Structural and Parametric Evolution of Continuous-time Recurrent Neural Networks

Cesar Gomes Miguel
Cognitive Sciences Research Group
Department of Electronic Engineering
University of São Paulo
cesargm@lsi.usp.br

Carolina Feher da Silva
Department of Physiology and Biophysics
Institute of Biomedical Sciences
University of São Paulo
carolina@icb.usp.br

Marcio Lobo Netto
Cognitive Sciences Research Group
Department of Electronic Engineering
University of São Paulo
lobonett@lsi.usp.br

Abstract

Neuroevolution comprehends the class of methods responsible for evolving neural network topologies and weights by means of evolutionary algorithms. Despite their good performance in several control tasks, most of these methods use variations of simple sigmoidal neurons. Recent investigations have shown the potential applicability of more realistic neuron models [7], opening new perspectives for the next generation of neuroevolutionary methods.

This work aims to extend a recent method known as NEAT *to evolve continuous-time recurrent neural networks (CTRNNs). The proposed model is compared with previous methods on a control benchmark test. Preliminary results reveal some advantages when evolving general CTRNNs over traditional models.*

1 Introduction

It is a well known fact that network topology, besides synaptic weights, is an important aspect of how an artificial neural network will learn to accomplish a given task [11, 20]. Although the term *learning* in the context of artificial neural networks (ANNs) often refers to the change of weights (parametric learning), optimizing topology (structural learning) is also desirable in more complex domains. But as the search space increases, so does the challenge to find an optimal architecture. A possible solution is to rely on stochastic searching methods such as genetic algorithms [5], which don't guarantee optimality, but fit well as a general technique when little or no information is known *a priori*.

The class of models concerned with evolving arbitrary topologies and weights for ANNs is known as neuroevolution [21, 18]. Despite the hard mathematical nature of the problem, those models are of major interest in fields such as adaptive behavior, artificial life, and embodied cognitive sciences due to their biological plausibility [2, 16].

The majority of neuroevolutionary methods are based on some variations of the simple sigmoidal neuron. Although these models are fine for general purpose applications or simulations, it has been shown in recent investigations the potential applicability of more biologically realistic models [7]. In particular, an interesting dynamical model that has gained more attention lately is the *Continuous-time Recurrent Neural Network [2]* (or simply CTRNN). It has been successfully used in evolutionary robotics, artificial life and general control tasks [3, 4, 6].

This paper extends a recent neuroevolutionary method known as NEAT (*NeuroEvolution of Augmenting Topologies* [18]) to evolve CTRNNs with arbitrary topology. The proposed model is compared to previous works in a double-pole balancing benchmark experiment.

The paper is organized as follows. Section 2 describes the Continuous-time Recurrent Neural Networks, which will be used to extend the NeuroEvolution of Augmenting Topologies method, reviewed in section 3. In section 4 the methods used to evaluate the proposed extended model are presented and the results follow in section 5. The last section elaborate on some final considerations.

2 Dynamical Neurons

Continuous-time Recurrent Neural Networks (CTRNNs) belong to a general class of dynamical neuron models known as leaky-integrators [1] [12, 2]. They provide an intermediate step between sigmoidal and spiking neurons (the second and third generation of models, respectively [14]). While retaining most of the implementation effectiveness from the previous models, they share to some extent important aspects of spiking neurons: input integration over time, variable internal state (an analogy to membrane potential), and the ability to dynamically change their state even in the absence of external inputs. The behavior of a CTRNN is governed by a system of differential equations of the following form:

$$\tau_i \frac{dy}{dt} = -y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j - \theta_j) + \sum_{k=1}^{S} S_{ki} I_k$$

where N is the total number of neurons, y_i is the internal state of neuron i, τ_i is a time constant (decay rate), w_{ji} is the connection strength between neurons i and j, S is the number of sensory inputs, I_k the reading from sensor k and S_{ki} is the connection strength from sensor k to neuron i. The output from the neuron is given by a standard logistic activation function (σ) and a bias (θ_i) term.

As with traditional ANNs, the bias controls how sensitive the neuron is to incoming connections from other neurons or sensors and the time constant defines how fast a neuron changes its internal activation state. For a general review on CTRNNs, see [1, 3].

3 Evolving Artificial Neural Networks

Most simulations embracing the use of neural networks as controllers are restricted to fixed topologies and the only characteristic that distinguishes them in a population of candidate solutions is their synaptic weights. These are usually represented by a vector of floating-point numbers that can be easily used as a chromosome encoding that single organism. However, the topology restriction implies that the number of possible behaviors is limited to a subset that may not contain the desired one. It is also unrealistic from the biological point of view since DNA mutations have been altering the complexity of the nervous system in the evolution of many species. A recent method known as NEAT [18] (*NeuroEvolution of Augmented Topologies*) was able to surpass this limitation using a sophisticated genetic algorithm that works with chromosomes of variable length and allows us to evolve complex neural networks with arbitrary topology.

[1] In analogy to electrical circuits: the neuron activation is equivalent to the voltage difference of a capacitor that gradually "leaks" a small amount of energy over time [6, 2]

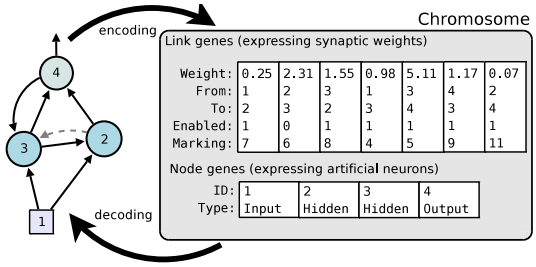

Figure 1. An example of chromosome encoding a neural network in NEAT.

Basically, the method uses a special and general neural network encoding technique (Figure 1) in which it is possible to apply mutation and crossover genetic operators without the problem of competing conventions [18, 21], which plagues most neuroevolution algorithms with its destructive effects on the parent's offspring. In NEAT, each link gene has a special identification called *innovation number*. This number represents a historical marking through generations, uniquely identifying each connection. The process of adding new genes (which can be a link or node gene) through mutations (and thus augmenting the neural network) is called complexification (Figure 2). Whenever a new connection gene is added to the chromosome, the algorithm checks for previous occurrences of the same mutation. If it has happened before, the same innovation number is assigned to that gene, else a new incremented innovation number is allocated and used instead. To perform crossover between chromosomes, NEAT pairs genes with the same innovation number, which are likely to play similar roles because they are homologous. This biologically inspired strategy makes crossover between chromosomes that represent different topologies meaningful and minimizes the risk of destroying existing structures.

3.1 Speciation

With the addition of new structures, it is very likely that the offspring will have a lower fitness value than its parents and would probably be taken out of the population by selection. A speciation method is applied to ensure that new innovations can be retained in a separate species and given enough time to adapt before competing with others for resources. With the help of historical markings, it is possible do determine the distance (how close they are based on the innovation numbers) among different chromosomes. A linear combination of excess (E) and disjoint (D) genes plus the average weight differences for matching genes ($\bar{W}$) are used to measure the distance from

individual i to j:

$$d(i,j) = \frac{1}{N}(c_1 E + c_2 D) + c_3 \bar{W} \quad (1)$$

where N is the size of the largest chromosome in the population. If $d(i,j) < \delta_t$ the individuals i and j belong to the same species. The threshold (δ_t) is usually obtained empirically (as most parameters in a genetic algorithm), but it is also possible to use a dynamic threshold that slightly changes its value at the end of each generation if the number of species is too high or too low.

NEAT uses *explicit fitness sharing* as the reproduction mechanism [5]. Individuals from the same species share their fitness, which stops a species from taking over the entire population even if most of its individuals have a high fitness value. Each individual's fitness is adjusted according to:

$$f_i^{'} = \frac{f_i}{\sum_{j=1}^{n} sh(d(i,j))} \quad (2)$$

where the *sharing function* (sh) is set to zero if the distance $d(i,j)$ is above δ_t or to one, otherwise. Note that $\sum_{j=1}^{n} sh(d(i,j))$ is reduced to the number of individuals the species has due to the distance function applied (Equation 1). This process helps maintaining the diversity of the population, reducing (but not eliminating) the chances of being trapped in a local minimum.

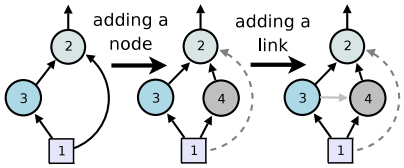

Figure 2. Mutation leads to complexification.

The adjusted fitness is used to assign a different number of offspring for every species. Let $\bar{F}_k$ be the average fitness of species k and $|P|$ the population's size. The spawn amount assigned to each species (k) is given by:

$$n_k = \frac{\bar{F}_k}{\bar{F}_{total}} |P| \quad (3)$$

where $\bar{F}_{total} = \sum_k \bar{F}_k$ is the total average fitness of all species. Each species's best member is kept for the next generation, while the lowest performing individuals are eliminated (selecting 20% of the species' best individuals for reproduction is often a good choice for retaining the diversity).

To avoid exploring a high dimensional search space at the beginning, all the chromosomes are created encoding a minimal network topology, i.e., the inputs are directly connected to the outputs.

3.2 Evolving CTRNNs with NEAT

Given the modular nature of a genetic algorithm, extending NEAT to evolve CTRNNs is straightforward (hereafter referred to as NEAT-CTRNN). Now each node gene encodes a dynamical neuron that adds a time constant (leaky rate) as a new attribute. At each time step the neuron's internal state changes according to its governing differential equation. The "leaky amount" is controlled by the time constant (τ) and represents the decay rate in absence of inputs or self-connections (eventually exhibiting no activation as $t \rightarrow \infty$). All the network's neurons are updated simultaneously at each time step and no constraint is placed upon the possible resulting topology.

4 Methods

The double-pole balancing problem (also known as cart-pole or inverted pendulum) has been traditionally used as a benchmark among different methods. In order to compare our results with previous approaches, we describe the problem following ideas from [19, 18].

The problem consists of balancing two poles connected to a cart applying a force to push the cart to the left or right (Figure 3). The system can be described by a set of non-linear coupled differential equations:

$$\begin{cases} \ddot{x} &= \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=1}^{N} \bar{F}_i}{M + \sum_{i=1}^{N} \bar{m}_i} \\ \ddot{\theta}_i &= -\frac{3}{4l_i}(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{p_i}\dot{\theta}_i}{m_i l_i}) \end{cases} \quad (4)$$

where N is the number of poles used, $\bar{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i (\frac{\mu_{p_i}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i)$ and $\bar{m}_i = m_i(1 - \frac{3}{4}\cos^2\theta_i)$ are respectively the effective force and mass of the i^{th} pole on the cart. Table 1 shows the meaning of each variable and the respective values used for the simulation. A simplified version of the task is also possible where one needs to balance a single pole, but it has been reported to be easily solved by most recent neuroevolution methods, posing no challenges for a real benchmark [18].

There are two types of experiments for the double-pole problem and we focus on the most difficult one, where a neural network must be able to balance the poles without velocity information (hereafter referred to as the DPNV problem). The DPNV is a non-markovian problem since a valid solution requires the use of short-term memory to compute the missing velocity information.

The simulation works as follows. Given an initial state (the system's variables at $t = 0$), at each time step only a partial state of the cart-pole system is fed to the network: the cart's position (x_t), the angle from vertical of pole one (θ_1), and pole two (θ_2). These inputs are normalized into

Symbol	Description	Value
x	cart's position	[-2.4, 2.4] m
θ_i	i^{th} angle pole from vertical	[-36, 36] °
F	force applied to the cart	[-10, 10] N
l_i	length of the i^{th} pole	0.5 and 0.05 m
M	mass of the cart	1 kg
m_i	mass of the i^{th} pole	0.1 and 0.01 kg
μ_c	cart's coef. of friction	5×10^{-4}
μ_p	pole's coef. of friction	2×10^{-6}

Table 1. Cart-pole's parameters.

$[-1, 1]$ using $(\frac{x}{4.8}, \frac{\theta_1}{0.52}, \frac{\theta_2}{0.52})$. A force proportional to the activation of the output neuron is applied to the cart.

The poles were considered to be balanced if the system's state variables remained within the following interval: $-2.4 < x < 2.4$, $-36° < \theta_1 < 36°$ and $-36° < \theta_2 < 36°$.

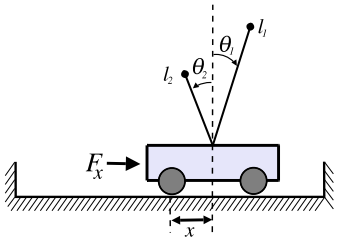

Figure 3. The cart-pole problem.

Following the previous experiments, the initial cart-pole state is set to: $x = \theta_1 = \dot{\theta}_1 = \dot{\theta}_2 = 0$, and $\theta_2 = 7°$. The system's differential equations as well as the CTRNNs were integrated using a 4th-order Runge-Kutta method with a time step of 0.01.

4.1 Implementation

Although NEAT has several open-source implementations in different programming languages, for the purposes of this paper we have used a recent Python implementation that simplifies the work for writing general experiments [2]. Small portions of the code for the DPNV experiment were wrapped from the original C++ implementation released by the author of NEAT [3]. This assures that the experiments were conducted minimizing the differences.

4.2 Genetic Algorithm

For all experiments a population of 150 individuals were used. A species was considered to be stagnated if its average fitness showed no improvement over the last 15 generations. There was no constraint limiting the number of possible species and the distance threshold was fixed at $\delta = 3.5$. Table 2 shows the remaining parameters. For each species, individuals were selected to reproduce using a tournament of size 2. All chromosomes' node genes in the first generation begin with no bias (*set to zero*) and a response value of 4.92 (obtained experimentally in [17]). The connection genes (encoding synaptic weights) are randomly assigned using a Gaussian distribution with zero mean and standard deviation of 1.0.

[2]NEAT Python: http://code.google.com/p/neat-python
[3]NEAT C++: http://www.cs.ucf.edu/~kstanley

Parameter	Value
Mutation probabilities	
Add node gene	0.001
Add connection gene	0.03
Bias mutation	0.2
Weight mutation	0.8
Toggle connection	0.05
Other parameters	
Excess coefficient (c_1)	1.0
Disjoint coefficient (c_2)	1.0
Weight coefficient (c_3)	2.0

Table 2. Main NEAT parameters.

4.3 Fitness Evaluation

The DPNV experiment requires a special fitness function to penalize oscillations, that is, a trivial solution that would evolve to move the cart back and forth quickly enough, keeping the poles wiggling in the air [18]. As proposed by Gruau [10], the fitness is given by $f = 0.1f_1 + 0.9f_2$, and the components are computed as follows:

$$f_1 = t/1000$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}\left(|x^i|+|\dot{x}^i|+|\theta_1^i|+|\dot{\theta_1}^i|\right)} & \text{otherwise} \end{cases}$$

where t is the number of time steps the poles remain balanced during 1000 total time steps. The second component (f_2) is maximized when the cart and long pole positions and velocities are minimized during the last 100 time steps.

At the end of each generation, the best performing network is evaluated again in a separate generalization test. The fitness will not be computed again–the purpose is to test the robustness and evaluate how well the best network performs when given a longer simulated time and different initial conditions. First, the network must balance the poles for at least 100000 time steps. If it succeeds, then the network has to balance the poles for 1000 time steps for 625 different initial conditions for the cart-pole equations (Eq. 4). Each of the following variables (x, $\dot{x}$, θ_1, $\dot{\theta}_1$) are

given the values $0.05, 0.25, 0.5, 0.75, 0.95$, thus resulting in $5^4 = 625$ different initial states.

To count as a solution, the network must be able to balance the poles for at least 200 times out of 625. This is the *generalization score* (GS). Networks with higher GS generalize better. If the network fails to solve the generalization task, the genetic algorithm moves on to the next generation.

5 Results

To evaluate NEAT-CTRNN, we compared its efficiency in solving the DPNV with other methods and NEAT itself. Table 3 shows the results (some data was not available or provided by the authors). The results for CE (Cellular Encoding [10]), CNE (Conventional NeuroEvolution [9]), and ESP (Enforced Subpopulations [8]) were reported by [8] and AGE by [4]. As for NEAT, the results were obtained using the original implementation.

Method	Evaluations	SE	GS	SE
CE	840000	–	300	–
CNE	87623	–	–	–
ESP	26342	–	–	–
AGE	25065	4360	317	–
NEAT	23777	718	257	1.95
NEAT-CTRNN	4048	105	274	2.11

Table 3. Results for the DPNV problem. SE stands for the standard error of the mean.

The NEAT-CTRNN version was able to find a solution in 4048 ± 205 evaluations on average (95% confidence interval). This is significantly better than NEAT, which found a solution in 23777 ± 1408 evaluations ($p < 0.001$). The generalization score (GS) for NEAT-CTRNN is slightly better than NEAT's ($p < 0.001$). The statistical analysis was performed under the assumption that the data follows a normal distribution, which is more common in literature, or a gamma distribution, which fits the data a lot better than the normal distribution.

Comparing NEAT-CTRNN to CE, CNE, ESP, and AGE statistically is hard because the standard deviation for most results weren't reported by [8] and [4], but the standard error for the number of evaluations reported for AGE strongly suggests NEAT-CTRNN performs better than AGE. Assuming that CE, CNE, and ESP aren't totally unreliable and don't have an extremely high standard deviation, it's reasonable to assume that NEAT-CTRNN also performs better than them. Results reported by [4] led to the conclusion that AGE was able to found a solution with fewer evaluations than NEAT. Although the work was published in 2006, they used old and outdated results from 2002 [18], when NEAT was first compared to previous methods. More recent results due to Gomez [8] showed that NEAT and AGE actually performs similarly and our own results confirm this.

Figure 4 shows a solution evolved by NEAT-CTRNN (bias connection not shown). Although not reported in previous works, in our experiments the best solution from each run had on average 0.25 ± 0.04 hidden nodes and 4.77 ± 0.14 enabled connections, averaged over 544 runs (note that in three runs out of four the minimal topology was found). NEAT-CTRNN did not fail to find a solution in all runs.

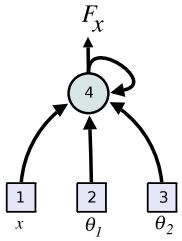

Figure 4. A minimal evolved solution.

In order to solve the DPNV problem, at least one recurrent connection is required. In our experiments, the initial population had only three inputs directly connected to the output. As soon as a mutation added a recurrent connection in the output node, this evolutionary advantage spread through the entire population in the following generations. Then it was just a matter of fine-tuning the weigths to find a solution.

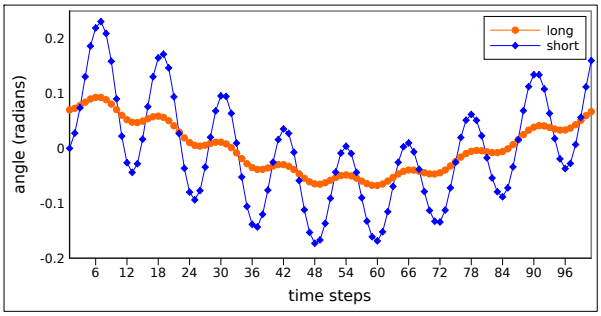

Figure 5. Angle variation for both poles.

The variation of the angle for both poles is shown in Figure 5. When the initial conditions are all set to zero except for $\theta_2 = 7°$ (short pole angle), the network is capable of keeping the poles balanced within a shorter range then previously required.

6 Final Considerations

With the ability to integrate input over time and continuously change its internal state (as opposed to traditional ANNs where activation works much like a clock in discrete steps), CTRNNs were able to control the behavior of a dynamical system with a smaller network, requiring less neurons and connections. This explains the lower number of evaluations when compared to prior attempts, since the genetic algorithm has to work on a smaller dimensional space.

Also as noted by [4], a particular issue with the DPNV is that of being frequently trapped in a local minimum. Higher fitness does not mean better generalization and while the genetic algorithm selects the best chromosomes, they may fail the generalization test while lower fitted individuals might get a better score. Since NEAT always keeps the member of each species with the highest fitness (even when stagnated), it is likely that most of the population will be trapped. A possible and simple solution is to avoid elitism.

The performance of the CE method (Cellular Encoding) was only reported in order to show how efficiently the methods have been evolving over time. Given the fact that NEAT-CTRNN was able to find a minimal solution most of the time and in fewer evaluations, the DPNV problem needs to be replaced by more challenging benchmarks.

Neuroevolution can be seen as a general class of machine learning algorithms and require little or no previous knowledge of the problem or solution. As with NEAT, NEAT-CTRNN can be further optimized if one considers more sophisticated learning methods such CMA-ES [13] or Truncated-Newton [15] to accurately search the weight space, while keeping a standard GA to search the space of network topologies. It is also worth mentioning that solving in fewer evaluations does not necessarily mean that method X outperforms Y in all possible tasks. At least in one simple set of problems we could show this, but the field of neuroevolution requires more theoretical and analytic results regarding the space of input parameters [1].

References

[1] R. D. Beer. Parameter space structure of continuous-time recurrent neural networks. *Neural Computation*, 18(12):3009–3051, 2006.

[2] R. D. Beer and J. C. Gallagher. Evolving dynamic neural networks for adaptive behaviour. *Adaptive Behaviour*, 1(1):91–122, 1992.

[3] J. Blynel and D. Floreano. Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *Proceedings of the seventh international conference on simulation of adaptive behavior on From animals to animats*, pages 272–281, Cambridge, MA, USA, 2002. MIT Press.

[4] P. Dürr, C. Mattiussi, and D. Floreano. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 9th Conference on Parallel Problem Solving from Nature*, volume 9, pages 671–680, Berlin, Germany, 2006. Springer-Verlag.

[5] G. D. E. and J. Richardson. Genetic Algorithms With Sharing for Multimodal Function Optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.

[6] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[7] R. V. Florian. Biologically inspired neural networks for the control of embodied agents. Technical report, Center for Cognitive and Neural Studies (Coneural), 2003.

[8] F. J. Gomez. *Robust non-linear control through neuroevolution*. PhD thesis, The University of Texas at Austin, Texas, USA, 2003. Department of Computer Sciences.

[9] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 16, page 1356–1361, 1999.

[10] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the GA*. PhD thesis, l'Ecole Normale Superieure de Lyon, 1994.

[11] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.

[12] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[13] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15:1–28, 2007.

[14] W. Maas. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[15] S. G. Nash. A survey of truncated-newton methods. *Journal of Computational and Applied Mathematics*, 124:45–59, 2000.

[16] R. Pfeifer and C. Scheier. *Understanding Intelligence*. MIT Press, Cambridge, Massachusetts, 1999.

[17] K. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, The University Of Texas At Austin, Department Of Computer Sciences, Texas, USA, August 2004.

[18] K. Stanley and R. Miikkulaien. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[19] A. P. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 667–673, Seattle, WA, USA, 1991. IEEE.

[20] A. Yamazaki, T. Ludermir, and M. de Souto. Global optimization methods for designing and training neural networks. In *Proceedings of the 7th Brazilian Symposium on Neural Networks*, pages 136–141, Washington, DC, USA, 2002. IEEE Computer Society.

[21] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:203–222, 1993.