DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch028

Neural agents can evolve to reproduce sequences of arbitrary length

Benjamin Inden [1] and Jürgen Jost[2,3]

[1]Artificial Intelligence Group, Bielefeld University, Bielefeld, Germany
binden@techfak.uni-bielefeld.de
[2]Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany
jost@mis.mpg.de
[3]Santa Fe Institute, Santa Fe, New Mexico, USA

Abstract

We demonstrate that neural agents can evolve behavioral sequences of arbitrary length. In our framework, agents in a two-dimensional arena have to find the secure one among two possible patches, and which of them is secure changes over time. Evolution of arbitrarily long behavioral sequences is achieved by extending the neuroevolution method NEAT with two techniques: Only newly evolved network structure is subject to mutations, and inputs to the neural network are provided in an incremental fashion during evolution. It is suggested that these techniques are transferable to other neuroevolution methods and domains, and constitute a step towards achieving open-ended evolution. Furthermore, it is argued that the proposed techniques are strongly simplified models of processes that to some degree occur naturally in systems with more flexible genetic architectures.

Introduction

Evolutionary robotics is an approach towards the design of controllers (and possibly also bodies) of robots that uses a process of artificial evolution. The controllers of the resulting robots are often in the form of artificial neural networks. It is conceivable that eventually robots with very complex goal-directed behaviors can be designed using this method. However, past work in the field has typically produced behaviors that are very simple as compared to behaviors of conventionally designed robot controllers.

Complex behaviors can arise by the interplay of environment, body, and controller even if the controller itself is not very complex (Pfeifer and Gómez, 2005). Nevertheless, within a given environment, there is obviously a correlation between the complexity of the controller and the complexity of the behavior. Therefore, it is desirable to have methods for the evolution of neural networks that can make the networks more and more complex over time.

For many of the earliest evolutionary robotics experiments, neural networks with a fixed topology, i.e., a fixed number of nodes and connections, were used (Nolfi and Floreano, 2000). That way, the achievable complexity is obviously limited. Later, a method called NEAT was introduced (Stanley and Miikkulainen, 2002). This neuroevolution method starts evolution using networks without any hidden nodes and subsequently adds neurons and connections by carefully designed mutation operators. It has been shown that complexification during evolution does indeed occur when using NEAT, and can lead to neural networks with in the order of, say, 10 to 20 hidden nodes (Stanley and Miikkulainen, 2004). NEAT has subsequently been widely used for various evolutionary robotics experiments.

Of course, ultimately one needs more complexity than that evolved by NEAT and similar methods. Therefore, a more recent trend in neuroevolution is the development of methods that use developmental or generative encodings for neural networks, which enables them to produce large neural networks from comparatively small genomes. Examples of such methods include HyperNEAT, NEATfields and Compressed Network Complexity Search (Stanley et al., 2009; Inden et al., 2012; Gomez et al., 2012). These methods enhance the scalability of neuroevolution significantly as compared to methods using a simpler direct encoding like NEAT.

While evolution can produce comparatively complex networks with hundreds of neurons using these methods, the complexity of the behavior that can be produced by evolution (i.e., adaptive behavior, not just random behavior) is still subject to some limitations. These limitations arise from the fact that a genome consisting of l elements, each of which can be chosen among an alphabet of size b, can encode a choice between b^l different phenotypes at most. Depending on the method used, this can mean that the maximum size of the encoded network will be limited (unless it is preset by the experimenter like in standard HyperNEAT) or the maximum algorithmic complexity of its connection pattern will be limited (unless the encoding includes techniques that refer to externally or environmentally supplied sources of complexity). Even if none of those two limitations apply, the fact remains that the number of referentiable phenotypes is limited, so the complexity behavior as achieved by selection remains limited unless the genome length l is increased. However, a selection-driven unlimited increase of l is usually not possible even though the methods allow for it in principle. To see why this is so, consider what happens as more and more genes are added to a given genome. If the mutation

rate per gene is fixed, then more and more mutations will arise in one genome as it gets longer. At some point, mutation will destroy the information in the genome despite the presence of selection. In the theoretical biology literature, this point is known as error threshold (Eigen, 1971; Stadler and Stadler, 2003). However, many neuroevolution methods (among them NEAT) typically apply only one mutation per genome regardless of its size. That way, mutations will not overpower selection, but on the other hand, per gene mutation rates will decrease. Together with them, the mutation rates on individual features of the phenotype will also decrease. As we are interested in adaptive behavior of increasing complexity, which consists of more and more individual features, the waiting times for adaptive changes of or extensions to particular features will increase until evolution practically comes to a halt. One common way of describing this is saying that the search space increases exponentially with the number of variables (the variables correspond to the phenotypic features here). One might think that as this happens, new opportunities arise for evolution in the form of extradimensional bypasses (Conrad, 1990; Bongard and Paul, 2001), but it is unlikely that this effect keeps pace with the increase of the search space for fitness functions as typically used in evolutionary robotics. In fact, it is a common heuristics in neuroevolution to make the search space as small as possible by use of prior information on the task. The NEAT method also starts with the smallest possible neural network topology and only then gradually explores larger topologies just to keep the search space small.

However, there is a way for evolution to avoid this dilemma: by changing the genetic architecture such that mutations will with a higher than random probability hit the right places. Ideally, successful features of an organism would be conserved by reducing the local mutation rates, whereas features under adaptive evolution would have increased local mutation rates. Mutation rates can be changed either by changing local application rates of particular mutation operators (like in some variants of self-adaptive evolution strategies (Beyer and Schwefel, 2002)), or by changing the mutational target sizes of the respective features if the used encoding and mutation operators allow for such reorganizations of the genetic representations of those features. In fact, it has been shown in simulations using the artificial life system AVIDA that the representations of conserved features can become compressed over time just by applying normal mutation and selection (Ofria et al., 2003). In principle, these changes of genetic architecture might also arise when using methods like HyperNEAT. However, selective pressure for these kinds of changes might primarily arise indirectly through increased evolvability of the offspring (for a review of different ideas, see (Hansen, 2006)) and may therefore be too weak in many situations, or become effective only over time scales that are currently beyond the reach of artificial evolution.

In this article, we propose a more direct method to guide mutations towards features under active evolution, and away from previously evolved adaptive features. The basic idea is that only those parts of a neural network that were created by mutations most recently can be mutated. Older structures are frozen and cannot be mutated any more. The NEAT method has a feature that makes implementation of this approach very easy: every time a neuron or a connection arises by mutation, it receives a globally unique identification number. One simple implementation is just to take this number from a counter that is increased whenever a new number is assigned. In that case, the numbers also provide an indication of relative time. The genes in a genome can be ordered according to the time of their creation, and only a fixed number of the newest genes are then available for mutations.

This method is related to earlier approaches on incremental evolution. E.g., one method uses a new module for each new fitness function term, and may restrict mutations in the older modules (Pasemann et al., 2001). However, our approach is more fine-grained and gradual, and makes use of the identification numbers that are used in NEAT and derived methods like HyperNEAT and NEATfields. No manual definition of modules or partitioning of fitness functions is necessary. We expect that if used in combination with other recent neuroevolution techniques, the technique presented here will enable open-ended evolution and complexification of neural agents.

Open-ended evolution is an interesting concept from the perspective of both evolutionary robotics and theoretical biology. A precise measure for evolutionary activity that was introduced several years ago (Bedau et al., 1998) links open-ended evolution to the unbounded growth of cumulative evolutionary activity, which can be achieved by unlimited growth of diversity and/or unlimited complexification. An abstract model that has unbounded growth of diversity has been introduced shortly afterwards (Maley, 1999). A more complex artificial life system with unbounded evolutionary activity has also been designed (Channon, 2006). One of us has recently presented an abstract model of open-ended coevolution (Inden, 2012). In that model, the genotype and phenotype of an organism is a string of integer numbers from some bounded range. The fitness of an organism depends on the results of matching its number string against those of organisms from the other population. In many variants of the model, mutations are only performed at the end of the string. The present article aims at initiating a transfer of this approach, and the resulting open-ended evolution, to the domain of neural agent evolution.

Methods

The *patches* task

An agent resides in a two-dimensional arena where x and y coordinates are constrained to the range $[-1, 1]$ each. At each time step, it can change its position by $o_i \cdot 0.1$ in both

dimensions simultaneously, where $o_i \in [-1, 1], i \in \{1, 2\}$, is the respective neural network output. This means it can get from one border of the arena to the other in twenty time steps. Every twenty time steps, the position of the agent is checked. Only if it is on the correct nest site in its arena, it will survive into the next round of twenty time steps. The left nest site covers all positions with $x \in [-1, -0.5)$, whereas the right nest site covers all positions with $x \in (0.5, 1]$. A long binary random sequence is generated before evolution begins, and determines which nest site is the correct one in each round. This means that the agents basically have to learn to produce a binary random sequence by evolution. The agents get the following input: a bias input, their current position, and a number of inputs indicating the number of the current round (as detailed below).

For every successful round, a reward of 1.0 is added to the fitness. In the final (unsuccessful) round, the agent obtains an additional reward of $\frac{1+pos_x}{4}$ if the right nest site was the correct target, or $\frac{1-pos_x}{4}$ if the left nest site was the correct target.

It can be seen from this description that actually only one spatial dimension is directly relevant for the task. The y direction is introduced for purposes of visualization and to provide an additional neutral dimension for future more complicated tasks.

The NEAT neuroevolution method

The NEAT (NeuroEvolution of Augmenting Topologies) method (Stanley and Miikkulainen, 2002) is a well known method for simultaneously evolving the topology and the connection weights of neural networks. It starts evolution with one of the simplest possible network topologies and proceeds by complexification of that topology. More specifically, the common ancestor of the whole population has one neuron for each output, each of which is connected to all inputs. There are no hidden neurons. Here, we start even simpler: Each output has one neuron, and each of this neurons is initially connected to 30% of the inputs on average. It has been shown previously that letting evolution select inputs for the neural network can result in superior performance if the input space is large as compared to starting with a fully connected network (Whiteson et al., 2005). Evolution then proceeds by adding neurons and connections.

Our NEAT implementation[1] uses mutation operators that are very similar to those of the original NEAT implementation for evolving the contents of the field elements. The most common operation is to choose a fraction of connection weights and either perturb them using a normal distribution with standard deviation 0.18, or (with a probability of 0.15) set them to a new value. The application probability of this weight changing operator is set to 1.0 minus the probabilities of all structural mutation operators, which amounts to 0.938 here. A structural mutation operator to connect previously unconnected neurons is used with probability 0.02, while an operator to insert neurons is used with probability 0.001. The latter inserts a new neuron between two connected neurons. The weight of the incoming connection to the new neuron is set to 1.0, while the weight of the outgoing connection keeps the original value. The idea behind this approach is to change the properties of the former connection as little as possible to minimize disruption of existing functional structures. The former connection is deactivated but retained in the genome where it might be reactivated by further mutations. There are two operators that can achieve this: one toggles the active flag of a connection and the other sets the flag to 1. They are used with probability 0.01 each.

Once a new gene arises by mutation, it receives a globally unique reference number. This number is generated by a global counter that is incremented every time a new gene arises. The innovation numbers are originally used by NEAT to align two genomes during the process of recombination (although in the experiments reported here, no recombination is used). They are also used to define a distance measure between networks, as will be explained in the next section.

The activation of the individual neurons is a weighted sum of the outputs of the neurons $j \in J$ to which they are connected, and a sigmoid function is applied on the activation: $o_i(t) = \tanh(\sum_{j \in J} w_{ij} o_j(t-1))$. Like in some other NEAT implementations, connection weights are constrained to the range $[-3, 3]$. There is no explicit threshold value for the neurons. Instead, a constant bias input is available in all networks.

Speciation selection

NEAT uses speciation selection by default. This method is fitness based, but uses some techniques to protect innovation that may arise during evolution against competition from fitter individuals that are already in the population. As a prerequisite, the globally unique reference numbers assigned to each gene are used to calculate a distance measure between two neural networks. The dissimilarity between two networks is calculated as $d = c_r \#ref_c + c_w \sum \Delta w$, where $\#ref_c$ is the number of connections present in just one of these networks, Δw are the connection weight differences (summed over pairs of connections that are present in both networks), and the c variables are weighting constants with $c_r = 1.0$, $c_w = 1.0$ by default.

Using this dissimilarity measure, the population is partitioned into species by working through the list of individuals. An individual is compared to representative individuals of all species until the dissimilarity between it and a representative is below a certain threshold. It is then assigned to this species. If no compatible species is found, a new species

[1] To be more accurate, we use an implementation of the NEAT-fields method, which is an extension of NEAT that makes possible the evolution of large neural networks with regularities (Inden et al., 2012). However, all extension features are switched off, so the method is reduced to pure NEAT.

is created and the individual becomes its representative.

The number of offspring assigned to a species is proportional to its mean fitness. This rather weak selection pressure prevents a slightly superior species from taking over the whole population, and enables innovative yet currently inferior solutions to survive. In contrast, the selection pressure between members of the same species is much stronger: the worst 60% of the individuals belonging to that species are deleted, after which the other individuals are selected randomly. Species that have at least five individuals in the next generation also take the best individual into the next generation without mutations. If the maximum fitness of a species has not increased for more than 100 generations and it is not the species containing the best network, its mean fitness is multiplied by 0.01, which usually results in its extinction. Also, in order to keep the number of species in a specified range, the dissimilarity threshold is adjusted in every generation if necessary. Here, the initial speciation threshold is 4.0, the population size is 1000, and the target number of species is between 35 and 45. All numerical parameters for speciation selection have been taken over from previous experiments on other tasks (Inden et al., 2012).

Genetic speciation will be used as point of comparison for the experiments reported here, but the main method used is speciation based on behavioral criteria. This means that the distance between neural agents is not calculated from their genes, but from a metric of the behavior space. For the *patches* task, the final position (x, y) of an individual is recorded, and the distance between two individuals is just $(x_1 - x_2)^2 + (y_1 - y_2)^2$. The initial speciation threshold is set to 0.1. Everything else works as for genetic speciation.

Tournament selection with a tournament size of two and an elite of size 10 is also used for comparison.

Incremental evolution of network architecture

Usually, mutations are applied on all genes with uniform probability. In contrast, the method introduced here allows mutations only on the c_m newest genes. The relative age of all genes is known because their innovation numbers are ordered by the time of their creation, so the smallest innovation number where mutations are still allowed can be calculated at the beginning of the mutation procedure from a list of all genes in the genome. $c_m$should obviously be greater than the number of genes in the common ancestors. For the *patches* task and the default input configuration described below, there are 18 genes in the common ancestor on average, and c_m is set to 25. The method is robust to some variation in this parameter.

All mutations are forbidden on older genes, including perturbations of the connection weights. However, connections between new and old neurons are allowed, as are split operations applied on connections from the input or to the output. The first exception makes connecting newly evolved structures with older structures possible, while the second opens up possibilities for newer structure to be connected to inputs and outputs.

Incremental provision of network input

For the task considered here, the agent needs to react differently in different rounds, therefore it needs to possess some information that is correlated to the number of the current round at any point in time. Given that neural networks can generate internal dynamics, they could be expected to track time entirely internally. One could also think of providing the current round as a binary number on several inputs, or provide some periodic inputs with different periods to simplify evolution of time-dependent behavior. Preliminary experiments have shown that all methods work to some degree, but not very well. Therefore, the approach chosen here is to provide an input for each round that is at 1.0 during that round, and at 0.0 at other times. That approach implies not only that the number of inputs is potentially unlimited, but also that it is growing linearly with the number of rounds.

In our NEAT implementation, the neuroevolution methods need to be provided with information on the input and output geometry by the task specific methods. For the purposes of the pure NEAT method as reported here, this information is just a list of network inputs and a list of network outputs[2]. Each input and output has a unique identification number just like the network genes have. This number is stored in the genome whenever a connection from input or to output is established by NEAT.

To evolve neural networks with a potentially unlimited number of inputs, three parameters have to be set in our approach. The first two are related to the task: The initial number of inputs $c_s = 20$ and the input increment number $c_i = 10$. Again, the exact values are task specific and can be varied within reasonable bounds without much influence on the performance. In the initial generation, c_s inputs are presented in the task geometry. Whenever the task components relating to these inputs are solved sufficiently well, the next c_i inputs are added to the task geometry. For the purposes of the experiments here, a solution is sufficiently good if it survives at least $n_i - c_i$ rounds, where n_i is the current number of network inputs provided. Once a sufficiently good individual is found, the number of provided inputs is increased for all individuals in the next generation.

The third parameter $c_c = 25$ is the number of newest inputs that are considered by the mutation operator for establishing new connections. Connections to older inputs will not be established. Of course, some inputs could be made exempt from this rule if they are considered to be important for later stages of evolution as well. But this does not seem to be necessary for the task considered here.

[2]For the NEATfields method, the task geometry contains more information related to how the inputs and outputs are structured into fields. Details can be found in Inden et al. (2012).

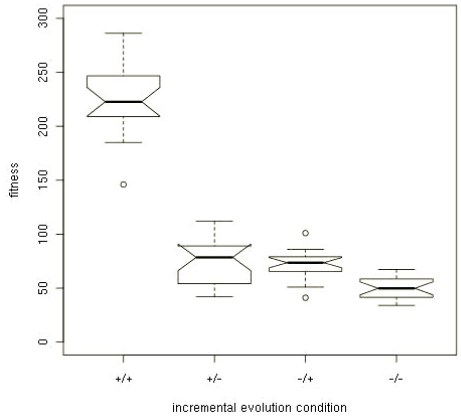

Figure 1: Performance of different configurations. The configurations are denoted by combinations such as "+/+", where the first symbol indicates the presence of the incremental network evolution technique, and the last symbol indicates the presence of the incremental input provision technique.

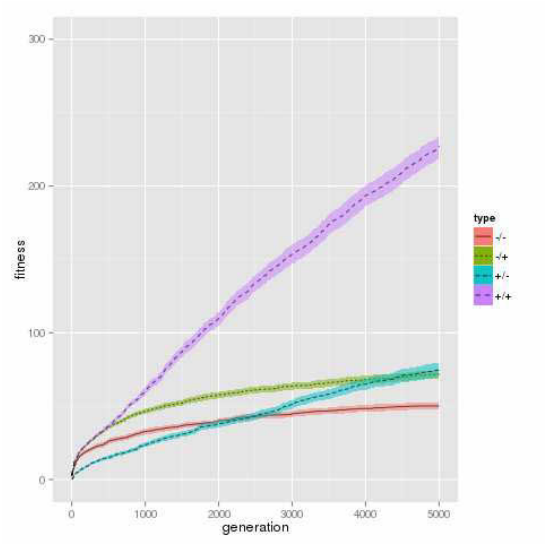

Figure 2: Mean highest fitness over the course of evolution for four different configurations denoted as in Fig. 1. The ribbons around the lines show the uncertainty in the mean (standard error).

Experiments and results

The first set of experiments is designed to show that the two techniques proposed here — incremental evolution of network architecture and incremental provision of network input — do indeed lead to evolutionary learning of sequences of arbitrary length if applied to the *patches* task. NEAT with both techniques is compared against NEAT with only one or none of these techniques. As Fig. 1 shows, using both techniques leads to significantly more performance than using one technique alone, which in turn is significantly better than using none of these techniques. When using both techniques, the highest fitness reached is 227.0 on average. (As for all other experiments reported here, 20 runs of 5000 generations each have been done, and significance has been established using Wilcoxon's rank sum test in addition to estimating it from the diagrams.) More importantly, as Fig. 2 shows, evolutionary dynamics is fundamentally different for the different configurations: When using both techniques, approximately linear growth of fitness (and of behavioral complexity by implication) with a steep slope occurs. If only the network is evolved incrementally, growth is much slower, although it may be linear as well in the depicted range. If only the inputs are provided incrementally, there is fast initial growth but later convergence. If none of the techniques is used, convergence is reached at even lower values.

It is also instructive to look at the neural network resulting from the run that achieved the highest fitness (Fig. 5). This particular network is encoded by 410 genes and consists of 39 neurons and 321 active connections (On average, the champions of all runs had 36.1 ± 1.3 neurons). It can be seen that the output for movement in the x direction is integrated at neuron 0, whereas many different neurons directly connect to the output responsible for movement in the neutral y direction. Not all inputs are connected to the network. For those that are, there is a rather regular pattern of radial spikes leading to the hidden layer of the network. However, some hiden neurons are also connected to a few older inputs. In general, the older neurons have a higher degree of connectedness. This pattern of connectivity can be interpreted as arising from a rather regular incremental evolution of structures together with some reuse of older structure.

When knocking out a single neuron at a time, the performance of the network is decreased for 23 (59%) of the neurons. This is a lower bound for the number of functional neurons in the network (more neurons could perhaps be shown to contribute to the function in experiments where multiple neurons are knocked out simultaneously). Furthermore, if neurons are ordered according to their time of creation by a mutation, an interesting pattern emerges (Fig. 3): The order of their creation is strongly correlated to the performance of the knockout network. A plausible explanation is that neurons that evolve later typically do not affect the behavior of the agents in the early rounds of the *patches* task. The structure that controls this behavior has been frozen. The specific task of those later neurons is to control behavior in the later rounds of the task.

A second set of experiments has been designed to examine the influence of different selection methods on the per-

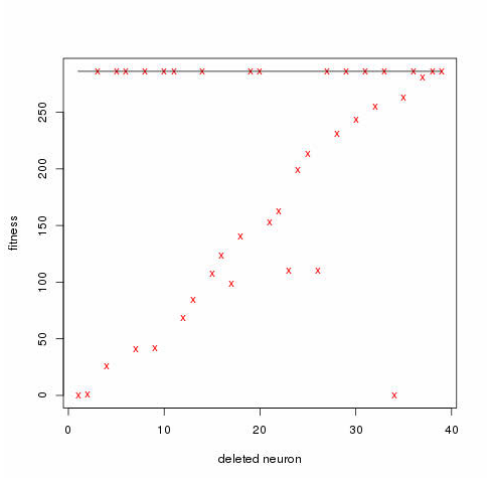

Figure 3: Fitness of an evolved network after knockout of a single neuron each. Neurons are ordered according to the time of their evolutionary emergence. The line at the top indicates the fitness of the unaltered network.

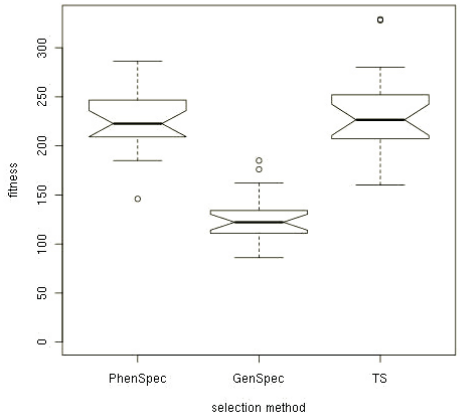

Figure 4: Performance of different selection methods for the *patches* task. PhenSpec, behavioral speciation; GenSpec, genetic speciation; TS, tournament selection.

formance. Earlier experiments have shown that solutions of sufficient quality for many tasks can only be found using particular selection methods like speciation selection or methods using novelty search (Stanley and Miikkulainen, 2002; Lehman and Stanley, 2011; Inden et al., 2013). Therefore, interesting results on representations and mutation operators could be masked if inappropriate selection methods are used. We have therefore adopted the general strategy of never just considering one aspect of neuroevolution in isolation. However, as Fig. 4 shows, performance does not differ significantly between behavior speciation and tournament selection for this task. Genetic speciation performs significantly worse than the other two methods.

Discussion

We have claimed that neural agents can learn behavioral sequences of arbitrary length using the techniques presented here, and that this represents a step towards open-ended evolution and complexification. Evidence has been provided in the form of approximately linear growth of behavioral complexity over 5000 generations. To our knowledge, the resulting adaptive behavioral complexity is a substantial advance over previous evolutionary robotics results, where agents where trained to perform a few sequential actions only.

However, it might be argued that there is nothing to guarantee further linear growth beyond what has been empirically shown, and that there even may be some inherent limitations in the method that might eventually (though much later than usual) make evolution come to a halt. To these kinds of arguments, two replies can be made: Firstly, we may (as discussed by previous authors (Bedau et al., 1998; Maley, 1999)) take a rather pragmatic approach to open-endedness. Eventually, evolution will always come to a halt because of resource limits in its environment. If we have a method that seems likely to be able to reach these resource limits provided that parameters are set appropriately, this may already qualify as leading towards open-ended evolution. Secondly, we have only described a simple implementation of our ideas because this was sufficient for the simple task studied here. More sophisticated implementations might remove some remaining limitations in the method. For example, as implemented currently, it is possible to connect new neurons to arbitrary old neurons. As the number of old neurons grows by complexification, the waiting times for connections to specific neurons again increase, which may lead to a slowdown in evolution. However, one could also easily set a second age threshold and only connect to old neurons that are younger than this second threshold. Similar solutions should be possible for other mutation operators.

Here, we let the agents only learn random sequences. When their length increases, so does their algorithmic complexity, as that complexity essentially measures the randomness of a sequence (Li and Vitányi, 1997). A fundamental question around the issue of open-ended evolution, however, is whether also the complexity concerned with the structural regularities can be made to increase by such a scheme (Ay et al., 2011). This is not yet addressed by our simulations. It would clearly be interesting to provide more regular sequences and find out how well some kind of generalization can be learned on top of this sequential learning of individual instances. We think that the ideas of incremental genome growth and restriction of mutations to the most recently evolved genes can also contribute to that issue.

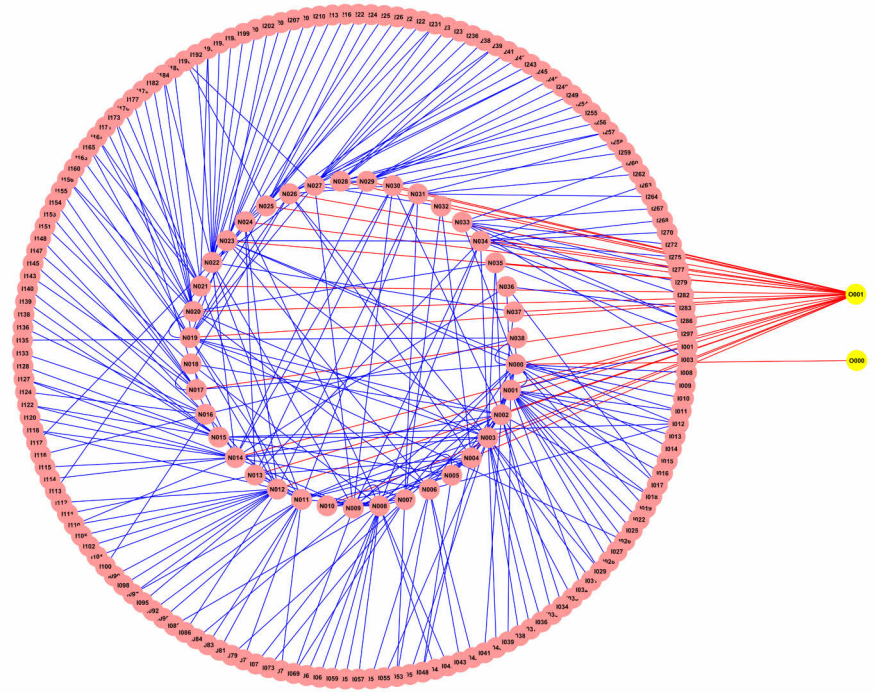

Figure 5: An evolved neural network that survives for 286 rounds. Inputs are displayed in the outer circle, hidden neurons in the inner circle, and outputs on the right. Connections to the output are displayed in red. Inputs and hidden neurons are in evolutionary order starting approximately at 3 o'clock and rotating clockwise from there.

One might ask to what extend open-ended complexification can be achieved in other domains using these methods. This is a question for further research. A first observation is that in the task studied here, individual parts of the task are presented sequentially (i.e., the agent only has to make a correct guess at some point in time once it has evolved to make correct guesses at all times before that time). This corresponds well with the incremental evolution of network structure enforced by our technique. It is unknown whether and under what conditions evolution can find a path for itself if the fitness function is less structured. It will also be interesting to see how well the methods work for real robots operating in more complex and noisy environments.

We also note that the presented techniques can be used with other neuroevolution methods that use unique identification numbers for genes. In fact, we have already implemented it for NEATfields and aim to combine incremental evolution with learning geometric regularities as is possible with NEATfields and some other methods.

Finally, while the presented technique might seem to be a very contrived addition to evolution, it could be argued that it is a strong simplification and an extreme case of something that does happen naturally to some degree in evolution if evolution occurs on complex genetic architectures. We have already mentioned that selection for compressed representations of individual features has been observed in an artificial life system (Ofria et al., 2003). It is also known that different regions in animal genomes are subject to different mutation rates, and that this is under genetic control (Martincorena et al., 2012). Evolution of genetic architectures has been the subject of intense research in recent decades (Hansen, 2006). Therefore, the ideas presented here for open-ended evolution and complexification may ultimately be relevant for a class of systems much wider than just neural agents.

Acknowledgments B. I. is currently funded by the Deutsche Forschungsgemeinschaft (DFG) in the Collaborative Research Center 673. Additional funding for the project "Coevolutionary complexification of autonomous agents" was supplied by Bielefeld University within the programme "Bielefelder Nachwuchsfonds".

References

Ay, N., Olbrich, E., Bertschinger, N., and Jost, J. (2011). A geometric approach to complexity. *Chaos*, 21:037103.

Bedau, M. A., Snyder, E., and Packard, N. H. (1998). A classification of long-term evolutionary dynamics. In et al., C. A., editor, *Artificial Life VI*.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strate-

gies — a comprehensive introduction. *Natural Computing*, 1:3–52.

Bongard, J. and Paul, C. (2001). Making evolution an offer it can't refuse: Morphology and the extradimensional bypass. In *Proceedings of the Sixth European Conference on Artificial Life*.

Channon, A. (2006). Unbounded evolutionary dynamics in a system of agents that actively process and transform their environment. *Genetic Programming and Evolvable Machines*, 7:253–281.

Conrad, M. (1990). The geometry of evolution. *BioSystems*, 24:61–81.

Eigen, M. (1971). Selforganization of matter and evolution of biological macromolecules. *Naturwissenschaften*, 58:465–523.

Gomez, F., Koutník, J., and Schmidhuber, J. (2012). Compressed network complexity search. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature*.

Hansen, T. (2006). The evolution of genetic architecture. *Annual Reviews of Ecology, Evolution and Systematics*, 37:123–157.

Inden, B. (2012). Open-ended coevolution and the emergence of complex irreducible functional units in iterated number sequence games. In *Proceedings of the 14th annual conference on genetic and evolutionary computation*.

Inden, B., Jin, Y., Haschke, R., and Ritter, H. (2012). Evolving neural fields for problems with large input and output spaces. *Neural Networks*, 28:24–39.

Inden, B., Jin, Y., Haschke, R., and Ritter, H. (2013). An examination of different fitness and novelty based selection methods for the evolution of neural networks. *Soft Computing*, 17:753–767.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19:189–223.

Li, M. and Vitányi, P. (1997). *An introduction to Kolmogorov complexity and its applications*. Springer.

Maley, C. C. (1999). Four steps toward open-ended evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Martincorena, I., Seshasayee, A. S. N., and Luscombe, N. M. (2012). Evidence of non-random mutation rates suggests an evolutionary risk management strategy. *Nature*, 485:95–98.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics — The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press.

Ofria, C., Adami, C., and Collier, T. C. (2003). Selective pressures on genomes in molecular evolution. *Journal of Theoretical Biology*, 222:477–483.

Pasemann, F., Steinmetz, U., Hülse, M., and Lara, B. (2001). Robot control and the evolution of modular neurodynamics. *Theory in Biosciences*, 120:311–326.

Pfeifer, R. and Gómez, G. (2005). Interacting with the real world: design principles for intelligent systems. *Artificial Life and Robotics*, 9:1–6.

Stadler, B. M. R. and Stadler, P. F. (2003). Molecular replicator dynamics. *Advances in Complex Systems*, 6:47–77.

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

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15:185–212.

Stanley, K. O. and Miikkulainen, R. (2004). Competitive coevolution through coevolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

Whiteson, S., Stone, P., Stanley, K. O., Miikkulainen, R., and Kohl, N. (2005). Automatic feature selection in neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*.