NeuroEvolution: The Importance of Transfer Function Evolution and Heterogeneous Networks

Andrew James Turner [1] and **Julian Francis Miller** [2]

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 most NeuroEvolutionary methods are capable of creating heterogeneous networks using the methods described.

1 Introduction

NeuroEvolution (NE) is the application of Evolutionary Algorithms (EA) to the training of Artificial Neural Networks (ANN). NE's history began by evolving the connection weights of fixed topology ANNs [23, 35]. This method brought many advantages over the still popular gradient based methods; such as back propagation [24]. These advantages include: being able to 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 [37]. 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 [26].

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 [27], SAGA [4] and CGPANN [8, 29]. This ability to automatically create suitable topologies is significant as topology has been shown to strongly influence the effectiveness of back propagation [10] and weight only evolving NE [30]. Evolving the topology of ANNs has even been shown to be more important to training than evolving connection weights [30]. 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].

Interestingly, NE can also be used to optimise the TF of each neuron within heterogeneous ANNs. 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 surrounding 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" [37], "The current emphasis in neural network research is on learning algorithms and architectures, neglecting the importance of transfer functions" [5] 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" [6]. 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 remained of this paper is structured as follows. Section 2 introduces some background of applying NE to evolving the TFs of ANNs. Section 3 describes the investigations which were undertaken using NE to evolve TFs, with the results given in Section 4. Finally Section 5 discusses the overall findings with final conclusions given in Section 6.

2 Background

There are vast number of ANN TFs found in the literature [6]. However, the majority of NE implementations only evolve homogeneous ANNs of logistic functions or Gaussian functions; which have both been shown capable of universal approximation; [7] and [20] 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) [11]; 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) [21], a modified Hierarchical Co-evolutionary Genetic Algorithm ($HCGA_2$) [34] and Cartesian Genetic Programming of Artificial Neural Networks (CGPANN) [8, 29]. 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 way in which NE can optimise neuron TFs is to use TFs which are described by a number of parameters [6]. The training methods then optimises these parameters for each individual neuron. A simple version of this technique has been used by CGPANN [13]; where the widths of Gaussian functions were optimised for each neuron. Again the parameter associated with each neuron was encoded in the chromosome by the addition of an extra gene for each neuron.

[1] The University of York, England, email: andrew.turner@york.ac.uk
[2] The University of York, England, email: julian.miller@york.ac.uk

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 variations of TFs. Another example where each neuron is described by a number of genes, is state-enhanced neural networks [19], where the dynamics of each neuron are evolved. These state-enhanced neural network exhibit memory which can be utilised on partly observable Markov decision tasks.

Up 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 an additional gene(s) per 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. Both of these methods are investigated here using two NE strategies and compared to evolving regular homogeneous ANNs.

3 Investigation

The investigation presented in this paper takes three parts. The first is to identify if the choice of TF impacts on the effectiveness of NE when using homogeneous ANNs. The second investigates allowing NE to select each neuron's TF from a predetermined list. The third investigates using NE to optimise parameters associated with each neuron's TF. It would also be possible to use NE to evolve ANNs with a range of TFs each of which had separate parameters to be optimised; but this was not undertaken here.

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 only evolves connection weights of fixed topology networks. CGPANN is a more complex NE method which evolves both the 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[3].

3.1.1 Conventional NeuroEvolution

CNE [23] operates by storing the connection weights of a fixed topology network as an array of floating point numbers; each within a range specified by the user. 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 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] Sometimes refereed to as TWEANNs - Topology and Weight Evolving Artificial Neural Networks.

3.1.2 Cartesian Genetic Programming of Artificial Neural Networks

CGPANN [8, 29] is the application of Cartesian Genetic Programming (CGP) to the evolution of ANNs. CGP [18, 17] is a form of Genetic Programming (GP) which represents computational structures as a directed graph of nodes indexed by Cartesian coordinates. CGP does not suffer from bloat [15, 31]; an Achilles heel of many GP methods [25]. CGP chromosomes also contain non-functioning genes enabling neutral genetic drift during evolution [33, 38]. CGP typically evolves acyclic networks but can also be easily adapted to evolve cyclic or recurrent networks. CGP typically uses point or probabilistic mutation and no crossover. CGP is easily applied to ANNs [8, 29] by the inclusion of connection weight genes and by using TFs suited to ANNs. CGPANN has all of the benefits of CGP and is a NE training method which can evolve the weights, topology [30] and TFs of ANNs. Although CGP freely 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 [16]. Similarly, a maximum neuron arity must be specified, however, the arity of each neuron can be lower than this maximum [29]. 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 weight value being the sum of the individual connection 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 1 gives an example of the type of ANN which can be created using CGPANN. It can be seen that the neuron inputs are highly unrestrained; they can connect to any previous neuron in the network. It can also be seen that the arity of each neuron can vary. Additionally any neuron can be used as an output; including input neurons. Figure 1 demonstrates that when using NE to optimise topology, evolution is capable of utilising topologies which would be unlikely to be considered by a human designer.

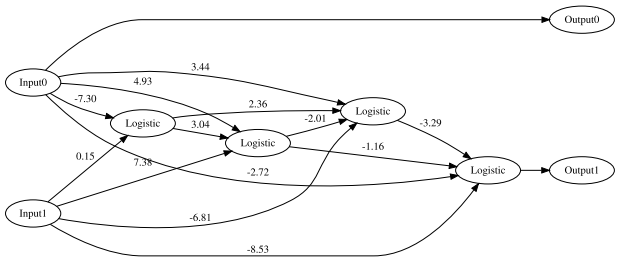

Figure 1. Depiction of the types of ANN created using CGPANN.

3.2 Transfer Functions

The TFs used for the first two parts of the investigation are the Heaviside step function, Equation 1, the Gaussian function, Equation 2, and the logistic function[4], Equation 3. Each of these TFs is shown

[4] The logistic function is often 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 function is therefore a specific type of sigmoid function along with other functions including the Gompertz function.

graphically in Figure 2. These particular TFs were selected as they are the most commonly used by ANNs.

As can be seen in Equations 2 and 3, the Gaussian and logistic function have been given in a form which contains a σ variable. Where σ is set as one for the typical form of these TFs. When using NE to evolve parameters associated with each neuron, the σ value can be evolved or optimised. Figures 3 and 4 show the Gaussian and logistic function respectively for a range of σ values.

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

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

$$f(x) = \frac{1}{1 + \exp(-\sigma x)} \tag{3}$$

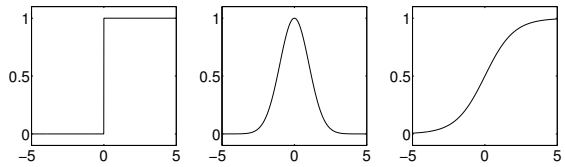

Figure 2. Form left to right: Heaviside step function, Gaussian function and the logistic function. With $\sigma = 1$ for the Gaussian and logistic functions.

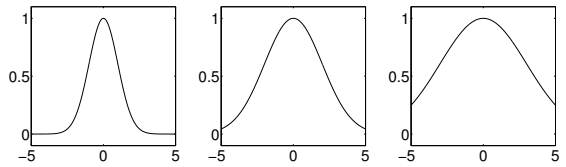

Figure 3. Variable Gaussian function. From left to right σ = 1,2 and 3.

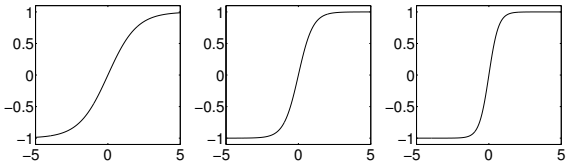

Figure 4. Variable logistic function. From left to right σ = 1,2 and 3.

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 were employed. The chosen benchmarks mainly include supervised learning classification tasks, a common application of ANNs, but also include a reinforcement learning control task (ball throwing).

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 are those typically used with these benchmarks. This is done to ensure the standardised use of these benchmarks; which is important when comparing machine learning methods.

3.3.1 Ball Throwing

The ball throwing benchmark [9] 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 5, with the equations describing the dynamics of the arm given in Equations 4 and 5; 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.

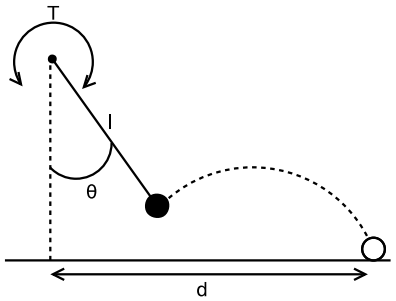

Figure 5. Depiction of the ball throwing benchmark.

$$\left(\dot{\theta}, \dot{\omega}\right) = \left(\omega, -c \cdot w + \frac{g \cdot \sin(\theta)}{l} + \frac{T}{m \cdot l^2}\right) \tag{4}$$

$$\omega = 0 \text{ if } |\theta| \geq \pi/2 \tag{5}$$

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	$[-5, 5]Nm$

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 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 [28] 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. 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.

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

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 [3]. The benchmarks consists of 194 data points describing samples taken from two spirals in Cartesian space; see Figure 6. The task is to classify to which spiral each sample belongs using only the (x, y) Cartesian coordinates. The target fitness is zero miss classifications. The ANN comprises of two inputs for the (x, y) Cartesian coordinates of each sample, and one output. when the output value is < 0.5 it is interpreted as one class and ≥ 0.5 as the other.

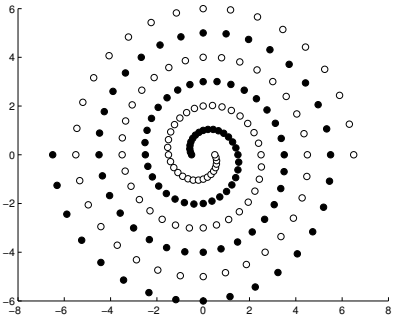

Figure 6. Depiction of the Two Spiral Classification Benchmark.

3.3.5 Proben1: Cancer1

The Cancer1 dataset[5] is a classification task taken from the Proben1 document [22]. The dataset was originally constructed at the University of Wisconsin Hospital [12]. 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 labeled 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 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. Therefore the optimum corresponds to a squared percentage error equal to zero.

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

4 Results

Three experiments are presented here which investigate the influence of TFs when using NE to train ANNs. All of the results presented are taken from fifty repeated runs. Each run was terminated after 1000 generations, all used a $(1 + 4)$-ES, 3% probabilistic mutation[6] and connection weights in the range ± 5. It should be noted that all of the results presented could likely be improved given more generations and are not representative of the maximum ability of any of the employed methods. 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 the effect size [32] statistics. A U-test value of < 0.05 indicates that the difference between two datasets is statistically significant. The effect size value

[5] The '1' in 'Cancer1' refers to the permutation of the dataset; see [22].

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

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 that the difference is significantly large.

4.1 Experiment 1 - Homogeneous Networks

In homogeneous ANNs the TF used by each neuron is the same, whereas in heterogeneous ANNs the network uses different types of TF for different neurons. The first experiment identifies whether, and to what extent, the choice of TF impacts on the effectiveness of training homogeneous ANN 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 5; 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 6 and 7; when using CNE and CGPANN respectively. When the difference is statistically significant the value is given in bold. The effect size of the differences between the fitnesses are given in Tables 8 and 9; when using CNE and CGPANN respectively. When the effect size is of medium or grater importance the value is given in bold.

From the results given in Tables 4 and 5 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 NE. Interestingly the most suitable TF was often dependant on the NE training method used. Interestingly again, for the classification tasks with testing sets, the best TF for training error percentage was not always the best for generalisation.

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

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	5.63	**6.41**	5.57	5.87
Full Adder	**16.00**	15.92	15.86	15.93
Monks Problem 1 Train	**9.82**	27.65	11.03	16.17
Monks Problem 1 Test	27.98	43.16	**25.87**	32.34
Two Spirals	70.00	**56.54**	81.52	96.35
Proben1: Cancer Train	10.50	5.44	**3.35**	6.43
Proben1: Cancer Test	14.44	7.49	**3.54**	8.49

Table 5. Average fitness of homogeneous ANN using different TFs trained using CGPANN.

Benchmark	Step	Gaussian	Logistic	Average
Ball Throwing	**9.34**	7.34	5.80	7.49
Full Adder	**15.94**	15.40	15.78	15.71
Monks Problem 1 Train	**10.71**	15.27	12.72	12.90
Monks Problem 1 Test	**13.44**	21.93	18.79	18.05
Two Spirals	67.42	**66.36**	80.64	71.47
Proben1: Cancer Train	**2.16**	2.55	2.50	2.40
Proben1: Cancer Test	2.71	2.74	**2.09**	2.51

Table 6. Statistical significance between the CNE results given in Table 4.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	3.55E-1	**9.55E-15**	4.68E-1
Full Adder	**4.33E-2**	**6.49E-3**	3.43E-1
Monks Problem 1 Train	**6.41E-18**	**3.81E-2**	**7.10E-18**
Monks Problem 1 Test	**6.92E-18**	**3.34E-2**	**6.94E-18**
Two Spirals	**2.41E-17**	**6.86E-18**	**4.94E-18**
Proben1: Cancer Train	**1.72E-14**	**6.34E-18**	**3.54E-1**0
Proben1: Cancer Test	**2.26E-13**	**6.19E-18**	**1.49E-11**

Table 7. Statistical significance between the CGPANN results given in Table 5.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**4.09E-10**	**2.06E-13**	1.22E-1
Full Adder	**1.39E-4**	1.01E-1	**1.52E-2**
Monks Problem 1 Train	**6.92E-3**	**2.12E-2**	**4.66E-2**
Monks Problem 1 Test	**1.75E-5**	**1.02E-2**	**9.30E-3**
Two Spirals	3.14E-1	**3.83E-17**	**2.53E-17**
Proben1: Cancer Train	**7.12E-7**	**5.07E-7**	7.91E-1
Proben1: Cancer Test	8.99E-1	**6.51E-3**	**2.85E-3**

Table 8. Effect Size between the CNE results given in Table 4.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	0.553	**0.940**	0.542
Full Adder	0.54	0.570	0.530
Monks Problem 1 Train	**∼1**	0.620	**0.999**
Monks Problem 1 Test	**∼1**	0.624	**∼1**
Two Spirals	**0.991**	**0.998**	**∼1**
Proben1: Cancer Train	**0.945**	**∼1**	**0.863**
Proben1: Cancer Test	**0.925**	**∼1**	**0.890**

Table 9. Effect Size between the CGPANN results given in Table 5.

Benchmark	Step-Gauss	Step-Log	Gauss-Log
Ball Throwing	**0.863**	**0.912**	0.587
Full Adder	**0.657**	0.552	0.608
Monks Problem 1 Train	**2.85E-3**	0.632	0.615
Monks Problem 1 Test	**2.85E-3**	**0.647**	**0.649**
Two Spirals	0.558	**0.988**	**0.990**
Proben1: Cancer Train	**2.85E-3**	**0.787**	0.571
Proben1: Cancer Test	0.507	**0.655**	**0.669**

4.2 Experiment 2 - Heterogeneous Networks

The second experiment identifies if allowing NE to select the TF for each neuron from a predetermined list is beneficial; and if so to what extent. Evolving the TF used by each neuron is considered beneficial if the result is better than the average of using each TF individually. This 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. Here the need to make this choice is removed, and hence it should be considered beneficial if it beats the average random choice. The average fitnesses of using each TF individually for the five benchmarks are given in Tables 4 and 5 for CNE and CGPANN respectively.

The average fitnesses achieved when evolving heterogeneous ANN are given in Tables 10 and 11 for CNE and CGPANN respectively. The results are given in bold if the fitness is better that the average of using each TF individually. The percentage of neurons which use each TF is also given in Tables 10 and 11; 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 10 and 11, 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 ANN. 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.

Table 10. Average fitness of heterogeneous ANN trained using CNE. The percentage of neurons which used each TF is also given.

Benchmark	Train	Test	Step	Gaussian	Logistic
Ball Throwing	**8.83**	-	36.0%	33.1%	30.9%
Full Adder	**16.00**	-	32.4%	34.1%	33.5%
Monks Problem 1	16.87	33.69	33.3%	31.5%	35.2%
Two Spirals	**63.46**	-	31.5%	35.2%	33.4%
Proben1: Cancer	**3.87**	**5.16**	31.8%	32.2%	36.0%

Table 11. Average fitness of heterogeneous ANN trained using CGPANN. The percentage of neurons which used each TF is also CGPANN.

Benchmark	Train	Test	Step	Gaussian	Logistic
Ball Throwing	**8.90**	-	35.5%	32.6%	31.9%
Full Adder	15.68	-	32.0%	37.9%	30.1%
Monks Problem 1	**11.02**	**16.72**	29.4%	34.8%	35.9%
Two Spirals	**70.24**	-	32.6%	35.4%	32.0%
Proben1: Cancer	**2.33**	2.69	30.9%	32.5%	36.6%

4.3 Experiment 3 - Evolving Transfer Function Parameters

The third experiment identifies if optimising parameters associated with each neuron is beneficial for NE. As discussed, the parameters to be optimised vary the shape of the Gaussian and logistic functions; see Section 3.2. In each case the ANNs use a fixed TF, Gaussian or logistic, but a parameter describing the shape of each neuron's TF is optimised or evolved. Here the parameter values for each TF is limited to the values 1, 2, or 3, see Equations 2 and 3; but this is not a requirement of the method.

Evolving parameters associated with each neuron 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.

The results of using variable Gaussian and variable logistic functions on the five benchmarks are given in Tables 12 and 13 respectively when using CNE. Similarly Tables 14 and 15 give the results when using CGPANN. In all cases the results are compared to those obtained for the non-variable form of the function. In the given Tables, a bold fitness value indicates that the variable TF performed better than the non-variable form. Additionally, bold values for the the U-test and effect size indicate statistical significance and a meaningful difference respectively. For instance, if the fitness, U-test and effect size values are all given in bold then the variable TF is shown to outperform the non-variable counterpart. If however the fitness value is not bold, but the U-test and effect size values are, then this shows that the non-variable TF outperformed the variable counterpart. If either of the U-test or effect size values are not bold the the difference between the two forms of TF is considered insignificant.

It can be seen in Tables 12-15, that 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 three of the twenty cases show the non-variable form outperforming the variable form with statistical significance and medium to large effect size. Seven of the twenty cases showed the variable form to outperform the non-variable form with statistical significance and a medium to large effect size. The remaining ten cases showed no significant difference between the variable and the non-variable TFs. Therefore using variable TFs is shown to be often beneficial and rarely worse. Interestingly, the variable form of the TFs were shown to be more beneficial for CNE than CGPANN. In five of the ten cases which used CNE, the variable TFs outperformed the non-variable TFs, compared to only two out of the ten cases for CGPANN. The spread of improvement between Gaussian TFs and the logistic TFs was roughly even; for the Gaussian TF three of the twenty cases found the variable form to be more beneficial whereas for the logistic TF this was four of the twenty cases.

Table 12. Average fitness of ANNs using the variable Gaussian TF trained using CNE.

Benchmark	Gaussian_Var	U-test	Effect Size
Ball Throwing	**8.15**	**2.66E-7**	**0.799**
Full Adder	**15.96**	4.07E-1	0.520
Monks Problem 1 Train	**26.24**	2.18E-2	0.633
Monks Problem 1 Test	**41.99**	**9.58E-3**	**0.650**
Two Spirals	66.26	**6.76E-12**	**0.898**
Proben1: Cancer Train	**3.09**	**5.28E-12**	**0.900**
Proben1: Cancer Test	**3.53**	**2.44E-11**	**0.886**

Table 13. Average fitness of ANNs using the variable logistic TF trained using CNE.

Benchmark	Logistic_Var	U-test	Effect Size
Ball Throwing	**6.21**	**6.79E-6**	**0.744**
Full Adder	**16.00**	**6.49E-3**	0.570
Monks Problem 1 Train	**10.45**	3.72E-1	0.552
Monks Problem 1 Test	27.00	1.37E-1	0.586
Two Spirals	**74.28**	**3.74E-16**	**0.970**
Proben1: Cancer Train	3.89	**2.94E-3**	**0.672**
Proben1: Cancer Test	4.79	**1.58E-4**	**0.718**

Table 14. Average fitness of ANNs using the variable Gaussian TF trained using CGPANN.

Benchmark	Gaussian_Var	U-test	Effect Size
Ball Throwing	**7.62**	2.21E-1	0.571
Full Adder	**15.72**	6.50E-2	0.586
Monks Problem 1 Train	**15.26**	8.21E-1	0.513
Monks Problem 1 Test	**21.59**	5.14E-1	0.538
Two Spirals	69.50	**2.77E-3**	**0.673**
Proben1: Cancer Train	**2.48**	5.01E-1	0.538
Proben1: Cancer Test	**2.31**	5.71E-2	0.608

Table 15. Average fitness of ANNs using the variable logistic TF trained using CGPANN.

Benchmark	Logistic_Var	U-test	Effect Size
Ball Throwing	**7.82**	**2.20E-7**	**0.766**
Full Adder	15.74	7.76E-1	0.511
Monks Problem 1 Train	**10.07**	3.65E-2	0.621
Monks Problem 1 Test	**17.26**	1.70E-1	0.579
Two Spirals	**75.60**	**2.37E-8**	**0.823**
Proben1: Cancer Train	**2.42**	1.01E-1	0.592
Proben1: Cancer Test	2.28	2.77E-1	0.561

5 Discussion

The results presented for the first experiment demonstrate 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; this accords with the 'No Free Lunch' theorem [36]. However, although intuitive, it is a significant result as a user is unlikely to know, in advance of training, which TFs are most suited for a given task. The user must therefore except possibly poor results, or repeat the learning process using a range of TFs.

Another interesting result from the first experiment is that, in the majority of cases, the Heaviside step function was found to be the most effective TF. The significantly more popular logistic function was found to be the most effective TF in only *one* case; comparing the training fitness values for the classification tasks. The step function was the original TF used by the McCulloch and Pitts neuron models [14]. The fact that the step function is incompatible with the back propagation algorithm, and is only suited to tasks with binary outputs, is the likely reason why 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 modern day ANNs; provided the task is compatible with binary outputs.

The second experiment demonstrated that allowing NE to select the TF of each neuron provided a better training method than the average of using homogeneous ANNs of each individual TF. This is significant because, as the first experiment shows, selecting the wrong TF for homogeneous networks has a large impact on the effectiveness of the final ANN. This coupled with the fact there is no way of knowing which TF will be most suited for a given task before training begins, puts homogeneous ANN at a strong disadvantage. This is an important result for NE as the addition of a gene describing the TF used by each neuron is probably compatible with all NE methods. The result may even be strengthened by the inclusion of other TFs not considered here. Also, as NE places no restrictions on the types of TFs used, the range of possible TF is limitless.

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; which would have indicated that it was the TF evolution found most suited toward the given task. There was, however, reasonable variation in the percentages of each type of TF used; showing that evolution was providing some form of pressure to use a particular type of TF i.e. it was not simply random. It could also be the case that many of the neurons which used a particular type of TF also used connection weights small in magnitude. For instance, if many of the neurons which used the logistic function also used connections weights which approached zero, then it would be clear that evolution had not found the logistic function useful towards the given task. However the magnitude of the connection weight were not considered here.

The third experiment demonstrated that, in the majority of cases, using NE to optimise parameters associated with each neuron provided either a better training method or had no significant effect compared to using using non parametrised TFs. This is also an important result as the 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.

A further result form the third experiment was that CNE benefited more from variable TFs than CGPANN. This is interesting as the main difference between CNE and CGPANN is that CGPANN can evolve topology as well as connection weights. This could indicate that there is some from of interaction between evolving topology and evolving TFs. This interaction is likely highly complex.

This paper used a very limited method for allowing NE to optimise parameters associated with each neuron; each neuron had a single parameter and the parameter value was limited to 1,2 or 3. It does however demonstrate the concept of optimising individual parameters for each neuron and, even in this this simple form, showed it to be advantageous. Further research should therefore allow for more complex TFs described by multiple parameters and placing fewer restraints on the values each parameter can take; such as in [2]. Additional further research could involve a combination of heterogeneous ANN where each neuron also has parameters to be optimised.

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 optimise the TFs employed by each neuron has been so far under utilised. This paper has demonstrated the use of two, non mutually exclusive, methods for allowing NE to optimise each neuron's TF. That is, selecting each neuron's TF from a predetermined list of TFs or by optimising parameters associated with each neuron. This 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 significance of the results presented in this paper are heightened by the fact that all NE methods are probably compatible with the two methods described. That is many NE method could benefit from evolving heterogeneous ANNs.

REFERENCES

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

[2] Marijke F Augusteijn and Thomas P Harrington, 'Evolving transfer functions for artificial neural networks', *Neural Computing & Applications*, **13**(1), 38–46, (2004).

[3] Stephan K Chalup and Lukasz Wiklendt, 'Variations of the two-spiral task', *Connection Science*, **19**(2), 183–199, (2007).

[4] D.T. Cliff, I. Harvey, and P. Husbands, 'Incremental evolution of neural network architectures for adaptive behaviour', in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'93)*, pp. 39–44, (1992).

[5] Włodzisław Duch and Norbert Jankowski, 'Survey of neural transfer functions', *Neural Computing Surveys*, **2**(1), 163–212, (1999).

[6] Wlodzislaw Duch and Norbert Jankowski, 'Transfer functions: hidden possibilities for better neural networks.', in *ESANN*, pp. 81–94, (2001).

[7] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, 'Multilayer feedforward networks are universal approximators', *Neural networks*, **2**(5), 359–366, (1989).

[8] Maryam Mahsal Khan, Masood Arbab Ahmad, Muhammad Gul Khan, and Julian F Miller, 'Fast learning neural networks using Cartesian Genetic Programming', *Neurocomputing*, **121**, 274–289, (2013).

[9] 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)*, pp. 619–626, (2010).

[10] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, 'Exploring strategies for training deep neural networks', *The Journal of Machine Learning Research*, **10**, 1–40, (2009).

[11] Yong Liu and Xin Yao, 'Evolutionary design of artificial neural networks with different nodes', in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pp. 670–675. IEEE, (1996).

[12] OL Mangasarian, R. Setiono, and WH Wolberg, 'Pattern recognition via linear programming: Theory and application to medical diagnosis', *Large-scale numerical optimization*, 22–31, (1990).

[13] Timmy Manning and Paul 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*, 165–176, Springer, (2013).

[14] W.S. McCulloch and W. Pitts, 'A logical calculus of the ideas immanent in nervous activity', *Bulletin of mathematical biology*, **5**(4), 115–133, (1943).

[15] J.F Miller, 'What bloat? Cartesian genetic programming on Boolean problems', in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 295–302, (2001).

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

[17] J.F Miller and P Thomson, 'Cartesian genetic programming', in *Proceedings of the Third European Conference on Genetic Programming (EuroGP)*, volume 1820, pp. 121–132. Springer-Verlag, (2000).

[18] *Cartesian Genetic Programming*, ed., Julian F. Miller, Springer, 2011.

[19] David Montana, Eric VanWyk, Marshall Brinn, Joshua Montana, and Stephen Milligan, 'Evolution of internal dynamics for neural network nodes', *Evolutionary Intelligence*, **1**(4), 233–251, (2009).

[20] Jooyoung Park and Irwin W Sandberg, 'Universal approximation using radial-basis-function networks', *Neural computation*, **3**(2), 246–257, (1991).

[21] Riccardo Poli, 'Parallel distributed genetic programming', *New Ideas in Optimization, Advanced Topics in Computer Science*, 403–431, (1999).

[22] 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).

[23] Belew Richard, K, Mcinerney John, and Schraudolph Nico, l N, 'Evolving networks: Using the genetic algorithm with connectionist learning', Technical report, Cognitive Computer Science Research group, Computer Science and Engr. Dept (C-014), Univ. California at San Diego, (1990).

[24] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams, 'Learning representations by back-propagating errors', *Nature*, **323**(6088), 533–536, (1986).

[25] Sara Silva and Ernesto Costa, '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).

[26] P. Smolensky, *Parallel distributed processing: explorations in the microstructure of cognition*, chapter Information processing in dynamical systems: foundations of harmony theory, 194–281, MIT Press, 1986.

[27] K.O. Stanley and R. Miikkulainen, 'Evolving neural networks through augmenting topologies', *Evolutionary computation*, **10**(2), 99–127, (2002).

[28] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, et al., 'The monk's problems a performance comparison of different learning algorithms', Technical report, Carnegie Mellon University, (1991).

[29] Andrew James Turner and Julian Francis Miller, 'Cartesian Genetic Programming encoded Artificial Neural Networks: A Comparison using Three Benchmarks', in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-13)*, pp. 1005–1012, (2013).

[30] Andrew James Turner and Julian Francis Miller, '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*, 213–226, Springer, (2013).

[31] Andrew James Turner and Julian Francis Miller, 'Cartesian Genetic Programming: Why No Bloat?', in *Genetic Programming: 17th European Conference, EuroGP-2014*, (2014). To appear.

[32] András Vargha and Harold D Delaney, '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).

[33] 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*, pp. 252–263. Springer Verlag, (2000).

[34] Daniel Weingaertner, Victor K Tatai, Ricardo R Gudwin, and Fernando J Von Zuben, 'Hierarchical evolution of heterogeneous neural networks', in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pp. 1775–1780. IEEE, (2002).

[35] A.P. Wieland, 'Evolving neural network controllers for unstable systems', in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pp. 667–673. IEEE, (1991).

[36] David H Wolpert and William G Macready, 'No free lunch theorems for optimization', *Evolutionary Computation, IEEE Transactions on*, **1**(1), 67–82, (1997).

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

[38] Tina Yu and Julian Fancis Miller, 'Neutrality and the evolvability of boolean function landscape', in *Genetic programming*, pp. 204–217. Springer, (2001).