Evolutionary Learning of Neural Structures for Visuo-Motor Control

Nils T Siebel[1], Gerald Sommer[1], and Yohannes Kassahun[2]

[1] Cognitive Systems Group, Institute of Computer Science,
Christian-Albrechts-University of Kiel, Germany
nils@siebel-research.de, gs@ks.informatik.uni-kiel.de
[2] Group for Robotics, DFKI Lab Bremen, University of Bremen, Germany
kassahun@informatik.uni-bremen.de

1 Introduction

Artificial neural networks are computer constructs inspired by the neural structure of the brain. The aim is to approximate the vast learning and signal processing power of the human brain by mimicking its structure and mechanisms. In an artificial neural network (often simply called "neural network"), interconnected neural nodes allow the flow of signals from special input nodes to designated output nodes. With this very general concept neural networks are capable of modelling complex mappings between the inputs and outputs of a system up to an arbitrary precision [13, 21]. This allows neural networks to be applied to problems in the sciences, engineering and even economics [4, 15, 25, 26, 30]. A further advantage of neural networks is the fact that learning strategies exist that enable them to adapt to a problem.

Neural networks are characterised by their *structure (topology)* and their *parameters* (which includes the weights of connections) [27]. When a neural network is to be developed for a given problem, two aspects need therefore be considered:

1. What should be the *structure* (or, *topology*) of the network? More precisely, how many neural nodes does the network need in order to fulfil the demands of the given task, and what connections should be made between these nodes?
2. Given the structure of the neural network, what are the optimal values for its *parameters*? This includes the weights of the connections and possibly other parameters.

1.1 Current Practice

Traditionally the solution to aspect 1, the network's *structure*, is found by trial and error, or somehow determined beforehand using "intuition". Finding the solution to aspect 2, its *parameters*, is therefore the only aspect that is usually considered in the literature. It requires optimisation in a parameter space that can have a very high dimensionality – for difficult tasks it can be up to several hundred. This so-called "curse of dimensionality" is a significant obstacle in machine learning problems [3, 22].[3] Most approaches for determining the parameters use the backpropagation algorithm [27, chap. 7] or similar methods that are, in effect, simple stochastic gradient descent optimisation algorithms [28, chap. 5].

1.2 Problems and Biology-inspired Solutions

The traditional methods described above have the following deficiencies:

1. The common approach to pre-design the network structure is difficult or even infeasible for complicated tasks. It can 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 [24]. They only work well with very simple (e.g., convex) target functions or if an approximate solution is known beforehand *(ibid.)*.

In short, these methods lack generality and can therefore only be used to design neural networks for a small class of tasks. They are engineering-type approaches; there is nothing wrong with that if one needs to solve only a single, more or less constant problem[4] but it makes them unsatisfactory from a scientific point of view.

In order to overcome these deficiencies the standard approaches can be replaced by more general ones that are inspired by biology. Evolutionary theory tells us that the *structure* of the brain has been developed over a long period of time, starting from simple structures and getting more complex over time. In contrast to that, the *connections* between biological neurons are modified by experience, i.e. learned and refined over a much shorter time span.

[3] When training a network's parameters by examples (e.g. supervised learning) it means that the number of training examples needed increases exponentially with the dimension of the parameter space. When using other methods of determining the parameters (e.g. reinforcement learning, as it is done here) the effects are different but equally detrimental.

[4] The No Free Lunch Theorem states that solutions that are specifically designed for a particular task always perform better at this task than more general methods. However, they perform worse on most or all other tasks, or if the task changes.

In this chapter we will introduce such a method, called *EANT*, *Evolutionary Acquisition of Neural Topologies*, that works in very much the same way to create a neural network as a solution to a given task. It is a very general learning algorithm that does not use any pre-defined knowledge of the task or the required solution. Instead, EANT uses evolutionary search methods on two levels:

1. In an outer optimisation loop called *structural exploration* new neural *structures* are developed by gradually adding new structure to an initially minimal network that is used as a starting point.
2. In an inner optimisation loop called *structural exploitation* the *parameters* of all currently considered structures are adjusted to maximise the performance of the networks on the given task.

To further develop and test this method, we have created a simulation of a *visuo-motor control* problem, also known in robotics as *visual servoing*. A robot arm with an attached hand is to be controlled by the neural network to move to a position where an object can be picked up. The only sensory data available to the network is visual data from a camera that overlooks the scene. EANT was used with a complete simulation of this visuo-motor control scenario to learn networks by reinforcement learning.

The remainder of this chapter is organised as follows. In Section 2 we briefly describe the visuo-motor control scenario and review related work. Section 3 presents the main EANT approaches: genetic encoding of neural networks and evolutionary methods for structural exploration and exploitation. In Section 4 we analyse a deficiency of EANT and introduce an improved version of EANT to overcome this problem. Section 5 details the results from experiments where a visuo-motor control has been learnt by both the original and our new, improved version of EANT. Section 6 concludes the chapter.

2 Problem Formulation and Related Work

2.1 Visuo-Motor Control

Visuo-motor control (or visual servoing, as it is called in the robotics community) is the task of controlling the actuators of a manipulator (e.g. a human or robot arm) by using visual feedback in order to achieve and maintain a certain goal configuration. The purpose of visuo-motor control is usually to approach and manipulate an object, e.g. to pick it up.

In our setup a robot arm is equipped with a camera at the end-effector and has to be steered towards an object of unknown pose[5] (see Figure 1). This is realised in the visual feedback loop depicted in Figure 2. In our case a neural network shall be used as the controller, determining where to move the

[5] The *pose* of an object is defined as its position and orientation.

Fig. 1. Robot Arm with Camera and Object

robot on the basis of the object's appearance in the image. Using the standard robotics terminology defined by Weiss et al. [32] our visuo-motor controller is of the type "Static Image-based Look-and-Move".

The object has 4 circular, 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 so that no Euclidean position of the object or robot is known to the controller. The system has the task of moving the arm such that the current image features resemble the desired image features. This is an iterative process. In some of our experiments the orientation of the robot hand was fixed, allowing the robot to move in 3 degrees of freedom, DOFs. In others the orientation of the hand was also controllable, which means the robot could move in 6 DOFs.

The *input* to the controller is the *image error* $\Delta y_n := y^\star - y_n$ and additionally image measurements which enable the neural network to make its output dependent on the context. In some of the experiments this was simply y_n, resulting in a 16-dimensional input to the network. In other experiments the additional inputs were the 2 distances in the image of the diagonally opposing markings, resulting in a 10-dimensional input vector. The *output* of the controller/neural network is a relative movement u_n of the robot in the camera coordinate system: $u_n = (\Delta x, \Delta y, \Delta z)$ when moving the robot in 3 DOFs, $u_n = (\Delta x, \Delta y, \Delta z, \Delta yaw, \Delta pitch, \Delta roll)$ when moving in 6 DOFs. This output is given as an input to the robot's internal controller which executes the movement. The new state x_{n+1} of the environment (i.e. the robot and scene)

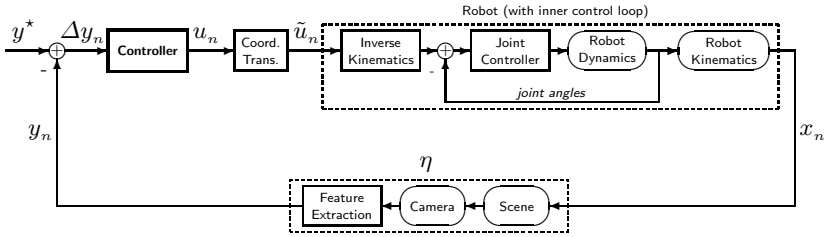


Fig. 2. Visual Feedback Control Loop

is perceived by the system with the camera. This is again used to calculate the next input to the controller, which closes the feedback loop shown in Figure 2.

In order to be able to learn and train the neural network by reinforcement learning a simulation of the visuo-motor control scenario was implemented. 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 s different robot start poses and 29 teach poses (desired poses) have been generated. In case of the 3 DOF scenario we used $s = 1023$ start poses, in the 6 DOF scenario $s = 58,677$ start poses. 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:

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

where

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

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 $\frac{8}{3}\,\text{mm} \times 2\,\text{mm}$.

For the 3 DOF data set the average (RMS) image error is $-0.85\,\text{mm}$ at the start poses, which means that a network N that avoids all robot movements (e.g. a network 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 in the 3 DOF case 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 (the length of the input vector). This redundancy makes it more difficult to train the neural networks, however, we see this challenge as an advantage for our research, and the problem encoding is a standard one for visual servoing.

2.2 Related Work: Visuo-Motor Control

Visuo-motor control is one of the most important robot vision tasks [14, 32]. Traditionally it uses a simple P-type controller—an approach known from engineering [5]. In these controllers the output is determined as the minimal vector that solves the locally linearised equations describing the image error as a function of the robot movement. This output is often multiplied by a constant scale factor α, $0 < \alpha < 1$ (dampening). Sometimes, more elaborate techniques like trust-region methods are also used to control the step size of the controller depending on its current performance [16]. From a mathematical point of view, visuo-motor control is the iterative minimisation of an error functional that describes differences of objects' appearances in the image, by moving in the search space of robot poses. The traditional approach to a solution then becomes an iterative Gauss-Newton method [9] to minimise the image error, using a linear model ("Image Jacobian") of the objective function.

There have also been approaches to visuo-motor control using neural networks, or combined Neuro-Fuzzy approaches like the one by Suh and Kim [11]. Urban et al. use a Kohonen self-organising map (SOM) to estimate the Image Jacobian for a semi-traditional visuo-motor control [31]. Zeller et al. also train a model that uses a Kohonen SOM, using a simulation, to learn to control the position of a pneumatic robot arm based on 2 exteroceptive and 3 proprioceptive sensor inputs [35].

Many of these methods reduce the complexity of the problem (e.g. they control the robot in as few as 2 degrees of freedom, DOFs) to avoid the problems of learning a complex neural network. Others use a partitioning of the workspace to learn a network of "local experts" that are easier to train [6, 12]. A neural network that controls a robot to move around obstacles

is presented in [23]. The network is optimised by a genetic algorithm, however, its structure (topology) is pre-defined and does not evolve.

There are a few methods that develop both the structure and the topology of a neural network by evolutionary means. These methods will be discussed in section 3.5 below. However, we have not seen these methods applied to visuo-motor control problems similar to ours.

To our mind it is a shortcoming of most (if not, all) existing visuo-motor control methods that the solution to the control task is modelled by the designer of the software. Whether it be using again an Image Jacobian, or whether it be selecting the size and structure of the neural network "by hand"—that is, by intuition and/or trial and error—*these methods learn only part of the solution by themselves*. Training the neural network then becomes "only" a parameter estimation, even though the curse of dimensionality still makes this very difficult.

We wish to avoid this pre-designing of the solution. Instead, *our method learns both the structure (topology) and the parameters of the neural network* without being given any information about the nature of the problem. To achieve this, we have used our own, recently developed method, *EANT*, *Evolutionary Acquisition of Neural Topologies* [19] and improved its convergence with an optimisation technique called CMA-ES [10] to *develop a neural network from scratch by evolutionary means* to solve the visuo-motor control problem.

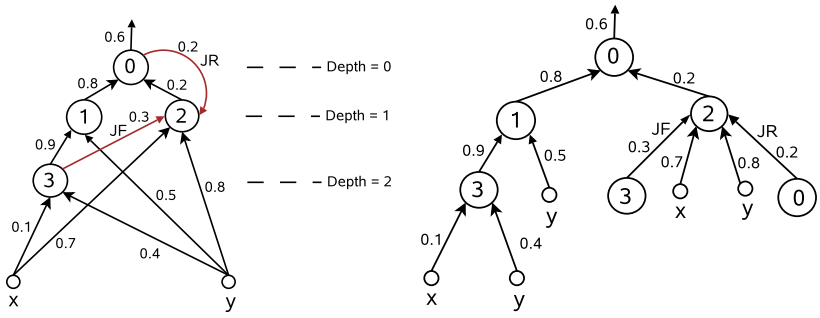
3 Developing Neural Networks with EANT

EANT, *Evolutionary Acquisition of Neural Topologies* [18, 19], is an evolutionary reinforcement learning system that is suitable for learning and adapting to the environment through interaction. It combines the principles of neural networks, reinforcement learning and evolutionary methods.

3.1 EANT's Encoding of Neural Networks: The Linear Genome

EANT uses a unique genetic encoding that uses a linear genome of genes that can take different forms. A gene can be a neuron, an input to the neural network or a connection between two neurons. We call "irregular" connections between neural genes "jumper connections". Jumper genes are introduced by structural mutation along the evolution path. They can encode either forward or recurrent connections.

Figures 3(a) through 3(c) show an example encoding of a neural network using a linear genome. The figures show (a) the neural network to be encoded. It has one forward and one recurrent jumper connection; (b) the neural network interpreted as a tree structure; and (c) the linear genome encoding the neural network. In the linear genome, `N` stands for a neuron, `I` for an input to



(a) Original neural network (b) Same network in tree format

N 0 W=0.6	N 1 W=0.8	N 3 W=0.9	I x W=0.1	I y W=0.4	I y W=0.5	N 2 W=0.2	JF 3 W=0.3	I x W=0.7	I y W=0.8	JR 0 W=0.2

(c) Corresponding Linear Genome

Fig. 3. An example of encoding a neural network using a linear genome

the neural network, `JF` for a forward jumper connection, and `JR` for a recurrent jumper connection. The numbers beside `N` represent the global identification numbers of the neurons, and `x` or `y` represent the inputs coded by the input gene. As can be seen in the figure, a linear genome can be interpreted as a tree based program if one considers all the inputs to the network and all jumper connections as terminals.

The linear genome encodes the topology of the neural network implicitly in the ordering of the elements of the linear genome. This enables one to evaluate the represented neural controller without decoding it. The evaluation of a linear genome is closely related to executing a linear program using a postfix notation. In the genetic encoding the operands (inputs and jumper connections) come before the operator (a neuron) if one goes from right to left along the linear genome. The linear genome is *complete* in that it can represent any type of neural network. It is also a *compact* encoding of neural networks since the length of the linear genome is the same as the number of synaptic weights in the neural network. It is *closed* under structural mutation and under a specially designed crossover operator (see below). An encoding scheme is said to be closed if all genotypes produced are mapped into a valid set of phenotype networks [17].

If one assigns integer values to the genes of a linear genome such that the integer values show the difference between the number of outputs and number

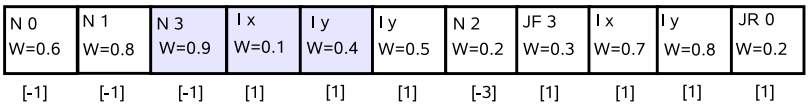


Fig. 4. An example of the use of assigning integer values to the genes of the linear genome. The linear genome encodes the neural network shown in Figure 3(a). The numbers in the square brackets below the linear genome show the integer values assigned to the genes of the linear genome. Note that the sum of the integer values is 1 showing that the neural network encoded by the linear genome has only 1 output. The shaded genes form a sub-network. The sum of these values assigned to a sub-network is always 1.

of inputs to the genes, one obtains the following rules useful in the evolution of the neural controllers:

1. The sum of integer values is the same as the number of outputs of the neural controller encoded by the linear genome.
2. A sub-network (sub-linear genome) is a collection of genes starting from a neuron gene and ending at a gene where the sum of integer values assigned to the genes between and including the start neuron gene and the end gene is 1.

Figure 4 illustrates this concept. Please note that only the number of inputs to neural genes is variable, so in order to achieve a compact representation only this number is stored within the linear genome.

3.2 Initialisation of the EANT Population of Networks

EANT starts with initial structures that are generated using either the full or the grow method as known from genetic programming [2]. Given a maximum initial depth of a tree (or in our case, a neural network), the full method generates trees where each branch has exactly the maximum depth. The grow method, on the other hand, adds more stochastic variation such that the depth of some (or all) branches may be smaller. Initial structures can also be chosen to be minimal, which is done in our experiments. This means that an initial network has no hidden layers or jumper connections, only 1 neuron per output with each of these connected to all inputs. Starting from simple initial structures is the way it is done by nature and most of the other evolutionary methods [33].

Using this simple initial network structure as a starting point, EANT incrementally develops it further using evolutionary methods. On a larger scale new neural structures are added to a current generation of networks. We call this process the "exploration" of new structures. On a smaller scale the current individuals (neural networks) are optimised by changing their parameters, resulting in an "exploitation" of these existing structures. Both of these optimisation loops are implemented as evolutionary processes. The

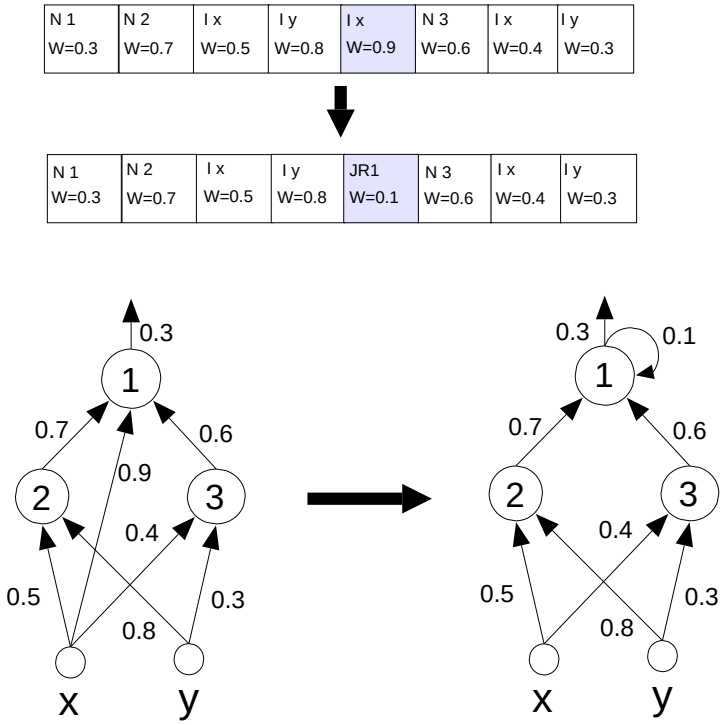


Fig. 5. An example of structural mutation. The structural mutation deleted the input connection to `N1` and added a self-recurrent connection at its place.

main search operators in EANT are structural mutation, parametric mutation and crossover. The search stops when a neural controller is obtained that can solve the given task.

3.3 Structural Exploration: Search for Optimal Structures

Starting from a simple initial network structure, EANT gradually develops it further using structural mutation and crossover.

The *structural mutation* adds or removes a forward or a recurrent jumper connection between neurons, or adds a new sub-network to the linear genome. It does not remove sub-networks since removing sub-networks causes a tremendous loss of performance of the neural controller. The structural mutation operates only on neuron genes. The weights of a newly acquired topology are initialised to zero so as not to disturb the performance of the network. Figure 5 shows an example of structural mutation where a neuron gene lost connection to an input and received a self-recurrent connection.

As a recently added new feature to EANT, new hidden neurons that are added by structural mutation are only connected to a subset of inputs. These

inputs are randomly chosen. This makes the search for new structures "more stochastic".

The *crossover* operator exploits the fact that structures originating from the same initial structure have some parts in common. By aligning the common parts of two randomly selected structures, it is possible to generate a third structure which contains the common and disjoint parts of the two parent structures. This type of crossover was introduced by Stanley and Miikkulainen [29] and has been incorporated into EANT. An example of the crossover operator under which the linear genome is closed is shown in Figure 6.

The structural selection operator that occurs at a larger timescale selects the first half of the population to form the next generation. New structures that are introduced through structural mutation and which are better according to the fitness evaluations survive and continue to exist. Since sub-networks that are introduced are not removed, there is a gradual increase in the complexity of structures along the evolution. This allows EANT to search for a solution starting from a structure of minimum complexity and developing it further without the need to specify the required network size beforehand.

3.4 Structural Exploitation: Search for Optimal Parameters

In the exploitation step, EANT optimises the weights (and possibly, other parameters) of the networks, i.e. it *exploits the existing structures*. This is accomplished by an evolutionary process that occurs at smaller timescale. This process uses parametric mutation as a search operator. Parametric mutation is accomplished by perturbing the weights of the controllers according to the uncorrelated mutation in evolution strategy or evolutionary programming [7].

3.5 Related Work: Evolving Neural Networks

Until recently, only small neural networks have been evolved by evolutionary means [33]. 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 neural networks simultaneously. Examples include EPNet [34], GNARL [1] and NEAT [29].

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 genes and connections (no crossover is used). A tendency to remove connections/genes 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" [7]).

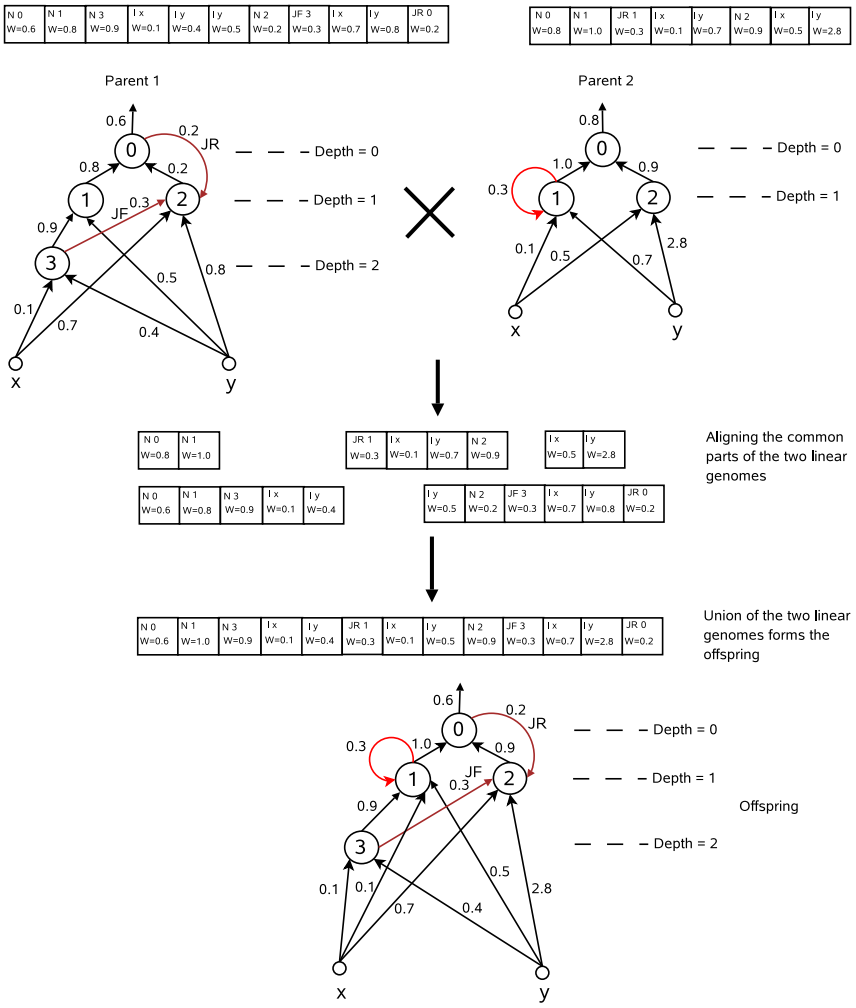


Fig. 6. Performing crossover between two linear genomes. The genetic encoding is closed under this type of crossover operator since the resulting linear genome maps to a valid phenotype network. The weights of the genes of the resulting linear genomes are inherited randomly from both parents.

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 [20]. 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 [34, 1]. The research work underlying NEAT, on the other hand, seems to suggest otherwise. The authors have designed and used the crossover operator described in section 3.3 above. It allows to produce valid offspring from two given neural networks 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 [7, chap. 9].

4 Combining EANT with CMA-ES

4.1 Motivation

During and after its development in the recent years EANT was tested on simple control problems like pole balancing where is has shown a very good performance and convergence rate—that is, it learns problems well with only few evaluations of the fitness function. However, problems like pole balancing are relatively simple. For instance, the neural network found by EANT to solve the double pole balancing problem without velocity information only has 1 hidden neuron. The new task, visuo-motor control, is much more complex and therefore required some improvements to EANT on different levels.

In order to study the behaviour of EANT on large problems we have implemented visuo-motor control simulators both for 6 DOF and 3 DOF control problems with 16/6 and 10/3 network in-/outputs, respectively. As a *fitness function* F for the individual networks N we used again the negative RMS of the remaining image errors, as defined in (1) above. This means that fitness takes on negative values with 0 being the optimal solution. Our studies were carried out on a small Linux PC cluster with 4 machines that each have 2 CPUs running at 3 GHz. This parallelisation was necessary because of the significant amount of CPU time required by our simulation. One evaluation of the fitness function in the 6 DOF case (58,677 robot movements and image acquisitions) takes around 1 second. In the 3 DOF case ("only" 1023 robot movements) the evaluation is of course considerably faster, however, parallelisation is still very helpful.

For our experiments we have set the following EANT parameters:

- population size: 20 individuals for exploration, $7\,n$ for exploitation, n being the size of the network
- structural mutation: enabled with a probability of 0.5
- parametric mutation: enabled; mutating all non-output weights
- crossover: disabled

- initial structure: minimal; 1 gene per output, each connected to all inputs

The results from our initial studies have shown that EANT does develop structures of increasing complexity and performance. However, we were not satisfied with the speed at which the performance (fitness) improved over time.

4.2 Improving the convergence with CMA-ES

In order to find out whether the structural exploration or the structural exploitation component needed improvement we took intermediate results (individuals with 175–200+ weights) and tried to optimise these structures using a different optimisation technique. Since our problem is highly non-linear and multi-modal (i.e. it has multiple local optima) we needed to use a global optimisation technique, not a local one like backpropagation, which is usually equivalent to a stochastic gradient descent method and therefore prone to get stuck in local minima. For our re-optimisation tests we used a global optimisation method called "CMA-ES" (Covariance Matrix Adaptation Evolution Strategy) [10] that is based on an evolutionary method like the one used in EANT's exploitation. CMA-ES includes features that improve its convergence especially with multi-modal functions in high-dimensional spaces.

One important difference between CMA-ES and the traditional evolutionary optimisation method in EANT is in the calculation of the search area for sampling the optimisation space by individuals. In EANT, each parameter in a neural network carries with it a learning rate which corresponds to a standard deviation for sampling new values for this parameter. This learning rate is adapted over time, allowing for a more efficient search in the space: parameters (e.g. input weights) that have not been changed for a long time because they have been in the network for a while are assumed to be near-optimal. Therefore their learning rate can be decreases over time, effectively reducing the search area in this dimension of the parameter space. New parameters (e.g. weights of newly added connections), on the other hand, still require a large learning rate so that their values are sampled in a larger interval. This technique is closely related to the "Cascaded Learning" paradigm presented by Fahlman and Lebiere [8]. It allows the algorithm to concentrate on the new parameters during optimisation and generally yields better optimisation results in high-dimensional parameter spaces, even if a relatively small population is used.

In EANT, the adaptation of these search strategy parameters is done randomly using evolution strategies [7]. This means that new strategy parameters may be generated by mutation and will then influence the search for optimal network parameters. However, one problem with this approach is that it may take a long time for new strategy parameters to have an effect on the fitness value of the individual. Since adaptation is random the strategy parameter may have been changed again by then, or other changes may have influenced

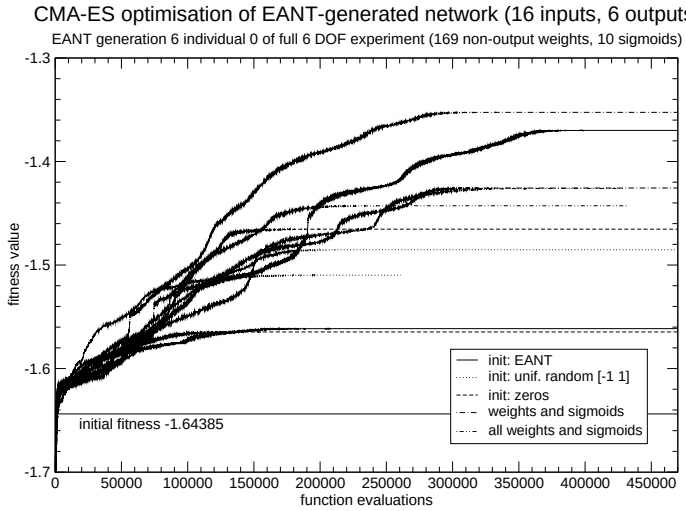


Fig. 7. Re-optimisation with CMA-ES of networks generated by EANT, 6 DOF

fitness. Therefore evolution strategies tend to work well only for evolutionary algorithms that work with large populations and/or over very many generations.

CMA-ES uses a method similar to the original evolution strategies concept. The sampling range of the parameters is also expressed by a standard deviation hyper-ellipsoid. However, it is not necessarily aligned with the coordinate axes (and hence, its axes identified with specific parameters). Instead, it is expressed in a general covariance matrix which is adapted over time (leading to the letters "CMA", Covariance Matrix Adaptation) depending on the function values within the current population. Different to EANT's evolution strategies, the strategy parameters that define the hyper-ellipsoid are not adapted by random mutation. Instead, they are 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.)*.

Using CMA-ES, we improved the fitness of a network for the 6 DOF problem from -1.64 to a level between -1.57 and -1.35 in several experiments. This can be seen in Figure 7 where the fitness is plotted against evaluations of the fitness function (short: "fevals" for "function evaluations"). Each such experiment has taken up to 24,000 CMA-ES generations (456,000 fevals) which has taken slightly more than 41 days to develop running on a single CPU. However, as can be seen in Figure 7, it should be noted that all of these experiments showed improvement over the original fitness in under 10 hours. Most of the CPU time was taken up by fevals but a small percentage also by CMA-ES internal calculations that—among other things—aim to reduce the

total number of fevals needed for the optimisation. While these results show that the optimisation still needs a large amount of CPU time (at least in the 6 DOF case) the improvement in fitness was very significant. Experiments with different initialisations have shown that the convergence of CMA-ES does not depend much on the initial values of the parameters. The variance in the resulting fitness values for optimisations with the same initial values is also an indicator for the multi-modality of our problem, which stems, in part, from our definition of the fitness function as a sum of many functions.

These 6 DOF experiments are very expensive to run due to their enormous CPU usage. However, with only few experiments the results have only little statistical significance. Therefore we made further experiments with networks of similar size for the 3 DOF visuo-motor control scenario. In over 700 CMA-ES re-optimisation experiments we always achieved very similar results to the ones shown in Figure 7 and conclude that the parameter optimisation with CMA-ES really does usually give a significant improvement in fitness value.

Our experiments have shown that by optimising only the parameters of the networks that were generated by EANT their fitness can be considerably increased. This indicates that for networks of this size the exploitation part of EANT that optimises these parameters can and should be improved. Encouraged by our positive results with CMA-ES we have therefore *replaced the structural exploitation loop of EANT* by a new strategy that uses *CMA-ES as its optimisation technique*.

5 Experimental Evaluation

5.1 Experimental Setup

After the studies with 6 DOF visuo-motor control described in Section 4 we decided that the 3 DOF case with 1023 start poses will be difficult enough to test the new CMA-ES-based EANT and compare it to the original version. With the same setup as before, 10/3 in-/outputs and starting from minimal structures (3 output neurons, each connected to all 10 inputs) we started EANT again on 4 PCs, each with 2 CPUs running at 3 GHz. One master process was responsible for the exploration of new structures and distributed the CMA-ES optimisation of the individuals (exploitation of current structures) to slave processes. We used the same fitness function $F(N)$ as before for evaluating the individual networks N, in this case

$$F(N) = -\sqrt{\frac{1}{1023}\sum_{i=1}^{1023}\left(\frac{1}{4}\sum_{j=1}^{4}\left\|(y^\star)_{2j-1,2j} - (y_i)_{2j-1,2j}\right\|_2^2 + b(y_i)\right)}, \quad (3)$$

which means that fitness takes again on negative values with 0 being the optimal solution.

Up to 15 runs of each method have been made to ensure a statistically meaningful analysis of results. Different runs of the methods with the same parameters do not differ much; shown and discussed below are therefore simply the mean results from our experiments.

5.2 Parameter Sets

Following our paradigm to put as little problem-specific knowledge into the system as possible, we used the standard parameters in EANT and CMA-ES to run our experiments. Additionally we introduced CMA-ES stop conditions that were determined in a few test runs. They were selected so as to make sure that the CMA-ES optimisation converges to a solution, i.e. the algorithm runs until the fitness does not improve any longer. These very lax CMA-ES stop criteria (details below) allow for a long run with many function evaluations (fevals) and hence can take a lot of time to terminate. In order to find out how much the solution depends on these stop criteria additional experiments were made with a second set of CMA-ES parameters, effectively allowing only $\frac{1}{10}$ of the fevals and hence speeding up the process enormously. Altogether this makes 3 types of experiments:

1. the original EANT with its standard parameters;
2. the CMA-ES-based EANT with a large allowance of fevals; and
3. the CMA-ES-based EANT with a small allowance of fevals

The optimisations had the following parameters:

Original EANT

- initial structure: minimal; 1 neuron per output, each connected to all inputs
- up to 20 individuals allowed in the exploration of new structures (global population size)
- structural mutation: enabled with probability 50 %; random addition of hidden neurons and forward connections enabled; recurrent connections and crossover operator disabled
- parametric mutation: enabled for non-output weights, probability 5 %
- exploration: new hidden neurons connected to all inputs
- exploitation: 3 to 6 parallel optimisations of the same individual, adaptively adjusted according to current speed of increase in fitness (threshold: 5 % fitness improvement during exploitation)
- exploitation population size $7\,n$ or $14\,n$, n being the size of the network; adaptation as before
- number of exploitation generations: 15 or 30; adapted as above

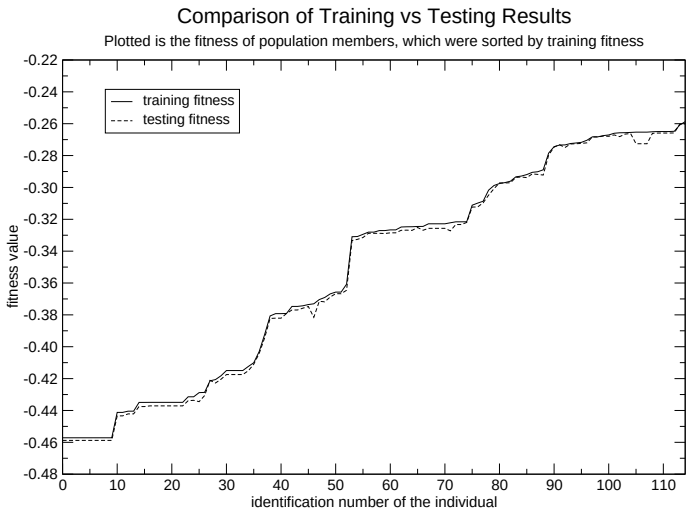


Fig. 8. Comparison Training vs. Testing Fitness Values

CMA-ES EANT, many fevals

- new hidden neurons connected to approx. 50 % of inputs (which makes the search for new structures stochastic)
- always only 3 parallel optimisations of the same individual
- $4 + 3 \log n$ CMA-ES individuals (equal to number of fevals per CMA-ES generation) when optimising n parameters
- CMA-ES stop criteria: improvement over previous 5,000 generations (iterations) less than 0.000001 or maximum standard deviation in covariance matrix less than 0.00005 or number of generations more than 100,000
- up to 2 optimisation results of the same individual may be kept, so that a single structure cannot take over the whole population in less than 5 generations (very unlikely; it has not happened)

All other parameters, especially the exploration parameters, remained the same as in the original EANT.

CMA-ES EANT, few fevals

The only change compared to the previous parameters is that all stop criteria of the CMA-ES optimisation are more strict by a factor of 10:

- CMA-ES stop criteria: improvement over previous 5,000 generations (iterations) less than 0.00001 or maximum standard deviation in covariance matrix less than 0.0005 or number of generations more than 10,000

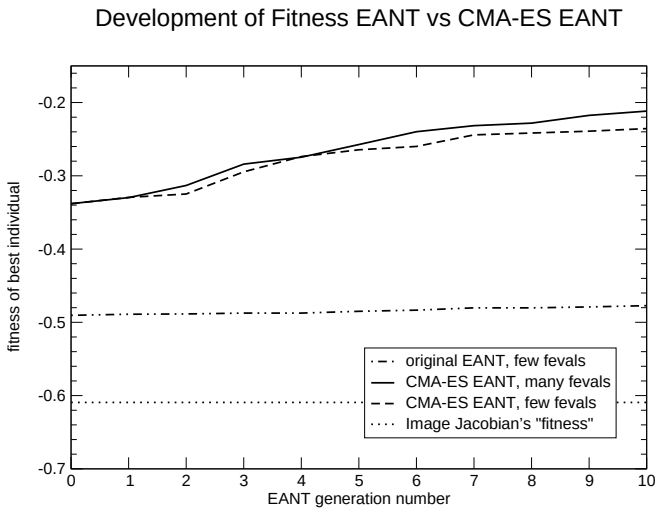


Fig. 9. Results from 3 DOF experiments: EANT and CMA-ES-based EANT

Due to the nature of the stop criteria in both cases the number of generations in the CMA-ES optimisations varied strongly (from 2,696 to 100,000). Therefore a parallelisation scheme with dynamic load balancing was employed.

5.3 Training and Testing

In order to carry out a meaningful analysis of the neural networks trained by the EANT system we have generated a test set of 1023 visuo-motor control tasks. They are comparable with the 1023 tasks the system was trained on. In particular, the fitness value when not moving the robot is the same. However, the testing data require completely different robot movements. All 115 neural networks that were generated as intermediate results during one run of EANT were tested, without any change to them, on the testing data. Figure 8 shows a comparison of the resulting fitness values of these individuals, sorted by training fitness. It can be seen that the training and testing fitnesses are very similar indeed. The maximum deviation of testing fitnesses compared to training fitnesses is 2.738 %, the mean deviation 0.5527 % of the fitness value. From this follows that the neural networks developed with our technique did not just memorise the correct responses of the network but are capable of generalising to different, but compatible tasks.

5.4 Results

Figure 9 shows the development of the fitness value, plotted against the EANT exploration generation since this is the determining factor for the complexity of the networks. It can be clearly seen that our new version of EANT which

uses CMA-ES in the exploitation converges much faster to networks with a good fitness. While the original EANT approach develops from around -0.490 to -0.477 in 11 generations both CMA-ES-based methods reach a value around -0.2 at that point.

The CMA-ES exploitation experiment with "many fevals" uses many more fevals per exploration generation than the EANT's original exploitation, resulting in longer running times of the algorithm. However, running times of the "few fevals" variant uses about the same number of function evaluations as the original EANT. When comparing the resulting fitness values of these two variants one can see a difference of only about 10 %. This indicates that the more strict stop criteria could be sufficient for the problem.

5.5 Discussion

Let us recall that the fitness values are (modulo $b(\cdot)$) the remaining RMS errors in the image after the robot movement. The value without moving the robot is -0.85. For comparison with the new methods we also calculated the fitness value of the traditional Image Jacobian approach. The robot movement was calculated using the (undampened) product of the Image Jacobian's pseudoinverse with the negative image error, a standard method [14]. The resulting fitness is -0.61. Why is this comparison meaningful? As in most optimisation techniques, both the Image Jacobian and our networks calculate the necessary camera movement to minimise the image error in *one* step. However, in practice optimisation techniques (which includes visuo-motor control methods) usually multiply this step by a scalar *dampening factor* before executing it. This dampening of the optimisation step is independent of the nature of the model that was used to calculate it[6].

This comparison with the standard approach shows that our networks are very competitive when used for visuo-motor control. It can of course be expected that a non-linear model will be able to perform better than a linear model. However, it should be taken into consideration that the Image Jacobian is an analytically derived solution (which is something we aim to avoid). Also, and more importantly, the Image Jacobian contained the exact distance (z coordinate) of the object from the camera. While this is easy to provide in our simulator in practice it could only be estimated using the image features.

The two types of experiments using CMA-ES have also shown that in our optimisation task CMA-ES is not very demanding when it comes to the number of fevals. The experiments with "few fevals" resulted in fitness values similar to those obtained with "many fevals" while keeping the CPU requirements about the same as the original EANT. Separate re-optimisation attempts with the original EANT exploitation that allowed for more fevals did not improve the performance of existing EANT individuals significantly.

[6] It is nevertheless useful to make it dependent on the correctness of the model, as it is done in *restricted step methods* [9].

It can therefore be assumed that the main performance increase stems from the use of CMA-ES, not from a the CPU time/feval allowance.

To conclude, it can be clearly seen that the use of CMA-ES in EANT's exploitation results in better performance of neural networks of comparable size while not increasing the computational requirements. The performance of neural networks developed with our new method is also very much better than that of traditional methods for visuo-motor control.

6 Summary

Our aim was to develop neural networks automatically that can be used as a controller in a visuo-motor control scenario. At the same time, our second goal was to develop a method that would generate such networks with a minimum of predetermined modelling. To achieve this, we used and improved our own evolutionary method called *EANT*, *Evolutionary Acquisition of Neural Topologies*. EANT uses evolutionary search methods on two levels: In an outer optimisation loop called *structural exploration* new networks are developed by gradually adding new structures to an initially minimal network. In an inner optimisation loop called *structural exploitation* the parameters of current networks are optimised. EANT was used with a complete simulation of a visuo-motor control scenario to learn neural networks by reinforcement learning.

Initial experiments with a 6 DOF visuo-motor control scenario have shown that EANT is generally well suitable for the task. However, the convergence of the original EANT method needed to be improved. Tests with resulting networks have indicated that it is most useful to work on the exploitation of network structures. Re-optimisation of existing networks with an optimisation method called *CMA-ES*, *Covariance Matrix Adaptation Evolution Strategy* have shown a significant improvement in the controller's performance.

Based on these results the exploitation loop of EANT was replaced with a new strategy that uses CMA-ES as its optimisation algorithm. Experiments with this new method have shown much improved results over the original EANT method for developing a visuo-motor controller in the 3 DOF scenario. It could also be seen that varying the CPU time available to the CMA-ES-based parameter optimisation in the exploitation part by a factor of 10 did not significantly influence the performance of the resulting networks. Their performance is also very much better than that of the traditional approach for visuo-motor control.

Our experimental results show that the new EANT method with CMA-ES is capable of learning neural networks as solutions to complex and difficult problems. The CMA-ES based EANT can be used as a "black-box" tool to develop networks without being given much information about the nature of the problem. It also does not require a lot of parameter tuning to give useful results. The resulting networks show a very good performance.

Acknowledgements

The authors wish to thank Nikolaus Hansen, the developer of CMA-ES, for kindly providing source code which helped us to quickly start integrating his method into EANT.

References

1. Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.
2. Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, San Francisco, USA, 1998.
3. Richard Ernest Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, USA, 1961.
4. Andrea Beltratti, Sergio Margarita, and Pietro Terna. *Neural Networks for Economic and Financial Modelling.* International Thomson Computer Press, London, UK, 1996.
5. Chris C Bissell. *Control Engineering.* Number 15 in Tutorial Guides in Electronic Engineering. CRC Press, Boca Raton, USA, 2nd edition, 1996.
6. Wolfram Blase, Josef Pauli, and Jörg Bruske. Vision-based manipulator navigation using mixtures of RBF neural networks. In *International Conference on Neural Network and Brain*, pages 531–534, Bejing, China, April 1998.
7. Ágoston E Eiben and James E Smith. *Introduction to Evolutionary Computing.* Springer Verlag, Berlin, Germany, 2003.
8. Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, Pittsburgh, USA, August 1991.
9. Roger Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, Chichester, 2nd edition, 1987.
10. Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
11. Koichi Hashimoto, editor. *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*, volume 7 of *Series in Robotics and Automated Systems.* World Scientific Publishing Co., Singapore, 1994.
12. Gilles Hermann, Patrice Wira, and Jean-Philippe Urban. Neural networks organizations to learn complex robotic functions. In *Proceedings of the 11th European Symposium on Artificial Neural Networks (ESANN 2003)*, pages 33–38, Bruges, Belgium, April 2005.
13. Kurt Hornik, Maxwell B Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
14. Seth Hutchinson, Greg Hager, and Peter Corke. A tutorial on visual servo control. Tutorial notes, Yale University, New Haven, USA, May 1996.
15. William R Hutchison and Kenneth R Stephens. The airline marketing tactician (AMT): A commercial application of adaptive networking. In *Proceedings of the 1st IEEE International Conference on Neural Networks, San Diego, USA*, volume 2, pages 753–756, 1987.

16. Martin Jägersand. Visual servoing using trust region methods and estimation of the full coupled visual-motor Jacobian. In *Proceedings of the IASTED Applications of Control and Robotics, Orlando, USA*, pages 105–108, January 1996.
17. Jae-Yoon Jung and James A Reggia. A descriptive encoding language for evolving modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 519–530. Springer Verlag, 2004.
18. Yohannes Kassahun and Gerald Sommer. Automatic neural robot controller design using evolutionary acquisition of neural topologies. In *19. Fachgespräch Autonome Mobile Systeme (AMS 2005)*, pages 259–266, Stuttgart, Germany, December 2005.
19. Yohannes Kassahun and Gerald 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.
20. Scott Kirkpatrick, Charles Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
21. James W Melody. On universal approximation using neural networks. Report from project ECE 480, Decision and Control Laboratory, University of Illinois, Urbana, USA, June 1999.
22. Tom M Mitchell. *Machine Learning*. McGraw-Hill, London, UK, 1997.
23. David E Moriarty and Risto Miikkulainen. Evolving obstacle avoidance behavior in a robot arm. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, Cape Cod, USA, 1996.
24. Arnold Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, June 2004.
25. Apostolos-Paul Refenes, editor. *Neural Networks in the Capital Markets*. John Wiley & Sons, New York, Chichester, USA, 1995.
26. Claude Robert, Charles-Daniel Arreto, Jean Azerad, and Jean-François Gaudy. Bibliometric overview of the utilization of artificial neural networks in medicine and biology. *Scientometrics*, 59(1):117–130, 2004.
27. Raúl Rojas. *Neural Networks - A Systematic Introduction*. Springer Verlag, Berlin, Germany, 1996.
28. James C Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, Hoboken, USA, 2003.
29. Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
30. Robert R Trippi and Efraim Turban, editors. *Neural Networks in Finance and Investing*. Probus Publishing Co., Chicago, USA, 1993.
31. Jean-Philippe Urban, Jean-Luc Buessler, and Julien Gresser. Neural networks for visual servoing in robotics. Technical Report EEA-TROP-TR-97-05, Université de Haute-Alsace, Mulhouse-Colmar, France, November 1997.
32. Lee E Weiss, Arthur C Sanderson, and Charles P Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal of Robotics and Automation*, 3(5):404–417, October 1987.
33. Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.
34. Xin Yao and Yong Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

35. Michael Zeller, Kenneth R Wallace, and Klaus Schulten. Biological visuo-motor control of a pneumatic robot arm. In Cihan Hayreddin Dagli, Metin Akay, C L Philip Chen, Benito R Fernandez, and Joydeep Ghosh, editors, *Intelligent Engineering Systems Through Artificial Neural Networks. Proceedings of the Artificial Neural Networks in Engineering Conference, New York*, volume 5, pages 645–650. American Society of Mechanical Engineers, 1995.