The Journal of Mind and Behavior
Winter and Spring 2008, Volume 29, Numbers 1 and 2
Pages 195–216
ISSN 0271–0137

The Evolution of a Cognitive Architecture for Emotional Learning from a Modulon Structured Genome

Stevo Bozinovski

and

Liljana Bozinovska

South Carolina State University

The paper addresses a central problem in evolutionary biology and cognitive science; evolution of a neural based learning phenotype from a structured genotype. It describes morphogenesis of a neural network-based cognitive system, starting from a single genotype having a modulon control structure. It further shows how such a system, denoted as GALA architecture, growing its own recurrent axon connections, can further develop into various structures capable of learning in different learning modes, such as advice learning, reinforcement learning, and emotion learning. The paper particularly considers the emotion learning systems and their motivational structure. A simulation experiment is provided to illustrate the theoretical issues discussed.

Keywords: modulon-based genotypes, motivation–emotion structures, evolutionary cognitive systems

In this work we propose a model for the development (morphogenesis) of a cognitive system capable of emotion learning, commencing from a structured genome. This work is a research effort in evolutionary biology and is related to other efforts in that direction (e.g., Bull, 1997; Cangelosi, Parisi, and Nolfi, 1994; Eggenberger, 1997; Reil, 1999; Vaario, Ogata, and Shimohara, 1997). It

This research work has been supported in part by the NSF EPSCoR grant Nr EPS 0447660 extension 2005–701 to the South Carolina State University, the PI being the first author. The authors would like to thank the Guest Editor for improving an earlier draft of the paper. Requests for reprints should be sent either to Stevo Bozinovski, Ph.D., AI/Robotics/Biocybernetics Laboratory, Department of Mathematics and Computer Science, South Carolina State University, Orangeburg, South Carolina 29117, or to Liljana Bozinovska, M.D. Ph.D., Neuroscience and Electrophysiology Laboratory, Department of Biological and Physical Sciences, South Carolina State University, Orangeburg, South Carolina 29117. Email: sbozinovski@scsu.edu or lbozinov@scsu.edu

is also a contribution to work on designing plastic evolvable systems based on neural networks (Elman, 1993; Gruau and Whitley, 1993; Nolfi, Miglino, and Parisi, 1994). The paper is a continuation of our previous efforts to develop a sound theory of consequence driven systems (Bozinovski, 1995). Relation of this theory to other research in emotion is discussed elsewhere (Bozinovski, 2003). In this paper, we will firstly describe the basic concepts of our theory, and then we will introduce new concepts to derive a morphogenetic mechanism for developing axon reconfigurable neural architectures that are capable of learning.

The Framework: Consequence Driven Systems Theory

Consequence driven systems are systems that are capable of building the concept of a consequence (for example, a concept of a response of the behavioral environment to a previously performed agent behavior). Consequence driven systems theory is an attempt to recognize, understand, and formalize this and related issues. Such systems need to be based on a model of representative architecture that will ground notions such as motivation, emotion, memory, intelligence, disposition, anticipation, curiosity, confidence, and behavior, among others. Furthermore, in order to be realistic, such systems need to be based on an architecture that can be evolved from a genotype that contains a message from the genetic environment.

Origin

Consequence driven systems theory originated in an early reinforcement learning research effort to solve the "assignment-of-credit" problem using a neural network. This effort was undertaken in 1981 within the Adaptive Networks (ANW) Group at the Computer and Information Science (COINS) Department of the University of Massachusetts, Amherst. Two instances of the assignment-of-credit problem were considered: the maze-learning problem and the pole-balancing learning problem. Two learning architectures were proposed as candidates for solving those problems: the Actor/Critic (A/C) architecture and the Crossbar Adaptive Array (CAA) architecture (Figure 1).

Although both architectures were designed to solve the same problem, the A/C architecture effort was challenged by the mazes from animal learning experiments where there are several neutral states and there is only one rewarding state (food). In contrast, the CAA architecture effort was challenged by the mazes from the VAX/VMS 1981 computer game *Dungeons and Dragons*, where there is a single goal state but many rewarding and punishment states along the way. So, from the very beginning the CAA effort adopt-

ed the concept of dealing with pleasant and unpleasant states, feelings and emotions, and genotype predispositions.

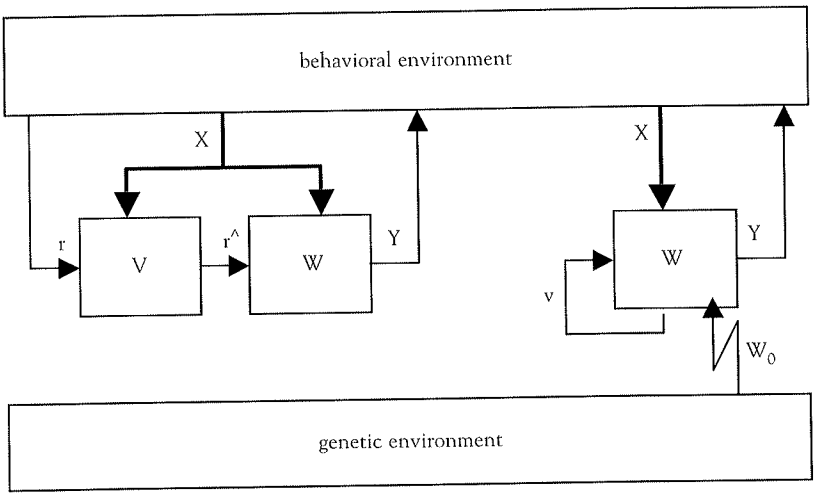

Figure 1: The A/C architecture (left) and the CAA architecture.

As Figure 1 shows, the obvious difference is that the A/C architecture needs two identical memory structures, V and W, to compute the internal reinforcement and the action, while CAA architecture for both computations uses only one memory structure, W, which is the same size as one of the A/C memory structures. The most important difference, however, is the design philosophy for reinforcement. In contrast to A/C architecture, which uses an external reinforcement r to compute internal reinforcement $r^{\wedge}$, CAA architecture uses genetically-introduced feelings and emotions. Crossbar Adaptive Array introduces a genetic environment and the concept of a *genome string* which defines the initial state, W_0, of the memory. The initial genome defines desirable states of the environment. Finally, by introducing the concept of *state evaluation*, CAA is able to develop a learning scheme using emotional value v, and does not require any external reinforcement.

Simulation experiments based on these architectures were carried out within the ANW group. The CAA approach proved more efficient and in fact was the only approach reporting a solution of the credit assignment problem in 1981 (Bozinovski, 1981a, 1981b, 1982). The A/C architecture reported its solution in 1983 (Barto, Sutton, and Anderson, 1983). The original CAA idea of having one memory structure for crossbar computation of both state values and action values was later implemented in reinforcement learning systems such as the Q-learning system (Barto, Sutton, and Watkins, 1990; Watkins, 1989). Q-learning (Watkins, 1989) uses exactly the same memory structure as

the CAA memory structure W, and denotes it as the Q-table. The main difference between the CAA and Q-learning approaches lies in the use of external reinforcement r. The CAA synaptic plasticity (learning) rule is in the form $w'_{ij} = w_{ij} + v'_k$, while the Q-learning rule is in the form $w'_{ij} = (1 - \alpha)w_{ij} + \alpha(r_{ij} + \gamma v'_k)$ where w_{ij} is the crossbar value (or Q-value) for action i in state j, v'_k is anticipated emotion in state k, r is the immediate external reinforcement in state j after performing action i, γ is a discount factor, and α is a forgetting parameter. In the CAA approach there is no external reinforcement: the parameter r could be considered only as an *internal cost*, $-c_i$, of performing action i, in a synaptic plasticity rule in the form $w'_{ij} = w_{ij} + v'_k - c_i$ (Bozinovski, 1995). Dyna architecture (Sutton, 1990) extends the concept of a crossbar adaptive array, introducing more arrays in order to build and use a model of the environment. We denote this extension as a "crossbar adaptive tensor (CAT) architecture" (Bozinovski, 1995).

The CAA approach introduced systems that can learn without external reinforcement — all the external reinforcement plasticity rules are classified into the supervised learning class of rules. We introduced a taxonomy of learning systems (Bozinovski and Bozinovska, 2001) which divides learning into supervised and unsupervised. Supervised learning contains advice learning and reinforcement learning, while unsupervised learning contains emotion learning and similarity learning. In supervised learning systems, a supervisor (e.g., teacher or other environmental feature) can give external reinforcement and/or advice as to how the agent should chose future actions. In unsupervised (self-supervised) systems, which have no external teacher of any kind, the agent has to develop an internal state evaluation system in order to compute an emotion. Other researchers, such as Gadanho (1999), recognize the distinction between external reinforcement learning as supervised learning and internal reinforcement learning as self-supervised learning. A kind of unsupervised learning can also be observed in agents that use some measure of similarity in tasks of adaptive classification (clustering) of input situations or data. Reinforcement learning (e.g., Barto, 1997) can be considered as a part of the supervised learning class (see also Peshkin and Savova, 2002).

Main Concepts of the Theory

Three Environments: Behavioral, Genetic, Internal

Consequence driven systems theory assumes that agents should be considered as three-environment systems. *Behavioral environment* is where the agents express their behavior. Agents have their *inner environment* where they build models of the behavioral environment. Some agents are able to build models of themselves as well. Agents are also connected to a *genetic environment* from which they receive initial conditions for existing in the behavioral environ-

ment. The initial knowledge transferred through the *imported genome* properly reflects, in the value system of the agent, the dangerous situations potentially encountered by agents in their behavioral environment. It is assumed that all agents import some genomes at the time of their creation, but not all of them are able to export their genomes after a learning period. This concept can be applied for biological as well as for non-biological agents. However, for non-biological agents, instead of genetic environment we use the concept of a generic environment. For example, a BIOS–ROM in computer systems is an example of an imported *generome* for a non-biological agent.

Understanding Interactions by Parallel Programming

In consequence driven systems theory, parallel programming is understood as *a way of thinking* about interactions. Parallel programming was used as a way of carrying out the experiments of pole balancing learning, where the environment (pole balancing dynamics) runs on one VAX/VMS terminal, while the CAA controller runs on another terminal (Bozinovski, 1981b). To the best of our knowledge, this was the first parallel programming application in neural network research. We will present a parallel version of the CAA agent–environment interaction below.

Time as Input/Output Concept: GALA Architecture

The theory emphasizes that *an agent's architecture should be able to understand temporal concepts in order to be able to self-organize in an environment.* In an agent's architecture the past tense is associated with the evaluation of its previous performance, the present tense is associated with the concept of emotion that the agent computes toward the current situation (or the current state), and the future tense is associated with the advice the agent will acquire for its future behavior.

The Crossbar Adaptive Array Architecture

Crossbar Adaptive Array architecture (Figure 1 above and Figure 2 below) is an instance architecture, evolved from the GALA architecture as an *emotion learning agent.* In a crossbar fashion, this architecture computes both state evaluations (emotions) and behavior evaluations. It contains four basic modules: crossbar learning memory, state evaluator, a behavior selector with a set of behaviors to select from, and a personality module providing personality parameters (Bozinovski and Bozinovska, 2001). In its basic routine, the CAA architecture first computes the emotion of being in the current state, and then, using a feedback loop, it computes the possibility of choosing at a future

time the behavior to which the current state is a consequence. The state evaluation module computes the global emotional state of the agent and broadcasts it to the memory. Using some kind of behavioral algebra, the behavior computation module initially performs a curiosity driven default behavior, but is gradually replaced by a learned one.

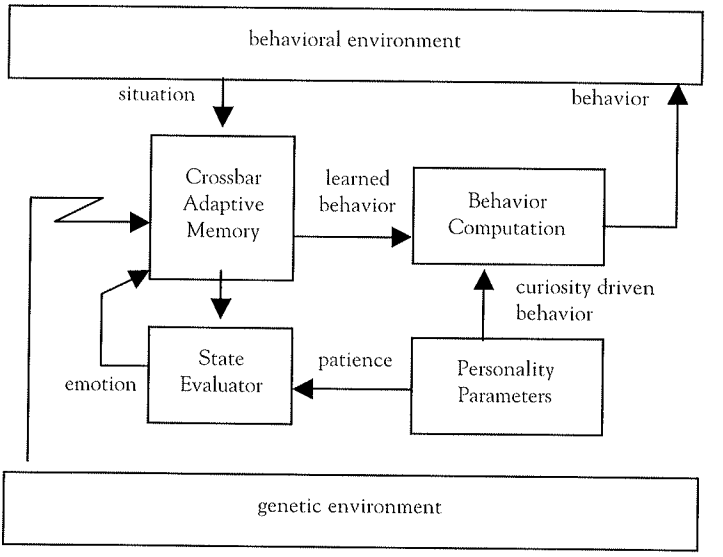

Figure 2: Crossbar adaptive array architecture.

The fourth module defines specific personality parameters of a particular agent, such as searching curiosity, emotional tolerance (patience), etc. It also provides some physiological parameters, such as urge signals. The urge signals produce behaviors that interrupt the behavior algebra of the agent. The fourth module is also engaged in computing export genomes of the CAA architecture.

Evaluate States, Backpropagate Emotions, Remember Behaviors

The consequence driven systems theory introduced the principle of remembering *only the behavior evaluations, not the state evaluations*. The behaviors are evaluated by their consequence states. Behavior evaluations are the only values stored. There is no need to store the state values, since they can be computed from the behavior values. The importance of this principle was emphasized by Watkins (1989) who established relations between dynamic programming (Bellman, 1957) and delayed reinforcement learning.

What are Motivations, What are Emotions: Motivational Graphs as Internal Mental Representations

In the consequence driven system theory emotions and motivations are *basic concepts of cognitive representations* of an agent. One distinguishes between motivations and emotions such that *motivations are associated to behaviors, while emotional values are associated to states of an agent.* Below we describe representational concepts for motivations and emotions: motivational graphs and motivational polynomials.

1. Motivational graphs. A motivational graph (or motivational net) is a representational concept for the value system of an agent (Bozinovski, 2003). Environmental situations are represented as emotionally valued nodes. The emotional value can be represented in different ways (for example, by using numerical values) but stylizations of facial expressions are preferred whenever possible (Bozinovski, 1982). Transitions between states are behaviors. The term "behavior" covers both the simple actions and also the possible complex network of actions. Some states are not reachable by the behavioral repertoire of the agent. The emotional value is given to a state either by the genetic mechanism, or by learning, after the agent visits that state. A state can be a neutral one, and can change its emotional value after being visited. There are different concepts of what an agent learns in an environment. It can learn the whole graph, like a cognitive map, or it can learn only a policy (the set of states and behaviors associated to those states). In the case of policy learning, the environment itself provides the actual map (the interconnection network between the states). The motivational graph concept includes the possibility that the environment is non-deterministic, or stochastic, in the sense that after executing the behavior, say B2 in state S1, it is not guaranteed that the next state will be the anticipated state S2. It is possible that after executing B2 the environment will present a state different than S2. However, it is assumed that the cost for performing behavior B2 has already been incurred after executing a behavior, even though the anticipated state is not obtained.

2. Motivational polynomials. For description of a motivational graph we use constructs which we call motivational polynomials. We use a kind of hieroglyph as symbol for expressing emotions and motivations (for example, the hieroglyph ☺). Each state is assigned an *emotional value* of being there. A state represents the environmental situation that the agent is currently in. Current state is defined by some color, for example grey.

A situation can have *features*. A set of features associated with a situation can be for example {x, s2,  ,  ,  ,  ,  ,  ,  ,  ,  ,  , . . .}. A situation can be represented by a monomial, such as S1, or  , or by a polynomial of its features, such as  +  . Each state is assigned an *attractor*, something that can possibly attract the agent toward that state. An attractor

can be represented by a motivational polynomial, example being + + + , or by a monomial, examples being and . If there is no distinguished attractor, the state can be evaluated by ø or left blank. It is assumed that there are also negative attractors (repellers). Examples of repeller monomials are , , , , . Depending on the attractor, some situations can be goal situations. Each state has a *repertoire of behaviors* that can be executed from that state. Behaviors can be denoted by symbols B1, B2, B3 . . . . The notation B2/S1 means "behavior B2 given state S1." Behaviors can also be represented by emotional monomials such as , , , , or as a polynomial, e.g., + .

Each behavior is assigned an *anticipated purpose*. The purpose for a behavior can be the set of attractors associated with the anticipated consequence state. Each behavior is also assigned a *cost* for executing that behavior. The cost can summarize a behavioral effort. The cost can be represented in *cost units*, an example being -units. However each agent can have an attitude, a feeling toward a cost that can be measured in emotional units. Each behavior is assigned a motivation. The notation (B2/S1) means motivation for performing B2 in S1. It is assumed that motivations are functions of anticipations, which in turn are functions of attractors and repellers.

Each state is assumed to have a possibility to execute a behavior, which we denote as an urge. The term "urge" is adopted from Loehlin (1968), but we use it to represent an urgent *interruptive behavior* used to service a need of the agent. In contrast to motivated behavior, urge behavior is a result of energy supplying and metabolism balance needs. It is usually assumed that the urge produces a circular path in a motivational graph, but the model allows other paths produced by urges. As special types of motivations, urges are usually represented by monomials like , , , , . Usually a monomial like is not considered an urge, but the model includes that possibility too.

Genome Driven Morphogenesis: From a Genome to a Neural Architecture

The GALA Modulon

We propose that a *hierarchically structured genome* is needed for development of a modular neural structure. More specifically, we propose a modulon-level hierarchy of genetic control structures and two types of genes: value encoding genes and function encoding genes. For our construction of a reconfigurable neural controller we assume a genotype consisting of genes, operons, and regulons, co-regulated by a single modulon. Here we will describe the *GALA modulon*, the genome structure out of which we will develop our neural GALA architecture. The *GALA modulon* (Figure 3) consists of the following

co-regulated genetic structures: *neur*-regulon, *learn*-gene, *eval*-gene, *clone*-gene, and *struct*-operon.

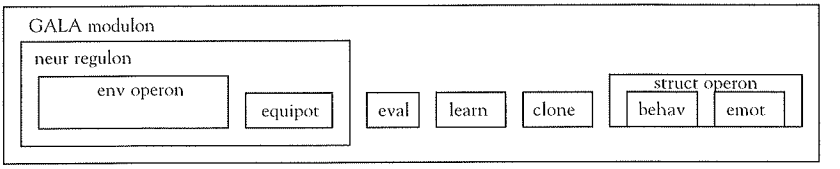

Figure 3: A one-chromosome genome as genotype for a neural network.

Neur-regulon consists of an operon and a gene, denoted as *env* operon and *equipot* gene. The *env* operon is the *environment encoding operon*. It expresses m genes $[w_{01}, w_{02}, \ldots, w_{0j}, \ldots, w_{0m}]$ which encode m synaptic weights of the phenotype neural cell. The value encoding genes $\{w_{0j}\}$ encode a motivation–emotion system of the phenotype agent. *Equipot* gene encodes the neural threshold which is assumed to be a function of the encoded weights. The *equipot* gene can be denoted as $f(w_{01}, w_{02}, \ldots, w_{0j}, \ldots, w_{0m})$ since it is a function of the *env* operon. *Learn*-gene encodes the synaptic plasticity rule for the synaptic weights, can be denoted as $[h(w, u, x, r)]$, where w is a synaptic weight, x is the synaptic input, and u and r are advice input and performance evaluation input, respectively. *Eval*-gene encodes a function, $g(w_{10}, w_{20}, \ldots w_{i0}, \ldots w_{n0})$. The number n is specified by the GALA modulon, as the number of neuron cell clones produced by the co-regulated *neur*-regulon. It is assumed that the functions $f(\ )$, $g(\ )$ *and* $h(\ )$ are *just specified* in the regulon; the regulon does not actually compute them. *Clone*-gene controls the parallel programming feature of the morphogenesis of the neural GALA architecture. It specifies number of copies that will be cloned of the phenotypes produced by the *neur*-regulon and *eval*-gene. *Struct*-operon contains all the structural information as how to assemble the whole GALA structure. Among others, it contains two genes, *behav*-gene and *emot*-gene. The *behav*-gene encodes a neuron that selects the output behavior, and the *emot*-gene selects the global emotion of the whole system. Not all the genes are expressed in the various phenotype organisms that could be generated by this genome. The *struct*-operon contains additional genes that decide, depending on the environment conditions, whether some genes will not be activated.

Let us note that more consistent with standard genetics naming convention (Brown, 1998) would be the following description of the GALA modulon structure: 4gal_312 (3neu2W_P (2envW_W (synW$_1$, . . .,synWn), equP), evaL, cloN, leaR, 2strAT(behA, emoT))). The leading numbers show the level of the control structure, 4 being modulon, 3 being regulon, 2 being operon: genes have no leading number. However in the text below we will use the naming we introduced in Figure 3.

Neural Cytogenesis

When the GALA modulon undergoes its cytogenesis process it firstly activates the *neur*-regulon, which in turn activates the *env*-operon. The resulting phenotype is a neural cell that contains the expressed m synaptic weights and has m synaptic inputs that are associated with m situations from the behavioral environment. While the GALA modulon merely specifies the output function, the phenotype neuron actually *computes* it. In computation of the neural output, the neuron first computes its (summary postsynaptic) neural potential $s_1 = \Sigma_{(1,m)}\ w_{1j}x_j$, where w_{1j} is the *influence* (weight) of the j-th synapse in computing the s_1. The *equipot*-gene specifies that the neural threshold is indeed equal to the neural potential, i.e., $\theta_1 = s_1$. Having that, the output signal computation is simple. Using *maximum potential principle*, we have:

$$y_1 = 0;\ y_1 = 1 \text{ only if } s_1 = max\ (s_1, \theta_1) \qquad [1]$$

The expression of the *learn*-gene gives the synaptic plasticity rule. The *synaptic plasticity* mechanism is defined as

$$w'_{1j} = w_{1j} + r_1 x_j u_1 \qquad [2]$$

This means that the neuron learns by a three-factor correlation rule, specifying that the past performance r and the current situation x and the future behavior advice u are needed in order for a neuron to learn how to perform later in situation x_j, if it is encountered again.

Neural Tissue Genesis

Next step is obtaining a neural tissue. The first neuron clones itself and the *struct*-operon defines a neural structure consisting of two neurons. The obtained structure grows collateral connections and connects the threshold values making them equipotential (or isopotential). An assumption of this theory is that the neural networks that perform certain tasks have a common threshold value. The computation of the common threshold value can be performed in various ways, and we chose the maximum potential principle. The common threshold is computed as:

$$\theta = max\ (\theta_1,\ \theta_2) = max\ (s_1, s_2) \qquad [3]$$

Now, since θ has a value of the greater of s_1 and s_2, the only neuron that will fire is the one with greater value of its neural potential. Thus, the equipotential neural network has an *inherent maximum selector mechanism*. In neural networks theory the maximum selector principle is usually implemented as synaptic

lateral inhibition *network*, so called "winner take all" mechanism, which becomes rather complex in the case of many thousands of neurons. We argue that the neural equipotential (equithreshold) mechanism is simpler yet biologically plausible. For the two-neuron network, the synaptic plasticity rule is:

$$w'_{ij} = w_{ij} + u_i x_j r, \text{ for } i = 1, 2 \text{ and } j = 1, \ldots, m \qquad [4]$$

It is easy to generalize for a network of n neurons, obtained after n times cloning of the initial neural cell. The resulting neural structure will be able to implement the maximum selector principle according to the relation

$$y_i = 1 \text{ only if } s_i = \theta \qquad [5]$$

where θ is the *common equipotential threshold* computed as $\theta = max\ \{\theta_1, \theta_2, \ldots, \theta_i, \ldots, \theta_n\}$. Also for the n-neuron network, the synaptic plasticity rule is:

$$w'_{ij} = w_{ij} + u_i x_j r, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \qquad [6]$$

Note that we assume that the system learns from the performance evaluation given to the whole structure (the organism as a whole) and not to a single neuron.

Evolving the GALA Architecture

Now let us consider cytogenesis of other neurons produced by the *eval*-gene. The *eval*-gene produces a neuron e_j that monitors the synaptic weights w_{*j} that are in the j-th column of the memory matrix. Various functions can be implemented for the e_j neurons, for example, the maximum function and the sum (softmax) function. To be consistent with all the functions in the same regulon, let us chose the maximum function, denoting $e_j = max\ \{w_{*j}\}$, where $*$ signifies all the weights associated with the j-th environmental situation. For the output of the system, the equithreshold selector will chose the maximum function, such that $e = max\ \{e_*\}$ where $*$ denotes all the neurons produced by the *eval*-gene.

Expressing the Struct-operon: Adding Output Modulation Neurons

The *struct*-operon will add one neuron that connects outputs from all the behavior-computing genes and one neuron that will collect information from all emotion-computing genes. Figure 4 shows the whole neural architecture including the output neurons that are the result of expressing the *behav* and *emot* genes of the *struct* operon. For sake of clarity, it shows only the first, i-th, and the n-th neuron produced from the *neur* regulon — the neurons in-between are not shown. Note that synapses $x_1 \ldots x_m$ apply to all the neurons.

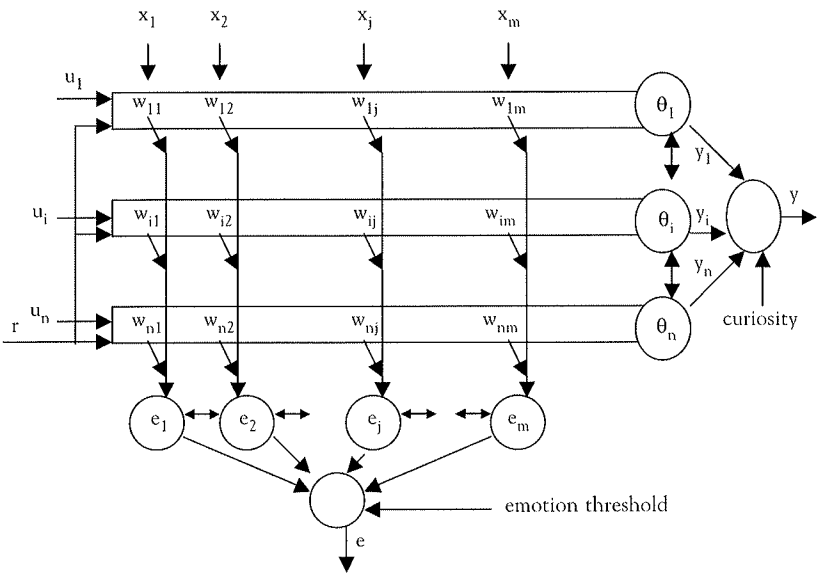

Figure 4: The genome driven morphogenesis of the GALA architecture.

Adding output neurons to the GALA neural structure enables greater flexibility in some applications and in understanding the agent's personality. Neurons can perform a kind of *behavioral algebra* and *emotional algebra*, respectively. They can perform various functions such as summing function, maximum function, and multiplexing function. As an example, the behavior-computing neuron can enable execution of other, curiosity-driven behavior, and is not limited to learned behaviors. Also, the emotion-computing neuron can accept influences from other neural systems before sending a signal about the emotional state of the system to the outside world and/or back to the learning structure of the agent.

Axon Reconnection Driven Morphogenesis: Evolving Learning Mechanisms

In this section we will consider further development of the GALA architecture using an axon growing mechanism. Recall the GALA architecture description as *GALA(in: x, u, r; out: y, e)* where x is current situation, r is past performance evaluation, u is future advice, y is current behavior, and e is current emotion. The same architecture can perform three types of learning: advice learning, reinforcement learning, and emotion learning by rewiring its axons and reducing inputs from the environment.

Advice Learning Agents

An advice learning agent is connected to the environment with all inputs and outputs, *GALA(in: x, u, r; out: y, e)*. The learning rule for this agent is

$$w'_{ij} = w_{ij} + r'x'_j u'_i, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \qquad [7]$$

where given a situation x_j, the supervisor advises u_i as to what behavior should be chosen in that situation. The advice is only given if the evaluation of a previous behavior is $r = 1$, which means that the *advice is needed*. If a previous behavior was correct, then $r = 0$ means that the advice is not needed. Note that some supervised learning systems do not include the parameter r in the learning process. However, we are only concerned with consequence driven teaching, where the teaching trial is applied as a consequence of the previous behavior of the learner, so the parameter r is needed. The notation $w'_j = w_{ij} + r'x'_j u'_i$ that we use is equivalent to the notation $w_{ij}(t) = w_{ij}(t\text{-}1) + r(t)x_j(t)u_i(t)$ which means that in the advice learning procedure all the three learning variables are present in the same learning step. Let us note that the emotion output in this case is not necessary to be computed, so the architecture can be described as *GALA(in: x, u, r; out: y)*.

Reinforcement Learning Agents

A reinforcement learning agent can be obtained by growing recurrent axon collaterals and connecting the current behavior output to the behavior advice input, given in the following description *GALA(in: x, r; out: y, e; recurrent: u* $\leftarrow$ *y)*. Now the agent inputs from the environment are only the situation and performance evaluation input. The behavior advice is computed internally: it is actually a previous behavior if it was evaluated as desirable. The learning rule now is:

$$w'_{ij} = w_{ij} + r'x_j y_i, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \qquad [8]$$

The notation $w'_{ij} = w_{ij} + r'x_j y_i$ is equivalent to notation $w_{ij}(t) = w_{ij}(t\text{-}1) + r(t)x_j(t\text{-}1)y_i(t\text{-}1)$. It is important to note that reinforcement learning introduces a shift in the time scale. The reinforcement is given by the environment as a consequence of previously taking the action i in the situation j. Reinforcement r is received from an *external teacher* (e.g., from the environment itself) as a scalar judgment rather than as advice judging the previous behavior of the agent. Let us also note that it is not necessary to compute the emotion, so the description can be *GALA(in: x, u, r; out: y; recurrent: u* $\leftarrow$ *y)*.

Emotion Learning Agents

The emotion learning agent can be obtained from a reinforcement learning agent by growing axon collaterals from the current emotional value output to the past performance evaluation input, *GALA(in: x; out: y, e; recurrent: u ← y, r ← e)*. The learning rule for this agent is:

$$w'_{ij} = w_{ij} + v'_k x_j y_i, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \qquad [9]$$

where $v'_k = e(x'_k)$ is the emotional value of the *consequence situation* x'_k which in turn is result of performing behavior i in the situation j. Another way of writing the above equation is $w_{ij}(t) = w_{ij}(t\text{-}1) + e(x_k(t))x_j(t\text{-}1)y_i(t\text{-}1)$. If we consider the cost per action, then we have:

$$w'_{ij} = w_{ij} - c_i y_i + v'_k x_j y_i, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \qquad [10]$$

where c_i is a cost for performing action i. If it is understood that x_k is a consequence *of* y_i in x_j, and we consider y_i and x_j as Boolean variables, the above equations can be written simply as:

$$w'_{ij} = w_{ij} + v'_k \qquad [11]$$

and

$$w'_{ij} = w_{ij} - c_i + v'_k \qquad [12]$$

We denote the two equations above as the "CAA learning rule" and the "benefit–cost CAA learning rule" respectively. In both of the learning rules the only connection to the environment is the situation x_j. No reinforcement from the environment is received. Both the variables v'_k and c_i, are computed as internal variables.

The advice learning and reinforcement learning agents do not necessarily require emotional output from the agent. They can learn from the advice and reinforcements they receive from the environment. For these types of agents there is no need for expressing the *eval*-gene of the GALA regulon. The emotional learning agent, in contrast, needs an internal emotional system in order to value the consequence of its behavior, and the *eval*-gene is essential for developing an emotional learning phenotype.

Measuring Units for Motivation and Emotion

We will now address the use of units for measuring emotion and motivation, since these are required in our simulation experiments.

Motivation function, ♥*(i/j)*. Represents the motivation for performing the behavior i given situation j. Its co-domain could be the ordered set, {☠, 💣,

⦸, ≈, ♥, 2♥, 3♥, . . .}. Measuring units are ♥-*units*. An example of an expression with that function is

$$♥(🚗/🌲) = 2♥ \qquad [13]$$

Emotion function 😐(*j*). Represents the evaluation of desirability of being in state *j*. The emotion function has its domain in the set of situations and its co-domain in some set of emotions. Usually a ternary co-domain set {☺, O, ☹} is sufficient use of this function in motivational nets. However, in some cases it is useful to measure the emotion in *emotional units*, like 2☺, 3☺, etc. An example of an expression with that function is

$$😐(🔑 + 🚂 + 🐟 + 🍽) = 4☺ \qquad [14]$$

Negative emotions are represented in ☹-units. A neutral value of an emotion is denoted as O (empty circle). States *j* can be represented by the situation the agent is in, or by features of that situation (as shown above), or by some combination of the situation features and the internal features of the agent.

The *anticipated emotion* function, or *motivation gain*, ☺'(*k*), represents the emotion of being in state *k*, which is a state on the way toward a goal. Since it is an expected emotion rather than a felt emotion, it is measured in motivational units, ♥. This emotion is signaled as an emotion signal in the process of learning.

Cost function, or *motivation loss*, 🕓(*i*). Represents the attitude associated with the anticipated effort of performing *i*. It is measured in (negative) ♥-units. Examples of expressions with that function are

$$🕓(🚤) = -3♥ \qquad [15]$$
$$🕓(🚀) = -9♥$$

Having all these definitions, we now define the *motivation learning function*, ♥'(*i*/*j*) as

$$♥'(i/j) = ♥(i/j) + ☺'(k) - 🕓(i) \qquad [16]$$

or shorter, in C^{++} notation:

$$♥'(i/j) \mathrel{+}= ☺'(k) - 🕓(i) \qquad [17]$$

Using the concept of motivational polynomials we can write several instances of this generic equation, examples being:

$$♥'(b/s_1) \mathrel{+}= ☺'(s_2) - 🕓(b) \qquad [18]$$
$$♥'(🚌/🏙) \mathrel{+}= ☺'(🚗) - 🕓(🚌)$$

Connecting the Genome to the Behavioral Environment: Motivational Structure of the Agent

Recall that the *env*-operon is the environment encoding operon. The value encoding genes $\{w_{0j}\}$ of this operon encode an emotion toward the respective environmental situations. Some of those genes represent desirable situations, some represent undesirable ones and some do not carry information about desirability. As an example, let the *env*-operon be a part of the genome inherited in an environment that contains nine relevant situations for the considered agent. The genes are represented as vector

[≈ ≈ ☹ ≈ ≈ ☺ ≈ ≈ ≈]

This means that the inherited genome points out that for this particular agent the situation S3 is undesirable while S6 is desirable, which defines non-modifiable weighted synapses in the phenotype neuron. The agent (species) carrying that genetic information will be attracted by situation S6 while avoiding situation S3. Note that in the process of cytogenesis of the initial neural cell, all the emotional values of the synapses with fixed weights are reproduced as well. When a species driven by its neural controller appears ("is born") in an environment, it carries fixed weight synapses but also synapses generated from neutral genes of the *env*-operon. The fixed weight synapses represent information obtained from the genetic environment that reflects the states of the behavioral environment. The neutral genes will define modifiable synapses, indicated by ≈. These synapses are modified as the result of a learning process. The modifiable synapses represent *motivational confidences*, or strengths. The element w_{ij} is the motivation strength of performing behavior Bi in the situation Sj. Using motivational symbols we can write it as ♥(Bi / Sj). Those motivational values can also be interpreted as tendencies, intentions, dispositions, or as other notions representing motives to perform some behavior.

After the learning process has occurred, the initial neural controller might have a mature structure as shown in the following emotion learning matrix:

situation	[S1	S2	S3	S4	S5	S6	S7	S8	S9]	
1st neuron	[≈	☠	☹	☠	≈	☺	≈	☠	8♥]	→ behavior B1
2nd neuron	[≈	2♥	☹	≈	≈	☺	☠	☠	≈]	→ behavior B2
3rd neuron	[≈	≈	☹	♥	♥	☺	☠	☠	≈]	→ behavior B3
situation evaluation	[O	☺	☹	☺	☺	☺	O	☹	☺]	

This matrix contains one situation vector, three neuron vectors and a situation evaluation vector. In this example all the motivations are neutral in sit-

uation S1, and the emotion in that state is neutral. In situation S2, there is a potential danger of performing B1, but it is compensated by two prior experiences showing that there is a path from this state toward a goal state. Situation S3 is defined genetically as a dangerous state in the behavioral environment. In situation S4 the species has learned that B1/S4 should be avoided and B3/S4 should be tried; if the behavior selection algorithm is a maximum-seeking one, this state is emotionally desirable. Situation S5 is a typical example of a learning process: it has been learned that B3/S5. Situation S6 is predefined genetically as a desirable state; in the behavioral environment it possibly contains an attractor. However, that is not represented in the genome: what is represented is only the emotional value of being in that environmental situation. Situation S7 can be evaluated differently depending upon the emotion computation algorithm. If the selection algorithm seeks a maximum, it can be evaluated as neutral. If, however, the behavior selection algorithm guarantees no-risk using the maximum principle (no risk from the stochasticity of the outcome), this situation might be evaluated as desirable. Situation S8 occurs after the learning is evaluated as a situation with no promising outcome. Evaluation of situation S9 shows that it assigns a high motivational value for performing behavior B1, possibly because a goal situation is easily reachable from S9, and thus an agent would seek a state in which it was able to execute this behavior.

Exporting Genomes: Lamarckian and Darwinian Way

The emotion learning matrix above shows the emotion evaluation string computed from the learning process. That string is a part of the *exporting env-operon*. That string is sent to the genetic environment to be assimilated by a new species that will later appear in the same behavioral environment. We assume that the output genome consists of several chromosomes, the *env*-operon being encoded as a separate chromosome. We also assume that this particular chromosome is not subject to a crossover process. However, crossover may take place among some other genes of the genome. So, as a result of the genome exporting computation, a mature agent (after learning) would be able to export, as a part of its genome, the following *Lamarckian chromosome*:

Lamarckian.output_chromosome1(speciesA) = (O☺☹☺☺☺O☹☺)

That is the case if genetic environment allows a Lamarckian evolution, the one where genetic information is acquired as a process of learning. To control the Lamarckian-type evolution, some threshold mechanism could be included in the genome exporting computation. If the exporting algorithm is of *Darwinian type*, then the export chromosome can be the same as the input *Darwinian chromosome*:

Darwinian.output_chromosome1(speciesA) = (OO☹OO☺OOO)

and some difference could occur only due to processes like mutation and crossover, rather then due to a learning process.

Illustration of the Theory: Simulation Experiment

We will now consider a simulation experiment in order to illustrate the presented theory. We will describe the genetic environment, the agent, the behavioral environment, and the result of the experiment in the behavioral environment, along with the emotion–motivation structure learned in the agent's memory.

Genetic environment. We assume a behavioral environment inhabited with two species, R1 and R2. That environment is connected to the genetic environment with two genomes, representing the mentioned species. Let us assume that the *env*-operons of both species are acquired by chromosomes having different encoding genes. Let the input chromosomes for the species be:

input_chromosome (R1) = (OOOOOOOOO☹OOOO☺OOOO☹)
input_chromosome (R2) = (OOOOOOO☹O☹OOOO☺OOOO☹)

This means that the sensory systems of both species can distinguish 20 situations in the environment. For both species, the situation 15 is considered desirable, while situations 10 and 20 are considered undesirable ones. The species R1 will consider situation 8 as neutral, while species R2 will consider it dangerous. The behaviors are executed with the same cost, such that the cost does not play a significant role in the simulation experiments. As an experimental hypothesis, it is expected that both species will develop different motivations and consequently different learned behavior.

The agent. The experiment is designed around the CAA architecture as shown in Figure 2 above. The CAA architecture is used as a neural controller of the two species in the experiment. The initial, genetic information about environmental situations is received through the genetic environment. The learning process is carried out through interaction with the behavioral environment. The memory matrix is used as a representation of the synaptic weights of the whole neural network. The learning routine is following: suppose in a situation s, a behavior b is executed and as a consequence a new situation is obtained from the environment. The new situation is evaluated on desirability, depending on either genetic information or learning information in the matrix column addressed by the received situation. Emotion is computed and is sent as a signal to the whole memory structure. Only one element, $\blacktriangledown(b/s)$, is updated by that emotion information (the one that indeed produced a behavior b in situation s, and is thus responsible for receiving the current situation as a consequence).

The agent–environment interaction. We can now go on to discuss the parallel programming version of the CAA agent computational procedure used in the experiment. Here (∗/∗) means matrix and (∗/s) means column vector. The program below contains two threads, CAA_agent and Behavioral_environment, that exchange information using the post/wait communication mechanism.

program CAA: agent–environment interaction
define environment graph $s'(*/*)$;
choose initial situation s_{init}: $s' = s_{init}$;
thread CAA_agent
import genome;
initialize crossbar elements ♥(∗/∗) from the imported genome;
repeat
state $s = s'$;
compute behavior b from ♥(∗/s);
post b;
wait s';
compute emotion signal ☺$'(s')$ from ♥(∗/s');
compute ♥(b/s) += ☺$'(s')$;
until 😐(s_{init})= ☺;
compute genome export from ♥(∗/∗);
export genome;
thread Behavioral_environment
repeat
situation $s = s'$;
wait b;
compute next situation $s' = s'(b/s)$;
post s';
forever;

The personality module, which is kind of an operating system for the whole CAA architecture, decides the end of the learning process. In learning how to behave in an environment (as is the case in this experiment), the end of the learning process is decided when a policy is assigned to the starting situation, since the policy learning goes backwards. At the end of the learning process, or at any time before, the personality module might decide to export a genome about what has been learned so far.

The learning procedure described above in terms of emotional and motivational functions is actually a secondary reinforcement learning procedure (Bozinovski, 1982). The convergence proof of the whole learning procedure is given in Bozinovski (1995). It is presented here as an illustration of a type of neural controller that can be evolved from a genome containing a modulon control structure.

Behavioral environment. For this experiment we chose an environment with 20 situations. It inhabits both the species R1 and R2. The situations 10 and 20 contain repellers for both the species, and situation 15 contains an attractor for both the species, and that has been reflected in the *env*-operon of the imported genome. Situation 8 carries no genetic message to species R1 but is reflected in the genome of the species R2 as an undesirable situation. Both species have S6 as the starting situation, and it is expected that both will learn a policy that will eventually lead them to the state 15.

Result of the experiment. Figure 5 shows the result of the behavior experiment in a behavioral environment that corresponds to the given genome from the genetic environment.

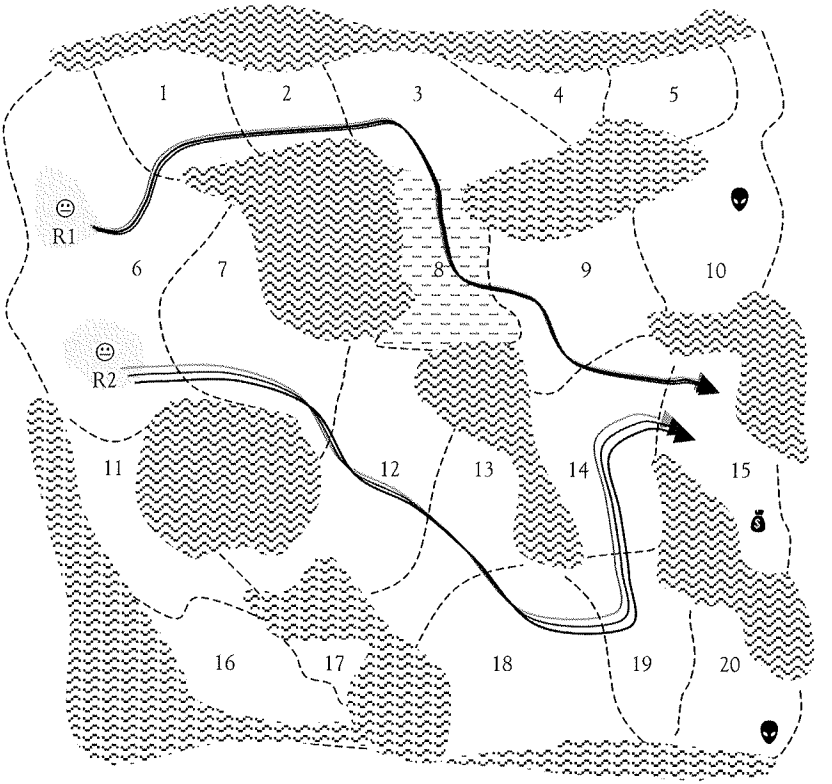

Figure 5: Learned behaviors in the species R1 and R2 due to different motivational mental representations of the environment.

This shows that as the result of developed plans in their mental representations, the two species indeed exhibit different behaviors in the same behav-

ioral environment, as hypothesized. The next matrix shows the final state of the memory of species R2 after the learning process that established the behavior shown in Figure 5.

situation	[1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20]
1st neuron	[≈	≈	≈	≈	☠	≈	2♥	☹	≈	☹	≈	3♥	☠	9♥	☺	≈	≈	6♥	8♥	☹]
2nd neuron	[≈	≈	≈	≈	≈	♥	≈	☹	≈	☹	≈	≈	≈	≈	☺	≈	≈	≈	≈	☹]
3rd neuron	[≈	≈	☠	≈	☠	≈	≈	☹	≈	☹	≈	≈	5♥	≈	☺	≈	≈	≈	☠	☹]

The matrix above shows the developed plan in the agent's mental representation for behaving in its behavioral environment. The plan is represented by the sequence of states 6–7–12–13–18–19–14–15 and is developed by learning a partial policy for behaving in this environment. As we can see, in some states (for example, situations 1, 2, 9, 11, 16, and 17) the agent will tend to execute the default behavior, which can be a random walk. In some situations (3, 5, 13, and 19) the agent has experienced undesirable consequences of some behaviors, and has memorized negative motivations for performing those behaviors. Note that what has been learned in the mental representation of the agent R2 is a kind of hill-climbing procedure. It is a *mentally represented hill of motivational values* (♥→2♥→3♥→5♥→6♥→8♥→9♥→☺) that has been built by the learning process. Finding itself on any level of the hill, the species will know how to proceed toward the goal.

Conclusion

The present work addresses a fundamental question in evolutionary biology and cognitive science: how to evolve, from its genome, a neural network that will be able to control agents capable of showing learning behavior. In particular, it considers the question of how that genome should be constructed. The framework used is the consequence driven systems theory. The basic result of this research is that in order to build such an evolvable agent *we need to start with a structured genome*, the one that contains hierarchical control structures such as modulons, regulons, and operons, for controlling various types of genes. The paper considers three stages of evolving a neural controller: (1) from genome to the first neural cells, (2) from neural cells to a neural assembly that functions as a whole, and finally (3) from the basic neural assembly to various instances of learning systems developed by growing axon connections. The paper also illustrates a mechanism how the genetic environment is related to the behavioral environment of an agent. It shows a mechanism by which genetic environment assigns initial emotional values of some behavioral environment situations. Having those situations as attractors and repellers, and using some motivation backpropagation mechanism in its learning process, a learning agent is able to build a policy for behaving in the behavioral environment.

References

Barto, A. (1997). Reinforcement learning. In O. Omidvar and D. Elliot (Eds.), *Neural systems for control* (pp. 7–30). San Diego, California: Academic Press.

Barto, A., Sutton, R., and Anderson C. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13*, 834–846.

Barto, A., Sutton, R., and Watkins C. (1990). Learning with sequential decision making. In M. Gabriel and J. Moore (Eds.), *Learning and computational neuroscience: Fundamentals of adaptive networks* (pp. 539–602). Cambridge, Massachusetts: The MIT Press.

Bellman, R. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bozinovski, S. (1981a). A self-learning system using secondary reinforcement: Report on the CAA network, ANW group report, November 25, COINS Department, University of Massachusetts at Amherst.

Bozinovski, S. (1981b). Inverted pendulum learning control. ANW Memo, December 10, COINS Department, University of Massachusetts at Amherst.

Bozinovski, S. (1982). A self-learning system using secondary reinforcement. In R.Trappl (Ed.), *Cybernetics and systems research* (pp. 397–402). Amsterdam: North Holland.

Bozinovski, S. (1995). *Consequence driven systems*. Bitola, Macedonia: Gocmar Press.

Bozinovski, S. (2003). Anticipation driven artificial personality: Building on Lewin and Loehlin. In M. Butz, O. Sigaud, and P. Gerard (Eds.), *Anticipatory behavior in adaptive learning systems* (pp. 133–150). Berlin: Springer Verlag.

Bozinovski, S., and Bozinovska, L. (2001). Self-learning agents: A connectionist theory of emotion based on crossbar value judgment. *Cybernetics and Systems: An International Journal, 32*, 637–669.

Brown, T (1998). *Genetics: A molecular approach*. London: Chapman and Hall.

Bull, L. (1997). On the evolution of multicellularity. In P. Husbands and I. Harvey (Eds.), *Fourth European Conference on Artificial Life*. Cambridge, Massachusetts: The MIT Press.

Cangelosi, A., Parisi, D., and Nolfi, S. (1994). Cell division and migration in a "genotype" for neural networks. *Network, 5*, 497–515.

Eggenberger, P. (1997). Creation of neural networks based on developmental and evolutionary principles. *International Conference on Artificial Neural Networks*, ICANN '97, Lausanne, Switzerland.

Elman, J. (1993). Learning and development in neural networks: The importance of starting small. *Cognition, 48*, 71–99.

Gadanho, S. (1999). Reinforcement learning in autonomous robots: An empirical investigation of the role of emotions. Doctoral dissertation, University of Edinburgh, Edinburgh, Scotland.

Gruau, F., and Whitley, D. (1993). Adding learning to the cellular development of neural networks. *Evolutionary Computation, 1*(3), 213–233.

Loehlin, J. (1968). *Computer models of personality*. New York: Random House.

Nolfi, S., Miglino, O., and Parisi, D. (1994). Phenotypic plasticity in evolving neural networks. In D. Graussier and J-D. Nicoud (Eds.), *From perception to action, conference proceedings* (pp. 146–157). Los Alamitos, California: IEEE Computer Society Press.

Peshkin, L., and Savova, V. (2002). On the biological plausibility of reinforcement learning by policy search. *Sixth International Conference on Cognitive and Neural Systems*. Boston, Massachusetts: Boston University Press.

Reil, T. (1999). Dynamics of gene expression in an artificial gene — implications for biological and artificial ontogeny. In D. Floreano, F. Mondada, and J-D. Nicoud (Eds.), *Fifth European Conference on Artificial Life*. Berlin: Springer Verlag.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximate dynamic programming. In B. Porter and R. Mooney (Eds.), *Machine Learning: Proceedings of the Seventh International Conference* (pp. 216–224). San Mateo, California: Morgan Kaufmann.

Vaario, J., Ogata, N., and Shimohara, K. (1997). Synthesis of environment directed and genetic growth. *Artificial Life V* (pp. 244–251). Cambridge, Massachusetts: The MIT Press.

Watkins, C. (1989). Learning from delayed rewards. Ph.D. Thesis, King's College, Cambridge, United Kingdom.