Self-Organisation of Neural Topologies by Evolutionary Reinforcement Learning

Nils T Siebel, Jochen Krause and Gerald Sommer
Cognitive Systems Group, Institute of Computer Science
Christian-Albrechts-University of Kiel
Olshausenstr. 40, 24098 Kiel, Germany
E-mail: nils@siebel-research.de, {jk,gs}@ks.informatik.uni-kiel.de

Keywords: Neural Networks, Evolutionary Algorithms, Reinforcement Learning

***Abstract*— In this article we present EANT, "Evolutionary Acquisition of Neural Topologies", a method that creates neural networks (NNs) by evolutionary reinforcement learning. The structure of NNs is developed using mutation operators, starting from a minimal structure. Their parameters are optimised using CMA-ES. EANT can create NNs that are very specialised; they achieve a very good performance while being relatively small. This can be seen in experiments where our method competes with a different one, called NEAT, "NeuroEvolution of Augmenting Topologies", to create networks that control a robot in a visual servoing scenario.

1 Introduction

As universal function approximators, artificial neural networks (NNs) are capable of modelling complex mappings between the inputs and outputs of a system up to an arbitrary precision [7, 11]. However, with an increase in complexity of a given task the required complexity of the NN also increases. Such a complex NN is difficult to develop due to the high dimensionality of the space in which its parameters live. This so-called "curse of dimensionality" has always been a significant obstacle in machine learning problems [2].

NNs are characterised by their *structure (topology)* and their *parameters* (which includes the weights of connections) [12]. A number of learning methods exist for generating them. Most of these methods, like the popular "backpropagation" algorithm [12, chap. 7], are methods to adjust the parameters of the network to a given problem, but not its structure. When using such methods the structure of the network has to be adjusted to the problem beforehand and "by hand", i.e. by the designer of the software. Once the structure is fixed, its parameters can be learned. Most of these learning methods can be viewed as a straightforward application of local optimisation algorithms and/or statistical parameter estimation. Backpropagation, for instance, is equivalent to optimisation by stochastic gradient descent [14, chap. 5]. These methods exhibit the following two problems:

(1) The common approach to pre-design the network structure can be difficult or even infeasible for complicated tasks. It may also result in overly complex networks if the designer cannot find a small structure that solves the task.
(2) Determining the network parameters by local optimisation algorithms like gradient descent-type methods is impracticable for large problems. It is known from mathematical optimisation theory that these algorithms tend to get stuck in local minima. They only work well for very simple (e.g., convex) problems or if an approximate solution is known beforehand.

We have previously developed a method, called EANT, "Evolutionary Acquisition of Neural Topologies", that automatically learns both the structure and the parameters of a NN to find a solution to a given problem [9, 13]. Both learning parts use evolutionary algorithms (EAs) [3], global optimisation methods that are less prone to get stuck in local minima. With these algorithms the NN is learned from scratch by reinforcement learning [16].

In this article we present recent improvements to EANT that further accelerate the generation of networks that perform well. In order to validate our method we present an experimental comparison of EANT and NEAT, a similar method.

The remainder of this article is organised as follows. Section 2 contains an overview over related methods for evolutionary NN learning and describes our approach to a solution. In Section 3 we formulate the visual servoing problem that is used for testing the NN learning methods. Section 4 contains results from experiments with EANT and NEAT; Section 5 concludes the article.

2 Methods for Evolutionary Learning of Neural Networks

In this section we review existing methods on evolutionary neural network (NN) learning and present our own algorithm, EANT. The paradigm is to learn *both the structure (topology) and the parameters of NNs* with evolutionary algorithms (EAs) without being given any information about

Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007)
Published by the Neuroinformatics Group, Bielefeld University, Germany, ISBN 978-3-00-022473-7
All contributions to WSOM 2007 are available online at: http://biecoll.ub.uni-bielefeld.de

the nature of the problem. The development of networks is realised through reinforcement learning [16]. This means that candidate solutions which have been generated by the EA are evaluated by testing them on the target application. A scalar value of their "fitness" is fed back to the algorithm to help it judge and determine what to do with this candidate. These learning algorithm do not depend on the availability of input-output pairs of the NN as supervised learning methods do.

2.1 Overview over Existing Methods

Until recently, only small NNs have been evolved by evolutionary means [18]. According to Yao, a main reason is the difficulty of evaluating the exact fitness of a newly found structure: In order to fully evaluate a *structure* one needs to find the optimal (or, some near-optimal) *parameters* for it. However, the search for good parameters for a given structure has a high computational complexity unless the problem is very simple *(ibid.)*.

In order to avoid this problem most recent approaches evolve the structure and parameters of the NNs simultaneously. Examples include EPNet [19], GNARL [1] and NEAT [15]. EPNet uses a modified backpropagation algorithm for parameter optimisation (i.e. a local search method). The mutation operators for searching the space of neural structures are addition and deletion of neural nodes and connections (no crossover is used). A tendency to remove connections/nodes rather than to add new ones is realised in the algorithm. This is done to counteract the "bloat" phenomenon (i.e. ever growing networks with only little fitness improvement; also called "survival of the fattest" [3]). GNARL is similar in that is also uses no crossover during structural mutation. However, it uses an EA for parameter adjustments. Both parametrical and structural mutation use a "temperature" measure to determine whether large or small random modifications should be applied—a concept known from simulated annealing [10]. In order to calculate the current temperature, some knowledge about the "ideal solution" to the problem, e.g. the maximum fitness, is needed.

The author groups of both EPNet and GNARL are of the opinion that using crossover is not useful during the evolutionary development of neural networks [19, 1]. The research work underlying NEAT, on the other hand, seems to suggest otherwise. The authors have designed and used a crossover operator that allows to produce valid offspring from two given NNs by first aligning similar or equal subnetworks and then exchanging differing parts. Like GNARL, NEAT uses EAs for both parametrical and structural mutation. However, the probabilities and standard deviations used for random mutation are constant over time. NEAT also incorporates the concept of speciation, i.e. separated sub-populations that aim at cultivating and preserving diversity in the population [3, chap. 9].

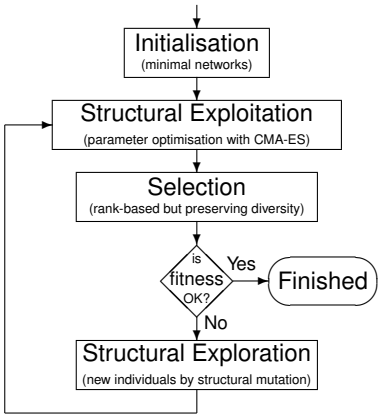


Figure 1: The EANT algorithm. Please note that CMA-ES has its own loop which creates a nested loop within EANT.

2.2 Developing Neural Networks with EANT

EANT ("Evolutionary Acquisition of Neural Topologies") is an evolutionary reinforcement learning system that realises NN learning with EAs both for the structural and the parametrical part [9]. EANT features a compact genetic encoding that uses a linear genome to represent a NN together with its parameters. The linear genome encodes the topology of the NN implicitly by the order of its elements (genes). The following gene types exist: neurons, inputs to the network, bias neurons, forward connections and recurrent connections. Linear genomes can be evaluated, without decoding, similar to the way mathematical expressions in postfix notation are evaluated. For example, a neuron gene is followed by its input genes. In order to evaluate it, one can traverse the linear genome from back to front, pushing inputs onto a stack. When encountering a neuron gene one pops as many genes from the stack as there are inputs to the neuron, using their values as input values. The resulting evaluated neuron is again pushed onto the stack, enabling this subnetwork to be used as an input to other neurons. Connection genes make it possible for neuron outputs to be used as input to more than one neuron. Together with the bias neurons that are implemented as having a constant value of 1.0, the linear genome can encode any NN in a very compact format. The length of the linear genome is equal to the number of synaptic network weights.

Figure 1 shows how EANT works. The different steps of the algorithm are explained in detail below.

Initialisation: EANT usually starts with minimal initial structures. A "minimal" network has no hidden layers or recurrent connections, only 1 neuron per output. Each neuron is connected to approx. 50 % of inputs; the exact percentage and selection of inputs are random. EANT gradually develops these simple initial network structures further using the structural and parametrical EAs discussed below. On a larger scale new neural structures are added

Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007)
Published by the Neuroinformatics Group, Bielefeld University, Germany, ISBN 978-3-00-022473-7
All contributions to WSOM 2007 are available online at: http://biecoll.ub.uni-bielefeld.de

to a current generation of networks. We call this "structural exploration". On a smaller scale the current individuals (structures) are optimised by changing their parameters: "structural exploitation".

Structural Exploitation: At this stage the structures in the current EANT population are exploited by optimising their parameters. Parametrical mutation in a previous version of EANT was implemented using evolution strategies (ES) [3]. This means that the strategy parameters in the EA, e.g. the standard deviation for random mutation, were themselves adapted by an EA. This has the advantage that the system needs even less knowledge of the problem than with a different EA, like evolutionary programming. However, using ES for parametrical mutation has the following disadvantages:

(1) After a strategy parameter has been adapted it takes many applications of the mutation operator on the corresponding network parameter until the new value of the strategy parameter can be judged. Even then it is unclear when looking at the change in fitness value whether the network performs better/worse because of this adapted strategy parameter or because of other changes that happened during those many generations.
(2) The number of strategy parameters adds to the number of total parameters in the system, increasing even further the dimensionality of the space in which ideal parameters are searched.

Disadvantage 1 can be ignored in settings where a very large population size is used. However, it does matter in the context of NN development where large population sizes are prohibitive unless the problem is very simple.

For these reasons newer versions of EANT use CMA-ES ("Covariance Matrix Adaptation Evolution Strategy") [6] in their parameter optimisation. CMA-ES is a variant of ES that avoids random adaptation of the strategy parameters. Instead, the search area that is spanned by the mutation strategy parameters, expressed here by a covariance matrix, is adapted at each step depending on the parameter and fitness values of current population members. CMA-ES uses sophisticated methods to avoid things like premature convergence and is known for fast convergence to good solutions even with multi-modal and non-separable functions in high-dimensional spaces *(ibid.)*.

When the parameter optimisation with CMA-ES starts it is given for each variable an initial standard deviation used in its sampling of values in the search space. These standard deviations will be used as a starting point only and are adapted over time. These values are set by EANT depending on the current age of the corresponding gene. Parameters for newer structural elements are given a wider search area than older ones. This feature is based on the observation that over time parameters for existing structures tend to become more or less constant as they have been optimised several times. Structural changes a other places may also influence the optimal parameter values for the older structural elements, but usually at a relatively small scale. This is related to the "Cascade-Correlation Learning" paradigm presented by Fahlman and Lebiere [4].

Selection: The selection operator determines which population members are carried on from one generation to the next. Our selection in the outer, structural exploration loop is rank-based and "greedy", preferring individuals that have a larger fitness. If two structures have almost the same fitness the smaller individual is given a higher rank. A consequence of this is that existing structures may (and often do) grow smaller if structural elements that do not help the performance are removed. In order to maintain diversity in the population, the selection operator also compares individuals by structure, ignoring their parameters. The operator makes sure that not more than 1 copy of an individual and not more than 2 similar individuals are kept in the population. "Similar" in this case means that a structure was derived from an another one by only changing connections, not adding neurons. Again, no network parameters are considered here.

Structural Exploration: In this step new structures are generated and added to the population. This is achieved by applying the following structural mutation operators to the existing structures: Adding a random subnetwork, adding or removing a random connection and adding a random bias. Removal of subnetworks (i.e. neurons together with all their connections) is not done as we found out that this almost never helps in the evolutionary process. The same is valid for a crossover operator, modelled after the one used in NEAT, which is currently not used. New hidden neurons are connected to approx. 50 % of inputs; the exact percentage and selection of inputs are random to enable stochastic search for new structures.

Differences to Other Methods: EANT is closely related to the methods described in the related work section above. One main difference is the *clear separation of structural exploration and structural exploitation*. By this we try to make sure a new structural element is tested ("exploited") as much as possible before a decision is made to discard it or keep it, or before other structural modifications are applied. Another main difference is the *use of CMA-ES in the parameter optimisation*. This should yield more optimal parameters more quickly, which is necessary when large networks are to be created. When EANT's *structural mutation operator* adds a new neuron to a given structure, it also *connects the new neuron* to a random number of other neurons and/or inputs, and the new neuron's output as input to other neurons. Further differences of EANT to other recent methods, e.g. NEAT, are a *small number of user-defined algorithm parameters* (the method should be as general as possible), its *compact, linear encoding of the NN* and the *explicit way of preserving diversity* in the population (unlike speciation in NEAT).

Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007)
Published by the Neuroinformatics Group, Bielefeld University, Germany, ISBN 978-3-00-022473-7
All contributions to WSOM 2007 are available online at: http://biecoll.ub.uni-bielefeld.de

Figure 2: Robot Arm with Camera and Object

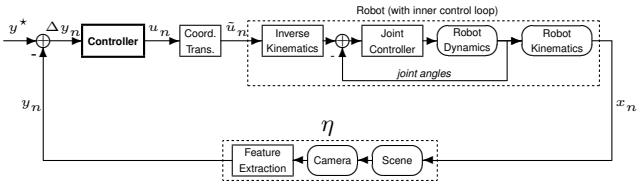


Figure 3: Visual Feedback Control Loop

3 The Visual Servoing Task

In order to study the behaviour of EANT and other algorithms on large problems we simulate the visual servoing setup shown in Figure 2. A robot is equipped with a camera at the end-effector and has to be steered towards an object of unknown pose. This is achieved in the visual feedback control loop depicted in Figure 3. In our system a NN shall be used as the controller, determining where to move the robot on the basis of the object's visual appearance.

The object has 4 identifiable markings. Its appearance in the image is described by the *image feature vector* $y_n \in \mathbb{R}^8$ that contains the 4 pairs of image coordinates of these markings. The desired pose relative to the object is defined by the object's appearance in that pose by measuring the corresponding *desired image features* $y^\star \in \mathbb{R}^8$ ("teaching by showing"). Object and robot are then moved into a start pose so that the position of the object is unknown to the controller. The input to the controller is the image error $\Delta y_n := y^\star - y_n$ and additionally the 2 distances in the image of the opposing markings, resulting in a 10-dimensional input vector. The output of the controller is a relative movement of the robot in the camera coordinate system: $(\Delta x, \Delta y, \Delta z) \in \mathbb{R}^3$.

From a mathematical point of view, visual servoing is the iterative minimisation of an error functional that describes differences of objects' visual appearances, by moving in the search space of robot poses. The traditional solution is equivalent to an iterative Gauss-Newton method [5] to minimise the image error, with a linear model ("Image Jacobian") of the objective function [8, 17].

In our case a NN is developed as a controller by reinforcement learning. For the assessment of the fitness (performance) of a network N it is tested by evaluating it in the simulated visual servoing setup. For this purpose 1023 different robot start poses and 29 teach poses (desired poses) have been generated. Each start pose is paired with a teach pose to form a task. These tasks contain all ranges and directions of movements. For each task, N is given the visual input data corresponding to the start and teach poses, and its output is executed by a simulated robot. The *fitness function* $F(N)$ measures the negative RMS (root mean square) of the remaining image errors after the robot movements, over all tasks. This means that our fitness function $F(N)$ always takes on negative values with $F(N) = 0$ being the optimal solution. Let y_i denote the new image features after executing one robot movement starting at start pose i. Then $F(N)$ is calculated as follows:

$$F(N) := -\sqrt{\frac{1}{1023}\sum_{i=1}^{1023}\left(\frac{1}{4}\sum_{j=1}^{4} d_j(y_i)^2 + b(y_i)\right)} \quad (1)$$

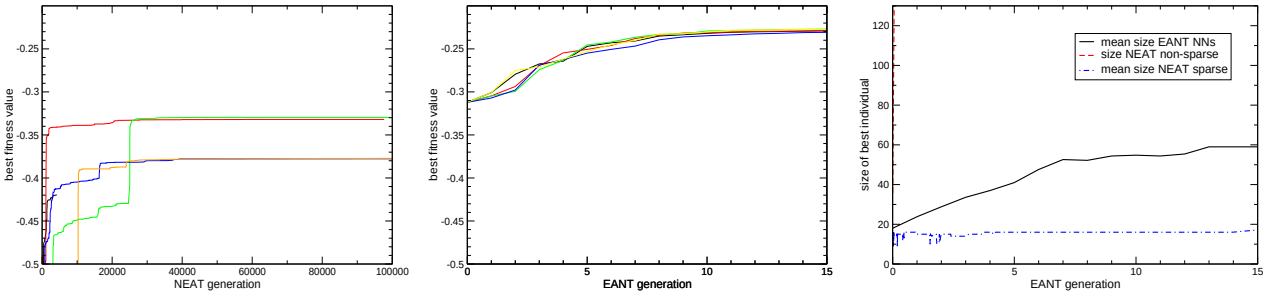
where

$$d_j(y_i) := \left\| (y^\star)_{2j-1,2j} - (y_i)_{2j-1,2j} \right\|_2 \quad (2)$$

is the distance of the jth marker position from its desired position in the image, and $(y)_{2j-1,2j}$ shall denote the vector comprising of the $2j-1$th and $2j$th component of a vector y. The inner sum of (1) thus sums up the squared deviations of the 4 marker positions in the image. $b(y)$ is a "badness" function that adds to the visual deviation an additional positive measure to punish potentially dangerous situations. If the robot moves such that features are not visible in the image or the object is touched by the robot, $b(y) > 0$, otherwise $b(y) = 0$. All image coordinates are in the camera image on the sensor and have therefore the unit 1 mm. The sensor (CCD chip) in this simulation measures $\frac{8}{3}$ mm $\times$ 2 mm. The average (RMS) image error is -0.85 mm at the start poses, which means that a network N that avoids all robot movements (e.g. a NN with all weights $= 0$) has $F(N) = -0.85$. $F(N)$ can easily reach values below -0.85 for networks that tend to move the robot away rather than towards the target object.

An analysis of the data set used for training the network was carried out to determine its intrinsic dimensionality. The dimensionality is (approximately) 4, the Eigenvalues being 1.70, 0.71, 0.13, 0.04 and the other 6 Eigenvalues below 1e-15. It is not surprising that the dimensionality is less than 10, and this redundancy makes it more difficult to train the NNs. However, we see this challenge as an advantage for our research, and the problem encoding is a standard one for visual servoing.

Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007)
Published by the Neuroinformatics Group, Bielefeld University, Germany, ISBN 978-3-00-022473-7
All contributions to WSOM 2007 are available online at: http://biecoll.ub.uni-bielefeld.de



(a) NEAT (sparse init.): fitness, best NN

(b) EANT: fitness, best NN

(c) mean NN size, NEAT and EANT

Figure 4: Results from 5 runs each of NEAT and EANT

4 Experimental Comparison of EANT and NEAT

In order to validate learning methods we use the simulated visual servoing scenario as described in the previous section, with 1023 start poses and the same definition of the fitness function F from equation (1). The 10 inputs and 3 outputs to the neural networks (NNs) are also as above. The computationally expensive evaluation of F which needs 1023 NN evaluations and simulated robot movements makes it a priority to develop networks with as few evaluations $F(N)$ as possible.

4.1 The NEAT System

NEAT by Stanley and Miikkulainen [15] has already been briefly introduced in Section 2.1. It uses one evolutionary optimisation loop in which structures and parameters of NNs are mutated, and networks recombined using a crossover operator. The implementation of NEAT used here is the Java-based NEAT4J which is available as a SourceForge project[1]. For reference the original NEAT code by Stanley has also been analysed.

The initial population of NEAT4J consists of randomly generated networks without hidden layers that are either fully or sparsely connected (at an option). In each generation the population is split into a number of species so that "compatible" individuals belong to the same species. The split is done using a compatibility measurement that incorporates network size, difference of weights and number of different genes. New species are created if necessary. If a species has a good average fitness, its size is increased, otherwise the size is decreased. Species become extinct if their size becomes zero or they excess a certain age. The best individual of each species is kept together with their offspring. New members of a species are spawned by crossover and mutation from their parents who are selected among the best individuals in this species. Mutation is done by a stochastic update of weights and structures. Nodes and connections are added with certain probabilities, but never removed. Existing connections can, however, be enabled or disabled by toggling a flag.

Search for Optimal NEAT4J Parameters: Unfortunately, there is no suggestion how NEAT's 13 evolution and 9 speciation parameters should be set. We have tried many settings and found out that the values from the examples of the original NEAT mixed with those of NEAT4J form a suitable starting point. The settings were then adapted to tune the system for our visual servoing task.

NEAT tends to enlarge networks if the *probability of toggling connections* on/off is low and slows down the growing of networks if it is high. After some test runs we decided to reduce the probability of toggling (PToggleLink 0.0001) so as to enable NEAT4J to sufficiently optimise the network weights before adding a lot of structure. For the same reason we also decreased the *probabilities for structural mutation* (PAddLink=0.0025, PAddNode=0.00125) after some test runs but left the *probabilities for weight changes* high (PMutation=0.25, PWeightReplaced=0.85). NEAT reacts very strongly to *bias neurons* and tends to add many of them. However, in a few test runs this made the evolution process get stuck without improving the fitness. We therefore deactivated biases altogether (which makes sense, considering the visual servoing task). An appropriate *population size* is hard to calculate but concerning the fitness increase over (wall-clock) time a smaller population size usually works better than a bigger. Hence, we tested two sizes of populations, 30 and 150. In most cases the smaller population only performed slightly worse. We did not note a significant change in the test outcome when varying parameters for *speciation*.

4.2 The EANT System

The EANT system which was described in detail in section 2.2 was used with the following parameters:

- up to 30 individuals in the structural exploration (global population size)

[1] `http://neat4j.sourceforge.net/`

- each individual spawns 2 children through structural mutation
- 2 parallel optimisations of the same individual by CMA-ES
- stop criteria for CMA-ES: maximum standard deviation in covariance matrix less than 0.00005 or iteration (CMA-ES generation) number over 500.

4.3 Results and Discussion

Figure 4 shows the development of the best individual's fitness value and size. Results from 5 experiments each of EANT and NEAT are shown, plotted against the generation number. EANT's (outer) generation number is increasing much slower than NEAT's because of the inner loop that is contained within the structural exploitation with CMA-ES. The generation spans and the graphs in Figure 4(c) have therefore been roughly aligned by the number of evaluations of the fitness function, which is the determining factor for the wall clock time used to run the method.

Development of Fitness: It can be seen that after around 25,000 generations the fitness values in NEAT reach -0.33 (better runs) and -0.38 (worse runs). They do not improve significantly further until generation 100,000, at which point the experiments were stopped. In EANT, a significant increase in fitness can be seen up to generation 15 (and further, as different experiments show). After 5 generations the average best individual has a fitness of -0.25, which increases to -0.23 at generation 15.

Let us recall that the fitness values are (modulo $b(\cdot)$) the remaining RMS errors in the image after the robot movement. Both methods quickly develop networks that reduce the image error from the initial -0.85 to as low as -0.23 with 1 robot movement. This is a very good result if one compares to the traditional Image Jacobian approach. Calculating the robot movement using the (undamped) product of the Image Jacobian's pseudoinverse with the negative image error, a standard method [8], yields a fitness of -0.61. Since both the Image Jacobian and our networks calculate the necessary camera movement to minimise the image error in *one* step this is a meaningful comparison and shows that these networks can indeed be used for visual servoing[2].

Development of Network Sizes: An analysis of the network sizes shows that NEAT's resulting networks stay "sparse" if that initialisation option was used. The best performing network has 17 genes, with only 2 hidden neurons. Only 1 gene was added between generation 3,000 and 100,000, which explains why the fitness does not increase any further. However, without the "sparse" option NEAT generates networks with sizes approx. 80–140 after 3,000 generations; their fitness is around -0.89 to -0.66.

EANT's networks are larger, in part due to the different initialisation. The mean size at generation 5 is 41 (fitness -0.25). Size increases slower as time goes on, with a mean size of 59 at generation 15 (fitness -0.23). NEAT's mean final network size of 17 is reached by EANT at generation 0 (with no hidden neurons). At this size the average fitnesses of the best individuals are -0.346 (NEAT) and -0.312 (EANT).

As time goes on EANT's structures continue to grow much further than NEAT's. Although NEAT does try to add new structure very often most of these structural elements are discarded. NEAT has a feature to keep newly created individuals even if they do not perform well in the first few generations of their existence but is seems that this feature does not help here.

The two methods, NEAT and EANT, differ in the way networks are generated, and NEAT performs worse in this scenario. Only when the networks are small and the probability of structural change is low compared to parametrical change can NEAT optimise networks well with its EA. If some options influence NEAT to produce larger networks they have a significantly worse performance compared to EANT networks of the same size. This could mean that NN parameters in NEAT are not optimised as well, or that structural elements exist that do not help the task well, or both.

Overall, EANT always created better networks than NEAT and required less parameter tuning to run successfully.

5 Conclusions and Future Work

In this article we have presented EANT, a method to develop both the structure and the parameters of neural networks (NNs) by evolutionary reinforcement learning. EANT differs from other recent methods by implementing a clear separation of structural and parametrical development and the use of CMA-ES during parameter optimisation. It also features a unique and compact genetic encoding of the NN.

In order to validate EANT, it was used with a complete simulation of a visual servoing scenario to learn NNs by reinforcement learning. The same task was given to NEAT [15], a similar method. Results from the experiments show that both evolutionary methods can develop networks that make "useful" robot movements, decreasing the image error and thereby moving towards the goal. The performance of both methods is also significantly better than the traditional visual servoing approach.

A comparison of both methods showed that the NNs created by EANT always have a significantly better performance. NEAT also performs good when configured to keep network sizes very small, but then the development of networks comes to a halt, showing almost no improvement over a long runtime. For similar network sizes, EANT's NN perform much better.

For these experiments EANT's parameter optimisation with CMA-ES has been reduced in complexity to make a

[2] Any dampening, if necessary, can be employed independent of the method that calculates the optimisation step (robot movement).

fair comparison possible; previous experiments used more CMA-ES generations [13]. Our current work is to study the dependence of EANT on CMA-ES's parameters.

Acknowledgements

The contribution of our colleague Yohannes Kassahun, i.e. the development of the original version of EANT within his PhD project in our research group, is gratefully acknowledged. The authors also wish to thank Nikolaus Hansen, the developer of CMA-ES, and Kenneth Stanley, the developer of NEAT, for kindly providing source code which helped us to quickly start applying their methods.

References

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

[2] R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, USA, 1961.

[3] Á. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, Berlin, Germany, 2003.

[4] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, Pittsburgh, USA, August 1991.

[5] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, Chichester, 2nd edition, 1987.

[6] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[7] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[8] S. Hutchinson, G. Hager, and P. Corke. A tutorial on visual servo control. Tutorial notes, Yale University, New Haven, USA, May 1996.

[9] Y. Kassahun and G. Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 259–266, Bruges, Belgium, April 2005.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[11] J. W. Melody. On universal approximation using neural networks. Report from project ECE 480, Decision and Control Laboratory, University of Illinois, Urbana, USA, June 1999.

[12] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer Verlag, Berlin, Germany, 1996.

[13] N. T. Siebel and Y. Kassahun. Learning neural networks for visual servoing using evolutionary methods. In *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand*, page 6 (4 pages), December 2006.

[14] J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, Hoboken, USA, 2003.

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

[16] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, USA, March 1998.

[17] L. E. Weiss, A. C. Sanderson, and C. P. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal of Robotics and Automation*, 3(5):404–417, October 1987.

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

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