Efficient Neural Network Pruning during Neuro-Evolution

Nils T Siebel, Jonas Bötel, Gerald Sommer

Abstract— **In this article we present a new method for the pruning of unnecessary connections from neural networks created by an evolutionary algorithm (neuro-evolution). Pruning not only decreases the complexity of the network but also improves the numerical stability of the parameter optimisation process. We show results from experiments where connection pruning is incorporated into EANT2, an evolutionary reinforcement learning algorithm for both the topology and parameters of neural networks. By analysing data from the evolutionary optimisation process that determines the network's parameters, candidate connections for removal are identified without the need for extensive additional calculations.**

I. Introduction

ARTIFICIAL Neural Networks have been the object of research for several decades. Originally inspired by the way our brain works, these "neural networks" have become a powerful tool for function approximation (usually regression analysis), classification and data processing. Based on the Hahn-Banach theorem, Cybenko's theorem proved in 1989 that the single-hidden-layer perceptron can act as a universal function approximator [1]. This proves the versatility of neural networks, and they have indeed been successfully applied to problems in the sciences, engineering and even economics [2], [3], [4], [5], [6].

While there is a common understanding of the underlying principles and mathematics, there is still no straightforward way to *construct* a neural network that solves a given task. In many cases creating a good network requires a great deal of domain knowledge and manual intervention, e.g. to determine the network's topology ("structure"), or to adjust the parameters of one's learning algorithm ("hyperparameters") to the given problem and data. Even with manual intervention and tuning, this may still be difficult or even impossible if the problem is non-trivial.

Most of the past research work has been solely on learning the parameters of a neural network; there are few constructive algorithms for a neural network's topology. Our goal is to develop such an algorithm, one which creates compact neural networks that solve a given task without requiring problem-specific tuning (parameterless).

Our method is called EANT2, "Evolutionary Acquisition of Neural Topologies Version 2" [7]. EANT2 constructs neural networks (both topology and parameters) for a given problem by reinforcement learning. It has been successfully applied to learning a robot control *(ibid.)* and classification tasks in image processing [8], [9] without setting any problem-specific parameters (apart from the number of network in- and outputs).

The authors are with the Cognitive Systems Group, Institute of Computer Science, Christian-Albrechts-University of Kiel, Germany (e-mail: {nts,jboe,gs}@ks.informatik.uni-kiel.de).

In this article we present a new variant of EANT2 which automatically detects and removes insignificant connections in a neural network. In contrast to existing methods for neural network pruning our approach works in our reinforcement learning scenario where no data pairs—and hence, no derivative of the network training function—is available. By using existing data from the network parameter optimiser, candidate connections for removal are also identified without the need for extensive additional calculations. The effects of pruning are studied in detail in robot learning experiments.

The remainder of the article is organised as follows. Section II introduces the terminology and describes related work. Details on our neuro-evolutionary method can be found in Section III. The pruning approach is described in Section IV and validated by experiments in Section V. Section VI concludes the article.

II. Preliminaries and Related Work

A. Neural Network Learning Paradigms

A neural network can be regarded as a function f that maps data points/vectors x from an input space $X \subseteq \mathbb{R}^n$ to vectors y in an output space $Y \subseteq \mathbb{R}^m$, i.e. $f : X \to Y$, $x \mapsto y$. For a neural network with a fixed topology this function f is parameterised by the parameters of the network, e.g. the values of synaptic weights. *Training* a neural network means to optimise these parameters such that the network is suitable for a given task. For this the optimisation process needs a measure of this suitability of given parameters, usually expressed by an *error* or *cost function* which is to be minimised during training. The main training/learning paradigms for neural networks are supervised learning, unsupervised learning and reinforcement learning.

1) Supervised Learning: Here a set of example data pairs $\{(x_i, y_i)\}_i$, $x_i \in X, y_i \in Y\ \forall i$, is given. The goal is to find a function $f : X \to Y$ (here, a neural network) that describes the mapping implied by the data points. The cost function is related to the mismatch between our mapping and the data. In classification problems, y_i is the *label* of the point x_i.

The most commonly used cost function is the mean-squared error (MSE), i.e. the mean squared difference between the network's output, $f(x_i)$, and its target value y_i, over all example pairs. A popular algorithm for the minimisation process is the *backpropagation algorithm* [10, chap. 4], which is essentially optimisation by stochastic gradient descent.

2) Unsupervised Learning: As in the supervised case, we are given data examples $\{x_i\}_i$, but not as pairs $\{(x_i, y_i)\}_i$. One form of unsupervised learning is clustering, further examples are the estimation of statistical distributions of data and blind source separation (e.g. based on Independent Component Analysis, ICA). Examples for unsupervised learning approaches by neural networks are Self-Organising Maps (SOM) and Adaptive Resonance Theory (ART) systems.

3) Reinforcement Learning (RL): In RL scenarios, data x is usually not given. Instead, the algorithm evaluates a candidate solution by direct interaction with the environment. One example would be a robot controller. A given network can move the robot at each time instance t by an action y_t which is based on sensor data x_t. The environment generates an observation o_t and often also an instantaneous cost $C(o_t)$, according to the (usually unknown) dynamics of the system. The aim is to discover a policy for selecting actions that minimises a measure of a long-term cost, i.e. the expected cumulative cost.

RL differs from supervised learning in that correct input/output pairs are never presented, nor are sub-optimal actions explicitly corrected. While this lack of information makes RL more flexible in its application it also means that the algorithm has no immediate hint in which direction to move in a (possibly very high-dimensional) search space. Therefore one focus is always on performance, which involves finding a balance between *exploration* (of uncharted territory) and *exploitation* (of current knowledge). The efficiency of a RL algorithm can be measured in the number of evaluations of the cost function (often short "fevals" for "function evaluations") it needs to find a good solution.

B. Pruning Neural Networks

Pruning, if done well, solves two important neural network learning problems:

- *Unnecessary connections:* Often not all network connections contribute to the solution, or they may create insignificant parallel data flow. This can generate numerical problems (ill-conditioning) [11] and lead to overfitting [12].
- *Large number of parameters:* With the number of parameters of a network the difficulty to determine them increases exponentially ("curse of dimensionality") [13]

There are several approaches for neural network pruning:

1) Methods Based on Absolute Parameter Values: The simplest method for pruning is the removal of the connection with the smallest connection weight [10]. The idea is that a small parameter value p_i can be well approximated by 0, which is equivalent to removing the corresponding connection altogether. In practise, however, it may turn out not to be true; in fact small parameters can still have a large influence on the behaviour of the network *(ibid.)*.

2) Optimal Brain Damage (OBD): This is an improved variant of the previous class of methods by LeCun *et al.* [14]. Instead of using absolute value $|p_i|$ OBD examines the 2nd derivative of the cost function w.r.t. p_i, i.e. the influence of p_i on the cost. Let $c(p_1, \dots, p_l)$ be the cost as a function of the network parameters. Then its Taylor expansion is

$$c(p+\delta) = c(p) + J_c(p)\delta + \frac{1}{2}\delta^T H_c(p)\delta + O(\|\delta\|^3). \quad (1)$$

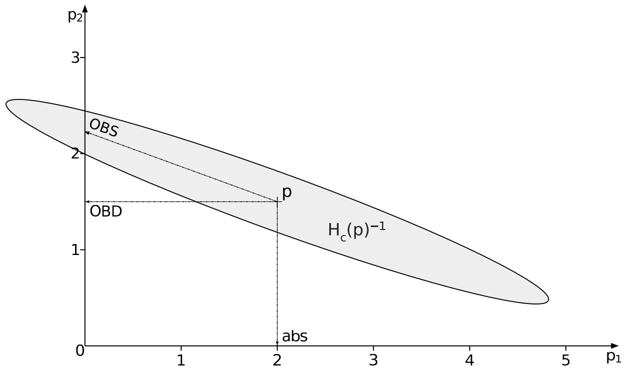


Fig. 1. Principles underlying popular pruning methods

Now, this method selects the connection for removal whose parameter p_i has the smallest corresponding value $|h_{ii}|$ in the Hessian $H_c(p) =: (h_{ij})_{i,j}$. As an added (or maybe, motivating) benefit of this selection the condition κ of the Hessian $H_c(p)$ can decrease, and hence, the numerical stability of the parameter optimisation process improve. The reason is that $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$, where λ_{max} and λ_{min} are the largest and smallest Eigenvalues of $H_c(p)$. However, this is only the case if the h_{ii} really are the Eigenvalues of $H_c(p)$, i.e. $H_c(p)$ is diagonal. Also, ignoring the term $O(\|\delta\|^3)$ in(1) may or may not yield a good approximation, depending on the structure of the data. It is a reasonable assumption in the vicinity of a local minimum [15], however, elsewhere this may not be the case. Hassibi and Stork also point out in their experiments $H_c(p)$ was seldom close to diagonal [16], which would mean that OBD removes the wrong connections.

3) Optimal Brain Surgeon (OBS): Based on these possible disadvantages of OBD, Hassibi and Stork have suggested a new method, OBS *(ibid.)*. In order to determine insignificant network connections the following measure is introduced for the ith connection, with parameter p_i:

$$s_i := \frac{1}{2}\frac{p_i^2}{h_{ii}^{-1}}, \quad (2)$$

where h_{ij}^{-1} are the entries of the inverse Hessian matrix, $H_c(p)^{-1}$. The algorithm removes the connection i' with the smallest value $s_{i'}$. In addition, the current parameter vector p is shifted so as to make up for the missing connection, prior to pruning (the removal is equivalent to setting $p_{i'} := 0$):

$$p' := p - \frac{p_{i'}}{h_{i'i'}^{-1}} H_c(p)^{-1} e_{i'}, \quad (3)$$

where e_i is the ith unit vector.

Figure 1 demonstrates all three presented principles for 2 parameters, $p_1 = 2$ and $p_2 = 1.5$. $H_c(p)^{-1}$ is visualised here by an ellipse, centred around the current point p, whose main axis correspond to its Eigenvectors, scaled by the Eigenvalues. The contour of the ellipse corresponds to identical values of the cost function c. In this situation the absolute value method, "abs" in the diagram, would set $p_2 := 0$. OBD would set $p_1 := 0$, thereby reducing the change to $c(p')$ in comparison. OBS would also set $p_1 := 0$

but also increase p_2 slightly to further reduce the influence on $c(p')$.

Kavzoğlu and Mather have shown that, just as one would expect, OBS preforms better than OBD, which in turn preforms better than "abs" [12]. However, a remaining problem is the necessity to estimate the diagonal of $H_c(p)$ (OBD) or even the complete matrix $H_c(p)^{-1}$ (OBS), which can be computationally very expensive unless the network is small.

C. Evolutionary Ways of Creating Neural Networks

Up to the late 90s only small neural networks have been evolved by evolutionary algorithms [17]. According to Yao, a main reason is the difficulty of evaluating the exact fitness (negative cost) 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.)*.

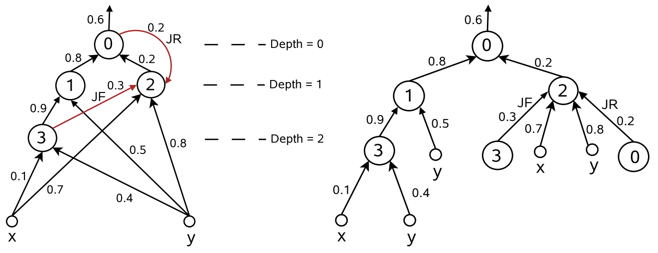
Most approaches evolve the structure and parameters of the neural networks simultaneously. Examples are EPNet [18], GNARL [19] and NEAT [20]. EPNet uses a modified backpropagation algorithm for parameter optimisation (a local method). Mutation operators for searching the space of neural structures are addition and deletion of neurons and connections (no crossover is used). EPNet has a tendency to remove connections/nodes rather than to add new ones. This is done to counteract "bloat" (i.e. ever growing networks with only little fitness improvement; called "survival of the fattest" in [21]). GNARL also does not use crossover during structural mutation. However, it uses an evolutionary algorithm for parameter optimisation. 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 [22]. In order to calculate the current temperature, the algorithm needs some knowledge about the "ideal solution" to the problem, e.g. the best fitness expected to be reached.

NEAT, unlike EPNet and GNARL, uses a crossover operator that allows to produce valid offspring from two given neural networks. It works by first aligning similar or equal subnetworks and then exchanging differing parts. Like GNARL, NEAT uses evolutionary algorithms 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 [21].

III. Learning Neural Networks with EANT2

A. The Algorithm

EANT2, "Evolutionary Acquisition of Neural Topologies Version 2", is an evolutionary reinforcement learning system that realises neural network learning with evolutionary algorithms both for the structural and the parametrical part. It is based on the previous method EANT [23] but uses different algorithms for structural mutation and parameter optimisation [24]. EANT2 represents neural networks and their parameters in a compact genetic encoding, the "linear genome". It encodes the topology of the network implicitly by the order of its elements (genes). The following basic gene types exist: neurons, network inputs, biases and forward connections. There are also "irregular" connections between neural genes which we call "jumper connections". Jumper genes can encode either forward or recurrent connections. Figure 2 shows 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 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, `x` and `y` are the inputs coded by input genes. 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.



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

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

(c) Corresponding Linear Genome

Fig. 2. An example of encoding a neural network using a linear genome

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 another neuron. Connection ("jumper") genes make it possible for neuron outputs to be used as input to more than one neuron, see `JF3` in the example above. Together with bias neurons the linear genome can encode any neural network in a very compact format; its length is equal to the number of synaptic network weights.

The steps of our algorithm, shown in Figure 3, are ex-

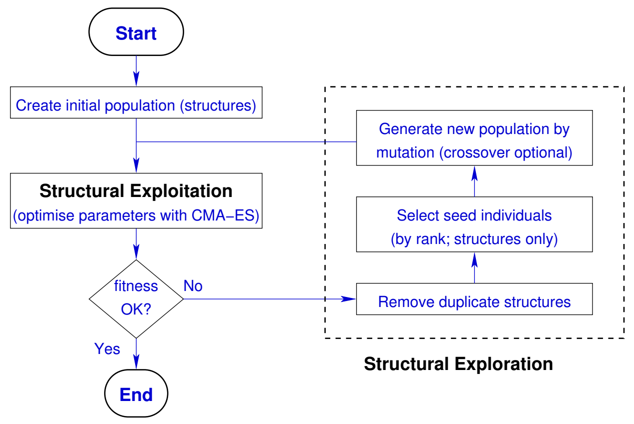

Start
Create initial population (structures)
Structural Exploitation
(optimise parameters with CMA–ES)
fitness OK?
No
Yes
End
Generate new population by mutation (crossover optional)
Select seed individuals (by rank; structures only)
Remove duplicate structures
Structural Exploration

Fig. 3. The original EANT2 algorithm.

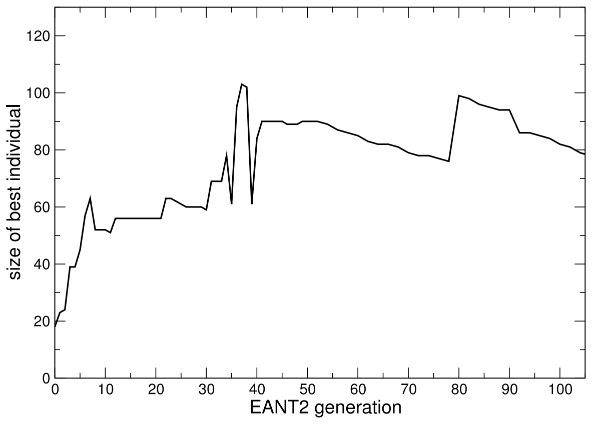


Fig. 4. Size development, EANT2 without pruning.

plained in detail below.

Initialisation: EANT2 usually starts with minimal initial structures. A minimal network has no hidden layers or recurrent connections, only 1 neuron per output, connected to some or all inputs. EANT2 gradually develops these simple initial structures further using the structural and parametrical evolutionary algorithms discussed below. On a larger scale new neural structures are added to a current generation of networks. We call this “structural exploration”. On a smaller scale the current structures are optimised by changing their parameters: “structural exploitation”.

Structural Exploitation: At this stage the structures in the current EANT2 population are exploited by optimising their parameters. Parametrical mutation is realised using CMA-ES (“Covariance Matrix Adaptation Evolution Strategy”) [25]. CMA-ES is a variant of Evolution Strategies that avoids random adaptation of strategy parameters. Instead, the search area spanned by the mutation strategy parameters, expressed by a covariance matrix, is adapted at each step depending on the current population. CMA-ES uses sophisticated methods to avoid problems 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.)*. It has been first successfully applied to reinforcement learning of neural network weights by Igel [26].

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. In order to maintain diversity in the population, it 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.

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 or removing a random subnetwork, adding or removing a random connection and adding a random bias. New hidden neurons are connected to approx. 50 % of inputs; the exact percentage and selection of inputs are random.

B. Comparison with Other Methods

EANT2 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. Further differences of EANT2 to other recent methods, e.g. NEAT, are a small number of user-defined algorithm parameters (the method should be as universal as possible) and the explicit way of preserving diversity in the population (unlike speciation).

In the past we have compared EANT2 with NEAT by applying both algorithms to the same problem [7]. The test environment was a visual servoing task run in a simulation. Both methods were to develop neural networks to control a robot in 3 degrees of freedom in order to align its gripper to an object. The robot movement was determined based on 10 image measurements, so the networks had 10 inputs and 3 outputs. The results showed that NEAT had more problems than EANT2 finding good parameters for given networks. EANT2 was at a clear advantage in this comparison. More details can be found in [7].

IV. Our Approach to Neural Network Pruning

During our experiments with EANT2 we often encountered difficult numerical conditions during parameter optimisation. This was mainly due to redundancy in the input data (10 inputs describe an essentially 4-dimensional problem)[1] but also due to parallel data flow in the network, which both lead to ill-conditioning.

[1]A transformation of the input data, e.g. by PCA, would defy our goal to develop a method that can work with the task’s natural problem formulation.

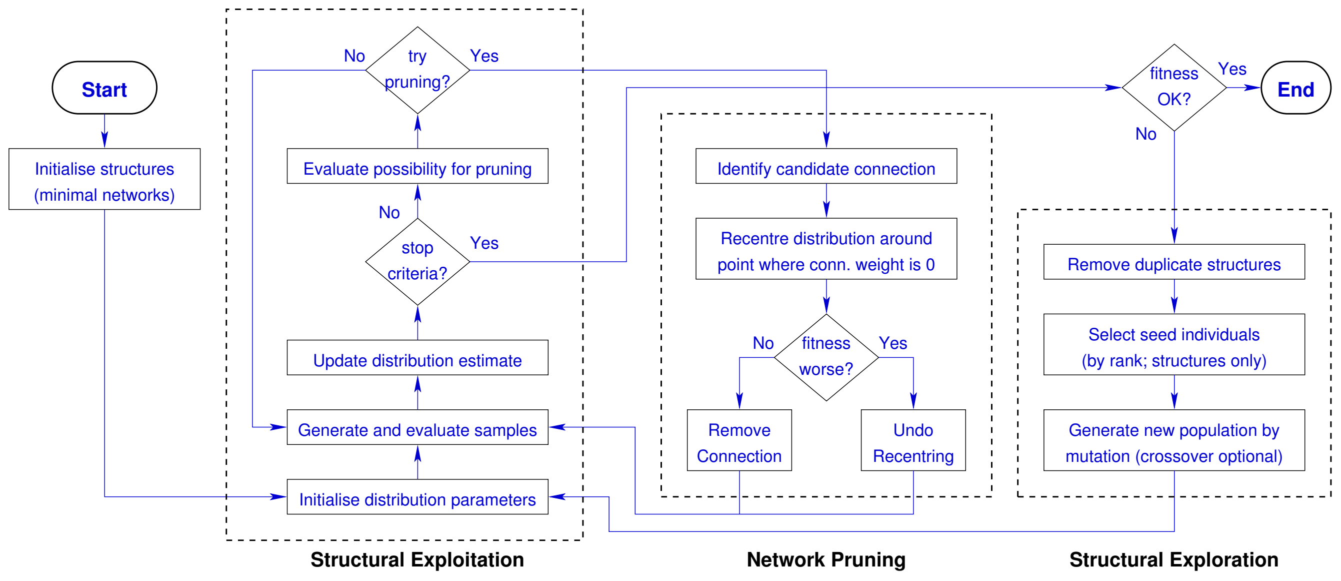

Start
Initialise structures (minimal networks)
No
try pruning?
Yes
Evaluate possibility for pruning
No
stop criteria?
Yes
Update distribution estimate
Generate and evaluate samples
Initialise distribution parameters
Structural Exploitation
Identify candidate connection
Recentre distribution around point where conn. weight is 0
No
fitness worse?
Yes
Remove Connection
Undo Recentring
Network Pruning
fitness OK?
Yes
End
No
Remove duplicate structures
Select seed individuals (by rank; structures only)
Generate new population by mutation (crossover optional)
Structural Exploration

Fig. 5. The new EANT2 algorithm with pruning during structural exploitation (i.e. neural network parameter optimisation).

In long-term experiments with EANT2 it could also be seen that the size slowly decreases over long time spans, see Figure 4. What happened is that from time to time neurons with a few connections were added so that the size of the best performing network increased. In most other EANT2 generations, however, connections were removed by random mutation, and the smaller network selected by the evolutionary algorithm if the fitness stayed the same. The slowly decreasing network sizes in the plot indicate that many redundant connections were present, and removed in this manner. Random removal of connections decreases the number of network parameters and thus makes the their optimisation easier (compare "curse of dimensionality"). However, it is inefficient as it takes a lot of time.

In order to reduce the problem dimension more quickly and at the same time alleviate the numerical difficulties we developed a derandomised pruning module for EANT2.

A. Main Idea

The reasoning behind our pruning strategy is similar to the OBS method above. The Hessian matrix of the cost function, $H_c(p)$, expresses the significance of the parameters p for the task. However, the calculation of $H_c(p)$ is impossible in a reinforcement learning scenario, and its approximation by finite differences is computationally expensive—especially so in our robot learning case where an evaluation of the cost function requires the simulation of 1023 robot movements.

Our advantage is that in our structural exploitation, we use CMA-ES [25] for the optimisation of the parameters. CMA-ES adapts a covariance matrix C of the parameters, which represents the local structure of the search space. When this adaptation converges, the matrix approximates the inverse Hessian matrix $H_c(p)^{-1}$ well (up to a constant factor) *(ibid.)*.

The main idea for pruning is that when parameters are correlated (i.e. their covariance is large), one of them can be removed. For example, if p_i and p_j are highly correlated but uncorrelated to other parameters, p_i can be locally changed such that a correctly correlated change in p_j cancels the influence of p_i's change on the cost/fitness function. This is valid in the area around the current mean parameter vector m where C describes the structure of the search space well. If this area includes the ith (or jth) axis then the parameter p_i (or p_j) can be set to 0—and hence, removed. The same argument is valid if more than two parameters are involved.

B. Implementation

1) Detecting Convergence: Before using measurements from the covariance matrix C to detect the significance of the network parameters p one needs to make sure that the underlying assumptions for the abovementioned reasoning are true:

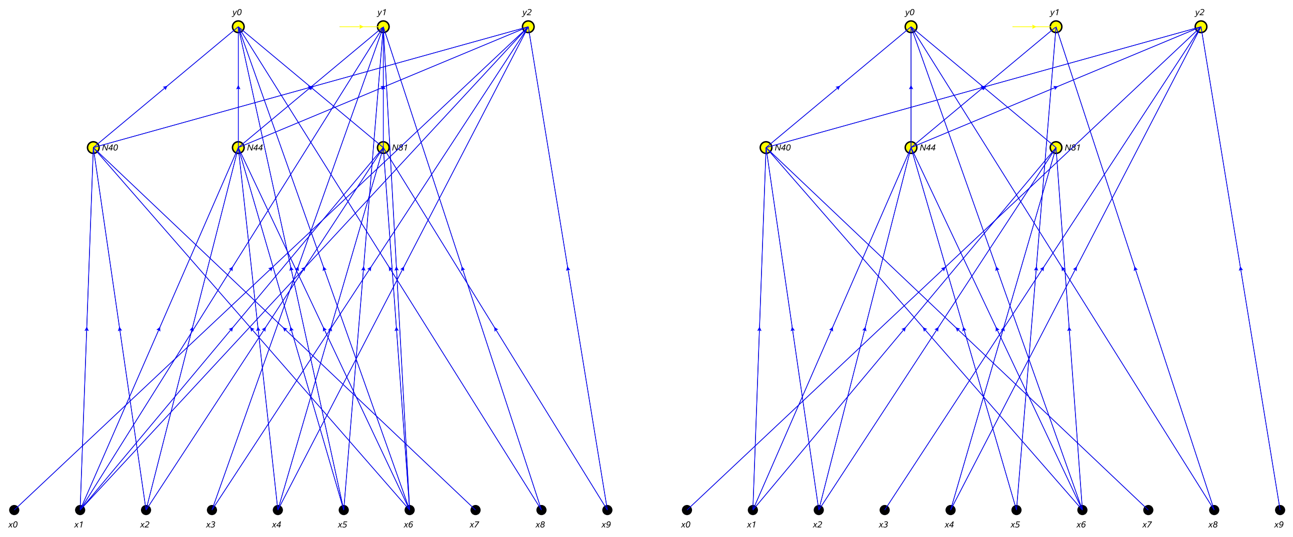
1) $c(p + \delta) \approx c(p) + \frac{1}{2}\delta^T H_c(p)\delta$ for small δ and
2) $C \approx H_c(p)^{-1}$.

We know that the first assumption is valid close to a local optimum of the cost function [15]. This is where the second assumption is also valid [25][2]. In order to detect this closeness to a local optimum we use a method already implemented in CMA-ES: the internal convergence detection (which analyses the evolution path) calculated for the adaptation of the step size σ. σ is used to scale the covariance matrix so as to speed up the movement in search space far away from the optimum and slow it down in its vicinity. New individuals (candidate solutions for parameters p) are generated by sampling from the normal distribution

$$p \sim \mathcal{N}(m, \sigma^2 C) \tag{4}$$

where m is the mean of the current population.

[2]In fact, our experiments showed that $C \approx H_c(p)^{-1}$ is valid much earlier during the adaptation of C.



(a) Before pruning

(b) After pruning

Fig. 6. Structural changes to a network by pruning

Our experiments have confirmed that monitoring the development of the value of σ is indeed a good way to detect convergence. In practise the development of σ is slightly smoothed over time by updating a running mean $\bar{\sigma}_k$ in the kth iteration as follows:

$$\bar{\sigma}_k := \lambda\sigma_k + (1-\lambda)\bar{\sigma}_{k-1} \tag{5}$$

with a value of $\gamma \in [.025; .01]$, smoothing more strongly when the dimension of p is large.

Using $\bar{\sigma}_k$, we initiate a pruning attempt when $\bar{\sigma}_k$ reaches a local minimum, i.e. it has decreased for a number of iterations (e.g., 250) and then increases again.

2) Identifying Candidate Connections for Pruning: In order to examine the significance of parameters (to find one which is not significant) we use the components of the Eigendecomposition of C, which are already available due to the way C is adapted:

$$C = BD^2B^T, \quad B = (b_{ij})_{i,j=1,\dots,l}, \quad D = (d_{ij})_{i,j=1,\dots,l}, \tag{6}$$

where D is a diagonal matrix holding the square roots of the Eigenvalues of C and l is the dimension of the parameter space as above.

Let m be again the mean of the current distribution and let v_j be C's jth principal axis, $v_j := b_j d_{jj}$ where b_j shall denote the jth column of B. Then for each parameter index i and each axis j we calculate the scaling factor s_{ij} by which v_j would need to be scaled so that $m + s_{ij}v_j$ reaches the hyperplane where $p_i = 0$. These values are given by

$$s_{ij} = -\frac{m_i}{b_{ij}d_{jj}} \tag{7}$$

since then $\widetilde{m}_i = m_i - s_{ij} \cdot b_{ij}d_{jj} = 0$ for $\widetilde{m} := m + s_{ij} \cdot v_j$.

Among these we select indices $i^\star$ and $j^\star$ that correspond to the smallest factor $|s_{ij}|$,

$$(i^\star, j^\star) \in \underset{(i,j)\in[1,\dots,l]^2}{\operatorname{argmin}} |s_{ij}| \tag{8}$$

and shift the parameters in space by setting $m := \widetilde{m}^\star$ with $\widetilde{m}^\star := m + s_{i^\star j^\star} \cdot v_{j^\star}$.

3) Evaluation of the new Structure: It is to be expected that the value of the cost function will change (probably, increase) when the parameters are shifted in space by $s_{i^\star j^\star} v_{j^\star}$. This is checked by calculating the new value of the cost function $\tilde{e}$ and comparing it to the original value e. The shift (and thereby, the removal of connection $i^\star$) is accepted if the cost is increased by less than 0.1 %, i.e. iff $\tilde{e} < 1.001\, e$.

The structure of the new algorithm is shown in Figure 5, as an expanded and amended version of the diagram in Figure 3 above.

V. Main Results

A. Initial Experiments

In our initial experiments we examined the change of the numerical condition of the covariance matrix by the removal of connections, since it was one of our goals to improve it. We started with a structure of size 38 and pruned with our algorithm, which removed 8 connections, see Figure 6, without changing the fitness significantly.

Then both networks were initialised with random parameters before optimising it with our structural exploitation. This time no further pruning was applied. Figure 7 shows the development of the condition of the covariance matrix over time, for these two networks. It can be seen that the condition of the original network reaches 1.8e+17 at which point the condition was limited to avoid numerical problems.

The condition of the pruned individual, on the other hand, did not exceed 9e+13.

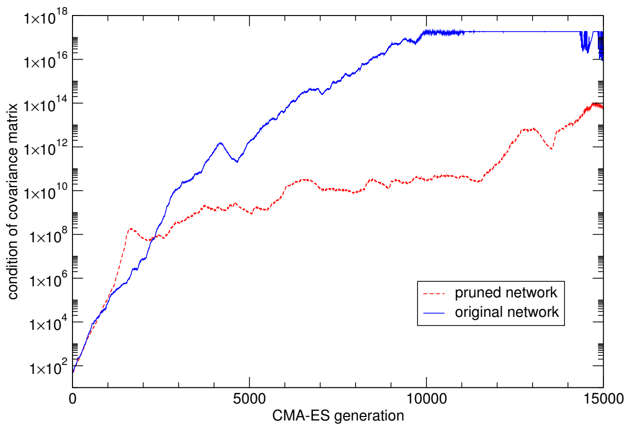


Fig. 7. Development of condition, original and pruned network

B. 3 DOF Visual Servoing

After these encouraging initial experiments we started our main experiments, where EANT2 was asked to develop robot controllers, sometimes with pruning enabled, sometimes disabled. The networks needed to steer a robot arm in 3 degrees of freedom based on 10-dimensional image data, which means the networks had 10 in- and 3 outputs. More details on the test setup can be found in [7]. We made 5 runs each with pruning enabled and disabled.

Figure 8 shows the development of the fitness value (negative visual error after robot movement) and the network size of the best individual over time. It can be seen that the fitness reached by both variants of EANT2 after 15 generations is very similar, and within the natural variation one expects with an evolutionary algorithm. On average, the fitness in generation 15 was -0.227654 with pruning and -0.226681 without. It is also apparent that the runs with pruning tend to reach their final fitness value more quickly.

The plots of the sizes of the neural networks vary over time since in each generation the size of the best individual (selected by fitness) is plotted, and this selection may cause some jitter in the values. Nevertheless it is very clear that pruning significantly reduced the size of the networks; the mean sizes in generation 15 were 59.8 with pruning and 73.6 without pruning—an average of size reduction of 23 %.

VI. Conclusions

The goal was to develop a method for neural network pruning that would work with our evolutionary reinforcement learning method EANT2. Standard methods like Optimal Brain Damage and Optimal Brain Surgeon do not work in reinforcement learning setups, and an approximation of the Hessian matrix of the cost function would be too computationally expensive.

Our approach uses measurements from the covariance matrix which is part of CMA-ES, the evolutionary algorithm that optimises the network parameters. We detect at which time the covariance matrix converges against the inverse of the Hessian and the optimiser is also at a local minimum. In this situation we can use the existing Eigendecomposition of the covariance matrix to predict for each parameter how much its removal would influence the cost function. The connection with the smallest influence is then removed by shifting all parameters in the search space such that their mean lies on the hyperplane where the parameter is 0. Then the corresponding connection is removed. If this shift influences the cost significantly this change is reverted, otherwise it is kept.

Initial tests of the new method show that pruning removes several connections of given networks that were developed by EANT2 without pruning. At the same time the condition of the covariance matrix is also significantly reduced, which makes the parameter optimisation numerically more stable.

For the main tests EANT2 was given the task of learning a robot controller with 10 network inputs and 3 outputs. It could be seen that the pruning-enabled EANT2 creates networks that have up to a quarter fewer connections, while having on average the same fitness values as their non-pruning counterparts.

To conclude, our experiments have shown that the new pruning module for EANT2 helps to generate compact neural networks that show the same performance as the networks created by the standard EANT2 while improving the numerical conditions during parameter optimisation.

References

[1] G. Cybenko, “Approximation by superposition of sigmoidal functions,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, December 1989.

[2] A. Beltratti, S. Margarita, and P. Terna, *Neural Networks for Economic and Financial Modelling*. London, UK: International Thomson Computer Press, 1996.

[3] W. R. Hutchison and K. 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*, vol. 2, 1987, pp. 753–756.

[4] A.-P. Refenes, Ed., *Neural Networks in the Capital Markets*. New York, Chichester, USA: John Wiley & Sons, 1995.

[5] C. Robert, C.-D. Arreto, J. Azerad, and J.-F. Gaudy, “Bibliometric overview of the utilization of artificial neural networks in medicine and biology,” *Scientometrics*, vol. 59, no. 1, pp. 117–130, 2004.

[6] R. R. Trippi and E. Turban, Eds., *Neural Networks in Finance and Investing*. Chicago, USA: Probus Publishing Co., 1993.

[7] N. T. Siebel and G. Sommer, “Evolutionary reinforcement learning of artificial neural networks,” *International Journal of Hybrid Intelligent Systems*, vol. 4, no. 3, pp. 171–183, October 2007.

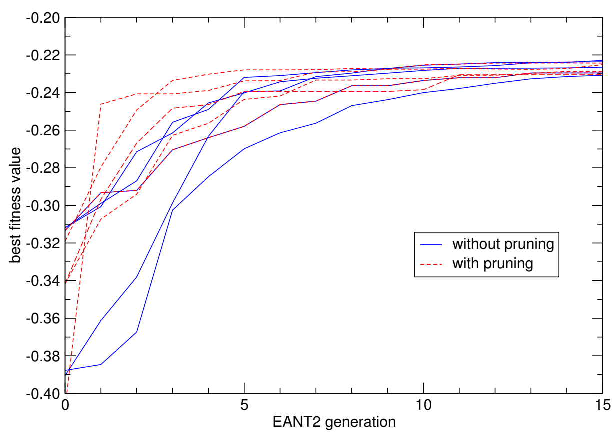
[8] ——, “Learning defect classifiers for visual inspection images by neuro-evolution using weakly labelled training data,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong, China*, June 2008, pp. 3926–3932.

[9] N. T. Siebel, S. Grünewald, and G. Sommer, “Creating edge detectors by evolutionary reinforcement learning,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong, China*, June 2008, pp. 3552–3559.

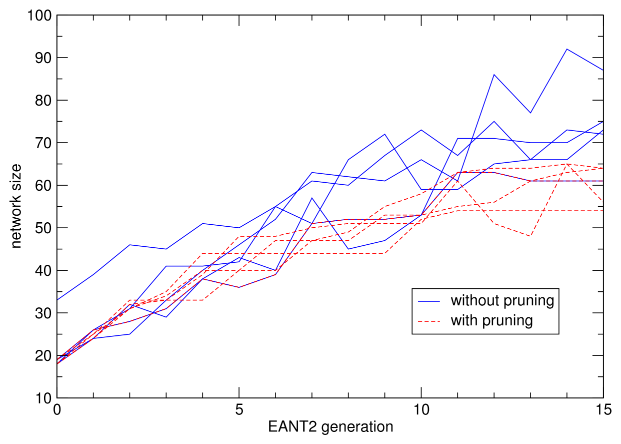
[10] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995.

[11] W. S. Sarle, “Ill-conditioning in neural networks,” Website, SAS Institute Inc., Cary, USA, September 1999, ftp://ftp.sas.com/pub/neural/illcond/illcond.html.

[12] T. Kavzoğlu and P. M. Mather, “Assessing artificial neural network pruning algorithms,” in *Proceedings of the 24th Annual Conference and Exhibition of the Remote Sensing Society (RSS 1998)*, Greenwich, UK, 1998, pp. 603–609.



(a) Fitness development



(b) Size development

Fig. 8. Comparison of 5 EANT2 runs each with/without pruning: development of fitness and size over EANT2 generation

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

[14] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems (NIPS 1990)*, 1990, pp. 598–605.

[15] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, 3rd ed. Springer Verlag, 2008.

[16] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proceedings of the 1993 Conference on Advances in Neural Information Processing Systems (NIPS 1993)*, 1993, pp. 164–171.

[17] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, September 1999.

[18] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, May 1997.

[19] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[20] K. O. Stanley and R. P. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

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

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[23] 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)*, Bruges, Belgium, April 2005, pp. 259–266.

[24] 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*, December 2006, p. 6 (4 pages).

[25] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[26] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*. IEEE Press, 2003, pp. 2588–2595.