Indirectly Encoding Neural Plasticity as a Pattern of Local Rules

In: *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB 2010).* New York, NY: Springer

Sebastian Risi and Kenneth O. Stanley

School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816
{risi,kstanley}@eecs.ucf.edu

Abstract. Biological brains can adapt and learn from past experience. In neuroevolution, i.e. evolving artificial neural networks (ANNs), one way that agents controlled by ANNs can evolve the ability to adapt is by encoding local learning rules. However, a significant problem with most such approaches is that local learning rules for every connection in the network must be discovered separately. This paper aims to show that learning rules can be effectively indirectly encoded by extending the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) method. *Adaptive HyperNEAT* is introduced to allow not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also patterns of arbitrary *learning rules*. Several such adaptive models with different levels of generality are explored and compared. The long-term promise of the new approach is to evolve large-scale adaptive ANNs, which is a major goal for neuroevolution.

Key words: Adaptation, Learning, HyperNEAT, Neuroevolution

1 Introduction

Research in neuroevolution, i.e. evolving artificial neural networks (ANNs) through evolutionary algorithms, often focuses on static ANNs (i.e. weights do not change during the network's lifetime). However, in many control and decision-making problems, the environment may change too quickly to allow phylogenetic adaptation; thus the controller needs to adapt *online* to maintain performance. For example, a robot may need to remember a location that changes over time.

One way that agents controlled by ANNs can evolve the ability to adapt over their lifetime is by encoding *local learning rules* in the genome that determine how their synaptic connection strengths should change in response to changing activation levels in the neurons they connect [1–3]. This approach resembles the way organisms in nature, which possess plastic nervous systems, cope with changing and unpredictable environments.

Although demonstrations of this approach have suggested the promise of evolving adaptive ANNs, a significant problem is that local learning rules for every connection in the network must be discovered separately. That is, although

interest has grown in recent years in *indirectly encoding* the weights of ANNs so that they can be discovered as patterns [4–9], the power of indirect encoding is rarely applied to encoding *learning rules*. Yet the distribution of rules across a network likely conforms to discoverable regularities just as weights.

Additionally, as noted by Yao [10], the right learning rule depends on the ANN architecture, which makes it difficult to design an optimal such rule a priori. Yao further points out that designing learning rules by hand, which is common in this area [1, 10], requires making assumptions that might not hold in practice.

This paper aims to show that learning rules can be effectively indirectly encoded by extending the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) method [6, 11, 12], which currently indirectly encodes large geometric patterns of fixed weights for high-dimensional problems [6,11–13]. The new method introduced here, called *adaptive HyperNEAT*, allows not only patterns of weights across the connectivity of an ANN to be generated by a function of its geometry, but also *patterns of learning rules*. The idea that learning rules can be distributed in a geometric pattern is new to neuroevolution but reflects the intuition that synaptic plasticity in biological brains is not encoded in DNA separately for every synapse in the brain. Thus the main idea in this paper is a step towards more biologically plausible adaptive systems.

An important contribution of this work is to show that there is a tradeoff between the *generality* of an indirect encoding of plasticity and its computational cost. Yet, as experiments in a variant of the T-Maze learning domain [3, 14] will show, in special cases, e.g. when the reward signature is nonlinear and the ANN topology is restricted, a most general encoding may be necessary. Thus, rather than offering a single approach to all problems, this paper reveals the existence of a continuum of adaptive encodings that trade off generality with computational expense. From this perspective the practitioner can make the most informed choice on the ingredients that may be necessary for a particular domain.

Building on the ability of HyperNEAT to evolve large-scale connectivity patterns, the long-term promise of the new approach is to evolve *large-scale adaptive ANNs*, which is a major goal for neuroevolution.

2 Background

The HyperNEAT method that enables learning from geometry in this paper is an extension of the original NeuroEvolution of Augmenting Topologies (NEAT) algorithm that evolves ANNs through a *direct* encoding [15, 16]. NEAT starts with a population of small, simple neural networks and then *complexifies* them over generations by adding new nodes and connections through mutation. By evolving networks in this way, the topology of the network does not need to be known a priori. The important feature of NEAT for the purpose of this paper is that it evolves *both* the topology and weights of a network.

However, in direct encodings like NEAT, each part of the representation maps to a single piece of structure in the solution [17]. The significant disadvantage

of this approach is that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. Thus HyperNEAT employs an *indirect* encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself [4–9].

In HyperNEAT, NEAT is altered to evolve an indirect encoding called *compositional pattern producing networks* (CPPNs [8]) *instead* of ANNs [6,11–13]. The main idea in HyperNEAT is that the CPPN, which is itself a network composed of a variety of activation functions, acts as a pattern generator that outputs a *pattern of connection weights* situated within the geometry of the ANN. The activation functions within the CPPN, such as sine and Gaussian, allow it to express regularities across the geometry of the ANN [6, 11, 12].

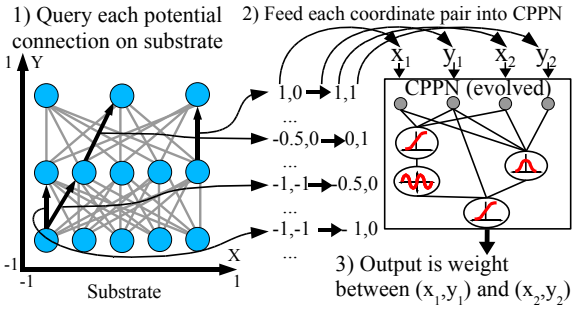


Fig. 1. How an ANN is Encoded by a CPPN. A collection of ANN nodes, called the *substrate*, is assigned coordinates that range from -1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) Internally, the CPPN (which is evolved) is a graph that determines which activation functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connections in space.

Formally, CPPNs are *functions* of geometry (i.e. locations in space) that output connectivity patterns whose nodes are situated in n dimensions, where n is the number of dimensions in a Cartesian space. Consider a CPPN that takes four inputs labeled x_1, y_1, x_2, and y_2; this point in four-dimensional space *also* denotes the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2), and the output of the CPPN for that input thereby represents the weight of that connection (Fig. 1). By querying every possible connection among a pre-chosen set of points in this manner, a CPPN can produce an ANN, wherein each queried point is a neuron position. Because the connections are produced by a function of their endpoints, the final structure is produced with *knowledge* of its geometry. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as the isomorphic connectivity pattern, which explains

the origin of the name *hypercube-based NEAT* (HyperNEAT). Connectivity patterns produced by a CPPN in this way are called *substrates* so that they can be verbally distinguished from the CPPN itself, which has its own internal topology. As a rule of thumb, nodes are placed on the substrate to reflect the geometry of the task [11–13]. That way, the connectivity of the substrate is a function of the task structure and while the task may be complex, the domain geometry is often intuitive.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot. Outputs for moving left or right can also be placed in the same order, allowing HyperNEAT to understand from the outset the correlation of sensors to effectors. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry) of a problem that are invisible to traditional encodings.

For a complete overview of HyperNEAT, see Gauci and Stanley [6] and Stanley et al. [12]. The next section extends this approach to evolve adaptive ANNs.

3 APPROACH: Adaptive HyperNEAT

The main idea in adaptive HyperNEAT is that CPPNs can not only encode connectivity patterns but also *patterns of plasticity rules*. As in the brain, different *regions* of the ANN should be more or less plastic and employ different learning rules, which HyperNEAT allows because it sees the ANN geometry. In general, a learning rule changes the weight of a connection based on presynaptic activity o_i, postsynaptic activity o_j, and the current connection weight w_{ij}:

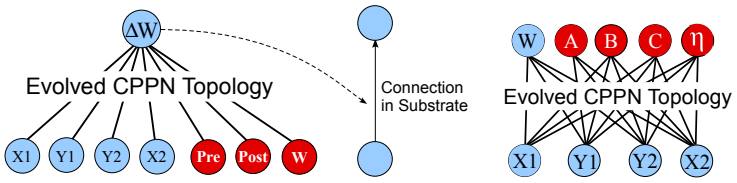
$$\Delta w_{ij} = \Phi(o_i, o_j, w_{ij}) \ . \tag{1}$$

In this paper three different adaptive HyperNEAT models are compared that are able to encode different levels of learning rule generality. The goal of this comparison is to elucidate the advantages and disadvantages of different levels of generality to modeling dynamic learning processes. All three models allow learning rules to be distributed as patterns across the connectivity of an ANN.

The most general **iterated model** (Fig. 2a) augments the four-dimensional CPPN that normally encodes connectivity patterns with three additional inputs: presynaptic activity o_i, postsynaptic activity o_j, and the current connection weight w_{ij}. That way, the synaptic plasticity of a connection between two two-dimensional points (x_1, y_1) and (x_2, y_2) can be described by

$$\Delta w_{ij} = \text{CPPN}(x_1, y_1, x_2, y_2, o_i, o_j, w_{ij}) \ . \tag{2}$$

The update of the synaptic weights can thereby be iteratively performed by the same CPPN that normally encodes network connectivity, which allows evolving increasingly complex learning rules. In effect, the CPPN encodes an entire dynamical system, including how changes depend on both location *and* activity. The CPPN is *requeried* on every tick of the clock to update the ANN weights.



(a) Iterated Model (b) ABC Model

Fig. 2. Adaptive HyperNEAT. CPPNs for the iterated (a) and ABC models (b) are shown. The CPPN in (a) is continually requeried during the lifetime of the agent to determine the weight change given the location of the connection, activation of the presynaptic and postsynaptic neuron, and the current weight as input. In contrast, the CPPN in (b) is only activated once to determine the three parameters A–C and the learning rate η, which control synaptic plasticity during the lifetime of the agent, in addition to the initial weight w.

The initial weight configuration is determined by querying the CPPN as in the original HyperNEAT approach (Sec. 2) with the presynaptic activity, postsynaptic activity, and weight inputs all set to zero.

The less general **Hebbian ABC model** augments the CPPN instead with four additional *outputs* (Fig. 2b): learning rate η, correlation term A, presynaptic term B, and postsynaptic term C. When the CPPN is initially queried, these parameters are permanently stored, which allows the synaptic weight to be modified during the lifetime of the agent by the following plasticity rule:

$$\Delta w_{ij} = \eta \cdot [A o_i o_j + B o_i + C o_j] \ . \tag{3}$$

Traditional approaches to evolving adaptive ANNs with direct encodings also evolve the coefficients of Equation (3) but because of the limitations of direct encodings often only employ *one* such evolved rule throughout all ANN connections [3, 18]. The difference here is that A, B, C, and η are indirectly encoded by HyperNEAT in a geometric pattern across the connectivity of the whole network. Therefore each connection could potentially employ a different rule if necessary. However, unlike the more general iterated model, this CPPN only produces variants of the ABC Hebbian rule. Thus the space of possible rules is more restricted.

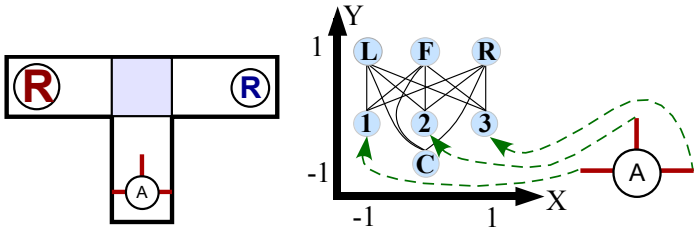
Finally, the simplest model is **plain Hebbian**. The CPPN has only one additional output that encodes the learning rate η:

$$\Delta w_{ij} = \eta \cdot o_i o_j \ . \tag{4}$$

This variant tests for the minimal sufficient dynamics to solve the T-Maze domain given in this paper, which is explained in the next section.

4 T-Maze Domain

T-Mazes are often studied in the context of operant conditioning of animals; they are also studied to assess the ability of plastic ANNs [3, 14]. The discrete



(a) T-Maze (b) Substrate

Fig. 3. T-Maze and Substrate Configuration. (a) In this depiction, high reward is located on the left and low reward is on the right side, but these positions can change over a set of trials. The challenge for the agent is to remember the location of the high reward from one trial to the next. (b) The autonomous agent A is equipped with three distance sensors and a reward color sensor that is set to zero during navigation.

T-Maze in this paper (Fig. 3a) consists of two arms that either contain a high or low reward. The agent begins at the bottom of the maze and its goal is to navigate to the reward position. This procedure is repeated many times during the agent's lifetime. One such attempted trip to a reward location is called a *trial*. A *deployment* consists of a set of trials. When the position of the reward sometimes changes, the agent should alter its strategy accordingly to explore the other arm of the maze in the next trial and *remember* the new position in the future (requiring adaptation). The goal of the agent is to maximize the amount of reward collected over all deployments.

5 Experiments

To generate a controller for the T-Maze domain, the evolved CPPNs query the substrate shown in Fig. 3b. The locations of inputs and outputs are designed to geometrically correlate (e.g. seeing something on the left correlates to turning left). Thus the CPPN can exploit the geometry of the agent. The agent is equipped with three rangefinder sensors that detect walls to the left, front, and right of the robot. The *Color* input (explained shortly) is set to the color of the collected reward at the maze end, which determines the amount of reward given to the agent. The three output neurons are *Left*, *Forward*, and *Right*. At each simulated time step, the agent continues to move straight at a constant pace if the *Forward* output has the highest activation level. Otherwise the agent turns 90 degrees in the direction of the highest activated neuron (Left or Right).

An agent crashes if it does not (1) maintain a forward direction in corridors or (2) turn either right or left when it encounters the junction. If the agent crashes then the current trial is terminated.

In this paper, *two* T-Maze scenarios are studied to elucidate the advantages and disadvantages of encoding different levels of plasticity rule generality. **Scenario 1** resembles the traditional T-Maze domain described in the previous section. Each agent is evaluated on four deployments with 100 trials each. The

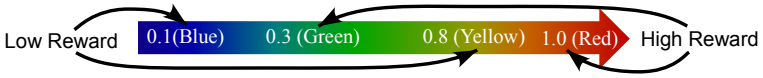


Fig. 4. Nonlinear Reward Color Encoding. The agent receives a high reward for green or red and a low reward for blue or yellow. The ANN color encoding together with the given ANN topology requires the agent to include a nonlinear learning rule.

starting position of the high reward alternates between deployments and switches positions after 50 trials on average. Color input values of 1.0 and 0.1 encode the high (red) and low (blue) reward, respectively.

In **scenario 2**, the agent is exposed to a total of four different colored rewards. The first deployment resembles scenario 1 with reward signatures of 0.1 and 1.0. However, in the *second* deployment, color input values of 0.3 and 0.8 are introduced to encode new high yellow and low green rewards, respectively (Fig. 4). Adding these intermediate reward colors yields a reward signature that is *not linearly separable*. Because the ANN controlling the agent does not have any hidden neurons, the learning rule must *itself* be nonlinear. Scenario 2 therefore makes a good domain for this study because it requires evolving a specific learning rule that depends on the ANN topology.

The fitness function, which is the same for all compared approaches and identical to Soltoggio et al. [3], is calculated as follows: Collecting a high reward has a value of 1.0 and a low reward is worth 0.2. A penalty of 0.4 is subtracted if the agent does not maintain forward motion in corridors or does not turn left or right at a junction. The total fitness of an individual is determined by summing the fitness values for each of the 100 trials over all deployments.

Note that although Risi et al. [19] showed that novelty search [20], which abandons objective-based fitness and instead simply searches only for novel behavior, significantly outperforms fitness-based search in the traditional T-Maze domain, a standard fitness function is employed in this paper to keep the experiment focused on the issue of adaptation.

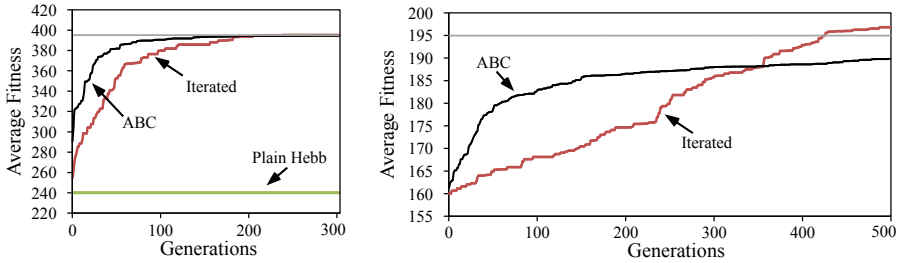
5.1 Experimental Parameters

All experiments were run with a modified version of the public domain SharpNEAT package [21] called HyperSharpNEAT. Runs consisted of 500 generations with a population size of 500 and 10% elitism. Sexual offspring (50%) did not undergo mutation. Asexual offspring (50%) had 0.94 probability of link weight mutation, 0.03 chance of link addition, and 0.02 chance of node addition. The available CPPN activation functions were sigmoid, Gaussian, absolute value, and sine, all with equal probability of being added. A connection is not ever expressed if the magnitude of its initial weight is below a minimal threshold of 0.4. Parameter settings are based on standard SharpNEAT defaults and prior reported settings for NEAT [15,16]. For all adaptive HyperNEAT models synaptic strength is bound within the range $[-1.0, 1.0]$.

6 Results

The standard T-Maze (scenario 1) is solved when the agent reaches a fitness of 395. A minimum amount of exploration (i.e. collecting the low reward) is required at the beginning of each deployment and when the reward positions switch. The T-Maze with nonlinear reward signature (scenario 2), consisting of two deployments with different reward signatures, is solved with a fitness of 195. All reported results are averaged over 20 runs.

Figure 5a shows the average training performance over generations for the standard T-Maze (scenario 1). It took the ABC model 141 generations (σ=141) on average to find a solution. The iterated model took 89 generations (σ=61) on average. While the fitness for the iterated model initially increases more slowly than for ABC, it finds a solution slightly (though not significantly) faster on average. The plain Hebbian model cannot solve the task. Although both the more general iterated model and the ABC model can solve the task, the iterated model is computationally more expensive because the CPPN must be continually requeried for every ANN connection.

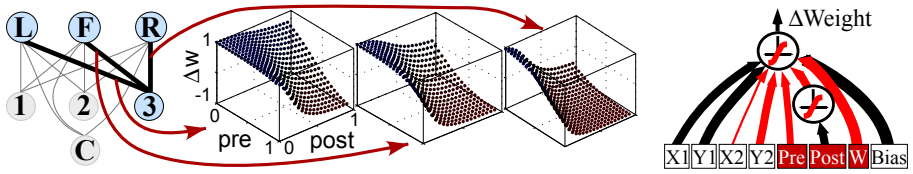


(a) T-Maze (b) T-Maze with Nonlinear Reward Signature

Fig. 5. Training Performance. The change in performance over evaluations for both scenarios is shown in this figure. All results are averaged over 20 runs. The horizontal line (top) indicates at what fitness the domain is solved. The iterated and ABC model are both able to solve the standard T-Maze domain (a) in about the same number of generations whereas the plain Hebbian approach does not show the necessary dynamics. The T-Maze domain with a nonlinear reward signature (b) requires a nonlinear learning rule, which only the iterated model discovers.

The average training performance over generations for scenario 2 is shown in Fig. 5b. The plain Hebbian rule is not tested in this variant because it is not able to solve the standard T-Maze. Whereas the iterated model solves the domain in 19 out of 20 runs, in 367 generations (σ=101) on average, ABC is not able to solve the task with the given ANN topology, which suggest the need for a nonlinear learning rule in this scenario (or potentially an ANN with hidden nodes). The more general iterated model is able to evolve such a rule.

Figure 6a shows CPPN-encoded learning rules of an ANN solution discovered by the iterated model. The function embodied by the CPPN (Fig. 6b) encodes



(a) Evolved Learning Rules (b) CPPN

Fig. 6. Discovered Learning Rules of an ANN Solution Created by the Iterated Model and the Underlying CPPN. The nonlinear learning rules shown in (a) are encoded by the evolved CPPN shown in (b). A geometric pattern of learning rules can been seen that varies with the target node's x location. The amount of synaptic change is a function of the pre- and postsynaptic activity and the corresponding positions of the pre- and postsynaptic neurons in the substrate (weight input w on the CPPN is set to zero in this depiction).

a geometric pattern of nonlinear learning rules. Interestingly, the evolved rules resemble postsynaptic-based learning rules that have been shown essential in the T-Maze domain [18].

7 Discussion and Future Work

The indirect HyperNEAT encoding is able to generate ANNs with millions of connections based on underlying geometric motifs [12]. This paper introduced an extension called adaptive HyperNEAT that generates not only patterns of weights across the connectivity of an ANN, but also patterns of learning rules (Fig. 6). The long-term promise of the new approach is therefore to evolve large-scale adaptive ANNs, which is a major goal for neuroevolution that may bring it closer to evolving brain-like structures.

While the ABC model together with an adequate ANN topology should be sufficient for most domains, the nonlinear variant of the T-Maze learning domain reveals that sometimes a more general encoding may be necessary. Although the ANN topology could potentially have been extended to allow a less general model to solve the nonlinear T-Maze domain, this experiment confirms the risk of making a priori assumptions about the type of necessary learning rules [10].

However, the generality of the indirect encoding of plasticity trades off with its computational cost. The most general iterated model is computationally expensive because the CPPN must be continually requeried for every ANN connection. The computational complexity for every time step is $O(n) + nO(m)$, where $O(n)$ and $O(m)$ are the costs of simulating an ANN with n connections and an underlying CPPN with m connections, respectively. Thus the most general model in its current form might be too computationally expensive for practical purposes that require large CPPNs and ANNs. However, it gives us a reference point from which to derive more specialized models such as the ABC model.

In the current iterated model the synaptic weights are updated at every time step. Characterizing how often a weight update is necessary is an important

future research direction that may allow cutting down the computational cost of even the most general model. Additionally, synaptic plasticity could be controlled by neuromodulation [2, 3, 19], which means that some neurons can enhance or dampen the neural plasticity of their target nodes. Such modulation could allow precise timing of CPPN weight queries. Finally, another potentially promising approach is combining the iterated and ABC models.

8 Conclusion

A new method called adaptive HyperNEAT was presented, which allows not only patterns of weights across the connectivity of an ANN to be indirectly encoded as a function of its geometry, but also patterns of arbitrary learning rules. Importantly, this paper shows that there is a tradeoff between the generality of an indirect encoding of plasticity and its computational cost. Yet, as a variation of the T-Maze domain demonstrates, the most general encoding may be necessary in some cases. The main conclusion is that the indirect HyperNEAT encoding may enable evolving large-scale adaptive ANNs.

9 Acknowledgments

This research was supported by DARPA under grant HR0011-09-1-0045 (Computer Science Study Group Phase 2).

References

1. Floreano, D., Urzelai, J.: Evolutionary robots with online self-organization and behavioral fitness. Neural Networks **13** (2000) 431–443
2. Niv, Y., Joel, D., Meilijson, I., Ruppin, E.: Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. Adaptive Behavior **10**(1) (2002) 5–24
3. Soltoggio, A., Bullinaria, J.A., Mattiussi, C., Dürr, P., Floreano, D.: Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-based Scenarios. In: Artificial Life XI, Cambridge, MA, MIT Press (2008) 569–576
4. Bentley, P.J., Kumar, S.: Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999). (1999) 35–43
5. Bongard, J.C.: Evolving modular genetic regulatory networks. In: Proceedings of the 2002 Congress on Evolutionary Computation. (2002)
6. Gauci, J., Stanley, K.O.: Autonomous evolution of topographic regularities in artificial neural networks. Neural Computation (2010) To appear.
7. Hornby, G.S., Pollack, J.B.: Creating high-level components with a generative representation for body-brain evolution. Artificial Life **8**(3) (2002)
8. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines Special Issue on Developmental Systems **8**(2) (2007) 131–162

9. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9**(2) (2003) 93–130
10. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87**(9) (1999) 1423–1447
11. Gauci, J., Stanley, K.O.: A case study on the critical role of geometric regularity in machine learning. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008), Menlo Park, CA, AAAI Press (2008)
12. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based indirect encoding for evolving large-scale neural networks. Artificial Life **15**(2) (2009) 185–212
13. Clune, J., Beckmann, B.E., Ofria, C., Pennock, R.T.: Evolving coordinated quadruped gaits with the hyperneat generative encoding. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Section on Evolutionary Robotics, Piscataway, NJ, USA, IEEE Press (2009)
14. Blynel, J., Floreano, D.: Exploring the T-Maze: Evolving Learning-Like Robot Behaviors using CTRNNs. In: 2nd European Workshop on Evolutionary Robotics (EvoRob'2003). Lecture Notes in Computer Science (2003)
15. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10** (2002) 99–127
16. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. **21** (2004) 63–100
17. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence **1**(1) (2008) 47–62
18. Soltoggio, A.: Neural Plasticity and Minimal Topologies for Reward-Based Learning. In: Proceedings of the 2008 8th International Conference on Hybrid Intelligent Systems, IEEE Computer Society (2008) 637–642
19. Risi, S., Vanderbleek, S.D., Hughes, C.E., Stanley, K.O.: How novelty search escapes the deceptive trap of learning to learn. In: GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2009) 153–160
20. Lehman, J., Stanley, K.O.: Exploiting open-endedness to solve problems through the search for novelty. In Bullock, S., Noble, J., Watson, R., Bedau, M., eds.: Proceedings of the Eleventh International Conference on Artificial Life (Alife XI), Cambridge, MA, MIT Press (2008)
21. Green, C.: SharpNEAT homepage. http://sharpneat.sourceforge.net/ (2003–2006)