Term essay for IT-8008 (IT-3708) – Sub-symbolic AI methods

Adaptive ANN-controlled intelligent agents

Stefano Nichele

Bio-inspired Machines, IDI, NTNU
Sem Sælandsvei 7-9, NO-7491
nichele@idi.ntnu.no

June 6, 2011

INDEX:

Abstract

This document represents the final essay for the PhD course IT8008 – Sub-symbolic AI Methods, which is part of my doctoral study program. This report aims to give an extensive coverage of the state-of-the-art in adaptive-ANN-controlled intelligent agents.

The human brain is a very efficient computational machine. Yet "slower" that a standard computer, it processes a huge amount of information in a highly parallelized and optimized way. Thus, the motivation of studying Artificial Neural Networks is that the brain can be considered as a model for creating intelligent machines capable of simulating the structure and behavior of a biological neural networks. Some tasks that may be trivial for a 3 years old baby can be very challenging for a computer (image and pattern recognition, associations, perceptions, spatial and temporal consciousness). One solution is to take inspiration from nature in order to reproduce the same behavior in machines.

1 Introduction

The human brain is one of the most complex and fascinating organs. Unlike other parts in our body, it is more than just a biochemical machine and its dynamics are still far beyond from being understood. The human central nervous system has been studied for hundreds of years but only recently neural models have been proposed.

One of the earliest work dates back to 1943, when McCullock and Pitts [23] introduced the first model of artificial neuron that could be build and simulated using electric circuits. Figure 1(a) shows a typical nerve cell while Figure 1(b) represents McCullock and Pitts' artificial neuron's model, where every neuron is a threshold unit with excitatory and inhibitory synapses. It fires an impulse *y* along its axon at time *n+1* if the weighted sum of its inputs *x1,...,xm* and weights *w1,...,wm* at time *n* exceeds the threshold value θ.

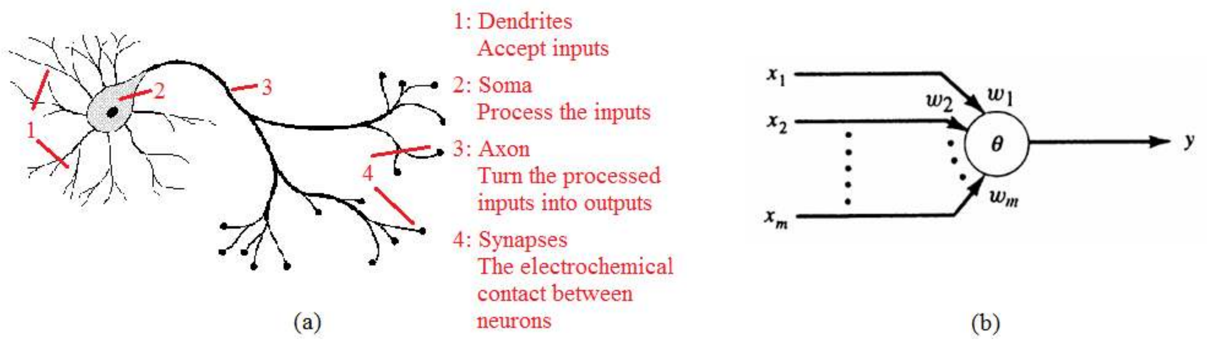

FIGURE 1: (a) a nerve cell; (b) McCullock and Pitts' artificial neuron [2].

In 1949 Hebb wrote the book “The Organization of Behaviour” [30]. He supported McCullock and Pitts’ idea, describing that every time a neural path is used, it becomes stronger. During the ‘50s a lot of research on neural networks has been conducted, culminating with the introduction of the first practical artificial neural network in 1958 by Rosenblatt: the perceptron [24]. Later, in 1969, Minsky and Papert [29] showed the weaknesses of the perceptron and, specifically, that data must be linearly separable. It was not possible to solve a simple XOR, even using multi-layered perceptrons. In the meantime, in 1959 and 1960, Wildrow and Hoff developed ADALINE and MADALINE, the first examples of neural networks applied to real problems, such as to eliminate echoes on phone lines.

In the following years, research on neural networks slowed down, most likely because of the high expectations in the collective imagination, driven by science fiction books and movies (Asimov’s books, the novel “2001: a space odyssey”, etc.), but not comparable practical results. Only in 1982, the field was revitalized by a scientific paper presented by Hopfield, where he showed how to use neural networks to solve dynamic problems, giving also a strong mathematical analysis. Japan and United States restarted immediately with research in the field of AI and neural networks. This new impulse led to the presentation of the back-propagation algorithm as learning technique for multi-layer perceptrons (in 1986 by Rumelhart, Hinton and Williams).

Nowadays, artificial neural networks (ANNs) have several applications, such as language processing, image compression, text and pattern recognition, signal processing, systems control and robotics. The main challenge is to build autonomous and “intelligent” robots using ANNs, simulating the behaviour of a brain. Simulating the whole human brain is still not feasible with the current technology, considering that it contains 100 billion neurons and 10,000 times as many connections.

Computational neuroscientists, such as Sebastian Seung, are trying to create a Connectome [11], a complete map of the neurons in a brain and their interactions. They think that experiences, memories and emotions are shaping the connectome continuously, creating new links and causing the old ones to die. These biological principles could be applied in the future to create more accurate models of artificial brains [12] and, maybe, to “download” a real brain in an artificial one or vice versa.

Looking beyond the horizon, some scientists think that biological systems can remain unchanged for several hundred years at cryogenic temperatures, also brains. Paying around 100,000 dollars it is possible to have your brain frozen after death and keep it stored in a nitrogen tank in the US [8]. It may be possible, with a future technology, to unfreeze the brain and recover the memory. Is this only a crazy idea or it will be possible to cheat death?

2 Glossary: from Neurons to Robots

Neuron: a nerve cell is called neuron. It is basically a biological cell that processes information. A neuron receives signals from other neurons through his dendrites and elaborates the data in the soma, the cell body. The generated signals are then transmitted through the axon which eventually branches into different filaments connected to dendrites of other neurons. These interconnections are called synapses [13]. When an impulse is propagated from one neuron to another, some chemicals called neurotransmitters are released to enhance or inhibit the signal. The strength of the synapses can be affected by its own activity in a sort of learning process. This memory-like phenomenon may be responsible for the human memory. A graphical representation of a neuron is given in Figure 1(a).

Biological neural network: it is a group of interconnected neurons. One example is the cerebral cortex.

Artificial neural network: "a computer program designed to model the human brain and its ability to learn tasks" [28]. An ANN is a group of interconnected artificial neurons, represented in Figure 1(b), where the activation function can be a threshold, a linear combination, a sigmoid or a Gaussian. They hold the same feature of their biological counterparts of learning an association between some input and output values, based on experience, developing a sort of memory association. The most used activation functions are shown in Figure 2.

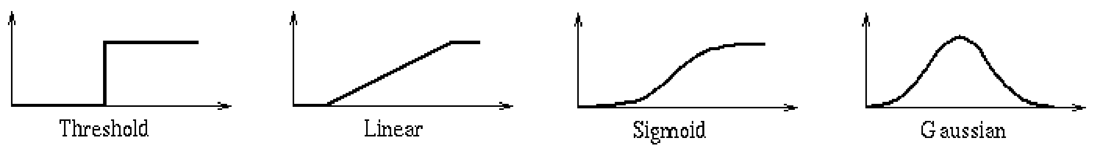

FIGURE 2: different types of activation functions: threshold, linear, sigmoid and Gaussian.

Adaptive ANN: the magic word is adaptation, the ability to change the behaviour in dynamic environments. Adaptation is based on learning or, in other words, to update the network structure and connection weights online to perform a specific task. The news here is that the behaviour of the ANN does not have to be programmed with specific rules. It is the network itself that learns the underlying rules from experience.

Intelligent Agent: in artificial intelligence, an agent is an entity that reacts to some perceptions from the environment and decides which action to perform in order to achieve a specific task. Intelligent agents (IAs) may be able to learn how to behave and to use this knowledge to be autonomous. A typical representation of a simple intelligent agent is shown in Figure 3.

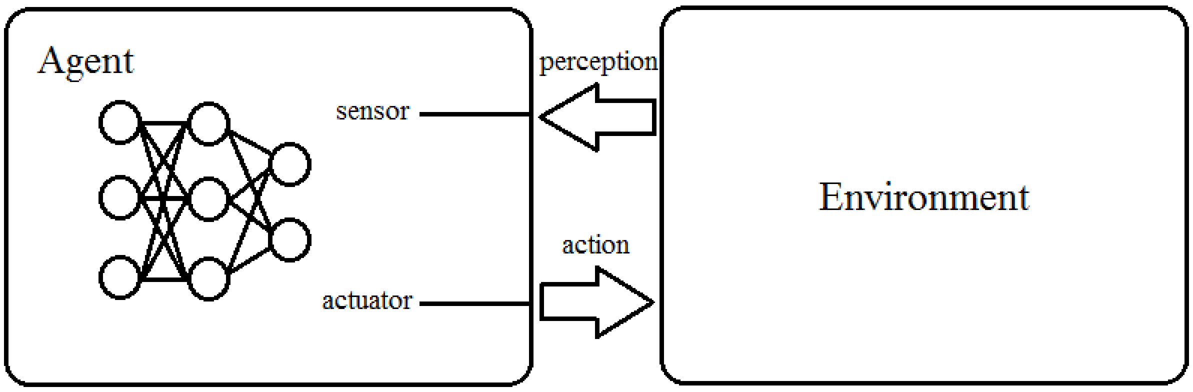

FIGURE 3: example of a simple intelligent agent.

Adaptive-ANN-controlled intelligent agents: a category of intelligent agents where the association between perception and reaction is governed by an artificial neural network that is adaptive and thus can learn. This kind of IA is widely used for robots controllers.

3 Formal Definition

An artificial neural network (ANN) is a collection of interconnected neurons. Input neurons receive input signals from the environment and deliver it through their weighted links to the following neurons. In some cases, there are some neurons that are not connected with the external environment. These units are also referred as hidden neurons [9]. The signals travel along the different layers of the network, eventually reaching output neurons that deliver signals to the environment. Each signal travels independently from other signals through the different weighted links and every single neuron can update its state in parallel. The network architecture is usually separated in layers, as shown in Figure 4.

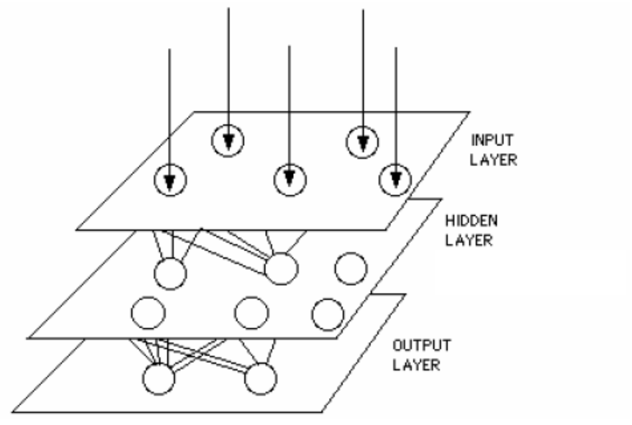

FIGURE 4: layer architecture of an artificial neural network [9].

When a signal passes through a connection, its value is reinforced or reduced by the link's weight. The output signal is usually calculated with the equation (1).

$$y_i = \sum_{j=1}^{N} \Phi\,(\,x_j * w_{ij}\,) \tag{1}$$

The function Φ may take into account a threshold θ (also called bias) that represents a level beyond which a neuron becomes active. Commonly used output functions are the Heaviside function (2), the linear function (3), and the sigmoid function (4), where k is a constant value.

$$\Phi(x) = \begin{cases} 1 & if\ x > \theta \\ 0 & otherwise \end{cases} \tag{2}$$

$$\Phi(x) = kx \tag{3}$$

$$\Phi(x) = \frac{1}{1+e^{-kx}} \tag{4}$$

Every network is defined by the number of neurons and their interconnections. The direction of the connections differentiates the ANNs into two groups:

- Feed-forward networks: the signals travel from the input layer to the output layer and no loops can occur.
- Feed-back networks (or recurrent networks): the dynamics of the network are more complex because there may be connections among neurons on the same layer or to a previous layer; the signals can travel backwards creating loops.

In feed-forward networks, for every input set one single output set is produced, independently from the previous state of the network (memory-less). On the other hand, with recurrent networks, there may be a sequence of outputs, each one dependent on the previous state of the network (dynamic system).

In order to let the artificial neural network compute a specific task, every connection has to be associated with a weight. This operation cannot be performed manually (the network cannot be "programmed"). What is needed is a learning process that helps the network to learn and update the values of the weights from experience. The mechanism of updating weight is called leaning rule. There are four main types of learning strategies:

- Supervised learning: the weights are learned from an example set, calculating the difference between the desired and the obtained result. Once the network is trained, it can be utilized to solve the assigned task. The number of training samples plays an important role, too few patterns may cause over-fitting, where the network performs well for a specified training data-set but fails for independent test patterns.

- Unsupervised learning: there is no teacher telling the network which is the correct answer for a specific input. Weights are modified by the network itself that tries to find the underlying properties of the data, finding patterns, correlations and similarities.
- Hybrid learning: a combination of the two previous methods, where some weights are updated using the first strategy and others using the second.
- Reinforcement learning: it is a special case of supervised learning. The difference is that the expected output is unknown and the only information available is whether the actual output is correct or not.

4 Adaptation

Adaptability of the artificial neural network is one of the most important properties. It gives the ability to change the network's behavior to respond to changes in the environment [14]. Since the network architecture is determined by its topology, connectivity and transfer function in each node, an ANN can adapt using different techniques:

- Adaptation by parameter adjustment
- Adaptation by neuronal property modification
- Adaptation by structure modification

Each different type of adaptation can be achieved in many ways. The most common strategies are presented in the following paragraphs.

4.1 Adaptation by parameter adjustment

The most important parameters in a neural network are the connection weights. In order to adjust the weights, there are different approaches, depending on the learning style (supervised and unsupervised) and several learning rules are presented in literature. The most common rules are:

- Hebbian learning rule;
- Delta rule;
- Backopropagation.

A description of each learning rule is given in the following sub-sections.

4.1.1 Hebbian Learning Rule

Observing the behavior of the nervous system, Hebb in 1949 theorized that if two connected neurons are active together, the weight of their connection becomes stronger [26]. For an artificial neural network, this means that the amount of change in a connection (Δw) is:

$$\Delta w_{ij} = \eta * y_i * x_j \tag{5}$$

This value is proportional to the activation values of the pre-synaptic (x) and post-synaptic (y) neurons, mitigated by a learning rate (η).

The Hebbian learning rule is mostly used with unsupervised learning strategies [16]. With supervised learning, the weights are modified during the training phase and once the training is finished, during the testing phase the weight update can be a problem.

4.1.2 Delta Rule

In supervised learning the adopted strategy aims to change the weights in order to minimize the error between the expected and the actual result [22]. If there is only one layer of connections such strategy is referred as Delta rule. The training rule can be described by the following function:

$$\Delta w_{ij} = \eta * (t_i - y_i) * x_j \tag{6}$$

where *t* represents the desired response and *(t – y)* is the produced error *(δ)*.

4.1.3 Backpropagation

The described delta rule can be applied only if the input vectors are linearly separable. If the inputs are nonlinearly separable, a hidden layer has to be introduced in order to recode the input vectors. Moreover, a nonlinear function has to be used. This method is called generalized delta rule or backpropagation (BP). It consists on the propagation of the delta error on the output neurons back to the hidden layer [3]. The used non linear function is the sigmoid.

With backpropagation, the training patterns are given in input randomly many times and the weights are modified after every presentation of the patterns. The error function (denoted by E in Figure 5), often calculated as the mean square error between the desired and obtained results, is minimized through gradient descent. The gradient represents the direction of maximum growth of the cost function. Since the goal is to minimize it, it is necessary to move towards the opposite direction. The error function is a n-dimensional surface, where the n-dimensions corresponds to the number of synaptic weights *(w)* and the gradient is the tangent to the surface, defined by a vector of partial derivates of the function in respect to each weight.

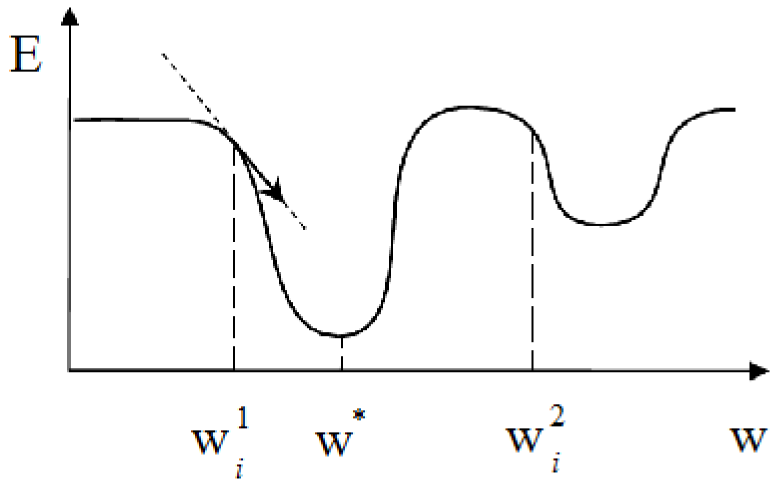

FIGURE 5: example of gradient descent. If the initial learning position is w_i^1 and the error has o be minimized in the direction towards w^*, the gradient is the tangent and the slope is represented by the derivate in that point.

Theoretically, the function E forms a hyper-surface in the n-dimensional space with dimensions $w = (w_{i1}, w_{i2}, ..., w_{in})$ and it has a minimum at a point w^* in the weight space, where partial derivate of E with respect to vector w is zero.

Following the steepest slope not always has the desired result. In fact, it is possible to stick in local minimum. This is shown if Figure 5, starting from w_i^2. A possible solution is to use momentum to avoid stagnation in local minima, adding inertia to the movement of the network in the error space.

4.1.4 Adaptive Resonance Theory

During online learning, it is important that the network adapts to new patterns without losing the accuracy on already learned information. In other words, the network has to show two contrasting properties: stability (ability to remember stored information) and plasticity (ability to adapt to new information). In 1987, Carpenter and Grossberg presented a technique called Adaptive Resonance Theory (ART) [17]. It is a hybrid strategy that separates the network in two parts, the gain network (F1) and the reset network (F2), as shown in Figure 6. Neurons in

F1 are fully connected to neurons in F2 and vice versa. These two layers of neurons differ. In fact the neurons in F2 adopt the strategy "winner-takes-all". Inputs are connected to layer F1 and a category is chosen and presented to F2. Then F2 sends back a signal called expectation to F1. If the network fails to resonate with the expectation signal then the input is a new uncategorized pattern which is associated to a new category (a neuron in F2) and the weights are adjusted accordingly to recognize also the new group.

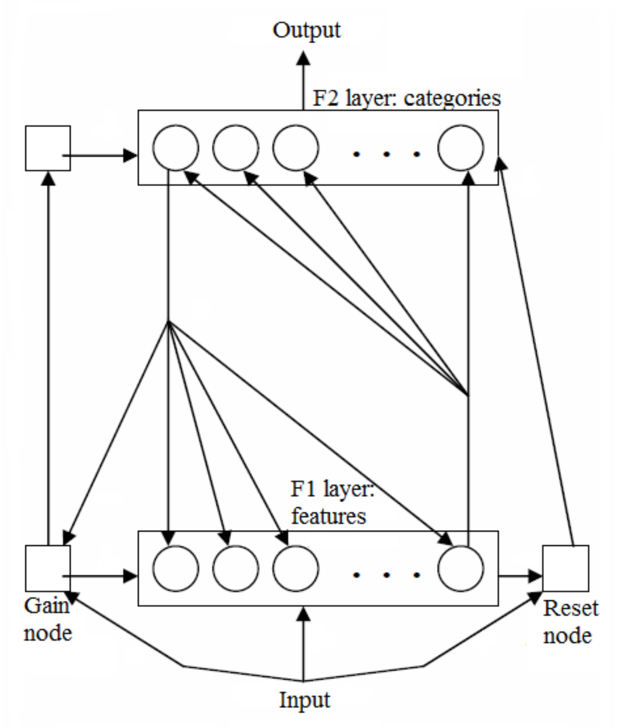

FIGURE 6: Adaptive Resonance Network (ART) diagram [15].

4.2 Adaptation by neuronal property modification

The word learning in the context of neural networks often refers at the connection weight modifications or, in other words, at the axon and synaptic level. The neuron itself, the nucleus, is considered as a passive element that processes information in a repetitive way. However, it is possible to include the neuron in the learning process, modifying the computed function inside the neuron. This is referred in literature as temperature modification [18]. As weights are optimized reducing an error function, the same could be done in order to minimize the temperature error inside the neuron. The temperature property is strictly related to the sigmoid function in each neuron and, in particular, to the shape and slope of the sigmoid curve. These parameters can be also changed during the learning process.

Another possibility is to use a more complex neuron with a different structure that uses a multivariate activation function. In this model, inputs in each neuron are not received through a single connection but through N terminals (or dimensions) and the activation function is defined as a N-dimensional hyper-surface [19]. The learning process is then governed by the same rules as in backpropagation.

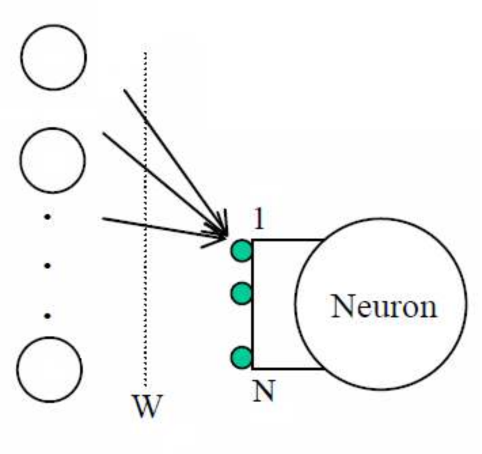

FIGURE 7: Multidimensional neuron diagram with N input terminals. Adapted from [19].

4.3 Adaptation by structure modification

In this strategy, what changes is the structure of the network instead of the learning rules. Several studies have been conducted on adaptation by structure modification. Here, two relevant implementations are presented.

4.3.1 Adaptive modification of hidden layer in RBF network

Radial Basis Function Networks (RBFNs) are particular neural networks where neurons in the hidden layer use radial basis functions as activation functions while the neurons in the input and output layers use linear functions. Such networks are mostly used for classification and pattern recognition problems. The number of neurons in the hidden layer represents the number of clusters in a multi-dimensional space and thus, it can be kept variable and modified online, in order to adapt to the characteristics of the input data [20]. The calculation of the cluster centroid can be determined using different clustering techniques, such as k-means, fuzzy logic clustering, hierarchical agglomerative clustering method (HACM) and so on.

4.3.2 Evolving connectionist systems

Kasabov [21] developed an adaptive leaning technique for speech recognition, based on the online modification of the network topology. Evolving Connectionist System (ECOS) starts with a skeletal neural network and adaptively update the network structure in response to the input data, adding or removing input and output nodes. It uses a hybrid strategy (supervised and unsupervised) to allocate nodes from the input data, behaving as a "self-programming" device.

5 Evolutionary ANNs

Learning is not the only form of adaptation that can be used. Evolutionary algorithms can be also utilized to optimize connection weights, network structure modelling, learning rules, input feature selection, weight initialization, etc.

5.1 Evolutionary algorithms

Evolutionary algorithms (EAs) are a group of stochastic optimization algorithms based on natural principles. They are able to explore a huge solution-space in a clever manner, finding an optimal solution in a reasonable time. They are based on the concept of natural evolution, where the genotype represents the genetic instructions and the phenotype is the organism that emerges from a process called morphogenesis, which consists of the interpretation of the genotype in a specific environment.

The genotype is usually a very large set of instructions and each one is triggered and executed under specific conditions. In other words, the phenotype is the result of a development process which is a nonlinear function of the genotype. For this reason, it is very difficult to predict the specific phenotype that will emerge from a given genotype. It is also very unlikely to understand which modification should be done on the genotype in order to change a characteristic in the phenotype. The only method available is to try and check the results. This is what is called evolution by natural selection.

In the context of neural networks, the components that we want to evolve (learning rules, weights, network structure) have to be mapped into artificial chromosomes that are subject to the evolution process and evaluated through a fitness function. High ranked elements then have more chances of being selected to contribute with their genetic material for the creation of future generations. In other words, the fittest element will survive and the weakest will die.

Two popular evolutionary techniques are Genetic Algorithms (GAs) and Genetic Programming (GP). In GAs the individuals are represented as bit strings on which different nature-inspired operators, such as crossover, and mutation are applied. GP utilizes trees to represent the individuals in the population, where each tree represents a sort of program and the evolution process a “programmer”. Operators can still be applied but it is important to check the validity of the result, since the target tree can be an invalid program.

In this paper we focalize on genetic algorithms. A detailed description is given in the next section, while practical applications are described right after.

5.2 Genetic Algorithms

GAs use artificial evolution as a process that reproduces the natural selection over a population of individuals. In this context, some key principles have to be fulfilled in order to have a true artificial evolution process:

- Heredity: parents and offspring are similar, the copy of the genome to the next generation is accurate;
- Variability: even if the copy process has to be accurate it is not perfect so the offspring is not an exact copy of the parents;
- Fecundity: variation influences the behaviour of the new generation and the change of behaviour affects on the success of reproduction.

A genetic algorithm is an idealized computational version of Darwinian evolution. In Darwinian evolution, organisms reproduce at differential rates, with fitter organisms producing more offspring than less fit ones. Offspring inherit traits from their parents; those traits are inherited with variation via random mutation, sexual recombination, and other sources of variation. Thus traits that lead to higher reproductive rates get preferentially spread in the population, and new traits can arise via variation.

A GA is often composed by the following steps:

- Generation of a random initial population of individuals.
- Evaluation of the “quality” of the phenotypes in the population through a Fitness Function. The level of fitness indicates the probability of success of the phenotype in the environment;
- Selection of high quality individuals in the population (high fitness means high probability of being chosen for the reproduction process);
- Crossover: recombination of the genotypes from the chosen phenotypes to produce offspring. This step can be done with different methods, an example is shown in Figure 8;

- Mutation: random modification of small parts of the new genotype with low probability (mutation rate).

This process is iterated for many generations, at which point hopefully one or more high-fitness individuals have been created.

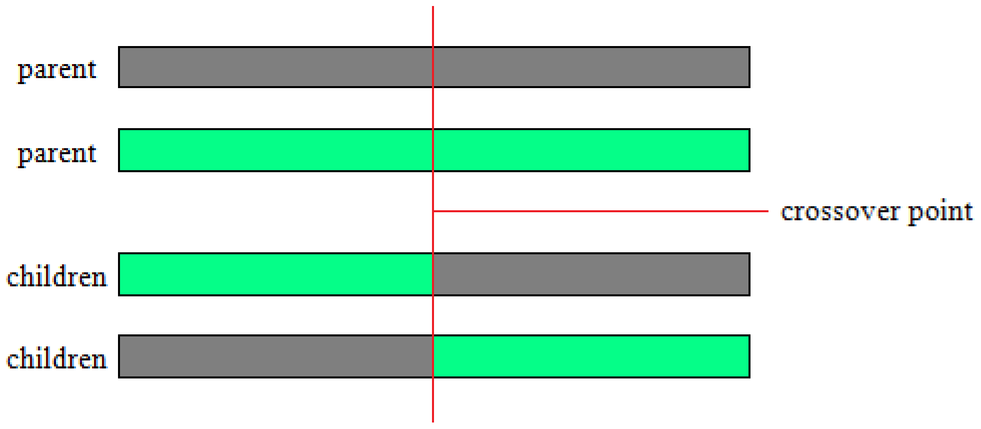

FIGURE 8: One-point uniform crossover: a single crossover point on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms. The resulting organisms are the children.

5.3 Evolving connection weights

Here the genome represents connection weights [10]. Sometimes, using backpropagation, the gradient descent remains trapped into local minima of the error function or, even worst, the function is non-differentiable. In such cases, it is possible to use evolution, where the environment is composed by the network architecture. The fitness function is formulated in terms of error between the actual and target output. Moreover, this function does not have to be differentiable or even continuous. The evolutionary process can be divided in two different phases:

1. The representation of the connection weights have to be decided (e.g., binary string);
2. The search operators have to be decided (e.g., crossover, mutation).

Differently tuned operators in combination with different representation could produce very dissimilar results in terms of performances. The most widely used representation is a binary string, where the binary representation of the connection weights is concatenated following a well defined orders, for example input layer first, hidden layer after and output layer last. Using a binary representation there are advantages and disadvantages. First of all, it is extremely important the number of bits used to represent a single weight. In fact, using too many bits would expand the search space whether using too few would reduce the approximation of binary represented real values. Moreover, it is very difficult to use crossover since the position of each weight is crucial and feature detectors found during the evolutionary process could be destroyed.

Another issue is the so-called permutation problem (or competing convention problem). It arises because the genotype-phenotype mapping is a many-to-one representation and different binary strings can abstract neural networks with the same functionality. This situation is shown in Figure 9.

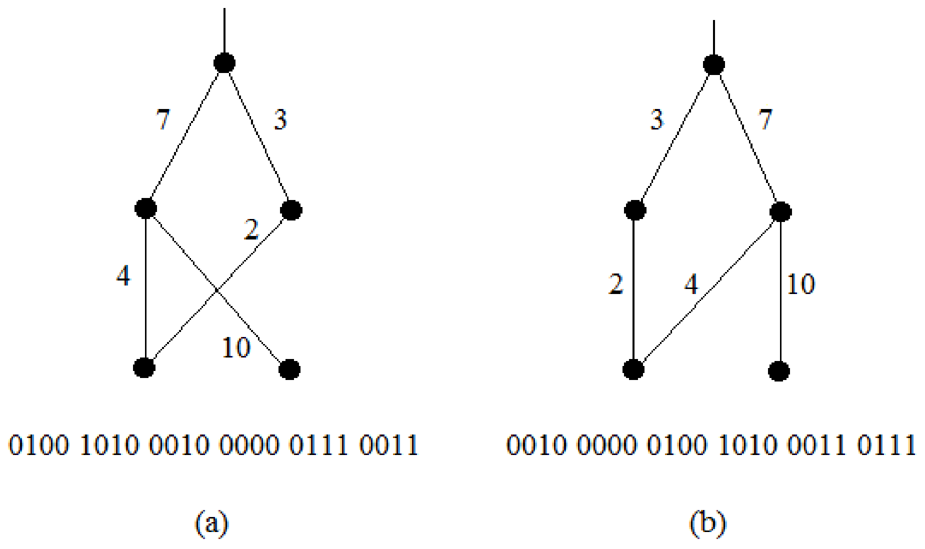

FIGURE 9: (a) and (b) represent two different neural networks in terms of binary representation (4 bits for each weight) with equivalent functionality [10].

Another possibility is to use a hybrid technique that exploits the benefits of both evolution and backpropagation. Specifically, EAs can be used to find a sub-region in the solution space and then locally search the final solution with BP.

5.4 Evolving architecture

In the previous section, a predefined fixed architecture has been considered along the whole evolution process. This section shows that it is possible to evolve topological properties such as connectivity or transfer function of each neuron. Usually, the network design and structural modeling is performed by humans. It is a kind of engineering task that mostly relies on experience. It is difficult to find a systematic way of automating the process. In general, two different approaches can be used:

- Constructive approach: staring from a minimal network the algorithm adds layers, nodes and connections, if needed;
- Destructive approach: starting from a maximal network, the unnecessary components are removed.

The evolution process can be formalized as a search in the architecture space and each point in the solution space represents a different architecture. The fitness function can measure different parameters, e.g., lowest training error, lowest structural complexity. Using a constructive technique, an issue that has to be taken into account is that the solution space can be infinite because the possible number of network components (nodes, layers) is unbounded [27]. Moreover, similar architectures can have different performances and different architectures can have similar behavior.

As for the evolution of connection weights, the network developer has to choose the genotype representation and the evolutionary algorithm characteristics. For the representation, it is possible to use direct or indirect encoding. Direct encoding includes all the connections and nodes details in a matrix, while indirect encoding specifies only some important network features. Unfortunately, for both encoding types the permutation problem is still present.

The transfer function in each neuron is also something that is often defined a priori. Usually, it is assumed to be the same for all the nodes, at least inside the same layer. It is possible to apply evolution to decide whether a node should use a specified transfer function (for example a sigmoid) or another (a Gaussian).

5.5 Evolving rules

The same learning rule can have very dissimilar performances with different architectures, it is somehow architecture dependent. For example, there exist many variant of the Hebbian learning rule, each one optimized for a specific architecture. However, it may be difficult to choose the learning rule in advance, especially if the network structure and topology is unknown. It may be better to adapt the learning rule in a dynamic way, depending on the underlying architecture and the specific task that has to be performed. In other words, the artificial neural network should have the ability of “learning to learn”. Using artificial evolution, it is possible to evolve learning rules, giving to ANNs an emergent creativity, similar to the creative process of biological systems.

In the category of rules evolution, we can identify two sub-branches:

- Evolution of algorithmic parameters;
- Evolution of the actual learning rules.

Instead of evolving the whole learning rule, it is possible to evolve only some algorithmic parameters. For example, if backpropagation is chosen, the evolution process can be used to adjust the values of learning rate and momentum. In order to use this approach, it may be better to encode into each chromosome the information of parameters and network architecture, in such a way to optimize the exploration of the solution space and search for a well balanced combination of architecture and algorithmic parameters. In some other cases, when the structural design is pre-defined, the evolution can be employed to optimize parameters towards a specific architecture.

Working in a dynamic environment, it may be necessary to evolve the learning rules as a whole. The key issue is to find a good representation for rule's dynamic behavior into fixed and static chromosomes. Usually, some assumptions are made, such as:

- The used learning rule is the same for all nodes;
- The update of weight relies only on local values: input node, output node and local connection weight.

With those assumptions, the learning rule is considered to be a linear function of the local values and their products.

5.6 Other types of ANN evolution

Another plausible way of applying evolution principles together with artificial neural networks is when the number of input features is big and the performances of the network are not good enough. If the number of inputs is large, the network requires more nodes (and thus a bigger size) and more training data in order to be trained (and, as a consequence, more time). There exist many statistical techniques for input preprocessing, for reducing the number of dimensions (e.g., random matrix method), for eliminating redundant input features. Sometimes, the network can have better performances with fewer inputs. This dimensionality reduction can also be achieved by evolution, representing each chromosome as a binary string with the value "1" if the input feature has to be used, or "0" otherwise. The fitness function is calculated with a real training of the network, utilizing only the corresponding active input features in the genetic string.

Another original and uncommon idea is to use evolution in order to produce new training samples, evolving a training set.

More in general, as described in the previous paragraphs, evolution of ANNs can be used in different ways and for very different purposes, to optimize the network structure, the parameters and so on. There have been some very interesting investigations on how to apply evolutionary algorithms to evolve ANNs as a whole. A possible framework [25] for EANNs is presented in Figure 10.

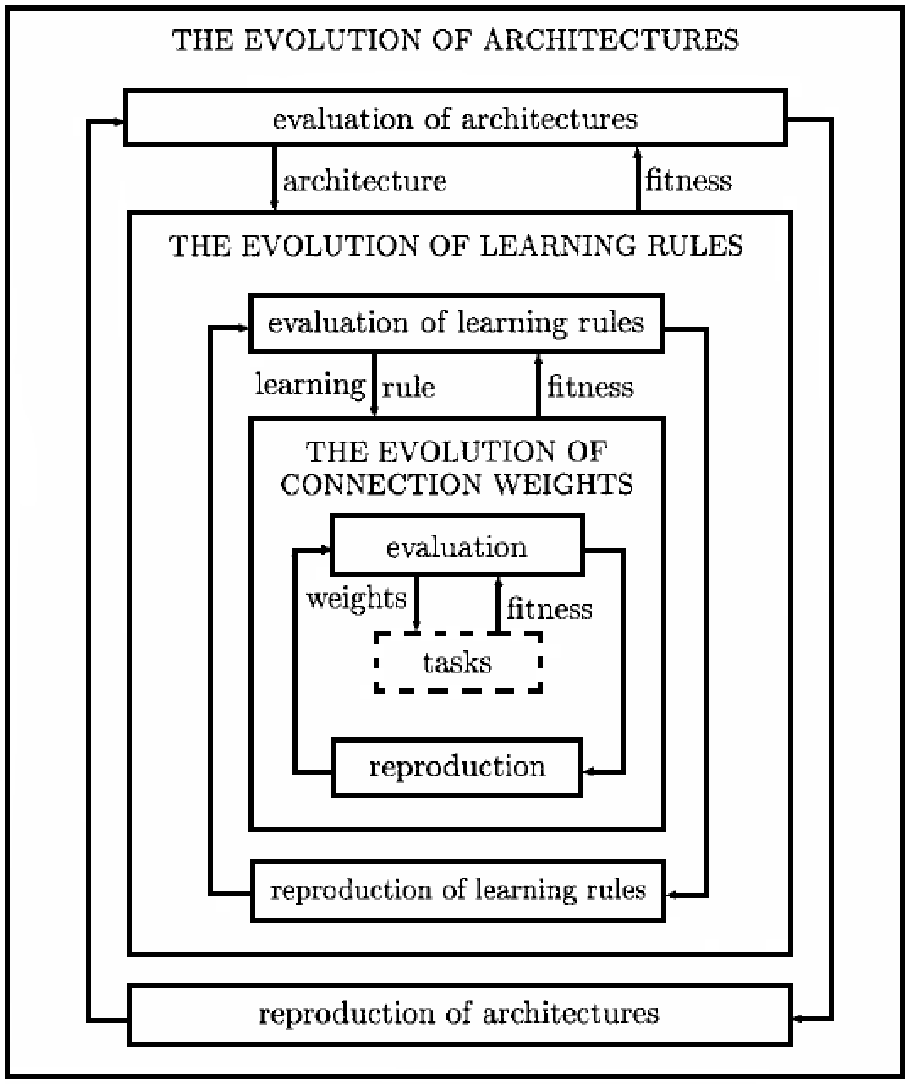
THE EVOLUTION OF ARCHITECTURES
evaluation of architectures
architecture
fitness
THE EVOLUTION OF LEARNING RULES
evaluation of learning rules
learning rule
fitness
THE EVOLUTION OF CONNECTION WEIGHTS
evaluation
weights
fitness
tasks
reproduction
reproduction of learning rules
reproduction of architectures

FIGURE 10: A general framework from evolution of artificial neural networks [25].

Such framework can be considered ad a general methodology for adaptive systems, where the three evolution levels can be increased (or decreased) with the introduction of new fitness interfaces between levels.

6 Introduction on Evolutionary Robotics

Evolutionary robotics is a field that exploits artificial evolution and development in order to create autonomous robots that are able to work without human intervention, developing their own skills without being programmed. Khepera is a widely used robot for experiments on evolution of artificial neural networks as robot controllers. It has infrared proximity sensors that could be used as input signals and motor controlled wheels as output. A computer running the evolution process is usually connected (wired or wireless) and the emergent individuals are downloaded and tested on the robot [1]. A representation of Khepera is shown in Figure 11. It is extremely important to test the behaviour of ANNs with real robots and real environments because not all the physical properties can be always encoded into simulations with enough accuracy and, sometimes, the actual behaviour could be very different from the simulated behaviour.

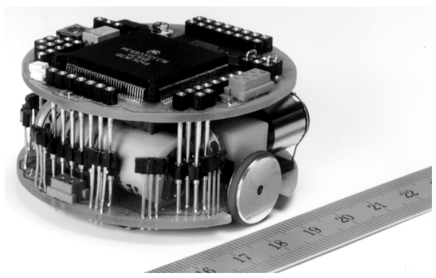

FIGURE 11: Khepera robot.

One of the most common experiments to test the capabilities of an ANN as a robot controller is the T-maze. Widely used for experimentation with rats, it consists on a maze with a T shape, where the robot is placed in the bottom and one reward point is placed in one of the sides of the T-shape. The robot is controlled by an ANN and the fitness function is based on the ability to reach the reward point [7]. The ANN, after many trials, will adapt to the environment and eventually learn how to reach the reward. After some tests, it may be useful to move the reward point on the other side, to test that the robot has actually learned how to reach it and not just to turn all the times in one direction. T-maze is represented in Figure 12.

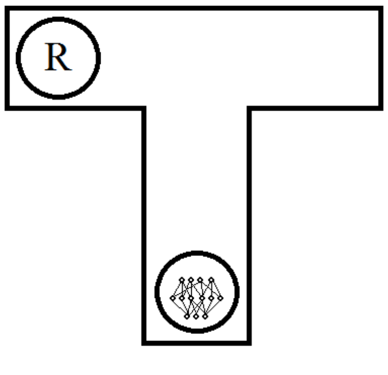

FIGURE 12: A T-maze with one reward point (R).

7 Intelligent Agents and Artificial Evolution of Robots

The process of evolving intelligent agents for robot control is a time consuming task, especially when the emergent control agents are tested on real physical robots in order to evaluate their fitness. On a real scenario, such as the one described by Floreano and Mondada in [31], the experiments carried out on real robots took 10 days whether the same experiments, in a simulated environment running several tests in parallel, would have taken only few hours. However, even if the simulations can be quite accurate and give a good approximation of a real situation, it is extremely important to test the agents on physical robots. Several factors are extremely hard to be simulated, such as:

- mechanical robustness: a physical robot is subject to hardware damage, due to simple wear or because in early generations the behaviour can be unpredictable and the robot may have collisions with other objects in the environment;
- energy supply: power has to be provided, either through batteries or connecting the robot to a power supply with wires;
- differences in real world sensing and actuation: it is extremely difficult to simulate the real world. Moreover, even apparently identical hardware (sensors, motors) may have slightly different behaviours due to minor differences in their mechanics or electronics. Other factors in the environment can influence the sensors and actuators' behaviour, such as temperature of operation, humidity etc. This means that different sensors, exposed to the same external stimulus, react differently;
- measured values are approximations: sensors do not give a 100% accurate measure but just a fuzzy approximation of the reality. Noise can be introduced in simulation, to mitigate this problem. However, the introduction of noise can cause performances drop when the same individual is simulated or tested on a physical robot.

A rule of thumb is that real world validation using physical robots it then necessary.

Khepera robot is equipped with 8 infra-red sensors (distributed around the body) as input devices and lateral wheels (that can rotate in both directions) as output devices. The vision module can operate in two different ways: passive, measuring the amount of infra-red light in the environment (roughly proportional to the amount of visible light), and active, measuring the amount of reflected light (depends on distance from an object and reflectance of the target surface). The active mode can be used as a proximity sensor, where the input signal is discrete and refreshed every 5 microseconds.

When we think about visual system, it is important to keep in mind the difference between what we see and what a robot sees. In fact, robot infra-red photoreceptors are just an approximation of biological neural photoreceptor cells. A Khepera snapshot “image” will look like the signal in Figure 13. That is what the robot sees!

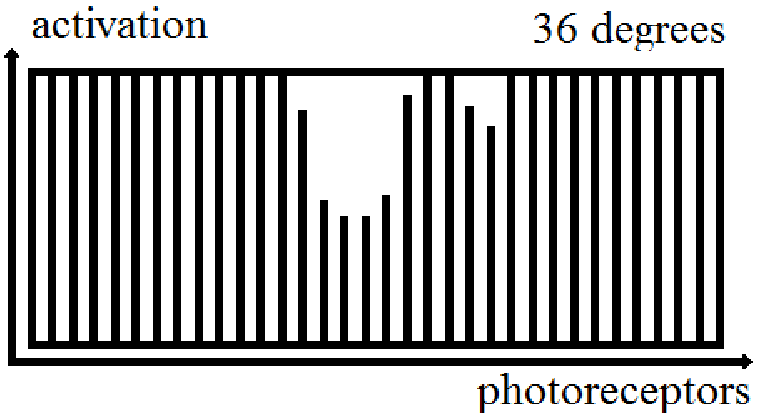

FIGURE 13: A representation of a snapshot from Khepera infra-red vision, covering a visual field of 36 degrees. The robot is in front of a black stripe against a white background. Figure adapted from [1].

7.1 Example: Navigation System

Simple motion with obstacle avoidance is a classical experiment with robots. In general, it does not require internal states or memory. It consists on the creation of an appropriate mapping between input stimuli and output signals. The fact that each movement depends on the incoming input signal, which actually depends on the previous movement, makes it difficult to design a robust navigation system that does not depend on the topology of the environment.

One of the first experiments in the field was conducted by Braitenberg in 1984. He designed a vehicle where wheels were directly connected to the input sensors through weighted connections, as a sort of neural network. This connection setup with excitatory links from the back sensors and inhibitory links from the front sensors have been tested on Khepera robots in a looping maze. Careful tuning of the connections between sensors and motors is required. If no a-priori knowledge about sensor / actuator connections or environment characteristics is present, evolution can be used to search for control agents able to perform the navigation task with obstacle avoidance. The fitness function can be measured as value proportional to the distance or the longest path covered by the robot without hitting any obstacle, in the shortest interval of time. The evolution process searches in the phase space of all the possible solutions where each point in the multidimensional space represents a possible individual and each axis represent a degree of freedom. Results presented in [1] shows that the evolved individuals perform better that Braitenberg’s vehicle.

The evolved neurocontrollers have developed an input-output mapping that is correlated to the physical morphology, shape, sensors and actuator characteristics of Khepera. Interesting questions are: what happens if we replace Khepera with another robot? Is the control agent able of generalization? Does it show any re-adaptation ability using different robots?

Mondada carried out experiments using a bigger mobile robot called Koala. It has six wheels instead of two and sixteen infra-red sensors (only eight were used in the experiments). The control agent was initially developed on a Khepera robot using evolution and afterwards, after 106 generations, the robot was replaced by a Koala. As presented in Figure 14, after a performance drop, the evolution process is able to find fitter individuals in few generations.

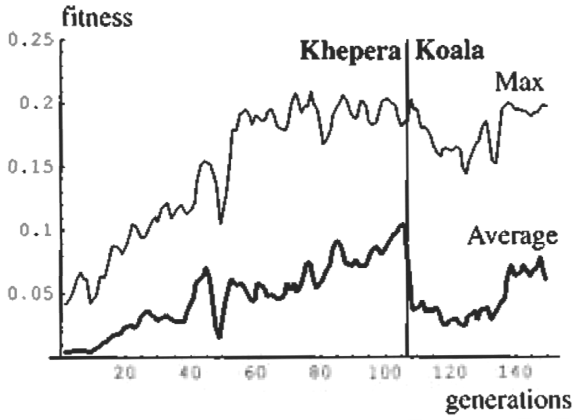

FIGURE 14: Average and Max fitness for each generation. Khepera is used until generation 106, after it is substituted by Koala.

7.2 Sensory-Motor Coordination and Behavior Decomposition

In reactive systems there is a sort of direct link between input and output and the system produces the same motor response if the same sensory input is received. There are some cases where even complicated reactive behaviors can be produced without internal representation of the external environment, using a kind of coordination between perception and action. This process is called sensory motor coordination and it consists on an agent-environment interaction based on the principle that the agent executes an action that changes the position of the agent in the environment or even modifies the environment itself.

Nolfi and Floreano [1] presented an example of sensory aliasing problem that can be solved with sensory-motor coordination. This problem consists on an agent controlled robot that has more than one environmental state (corresponding to the same sensory pattern) that should produce different motor responses. This is done by performing an action that consists on searching for another sensory input that can solve the ambiguity. Experiments were conducted using a Khepera robot equipped with infrared sensors and a linear camera. Infrared sensors were used to detect obstacles proximity and the camera was used to discriminate white and black obstacles. The goal was to avoid black obstacles and approach white ones.

Not every time it is possible to solve complicated tasks with sensory motor coordination, sometimes internal states are necessary. To do that, different neural architectures with a different number of units in an intermediate neural layer were trained with back-propagation algorithm. It turned out that the trained networks (with or without intermediate layer) were able to perform the discrimination task maximum in the 25% of the cases, due to paterns that were not easily separable. Better performances have been obtained using artificial evolution to discover the weights of the neural controller.

Another approach to exploit sensory motor coordination is behavior decomposition. In a behavior based approach, the whole system behavior is divided into sub-behaviors, implemented as different modules. For example the discrimination problem can be approached dividing the whole behavior as shown in Figure 15. However, decomposition can produce sometimes sub-tasks that are more complex that the original problem.

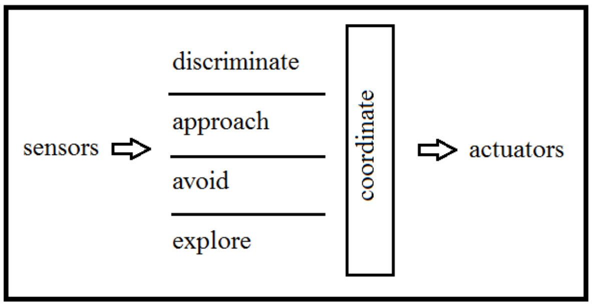

FIGURE 15: Decomposition and integration approach. The whole behaviour emerges out of the integration of sub-behaviours [1].

Behavior decomposition can be also achieved with modularity, where each sub-behavior is associated with a specific sub-component of the controller, called module. A practical example presented by Nolfi [32] is the garbage collection robot. Khepera is placed in an environment surrounded by walls and the goal is to remove the garbage objects. Five different architectures were testes: A- standard feed-forward architecture, B- architecture with an internal hidden layer, C- recurrent architecture, D- modular architecture with two pre-designed modules, E- emergent modular architecture. In this last case, the number of modules and the weights were evolved while the robot was interacting with the environment. The results, presented in Figure 16, show that the emergent modular architecture finds earlier a solution to the problem, even earlier than the handmade modular architecture.

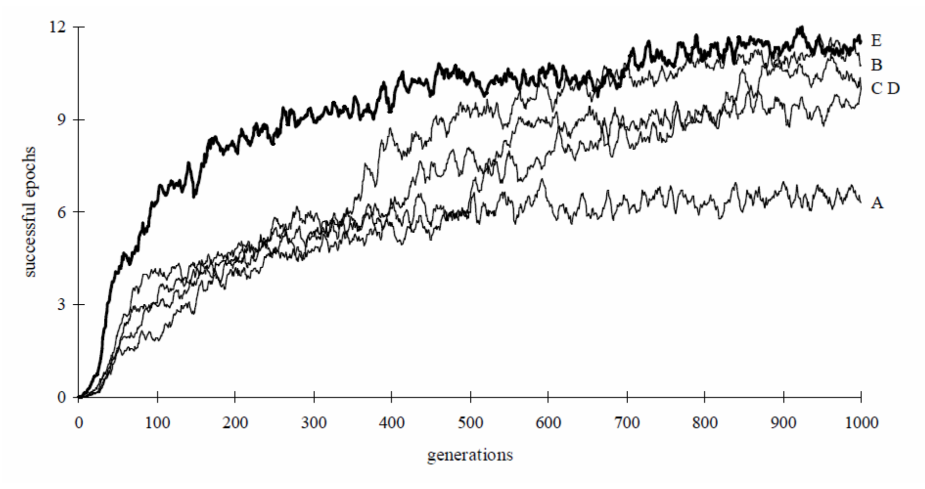

FIGURE 16: Evaluation of the garbage collection task using different architectures [32].

7.3 Co-evolution

In the previous chapters we have seen that evolution is a powerful tool. However, it is used from one generation to another, creating offspring that is more likely to have better performances. Learning, on the other hand, occurs in a specific generation, where the individuals in the population learn along their life-cycle.

If there is more than one population, competitive co-evolution can increase the power of artificial evolution. In literature, the best known example is the co-evolution of predators and prey. In order to survive, the two species have to evolve new strategies from one generation to another: the environment is continuously changing. In the beginning of the evolutionary process, both species are supposed to have a poor behaviour. Gradually, through evolution, both species will develop more complex strategies, producing a sort of incremental evolution. Experiments with Khepera robots show that there is no guarantee that the species will refine their strategy but, on the other hand, they may completely change strategy, adopting a technique that is not more complex but just different, with a better performance for the specific situation but not suitable against one of the strategies of the opponents in the previous generations. This is also known as the cycling problem.

7.4 Walking Machines

One of the most challenging tasks for robots is walking. It requires a greater effort of sensor-motor coordination, compared to wheeled robots. In [1] several examples are presented, depending on the morphology and number of legs of the robots. Many of the experiments have several common aspects: robots are only capable of static walk (if the robot stops the walk, it remains in equilibrium without falling, this is achieved with four legs or more), neural networks are integrated with oscillators in order to synchronize the motor movement, the environment is not taken into account, the evolution process goes through stages and the controller for one leg is evolved. Afterwards, it is replicated for all the other legs and evolution is used again to create the coordination among legs.

Lately Bongard, Zykov and Lipson [33] at the Computational Synthesis Lab of Cornell University, investigated the behaviour of a four-legged robot that was able to infer its own structure and recover in case of damage in one of the legs. This was achieved through exploration of different self-models by performing different actions, random in the beginning and optimized locomotive sequences after many steps. This field of research can help to understand how to allow robots to create models of themselves and achieve higher levels of machine cognition.

8 Discussion

The famous American futurist Ray Kurzweil wrote in his book “The Singularity is Near” that in the 2030s mind updating will be reality. In other words it will be possible to emulate the whole human brain into a computer system. The jump between today’s intelligent agents governed by evolving artificial neural networks and a real brain is huge. Nowadays artificially intelligent robots are able to perform specific tasks in a clever manner without being directly programmed, they learn from experience. This is just the first step toward an intelligent machine.

What makes us intelligent is not only the ability to learn from the past and execute actions based on the environment stimuli together with our knowledge. It is our ability to recognize ourselves, a self-consciousness and self-awareness [4] [5] [6]. We are able to locate ourselves in space and time into the environment. Based on the information that we retrieve from the external world, we are able to understand where we are located and make spatial and temporal comparisons and reasoning. Even less evolved animals are able to do that.

Let us consider two birds on the ground and a breadcrumb few meters away from them. The closest bird will analyze the situation and see the food. Most likely, this bird will immediately react and fly there to eat the bread. The second bird, which is few meters away from the first bird and more distant from the breadcrumb will also analyze the situation. His brain will recognize the food and, based on previous experience, it will understand that he should fly there also. But at this point, the self-awareness will come in action. The second bird will place himself into the environment and understand his location compared to the position of the other bird, in respect to the bread. He will also analyze the time necessary to reach the food and he will eventually understand that he will need more time than the other bird to reach the bread. He will not fly there!

If two ANN controlled robots are instructed separately with a T-maze set-up with one reward and later placed in another environment together, again with one reward, they can be considered intelligent if and only if they show self-consciousness and self-awareness abilities. It is not enough that both will move towards the reward. This behaviour of competing for the reward could make sense only if they are placed more or less at the same distance from the reward point. Otherwise, as shown in Figure 17, robot A should reach the reward R and robot B should not move.

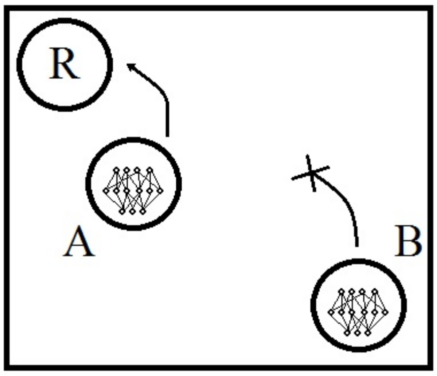

FIGURE 17: Two robots and only one reward.

9 Conclusion

In this paper, an extensive coverage of the state-of-the-art in adaptive-ANN-controlled intelligent agents has been given. In the first chapter, the analogy between biological and artificial neurons has been presented, together with a history of artificial neural networks. The second chapter is a sort of glossary of relevant terms and it serves as a basis for the formal definition of ANN and the adaptation techniques, presented in chapters 3 and 4 respectively. Adaptation refers to the ability of the network to respond to changes in the environment. It can be achieved by structure modification, by neuronal property modification or by parameter adjustment, using learning rules such as Hebbian rule, Delta rule or Backpropagation. Evolution can be also used for network optimization. Several evolutionary strategies have been presented together with a general framework for adaptive systems. Chapter 6 describes how to implement ANNs as control agents for robots and Chapter 7 illustrates more in details, with practical examples, how to integrate neural networks and evolution to obtain "intelligent" agent controlled robots capable of sensory motor coordination and other complex tasks such as walking. Finally, chapter 8 discusses the point of view of the author regarding the gap that has to be filled in order to really consider the robot's behaviour intelligent. Key words here are self-awareness and self-consciousness.

Bibliography

[1] S. Nolfi and D. Floreano. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press: Cambridge, MA. 2000. ISBN 978-0-262-64056-5

[2] K. Mainzer. Thinking in Complexity. Springer. 1994. ISBN 978-3-540-72227-4

[3] D. Floreano and C. Mattiussi Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. MIT Press: Cambridge, MA. 2008. ISBN 978-0-262-06271-8

[4] P. Husbands, O. Holland and M. Wheeler. The Mechanical Mind in History. MIT Press: Cambridge, MA. 2008. ISBN 978-0-262-08377-5

[5] W. J. Freeman. How Brains Make Up Their Minds. Columbia University Press, New York. 2001. ISBN 0-297-84257-9

[6] H. L. Dreyfus. What Computers Still Can't Do: A Critique of Artificial Reason. MIT Press: Cambridge, MA. 1992. ISBN 978-0-262-54067-4

[7] K. O. Ellefsen. Evolving the Ability to Learn in Artificial Neural Networks. IDI Doctoral Conference 2011. Trondheim. 2011

[8] J. Lemler, S. B. Harris, C. Platt and T. M. Huffman. The Arrest of Biological Time as a Bridge to Engineered Negligible Senescence. 10th Congress of The International Association of Biomedical Gerontology. Queen's College. Cambridge, England. 2003.

[9] W. Clark. Artificial Neural Networks. GEOG 6010. 2006. Slides Presentation.

[10] Xin Yao. Evolving Artificial Neural Networks. Proceedings of the IEEE vol. 98, NO. 9, September 1999

[11] S. Seoung. I am my Connectome. TEDGlobal 2010. Talk 2010

[12] J. Lehrer. Making Connections. Nature vol. 257, 29 January 2009. Macmillan Publisher Limited. 2009

[13] A. K. Jain, J. Mao and K. M. Mohiuddin. Artificial Neural Networks: A Tutorial. Theme Feature – Proceedings of IEEE Computer March 1996. IEEE 1996

[14] R. M. Palnitkar and J. Cannady. A Review of Adaptive Neural Networks. IEEE Southeast Con 2004. Theme "Engineering Connect". 2004

[15] J. A. Freeman and D. M. Skapura. Neural Networks: Algorithms, Applications, and Programming Techniques. Addison Wesley, New York. 1992

[16] R. L. Harvey. Neural Network Principles. Englewood Cliffs, NJ. Prentice All Inc. 1994

[17] D. C. Simon and T. W. Long. Adaptive Optimization of Aircraft Engine Performance Using Neural Networks. U.S. National Aeronautics and Space Administration. 1995

[18] R. Tawel. The Adaptive Neural Model: An Architecture for the Rapid Learning of Nonlinear Topological Transformations. U.S. National Aeronautics and Space Administration. 1992

[19] M. Solazzi and A. Uncini. Artificial Neural Networks with Adaptive Multidimensional Spline Activation Functions. Proceedings IEEE-INNS-ENNS International Joint Conference on Neural Networks. 2000

[20] A. Alexandridis et al. A new Algorithm for Online Structure and Parameter Adaptation of RBF Networks. Neural Networks xx. 2003

[21] N. Kasabov and G. Iliev. Hybrid System for Robust Recognition on Noisy Speech Based on Evolving Fuzzy Neural Networks and Adaptive Filtering. Proceedings IEEE-INNS-ENNS International Joint Conference on Neural Networks. 2000

[22] D. J. Chalmers. The Evolution of Learning: An Experiment in Genetic Connectionism. Proceedings of the 1990 Connectionist Models Summer School. 1990

[23] W. S. McCullock and W. Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. Bull Mathematical Biophysics, vol. 5. 1943

[24] R. Rosenblatt. Principles of Neurodynamics. Spartan Books, New York. 1962

[25] Xin Yao. Evolutionary Artificial Neural Networks. International Journal of Neural Systems, vol. 4, n. 3. 1993

[26] J. Hertz, A. Krogh and R. Palmer. Connectionist Learning Procedures. Artificial Intelligence, vol. 40, n. 1-3. 1989

[27] G. F. Miller, P. M. Todd and S. U. Hegde. Designing Neural Networks Using Genetic Algorithms. Proceedings of 3rd International Conference on Genetic Algorithms and Their Applications. Morgan Kauffman. 1989

[28] S. Haykin. Neural Networks: A Comprehensive Foundation. Macmillan, New York, NY. 1994

[29] M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry. Mit Press, Cambridge MA. 1969

[30] D. O. Hebb. The Organization of Behavior. John Wiley and Sons, New York. 1949

[31] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, 26 (3): pp. 396-407. 1996

[32] S. Nolfi. Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration. T. Gomi (Ed.), Evolutionary Robotics: From Intelligent Robots to Artificial Life. Ontario. Canada: AAI Books. 1997

[33] J. Bongard, V. Zykov and H. Lipson. Resilient Machines Through Continuous Self-Modeling. Science, Vol. 314, pp. 1118-1121. 2006