Neuroevolution with Analog Genetic Encoding

Peter Dürr, Claudio Mattiussi, and Dario Floreano

Laboratory of Intelligent Systems, Institute of Systems Engineering, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland
http://lis.epfl.ch

Abstract. The evolution of artificial neural networks (ANNs) is often used to tackle difficult control problems. There are different approaches to the encoding of neural networks in artificial genomes. Analog Genetic Encoding (AGE) is a new implicit method derived from the observation of biological genetic regulatory networks. This paper shows how AGE can be used to simultaneously evolve the topology and the weights of ANNs for complex control systems. AGE is applied to a standard benchmark problem and we show that its performance is equivalent or superior to some of the most powerful algorithms for neuroevolution in the literature.

1 Introduction

The use of artificial neural networks (ANNs) in control systems is a widely covered research topic. The complexity of control tasks often makes it difficult to design ANNs manually. Therefore, it is a common approach to use evolutionary algorithms for this kind of problem. Not only the synaptic weights, but also the structure of the network can be subject to neuroevolution. Thus one of the challenges is to find an appropriate genotype-phenotype mapping for both the topology and the weights. In the literature we find different methods to accomplish this. The most straightforward approach is *direct encoding* (e.g. used by [1,2,3,4]) where the genome is composed of a list of genes, each representing either a neuron or a link between two neurons. Genomes of this type can be decoded very easily, but their length grows rapidly with increasing complexity of the network. Another popular approach is *developmental encoding*, as shown in [5,6,7,8,9], which is based on the use of a genome that directs a developmental process leading to the construction of the network. This allows a more compact representation of complex networks, but the developmental process, linking the genome to the developed network, typically makes it difficult to find suitable genetic operators. Somewhat different is the *implicit encoding*. Derived from the observation of biological genetic regulatory networks (GRNs) (see [10] for more details), implicit encoding is a very interesting approach, which is quite popular as a representation for GRNs [11] but is not very commonly used on ANNs.

Analog Genetic Encoding (AGE) [10,12] is an implicit method, which - so far - has only been applied to very simple problems of neuroevolution. The goal of this paper is to show that it is possible to solve more complex problems using AGE and that it outperforms other established methods. The double pole balancing

T.P. Runarsson et al. (Eds.): PPSN IX, LNCS 4193, pp. 671–680, 2006.

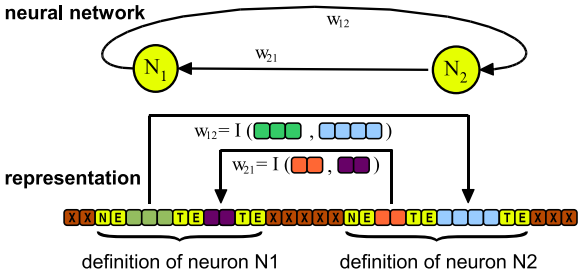

Fig. 1. AGE provides a solution to the encoding of networks in digital genomes. The genome contains genes encoding the devices that form the network.

without velocity information, which has been used as a standard problem in various publications (e.g. [13,14,15]), has been selected as benchmark. The results allow a direct comparison to the above-mentioned methods, thus showing the high performance of AGE, quantitatively and qualitatively.

2 Analog Genetic Encoding (AGE)

2.1 The Challenge

The evolution of an artificial neural network requires the encoding of the network in a genome. In the general case, an arbitrary network of hidden neurons, connected to a given number of fixed input and output neurons has to be evolved. The analog genetic encoding as presented in [10,12] provides a very plausible approach to this problem. The basic idea of AGE is to define a genetic representation that allows the interpretation illustrated in Figure 1.

2.2 Device Representation

The genome is constituted by a sequence of characters from a finite genetic alphabet. Here, the 26 characters of the ASCII uppercase alphabet are used (see [10] for justification). The experimenter defines the kind of devices that can appear in the network. (Here a single type of dynamic neuron is used.) For each device type, a specific *device token* has to be defined. Each device token signals the start of a gene, that is a fragment of the genome which encodes an instance of the corresponding neuron. Furthermore, a *terminal token* is defined, whose role is to delimit the sequence of characters that must be associated with a terminal of the corresponding device. These are the so-called *terminal sequences*. A neuron is hence encoded by a device token, followed by a number of terminal sequences, each delimited by a terminal token[1].

[1] Tokens are typically short sequences of letters. The tokens used in the experiment can be found in Figure 2 (hidden neurons) and Figure 3 (input and output neurons).

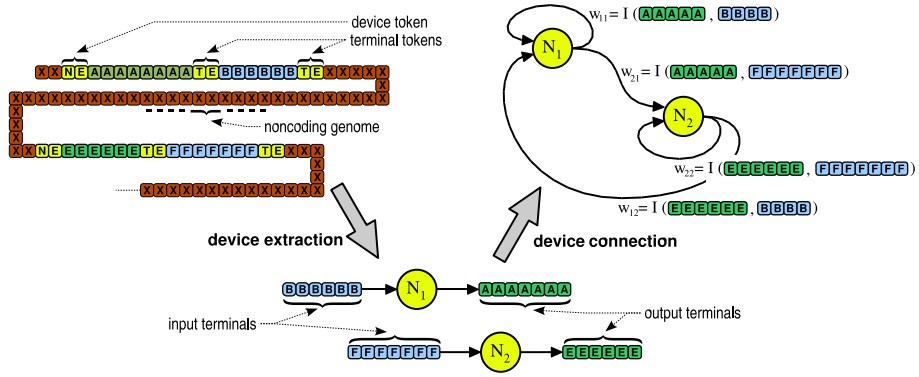

Fig. 2. Neurons can be represented as symbolic devices with two associated terminal sequences: one for the output terminal and one for the input terminal. The device extraction process obtains them from the genome by assigning the sequences of characters between the device token ("NE") and the terminal tokens ("TE") to the respective terminal. The terminal sequences of the different neurons are then used to determine the synaptic weights of the network. The interaction map $I(s_1, s_2)$ assigns a weight to a pair of sequences, so that we can for example calculate $w_{11} = I(s_{11}, s_{12})$. The entire weight matrix can be calculated by doing this for all pairs of terminal sequences in the network.

2.3 Device Extraction

Given a list of device tokens and a terminal token, a list of devices can be decoded from a genome by simply extracting the devices one by one. To this end, the genome is scanned in search of device tokens and if one is found, the fragment of genome following the token is scanned for the necessary terminal tokens. Then the sequences of characters between the device token and the terminal token (or between two terminal tokens respectively) are assigned to the corresponding terminal of the neuron. If a device token in the genome is not followed by the required number of terminal sequences, the gene is considered invalid and the decoding continues with the next device token in the genome.

2.4 Device Connection

To determine the synaptic weights between the neurons, a so-called *interaction map* $I(s_1, s_2) = N(L(s_1, s_2))$ is needed. This function assigns a synaptic weight value to any given pair of terminal sequences. The inner function $L(s_1, s_2)$ is called *sequence interaction map* and returns a distinct interaction score i for each pair of terminal sequences s_1 and s_2. For these experiments, the value of the sequence interaction map corresponds to the value of the local alignment score[2] between the two sequences s_1 and s_2 (see [16]). The parameters of the

[2] The local alignment score is a function which operates on pairs of sequences of arbitrary length and has some very desirable properties from an evolutionary point of view, which are more extensively discussed in [10,16].

local alignment function (the circulant *substitution matrix* and the *indels vector*) used here are:

	A	B	C	D	E	F	G	...	U	V	W	X	Y	Z
A	5	2	1	0	-1	-2	-5	...	-5	-2	-1	0	1	2
B	2	5	2	1	0	-1	-2	...	-5	-5	-2	-1	0	1

⋮

and

	A	...	Z
-	-3	...	-3

The *network-specific interaction map* $N(i)$ transforms the (integer) sequence interaction values to the (floating point) values of the synaptic weight between the terminals. Here a logarithmic quantization $N : [1, 37] \longrightarrow [0.001, 1000]$ was used, mapping interaction scores from $i_{min} = 1$ to $i_{max} = 37$ to weights in an interval from $w_{min_P} = 0.001$ to $w_{max} = 1000$. Interaction scores lower than i_{min} lead to a weight $w_{min} = 0$, scores above i_{max} lead to a weight of w_{max}.

We can calculate the whole weight matrix of the network by applying the interaction map to all pairs of input and output terminals. Since no inhibitory neurons are used, the two outputs of a neuron provide a positive and negative output of its state. If there are n neurons, the entries w_{ij} of the weight matrix W are accordingly defined as

$$w_{ij} = N(L(i_{s_{input}}, j_{s_{output+}})) - N(L((i_{s_{input}}, j_{s_{output-}}))$$

for $i = [1, n]$ and $j = [1, n]$.

2.5 External Connections

Based on the same principle, it is very easy to incorporate the connections to the external input and output neurons. For each type of signal a separate, external neuron type with a distinct token is defined (see Figure 3). The connection weights from the external neurons to the hidden neurons can be calculated using the same method as above.

2.6 Genetic Operators

Artificial (and natural) evolution relies on the reorganization of the genome. Contrary to other methods, the AGE genome is very robust to such operations, since it is of variable length and there is no special protection needed to keep the genome decodable. There is actually no distinction between tokens, coding and non-coding regions of the genome. In the experiments, the following operators where used:

- *Character deletion, insertion, and substitution.* A character is removed, inserted or substituted in the genome.
- *Fragment deletion, transposition and duplication.* Two points of the genome are chosen and the intervening fragment is deleted, transferred or copied to another point of the genome.

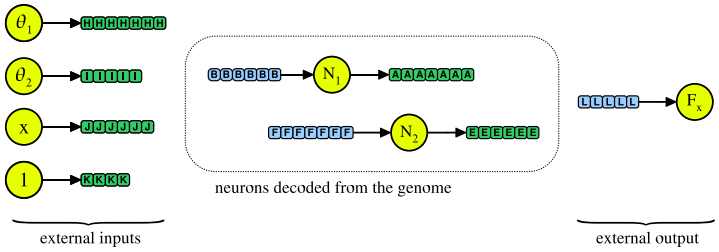

Fig. 3. The external input and output neurons are encoded as separate devices (exactly like the hidden neurons in Figure 2). For each sensor input, the motor output and a bias input, a device type with an associated token is used to decode the respective neurons from the genome. The tokens used for the external inputs are "AA", "AB", "PA" and "BB", the token for the output is "OA".

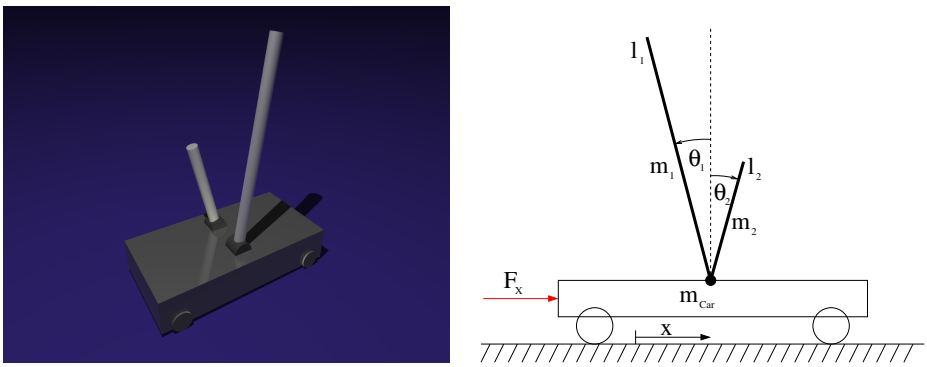

Fig. 4. The double pole balancing problem (DPNV). Two poles mounted on a car have to be balanced, using measurements of the pole angles and the position of the car.

- *Device insertion.* The descriptor of a device (e.g. a hidden neuron) is inserted in the genome. The terminal sequences are randomly generated.
- *Homologous Crossover.* Fragments of the genome are recombined using homologous crossover (see [10] for more details).
- *Genome duplication.* The whole genome is duplicated.
- *Generation of an initial population.* The initial population is created by generating individuals with a random genome and inserting a given number of different neurons with random terminal sequences.

3 Double Pole Balancing as a Benchmark Test

In order to compare the different approaches in neuroevolution on a practical rather than a purely theoretical level, a benchmark test is needed. The double pole balancing problem without velocity information (DPNV) is quite challenging

compared to the fairly simple single pole balancing problems[3], while it is still easy to understand and simple enough to be simulated without huge computational efforts. Stanley and Miikkulainen [15] compare the results of the only neuroevolution methods which have reportedly solved the DPNV problem by evolving topology and weights of neural networks: Gruaus *Cellular Encoding* (CE, [13]), Gomez and Miikkulainens *Enforced Sub Populations* (ESP, [14]), and Stanley and Miikkulainens *Augmenting Topologies* (NEAT, [15])[4].

3.1 The Controlled System

The double pole balancing setup (see Figure 4), consists of a car with mass $m_{Car} = 1[kg]$ and one degree of freedom x, on which two poles of different lengths $l_1 = 1[m]$ and $l_2 = 0.1[m]$ are mounted. The poles have the masses $m_1 = 1[kg]$ and $m_2 = 0.1[kg]$. Based on the measured values of the joint angles θ_1 and θ_2 and the position of the car x, the controller is required to balance both of the poles by applying a force F_x (with a maximal magnitude of $F_{max} = 10[N]$). Assuming rigid body dynamics and neglecting friction, the system can be described by the equations of motion as in [18]. The numerical simulation of the system is based on a 4th-order Runge-Kutta integration of these equations with a time step of $\Delta t = 0.01s$.

3.2 Fitness Assignment

In their original publication, Gruau, Whitley and Pyeatt [13] suggest the following approach for the assessment of candidate solutions. In order to avoid demanding calculations for every fitness evaluation, they split the definition of the fitness value from the decision to judge a solution successful by applying a simple fitness function to every individual in the population and an extensive test series on the best individual of the population. Although it saves a lot of computation time, this is very questionable, since it is not a priori clear that individuals with a high fitness will perform well in the extensive test. But since the benchmark data collected by [14] and [15] relies on this measure, the same approach is used here.

The Fitness Function. In order to assign a fitness value to an individual, a numerical simulation is carried out over a maximum of 1000 timesteps, starting from given initial conditions ($\theta_1(0) = 0.0785$, $\dot{\theta_1}(0) = \theta_2(0) = \dot{\theta_2}(0) = x(0) = \dot{x}(0) = 0$). For each timestep the position of the car and the pole angles are observed and the simulation continues only as long as they stay in a given range:

[3] They can typically be solved in a few generations with simple evolutionary algorithms, or with random search in the parameter space.

[4] In [17] Igel shows that it is possible to outperform these methods by using an evolution strategy (CMA-ES) to optimize the weights of a fixed topology ANN. But since in general evolution of both weights and topology is needed, his results are not really comparable to those presented here.

$$-\theta_1^{Max} \leq \theta_1 \leq \theta_1^{Max} \tag{1}$$
$$-\theta_2^{Max} \leq \theta_2 \leq \theta_2^{Max} \tag{2}$$
$$-x^{Max} \leq x \leq x^{Max} \tag{3}$$

where $\theta_1^{Max} = \theta_2^{Max} = 36°$ and $x^{Max} = 2.4[m]$. The fitness value F is defined as

$$F = 0.1f_1 + 0.9f_2 \qquad with \tag{4}$$
$$f_1 = \frac{t}{1000} \tag{5}$$
$$f_2 = \begin{cases} 0 & if \quad t < 100, \\ \frac{0.75}{\sum_{i=t-100}^{t}\left(|x^i|+|\dot{x}^i|+|\theta_1^i|+|\dot{\theta}_1^i|\right)} & otherwise. \end{cases} \tag{6}$$

where t is the number of time steps the system remains inside the boundaries (1), (2) and (3).

The Generalization Score. The best individual (i.e. the one with the highest fitness value F) of every generation is tested for its ability to balance the system for a longer time period. If a potential solution passes this test by keeping the system balanced for 100'000 timesteps, the so called *generalization score* (GS) of this particular individual is calculated. This score measures the potential of a controller to balance the system starting from different initial conditions. It is calculated with a series of experiments, running over 1000 timesteps, starting from 625 different initial conditions. The initial conditions are chosen by assigning each value of the set $\Omega = [0.05\ 0.25\ 0.75\ 0.95]$ to each of the states x, $\dot{x}$, θ_1 and $\dot{\theta}_1$, scaled to the range of the variables (as specified in the following section). The short pole angle θ_2 and its angular velocity $\dot{\theta}_2$ are set to zero. The GS is then defined as the number of successful runs from the 625 initial conditions and an individual is defined as a solution if it reaches a generalization score of 200 or more.

3.3 The Artificial Neural Network

Neuron Model. The neurons used here are simple continuous time recurrent neurons as in [19]. The time constant is set to $\tau = 0.001[s]$. The resulting network state equation

$$\tau\dot{\mathbf{y}} = -\mathbf{y} + \mathbf{W}\sigma\left(\mathbf{y} + \theta\right) + \mathbf{I} \tag{7}$$
$$where \qquad \sigma(x) = \frac{1}{1+e^{-x}} \tag{8}$$

is integrated with a separate embedded Runge-Kutta-Fehlberg (4,5) method. For this benchmark problem, the bias vector θ is set to zero and an external input with the constant output of 1.0 is connected to the network. To match the conditions of the original experiment [13], the input neurons are fed with scaled measurement values ($\theta_{1_{neur}} = \frac{\theta_1}{0.52}$, $\theta_{2_{neur}} = \frac{\theta_2}{0.52}$ and $x_{neur} = \frac{x}{4.8}$). The outputs of the motor output neuron ranging from -1 to 1 are scaled to forces from $-F_{max}$ to F_{max}.

3.4 Genetic Algorithm

The genetic algorithm used in the experiment is a standard generational GA with the AGE specific genetic operators as explained above and tournament selection. The mating pool size is 30 and there is an elite of size 1, thus the total population size is 31. The tournament size is set to 2. The probability of homologous recombination is 0.1 with 5 characters required to be similar for recombination to take place. The probabilities of nucleotide substitution, insertion and deletion are set to 0.001, the probabilities of fragment duplication, insertion and deletion to 0.01. Random devices are inserted with a probability of 0.01 with terminals of length 20.

In order to improve the performance of the algorithm, the GA is restarted whenever it gets stuck (i.e. when no improvement in the fitness of the best individual is observed after 15 generations). This choice was inspired by experiments with NEAT reported in [20], where subpopulations, which do not improve within 15 generations are removed. To avoid bootstrapping problems, the GA is initialized with a large initial population of 1000 individuals, each with a random genome of 500 to 800 characters. Each individual in the initial population gets a complete set of input, output and bias neurons plus one or two hidden neurons with randomly generated terminal sequences.

4 Results

Table 1 shows the results of AGE compared to the other methods, which have reportedly solved the DPNV so far. Both the number of fitness evaluations and the generalization score are about equal or better than the results obtained by NEAT. The average number of function evaluations needed by AGE is smaller than the best results previously reported in the literature. It seems that AGE is able to produce better solutions in a smaller number of generations. The example solution in Figure 5 (which obtained a GS of 525) shows that simple structures can obtain relatively high generalization scores. Initialized with only one or two hidden neurons, AGE tends to exploit these small structures and finds elegant solutions.

An odd property of the DPNV benchmark with the split fitness is that high fitness scores do not automatically lead to good generalization properties. In the

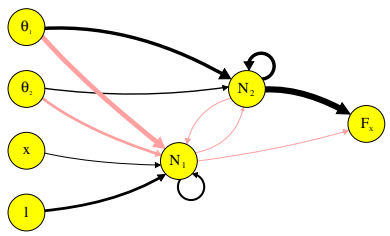

Fig. 5. An example neural network, found by AGE. Despite its simple structure, it generalizes really well (with a GS of 525).

Table 1. The results of the double pole balancing with no velocity information (DPNV). CE is cellular encoding [13], ESP is enforced subpopulations [14], NEAT is augmenting topologies [15]. All results are averaged over 20 evolutionary runs. AGE has to be restarted about 10 times on average to obtain a solution.

Method	Evaluations	Standard Deviation	Generalization
CE	840000	n.a.	300
ESP	169466	n.a.	289
NEAT	33184	21790	286
AGE	25065	19499	317

experiment, some populations with relatively high fitness of the best individual got stuck without producing a solution which could pass the long run test, whereas other populations with relatively low fitness could produce good solutions very quickly. The fact that the fitness function and the generalization test do not correlate well indicates that a better fitness function should be chosen for future benchmark experiments.

5 Conclusion

The results obtained in the standard benchmark double pole balancing problem with no velocity information show that it is possible to use analog genetic encoding to evolve neural networks for a difficult control task. They also indicate that AGE outperforms the best algorithms existing in the literature for the evolution of ANN topology and weights, producing compact, high quality solutions within a small number of fitness evaluations.

Acknowledgments. The implementation of the benchmark problem relies directly on source code by Kenneth O. Stanley. Thanks to Daniel Marbach for his collaboration on the implementation of AGE and a lot of inspiring discussions. Thanks to Sara Mitri for her valuable comments and further thanks to the anonymous reviewers for their helpful remarks.

References

1. Maniezzo V.: Genetic evolution of the topology and weight distribution of neural networks. IEEE Transactions on Neural Networks, vol. 5, no. 1 (1994) 39–53
2. Pujol J., Poli R.: Evolving the topology and the weights of neural networks using a dual representation. Applied Intelligence, vol. 8, no. 1 (1998) 73–84
3. Kobayashi K., Ohbayashi M.: A new indirect encoding method with variable length gene code to optimize neural network structures. Proceedings of the International Joint Conference on Neural Networks, vol. 6(1999) 4409–4412
4. Stanley K., Miikkulainen R.: Evolving neural networks through augmenting topologies. Evolutionary Computation, vol. 10, no. 2 (2002) 99–127

5. Cangelosi A., Parisi D., Nolfi S.: Cell division and migration in a genotype for neural networks. Network: Computation in Neural Systems, vol. 5, no. 4 (1994) 497–515
6. Gruau F.: Automatic definition of modular neural networks. Adaptive Behaviour, vol. 3, no. 2, (1995) 151–183
7. Nolfi S., Parisi D.: Genotypes for neural networks. The Handbook of Brain Theory and Neural Networks, M. Arbib, Ed. Cambridge, MA: MIT Press (1995) 431–434.
8. Eggenberger P.: Creation of neural networks based on developmental and evolutionary principles. Proceedings of the International Conference on Artificial Neural Networks, Lausanne, Switzerland (1997)
9. Astor J., Adami C.: A developmental model for the evolution of artificial neural networks. Artificial Life, vol. 6, no. 3 (2000) 189–218
10. Mattiussi, C.: Evolutionary synthesis of analog networks. Ph.D. dissertation n.3199, EPFL, Lausanne (2005)
11. Bongard, J.: Evolving modular genetic regulatory networks. Proceedings of the IEEE 2002 Congress on Evolutionary Computation, CEC2002. Piscataway, NJ: IEEE Press (2002) 1872–1877
12. Mattiussi, C., Floreano, D.: Evolution of analog networks using local string alignment on highly reorganizable genomes. Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware (2004) 30–37
13. Gruau,F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. Genetic Programming 1996: Proceedings of the First Annual Conference (1996) 81–89
14. Gomez, F. J., Miikkulainen, R.: Solving non-markovian control tasks with neuroevolution. Proceedings of the International Joint Conference on Artificial Intelligence (1999) 1356–1361
15. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2) (2002) 99–127
16. Gusfield, G.: Algorithms on strings, trees, and sequences. Cambridge: Cambridge University Press (1997).
17. Igel C.: Neuroevolution for reinforcement learning using evolution strategies. Congress on Evolutionary Computation 2003 (CEC 2003) 2588–2595
18. Wieland, A.P.: Evolving neural network controllers for unstable systems. Proceedings of the International Joint Conference on Neural Networks (1991) 667–673
19. Beer, R.D.: On the dynamics of small continuous-time recurrent neural networks. Adaptive Behavior 3(4) (1995) 469–509
20. Stanley K.O.: Efficient evolution of neural networks through complexification. Ph.D. dissertation, University of Texas at Austin (2004)