Spectrum-Diverse Neuroevolution With Unified Neural Models

Danilo Vasconcellos Vargas and Junichi Murata, *Member, IEEE*

***Abstract*—Learning algorithms are being increasingly adopted in various applications. However, further expansion will require methods that work more automatically. To enable this level of automation, a more powerful solution representation is needed. However, by increasing the representation complexity, a second problem arises. The search space becomes huge, and therefore, an associated scalable and efficient searching algorithm is also required. To solve both the problems, first a powerful representation is proposed that unifies most of the neural networks features from the literature into one representation. Second, a new diversity preserving method called spectrum diversity is created based on the new concept of chromosome spectrum that creates a spectrum out of the characteristics and frequency of alleles in a chromosome. The combination of spectrum diversity with a unified neuron representation enables the algorithm to either surpass or equal NeuroEvolution of Augmenting Topologies on all of the five classes of problems tested. Ablation tests justify the good results, showing the importance of added new features in the unified neuron representation. Part of the success is attributed to the novelty-focused evolution and good scalability with a chromosome size provided by spectrum diversity. Thus, this paper sheds light on a new representation and diversity preserving mechanism that should impact algorithms and applications to come.**

***Index Terms*—General artificial intelligence, neuroevolution, NeuroEvolution of Augmenting Topology (NEAT), reinforcement learning, spectrum diversity, topology and weight evolving artificial neural network (TWEANN), unified neuron model.**

I. Introduction

LEARNING algorithms are the only solution available to problems that are too complex to be solved with hand-coded programs. They are, for example, used to recognize speech in cell phones, aid the trade in stock markets, process and extract knowledge from problems with big data, play games comparatively with humans, and so on. Moreover, they are a natural solution to the problems that are expected to change over time, since they can learn, i.e., adapt to changes.

The impact of learning algorithms nowadays is enormous, yet the area has the potential to expand much further. For that, however, it is necessary that learning algorithms can be able to work on new problems without the presence of an expert. Current algorithms behave sometimes erratically on new problems or require a lot of trial and error as well as prior knowledge to work.

This motivates us to create algorithms that can learn under problems never seen before and without any prior knowledge. Moreover, this type of investigation has not only a socioeconomic value but also a scientific one. Learning can be defined as the ability of improving computational models (representations) to solve problems. It is the idea of starting from zero or little prior knowledge and arriving at a reasonably good solution. Naturally, the more prior knowledge is inserted into a learning algorithm, the less its learning capabilities are important. There is still much to be understood about the learning limitations and inherent tradeoffs associated.

To build an algorithm that can face very different problems, it is necessary to have a general representation. The most general representation is perhaps a graph with nodes composed of various types of functions. Provided that brains are also the graphs of neurons, this justifies the bioinspiration in neural networks. The only way to optimize not only connection weights but also topology of neural networks is with neuroevolution, because evolution can optimize any kind of problem including ones, where the model itself grows or decreases in size. Consequently, neuroevolution is chosen as the underlining basis for this paper.

In this paper, a neuroevolution-based learning algorithm capable of learning various classes of problems without any prior knowledge is proposed. Moreover, this new neuroevolution-based learning algorithm tries to unify most of the neuron representations into one. Since every kind of neuron is specifically good for a certain type of task, the natural selection of the neuron type for each problem suits well the objective of solving a wide range of problems without any prior knowledge. Furthermore, a new niching mechanism and a diversity paradigm are created to aid the evolution of this complex representation. The proposed method is called spectrum-diverse unified neuroevolution architecture (SUNA). In summary, SUNA has the following features.

1) *Unified Neuron Representation:* A new neuron representation is proposed that unifies most of the proposed neuron variations into one, giving the method a greater power of representation.
2) *Novelty-Based Diversity Preserving Mechanism:* Solutions are not only different, but the frequency and the types of neurons and connections used may suggest a completely different approach to solve the problem. For example, the use of many neuromodulated connections

Manuscript received September 15, 2015; accepted March 28, 2016. Date of publication April 27, 2016; date of current version July 18, 2017.

D. V. Vargas is with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan (e-mail: danilo.v.vargas@gmail.com).

J. Murata is with the Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan (e-mail: murata@cig.ees.kyushuu.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2016.2551748

against normal connections in a network is a different approach. Such diversity in the approaches used should be preserved, and therefore, a new novelty-based diversity is proposed, which keeps track of previous solutions and preserves the different approaches spotted.

3) *Meaningful Diversity for Large Chromosomes:* Instead of comparing chromosomes, this diversity measure compares the values of a set of phenotype-based or genotype-based metrics. In other words, spectra[1] of features are compared. By comparing spectra, the diversity is still meaningful for large chromosomes[2] as well as it is problem independent.
4) *The State-of-the-Art Results:* In all five classes of problems tested, the proposed algorithm had either better or similar results when compared with NeuroEvolution of Augmenting Topology (NEAT).
5) *Few Parameters:* SUNA has only 8 parameters, while NEAT has 33 parameters.

II. Neuroevolution

Neuroevolution is an area of research resulting from the combination of the representation power of artificial neural networks [1], [2] with the optimization capabilities of evolutionary methods. For example, instead of using the usual gradient descent methods, such as backpropagation, an evolutionary algorithm is used to train the neural network.

Many works investigated the evolution of fixed-topology neural networks [3]–[6]. However, it is hard to specify a good representation in advance. First, the complexity of the representation should match the complexity of the problem; therefore, a completely automated algorithm is impossible. Second, although it is possible to represent any function with a single hidden layer, it does not mean that any function is equally easy to represent with a single hidden layer. With the increase of hidden neurons, the search space becomes increasingly huge. That is, the optimization algorithm has to deal not only with plateaus and difficult fitness landscapes, but also with an additional difficulty in the representation, because many parameters are present as well as competing conventions (different neural networks that compute the same and exact function).

Deep neural networks (DNNs) solve the second mentioned problem, i.e., by using many layers, it is possible to learn the representations in higher levels of abstractions [7], [8]. This justifies why this type of network sets records in image recognition [9], [10] and speech recognition [11], [12] and has good results in many other types of applications. Still, in order for DNNs to work, specialists are needed to set the network parameters and design its topology.

Recently, motivated by the limitations of fixed-topology approaches, there is an increasing interest in algorithms that evolve both the topology and the weights. To these algorithms, it is given the name of topology and weight evolving artificial neural network (TWEANN). Some of the most famous direct encoding TWEANNs are cellular encoding [13], GeNeralized Acquisition of Recurrent Links [14], NEAT [15], Evolutionary Acquisition of Neural Topologies [16], and EPNet [17]. Naturally, some of them also have indirect encoding versions, while others can be extended to indirect encoding without problems. For a detailed review of TWEANNs, please refer to [18].

One of the most successful direct encoding TWEANNs is the NEAT algorithm, which has many notable features that made it so popular. We will explain these notable features and briefly describe how NEAT works in Section III.

III. NEAT

One of the central ideas of NEAT is the innovation number, which works as a gene identification and tracking number. Innovation number is an additional value attached to a gene. It enables the recognition of the same gene in different individuals, and therefore, it is possible to match the same genes while doing crossover. Moreover, it is possible to avoid the problems associated with competing conventions during crossover.

NEAT uses a variable-size genetic encoding, where each individual has a list of node genes and a list of connection genes.

1) Node genes specify if each of their associated nodes is an input, output, or hidden node as well as state its innovation number.
2) Connection genes determine the connections between two nodes by specifying the in-node, out-node, and weight of the connection. Moreover, there is an associated enable bit (whether or not this gene is expressed) and an innovation number.

There are four types of genetic operators in NEAT: two types of structural mutations, a simple weight mutation and crossover. Structural mutations add complexity to the network by either including a new node in the middle of an existing connection or by adding a new connection between existing nodes (see Fig. 1). In addition, there is the usual weight mutation present in many other methods, where some individuals have the weight value of their connections either perturbed or set to a new random value. Crossover is done by lining up the innovation numbers from both parents in the chronological order. Matching genes are inherited, while unmatching ones are either inherited randomly when parents are equally fit or inherited from the fittest parent when one parent is fitter than the other. This type of crossover avoids an expensive structural analysis of similarity while still preserving some similar structure present.

Different structures take different times to improve through evolution, e.g., complex structures take longer to improve than simpler ones. Therefore, protecting different structures from unfair competition is necessary if one wants to have an unbiased evolution. NEAT solves this by introducing speciation. Each generation, individuals are separated into species by calculating a distance based on their matching and unmatching genes (i.e., the same and different innovation numbers).

[1]The word spectrum here is used to denote a set of values of histograms or metrics, defining the overall characteristics of the chromosome in a compressed manner.

[2]High-dimensional spaces are exponentially more sparse than low-dimensional ones. That is the reason why distance metrics for long chromosomes are less meaningful. To avoid this complication, here the dimensions are converted into a small number of features, which compose the spectrum of the chromosome.

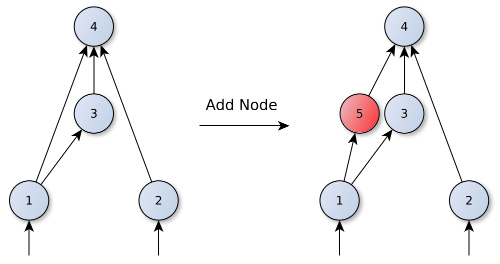
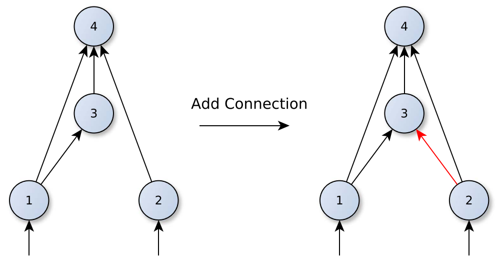

Fig. 1. Structural mutations in NEAT.

Individuals that have their distance below a given threshold are from the same species. An explicit fitness sharing [19] is used to make it harder for any species to grow and dominate the entire population. Thus, species have a determined number of offspring slots proportional to the average fitness of its individuals, while individuals compete with other individuals within the same species for the allocated offspring slots. The entire population is replaced by its offspring.

NEAT starts the search from a relatively small network, where all inputs are connected to all outputs without hidden nodes. Notice that the initial population diversity resides in the connection weights, which are randomized rather than in the structure. This biases the evolution toward minimal solutions, while the genetic operators complexify the structure incrementally by adding more nodes and connections throughout the evolution process.

IV. Neural Networks' Features: A Unifying View

The number of types of neural networks is extremely vast. Although they are called neural networks, they differ in so many aspects such as learning system, architecture,[3] and model that it is difficult to see them as a unity. Moreover, the features of the neural network used usually reflect the problem it is facing, and therefore, a unified neural network model is also required to automatically find the solution, independently of problem type.

Motivated by the unification benefits, this section will focus on describing many model features in a unified way. Setting activation functions aside allows us to divide the model features of a neural network into four broad aspects:

1) different neuronal time scales;
2) inhibition and excitation of neurons;
3) synaptic plasticity;
4) feedback.

[3]Architecture is defined here as the features or dynamics of blocks of neurons, such as bottlenecks in the network, connection probabilities, and topology dynamics.

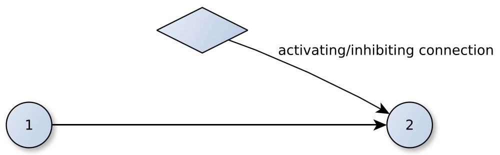

Fig. 2. Neuromodulation of neurons is the activation/inhibition of neurons by another neuron. In this figure, the neuromodulatory neuron (represented by a diamond) is activating or inhibiting neuron 2.

A. Different Neuronal Time Scales

Most neural networks use only one fixed fast time constant, which makes all neurons vary its output as fast as possible in relation to their input. This is important feature in executing tasks, which need fast reaction time. However, a slower neuron can take into account a series of past inputs enabling it to decide over the long-term events and even store information for other neurons. The importance of time scales has been shown in many articles, for example, in self-localization of a neuroevolved robot in a corridor (the sensory-motor patterns in different time scales are used) [20], in reinforcement learning with multimodels and subgoals [21], with the long short-term memory recurrent neural network (RNN), which use memory cells capable of storing/accumulating signals for a very long time (much more than usual RNNs) until a forget signal is received [22]. Different neuronal time scales even allow hierarchical control to emerge naturally when a bottleneck is introduced in the neuron topology, creating a layer with long-term task-related actions and another layer with short-term reactionary actions [23].

In fact, biological neurons have a membrane time constant that vary widely from neuron to neuron (the time constant depends on type, density, and regulation of the present ion channels) [24]. Therefore, different time scales are present in biological neurons and experiments even show how this temporal hierarchy of neurons relates with the anatomical hierarchy [25]. In other words, the temporal hierarchy is an essential part of the brain.

B. Inhibition and Excitation of Neurons

Inhibition and excitation of neurons (the term neuromodulation of neurons will also be used) is the ability of some neurons to inhibit (deactivate) or excite (activate) other neurons (see Fig. 2). It is widely known that the biological neural networks are heavily based on inhibition/excitation principles with the groups of neurons influencing how other neurons will behave and if they are going or not going to activate [26], [27].[4] Although not explicitly stated, the dynamics of inhibition and excitation are also present in many machine learning methods, such as decision trees (DTs). If we consider nodes in a DT to be like neurons (in neural trees, the DT's nodes are exactly the same as neurons [28]–[30]), instead of processing information through some function and passing to the next connected neurons, they process information and choose one of the connected neurons to activate.

[4]In biology, the inhibition and excitation of neurons by other neurons is also called neuromodulation, but the term is rarely used in this sense in the computational field of neuroevolution.

Self-organizing map (SOM) is another example that uses excitation and inhibition. SOM makes neurons compete for the input, where only one will win and activate [31]. Both SOM and DT usually use a crisp division, but a fuzzy division can easily be accomplished with fuzzy memberships in a net of excitation/inhibition neurons.

From an information processing perspective, the importance of using inhibition and excitation lies in using the divide-and-conquer approach to reduce complexity. It is also possible to create network modules with inhibition and excitation as well as many other complex dynamics that are difficult to be created with other models. Moreover, inhibition/excitation is a completely different model from the usual neuron model used by most neural networks. Therefore, it may allow dynamics that are complex to build with the usual neuron model to be easily built.

C. Synaptic Plasticity

Synaptic plasticity has been motivated largely by its biological foundations, but few is known about its useful information processing properties. There are homosynaptic and heterosynaptic mechanisms [32].

1) *Homosynaptic:* Weight update is only a function of presynaptic and postsynaptic neurons (e.g., Hebbian learning).
2) *Heterosynaptic:* The synapse is neuromodulated, i.e., the same as homosynaptic with the addition of a modulatory neuron that strongly influences the Hebbian plasticity (e.g., modulated Hebbian learning).

In homosynaptic mechanisms, researchers reported that the use of synaptic plasticity aided their methods to adapt to environmental changes [33], but depending on the task, it may not give any improvements [34]. It seems that the change of weights based on the relationship between connected neurons (e.g., Hebbian learning) creates certain complex dynamical systems as a result, but there is no reason why the same dynamical systems could not be created with a given structure and some neurons, i.e., without changing weights online.

Having said that the heterosynaptic mechanisms employ neuromodulation[5] of connections and differ substantially from the homosynaptic mechanisms. The results obtained from using such mechanisms are much more encouraging [35]–[39]. From an information processing point of view, the neuromodulation allows for another neuron to influence the relationship between two connected neurons. Recall that, in the usual model, the relationship between the neurons defined by a connection is never dependent on another neuron. Therefore, the heterosynaptic mechanism expands arguably the representation capabilities of the computation model. This paper will use only heterosynaptic mechanisms. Fig. 3 shows a simple example.

D. Feedback

Feedback is one of the ways that a network has to keep information from previous runs (e.g., one way of codifying memory). This is the reason why neural networks with feedback can solve non-Markov problems. Most importantly, it is the only way that neurons in a processing sequence may have access to outputs from subsequent neurons (see Fig. 4 for the examples of network structures with feedback). This property is extensively used in control systems, electronic engineering, biology, social sciences, and many other areas.

[5]Neuromodulation, in artificial neural networks, is the modulation of connection weight(s) by another neuron.

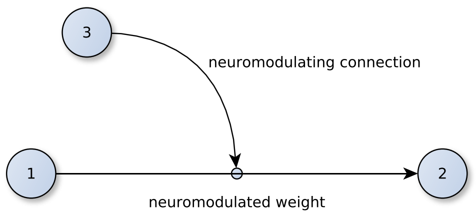

Fig. 3. Given two neurons 1 and 2, the modification of their connection by another neuron is called neuromodulation of connections. To avoid confusions with other inhibition/activation types of neuromodulation, the diagram used here differs from the usual ones found in neuroevolution papers.

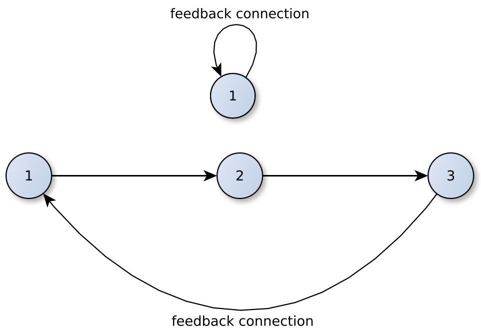

Fig. 4. Two examples of neuron feedback.

V. Unified Neural Model

In this section, the proposed unified neural model will be described.

A. Extended Neuron

Neural models are built from interconnections and features of simple units, the neurons. The neuron used here is an extended version of the widely used neuron model employed by the perceptron, multilayer perceptron, and most other neural network algorithms (referred to as usual neuron model in this paper). This extension is done in two ways.

1) *Adaptation Speed:* Every neuron has its own reacting speed, that is, some neurons may take longer to change their outputs.
2) *Type:* Neurons have various activation functions.

Fig. 5 shows a diagram of the behavior of a single neuron. Mathematically speaking, it can be determined by the following equations:

$$a = f\left(\sum_{i=1}^{n}(w_i x_i)\right) \tag{1}$$

$$\begin{aligned} \mathrm{Ins}_t &= \mathrm{Ins}_{t-1} + \frac{1}{adaptationSpeed}(a - \mathrm{Ins}_{t-1}) \\ y &= \mathrm{Ins}_t. \end{aligned} \tag{2}$$

Equation (1) is basically the same as the usual neuron model, where every input x_i is multiplied by its associated weight w_i

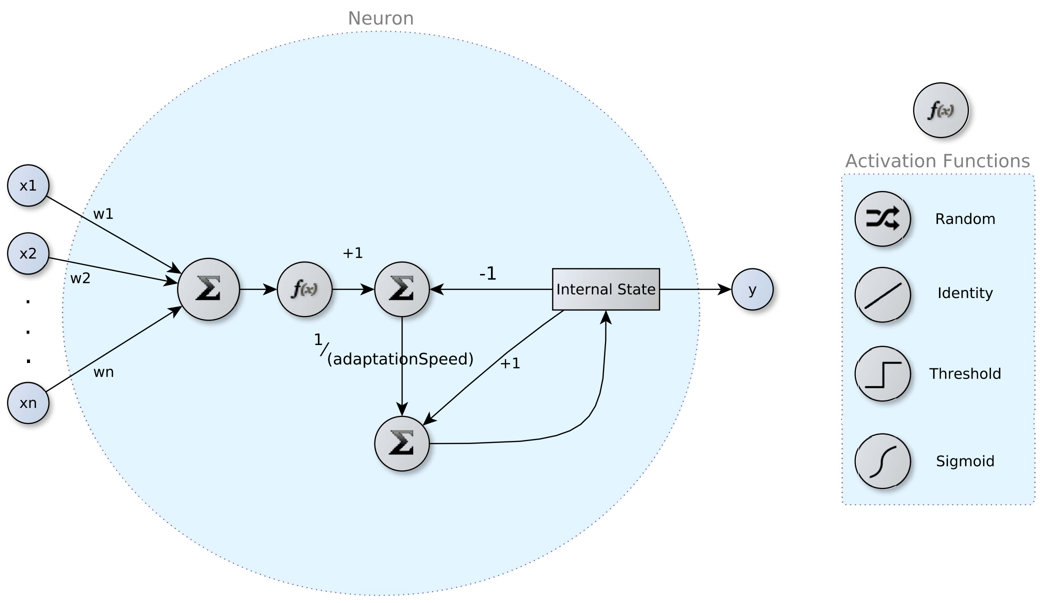

Fig. 5. Neuron activation process and the possible activation functions.

for n inputs. The result a from applying the activation function $f()$ is, however, not used to update the output y directly. It is used instead in a delta rule like equation [see (2)] that takes the neuron adaptation speed $adaptationSpeed$ as parameter to update its previous internal state Ins_{t-1}. The reasoning behind this equation is that if the neuron input changes, the neuron output would change completely after $adaptationSpeed$ iterations, i.e., creating a slower neuron. Finally, the current internal state Ins_t is used as output.

The reason behind these modifications goes beyond bioinspiration. Different adaptation speeds result in a multiscale network that is capable of taking long-term changes into account. For example, finding an approximation to the average of a variable is as simple as connecting a slower neuron to it. Moreover, the use of a single type of neuron (only one activation function) is an unnecessary bias. The use of some types of neurons allows for the neural network to fit different problems without increasing the searching space too much.

B. Control Neuron and Control Signals

One of the motivations behind TWEANNs is to evolve problem-related structures automatically. In fact, sometimes, a simple structure that switches between different behaviors (set of neurons) is enough to solve a task [40]. However, there is no way to turn OFF a set of neurons in current TWEANNs, so a question is raised. How can structures such as a basic switch system come to life?

We propose to solve this problem by adding a different signal type called control signal. Control signals will be solely responsible for activating and deactivating neurons, i.e., they will not interfere with their inputs. Therefore, (3) is added to decide whether the extended neuron activates or not

$$\text{stimulation} = \sum_{i=0}^{n}(w_i \text{cs}_i)$$
$$\text{activation} = \begin{cases} \text{true}, & \text{if stimulation} >= \text{threshold} \\ \text{false}, & \text{otherwise} \end{cases} \tag{3}$$

where cs_i is the control signal and w_i its relative weight for n control signals. Therefore, the neuron activates if the stimulation is more or equal to the parameter threshold, because the activation is set to true. Otherwise, the neuron does not activate. Here, the threshold is fixed for all neurons.

To make all these possible, control signals need to be somehow present. This makes necessary the introduction of a special neuron called control neuron. The control neuron is exactly like all other neurons aside from the fact that it does not output normal signals but control signals. All control neurons have threshold activation functions.

C. Genome

To evolve neural networks, it is necessary to define a genome. The genome used is a direct encoding scheme made by a list of connections and a list of neurons (see Fig. 6).

The evolvable parameters and their respective range values are shown in Table I, where N is the set of neurons present in the respective chromosome. Notice that the following parameter values are not part of the evolvable parameters although they are present inside the genome: input identity (input neuron with identity activation function), output identity (output neuron with identity activation function) (see Fig. 6), and interface index. This happens because they are only created by the initializing step of the evolution and kept fixed afterward.

D. Order of Neuron Execution

To let the control neurons decide which neurons are going to execute, the control neurons need to be executed first. Afterward, control signals are established and the remaining neurons are executed. In other words, the following order is respected:

1) input neurons;
2) control neurons that do not receive connections from other control neurons;

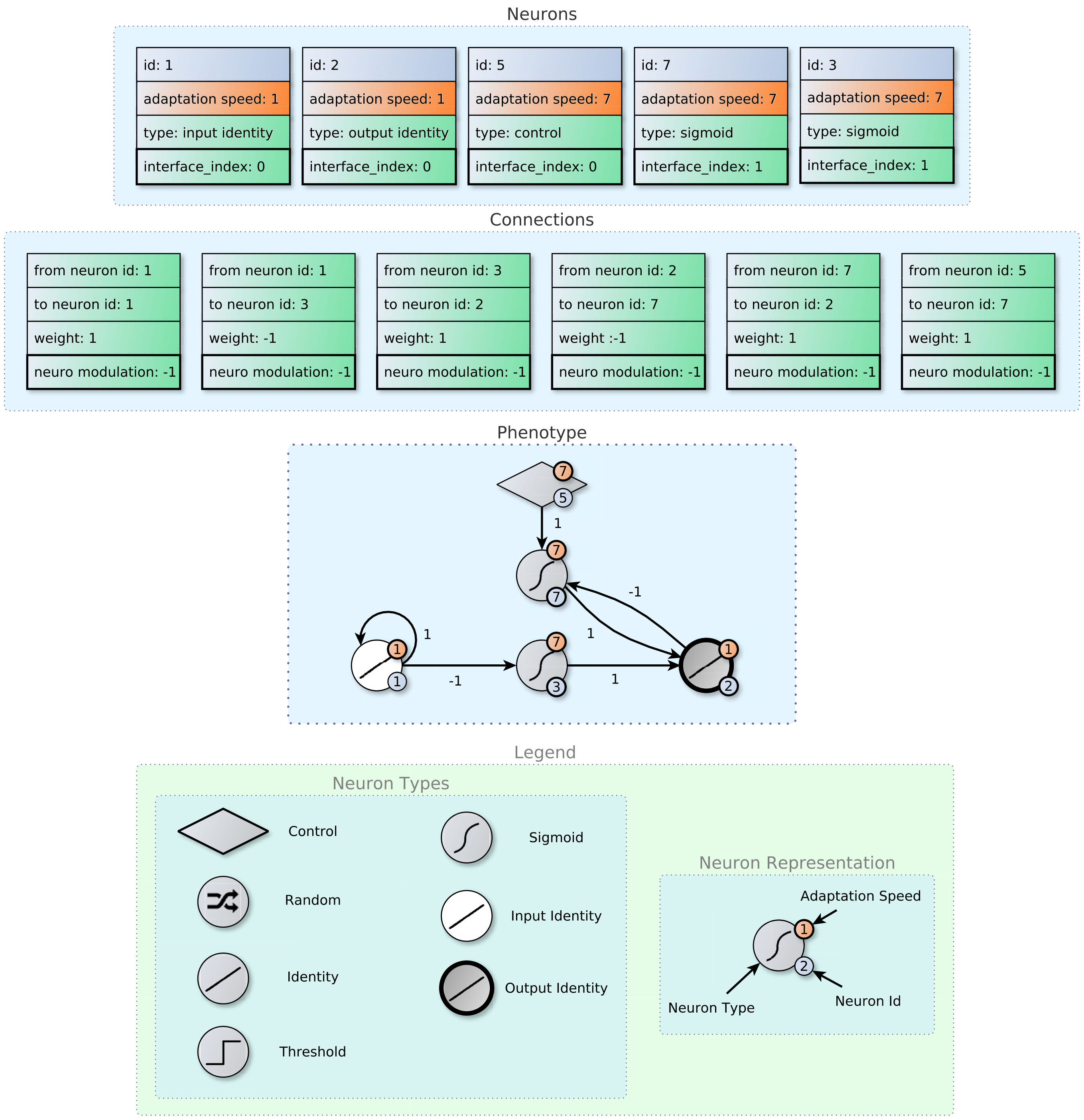

Fig. 6. Example demonstrating how the genome maps its phenotype.

3) remaining control neurons that are activated;
4) remaining neurons that are activated and have an absolute output higher than 0.001.

Input and output are always cleared after the network execution even when the respective neurons were not activated. That is, if a given input or output neuron was not activated, the respective input or output is set to zero.

VI. Evolving the Unified Neural Model

To evolve structures, special evolutionary methods are needed. A new evolutionary method is proposed that is based on a new diversity paradigm. Here, we show how the diversity of structures can be achieved without creating a dependence on the problem.

An overview of the evolution process is shown in Fig. 7. The steps will be described in detail in Sections VI-A–VI-F.

A. Initialization

First, all chromosomes in the population are initialized to be a set of input and output neurons, one for each input/output present in the problem. For these neurons, the interface index is set to be the index of the respective input/output. To create

TABLE I
EVOLVABLE PARAMETER SPACE

Related to	Parameter	Range
Neuron	$adaptationSpeed$	$\{1, 7, 49\}$
	Type	$\{control, random, sigmoid, threshold, identity\}$
Weight	From Neuron	$\{N\}$
	To Neuron	$\{N\}$
	Weight	unbounded
	Neuromodulation	$\{-1, N\}$

an initial diversity, I_m initial mutations (see Section VI-F) without weight perturbations are applied for all chromosomes.

B. Evaluation

In the evaluation stage, every chromosome present in the population is activated for a whole trial. At the end of each trial, the average reward received throughout is used as fitness. This process is repeated until the entire population has been activated. Afterward, the spectrum of every chromosome is calculated.

C. Spectrum Diversity

Diversity and protection of innovation is a challenging problem for TWEANNs, basically because finding a metric to compare the structures of TWEANNs is difficult. Speciation tries to approximate this comparison by using innovation numbers (a certain type of id) on the genes, and therefore, the alignment of such genes gives a fast approximation of structure similarities [15]. However, two structures originated from different evolutionary paths are always seen as totally different and unmatching structures, because they differ in innovation numbers. This happens even if they are the same or similar in structure. And since genes are compared one per one, it does not scale well with the size of chromosome.

In physics and chemistry, a similar problem takes place. Atoms combine with each other into complex structures, but it is discovering these structures, which is a hurdle. To solve these problems, various types of spectra are used (e.g., nuclear magnetic resonance spectrum).

Similarly, here we propose to use spectra to identify and compare structures. The spectrum is itself designed for the representation though composed of a set of genotype's and phenotype's properties. Computationally speaking, the spectrum is an array with each property being an element of this array. Here, the spectrum is composed of the following properties:

1) the number of identity neurons;
2) the number of sigmoid neurons;
3) the number of threshold neurons;
4) the number of random neurons;
5) the number of control neurons;
6) the number of slower neurons (adaptation speed greater than one).

This combination of properties divides the set of solutions by how they approach to solve the problem.

Once the spectrum of all chromosomes are calculated, their spectra are fed into the novelty map [41] (Section VI-D), where only the spectra that are mapped into the same cell creates a group (species) and compete. The entire process of calculating the spectra and insertion in the novelty map is called spectrum diversity, because the spectrum of chromosomes is used to separate the population into groups(species), i.e., diversity is kept by creating species based on their spectrum similarity.

Spectrum diversity uses a measure of diversity based on the distance between the spectra of two structures. The advantages of using this type of diversity are as follows.

1) *Problem Independence:* The metric is not defined based on problem characteristics, and therefore, it works independent of problem type and size.
2) *Scalability:* Since the spectrum should describe the properties of the structure as a whole, the size of the chromosome does not interfere with the metric.

Notice that the measures from complex networks and graph theory may also be used, joining the discipline of complex networks with TWEANNs. Actually, by using metrics from complex networks, it is possible to achieve many types of structure diversity, for example, degree distribution diversity, assortativity diversity, and so on.

D. Novelty Map

Novelty map is a table with the most novel inputs according to a given novelty metric (see Fig. 8). When an input is presented to the map, competition takes place where the cell with the closest weight array wins. This winner cell is activated and can be used in many ways. Here, the winner cell is used to identify to which species (group) the input pertains. Afterward, the table is updated by substituting the weight array of the least novel cell (according to the novelty measure) with the input array if and only if the input array has higher novelty.

This table shares some similarities with SOM [42] and neural gas [43], but its behavior is independent of input frequency and it uses fewer cells to map the same input space (cell's efficiency).

The novelty metric used in this paper is the uniqueness. Let S be a set of arrays. Uniqueness is defined for an array a_i in relation to the other arrays in S by

$$U = S \setminus \{a_i\} \tag{4}$$

$$\text{uniqueness} = \min_{a_k \in U}(\text{dist}(a_i, a_k)). \tag{5}$$

1) Novelty Map Population: Novelty map population is a novelty map, where cells are subpopulations and inputs are some value related to the individual. Here, the spectrum of chromosomes is used as input.

E. Selection

In the selection stage, only one individual survives inside every cell (species) of the novelty map. To decide which

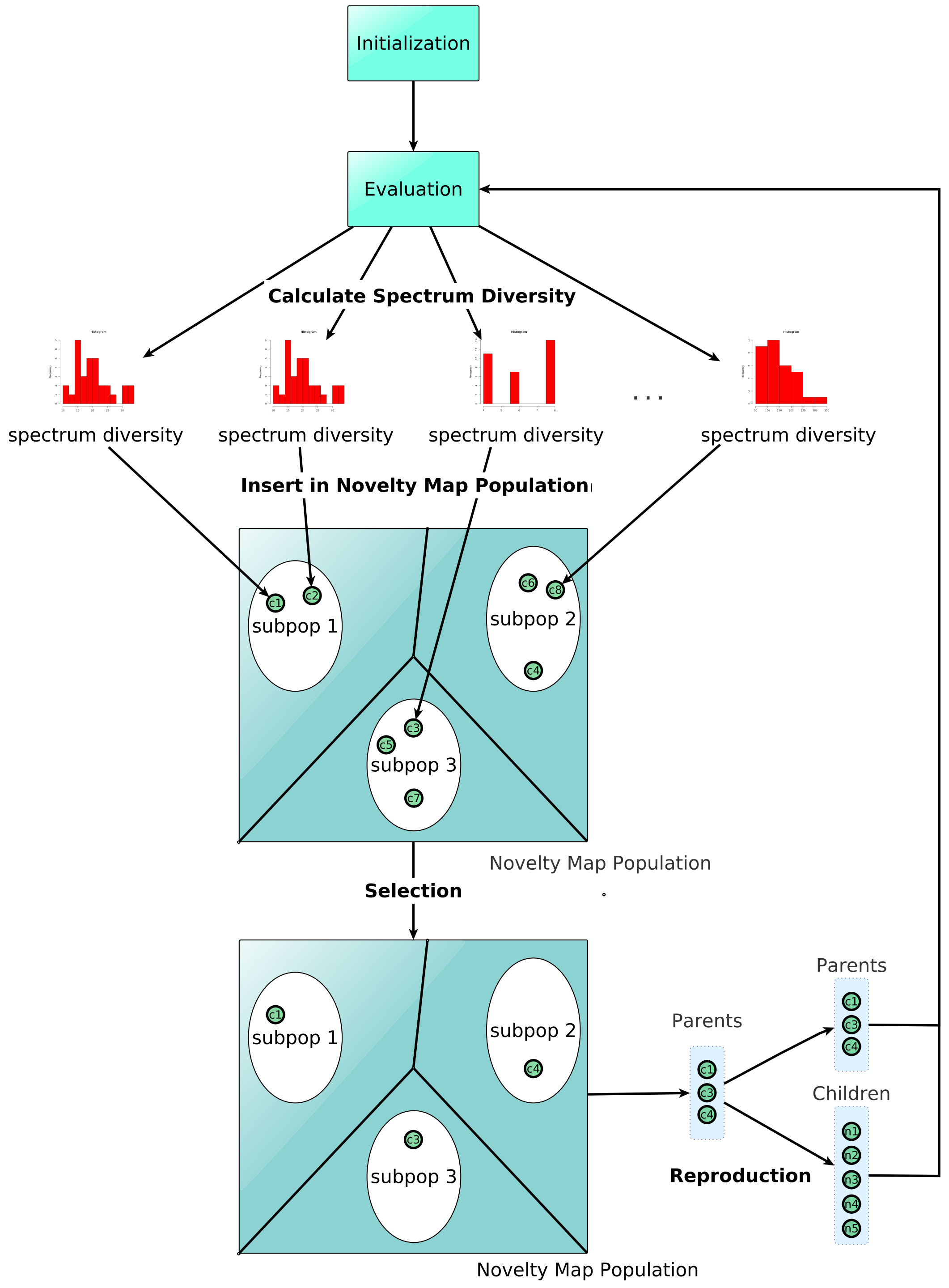

Fig. 7. Evolution's overview.

individual is going to survive, the individuals are compared based on their fitness. In case of a draw, the individual that uses less neurons wins (i.e., the least complex wins).

In this sense, subpopulations are the same as niches, because chromosomes that fall inside the same subpopulation have a similar histogram and, therefore, are similar themselves. Notice that there is no competition between niches. There is also no preference or advantage to niches that have better average fitness, such as with fitness sharing procedures. Different approaches will form niches (subpopulations) and survive even if they have poor fitness. As shown in [44], only by avoiding any kind of competition (direct or indirect) between niches (subpopulations), it is possible to avoid deleterious competition. Giving the advantages to niches that are already strong will further decrease the chances of different approaches to surge. Moreover, niches that are already good

```
Require: $|P| > Max_n > 0$ and $Nmetric() \neq \emptyset$
1: $NM \leftarrow \emptyset$ {All cells are empty}
2: loop
3:   if An input $I$ is presented to the Novelty Map then
4:     if $|NM| < Max_n$ then
5:       $NM \leftarrow NM \cup \{I\}$
6:     else
7:       $\{A \in NM \mid \forall K \in NM \wedge Nmetric(A) < Nmetric(K)\}$
8:       if $Nmetric(I) > Nmetric(A)$ then
9:         $NM \leftarrow NM \cup \{I\}$
10:        $NM \leftarrow NM \setminus \{A\}$
11:      end if
12:    end if
13:    return $H : \{H \in NM \mid \forall K \in NM \wedge dist(I,H) < dist(I,K)\}$ {Return cell which is closest to input $I$}
14:  end if
15: end loop
```

Fig. 8. Novelty map algorithm. P is the set of individuals composing the population, $Nmetric()$ is a given novelty metric, NM is the set of cells in the novelty map, Max_n is the maximum size of NM, and $dist()$ is the Euclidean distance.

will progress independently of having advantages over other niches.

After selection, the individuals that survived become parents and the algorithm enters in the reproduction stage. As a matter of fact, to keep the number of parents constant, if a subpopulation is empty, a random valid parent from a different subpopulation is chosen and copied instead.

F. Reproduction

With the parents defined, the remaining vacancies in the population are filled by the following sequence of procedures.

1) A parent is chosen at random.
2) S_m step mutations are applied to the parent to create a child.
3) All connections have a 50% chance of being perturbed with a random value in the range $[-w, w]$, where w is the current weight of the respective connection.

A mutation can be any of the following procedures.

1) Add a neuron. Everything is randomly chosen, but first the neuron has an additional probability of being a control neuron. Furthermore, two new connections are created connecting this neuron to the network (one random connection from this neuron and another random one to this neuron).
2) Delete a neuron (all the connections to the respective neuron are also deleted). Input/output neurons cannot be deleted.
3) Add a connection. The connection has a probability of being neuromodulated (neuromodulation probability) and initial weights are either 1 or −1. Everything else is randomly decided.
4) Delete a connection.

Any of the above mutation procedures have a chance of occurring defined by the mutation probability array. The mutation probability array (M_{pa}) is composed of four real numbers relative to the probability of, respectively, adding a neuron, delete a neuron, add a connection, and delete a connection. So, for example, $\{0.1, 0.1, 0.4, 0.4\}$ would mean 10% chance of adding a neuron, 10% chance of deleting a neuron, 40% chance of adding a connection, and 40% chance of deleting a connection.

VII. Experiments

In this section, both SUNA and NEAT will be tested on five different classes of problems, each one requiring different learning features to solve. The set of parameters are kept the same for all the problems for both algorithms (see Section VII-A).

A. Experiments' Settings

The NEAT code used is the 1.2.1 version of NEAT C++ software package [45]. Table II shows the NEAT parameters. Actually, the used parameters are the same as the one provided with the package for solving double pole (DP) balancing task. A couple of variations of it were tested, but the original one performed better. Moreover, some modifications were necessary to make NEAT work on problems with negative fitness, since it does not work out of the box. Therefore, a value big enough (2000) was always added to the final accumulated fitness, transforming the negative fitness into a positive one. In addition, to satisfy NEAT's requirements, for all problems, the input range was converted into $(-1, 1)$, while the output range was transformed into $(0, 1)$. This conversion naturally informs the algorithm about the maximum/minimum of both input and output, simplifying the problems at hand. Such changes are not necessary for the proposed method.

Regarding the proposed method, a couple of settings were tested to arrive in the parameters described in Table III. In addition to the tests compared with NEAT, ablation tests are also conducted to evaluate the importance of most of its features as well as understand when they are important. These tests are conducted over harder versions of the same problems (problems without transformation of input/output). In other words, for these tests, the algorithm does not know the range of input/output, verifying further its learning capabilities.

All the results are averaged over 30 runs[6] and only the best result among 100 trials[7] is plotted. Every run, when not specified otherwise, has a maximum of 2×10^5 trials.

B. Mountain Car

The mountain car problem [46] is defined by the following equations (Fig. 9 shows an illustration of the problem):

$$\begin{aligned} \text{pos} &\in (-1.2, 0.6) \\ v &\in (-0.07, 0.07) \\ a &\in (-1, 1) \\ v_{t+1} &= v_t + (a_t) * 0.001 + \cos(3 * \text{pos}_t) * (-0.0025) \\ \text{pos}_{t+1} &= \text{pos}_t + v_{t+1} \end{aligned} \tag{6}$$

[6]Run is defined as a sequence of trials until the maximum number of trials is reached.

[7]Trial is a set of iterations between agent and environment until a problem defined stopping criteria is met.

TABLE II
PARAMETERS FOR NEAT

Parameter	Value	Parameter	Value
trait_param_mut_prob	0.5	trait_mutation_power	1.0
linktrait_mut_sig	1.0	nodetrait_mut_sig	0.5
weigh_mut_power	2.5	recur_prob	0.00
disjoint_coeff	1.0	excess_coeff	1.0
mutdiff_coeff	0.4	compat_thresh	3.0
age_significance	1.0	survival_thresh	0.20
mutate_only_prob	0.25	mutate_random_trait_prob	0.1
mutate_link_trait_prob	0.1	mutate_node_trait_prob	0.1
mutate_link_weights_prob	0.9	mutate_toggle_enable_prob	0.00
mutate_gene_reenable_prob	0.000	mutate_add_node_prob	0.03
mutate_add_link_prob	0.05	interspecies_mate_rate	0.001
mate_multipoint_prob	0.6	mate_multipoint_avg_prob	0.4
mate_singlepoint_prob	0.0	mate_only_prob	0.2
recur_only_prob	0.0	pop_size	100
dropoff_age	15	newlink_tries	20
print_every	5	babies_stolen	0
num_runs	1		

TABLE III
PARAMETERS FOR SUNA

Parameter	Value
Number of initial mutations (I_m)	200
Number of step mutations (S_m)	5
Population size ($\|P\|$)	100
Maximum Novelty Map population (Max_n)	20
Mutation probability array (M_{pa})	$\{0.01, 0.01, 0.49, 0.49\}$
Neuromodulation probability	0.1
Control neuron probability	0.2
Excitation threshold	0.0

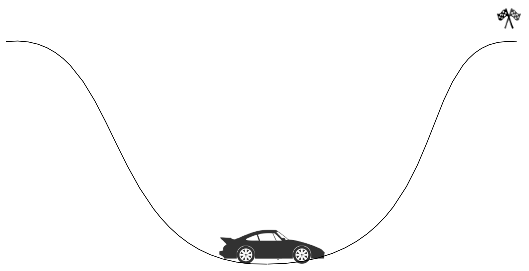

Fig. 9. Mountain car problem. The car's objective is to reach the flags uphill, although its acceleration is not enough to climb the mountain.

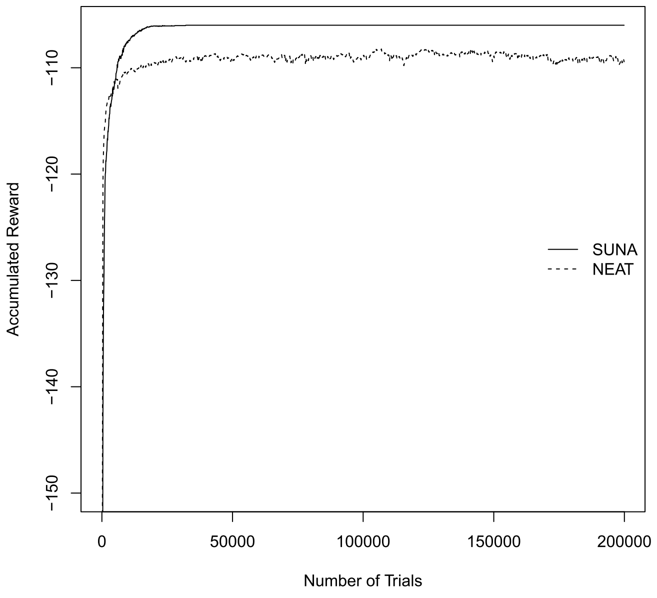

Fig. 10. Comparison of SUNA and NEAT in the mountain car problem.

where pos is the position of the car, a is the car's action, and v is the velocity of the car. The starting velocity and the position are 0.0 and -0.5, respectively. If $v < 0$ and pos ≤ -1.2, the velocity is set to zero. When the car reaches pos ≥ 0.6, the trial is terminated and the algorithm receives 0 as reward. In all other positions, the algorithm receives -1 as a reward. Moreover, if the algorithm's steps exceed 10^3, the trial is terminated and the common reward of -1 is returned to the algorithm.

NEAT can only deal with certain ranges of input/output; therefore, the input was converted into the $(-1, 1)$ range, while the output was converted into the $(0, 1)$ range. Naturally, this conversion informs the algorithm about the maximum/minimum of both input and output, an information that was not readily available in the original problem; therefore, the problem is to some extent simplified.

Fig. 10 shows a comparison between SUNA and NEAT in the mountain car problem. SUNA performs better than NEAT, converging as fast as NEAT even though SUNA sees a bigger search space. The reason behind the worse performance in NEAT seems to be related to unstable individuals taking over the population from time to time, making the results vary around the best solution but rarely staying in that position. Moreover, procedures that check for stagnation make things worse when NEAT has already reached a good result.

The nonsimplified version of the problem, i.e., with raw input and output is shown in Fig. 11. SUNA performs equally well on all problems, showing that the presence or absence of normalization in the input/output is not an issue. NEAT, on the contrary, struggles with raw input, and when the raw output is present, it performs very poorly. It is understandable, and NEAT has only sigmoid input nodes that squeeze values into the $(-1, 1)$ range and outputs nodes in the $(0, 1)$ range and, therefore, cannot cope with this type of problem. Having said that it is still a limitation, compromising the automatization and limiting the application to some problems.

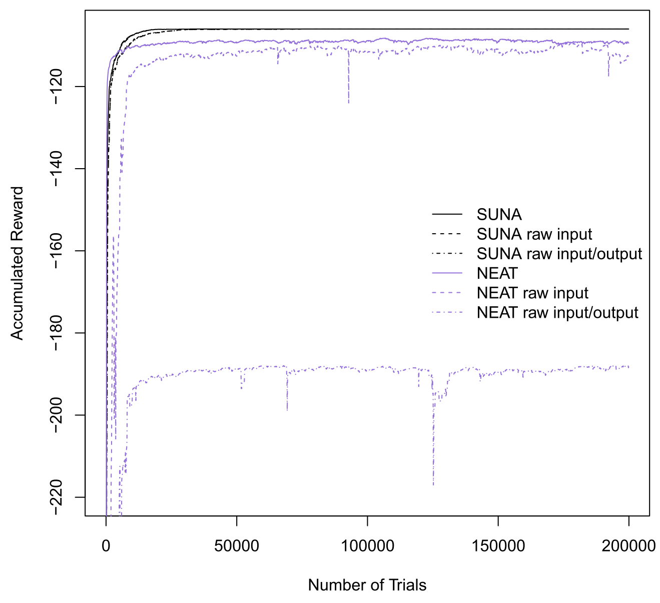

Fig. 11. In real-world problems, the maximum/minimum of either input or output is often not available; therefore, a normalization is not readily possible. Here, the tests show the performance of both SUNA and NEAT when both input and output are normalized (SUNA and NEAT), when the input is raw (SUNA raw input and NEAT raw input) or when both input and output are raw (SUNA raw input/output and NEAT raw input/output). The tests were made in the mountain car problem.

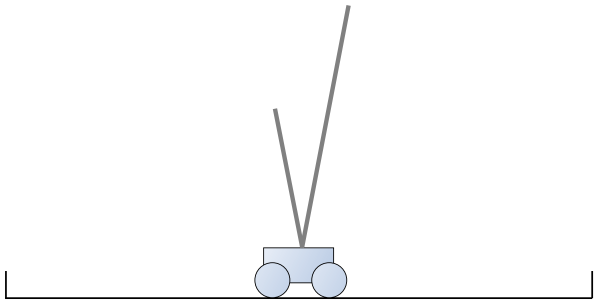

Fig. 12. Illustration of the DP balancing problem.

C. Double Pole Balancing

DP balancing is the famous problem of balancing a pole above a cart by only moving the cart forward or backward (see Fig. 12). This problem has six observable variables. The dynamics used here are the same mentioned by the following papers [4], [15], [47]. In fact, the code was adapted from [45], which is the same code used in other papers [15], [47]. All the original parameters of the problem were also kept the same.

Fig. 13 shows that SUNA and NEAT converge to the same final performance. In other words, both the algorithms learned to balance the pole for 10^5 steps (the maximum allowed) in all the tests. SUNA converges slower than NEAT. This is expected due to the greater complexity of the SUNA's model.

D. Non-Markov Double Pole Balancing

Non-Markov DP (NMDP) balancing has the same dynamics as the DP balancing. However, the agent only observes three variables (position of the cart and angles of both poles) instead of six. To keep the poles balanced, it is necessary to estimate the velocity of the cart and pole angles. Therefore, this problem presents an additional difficulty. The previous input needs to be somehow kept and used to compose an estimate of velocity.

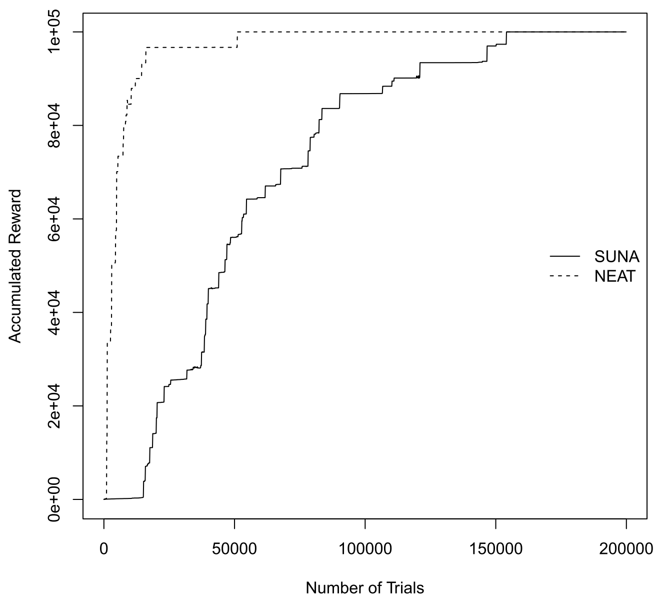

Fig. 13. DP balancing.

Notice that in this paper, the DP balancing fitness function is used, i.e., a modified fitness function that penalizes oscillations is not used [15]. Thus, this formulation of the NMDP balancing problem is relatively more difficult.

Comparison results are shown in Fig. 14. SUNA performs much better than NEAT in this problem. In fact, SUNA achieves 70% of the maximum possible average accumulated reward, while NEAT reaches only 37%. But what features present in the proposed algorithm enable it to outperform NEAT? This question is answered in Section VII-G with the ablation tests. In short, there are basically three important features: slow neurons (responsible for computing approximation of derivatives, averages, and so on), real weights (accurate computation of estimates), and random neurons (exploration).

E. Multiplexer

Multiplexer is a binary problem with a single bit output and with the input composed of data and address variables. To compute the correct output, abits address variables are used to select one of the other 2^{abits} data variables. This selected data variable is the correct output. In this manner, the algorithm must first separate the address input from the data input and then use the address information to select the data part of the input. This problem is coded as a reinforcement learning problem, where each correct answer is rewarded with 0 and each incorrect answer is rewarded with -1. Every possible set of inputs is presented during a trial in the random order, not allowing learning algorithms to memorize the sequence of outputs instead of the multiplexer function.

Fig. 15 shows that the SUNA outperforms NEAT in the multiplexer problem with abits $= 3$ (i.e., three address variables and eight data variables). After 2×10^5 trials, an average of -620 ($\approx$70% accuracy) accumulated reward is achieved by SUNA, while NEAT reaches -720 ($\approx$65% accuracy). However, SUNA continues improving its accuracy further,

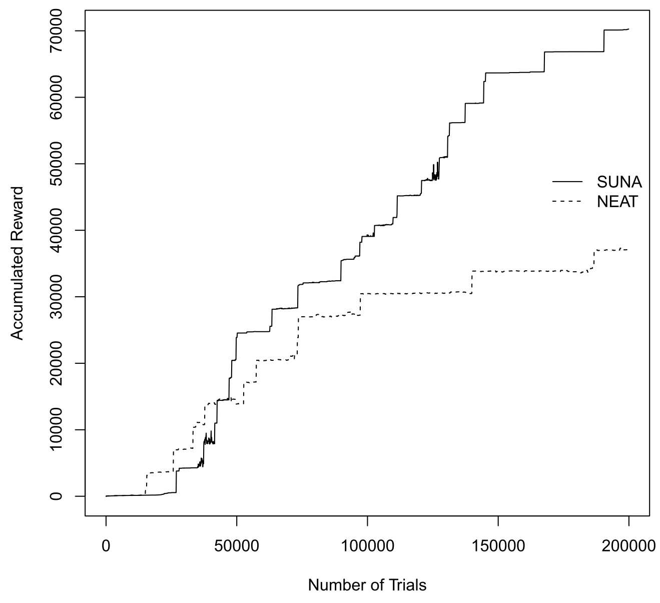

Fig. 14. Performance of both SUNA and NEAT in the NMDP balancing problem.

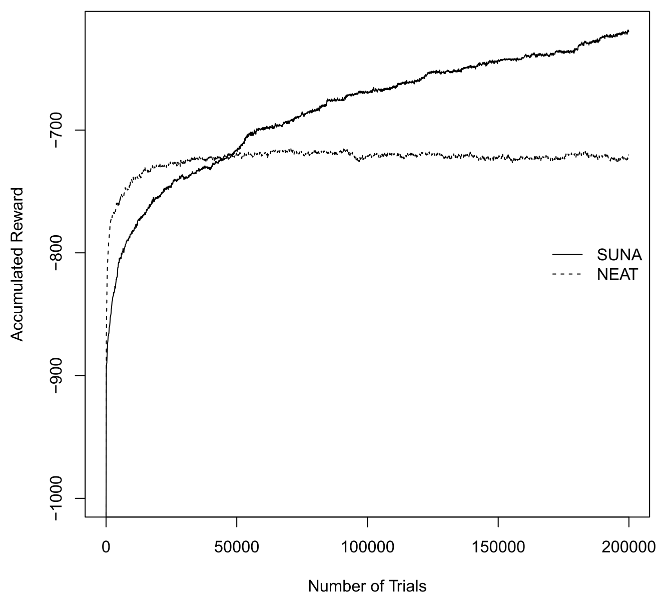

Fig. 15. SUNA outperforms NEAT in the multiplexer problem.

and if more trials are allowed, it would reach much higher accuracy. On the other hand, NEAT rapidly reaches its peak of accuracy without further improvements. Ablation tests done in Section VII-G show that one of the features that allows SUNA to outperform NEAT is neuromodulation. See Section VII-G for details.

F. Function Approximation

In this problem, the algorithm has to learn a sequence given by the following equation:

$$y = \frac{x^3}{1000} + 0.4x + 20\sin\left(\frac{x}{10}\right) + 20\sin(100x) \tag{7}$$

where x and y are, respectively, the input and output of the problem. Moreover, the value of x varies from -100 to 100 in unit time steps. The reward is the negative modulus of the difference between the agent's action (agent's estimation of y) and the correct sequence output y.

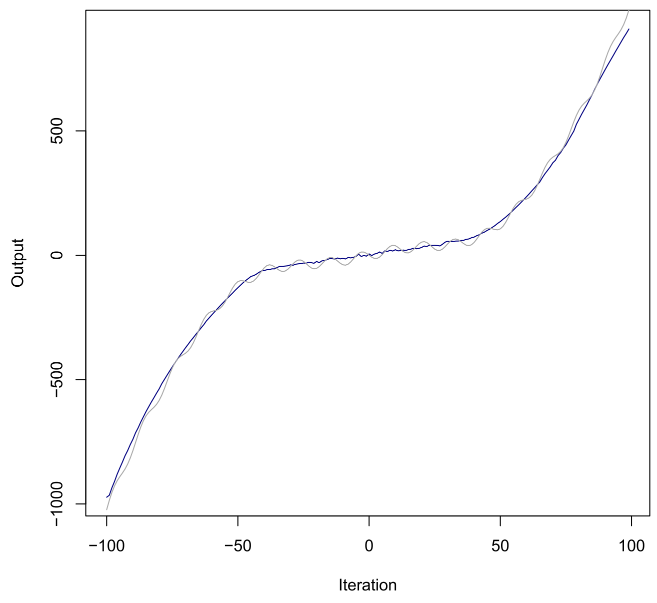

Fig. 16. Results for the FA problem learned by SUNA. Average of the final learned sequence (dark blue line) and the reference sequence (light gray line) is shown.

This problem cannot be solved properly by NEAT, since its input and output need to be the real numbers of unknown range; otherwise, the problem is unreasonably easy. Therefore, here only SUNA's result is plotted (see Fig. 16). SUNA gets a very close approximation to the curve. Section VII-G will expose the importance of slow neurons and linear neurons in these classes of problems.

G. Ablation Tests

In contrast to the previous tests, here the tests do not use any preprocessing system, such as the conversion of input/output to a certain range. Therefore, all input/output conversions necessary for using NEAT were excluded. A natural consequence is that the previous problems get more difficult. To enable enough time for the algorithm to converge in these more difficult problems, runs have a maximum of 10^6 trials instead of 2×10^5 used in the previous tests. These types of tests were preferred to stress SUNA and verify its capabilities.

The ablation tests are composed of nine tests. A control test and eight ablation ones. Table IV shows the percentage in improvement or worsening after the ablation, and Table V shows the statistical significance of the tests. Sections VII-G1–VII-G8 will discuss every ablation in detail.

1) No Linear Neurons: Linear neurons are specially important in the problems with real input/output and input/output with a wide range. The ablation test on function approximation (FA) confirms this affirmation, since the results degrade strongly when linear neurons are removed. In fact, this result is statistically significant.

2) No Control Neurons: As explained before in Section IV-B, with the introduction of control neurons,

TABLE IV

PERCENTAGE OF IMPROVEMENT (POSITIVE) OR WORSENING (NEGATIVE) OF THE ABLATION RESULTS IN RELATION TO THE ORIGINAL ONE. THE PROBLEMS ARE DP, FA, MOUNTAIN CAR (MC), MULTIPLEXER (MUX), AND NMDP

Test Type	DP	FA	MC	MU	NMDP
No control neuron	0%	5.87%	0%	−10.55%	15.76%
No linear neuron	0%	−22.38%	0%	−1.76%	11.18%
No neuromodulation	0%	6.92%	0%	−66.17%	7.88%
No random neuron	−6.28%	7.37%	0%	8.26%	−56.54%
No real weights	−3.29%	9.43%	0%	6.16%	−99.85%
No sigmoid neuron	0%	−3.73%	0%	−3.35%	1.97%
No slow neuron	−6.46%	−32.69%	0%	12.61%	−35.94%
No threshold neuron	−3.13%	6.43%	0%	1.56%	11.77%

TABLE V

STATISTICAL SIGNIFICANCE FOR THE ABLATION TESTS. TWO MANN–WHITNEY STATISTICAL TESTS WERE CONDUCTED FOR EVERY ABLATION TEST IN RELATION TO THE CONTROL ONE. ONE STATISTICAL TEST VERIFIES IF THE RESULTS ARE BETTER THAN THE CONTROL ONE AND THE OTHER STATISTICAL TEST VERIFIES THE CONTRARY. *W* OR *B* MEANS THAT THE ABLATION HAS A FINAL RESULT, WHICH IS, RESPECTIVELY, WORSE OR BETTER THAN THE CONTROL ONE. ONLY RESULTS SIGNIFICANT ENOUGH (p-VALUES BELOW 0.05) ARE CONSIDERED. THE PROBLEMS ARE DP, FA, MOUNTAIN CAR (MC), MULTIPLEXER (MUX), AND NMDP

Test Type	DP	FA	MC	MU	NMDP
No control neuron	()	()	()	W(0.01)	B(0.04)
No linear neuron	()	W $(3.39e-6)$	()	()	()
No neuromodulation	()	B (0.04)	()	W $(2.16e-11)$	()
No random neuron	()	B (0.03)	()	B(0.02)	W $(8.05e-5)$
No real weights	()	B (0.01)	()	()	W $(6.78e-12)$
No sigmoid neuron	()	()	()	()	()
No slow neuron	()	W $(2.87e-6)$	()	B(0.002)	W (0.01)
No threshold neuron	()	()	()	()	()

it is now possible to create systems that switch ON/OFF behaviors. Therefore, the representation capabilities improved strongly, but the tests with the ablation of control neurons show mixed results. Consequently, control neurons do not seem to be used up to their maximum representation potential by the current evolutionary algorithm.

There is a reason for the above. Since certain groups of neurons work together forming functions, representationwise, a higher level of control can be achieved if the groups of neurons are activated or deactivated. However, if the connections do not perfectly connect to all neurons involved in the function, instead of adding a higher level of control, the current level of control is disrupted and control neurons become instead another piece inside the monolithic block of neurons. In other words, control neurons, which are unwisely connected to the network, do not switch ON/OFF behaviors (a set of neurons) but just neurons.

3) No Neuromodulation: The ablation tests show that neuromodulation is of utmost importance (a statistically significant 66% worsening if removed) for the multiplexer problem. Actually, it is easy to see why neuromodulation is important in a problem such as multiplexer that has input selection as its main task, because neuromodulation does exactly that it modulates a node based on another one.

4) No Random Neurons: Random neurons may seem at first glance somewhat weird, summing with the fact that they are of extreme importance for the NMDP problem is yet more perplexing. However, notice that the random neurons are exactly important on problems that require some form of exploration of the environment (e.g., NMDP) while being irrelevant, where no exploration is necessary (function approximation and multiplexer).

It is surprising that even without any kind of hard-coded exploration dynamics, the algorithm develops by itself, at least initially, dynamics that explore the environment.

5) No Real Weights: This time tests verify the importance of using real weights, i.e., these ablation tests use only −1, 1, or neuromodulation (another neuron's output) as possible weight values. The results show that setting the NMDP balancing aside, in all problems, the ablated algorithm's performance is similar to when real weights are used. Actually, the results in the DP balancing problem have improved a bit after the removal of real weights. However, for the NMDP balancing problem, the removal of real weights causes a substantial decrease in performance.

This can be explained by the following reasoning. The absence of real weights is a constraint added to the problem, and when this constraint helps to find a good solution (in the case of DP balancing), it is surely welcome, but sometimes it makes impossible to find good solutions (NMDP balancing). Notice that the only difference between these two problems (NMDP and DP) is the presence/absence of three velocity variables, and therefore, when they are not present, it is necessary to construct them with recurrence and some calculation. The recurrence is present, but it seems that this calculation is never precise enough without real weights, and therefore, the algorithm is never able to solve the non-Markov version.

6) No Sigmoid Neurons: The removal of sigmoid neurons is shown to have no impact on the system performance. All comparisons had a small percentage difference and were not statistically significant. This may be related to the similarity between sigmoid and threshold neurons. Sigmoid neurons have a nonmodifiable steep curve given by a hyperbolic tangent, which has the same output for almost all inputs, aside from the small portions of the input range near zero. Therefore, the system does not suffer from losing one or the other.

7) No Slow Neurons: Slow neurons are important when the information needs to be preserved from the previous iterations. That is why they are very important to NMDP and FA problems. Notice that slow neurons allow for the computation of an average of its input (i.e., the input gets accumulated and is naturally averaged in its output); therefore, they are the good estimates of rates of variance as well as derivatives.

8) No Threshold Neurons: As mentioned before, threshold neurons and sigmoid neurons have similar activation functions. Therefore, the removal of one or the other does not impact performance. This justifies the nonstatistically significant results as well as small percentage difference.

VIII. Laws of Learning

It was shown that each problem class requires some features from the learning algorithm in order to be tackled. However, these features are not specific to each problem class. Many problem classes require similar learning features. Therefore, the question of whether there is a minimal set of features required to solve all problems is raised.

Thus, similarly to the laws of physics which are the minimum set of dynamics that describes the universe, we raise here the following question: "Are there a minimum set of learning features (neuron types, recurrency, and so on) that would allow a learning algorithm to learn any problem?"

We think there is a minimum set. In this paper, we identified some important learning features. But there are certainly many more that need to be identified and unified.

IX. Extensions to NEAT and Extensions to SUNA

NEAT has various extensions, such as FS-NEAT [48], where the initial topology is set to only one input connecting to the output instead of the initial fully connected topology between inputs and outputs, radial basis function (RBF)-NEAT [49], which uses an additional mutation that adds RBF nodes, Cascade-NEAT [50], where only the connections to the last added node are evolved, and SNAP-NEAT [50], which integrates the mutations of three algorithms (NEAT, RBF-NEAT, and Cascade-NEAT) into one.

Regarding indirect encoding, compositional pattern producing networks (CPPNs) are an abstraction of natural development that uses neural networks to create an indirect encoding instead of developmental encodings. Usually, CPPNs are developed with the algorithm called CPPN-NEAT that uses NEAT to evolve it [51]. Moreover, HyperNEAT is an indirect encoding-based algorithm, which uses CPPN-NEAT to create the patterns of weights in substrates [52].

In fact, many of the NEAT extensions can be easily applied to SUNA.

1) Similar to RBF-NEAT, RBF nodes can be included in SUNA.
2) Connections to old nodes can be frozen, having only the connections to recent added nodes evolve (similar to Cascade-NEAT).
3) Additional mutation operators can be used or mixed, integrating other approaches (SNAP-NEAT).
4) CPPNs can be developed using SUNA in the same way that they were developed with NEAT, using a CPPN-SUNA algorithm.
5) HyperSUNA can be developed using either a CPPN-SUNA or by substituting CPPN with SUNA entirely.

X. Conclusion

This paper proposed a new TWEANN that joins most of the neural network features into one unified representation. In fact, a natural consequence of this unified representation is a much more powerful representation capability. This affirmation was verified by the tests, where the proposed algorithm outperformed NEAT on most of the problems. Moreover, ablation tests demonstrated that good results could only be obtained with the presence of the added new features. Therefore, this paper gives the steps forward an algorithm that can learn any problem and we raise the question of if there is a minimal set of features that can let an algorithm learn anything.

Actually, representation alone is not enough. A diversity preserving method called spectrum diversity was also necessary. Spectrum diversity allows the chromosomes that are novel enough to be kept even when their fitness is abysmally poor. Consequently, this allows different approaches to develop without the deleterious competition of faster evolving ones. Moreover, spectrum diversity scales better with the size of chromosome than speciation because of its use of distance metrics based on chromosome spectra instead of the genes themselves. This diversity method together with the chromosome spectrum concept should find uses in many other algorithms, specially the ones that utilize a great variety of gene types in their genotype as well as applications, where the chromosomes are high dimensional.

Thus, this paper sheds light on a new representation as well as a new diversity keeping method and its associated spectrum concept, opening up arguably better possibilities for the creation of algorithms. Moreover, the overall good results motivate further applications of the approach to real-world problems and extensions, such as an indirect encoding version.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1998.

[3] D. Whitley, S. Dominic, R. Das, and C. W. Anderson, "Genetic reinforcement learning for neurocontrol problems," *Mach. Learn.*, vol. 13, no. 2, pp. 259–284, Nov. 1993.

[4] A. P. Wieland, "Evolving neural network controllers for unstable systems," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN-Seattle)*, vol. 2. Jul. 1991, pp. 667–673.

[5] N. Saravanan and D. B. Fogel, "Evolving neural control systems," *IEEE Intell. Syst.*, vol. 10, no. 3, pp. 23–27, Jun. 1995.

[6] D. E. Moriarty and R. Miikkulainen, "Forming neural networks through efficient and adaptive coevolution," *Evol. Comput.*, vol. 5, no. 4, pp. 373–399, 1997.

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[10] C. Szegedy *et al.* (2014). "Going deeper with convolutions." [Online]. Available: http://arxiv.org/abs/1409.4842
[11] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for training large scale neural network language models," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand. (ASRU)*, Dec. 2011, pp. 196–201.
[12] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[13] F. Gruau *et al.*, "Neural network synthesis using cellular encoding and the genetic algorithm," Ph.D. dissertation, School Lab. l'Informatique Parallilisme, École Normale Supérieure de Lyon, Lyon, France, 1994.
[14] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.
[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[16] Y. Kassahun and G. Sommer, "Efficient reinforcement learning through evolutionary acquisition of neural topologies," in *Proc. 13th Eur. Symp. Artif. Neural Netw. (ESANN)*, Apr. 2005, pp. 259–266.
[17] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.
[18] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, Mar. 2008.
[19] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. Genetic Algorithms Genetic Algorithms Appl.*, Hillsdale, NJ, USA, 1987, pp. 41–49.
[20] S. Nolfi, "Evolving robots able to self-localize in the environment: The importance of viewing cognition as the result of processes occurring at different time-scales," *Connection Sci.*, vol. 14, no. 3, pp. 231–244, 2002.
[21] D. Precup and R. S. Sutton, "Multi-time models for temporally abstract planning," in *Proc. Adv. Neural Inf. Process. Syst.*, 1998, pp. 1050–1056.
[22] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *J. Mach. Learn. Res.*, vol. 3, no. 1, pp. 115–143, 2003.
[23] R. W. Paine and J. Tani, "How hierarchical control self-organizes in artificial adaptive systems," *Adapt. Behavior*, vol. 13, no. 3, pp. 211–225, Sep. 2005.
[24] I. B. Levitan and L. K. Kaczmarek, *The Neuron: Cell and Molecular Biology*. London, U.K.: Oxford Univ. Press, 2002.
[25] S. J. Kiebel, J. Daunizeau, and K. J. Friston, "A hierarchy of time-scales and the brain," *PLoS Comput. Biol.*, vol. 4, no. 11, p. e1000209, 2008.
[26] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, *Principles of Neural Science*, 5th ed. New York, NY, USA: McGraw-Hill, 2013.
[27] E. Marder and V. Thirumalai, "Cellular, synaptic and network effects of neuromodulation," *Neural Netw.*, vol. 15, nos. 4–6, pp. 479–493, Jun./Jul. 2002.
[28] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining Knowl. Discovery*, vol. 2, no. 4, pp. 345–389, Dec. 1998.
[29] M. Golea and M. Marchand, "A growth algorithm for neural network decision trees," *EPL (Europhys. Lett.)*, vol. 12, no. 3, p. 205, 1990.
[30] J. A. Sirat and J.-P. Nadal, "Neural trees: A new tool for classification," *Netw., Comput. Neural Syst.*, vol. 1, no. 4, pp. 423–438, 1990.
[31] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.
[32] J.-B. Mouret and P. Tonelli, "Artificial evolution of plastic neural networks: A few key concepts," in *Growing Adaptive Machines*. Berlin, Germany: Springer, 2014, pp. 251–261.
[33] J. Urzelai and D. Floreano, "Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments," *Evol. Comput.*, vol. 9, no. 4, pp. 495–524, Dec. 2001.
[34] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving adaptive neural networks with and without adaptive synapses," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 4. Dec. 2003, pp. 2557–2564.
[35] A. Soltoggio, P. Dürr, C. Mattiussi, and D. Floreano, "Evolving neuromodulatory topologies for reinforcement learning-like problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Sep. 2007, pp. 2471–2478.
[36] A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano, "Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios," in *Proc. ALIFE*, 2008, pp. 569–576.
[37] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley, "How novelty search escapes the deceptive trap of learning to learn," in *Proc. 11th Annu. Conf. Genet. Evol. Comput.*, 2009, pp. 153–160.
[38] Y. Niv, D. Joel, I. Meilijson, and E. Ruppin, "Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors," *Adapt. Behavior*, vol. 10, no. 1, pp. 5–24, Jan. 2002.
[39] T. Kondo, "Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control," *Appl. Soft Comput.*, vol. 7, no. 1, pp. 189–202, Jan. 2007.
[40] D. Lessin, D. Fussell, and R. Miikkulainen, "Open-ended behavioral complexity for evolved virtual creatures," in *Proc. 15th Annu. Conf. Genet. Evol. Comput. Conf.*, 2013, pp. 335–342.
[41] D. V. Vargas, H. Takano, and J. Murata, "Novelty-organizing team of classifiers in noisy and dynamic environments," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, May 2015, pp. 2937–2944.
[42] T. Kohonen, *Self-Organizing Maps*, vol. 30. Berlin, Germany: Springer-Verlag, 2001.
[43] B. Fritzke, "A growing neural gas network learns topologies," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 7. 1995, pp. 625–632.
[44] D. V. Vargas, J. Murata, H. Takano, and A. C. B. Delbem, "General subpopulation framework and taming the conflict inside populations," *Evol. Comput.*, vol. 23, no. 1, pp. 1–36, Mar. 2015.
[45] K. Stanley, I. V. Karpov, E. Bahceci, and T. D'Silva. (2011). *NEAT C++*. [Online]. Available: http://nn.cs.utexas.edu/keyword?neat-c
[46] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. 8th Adv. Neural Inf. Process. Syst.*, 1996, pp. 1–7.
[47] F. J. Gomez and R. Miikkulainen, "Solving non-Markovian control tasks with neuroevolution," in *Proc. IJCAI*, vol. 2. 1999, pp. 1356–1361.
[48] A. Ethembabaoglu and S. Whiteson, "Automatic feature selection using FS-NEAT," Intell. Auto. Syst. Group, Univ. Amsterdam, Amsterdam, The Netherlands, Tech. Rep. IAS-UVA-08-02, 2008.
[49] N. Kohl and R. Miikkulainen, "Evolving neural networks for fractured domains," in *Proc. 10th Annu. Conf. Genet. Evol. Comput.*, 2008, pp. 1405–1412.
[50] N. Kohl and R. Miikkulainen, "An integrated neuroevolutionary approach to reactive control and high-level strategy," *IEEE Trans. Evol. Comput.*, vol. 16, no. 4, pp. 472–488, Aug. 2012.
[51] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genet. Program. Evolvable Mach.*, vol. 8, no. 2, pp. 131–162, Jun. 2007.
[52] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, 2009.

Danilo Vasconcellos Vargas received the B.Eng. degree in computer engineering from the University of São Paulo, São Paulo, Brazil, in 2009, and the M.Eng. degree from Kyushu University, Fukuoka, Japan, in 2014, where he is currently pursuing the Ph.D. degree.

His current research interests include general learning systems which include research in evolutionary computation, machine learning, artificial intelligence, neural networks, optimization, and their applications.

Mr. Vargas received the Baden-Wüttemberg Scholarship to study one year with Ulm University, Ulm, Germany, in 2007. He has been receiving the Monbukagakusho Scholarship to study with Kyushu University since 2011.

Junichi Murata (M'98) received the master's and D.Eng. degrees from Kyushu University, Fukuoka, Japan, in 1983 and 1986, respectively.

He then became a Research Associate with the Faculty of Information Science and Electrical Engineering, Kyushu University, where he is currently a Professor. His current research interests include learning systems, optimization techniques, and their applications, especially to energy management systems.

Prof. Murata is a member of The Society of Instrument and Control Engineers, Institute of Systems, Control and Information Engineers, and IEEJ.