Cartesian Genetic Programming encoded Artificial Neural Networks: A Comparison using Three Benchmarks

Andrew James Turner
University of York
Department of Electronics
Heslington, York
YO10 5DD, UK
at568@york.ac.uk

Julian Francis Miller
University of York
Department of Electronics
Heslington, York
YO10 5DD, UK
julian.miller@york.ac.uk

ABSTRACT

Neuroevolution, the application of evolutionary algorithms to artificial neural networks (ANNs), is well-established in machine learning. Cartesian Genetic Programming (CGP) is a graph-based form of Genetic Programming which can easily represent ANNs. Cartesian Genetic Programming encoded ANNs (CGPANNs) can evolve every aspect of an ANN: weights, topology, arity and node transfer functions. This makes CGPANNs very suited to situations where appropriate configurations are not known in advance. The effectiveness of CGPANNs is compared with a large number of previous methods on three benchmark problems. The results show that CGPANNs perform as well as or better than many other approaches. We also discuss the strength and weaknesses of each of the three benchmarks.

Categories and Subject Descriptors

I.2.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms

Keywords

Genetic programming, Neural networks, Machine learning

1. INTRODUCTION

Neuroevolution (NE) is the application of evolutionary algorithms to the training of artificial neural networks (ANNs). NE has many advantages over traditional gradient based methods. These include: decreased likelihood of becoming trapped in local optima, not requiring gradient information which can be computationally expensive to obtain and is suited to applications without a precise fitness function [47].

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

NE methods which evolve weights and topologies also have the advantage of not requiring the user to assume a suitable topology, something which is often not known in advance.

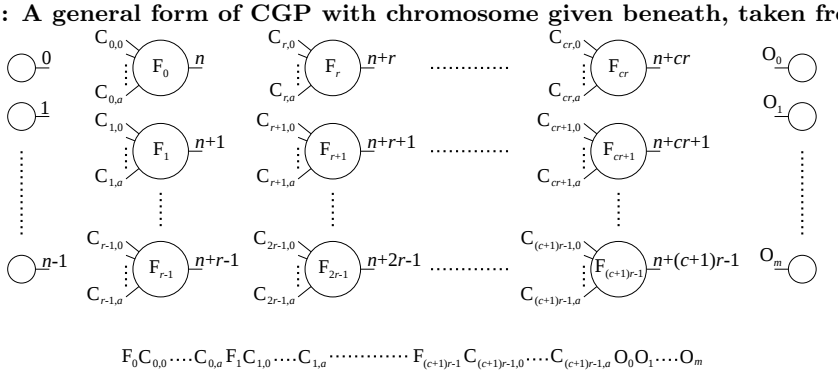
A popular general class of techniques which employ evolutionary techniques is Genetic Programming (GP) [22]. GPs traditionally arrange nodes in a tree structure. This tree structure, however, is not suited to ANNs which are commonly arranged in much less limited graph structures. For example, graph structures allow the reuse of nodes, a key property of ANNs and something which is not possible when directly using tree structures. Cartesian Genetic Programming (CGP) [28], is a form of GP whereby the nodes are arranged in a graph structure, making it much more suited to ANNs. CGP also has many advantages over traditional GP, including redundancy in the chromosomes [29] and natural resilience to bloat [27]. Another previous study of applying a graph-based GP to ANNs was undertaken using PDGP [38]. Although PDGP showed promising results, further published research on its application to ANNs is not found in the literature.

This paper reports the application of CGP to ANNs (CGPANN) on a number of benchmark problems and presents the results in comparison to other NE and machine learning (ML) techniques. The benchmarks chosen cover two large areas of ML, control and classification.

An interesting aspect of CGPANN is that it allows multiple connections between pairs of nodes. This artifact could also be present in other NE methods, such as NEAT [41] or SANE [31], but is not discussed in the literature. This behaviour enables CGPANN to evolve each node's arity. This combined with the ability to evolve weights, topology, number of nodes and the nodes transfer functions means CGPANN has complete control over the evolved ANNs. Other possible consequences of allowing multiple connections between two nodes are also discussed in this paper.

An analysis of the utility of the benchmarks themselves for comparing ML techniques is also discussed. There is nothing wrong with the benchmarks, but often their usefulness is undermined because many published methods use slightly different implementations. This raises the questions about whether or not differences in implementation are significant in the comparisons between different algorithms. For instance, when Stanley et al. published their results of applying NEAT to the double pole experiment [41], they used a slightly modified sigmoidal transfer function. So when comparing NEAT to other NE techniques, which do not use their modified sigmoidal transfer function, it is not known if the

Figure 1: A general form of CGP with chromosome given beneath, taken from [28]



differences between results are due to the NE techniques, the use of the modified sigmoidal transfer function, or both.

2. CARTESIAN GENETIC PROGRAMMING

CGP [28, 30] is a form of GP in which computational structures are organised as directed, often acyclic graphs indexed by their Cartesian coordinates. Each node may take its inputs from any previous node or program input (although recurrent graphs can also be implemented). The program outputs are taken from the output of any internal node or program input. This structure leads to many of the nodes described by the CGP chromosome not contributing to the final operation of the phenotype, these inactive, or "junk", nodes have been shown to greatly aid the evolutionary search [29, 44, 50]. The representational feature of inactive genes in CGP is also closely related to the fact that it does not suffer from bloat [27].

The nodes described by CGP chromosomes are arranged in a rectangular $r \times c$ grid of nodes, where r and c respectively denote the user-defined number of rows and columns. In CGP, nodes in the same column are not allowed to be connected together (as in multi-layer perceptrons). CGP also has a connectivity parameter l called "levels-back" which determines whether a node in a particular column can connect to a node in a previous column. For instance if $l = 1$ all nodes in a column can only connect to nodes in the previous column. Note that levels-back only restricts the connectivity of nodes; it does not restrict whether nodes can be connected to program inputs (terminals). It was later realised however that any architecture (limited by the number of nodes) could be constructed by arranging the nodes in a $1 \times n$ format where the n represents the maximum number of nodes (columns) and choosing $l = n$. Using this representation the user does not need to specify the topology, which is then automatically evolved along with the program.

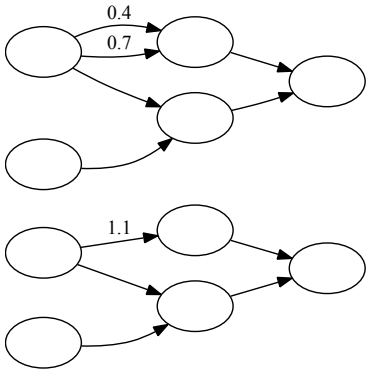
Figure 1 gives the general form of a CGP showing that a CGP chromosome can describe multiple input multiple output (MIMO) programs with a range of node transfer functions and arities. In the chromosome string, also given in Figure 1, F_i denote the function operation at each node, C_i index where the node gathers its inputs and each O_i denote which nodes provide the outputs of the program. It should be noted that CGP is not limited to only one data type, it may be used for Boolean values, floats, images, audio files, videos etc. CGP uses a $(1 + \lambda) - ES$, the mutation operator and rank selection; no crossover operator is used. The λ value used by CGP is commonly set as four, which is the value used throughout this paper. The connection genes in the chromosomes are initialised with random values that obey the constraints imposed by the three CGP structural parameters, r, c, l. The function genes are randomly chosen from the allowed values in the function lookup table. The output genes O_i are randomly initialised to refer to any node or input in the graph. The standard mutation operator in CGP is very simple, it is point mutation [28]. It works by randomly choosing a valid allele at a randomly chosen gene location. The reason why both a simple operator and a simple evolutionary algorithm are so effective is related to the presence of non-coding genes. Simple mutations can connect or disconnect whole sub-programs, thus this operator, during evolution explores a distribution of program topologies. Also the algorithm navigates the search space using neutral drift. For a more detailed description of CGP and these issues see [28].

2.1 Encoding Neural Networks

ANNs are a natural application for CGP as both are structured using directed acyclic graphs. CGP can also evolve cyclic graphs as used in recurrent ANNs. When CGPANNs are used to encode ANNs, the inputs are the same as used for CGP, the connections are also left unchanged, the functions are those suited to ANNs (sigmoid, hyperbolic tangent etc) and the outputs are taken as the output of any hidden node or input node. The only alteration necessary is an additional gene for each connection in order to encode the connection weights, as used by ANNs. These weights are real values within a range specified by the user; set as $[-1, 1]$ throughout this paper.

As a direct result of the encoding method used by CGPANN it is possible, and indeed probable, that there will be multiple connections between two nodes. This possibly novel behaviour is due to the fact that the inputs to each node are evolved and no interference is made to ensure they do not connect to the same node multiple times. The presence of these multiple connections has three consequences. The first is that the maximum effective weight range set by the user (often $[-1, 1]$) can be exceeded. This is due to the multiple connections between nodes being equivalent to one connection with the sum of the weights, see Figure 2. Secondly, having multiple connections between two nodes results in less connections to other nodes; this has the effect of variable arity, again see Figure 2. Thirdly having multiple connections between nodes could create resilience to future mutations, as removing one connection does not remove the

Figure 2: Depiction that multiple inter-node connections (above) can be simplified as one (below)



effect of the previous node on the latter. This ability to have multiple connections between nodes maybe possible in many NE methods which evolve topologies. However methods such as NEAT [42] or SAIN [31], which appear may be able to evolve multiple connections between nodes unless actively prevented, do not discuss the matter.

The user defined basic parameters which govern the operation of a CGPANN are: maximum number of generations, mutation percentage, maximum number of nodes (i.e. columms with rows set to one), connection weight range and maximum arity. Other parameters can also include: rows and columns, "levels-back" and transfer functions. In this paper, for simplicity, only the basic parameters are used with the "levels-back" parameter set as the number of nodes (columns). Although CGPANN (and CGP in general) can use many types of mutation operands, a simple point mutation is used throughout this paper; where each gene in the chromosome is changed to a new random value with a given probability.

In summary the advantages of evolving ANN using CGPANN include: MIMO networks, recurrent and non-recurrent networks, evolvable topology, evolvable arity, evolvable transfer functions, evolvable weights, no bloat and the presence of redundancy in the chromosomes aiding evolution.

3. RECENT RELATED WORK

To date there is very little published work on the application of CGP to ANN; what has been published we discuss here.

M. Khan et al. published [19, 20] describing the application of CGPANN to both the single and double pole balancing benchmarks. They showed remarkable results for both tasks, solving them faster (in terms of number of evaluations) than any other NE published method. M. Khan et al. also implemented a recurrent form of CGPANN [18] which was again applied to the double pole balancing problem. The reason for repeating the double pole balancing benchmark is to offer a fairer comparison with other published results. When M. Khan et al. conducted their experiments they allowed the transfer functions used within the nodes to also be evolved, by choosing between sigmoid and hyperbolic tangent. Although the ability to evolve the transfer function, along with the weights and topology, is one of the many advantages of NE methods, it was not used by other reported NE methods and so weakens the comparison. They also used a "bang-bang" control system, whereas most reported results for this benchmark use a continuous force. Additionally they added connection switches to the connections between nodes; these switches were evolved within the chromosomes as binary values indicating whether a connection is made between the nodes. Although this additional gene may be advantageous to the evolutionary process, this paper uses CGP in its original form; further alterations will be researched in the future.

Another application of CGP to ANN was published by G. Khan et el. who used CGP to evolve a multi-chromosome representation of realistic neurons to create neural networks which more closely mimic the biological brain. Their method was applied to the classic artificial intelligence problems wumpus world [16] and the game of checkers [17], where they showed their solution was capable of continuous learning within the task environment.

4. THE BENCHMARKS

The three benchmarks chosen to assess the effectiveness of CGPANN cover a range of different applications to which NE can be applied. They include the well known double[1] pole balancing [45], the relatively new ball throwing [21] and Cancer1 taken from the Proben1 set of benchmarks for ANNs [37]. The first two benchmarks are control based problems. The double pole balancing is a continuous (balancing) single output control problem and the ball throwing is a multiple output problem which includes a one off event (throwing) aspect. The final benchmark is a popular, and well used, classification problem.

4.1 Double Pole Balancing

The double pole balancing benchmark has been used for many decades in the artificial intelligence literature and is simply the task of balancing two poles attached to a cart with all components limited to one degree of freedom; see [45] for diagrams and additional information. The equations which govern the dynamics of the double pole experiment are given in Equations 1, 2, 3 and 4 with the symbol definitions and values given below.

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i} \tag{1}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{2}$$

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right) \tag{3}$$

$$\tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right) \tag{4}$$

Where x is the carts position limited to $[-2.4, 2.4]m$, θ_i is the i$^{\text{th}}$ poles angle from virtual limited to $[-36, 36]deg$, F is the force applied to the cart limited to $[-10, 10]N$, l_i is the half length of the i$^{\text{th}}$ pole with $l_1 = 0.5m$ and $l_2 = 0.05m$, M is the cart mass set as $1.0kg$, m_i is the mass of the i$^{\text{th}}$ pole with $m_1 = 0.1kg$ and $m_2 = 0.01kg$, μ_c is the cart-track

[1]There is also a single pole version but it appears be less standardised (different initial conditions, parameters and limits are used), and is also very easy for modern methods to solve, weakening any comparison.

friction set as 0.0005 and μ_{pi} is the i$^{\text{th}}$ pole-cart friction set as 0.000002.

The simulations are run using Euler integration with a time step of $0.01sec$ and the controller outputs were updated every $0.02sec$; the simulations were run for 100000 time steps in each experiment. The simulations were always initialised with the cart in the center of the track, the longer pole at $1deg$ from vertical and the short pole at vertical.

The fitness of the solutions are determined by the number of time steps that the controller can keep both poles within the range $[-36, 36]deg$. Some implementations used by others, such as [5], penalise rapid swinging[2]; this was not used in the experiments presented here. The inputs to the controller were the usual cart position, cart velocity, both pole positions and both pole velocities.

The inputs are scaled to be within $[-1, 1]$ by assuming their values would always be in the following ranges: cart position $[-2.4, 2.4]m$, cart velocity $[-1.5, 1.5]m/s$, pole positions $[-36, 36]deg$ and pole velocities $[-115, 115]deg/s$.

The output of the controller is also in the range $[-1, 1]$ which is then scaled to be between $[-10, 10]N$ in what is commonly called "continuous force" as opposed to the binary "bang-bang force" which is sometimes used. A restriction placed on the controller, also used by [6, 8, 45], is to limit the magnitude of the applied force to always be greater than $\frac{1}{256} \times 10N$; this ensures the controller cannot position the poles to be almost at the point of balance and then do nothing until they fall.

4.2 Ball Throwing

The Ball Throwing experiment is taken from [21] where it was used as a comparison between CoSyNE and Compressed CoSyNE training algorithms. The goal of the task is to throw a ball, initially attached at the end of a swinging arm, a distance of $9.50m$; slightly below the maximum possible distance of $10.202m$. The arm is attached to a pivot and initially hangs downwards under gravity; with the ball located at the lower end. The controller can only *see* two variables: the angle of the arm from vertical and the arm's angular speed. The controller also has two outputs: the torque applied to the arm and whether or not to release the ball.

Equation 5 describes the arm dynamics with the released ball obeying regular Newtonian mechanics until the ground is reached.

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

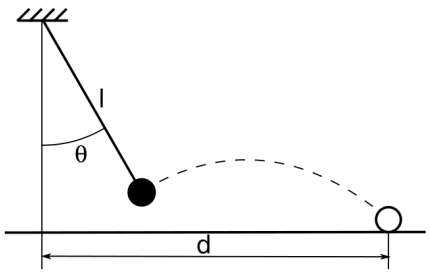
Where θ is the arm angle, ω is the arms angular speed, c is a constant to control the friction set as $2.5s^{-1}$, l is the arm length set as $2m$, g is the gravity set as $9.81ms^{-2}$, m is the mass of the ball set as $0.1kg$, and finally, T is the torque applied to the arm whose value is limited to $[-5, 5]Nm$.

Figure 3 shows the set up of the benchmark with θ and l labelled. It can also be seen in Figure 3 how d, the distance the ball is thrown, is measured. The value of d is then used as the fitness assigned to each possible solution.

An additional constraint is that ω is set to zero when $|\theta| \geq \pi/2$. The simulation is run using Euler integration with a time step of $0.01s$ and the controller outputs are also updated every $0.01s$.

[2]In these experiments [5] also removed velocity information.

Figure 3: Depiction of the Ball Throwing Benchmark taken from [21]



The inputs to the ANN are scaled to fit the range $[0, 1]$ by knowing that θ can never exceed $\pm\pi/2$ and by the fact that ω can never exceed $\pm 5rad/s$ (found through simulation; actual maximum value $\pm 4.4915rad/s$). The torque output of the ANNs is also scaled to meet the range $[-5, 5]Nm$. The ball is released when the release output is greater than 0.5.

As the original paper [21] did not state the transfer function used, it was assumed to be a regular sigmoid. The values used to scale the inputs into a $[0, 1]$ range were also not provided by the original paper and so logical choices were made.

4.3 Proben1: Cancer1

The Proben1 [37] document is a collection of many, mainly real world, classification benchmark problems for ANNs; along with best practices concerning how to report and compare results. Cancer1 is one of the fifteen classification benchmark problems and was originally constructed at the University of Wisconsin Hospital [24]; the '1' refers to the permutation of the data as described in [37].

There are a total 699 entries in the breast cancer data set each with the following details (scaled between $[0, 1]$): clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nuclei and mitosis. Each data set also contains two classification flags; benign and malignant. The data set contains 458 (66%) benign cases and 241 (34%) malignant. The first 525 entries are used for training and the following 174 are used for testing. In some cases the testing set is further divided into testing and validation so the generalisation ability can be assessed during the search for a solution; this was not done in this paper. As suggested in [37] the squared error percentage was used for the fitness function, Equation 6. 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 outputs. The produced classifier is then tested on the test data set, producing a test error percentage which is then used to compare different techniques. The training error percentage is also often given for comparison.

$$E = 100 \cdot \frac{o_{max} - o_{min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2 \tag{6}$$

The evolved ANNs have two outputs corresponding to whether the condition is benign or malignant. The output with the highest value is taken as the predicted class; in the case of a draw the entry is classified as benign.

5. RESULTS

When applying CGPANN to the benchmarks, the transfer function was fixed as unipolar sigmoid (Equation 7) in the case of the ball throwing and Proben1 Cancer1 and as bipolar sigmoid (Equation 8) for the pole balancing. Although CGPANN can use evolution to dictate which transfer functions are used, this was not undertaken here to strengthen the comparison with other results in the literature. The weight values used between nodes were limited to the range $[-1, 1]$ for all of the experiments. Additionally for each benchmark the following parameters were varied so as to be suitable in each case: mutation rate, maximum arity and maximum number of nodes.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{8}$$

The number of runs was set as one hundred for the pole balancing and ball throwing and as fifty for Proben1 Cancer1. Each run used different initial random chromosomes and was averaged to produce the results presented in this paper. In cases where a run failed to find a solution it was assumed that a solution was found on the last evaluation and the number of failed runs stated.

5.1 Double Pole Balancing

For this benchmark, the CGPANN parameters found which produced the best results were as follows: a mutation rate of five percent, a maximum of one hundred nodes each with a maximum arity of thirty. Using these parameters all of the one hundred runs found a solution in the allowed 16001 evaluations.

The results of applying CGPANN to the double pole balancing benchmark are given in Table 1 where they are compared to other NE methods. As can be seen in Table 1, CGPANN compares very well, outperforming popular methods SAIN, ESP and NEAT, being comparable to CoSyNE and CMA-ES but requiring significantly more evaluations than DirE.

Unfortunately the benchmark used by some of the other NE methods was implemented slightly differently, and so a true comparison is not possible. These differences include: NEAT used their own modified sigmoid function, DirE included an extra unit bias input, ESP used a weight range of $[-6, 6]$ and CoSyNE $[-10, 10]$ (the others used a range of $[-1, 1]$). It is likely that the transfer function, extra bias input and connection weight ranges will have an effect on the difficulty of this benchmark, but to what extent is not known.

5.2 Ball Throwing

For this benchmark, the CGPANN parameters found which produced the best results were as follows: a mutation rate of ten percent, a maximum of forty nodes each with a maximum arity of ten. Using these parameters ninety-two of the one hundred runs found a solution in the allowed 20001 evaluations.

Table 1: Comparison of Results for the Double Pole Balancing Benchmark

Method	Evaluations	Std. Dev	Avg'ed Over
EP [45]	307200	-	-
CE [8]	34000	-	>30
CNE [7]	22100	-	50
EuSAIN [35]	≈ 19000	-	100
SAIN [7]	12600	-	50
Q-MPL [7]	10583	-	50
ESP [7]	3800	-	50
NEAT [41]	3578	2704	120
NEvA [43]	2177	-	50
CGPANN	**1111**	**1476**	**100**
CoSyNE [6]	954	-	50
CMA-ES [12]	895	-	50
DirE [8]	410	-	>30

Table 2: Comparison of Results for the Ball Throwing Benchmark

Method	Evaluations	Std. Dev.
CoSyNE [21]	10224	-
Compressed CoSyNE [21]	8220	-
CGPANN	**6069**	**5990**

The results of applying CGPANN to the ball throwing benchmark are given in Table 2. It can be seen from Table 2 that CGPANN outperforms both of the given forms of CoSyNE.

5.3 Proben1: Cancer1

When studying which CGPANN parameters were suited to the Proben1 Cancer1 benchmark, the well know curse of over-training was observed; whereby a ANN can correctly classify more of the training set at the expense of its ability to correctly classify the testing set. Therefore two results are presented for this benchmark: the best result obtained in terms of the training error percentage and also in terms of the testing error percentage. Both of these results are given in Table 3, where "Tr" is the best training error percentage case and "Te" is the best testing error percentage case.

For this benchmark the lowest testing error percentage of 1.89% was found using the following parameters: a mutation rate of eight percent, a maximum of one hundred nodes each with a maximum arity of twenty-five. These parameters produced a training error percentage of 2.68%. The lowest training error percentage of 2.34% was found using the following parameters: a mutation rate of one percent, a maximum of one hundred nodes each with a maximum arity of forty. These parameters produced a testing error percentage of 2.18%. For all of the Cancer1 experiments the number of evaluations was 20001.

Most of the results published in the literature, given in Table 3, do not conform to the strict rules laid out in the Proben1 documentation [37]; which weakens the comparison. Of the results which do not follow the methodology of the Proben1 document, the methods used vary and are not standardised. The differences include: size of the data set, preprocessing of the data, sizes of training, validation

Table 3: Comparison of Results for the Proben1: Cancer1 Benchmark

Method	Proben Compliant	% Train Err	% Test Err
LM [9]	No	5.5	12.2
MLP [39]	No	-	6
SCG [9]	No	0.2	5.4
MLP [32]	No	-	5.18
RAIC [10]	No	-	5.01
NEFCLASS [32]	No	-	4.94
SFC [1]	No	-	4.43
C4.5 [10]	No	-	4
LLS [9]	No	4.0	4.0
Fuzzy-GA [33]	No	3.00	3.98
CMAC ANN [46]	Yes	0.59	3.94
RBFNN-Kalman [40]	No	-	3.6
BP [34]	Yes	-	3.506
GDX [9]	No	2.3	3.3
RBFNN-RLS [40]	No	-	3.2
LLWNN-RLS [40]	No	-	2.8
AR + ANN [14]	No	-	2.6
SBS-BP-PSO [11]	No	-	2.49
ACS [34]	Yes	-	2.184
CGPANN (Tr)	**Yes**	**2.34**	**2.18**
MFNNCA[3] [13]	Yes	24.86	2.00
GA-MOO-ANN [3]	Yes	-	1.9
CGPANN (Te)	**Yes**	**2.68**	**1.89**
M-RAN [49]	Yes	-	1.72
LP MSM [23]	No	0.0	1.7
LS-SVM [36]	No	-	1.47
MFN [49]	Yes	-	1.38
EPNet [48]	No	3.773	1.376
LSA machine [4]	No	-	1.2
SBS-BP-LM [11]	No	-	1.17
HS [15]	No	-	0.71

and test sets, different permutations of the data set and the fitness functions used. Of the results which do follow the Proben1 document, some split the testing dataset into testing and validation; this is allowed and documented by the Proben1 standards. This validation dataset can then be used to test how well each solution has generalised and is used as an early stopping criteria. An early stopping criteria was not used by the CGPANN presented here; however CMAC ANN, MFNNCA and GA-MOO-NN did use an early stopping criteria.

6. DISCUSSION

As can be seen by the results given in Tables 1, 2 and 3, CGPANN preformed very well on the double pole balancing compared to other NE methods, outperformed both forms of CoSyNE on the ball throwing and performed well on the Proben1: Cancer1 benchmark compared to a wide range of ML methods. It should be noted that many of the methods used as comparison were not topology optimising techniques; therefore decisions including the number of nodes to be used and their arrangement must be made prior to their use. CGPANN, in contrast, does not need the user to specify a topology, and only requires a maximum number of nodes to be defined. Although not undertaken in this paper, CGPANN can also evolve the transfer functions to be used within the nodes; networks containing many different node types could then be generated without increased complexity to the implementation. Therefore when creating ANNs using CGPANN the user does not need to make decisions about the number, arrangement or even type of nodes to be evolved; information which is often not available in real world tasks. On the other hand if a specific topology or transfer function is known to be suitable, CGPANN can then be restrained to these conditions, leaving evolution to only operate on the unknowns; again making it very suited to real world tasks where some domain knowledge is available.

From the double pole balancing and the Proben1: Cancer1 results sections, it is clear that there is an issue with non-standardised use of these benchmarks. The majority of the double pole balancing did follow the same standards, with the exceptions stated. This was not the case however for the Proben1: Cancer1 benchmark. It should be noted that this does not reflect poorly on the published papers, for example, some of the Cancer1 results pre-date the Proben1 document and others were not concerned with the comparison and simply wanted to show that medical diagnosis is an application for a given method. It should be stated however that there are enormous benefits of following the same standards, such as being able to repeat experiments, truly compare methods/ results and then being able to accurately investigate why different methods perform differently on different tasks.

Allowing multiple connections between nodes could be interpreted as another example of slightly different ways in which a benchmark could be implemented; this is not the case. When CGP was originally applied to Boolean circuits [26] it was perfectly reasonable, for example, for both inputs to a two input AND gate to be connected to the same point. As the implementation of CGPANN presented here is CGP in its original form but applied to ANN, no extra criteria was made to prevent multiple connection between nodes. This is an artefact of CGP and not an example of a difference in benchmark implementation. However, if it was decided that exceeding a given weight range was undesirable, it could easily be prevented whilst maintaining the ability of CGPANN to evolve the arity of the nodes'. For example, only the first connection could be decoded into the phenotype, or the average of the weights could be used.

The presence of multiple connections between nodes allows CGPANN to evolve the arity of each node. Therefore CGPANN can evolve the arity, weights, topology and transfer functions of ANNs, everything necessary to fully describe an ANN. It is not known currently whether the ability to evolve arity in this way offers an advantage. It is known however that it was extensively utilised by the generated ANNs, as many instances of multiple connections were found in every final ANN inspected for all three benchmarks. If it is found that multiple connections between nodes is beneficial it could be utilised by other topology evolving NE methods. On the other hand if it is found to be insignificant or detrimental it can easily be removed from CGPANN with no impact to its operation.

The average number of evaluations (or epochs) required

[3]The experiment was repeated for a range of epochs and the best result given.

to find a solution is not often given when ML methods are applied to classification problems. This however has the issue that different methods may require significantly more evaluations to achieve their result. As an extreme, a truly random search given infinite time will always find the best solution but is clearly not the best search method. It is also true that some methods get trapped in local optima and, regardless of the amount of time available, will never find better solutions. A significant advantage of NE methods is that they are much less susceptible to getting trapped in local optima and will often continue to find better solutions with additional time. It is therefore recommended that when applying ML methods to classification probelms, the number of trial solutions tested before a solution is reached is stated.

Between submission and acceptance of this paper there were two publications very closely related to the research presented in this paper. The first is the use of CGPANN on a range of medical diagnosis problems [2] including the breast cancer dataset taken from the University of Wisconsin Hospital [24]; as used in this paper. The second is an implementation of CGPANN which uses the crossover evolutionary operator and radial bias functions within the nodes [25]; this paper also used the breast cancer dataset to evaluate performance.

7. CONCLUSION

CGP is naturally suited to the evolution of ANNs due to its ability to describe every aspect of a ANN; weights, topology, number of nodes, arity and transfer function. This complete description means that very few assumptions concerning the form of solutions need to be made prior to applying CGPANN to a problem. Additionally, a CGPANN can be constrained when domain knowledge is available, making CGPANN suited to real world problems. The results presented in this paper show that CGPANN compares strongly to many other NE and ML techniques; many of which require the user to select suitable topologies prior to training. CGPANN has many of the advantages associated with other NE methods, resilience to local optima, not requiring gradient information and being suited to reinforcement learning, coupled with advantages more specific to CGP, resilience to bloat and redundancy in the chromosomes.

8. REFERENCES

[1] J. Abonyi and F. Szeifert. Supervised fuzzy clustering for the identification of fuzzy classifiers. *Pattern Recognition Letters*, 24(14):2195–2207, 2003.

[2] A. M. Ahmad and G. M. Khan. Bio-signal processing using cartesian genetic programming evolved artificial neural network (cgpann). In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 261–268. IEEE, 2012.

[3] F. Ahmad, N. Mat Isa, Z. Hussain, and S. Sulaiman. A genetic algorithm-based multi-objective optimization of an artificial neural network classifier for breast cancer diagnosis. *Neural Computing & Applications*, pages 1–9, 2012.

[4] A. Albrecht, G. Lappas, S. Vinterbo, C. Wong, and L. Ohno-Machado. Two applications of the LSA machine. In *Proceedings of the 9th International Conference on Neural Information Processing*, volume 1, pages 184–189. IEEE, 2002.

[5] F. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1356–1361, 1999.

[6] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *The Journal of Machine Learning Research*, 9:937–965, 2008.

[7] F. J. Gomez and R. Miikkulainen. *Robust non-linear control through neuroevolution*. PhD thesis, 2003.

[8] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. MIT Press, 1996.

[9] B. Guijarro-Berdiñas, O. Fontenla-Romero, B. Pérez-Sánchez, and P. Fraguela. A linear learning method for multilayer perceptrons using least-squares. *Intelligent Data Engineering and Automated Learning*, pages 365–374, 2007.

[10] H. Hamilton, N. Shan, and N. Cercone. RIAC: A rule induction algorithm based on approximate classification. Technical report, University of Regina, 1996.

[11] M. Huang, Y. Hung, and W. Chen. Neural network classifier with entropy based feature selection on breast cancer diagnosis. *Journal of medical systems*, 34(5):865–873, 2010.

[12] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation*, volume 4, pages 2588–2595. IEEE, 2003.

[13] S. Kamruzzaman, A. Hasan, A. Siddiquee, M. Mazumder, and E. Hoque. Medical diagnosis using neural network. In *Proceedings of the International Conference on Electrical and Computer Engineering (ICECE-2004)*, pages 537–540, 2004.

[14] M. Karabatak and M. Ince. An expert system for detection of breast cancer based on association rules and neural network. *Expert Systems with Applications*, 36(2):3465–3469, 2009.

[15] A. Kattan, R. Abdullah, and R. Salam. Harmony search based supervised training of artificial neural networks. In *International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pages 105–110. IEEE, 2010.

[16] G. Khan, J. F. Miller, and D. Halliday. A developmental model of neural computation using cartesian genetic programming. In *Genetic And Evolutionary Computation Conference: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, volume 7, pages 2535–2542, 2007.

[17] G. Khan, J. F. Miller, and D. Halliday. Developing neural structure of two agents that play checkers using cartesian genetic programming. In *Proceedings of the 2008 GECCO conference on Genetic and evolutionary computation*, pages 2169–2174. ACM, 2008.

[18] M. Khan, G. Khan, and J. F. Miller. Efficient representation of recurrent neural networks for markovian/non-markovian non-linear control problems. In *Intelligent Systems Design and*

Applications (ISDA), 2010 10th International Conference on, pages 615–620. IEEE, 2010.

[19] M. Khan, G. Khan, and J. F. Miller. Evolution of optimal ANNs for non-linear control problems using cartesian genetic programming. In *Proceedings of International Conference on Artificial Intelligence (ICAI 2010)*, 2010.

[20] M. M. Khan, G. M. Khan, and J. F. Miller. Evolution of neural networks using cartesian genetic programming. In *Proceedings of IEEE World Congress on Computational Intelligence*, 2010.

[21] J. Koutník, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-10)*, 2010.

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

[23] K. Mangasarian. Neural network training via linear programming. *Advances in Optimisation and Parallel Computing*, pages 56–67, 1992.

[24] O. Mangasarian, R. Setiono, and W. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization*, pages 22–31, 1990.

[25] T. Manning and P. Walsh. 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*, pages 165–176. Springer, 2013.

[26] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142. Citeseer, 1999.

[27] J. F. Miller. What bloat? cartesian genetic programming on boolean problems. In *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, 2001.

[28] J. F. Miller, editor. *Cartesian Genetic Programming*. Springer, 2011.

[29] J. F. Miller and S. Smith. Redundancy and computational efficiency in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 10(2):167–174, 2006.

[30] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000)*, volume 10802, pages 121–132. Springer-Verlag, 2000.

[31] D. Moriarty and R. Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine learning*, 22(1):11–32, 1996.

[32] D. Nauck and R. Kruse. Obtaining interpretable fuzzy classification rules from medical data. *Artificial intelligence in medicine*, 16(2):149, 1999.

[33] C. Pena-Reyes and M. Sipper. A fuzzy-genetic approach to breast cancer diagnosis. *Artificial intelligence in medicine*, 17(2):131–155, 1999.

[34] N. Pokudom. Determine of appropriate neural networks structure using ant colony system. In *ICCAS-SICE, 2009*, pages 4522–4525. IEEE, 2009.

[35] D. Polani and R. Miikkulainen. Fast reinforcement learning through eugenic neuro-evolution. Technical report, University of Texas at Austin, Austin, TX, 1999.

[36] K. Polat and S. Güneş. Breast cancer diagnosis using least square support vector machine. *Digital Signal Processing*, 17(4):694–701, 2007.

[37] L. Prechelt. Proben1: A set of neural network benchmark problems and benchmarking rules. *Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep*, 21:94, 1994.

[38] J. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8(1):73–84, 1998.

[39] A. Raad, A. Kalakech, and M. Ayache. Breast cancer classification using neural network approach: MLP and RBF. *Networks*, 7(8):9, 2012.

[40] M. Senapati, A. Mohanty, S. Dash, and P. Dash. Local linear wavelet neural network for breast cancer recognition. *Neural Computing & Applications*, pages 1–7, 2011.

[41] K. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1757–1762. IEEE, 2002.

[42] K. O. Stanley. *Efficient evolution of neural networks through complexification*. PhD thesis, The University of Texas at Austin, 2004.

[43] Y. Tsoy and V. Spitsyn. Using genetic algorithm with adaptive mutation mechanism for neural networks design and training. In *Science and Technology. Proceedings. The 9th Russian-Korean International Symposium on*, pages 709–714. IEEE, 2005.

[44] V. K. Vassilev and J. F. Miller. The Advantages of Landscape Neutrality in Digital Circuit Evolution. In *Proc. International Conference on Evolvable Systems*, volume 1801 of *LNCS*, pages 252–263. Springer Verlag, 2000.

[45] A. Wieland. Evolving neural network controllers for unstable systems. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 667–673. IEEE, 1991.

[46] J. Wu. MIMO CMAC neural network classifier for solving classification problems. *Applied Soft Computing*, 11(2):2326–2333, 2011.

[47] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[48] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *Neural Networks, IEEE Transactions on*, 8(3):694–713, 1997.

[49] L. Yingwei, N. Sundararajan, and P. Saratchandran. Performance evaluation of a sequential minimal radial bass function (RBF) neural network learning algorithm. *Neural Networks, IEEE Transactions on*, 9(2):308–318, 1998.

[50] T. Yu and J. F. Miller. Neutrality and the evolvability of boolean function landscape. *Genetic programming*, pages 204–217, 2001.