneous ANN are given in Tables 16 and 17 for CNE and CGPANN respectively. The results are given in bold if the fitness is better, or equal, to the average of using each TF individually; or the average number of generations required to find a perfect solution is equal or lower. The percentage of neurons which use each TF is also given in Tables 16 and 17; this is only for the active nodes in the CGPANN case. No statistical analysis can be undertaken for this experiment as the comparison is against the average result of using each TF individually.

As can be seen in Tables 16 and 17, in the majority of cases evolving heterogeneous ANNs outperformed the average result of evolving homogeneous ANNs. This indicates that evolving heterogeneous ANNs is typically a better strategy than evolving homogeneous ANNs. This holds unless the user knows in advance which TF is most suited to a given task; in which case that TF should be used. Interestingly in the case of CNE applied to the ball throwing benchmark, evolving heterogeneous ANNs produce an equal fitness to that of using the best TF alone. Additionally for CNE applied to the Proben1:Cancer1 benchmark, heterogeneous ANNs produce a better fitness on the test set than using the best TF alone. This indicates that it may be the ability to mix TFs, as well as choose which are used, which is offering an evolutionary advantage.

Table 16 Average fitness achieved using heterogeneous ANNs trained using CNE. The percentage of neurons which used each TF is also given.

Benchmark	Train	Test	Generations	Step	Gaussian	Logistic
Ball Throwing	**9.71**		2931.04	34.3%	34.1%	31.6%
Full Adder	**16.00**		**201.90**	32.3%	36.0%	31.7%
Monks Problem 1	**3.597**	27.30	-	33.2%	34.5%	32.4%
Two Spirals	**38.52**		-	37.2%	32.7%	30.1%
Proben1: Cancer	**1.505**	**4.379**	-	33.6%	30.1%	36.3%

Table 17 Average fitness achieved using heterogeneous ANNs trained using CGPANN. The percentage of neurons which used each TF is also given.

Benchmark	Train	Test	Generations	Step	Gaussian	Logistic
Ball Throwing	**9.68**	-	**1000.76**	31.4%	34.0%	34.5%
Full Adder	**16.00**	-	**698.10**	32.5%	35.2%	32.3%
Monks Problem 1	**1.226**	**6.139**	-	38.4%	27.3%	34.3%
Two Spirals	**50.20**	-	-	34.0%	34.3%	31.8%
Proben1: Cancer	1.086	3.126	-	34.9%	33.5%	31.6%

4.3 Experiment 3 - Evolving Transfer Function Parameters

The third experiment is to identify if optimising parameters associated with each neuron is beneficial for NE. As previously discussed, the parameters to be optimised vary the shape of the Gaussian and logistic functions; see Section 3.2. In each case the ANNs are comprised of the same TF, Gaussian or logistic, but a parameter controlling the shape of each neuron's TF is evolved. Here the parameter values for each TF are limited to the set {1, 2, 3}, see Equations 3 and 4; but this is not a requirement of the method.

Evolving parameters associated with each neuron's TF will be considered beneficial if it produces stronger results than the use of the non-parametrised counterpart e.g. if variable Gaussian produces stronger results than the standard Gaussian TF. This comparison is used as it isolates the aspect we are interested in and tests whether the ability to vary the shape of each neuron's TF provides any benefit.

The average fitnesses achieved using variable Gaussian and variable logistic functions on the five benchmarks are given in Tables 18 and 20 respectively when using CNE. In cases where the target fitness is always reached the average number of generations used are given in Tables 19 and 21. Similarly, Tables 22 and 24 give the results when using CGPANN. Again in cases where the target fitness is always reached the average number of generations used are given in Tables 23 and 25. In all cases the results are compared against those obtained for the non-variable form of the TFs. In the given tables, a bold fitness/number of generations indicates that the variable TF performed better than the non-variable form. Additionally, a bold value for the the U-test indicates statistical significance and a bold value for the effect size indicates a medium or large effect size. For instance, if the fitness, U-test and effect size values are all given in bold then the variable TF is shown to strongly outperform the non-variable counterpart with statistical significance. If however the value is not bold, but the U-test and effect size values are bold, then this shows that the non-variable TF strongly outperformed the variable counterpart with statistical significance. If the U-test is bold but the effect size is not bold, then then this indicates a weak difference with statistical significance. If the U-test value is not bold then the difference between non-variable and variable TFs is not statistically significant and is considered insignificant.

It can be seen in Tables 18-25, in the majority of cases, the variable version of the TF outperformed the non-variable form. Additionally, many instances where the variable form is superior are also shown to be statistically significance with a medium to large effect size. Only four of the twenty cases show the non-variable form outperforming the variable form with statistical significance and medium to large effect size. Eleven of the twenty cases showed the variable form outperforms the non-variable form with statistical significance with a medium to large effect

size. Of the remaining five cases, one showed variable TFs offered a weak disadvantage and the remainder showed no significant difference[8]. Therefore using variable TFs is shown to be often beneficial and rarely worse.

The differences can also be given in terms of the NE method used. In seven of the ten cases which used CNE, the variable TFs offered an advantage compared with three cases where it was a disadvantage. For CGPANN, five out of the ten cases showed variable TFs offered an advantage compared to one of the ten cases showing a disadvantage. The results can also be given in terms of the TF employed. In six of the ten cases the variable logistic TF offered and advantage and in two cases a disadvantage. In five of the ten cases the variable Gaussian TF offered an advantage and in three cases a disadvantage.

Table 18 Average fitness of ANNs using the variable Gaussian TF trained using CNE.

Benchmark	Gaussian_Var	U-test	Effect Size	Over all
Ball Throwing	**9.58**	**1.20E-04**	**0.72340**	Adv (Strong)
Full Adder	16.00	-	-	-
Monks Problem 1 Train	**13.500**	0.54481	0.53520	-
Monks Problem 1 Test	**37.634**	**3.25E-2**	**0.62420**	Adv (Strong)
Two Spirals	49.36	**5.42E-15**	**0.95340**	DisAdv (Strong)
Proben1: Cancer Train	**1.512**	**2.41E-07**	**0.79700**	Adv (Strong)
Proben1: Cancer Test	**3.598**	**1.10E-10**	**0.87320**	Adv (Strong)

Table 19 Average number of generations required using variable Gaussian TF trained using CNE.

Benchmark	Gaussian_Var	U-test	Effect Size	Over all
Full Adder	**221.08**	**6.54E-3**	**0.65800**	Adv (Strong)

Table 20 Average fitness of ANNs using the variable logistic TF trained using CNE.

Benchmark	Logistic_Var	U-test	Effect Size	Over all
Ball Throwing	**9.62**	**3.22E-18**	**0.98000**	Adv (Strong)
Full Adder	16.00	-	-	-
Monks Problem 1 Train	0.710	**1.67E-2**	**0.60720**	DisAdv (Strong)
Monks Problem 1 Test	19.218	**8.04E-04**	**0.69460**	DisAdv (Strong)
Two Spirals	**58.70**	**6.51E-18**	**1.00000**	Adv (Strong)
Proben1: Cancer Train	**1.684**	0.124	**0.58800**	-
Proben1: Cancer Test	3.598	**1.74E-2**	**0.63660**	DisAdv (Strong)

[8] In the case where training and testing results are given the training result is used for comparisons. This is because no steps to combat over training were made.

Table 21 Average number of generations required using variable logistic TF trained using CNE

Benchmark	Logistic_Var	U-test	Effect Size	Over all
Full Adder	**270.18**	**1.61E-3**	**0.68320**	Adv (Strong)

Table 22 Average fitness of ANNs using the variable Gaussian TF trained using CGPANN.

Benchmark	Gaussian_Var	U-test	Effect Size	Over all
Ball Throwing	**9.60**	3.72E-1	0.55200	-
Full Adder	16.00	-	0.50	-
Monks Problem 1 Train	**6.725**	**2.72E-4**	**0.69980**	Adv (Strong)
Monks Problem 1 Test	**11.903**	**6.41E-4**	**0.69380**	Adv (Strong)
Two Spirals	54.78	8.02E-5	**0.72880**	DisAdv (Strong)
Proben1: Cancer Train	1.375	4.69E-1	0.54160	DisAdv (Weak)
Proben1: Cancer Test	**2.690**	2.71E-1	**0.56340**	-

Table 23 Average number of generations required using variable Gaussian TF trained using CGPANN.

Benchmark	Gaussian_Var	U-test	Effect Size	Over all
Ball Throwing	**4730.86**	1.12E-1	**0.59240**	-
Full Adder	**507.98**	7.75E-1	0.51680	-

Table 24 Average fitness of ANNs using the variable logistic TF trained using CGPANN.

Benchmark	Logistic_Var	U-test	Effect Size	Over all
Ball Throwing	**9.66**	7.17E-1	0.52120	-
Full Adder	16.00	-	-	-
Monks Problem 1 Train	**0.290**	2.25E-1	0.53000	-
Monks Problem 1 Test	**4.287**	9.42E-1	0.50440	-
Two Spirals	**63.00**	**2.81E-9**	**0.84460**	Adv (Strong)
Proben1: Cancer Train	**1.082**	**5.03E-7**	**0.78760**	Adv (Strong)
Proben1: Cancer Test	2.966	**7.39E-4**	**0.69220**	DisAdv (Strong)

Table 25 Average number of generations required using variable logistic TF trained using CGPANN

Benchmark	Logistic_Var	U-test	Effect Size	Over all
Ball Throwing	**1959.20**	**2.53E-9**	**0.84600**	Adv (Strong)
Full Adder	**630.16**	9.32E-2	**0.59760**	-

4.4 Experiment 4 - Evolving Heterogeneous Networks and Transfer Function Parameters

The fourth experiment investigates allowing NE to evolve heterogeneous ANNs whilst also optimising parameters associated with each neuron's TF. The function set contains the step, Gaussian and logistic functions where the Gaussian and logistic functions can have their σ value in the set {1,2,3}; the step function is not parametrised.

Evolving heterogeneous ANNs which also optimise TF parameters will be considered beneficial if it produces stronger results than the use of the non-parametrised heterogeneous counterpart; given in Section 4.2. This comparison is made to see if including variable TFs can improve again upon the use of heterogeneous ANN.

The results of evolving heterogeneous ANN with variable TF parameters are given in Tables 26 and 28 when using CNE and CGPANN respectively. The tables give the average fitness for applying heterogeneous ANN with variable TFs to each of the five benchmarks. The average fitness is given in bold if it represents a better average fitness than that found when evolving non-parametrised heterogeneous ANNs. The U-test and effect size are also in the same format as in Section 4.3. As previously, if perfect solutions are always found the number of generations required to find perfect solutions is also give so as to allow comparison. These results are given in Tables 27 and 29 for CNE and CGPANN respectively.

As can be seen in Tables 26-29, in the majority of cases the addition of varying TF parameters to evolving heterogeneous ANNs does not improve the performance. When using CNE it produced statistically significant superior results for the Ball Throwing benchmark and produced statistically significant worse results for the Two Spirals benchmark. When using CGPANN it produced no statistically significant superior results and produced statistically significantly worse results for the Two Spirals benchmark. In all other cases there was no statistical significance between the two techniques. It can therefore be concluded that the addition of variable TFs to evolving heterogeneous ANNs does not improve the performance over the use of non-variable TFs.

Table 26 Average fitness achieved using variable heterogeneous ANNs trained using CNE.

Benchmark	Heterogeneous	U-test	Effect Size	Over all
Ball Throwing	9.67	0.230	**0.56980**	-
Full Adder	16.00	-	-	-
Monks Problem 1 Train	**3.387**	7.00E-1	0.52240	-
Monks Problem 1 Test	**25.065**	5.93E-2	**0.60960**	-
Two Spirals	49.86	**5.58E-12**	**0.89960**	DisAdv (Strong)
Proben1: Cancer Train	**1.383**	1.31E-1	**0.58660**	-
Proben1: Cancer Test	**3.287**	**4.71E-04**	**0.70160**	Adv (Strong)

Table 27 Average number of generations of ANNs using variable heterogeneous TFs trained using CNE.

Benchmark	Heterogeneous	U-test	Effect Size	Over all
Ball Throwing	**2288.98**	**2.57E-2**	**0.62960**	Adv (Strong)
Full Adder	282.22	5.36E-2	**0.61220**	-

Table 28 Average fitness achieved using variable heterogeneous ANNs trained using CGPANN.

Benchmark	Heterogeneous	U-test	Effect Size	Over all
Ball Throwing	9.67	3.78E-1	0.55140	-
Full Adder	16.00	-	-	-
Monks Problem 1 Train	**0.8226**	7.82E-1	0.50960	-
Monks Problem 1 Test	6.907	2.44E-1	0.56780	-
Two Spirals	58.50	**4.52E-9**	**0.84000**	DisAdv (Strong)
Proben1: Cancer Train	1.086	4.42E-1	0.54420	-
Proben1: Cancer Test	3.126	9.75E-1	0.50200	-

Table 29 verage number of generations of ANNs using variable heterogeneous TFs trained using CGPANN.

Benchmark	Heterogeneous	U-test	Effect Size	Over all
Ball Throwing	1025.80	5.06E-1	0.53880	-
Full Adder	**480.40**	2.52E-1	**0.56660**	-

4.5 Box and Whisker Plots

For high-level inspection, all of the previously given results are presented as box and whisker plots. The boxes represent the lower and upper quartile ranges and the whiskers are set as 1.5 the interquartile range. Points greater than 1.5 of the interquartile range are labelled with a red '+' and points greater than 3 times the interquartile range labelled with a red '$\circ$'. The median of each plot is given as a solid red line and the arithmetic mean is given as a dashed black line.

The average homogeneous fitness, given in Section 4.1, is also given as a dashed green line spanning the three homogeneous functions; step, Gaussian and logistic. The average homogeneous fitness is calculated as the arithmetic mean of the arithmetic means of the fitnesses achieved for the step, Gaussian and logistic function. As a perfect fitness was often achieved for the Ball Throwing and the Full Adder benchmarks, the average number of generations to find the perfect solution are also given as box plots.

The CNE results are given in Tables 9-16 and the CGPANN results are given in Tables 18-25.

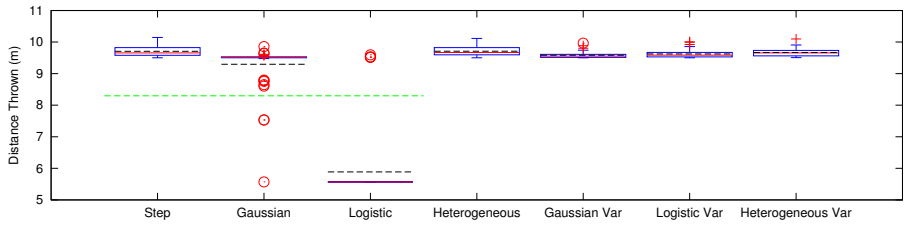

Fig. 8 Fitnesses achieved from applying CNE to the Ball Throwing Benchmark.

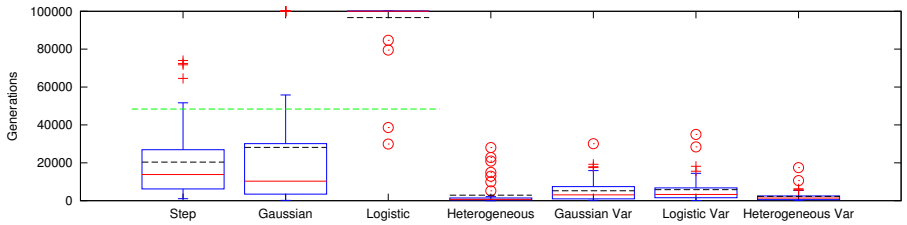

Fig. 9 Generations required from applying CNE to the Ball Throwing Benchmark.

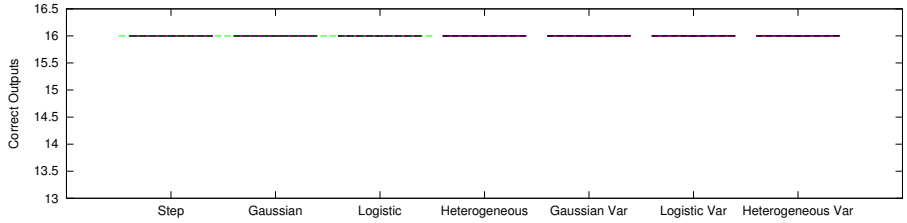

Fig. 10 Fitnesses achieved from applying CNE to the Full Adder Benchmark.

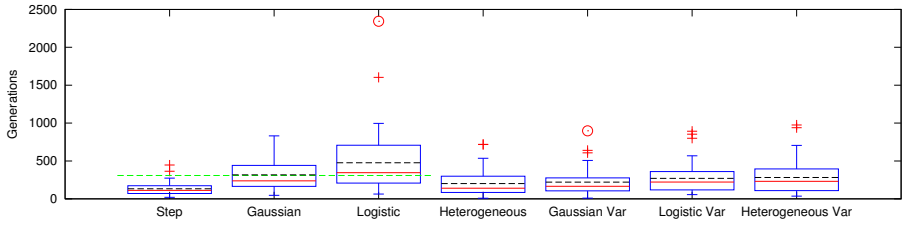

Fig. 11 Generations required from applying CNE to the Full Adder Benchmark.

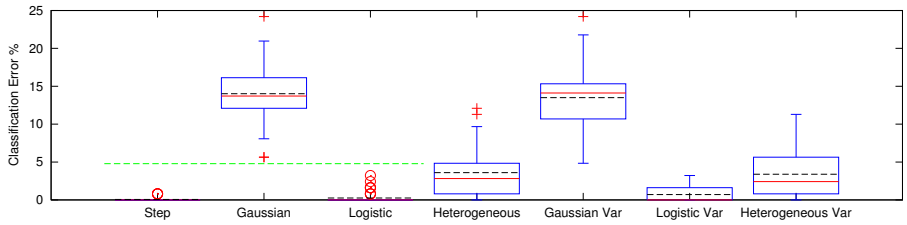

Fig. 12 Fitnesses achieved from applying CNE to the Monks Problem 1 Benchmark - Training.

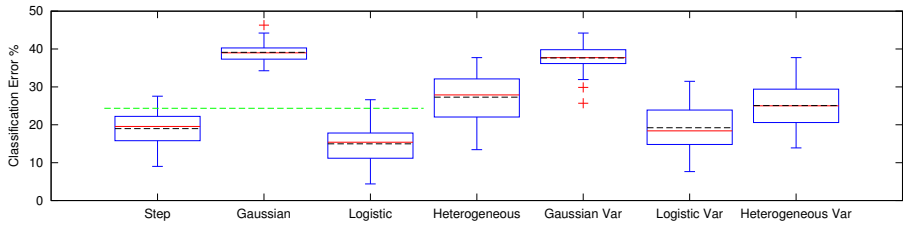

Fig. 13 Fitnesses achieved from applying CNE to the Monks Problem 1 Benchmark - Testing.

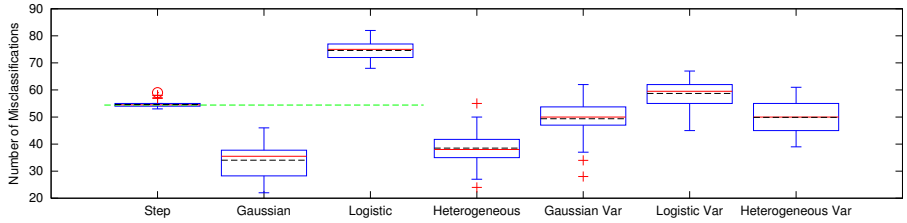

Fig. 14 Fitnesses achieved from applying CNE to the Two Spirals Benchmark.

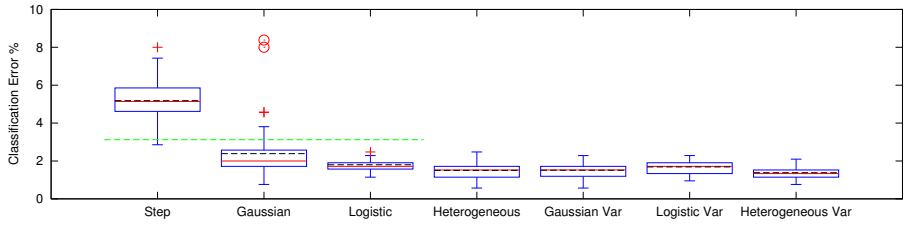

Fig. 15 Fitnesses achieved from applying CNE to the Proben Cancer1 Benchmark - Training.

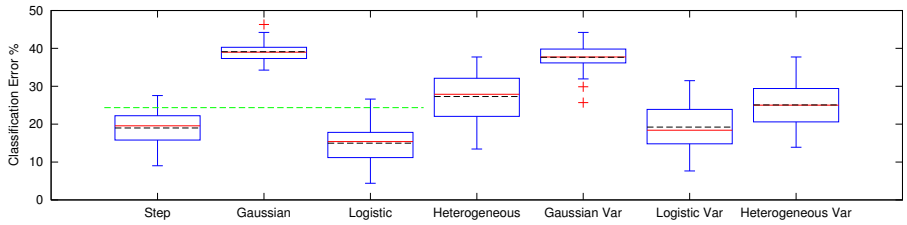

Fig. 16 Fitnesses achieved from applying CNE to the Proben Cancer1 Benchmark - Testing.

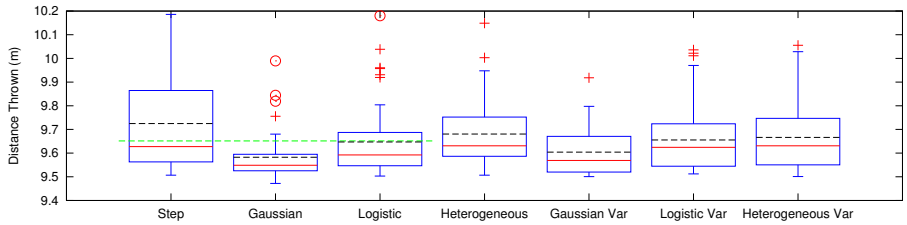

Fig. 17 Fitnesses achieved from applying CGPANN to the Ball Throwing Benchmark.

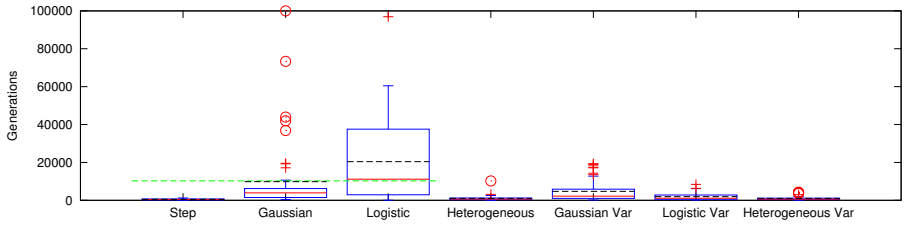

Fig. 18 Generations required from applying CGPANN to the Ball Throwing Benchmark.

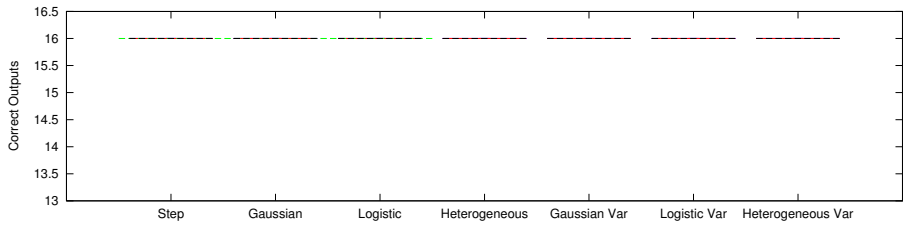

Fig. 19 Fitnesses achieved from applying CGPANN to the Full Adder Benchmark.

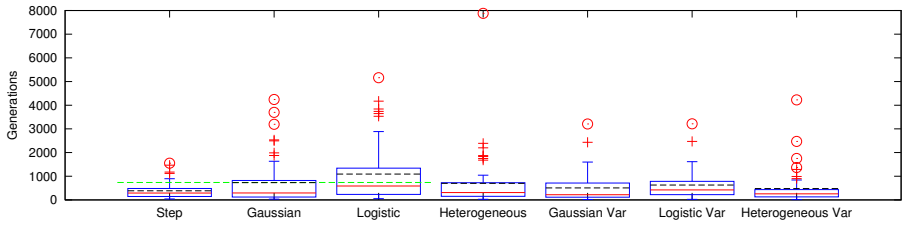

Fig. 20 Generations required from applying CGPANN to the Full Adder Benchmark.

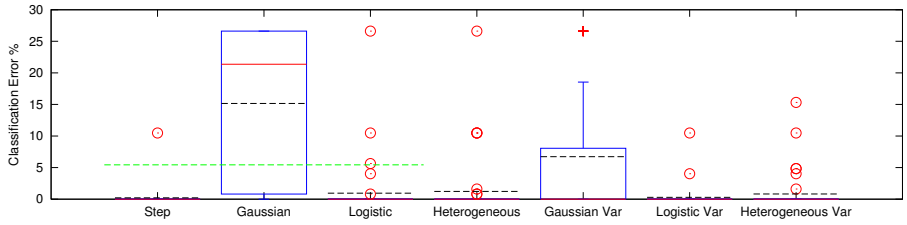

Fig. 21 Fitnesses achieved from applying CGPANN to the Monks Problem 1 Benchmark - Training.

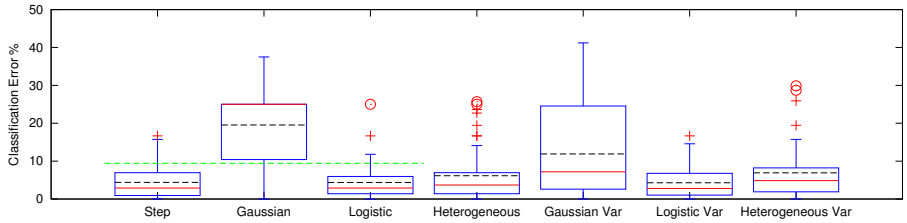

Fig. 22 Fitnesses achieved from applying CGPANN to the Monks Problem 1 Benchmark - Testing.

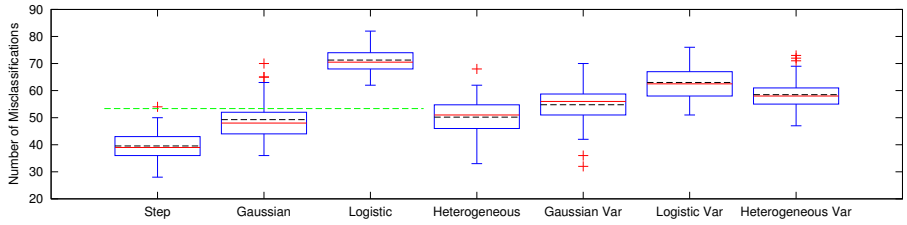

Fig. 23 Fitnesses achieved from applying CGPANN to the Two Spirals Benchmark.

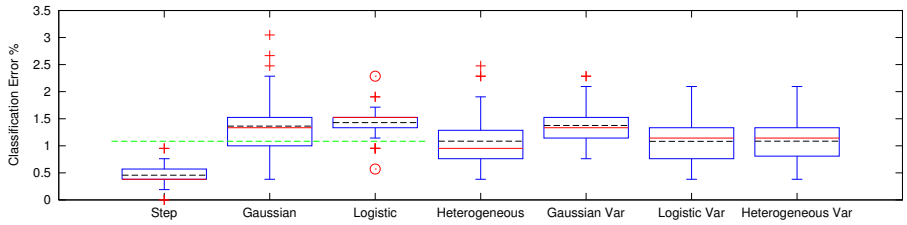

Fig. 24 Fitnesses achieved from applying CGPANN to the Proben Cancer1 Benchmark - Training.

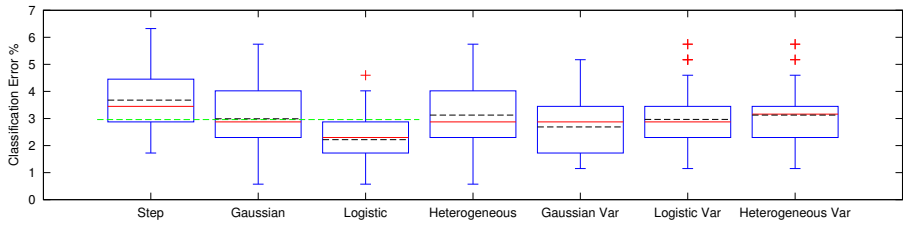

Fig. 25 Fitnesses achieved from applying CGPANN to the Proben Cancer1 Benchmark - Training.

5 Discussion

This paper has empirically demonstrated that when using NE to create homogeneous ANNs the choice of TF has a large impact on the fitness of the solutions found. It was also shown that no single TF produced the best results in all cases. Therefore when using homogeneous ANNs one must accept possibly inferior results or repeat training with a range of TFs. This paper has also empirically demonstrated that, on average, evolving heterogeneous ANNs produces better results than the average of using each TF individually. Additionally it has been shown that optimising parameters associated with each neuron's TF produces better results, on average, than using the typical fixed TFs. Interestingly however, a combination of evolving heterogeneous ANNs where each neuron's TF can also be adapted was shown not to produce better results than simply evolving heterogeneous ANNs of static TFs.

The results presented in Section 4.1 demonstrated that the choice of TF has a large impact on the effectiveness of NE. This is an intuitive result as it is likely that particular TFs are more or less suited suited to given tasks; it appears to mirror the 'No Free Lunch' theorem [46] but concerning TFs. However, although intuitive, it is a significant result as a user is unlikely to know, in advance of training, which TF is most suited to a given task.

A further interesting and unexpected result from the first experiment is that, in many cases, the Heaviside step function was found to be the most effective TF; particularly for CGPANN. The step function was the original TF used by the McCulloch and Pitts neuron models [20]. The fact that the step function is incompatible with back propagation algorithms and is only suited to tasks with binary

outputs is probably the reason other TFs have been favoured. Here, however, it has been shown that when using NE the Heaviside step function is still a suitable TF for contemporary ANNs provided the task is compatible with binary outputs.

The second experiment demonstrated that allowing NE to evolve heterogeneous ANNs produced better results, on average, than the average result obtained by evolving homogeneous ANNs of each TF. This is significant because, as the first experiment demonstrates, the choice of TF has a large impact on the effectiveness of the final ANNs. This, coupled with the fact there is no way of knowing which TF will be most suited to a given task before training begins, puts homogeneous ANN at a disadvantage. The importance of this result is heightened by the fact that the majority of NE methods are probably capable of evolving heterogeneous ANNs. The evolution of heterogeneous ANNs may even be further improved by the inclusion of additional TFs not considered here; and as NE places no restrictions on the types of TFs used, the range of possible TF is limitless. It may even be the case that certain TFs complement each other while others may not.

In the application of CNE to the Ball Throwing and Proben1:Cancer1 (training) benchmarks, evolving heterogeneous ANN produced equal and better average fitnesses than for any of the homogeneous ANNs investigated. This indicates that evolution may be positively combining TFs to produce better results than could be achieved using each TF individually.

A further result from the second experiment concerns the percentage of neurons which used each type of TF in the evolved heterogeneous ANNs. Interestingly, it was never the case that one type of TF strongly dominated the network. If this had occurred it would have indicated that evolution has found a particular TF to be the most suited toward the given task. There was, however, reasonable variation in the percentages of each type of TF used for CNE applied to the Full Adder and Two Spirals benchmarks and CGPANN applied to the Monks Problem1 benchmark. This shows that in curtain conditions evolution is providing some form of pressure to use a particular type of TF i.e. it is not simply random. The fact that a particular TF was not favoured in many cases may also indicate that evolution is combining the functionality of all the TFs.

The third experiment demonstrated that, in the majority of cases, using NE to optimise parameters associated with each neuron provided superior results compared to using using non-parametrised TFs. This is also an important result as the inclusion of an additional gene (or genes) which alter the characteristics of each neuron's TF is again probably compatible with all NE methods.

It was also seen in the third experiment that the logistic TF benefited from being parametrised significantly more than the Gaussian TF. This was despite the non-parametrised Gaussian TF producing worse results than the logistic TF overall i.e. there was more room for improvement. It appears that the logistic TF strongly benefits from being parametrised. It could be the case however that this result is dependent on the range of values the TF parameters can take; this was not investigated in this paper. For instance the Gaussian TF may work best over a much smaller or much larger range of values.

In Sections 4.2 and 4.3 evolving heterogeneous ANNs and evolving parameters associated with each neuron's TF were individually shown to be beneficial for NE. However, in Section 4.4 it was show that when combined they produced no benefit, on average, than evolving heterogeneous ANNs with fixed TF parameters. It may be the case that using parameterised TFs and heterogeneous ANN pro-

duce a similar benefits, however, using them together increases the search space dimensionality without increasing the density of good solutions. It could also be possible that performance depends subtly on evolutionary parameter settings so that when these methods are combined new parameter settings are required for optimum performance. Further research would be required to determine this.

As mentioned in Section 1, homogeneous ANNs comprised of logistic or Gaussian TFs can be universal approximators. This means that with the correct connection weight values and combination of TFs, standard homogeneous ANNs are capable of implementing everything which heterogeneous can also implement. However, this fact says nothing about how efficiently standard homogeneous ANNs can implement certain configurations. Where here the term efficiently refers to both the number of neurons required and the time needed to configure the ANNs. Therefore the fact that ANNs are universal approximators is not enough to be considered useful. It is also necessary that the ANNs can also be configured towards a given task. To this end it appears that homogeneous ANNs are, on average, more efficiently configurable.

Although this paper has demonstrated using NE to evolve ANNs it only used a limited set of TFs and optimised only a single parameter associated with each neuron over a small range. Further research should therefore investigate additional TFs and allow for more complex TFs described by multiple parameters; such as those described in [2]. Although certain TFs have been shown to be universal approximators this tells us nothing about how "trainable" / "evolvable" they are. For instance other TFs, such as the step function, were demonstrated to produce better results than other universal approximator TFs.

6 Conclusion

The use of NE to optimise the weights and topology of ANNs is well established and offers a number of advantages over traditional training methods; such as back propagation. However, the use of NE to create heterogeneous ANNs has so far been under researched and under utilised. This paper has demonstrated the use of two methods for allowing NE to create heterogeneous ANNs. That is, selecting each neuron's TF from a predetermined list of TFs or by optimising parameters associated with each neurons TF. The paper has also shown that the effectiveness of using NE to train homogeneous ANNs is highly dependent on the selected TF. Using NE to optimise each neuron's TF has been empirically demonstrated to alleviate this issue.

The results presented in this paper are significant as the methods described for creating heterogeneous ANNs are likely to be compatible with all NE methods. That is many NE method could benefit from evolving heterogeneous ANNs.

Acknowledgements I would like to thank the reviewers for their feedback which help make for a much stronger paper.

References

1. Angeline, P., Saunders, G., Pollack, J.: An evolutionary algorithm that constructs recurrent neural networks. Neural Networks, IEEE Transactions on **5**(1), 54–65 (1994)

2. Augusteijn, M.F., Harrington, T.P.: Evolving transfer functions for artificial neural networks. Neural Computing & Applications **13**(1), 38–46 (2004)
3. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. Advances in neural information processing systems **19**, 153 (2007)
4. Cantú-Paz, E., Kamath, C.: An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **35**(5), 915–927 (2005)
5. Chalup, S.K., Wiklendt, L.: Variations of the two-spiral task. Connection Science **19**(2), 183–199 (2007)
6. Chebira, A., Madani, K.: Advances in Soft Computing, vol. 19, chap. A Neural Network Based Approach For Sensors Issued Data Fusion, pp. 155–160. Physica-Verlag HD (2003)
7. Cliff, D., Harvey, I., Husbands, P.: 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)
8. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys **2**(1), 163–212 (1999)
9. Duch, W., Jankowski, N.: Transfer functions: hidden possibilities for better neural networks. In: ESANN, pp. 81–94 (2001)
10. Duch, W., Jankowski, N., Maszczyk, T.: Make it cheap: learning with o (nd) complexity. In: Neural Networks (IJCNN), The 2012 International Joint Conference on, pp. 1–4. IEEE (2012)
11. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence **1**(1), 47–62 (2008)
12. Glorot, X., Bengio, Y.: 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)
13. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5), 359–366 (1989)
14. Khan, M.M., Ahmad, M.A., Khan, M.G., Miller, J.F.: Fast learning neural networks using Cartesian Genetic Programming. Neurocomputing **121**, 274–289 (2013)
15. Koutník, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-10), pp. 619–626 (2010)
16. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring strategies for training deep neural networks. The Journal of Machine Learning Research **10**, 1–40 (2009)
17. Liu, Y., Yao, X.: Evolutionary design of artificial neural networks with different nodes. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, pp. 670–675. IEEE (1996)
18. Mangasarian, O., Setiono, R., Wolberg, W.: Pattern recognition via linear programming: Theory and application to medical diagnosis. Large-scale numerical optimization pp. 22–31 (1990)
19. Manning, T., Walsh, P.: Improving the performance of CGPANN for breast cancer diagnosis using crossover and radial basis functions. In: Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, pp. 165–176. Springer (2013)
20. McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biology **5**(4), 115–133 (1943)
21. Miller, J.F.: What bloat? Cartesian genetic programming on Boolean problems. In: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp. 295–302 (2001)
22. Miller, J.F.: Cartesian Genetic Programming. Springer (2011)
23. Miller, J.F., Smith, S.: Redundancy and computational efficiency in Cartesian genetic programming. Evolutionary Computation, IEEE Transactions on **10**(2), 167–174 (2006)
24. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proceedings of the Third European Conference on Genetic Programming (EuroGP), vol. 1820, pp. 121–132. Springer-Verlag (2000)
25. Montana, D., VanWyk, E., Brinn, M., Montana, J., Milligan, S.: Evolution of internal dynamics for neural network nodes. Evolutionary Intelligence **1**(4), 233–251 (2009)
26. Park, J., Sandberg, I.W.: Universal approximation using radial-basis-function networks. Neural computation **3**(2), 246–257 (1991)
27. Poli, R.: Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. COGNITIVE SCIENCE RESEARCH PAPERS-UNIVERSITY OF BIRMINGHAM CSRP (1996)

28. Poli, R.: Parallel distributed genetic programming. New Ideas in Optimization, Advanced Topics in Computer Science pp. 403–431 (1999)
29. Prechelt, L.: Proben1: A set of neural network benchmark problems and benchmarking rules. Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep **21**, 94 (1994)
30. Richard K, B., John, M., Nico l N, S.: 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)
31. Rumelhart, D.E., Hintont, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
32. Schmidt, M., Lipson, H.: Comparison of tree and graph encodings as function of problem complexity. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1674–1679. ACM (2007)
33. Silva, S., Costa, E.: Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genetic Programming and Evolvable Machines **10**(2), 141–179 (2009)
34. Smolensky, P.: Parallel distributed processing: explorations in the microstructure of cognition, chap. Information processing in dynamical systems: foundations of harmony theory, pp. 194–281. MIT Press (1986)
35. Stanley, K., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
36. Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S., Fisher, D., et al.: The monk's problems a performance comparison of different learning algorithms. Tech. rep., Carnegie Mellon University (1991)
37. Turner, A.J.: Cartesian Genetic Programming Library (2014). URL `http://cgplibrary.co.uk/`
38. Turner, A.J., Miller, J.F.: 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)
39. Turner, A.J., Miller, J.F.: The Importance of Topology Evolution in NeuroEvolution: A Case Study Using Cartesian Genetic Programming of Artificial Neural Networks. In: Research and Development in Intelligent Systems XXX, pp. 213–226. Springer (2013)
40. Turner, A.J., Miller, J.F.: Cartesian Genetic Programming: Why No Bloat? In: Genetic Programming: 17th European Conference, EuroGP-2014, *LNCS*, vol. 8599, pp. 193–204. Springer-Verlag Berlin Heidelberg (2014)
41. Turner, A.J., Miller, J.F.: Recurrent Cartesian Genetic Programming. In: 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014), *LNCS*, vol. 8672, pp. 476–486 (2014)
42. Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)
43. Vassilev, V.K., Miller, J.F.: The Advantages of Landscape Neutrality in Digital Circuit Evolution. In: Proc. International Conference on Evolvable Systems, *LNCS*, vol. 1801, pp. 252–263. Springer Verlag (2000)
44. Weingaertner, D., Tatai, V.K., Gudwin, R.R., Von Zuben, F.J.: Hierarchical evolution of heterogeneous neural networks. In: Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, vol. 2, pp. 1775–1780. IEEE (2002)
45. Wieland, A.: Evolving neural network controllers for unstable systems. In: Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on, vol. 2, pp. 667–673. IEEE (1991)
46. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on **1**(1), 67–82 (1997)
47. Yao, X.: A review of evolutionary artificial neural networks. International journal of intelligent systems **8**(4), 539–567 (1993)
48. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87**(9), 1423–1447 (1999)
49. Yu, T., Miller, J.F.: Neutrality and the evolvability of boolean function landscape. In: Genetic programming, LNCS, pp. 204–217. Springer (2001)