Developmental neuronal networks as models to study the evolution of biological intelligence

Arend Hintze[1,2], P. Robin Hiesinger[3], Jory Schossau[2,4]

[1]Department of Complex Dynamical Systems and MicroData Analytics, Dalarna University, Dalarna, Sweden
[2]BEACON Center for the Study of Evolution in Action, Michigan State University, MI, United States of America
[3]Division of Neurobiology, Institute for Biology, Freie Universität Berlin, Germany
ahz@du.se

Abstract

Artificial neural networks (ANNs) learn through iterative adjustment of synaptic weights, most commonly by backpropagation or similar training methods. By contrast, learning in the neuro-evolution approach of ANNs as well as in biological neural networks can also be achieved through iterative adjustment of synaptic weights based on mutational changes of a genome. Evolutionary learning of ANNs is typically based on changes in the genome that directly affect synaptic weights, while the biological genome affects changes to synaptic weights through indirect and developmental encoding of the molecular composition of synapses. Methods for indirect and developmental encoding in ANNs are comparably scarce and their constraints and advantages remain incompletely explored. Here, we identify robustness to environmental challenges as a key feature associated with indirect and developmental encoding using a neuro-evolution approach. In biological systems, the developmental process 'unfolds' synaptic connectivity information in a time- and energy-consuming process that is sensitive to environmental conditions like temperature. Our tests for robustness to developmental temperature reveal successful adaptation for indirect encoding followed by a developmental process compared to lower performance of developmental or indirect encoding alone. These findings set the stage for a more comprehensive neuro-evo-devo approach that acknowledges the necessity of neural network development for biological intelligence.

Introduction

The quest for general purpose artificial intelligence (AI) has long been accompanied by the question to what extent such an AI should resemble a biological brain. Artificial Neural Network (ANNs) currently represent the most successful approach in artificial intelligence research. While inspired by nature, significant differences remain. ANNs are very often structured as layers with randomly assigned weights. Their transfer and threshold function is chosen ad hoc or informed by the expertise of the user. Each layer is represented as a matrix of weights with an additional vector of bias weights. Backpropagation is then used to train the weights until performance is satisfying. Once training is complete, the ANN can be 'deployed' without continued learning (Norvig and Intelligence, 2002).

By contrast, the neuro-evolution approach seeks to reverse engineer the process that created brains in the first place: Darwinian evolution. The structure of the network and the weights are trained through the use of genetic algorithms, not backpropagation. While backpropagation is capable to change all weights of the network during the backwards path to optimize functionality, evolution modifies a 'genome' that changes the weights and selects from different versions of the network over many generations. New networks are created by mutations that directly or indirectly change the synaptic weights, which often find novel and very creative solutions to problems (Lehman et al., 2018).

The most commonly used approach in ANN evolution experiments is direct encoding, where genomes specify each weight of an ANN separately. This creates large genomes and an extremely large search space which needs to be explored one mutant at a time. Therefore, indirect encoding schemes have been developed, so that the sites of the genome are not necessarily representing individual weights or connections but often control larger aspects of the network topology and weight space. One of the most successful examples of such an indirect encoding is hyperneat (Stanley and Miikkulainen, 2002). The encoding is facilitated by a compositional pattern-producing network (CPPN) a form of sparse neural network similar to a genetic programming tree. This CPPN is then used to specify the weights of an ANN. In essence, this translation allows the reduction of the search space, or, because mutations carry a stronger 'punch', a more effective exploration of the space.

Biological neural networks are never directly encoded. Mutations in the genome affect components of synapses that are highly unlikely to change one synapse at a time. Instead, mutations affect network topology and synaptic weights through two indirect mechanisms: First, development in time, under continuous feedback from the genome, leads to network structures and synapse types that could not be 'read' in the genome; second, most mutations will affect components that change the properties and weights of many synapses of a certain time simultaneous (think neuromodulators). Indirect encoding of the hyperneat type appproximates only this second type of indirect encoding. By contrast, developmental 'unfolding' of genome information is a time- and energy-consuming process that considerably complicates the encoding and computation of the network to be tested in evolutionary experiments. As such, it may appear disadvantageous for computational approaches, yet developmental encoding is an unavoidable basis of all biological neural network whose (dis-) advantageous properties remain largely unexplored.

Developmental Neural Networks (DNNs) are types of ANNs that employ a developmental process. While there is no sharp delineation between developmental and other indirectly encoded networks, a good rule of thumb might be, that the translational step in developmental encodings takes multiple updates. Hence, a DNN is not just based on mapping, but mapping facilitated by a process. This process, so to speak, "unfolds" the information stored in the genome over multiple time steps, instead of a one-off translation as in many indirectly encoded systems. Note that some indirectly encoded systems apply rules or operators that change the network sequentially, without applying the same rules over and over again, as an L-system for example would do (Lindenmayer, 2009). We therefore suggest to distinguish developmentally encoded from other indirectly encoded systems by this criterion. If a rule or rules are applied continuously and repetitively over multiple time steps changing the network over time, we speak of a developmental network, while indirect encodings might be sequential in nature, but do not repeatedly apply rules or changes.

The developmental process, beyond changing the search space drastically, provides the additional benefit of being adaptive. Depending on the presence of a certain signal or environmental condition, a network can differentiate into a more specialized version. This allows the genome of a developmental network to encode multiple variants, each specific to a certain environments (Miller, 2003; Miller and Wilson, 2017). Similarly, we can imagine that the developmental rules remain active during the lifetime of the network, which allows for constant fine-tuning and adaptation. This concept is also known as learning to learn (Schmidhuber, 1987). While there are implementations of systems which evolve rules that control autonomous lifetime learning (Sheneman and Hintze, 2017), we only know of one system that does both, implement a developmental encoding which remains also active during the lifetime of the network to facilitate learning (Miller et al., 2019).

The genomic encoding of biological neural networks ensures flexibility, adaptability and variability. A single genome can in fact lead to stochastic differences in neural networks that are selectable at the level of behavior (Linneweber et al., 2020). In addition, a single genome can produce different neural networks based on environmental differences. Biological neural networks are not only selected for functional performance under certain conditions, but also for robustness to unpredictable changes of these conditions. For example, temperature affects distinct developmental processes differently. Slower interactions between synaptic partners may allow more neurons to stabilize contacts, while higher interaction kinetics can effectively prevent certain synaptic partnerships and thereby change the network (Kiral et al., 2020). The brains of many animals have evolved to develop functional networks across a wide range of temperatures. How a single genome encodes robustness to such environmental variability, and to what extent the resulting networks are in fact similar, are open questions in biology. The neuro-evolution approach offers the opportunity to probe these questions, while at the same time exploring the potential of developmental encoding for AI.

In this study we compare direct, indirect, and developmental encoded systems for their ability to evolve intelligent agents with problem solving abilities for two tasks under different environmental conditions. The first task requires the system to evolve robustness against external temperature fluctuations. Agents develop either at a high or a low temperature and are then placed into a maze. In the other experimental condition, agents develop at a high or low temperature but are then placed into two different mazes, where the temperature is indicative of the maze they will face. This will allow the developmental system to evolve the ability to create individually different responses for each maze based on the temperature of development.

We use gene regulatory networks (GRNs) as a model for the developmental step. In our hands, large GRNs are not easily amenable to evolution experiments, we set out to test combinations with indirect encoding before or after the developmental step. In biological systems, indirect encoding precedes the development of neural networks, providing a test case for a neuro-evo-devo system.

We show that, indeed, only developmental encoding following indirect encoding leads to successful evolution of robustness and adaptability of the system. We further find, that the evolutionary adaptation of these systems presents a much greater challenge than direct or indirect encoding schemes. Simpler systems can solve the problems they are applied to better, while the adaptability of developmental encoding implies multiple solutions based on a single genome.

A Computational Neuro-Evo-Devo Model

Our model is composed of multiple layers that translate the information encoded in a linear genome (list of numbers) into a Recurrent Neural Network (RNN) controlling an agent to navigate a 2-dimensional maze (see Figure 1 for an illustration). Each agent has 6 distance sensors facing 90, 45, 0, -45, and -90 degrees from direct forward, and one sensor facing backward. The distance to the first wall along the line of sight for a sensor is the value the sensor reads. Agents have two outputs, one defines a turning angle ($[-45°, +45°]$) and the other the forward movement. Negative values for forward movement are interpreted as back-

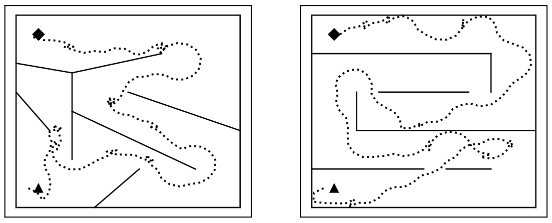
Figure 1: The two types of mazes used. The initial location is marked as a triangle, the goal is indicated by a diamond. The dotted line shows the path traversed by an evolved agent.

ward movement. Agents are allowed a lifetime of 200 units of time. For each unit of time they are awarded with a value equal to the Dijkstra distance to the target. As such, the fitness function computes the integral over the distance to the target.

We can experimentally modulate the complexity of the translation process that constructs an RNN from a genome, allowing differential comparisons of direct, indirect, and developmental encodings.

- **Direct encoding** The genome is sequentially translated to fill the weight matrix of the RNN, which controls the agent locomotion.
- **Indirect encoding using a CPPN (hyper encoding)** The genome is first translated to a CPPN. The CPPN is then used to populate the weight matrix of the RNN.
- **Developmental encoding** The genome is translated to fill the weight matrix of a gene regulatory network (GRN). This GRN is then run for 200 time steps, resulting in a gene expression vector. The expression levels of this vector is then sequentially translated into the weight matrix of the RNN.
- **Hyper developmental encoding** The genome is used to define a CPPN, which in turn is used to populate the weight matrix of the GRN. The GRN is run for 200 time steps, and the resulting vector of gene expression values is used to populate the weight matrix of the RNN.
- **Developmental hyper encoding** Here, the genome is translated to fill the weight matrix of a GRN. The GRN is run for 200 time steps, and the resulting gene expression vector is used to define a CPPN. The CPPN then defines the weights of the RNN.

Genome

The genome is a list of continuous values, initially drawn from a uniform random distribution to create each agent of the first generation. Upon replication, the genome undergoes independent point mutations with a probability of 0.001 per site, replacing each mutated site with a new uniform random value. Genomes can also experience duplications and deletions of 128-512 site long stretches, with a probability of 0.2 every time the genome is replicated.

Compositional Pattern-Producing Network - CPPN

The CPPN receives two continuous inputs values, and ultimately computes a single output value. The computation is distributed over 16 hidden nodes, each having two inputs and one output. The wiring, weights, and individual functions of each node are defined by an encoding input vector, depending on how it is used in the developmental model. Each input wire to a node applies a weight multiplication (weight $[-1, 1]$), and after a summation, a threshold function (sin, cos, tanh, or sigmoid) is applied resulting in the final node output. After all nodes are computed, a final neural network layer is used with 16 input values and weights, performing a summation transfer function and a sigmoid threshold function to return a single value.

Gene Regulatory Network - GRN

The GRN model allows the regulation of expression levels for N genes in the range $[-1.0, 1.0]$. Observe that these are not concentrations, as they do not sum to 1.0. To begin with, each gene i is set to an expression level of $E_i = 1.0/N$. The up- or down-regulation of each gene is defined by an NxN gene interaction matrix C. This matrix is defined by an input vector with one value per entry in the matrix (values from the range $[-1.0, 1.0]$). The new expression value E_i' of each gene E_i is computed as:

$$q_i = \sum_{j=0}^{N} E_j C_{i,j} \tag{1}$$

$$Q_i = \frac{2.0}{1.0 + e^{-\alpha q_i}} - 1.0 \tag{2}$$

$$E_i' = E_i - \beta(E_i - Q_i) \tag{3}$$

Where α is the temperature, and $\beta = 0.2$ a dampening factor. The next time step takes the expression levels of the last as an input. As a result, expression values can change over time.

Recurrent Neural Network - RNN

The RNN receives 6 input values, and together with 4 recurrent values computes the 4 new recurrent and 2 output values in a single layer update. This implies the weight matrix for the RNN to be $10x6$. Each node uses a summation aggregation and hyperbolic tangent threshold function.

Genetic Algorithm

For each experimental condition we performed 50 replicate evolutionary runs. For each replicate, a randomly generated population of 100 agents was evolved for $5,000$ generations. At the end, the line of decent (LoD) was reconstructed, and every 50th organism on the LoD recorded for later analysis.

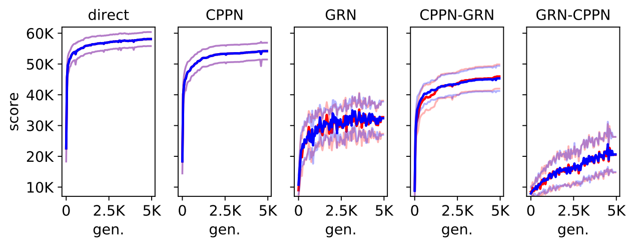

Figure 2: In blue the performance in the cold development, in red the performance in the hot development. The shaded lines indicate the standard error over the 50 replicate experimental runs performed.

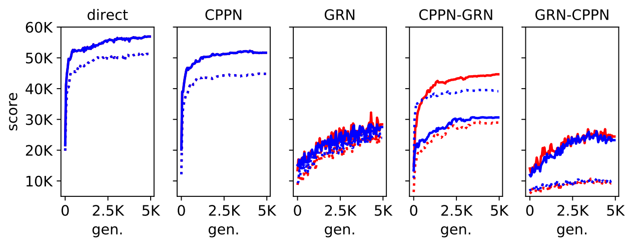

Figure 3: In blue the performance in the cold development, in red the one in the hot development. The solid line represents performance in maze A, the dotted one performance in maze B. The average over 50 replicate experiments is shown.

Results

Five different systems were tested: direct encoding, indirect encoding, GRN, CPPN-GRN, and GRN-CPPN. For each generation during evolution, one of two randomly picked temperatures (α of 0.8 or 0.5) was used during the developmental step, and the population was allowed to evolve in environment B. Observe that only those systems that use a GRN can technically respond to the temperature, the “direct” and “CPPN only” implementations served as controls. For the evaluation, agents along the line of decent were again evaluated in environment B collecting more data. We find that both direct and indirect encodings perform well on average. The implementation using the GRN or the GRN-CPPN performed poorly, while the CPPN-GRN implementation performed better, but not optimally (see Figure 2). Under close inspection, most agents reach the target area, just not as fast as the direct or indirectly encoded agents (data not shown). The GRN and GRN-CPPN encodings very often got stuck in the maze quite early, and no replicate could be found consistently reaching the goal. Hence, developmental encoding only produced agents with good problem solving abilities if preceded by indirect encoding, similar to biological systems. An interesting observation that will be further analyzed in the future.

Next, we tested the same five systems but under predictive developmental conditions. Here, agents developed again in either the hot or cold condition (α of 0.8 or 0.5). however, they were then placed in maze A or B according to the temperature they developed in. This allows a developing system to take advantage of the temperature in order to develop into two distinctly different configurations: one for each maze. As before, direct or indirect encoded systems adapt appropriately, whereas the direct encoded GRN or the GRN-CPPN variation performs poorly (see Figure 3). The most interesting observation comes from the CPPN-GRN encoded system. During evolution, the temperature presentation was consistently correlated with the maze presentation (A or B). Consequently, when tested on all four temperature and maze combinations, we find their performance optimal under the conditions they experienced during development. This difference in performance also proves that the developmental process resulted in different controllers, otherwise the performance would be more similar as in the case of the direct and indirectly encoded versions.

Discussion

We present a computational neuro-evo-devo system that seeks to incorporate the constraints and opportunities associated with the unavoidable developmental encoding characterisitic of biological neural networks. The system also includes a dependence on temperature, as every natural system does. Additionally, the system presented here also includes an indirect linkage between the genome and the factors controlling the developmental process, as single genes neither determine a single neural connection, nor a specific developmental detail. We find that this arrangement of indirect encoding controlling a temperature dependent developmental step (CPPN-GRN) performs better than a direct encoded developmental step, or if the developmental step is directly encoded and then translated indirectly into the neural controller (GRN-CPPN).

This neuro-evo-devo system immediately shows two important properties that natural systems also show. When facing temperature changes or other uncertainties, natural systems develop robustness towards these fluctuations, as does the computational system presented here. When the temperature changes are predictive of the environment, the system presented here, takes advantage of this information and development produced two distinct controllers independently optimized for the environment they face. Interestingly, the direct and only CPPN encoded version adapt much easier and result in controllers with much higher performance. This is obviously more important when these systems are trained to solve tasks, but as they do not experience the same problems as natural systems do, they also can not evolve the same type of robustness or adaptability. What seems like a performance advantage of none developmental systems might be obfuscating capabilities and research opportunities developmental systems could provide for more biological oriented modeling questions.

References

Kiral, F. R., Linneweber, G. A., Mathejczyk, T., Georgiev, S. V., Wernet, M. F., Hassan, B. A., von Kleist, M., and Hiesinger, P. R. (2020). Autophagy-dependent filopodial kinetics restrict synaptic partner choice during drosophila brain wiring. *Nature communications*, 11(1):1–14.

Lehman, J., Clune, J., and Misevic, D. (2018). The surprising creativity of digital evolution. In *Artificial Life Conference Proceedings*, pages 55–56. MIT Press.

Lindenmayer, D. (2009). *Forest pattern and ecological process: a synthesis of 25 years of research*. CSIRO publishing.

Linneweber, G. A., Andriatsilavo, M., Dutta, S. B., Bengochea, M., Hellbruegge, L., Liu, G., Ejsmont, R. K., Straw, A. D., Wernet, M., Hiesinger, P. R., et al. (2020). A neurodevelopmental origin of behavioral individuality in the drosophila visual system. *Science*, 367(6482):1112–1119.

Miller, J. F. (2003). Evolving developmental programs for adaptation, morphogenesis, and self-repair. In *European Conference on Artificial Life*, pages 256–265. Springer.

Miller, J. F. and Wilson, D. G. (2017). A developmental artificial neural network model for solving multiple problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 69–70.

Miller, J. F., Wilson, D. G., and Cussat-Blanc, S. (2019). Evolving developmental programs that build neural networks for solving multiple problems. In *Genetic Programming Theory and Practice XVI*, pages 137–178. Springer.

Norvig, P. R. and Intelligence, S. A. (2002). *A modern approach*. Prentice Hall.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München.

Sheneman, L. and Hintze, A. (2017). Evolving autonomous learning in cognitive networks. *Scientific reports*, 7(1):1–11.

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