Efficient Evolution of Neural Networks

Pagliuca, Paolo

http://hdl.handle.net/10026.1/15144

http://dx.doi.org/10.24382/547
University of Plymouth

UNIVERSITY OF PLYMOUTH

EFFICIENT EVOLUTION OF NEURAL NETWORKS

by

PAOLO PAGLIUCA

A thesis submitted to the University of Plymouth in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing, Electronics and Mathematics

[In collaboration with ISTC-CNR (Institute of Cognitive Sciences and Technologies)]

September 2018

Acknowledgments

First and foremost, I would like to thank my supervisor, Stefano Nolfi, who supported me during the last four years, even when I encountered some personal problems. His passion, knowledge, advice and trust have been fundamental for me to keep on track and succeed in completing this work. He is a special person and I am proud to work with him.

I am also grateful to all my colleagues at the Institute of Cognitive Sciences and Technologies, in particular the LARAL group (in sparse order Onofrio Gigliotta, Diana Giorgini, Vito Trianni, Alessandra Vitanza, Filippo Cantucci, Jonata Tyska Carvalho, Nicola Milano, Luca Simione, Dario Albani, Tomassino Ferrauto, Gianluca Massera, Eliseo Ferrante, Giuseppe Morlino), for their contribution to my work, but especially for the funny moments spent in these years, in particular during our after lunch table-soccer matches. Their friendship is really important for me. I find it difficult to imagine a better work environment. I would thank also my second supervisor, Angelo Cangelosi, who always gave me support during these years, and the Plymouth University.

A special thank goes to Silvia Felletti, a colleague who has become one of my best friends. She constantly encouraged me when research did not progress. We shared plenty of funny moments, including table-soccer matches, lunches and dinners, evenings spent by playing table games, etc.

A particular mention is for Monica, who is a special friend. I am also grateful to other people known at the Institute for relatively short periods (Chiara, Cristina and Martina).

I am infinitely grateful to my loving family (my mother Nadia, my father Domenico, my brother Antonello, my sister Nina, my aunt Andreina and my grandmother Anna), who supported me all days and gave me all I needed to have a good life. Although sometimes I do not appreciate some aspects of their characters, they are special people and I love them.

I thank my best friends Franco, Andrea, Andrea, Ivano, Valerio and Luca for all the moments spent together during the last twenty years. Without them, I did not manage to arrive at this point of my life. We shared many important moments and we never stopped to support each other, especially when some tragic episodes happened. I would also thank some other important friends met during my University studies (in random order Simone, Valentina, Cosmo and Matteo). Moreover, I would thank my friend Silvia, a person that deserves all of my respect for what she achieved in her life.

A thank you also goes to my former colleagues Emanuele, Lara, Luca, Marco, Flavio, Marco, Marco, Giuseppe, Emanuele, Valeriano, Pierpaolo, Daniele, Luciano, Marco, Antonio and Mario. They have been very important for my personal growth, although I was not able to show them my love.

Finally, I would infinitely thank the people met during the last three years, who helped and still help me become a better person.

This work is dedicated to the loving memory of my grandmother Rosa, who gave me her infinite love till the end, my grandfather Gino, a lovable person and a hard worker, Loreto and Margherita, two special people that made me feel like a son and that I loved so much and Loredana, who has seen me grow since I was a child.

Abstract

This thesis addresses the study of evolutionary methods for the synthesis of neural network controllers. Chapter 1 introduces the research area, reviews the state of the art, discusses promising research directions, and presents the two major scientific objectives of the thesis. The first objective, which is covered in Chapter 2, is to verify the efficacy of some of the most promising neuro-evolutionary methods proposed in the literature, including two new methods that I elaborated. This has been made by designing extended version of the double-pole balancing problem, which can be used to more properly benchmark alternative algorithms, by studying the effect of critical parameters, and by conducting several series of comparative experiments. The obtained results indicate that some methods perform better with respect to all the considered criteria, i.e. performance, robustness to environmental variations and capability to scale-up to more complex problems. The second objective, which is targeted in Chapter 3, consists in the design of a new hybrid algorithm that combines evolution and learning by demonstration. The combination of these two processes is appealing since it potentially allows the adaptive agent to exploit a richer training feedback constituted by both a scalar performance objective (reinforcement signal or fitness measure) and a detailed description of a suitable behaviour (demonstration). The proposed method has been successfully evaluated on two qualitatively different robotic problems. Chapter 4 summarizes the results obtained and describes the major contributions of the thesis.

Author's declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Doctoral College Quality Sub-Committee. Work submitted for this research degree at the University of Plymouth has not formed part of any other degree either at the University of Plymouth or at another establishment.

Relevant scientific seminars and conferences were regularly attended at which this work was presented. Two articles have been accepted for publication in refereed journals.

This study was carried out in collaboration with the Istituto di Scienze e Tecnologie della Cognizione (ISTC) - Consiglio Nazionale delle Ricerche (CNR), Rome.

This thesis contains works being the result of collaborations with other researchers. The author contribution to the reported works over the total was about 80% for the work described in chapter 2 and 60% for the work described in chapter 3.

Word count for the main body of this thesis: **34130**

Publications

Paolo Pagliuca and Stefano Nolfi (2015). "Integrating learning by experience and demonstration in autonomous robots". Adaptive Behavior 23 (5), pp 300-314.
DOI: https://doi.org/10.1177/1059712315608424

Paolo Pagliuca, Nicola Milano and Stefano Nolfi (2018). "Maximizing adaptive power in neuroevolution". PloS one 13 (7).
DOI: https://doi.org/10.1371/journal.pone.0198788

Nicola Milano, Paolo Pagliuca and Stefano Nolfi (2019). "Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits". Evolutionary Intelligence 12 (1), pp 83-95.
DOI: https://doi.org/10.1007/s12065-018-00197-z

Paolo Pagliuca and Stefano Nolfi (2019). "Robust optimization through neuroevolution". PloS one 14 (3).
DOI: https://doi.org/10.1371/journal.pone.0213193

Signed:

Date:

Contents

Acknowledgments ....
Author's declaration ....
Contents ....
List of Figures ....
List of Tables ....
List of Acronyms ....

Chapter 1. Neuroevolution 2
Introduction 2
1.1. Neuroevolution 2
1.1.1. Basic algorithm 2
1.1.2. Advantages and disadvantages of neuroevolution 3
1.1.3. Applications domains 5
1.2. Research topics in neuroevolution 6
1.2.1. Fitness function 6
1.2.2. Evolutionary algorithms 10
1.2.3. Evolution and learning 13
1.3. Scientific objective of the thesis 17
Chapter 2. A Systematic Comparison of Promising Evolutionary Methods 20
Introduction 20
2.1. Method 21
2.1.1. The double-pole balancing problem 21
2.1.2. The NeuroEvolution of Augmenting Topologies (NEAT) method 26
2.1.3. Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE) 28
2.1.4. Cartesian Genetic Programming of Artificial Neural Network (CGPANN) 29
2.1.5. The Exponential Natural Evolutionary Strategy (xNES) method 30
2.1.6. The Stochastic Steady State (SSS) method 31
2.1.7. Parallel Stochastic Hill Climber (PSHC and PSHC*) 33
2.2. Results 37
2.3. Discussion 51
Chapter 3. Integrating Learning by Experience and Demonstration in Autonomous Robots 55
Introduction 55
3.1. Exploration experiment 59
3.1.1. Learning by experience 61

3.1.2. Learning by demonstration 65
3.1.3. Integrated learning by experience and by demonstration 67
3.2. Results for the exploration experiment 70
3.3. Spatial positioning with heading experiment 74
3.4. Results for the spatial positioning with heading experiment 79
3.5. Discussion 85
Chapter 4. Conclusions 87
Conclusions 87
4.1. Contribution to knowledge 89
References 91

List of Figures

Figure 2.1. The double pole balancing problem.. 22
Figure 2.2. Performance of the best agents evolved through different methods during the evolutionary process. The top and bottom pictures display that data of the standard and delayed double-pole problems, respectively. The left and right pictures display the data of the experiment carried out in the Fixed and Randomly varying initial conditions, respectively. Each curve displays the average result of the best 30 networks evolved in the 30 corresponding replications of each experiment. Data refer to the best individual of the population.................. 44
Figure 2.3. Long double-pole problem. Performance of the best agents during the course of the evolutionary process. The left and right pictures display the data of the experiment carried out in the Fixed and Randomly varying initial conditions, respectively. Each curve displays the average result of the best 30 networks evolved in the 30 corresponding replications of each experiment. Data refer to the best individual of the population... 46
Figure 3.1. Exploration Experiment. The robot and the environment. 60
Figure 3.2. Boxplots of performance obtained, in the learning by experience (E), learning by demonstration (D) and integrated learning by experience and demonstration (E&D) experimental conditions. Each boxplot displays the performance obtained by post-evaluating the best 10 robots of each replication for 500 trials in environments including 5 obstacles. Boxes represent the inter-quartile range of the data, while the horizontal lines inside the boxes mark the median values. The whiskers extend to the most extreme data points within 1.5 times the inter-quartile range from the box. For the (D) condition I only report the result obtained in the offline mode that yielded better results than the online mode. 71
Figure 3.3. Trajectories displayed by the best robots of the three experimental conditions and by the demonstrator robot during a typical trial. To ensure that the trials shown are representative, they have been selected among the trials that produced a performance similar to the average performance obtained in each condition. The blue circle indicates the position of the robot at the end of the trial. The red circles indicate the position of the obstacles. 73
Figure 3.4. Spatial positioning with heading experiment. Up: The robot and the environment. The black arrow indicates the relative position and orientation that the robot should reach (i.e. the robot should reach the target with a relative position and orientation that could enable it to push the red object toward the blue object). **Bottom:** Exemplification of how the position of both the robot and the red cylinder varies among trials. The top circle indicates the blue cylinder. The intermediate and bottom rectangles indicate the areas in which the red cylinder and the robot are placed, respectively. The exact position of the red cylinder and of the robot is randomly chosen inside the selected rectangular area. 75
Figure 3.5. The boxplots display the performance carried out by the robots at the end of the training process in the learning by experience (E), learning by demonstration (D) and integrated learning by experience and demonstration (E&D). Boxes represent the inter-quartile range of the data, while the horizontal lines inside the boxes mark the median values. The whiskers extend to the most extreme data points within 1.5 times the inter-quartile range from the box. Data obtained by post-evaluating the best 10 robots of the 10 replications of each experimental condition for 180 trials. For the D condition I only report the result obtained in the online mode that yielded better results with respect to the offline mode... 81
Figure 3.6. Average difference between the actions produced by the best robot trained in the three experimental conditions (E, D, and E&D) and the actions that would be produced by the

demonstrator robot in the same robot/environmental conditions. Data are calculated by averaging the summed square difference between the actual and desired speeds of the two wheels, normalised in the range [0.0, 1.0], during 180 trials. 82

Figure 3.7. Trajectories displayed by the best robots obtained in the learning by experience (top-right), learning by demonstration (bottom-left) and learning by experience and demonstration condition (bottom-right) in a trial in which they start from the same initial position and orientation and in which the red cylinder is placed in the same position. The top-left picture shows the trajectory produced by the demonstrator robot. The red and blue circles represent the position of red and blue cylinder. 83

List of Tables

Table 2.1. Initial states used during different trials carried out in the Fixed Initial States condition.. 24
Table 2.2. Range of the states used to set the initial state in the Randomly Varying Initial States condition.. 24
Table 2.3. Evaluations required to solve the problem averaged over 30 replications of the experiments and percentage of replications that successfully solved the problem (i.e. that found a network capable of balancing the pole until the end of the trial). In the methods with fixed topologies, I used full recurrent architectures. The data reported in the table are those obtained with the best combination of parameters: xNES (fixed positions: LearningRate 0.5, NumHiddens 0), PSHC (MutRate 50%, Stochasticity 10%, Interbreeding 10%, NumHiddens 1), CoSyNE (MutRate 90%, SubPopulations 5, NumHiddens 1), SSS (fixed positions: MutRate 50%, Stochasticity 20%, NunHiddens 1), CGPANN (MutRate 3%; NumIncomingConnections 8, NumHiddens 2), and NEAT (popSize 100, NumHiddens 1.8, see Stanley and Miikkulainen [2002] for the other parameters). In the case of NEAT, the number of hidden neurons indicates the average number of internal neurons possessed by first solution obtained in each replication. The network of the fastest replication included 1 hidden neuron. 37
Table 2.4. Performance and generalization ability of neural network controllers evolved with different methods on the double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. The data reported in the table are those obtained with the best combination of parameters: NEAT (fixed positions: Population 1000; varying positions: Population 1000, for the other parameters see Stanley and Miikkulainen, 2002), CoSyNE (fixed positions: MutRate 5%, SubPopulation 60; varying positions: MutRate 10%, SubPopulation 60), CGPANN (fixed positions: MutRate 20%, Incoming Connections 8); varying positions: MutRate 1%, Incoming Connections 4), xNES (fixed positions: LearningRate 0.2; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 7%, Stochasticity 10%; varying positions: MutRate 3%, Stochasticity 10%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%, Interbreeding 10%; varying positions: MutRate 7%, Stochasticity 0%, Interbreeding 10%). 39
Table 2.5. Performance and generalization ability of neural network controllers evolved with different methods on the delayed double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. The best performance is indicated in bold. The data reported in the table are those obtained with the best combination of parameters: NEAT (fixed positions: Population 1000; varying positions, Population 1000, for the other parameters see Stanley and Miikkulainen, 2002), CoSyNE (fixed positions: MutRate 10%, Subpopulations 60; varying positions: MutRate 5%, Subpopulations 60), CGPANN (fixed positions: MutRate 20%, Incoming Connections 8; varying positions: MutRate 1%, Incoming

Connections 5), xNES (fixed positions: LearningRate 0.2; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 3%, Stochasticity 100%; varying positions: MutRate 7%, Stochasticity 70%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%; Interbreeding10%, varying positions: MutRate 3%, Stochasticity 0%, Interbreeding 10%).................................... 40
Table 2.6. Average number of evaluations required to reach a fitness equal or greater than 0.9 for the first time. Asterisks indicate the conditions that failed to reach this threshold in more than 15 out of 30 replications.. 42
Table 2.7. Performance and generalization ability of neural network controllers evolved withthe SSS method on the standard and delayed double-pole balancing problem in the Varying Initial States experimental condition in experiment carried out with different combination of parameters. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. 43
Table 2.8. Performance and generalization ability of neural network controllers evolved with different methods on the long double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart has been set randomly. The best performances are indicated in bold. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. The data reported in the table are those obtained with the best combination of parameters: xNES (fixed positions: LearningRate 0.5; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 20%, Stochasticity 30%; varying positions: MutRate 3%, Stochasticity 0%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%, Interbreeding10%; varying positions: MutRate 3%, Stochasticity 0%, Interbreeding 10%), PHSC* (fixed positions: MutRate 7%, EliminateConnection 40%, Stochasticity 70%, Interbreeding 10%; varying positions: MutRate 1%, EliminateConnection 15%, Stochasticity 50%, Interbreeding 10%).... 45
Table 2.9. Performance and generalization ability of the agents evolved with the PSHC and the SSS method in the Fixed Initial States condition in the experiments carried out with or without stochasticity. The numbers refer to the data obtained with the best combination of parameters, i.e. with the best mutation rate in the case of the data reported in the second column, and with the best mutation rate and stochasticity level in the case of the data reported in the third column. 48
Table 2.10. Statistical difference between the generalization performance obtained in the Fixed and Randomly Varying Initial States conditions for the experiment carried with the xNES, SSS and PSHC methods. Data collected on the experiments carried with the best parameters (see Table 2.4, Table 2.5 and Table 2.8). Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. 49
Table 2.11. Performance and generalization capabilities of neural network controllers evolved with the SSS method in the Randomly Varying Initial States condition by using a variable number of trials. Data collected by using the best parameters. The numbers in square brackets

indicate the average performance obtained by post-evaluating the corresponding 30 best evolved agents for 1000 trials during which the initial state of the cart have been set randomly within the range shown in Table 2.2. Each number indicates the average performance obtained during 30 replications of the corresponding experiment. 50
Table 3.1. Performance and percentage of times the best robots turn in their preferred direction in the three experimental conditions. Some trained robots turn preferentially left, others preferentially right. Others turn both left and right, depending on the circumstances. These latter robots do not have a preferred direction, they turn left and right with similar probabilities. Top part: data for the best robot of all replications. Bottom part: average results of the 10 best robots of each replication. 74

List of Acronyms

CGPANN	Cartesian Genetic Programming of Artificial Neural Network
CoSyNE	Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE)
D	Learning by Demonstration
E	Learning by Experience
E&D	Learning by Experience and Demonstration
EA	Evolutionary Algorithm
ES	Evolutionary Strategy
ESP	Enforced SubPopulations
FARSA	Framework for Autonomous Robotics Simulation and Analysis
HESP	Hierarchical Enforced SubPopulations
HyperNEAT	Hypercube-based NEAT
NEAT	NeuroEvolution of Augmenting Topologies
NES	Natural Evolution Strategies
PSHC	Parallel Stochastic Hill Climber
SANE	Symbiotic Adaptive NeuroEvolution
SSS	Stochastic Steady State
xNES	Exponential Natural Evolution Strategies

Chapter 1. Neuroevolution

Introduction

1.1. Neuroevolution

The term neuroevolution refers to the evolution of neural networks selected for the ability to solve a given problem (Yao, 1999). More specifically, neuroevolution is a biologically-inspired machine learning technique for evolving the characteristics of a neural network (i.e., weights, topologies, neuron's transfer functions, learning rules, etc.) in order to learn a specific task (Miikkulainen, 2010). "Neuroevolution applies evolutionary algorithms to construct artificial neural networks, taking inspiration from the evolution of biological nervous systems in nature" (Lehmann and Miikkulainen, 2013).

Research in neuroevolution was primarily motivated by the attempt to reproduce, and thus better understand the evolution of biological nervous systems. In this sense, neuroevolution can be seen as a tool for investigating evolution in nature. Consequently, "neuroevolution applies abstractions of natural evolution (i.e. evolutionary algorithms) to construct abstractions of biological neural networks (i.e. artificial neural networks)" (Lehmann and Miikkulainen, 2013). From an engineering point of view, instead, neuroevolution aims at evolving artificial neural networks displaying complex behaviours and able to solve non-trivial tasks. In this respect, neuroevolution represents a practical method to synthesize networks capable of performing desired tasks.

Neuroevolution received considerable attention in the last two decades, and several approaches have been applied. They will be reviewed in detail in section 1.2

1.1.1.Basic algorithm

In the classic framework of neuroevolution, a population of genetic encodings of neural networks (the individuals) is evolved in order to find a network able to effectively perform a given task. Each encoding in the population (referred as genotype) is decoded into the corresponding controller/neural network (termed phenotype). The neural network is evaluated for the ability to perform a certain task. Depending on how well the network carries out the task, a fitness value is associated to the corresponding genotype. The fitness measures the network's ability in solving the task. After all members of the population have been evaluated, selection takes place, and a new population is created for the next generation by means of genetic operators. More specifically, the genotypes with highest fitness values are mutated and/or crossed over with each other, and the resulting offspring replace the genotypes with the lowest fitness values in the population (as in the case of natural evolution, therefore, evolving individuals are subjected to competitive selection and reproduction with variation). The whole process is repeated until a network with a sufficiently high fitness is found or a stopping criterion is met (e.g., a predefined maximum number of evaluations are exceeded).

1.1.2.Advantages and disadvantages of neuroevolution

Neuroevolution constitutes a general and effective method that can be applied to a wide range of problems. It presents several advantages with respect to alternative training methods for neural networks, like supervised learning (e.g., back-propagation algorithm, see Rumhelart, Hinton and Williams, 1986) and reinforcement learning (Sutton and Barto, 1998). First of all, neuroevolution is highly general. Unlike supervised neural network learning algorithms that require labelled input-output pairs, neuroevolution allows learning without such explicit targets, with only sparse feedback. In particular, since it does not depend on gradient information, it can be applied to problems in which this information is unavailable, too noisy, or too costly to be obtained. As described in section 1.1, by operating simply on the basis of a scalar fitness value that rates the extent to which the network solves the given problem, it can be applied to any problem. Neuroevolution is suitable for domains, like game playing, vehicle control and robotics, in which the optimal actions at each point in time are not always known. In these scenarios, only after performing a sequence of actions, it is possible to observe how well the sequence worked. Neuroevolution permits to discover neural networks displaying effective behaviours given only sparse feedback (i.e., a fitness value). Moreover, unlike other learning algorithms, neuroevolution can be applied to any type of neural network independently of the architecture and the neuron's transfer function. In addition, differently from most neural learning methods that change only the strengths of neural connections (i.e., connection weights), neuroevolution can be applied to any type of neural network and can be used to adapt all the characteristics of the network including the connection weights, the architecture/topology of the network, the transfer function of the neurons (i.e., the type of computation performed by individual neurons), and the characteristics of the system (if any) in which the network is embedded (see for example Wieland, 1991; Floreano and Mondada, 1996; Moriarty and Miikkulainen, 1996 and 1997; Gomez and Miikkulainen, 1997; Floreano and Urzelai, 2000 and 2001; Stanley and Miikkulainen, 2002; Igel, 2003; Durr, Mattiussi and Floreano, 2006; Gomez, Schmidhuber and Miikkulainen, 2008; Khan, Khan and Miller, 2010; Turner and Miller, 2013 and 2014). Neuroevolution can also be combined with learning and used to evolve the learning rules that are used to change the connection weights of the network (see for example Belew, McInerney and Schraudolph, 1991; Floreano and Mondada, 1996; Floreano and Urzelai, 2000 and 2001). Neuroevolution is also effective in partially observable domains since, unlike most Reinforcement Learning methods, does not rely on the Markov assumption. Finally, neuroevolution can be used to generate a population of diverse solutions.

Disadvantages include computational cost: neuroevolution is generally computationally more expensive than alternative methods. For learning tasks in which input-output pairs or labelled data are available, neuroevolution tend to be less effective than supervised learning algorithms. Another drawback is related to the speed of evolutionary algorithms, which may require a lot of evaluations before finding a solution for a given task. The types of solutions that will be developed by neuroevolution are hard to predict (Risi and Togelius, 2017). This might represent an advantage when the experimenter is unable to identify in details how the problem can be solved, while it might be a disadvantage in other cases. Finally, designing a good fitness function can be challenging.

1.1.3.Applications domains

Continuous control constitutes an ideal application domain for neuroevolution since controllers can be conveniently realized with neural networks and since the feedback information that can be used to drive the adaptive process is typically sparse. Indeed, in this domain elaborating metrics for measuring how well an agent is doing is relatively straightforward. On the contrary, identifying the actions that the agent should perform in any possible situation in order to carry out a certain function is typically extremely hard. As a consequence, the large majority of evolutionary robotics experiments concerning the evolution of physical robots rely on neuro-evolution. Examples of problems in this domain of application include wheeled and legged robots displaying locomotion/navigation capabilities (Nolfi, Miglino and Parisi, 1994; Floreano and Mondada 1996; Nolfi and Parisi, 1997; Stanley, Bryant and Miikkulainen, 2003; Valsalam and Miikkulainen, 2008; Bongard, 2011) or collecting objects/garbage (Calabretta, Nolfi, Parisi and Wagner, 2001), swarm of robots displaying coordinated activities (Sperati, Trianni and Nolfi, 2008; Gomes, Urbano and Christensen, 2013), humanoid robots displaying reaching and grasping capabilities (Savastano and Nolfi, 2012; Tikhanoff, Pattacini, Natale and Metta, 2013; Leitner, Frank, Forster and Schmidhuber, 2014), predator and prey robots displaying pursuing and escaping behaviours (Nolfi and Floreano, 1998; Stanley and Miikkulainen, 2004).

In some works within this area, the evolution of the agents' neural controllers has been combined with the evolution of the morphological characteristics of the agents (Cliff, Husbands and Harvey, 1993; Harvey, Husbands and Cliff, 1994; Sims, 1994; Bongard and Pfeifer, 2001; Endo, Yamasaki, Maeno and Kitano, 2002; Bongard, 2011).

Another important application domain is constituted by computer games. For a detailed review of other research about neuroevolution in games, see Risi and Togelius (2017). Problems considered include continuous problems in which the agent operates on the basis of continuous information and control actions that are continuous, and games which rather involve agents capable of choosing actions within a finite set. An example of the first category is car racing, see for example Cardamone, Loiacono and Lanzi, 2009a, 2009b, 2010a and 2010b; Muñoz, Gutierrez and Sanchis, 2009 and 2010; Koutnik, Cuccu, Schmidhuber and Gomez, 2013; Pan, You, Wang and Lu, 2017.

Examples of the second type include Chess (Fogel, Hays, Hahn and Quon, 2004), Checkers (Chellapilla and Fogel, 1999; Hughes, 2003; Schaeffer, Burch, Björnsson, Kishimoto, Müller, Lake, Lu and Sutphen, 2007) and Othello (Moriarty and Miikkulainen, 1994 and 1995; Fogel, 2001; Chung, Tan and White, 2005; Lucas and Runarsson, 2006; Runarsson and Lucas, 2014).

However, neuroevolution can be successfully applied also to classification or regression problems (Schmidhuber, Wierstra and Gomez, 2005; Chandra and Yao, 2006; Chen and Alahakoon, 2006; Schmidhuber, Wierstra, Gagliolo and Gomez, 2007; Verbancsics and Harguess, 2015; Cerecedo-Cordoba, Gonzalez Barbosa, Teran-Villanueva, Frausto-Solis and Martinez Flores, 2017; Dale, 2018).

1.2. Research topics in neuroevolution

After the first pioneering works that set up the basis of the methodology (Whitley, Starkweather and Bogart, 1990; Parisi, Cecconi and Nolfi, 1990; Belew, McInerney and Schraudolph, 1990; Yao, 1994; Ackley and Littmann, 1991), the research in neuroevolution addressed different aspects that will be reviewed in the following subsections.

1.2.1.Fitness function

As pointed out in section 1.1, neuroevolution is the application of evolutionary algorithms to neural networks. Furthermore, neuroevolution has the appealing property of requiring only sparse feedback, in the form a fitness value, to work efficiently. It is worth noting that, in order to evolve neural networks, several design choices must be taken, involving the fitness function, the evolutionary algorithm used, the type of network and how to represent the characteristics of the network. A critical issue for the success of neuroevolution is the choice of the fitness function, which has two key roles: (i) defines the goal and (ii) drives the search process (Doncieux and Mouret, 2014).

Therefore, designing an effective fitness function is a far from trivial task (Lima, Gracias, Pereira and Rosa, 1996; Nolfi and Floreano, 2001; Nelson, Barlow and Doitsidis, 2009; Doncieux, Mouret, Bredeche and Padois, 2011; Bongard, 2013; Doncieux and Mouret, 2014; Trianni, 2016).

Fitness functions have been classified according to different criteria (see Nolfi and Floreano, 2001; Nelson, Barlow and Doitsidis, 2009). Nolfi and Floreano (2001) described the different ways in which a fitness function can be designed by introducing a notion of fitness space characterized by the following dimensions: (i) the functional-behavioural dimension, that specifies whether the fitness function measures to what extent the problem has been solved versus the way in which the problem is solved; (ii) the explicit-implicit dimension, which indicates whether the fitness function rates only a single general criterion or also specifies several sub-criteria that have to be satisfied; (iii) the external-internal dimension, that indicates whether or not the information used to calculate the fitness function is accessible to the evolving agent. Nelson, Barlow and Doitsidis (2009) proposed a classification based on the level of a priori knowledge used for the definition of the fitness function. Depending on the amount of information included in the fitness function, they defined the following classes: (i) training data fitness functions, that require a great amount of prior knowledge (Nelson, Grant and Lee, 2002; Dima, Hebert and Stentz, 2004); (ii) behavioural fitness functions, i.e. task-specific hand-crafted functions evaluating several aspects of the controller's behaviour and requiring a high level of prior knowledge of the task. Typically, these functions include implicit/explicit clues about the behaviour required to solve the given task (Floreano and Mondada, 1996; Lund and Miglino, 1996); (iii) aggregate fitness functions, that require very little knowledge about the domain. In particular, they reward controllers for their ability to solve a task independently of how it is completed; (iv) competitive fitness functions, exploiting competition between members of an evolving population so that the behaviour of one controller influences the behaviour, and therefore the fitness measure, of the other(s) (see Nelson, Grant, Barlow and White, 2003; Nelson and Grant, 2006); (v) co-competitive fitness functions, typically used in situations in which two different populations compete against each other within the same environment, like in prey-predator scenarios (see Nolfi and Floreano, 1998), and the fitness of one population affects the fitness of the other one; (vi) incremental fitness functions in which the complexity of desired function is progressively increased and

(vii) environmental incremental fitness functions in which the complexity of the environment is progressively increased (see Harvey, Husbands and Cliff, 1994; Gomez and Miikkulainen, 1997; Bongard, 2011).

Doncieux and Mouret (2014) focused instead on the selective pressures that drive the evolutionary process, rather than classifying fitness functions. The authors identified two classes of selective pressures: (i) goal refiners and (ii) process helpers. Goal refiners aim to change the optimum of the fitness function, hence alter the search space (e.g., the addition of noise, see Jakobi, Husbands and Harvey, 1995; Jakobi, 1997). Conversely, process helpers alter the search process without modifying the optimum of the fitness function (e.g., behavioural diversity, see Mouret and Doncieux, 2009 and 2012). The former addresses the reality gap issue (i.e., the problem observed when transferring evolved controllers from simulation to reality, see Jakobi, Husbands and Harvey, 1995; Jakobi, 1997 and 1998; Zagal, Ruiz del Solar and Vallejos, 2004; Koos, Mouret and Doncieux, 2010 and 2013). The latter addresses the premature convergence or bootstrap problem (Nolfi, 1998; Mouret and Doncieux, 2009; Israel and Moshaiov, 2012; Silva, Duarte, Correia, Moura Oliveira and Christensen, 2016), i.e. the issue of being trapped into local optima solutions that prevent the discovery of the global optimum. Both goal refiners and process helpers mitigate the problem of crafting the fitness function. The authors also split the two categories according on whether they incorporate some knowledge on how to solve the task (task-specific) or not (task-agnostic).

Incremental evolution refers to approaches in which the neural network is evolved for the ability to first solve a simple version of the problem and then progressively more complex versions. As stated above, this can be performed by varying the fitness function and/or the environmental conditions during the course of the evolutionary process. Harvey, Husbands and Cliff (1994) introduced incremental evolution in their seminal work in which robots must develop the ability to move towards a white triangle on a dark background, but not towards a white rectangle of similar size. Gomez and Miikkulainen (1997) utilized incremental evolution to evolve controllers for both prey capture and double-pole balancing tasks. Bongard (2011) applied incremental evolution to simulated robots that undergo morphological changes during evolution.

Often fitness functions include different fitness components that are used to reward agents for the ability to achieve different goals or for the ability to obtain sub-goals that are instrumental for the achievement of the agents' general goal. For example, reaching can be instrumental for grasping and consequently agents evolved for the ability to grasp objects can receive a small reward when they manage to reach the object to be grasped and a larger reward when they succeed in grasping the object (Bianco and Nolfi, 2004; Massera, Ferrauto, Gigliotta and Nolfi, 2014). Similarly, robots evolved for the ability to clean an arena from objects might receive a smaller reward for picking up an object and a larger reward for releasing the object outside the arena (Nolfi, 1997), etc. Usually the scores gained for each sub-goal are then combined in a single scalar value by using a weighted sum. An alternative approach is multi-objective optimization. Multi-objective optimization refers to the process of trying to optimize simultaneously many objectives, most of which are typically in conflict (Fonseca and Fleming, 1995; Van Veldhuizen and Lamont, 2000; Konak, Coit and Smith, 2006). Differently from single-optimization problems, in a multi-objective optimization problem it is difficult to identify a single solution and the best strategies usually represent good compromises between the different objectives. In this domain, the Pareto optimality measure (Edgeworth, 1881; Pareto, 1896) is adopted. More specifically, a solution for the multi-objective optimization problem is Pareto optimal if "there are no other solutions which would decrease one criterion without simultaneously increasing in at least one other criterion". Put more formally, a solution for the multi-objective optimization problem is said to be Pareto optimal if and only if the two following criteria are met: (i) the solution is not worse than the other candidate solutions with respect to all of the considered objectives, and (ii) the solution is strictly better than the others in at least one objective (Deb, 2011).

Solteiro Pires, Tenreiro Machado and de Moura Oliveira (2004) employed multi-objective optimization for robot trajectory planning. Castillo and Trujillo (2005) utilized genetic algorithms for multi-objective robot path planning. A similar work is the one of Castillo, Trujillo and Melin (2006). Gong, Zhang and Zhang (2011) applied particle swarm optimization (Eberhart and Kennedy, 1995) for multi-objective robot path planning. Mouret and Doncieux (2008) combined incremental evolution and multi-objective optimization for evolving controllers for a robot that must detect a light source. Israel and Moshaiov (2012) used multi-objective optimization in a robot soccer game scenario. For a review on multi-objective optimization, see Fonseca and Fleming, 1995; Coello Coello, 1999; Van Veldhuizen and Lamont, 2000; Zitzler, Laumanns and

Bleuler, 2004; Coello Coello, 2006; Konak, Coit and Smith, 2006; Trianni and Lopez-Ibanez, 2014).

An alternative approach to the use of objective fitness functions that recently received considerable attention among the research community is the so-called novelty search (Lehman and Stanley, 2008) that consists in selecting individuals that differ with respect to previously selected ones rather than for their capability of performing the specific task. The advantage of novelty search is that it is less sensitive to the issue of getting trapped into local minima (Mouret and Doncieux, 2009; Israel and Moshaiov, 2012; Silva, Duarte, Correia, Moura Oliveira and Christensen, 2016). When the space of possible solutions is very large and/or infinite, however, the chance to discover high quality solutions by simply selecting solutions that are novel is limited. Moreover, the cost for verifying whether a solution is novel increases exponentially across generations. For a review of the critical factors for novelty search, see Kistemaker and Whiteson (2011). Novelty search has been applied successfully in different robotic domains, including maze navigation (Lehman and Stanley, 2008, 2010 and 2011; Risi, Vanderbleek, Hughes and Stanley, 2009; Risi, Hughes and Stanley, 2010; Gomes, Mariano and Christensen, 2015), artificial ant (Lehman and Stanley, 2010), foraging (Risi, Hughes and Stanley, 2010), biped locomotion (Lehman and Stanley, 2011) and swarm robotics (Gomes, Urbano and Christensen, 2013).

1.2.2. Evolutionary algorithms

Evolutionary algorithms (EAs) are a "class of direct, probabilistic search and optimization algorithms gleaned from the model of organic evolution" (Back, 1996). Evolutionary algorithms were introduced in 60s with the aim at attempting to reproduce the human brain's capability of solving problems. They are based on Darwinian's theory of evolution, which states that adaptive changes occur as result of natural selection, according to the principle of "survival of the fittest".

Most of the evolutionary algorithms developed within the evolutionary computation community have been applied to evolve neural networks. Moreover, some specially designed algorithms have been developed.

General-purpose algorithms can be grouped into three mainstreams: genetic algorithms (Holland, 1975) evolutionary strategies (Rechenberg, 1973; Schwefel, 1977; Hansen and Ostermeier, 2001; Igel, 2003; Wierstra, Schaul, Peters and Schmidhuber, 2008), and evolutionary/genetic programming algorithms (Koza, 1992; Miller and Thomson, 2000).

Genetic algorithms were firstly introduced by Holland (1975), whose aim was to study the phenomenon of adaptation as it occurs in nature and transfer it into computer programmes (Mitchell, 1998). In Holland's genetic algorithm, a population of strings with fixed length (each one representing a candidate solution to the problem) was evolved for a certain number of generations according to Darwin's theory of evolution by means of selection and genetic recombination. Selection depends on the fitness of the candidate solutions, i.e. fitter individuals are more likely to be selected for reproduction. Genetic recombination consists in combining parts of two strings in order to generate offspring strings, i.e. candidate solutions for the next generation.

Evolutionary strategies (ESs) are optimization methods belonging to the class of evolutionary algorithms, which apply mutation, recombination and selection to a population of candidate solutions in order to evolve more effective solutions (Beyer, 2007). Evolutionary strategies were simultaneously developed in middle 70s by Rechenberg (1973) and Schwefel (1977), and then extended by other authors (Hansen and Ostermeier, 2001; Igel, 2003; Wierstra, Schaul, Peters and Schmidhuber, 2008). Harvey (2001) and Whitley (2001) claimed the superiority of evolutionary strategy methods (that use small populations, steady state selection, and that rely primarily on mutations to generate variations) over genetic algorithms (that use large populations, generational selection, and that rely primarily on recombination to generate variations). This can be explained by considering that, in the context of neuroevolution, any phenotype (i.e. any network computing a given function) can be generated by a large number of alternative genotypes. Consequently, evolution can keep exploring the search space and improving the fitness of the population over the long term even when the population is genetically converged (Harvey, 2001).

Genetic programming is a technique in which computer programmes are evolved for a given number of generations/evaluations in order to solve a task. In genetic programming, computer programmes are encoded as set of genes that are modified by operators of mutation and crossover. Cartesian genetic programming (Miller and Thomson, 2000) is an extension of genetic programming whereby individuals are encoded in strings of integers representing graphs of nodes connected by links. Each node is defined by three integers specifying the inputs to the node and the function computed by the node.

As stated above, all these methods can be used and have been used to evolve neural networks. Moreover, some evolutionary methods have been designed specifically for the evolution of neural networks (see for example Nolfi, Miglino and Parisi, 1994; Husbands, Harvey, Cliff and Miller, 1994; Stanley and Miikkulainen, 2002; Durr, Mattiussi and Floreano, 2006; Khan, Khan and Miller, 2010). An advantage of these methods is that they can be used to evolve both the topology and the connection weights of the neural network.

NeuroEvolution of Augmenting Topology (NEAT) method (Stanley and Miikkulainen, 2002) represents one of the first attempts to evolve both the connection weights and the architecture of a neural network. In particular, it is characterized by the development of progressively more complex topologies from simple networks including only sensory and motor neurons. Another relevant feature concerns the utilization of innovation numbers associated to genes that permit to cross over networks with varying topologies. Finally, NEAT makes use of speciation and fitness sharing (Goldberg and Richardson, 1987) to preserve innovations. The initial population is composed of genotypes of equal length. Each gene contains four numbers encoding the ID number of the neurons that send and receive the connection, the weight of the connection, and the history marker constituted by a progressive innovation number.

Offspring are generated on the basis of the following genetic operators: (i) a crossing-over operator that uses the innovation numbers to identify the genes encoding the same structure in the two reproducing parents, (ii) an add-node genetic operator that replaces an existing connection gene with two new connection genes that connect the original input neuron to a new internal neuron with a weight of 1.0 and the new internal neuron to the original output neuron with a weight equal to the weight of the original connection, (iii) a mutate-weight genetic operator that perturbs the weight of each connection with a given probability, and (iv) an add-connection operator that adds a connection between two pre-existing neurons. Extended versions of NEAT have been later proposed by Gauci and Stanley (2007) and by Stanley, D'Ambrosio and Gauci (2009).

Another family of methods dedicated to the evolution of neural network is constituted by cooperative algorithms that operate by evolving solution components instead of full solutions. In this manner, the overall problem of finding a solution network is broken into several smaller sub-problems. Moriarty and Miikkulainen (1996 and 1997) introduced Symbiotic Adaptive NeuroEvolution (SANE), a method that evolves neurons with associated connection weights in order to form a neural network able to solve a task. Gomez and Miikkulainen (1997) extended SANE approach by dividing neurons in subpopulations with a method called Enforced Sub-Populations (ESP). Gomez and Schmidhuber (2005) extended ESP in HESP (Hierarchical ESP), a neuroevolutionary algorithm that simultaneously evolves networks at level of both neurons and full networks. Gomez, Schmidhuber and Miikkulainen (2008) proposed the Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE), a method that uses cooperative co-evolution at the level of synaptic weights. More specifically, for each connection weight, there is a separate subpopulation consisting of real valued weights. CoSyNE has been applied to the double-pole balancing task. A detailed description of CoSyNE is provided in subsection 2.1.3.

1.2.3. Evolution and learning

Another advantage of neuroevolution consists in the possibility to combine evolution and learning. As explained in Floreano, Husbands and Nolfi (2008), "Evolution and learning (or phylogenetic and ontogenetic adaptation) are two forms of biological adaptation that differ in space and time. Evolution is a process of selective reproduction and substitution based on the existence of a population of individuals displaying variability at the genetic level. Learning, instead, is a set of modifications taking place within each single individual during its own lifetime. Evolution and learning operate on different time scales. Evolution is a form of adaptation capable of capturing relatively slow environmental changes that might encompass several generations (e.g., the perceptual characteristics of food sources for a given species). Learning, instead, allows an individual to adapt to environmental modifications that are unpredictable at the generational level. Learning might include a variety of mechanisms that produce adaptive changes in an individual during its lifetime, such as physical development, neural maturation, variation of the connectivity between neurons, and synaptic plasticity. Finally, whereas evolution operates on the genotype, learning affects only the phenotype and phenotypic modifications cannot directly modify the genotype".

In general, adaptation in natural organisms typically involves multiple adaptive/learning processes. Indeed, many natural organisms adapt both phylogenetically, as a result of natural evolution, and ontogenetically, as a result of learning. Moreover, humans frequently learn by using a combination of demonstration and trial and error. "When learning to play tennis, for instance, an instructor will repeatedly demonstrate the sequence of motions that form an orthodox forehand stroke. Students subsequently imitate this behaviour, but still need hours of practice to successfully return balls to a precise location on the opponent's court" (Kober, Bagnell and Peters, 2013, p. 1257).

Research on evolution and learning has focused on two aspects: (i) analysing the benefits, in terms of performance, of combining these two adaptation techniques, and (ii) understanding the role of the interaction between evolution and learning in natural organisms (Nolfi and Floreano, 1999). In this framework, learning displays several adaptive functions: (i) it might help and guide evolution by channelling evolutionary search towards the most promising directions of the search space; (ii) it permits to evolve individuals that adapt on the fly to fast environmental variations; (iii) it might allow evolution to discover more effective solutions and sophisticated behaviours; (iv) it could permit to scale up to more complex problems involving large search spaces (Nolfi, 2000). However, learning has also costs (Mayley, 1997). In particular, learned skills strongly depend on learning experiences. Consequently, when conditions are not favourable, learning might fail to discover the required capabilities (Nolfi, 2000).

A pioneering work about the effect of learning on evolution is the one of Hinton and Nowlan (1987). The authors considered an experimental setting in which there is a one-to-one mapping from genotype to phenotype. The genotype is a string of bits, while the phenotype is a neural network. Given a genotype, a 1-bit corresponds to the presence of a particular connection in the network. Conversely, a 0-bit means that the corresponding connection is absent. In this abstract task only a specific combination of genes (i.e., a genotype with all 1-bits) gets a fitness score of 1, whereas all other genotypes receive a fitness score of 0. In this extreme case, finding a solution is highly problematic since the fitness surface looks like a flat area with a spike in correspondence of the good combination. In such a scenario, evolution and random search have the same chance to discover the solution. To study the effect of the combination of evolution and learning, the author considered a control situation in which the alleles could also assume a "*" value and learning operates by simply assigning random values to these alleles. The addition of learning changes the shape of the fitness function. More specifically, the area around the good combination is enlarged and smoothened. The collected results demonstrated how indeed the combination of learning and evolution permits to find the solution of the problem, while the utilization of learning or evolution alone fails. To date, however, this method has not been successfully applied to realistic problems, for example the evolution of robots selected for the capability to solve a problem that cannot be solved by using evolution alone. Other limitations of Hinton and Nowlan's model are: 1) there is no distinction between genotype and phenotype, 2) learning is modelled as a random process and 3) there is no distinction between learning task and evolutionary task.

The seminal work of Hinton and Nowlan has been a source of inspiration for many other works and several researchers used neuroevolution as a tool for investigating the combination of evolution and learning (Belew, McInerney and Schraudolph, 1990; Ackley and Littman, 1991; Nolfi, Elman and Parisi, 1994; Floreano and Mondada 1996; Sasaki and Tokoro, 1997; Floreano and Urzelai 2001). Some studies stated that the combination of evolution and learning is advantageous over the application of the single approaches alone (Fontanari and Meir, 1990; Ackley and Littman, 1991; Gruau and Whitley, 1993; Nolfi, Elman and Parisi, 1994; Whitley, Gordon and Mathias, 1994; Nolfi and Parisi, 1997; Nolfi and Floreano, 1999; Borenstein and Ruppin, 2003; Mery and Kawecki, 2004; Schembri, Mirolli and Baldassarre, 2007; Suzuki and Arita, 2007; Paenke, Jin and Branke, 2009; Lande, 2009; Liu and Iba, 2011; Wund, 2012; Suzuki and Arita, 2013), while others showed that learning actually decelerates evolution (Anderson, 1995; Ancel, 2000; Dopazo, Gordon, Perazzo and Risau-Gusman, 2001; Borenstein, Meilijson and Ruppin, 2006; Paenke, Sendhoff, Rowe and Fernando, 2007; Paenke, Sendhoff and Kawecki, 2009; Gajer, 2009 and 2012; Handzel, 2013; Saito, Ishihara and Kaneko, 2013). Moreover, other works focused on the analysis of benefits and limitations of plasticity/learning or on the analysis of the conditions under which learning accelerates/decelerates evolution (Mayley, 1996a, 1996b and 1997; DeWitt, Sih and Wilson, 1998; Price, Qvarnström and Irwin, 2003; Pigliucci, 2005; Wiles, Watson, Tonkes and Deacon, 2005; Paenke, Sendhoff and Kawecki, 2007). For a review of this research, see Richards (2008) and Sznajder, Sabelis, and Egas (2012).

The picture emerging from these works is that the combination of evolution and learning can be advantageous when the task/environmental conditions vary during the operation of the network (Nolfi and Parisi, 1996; Floreano and Nolfi, 1997). Nevertheless, there are not clear evidences that methods combining evolution and learning outperform methods operating on the basis of evolution only when the characteristics of the adaptive problem are stable, at least when the computational cost is kept constant. Besides, other research demonstrated how the ability to adapt to varying environmental conditions can also be obtained through other methods, namely by evolving networks with recurrent connections (Yamauchi and Beer, 1994; Tuci, Quinn and Harvey, 2002) and/or by evolving networks including regulatory mechanisms (Philippides, Husbands, Smith and O'Shea, 2005; Petrosino, Parisi and Nolfi, 2013).

An alternative approach that received considerable attention in the last two decades consists in the combination of learning by experience and learning by demonstration (Rosenstein and Barto, 2004; Judah, Roy, Fern and Dietterich, 2010; Kober, Bagnell and Peters, 2013). This approach exploits orthogonal properties from two different types of learning, i.e. imitating a human demonstration and refining a strategy through trial-and-error. Moreover, it can be easily used along with an evolutionary algorithm (i.e., evolutionary strategy or genetic algorithm) to evolve controllers (i.e., neural networks) able to display sophisticated behaviours that are similar, but not identical, to the learned strategies. Research in this area has been primarily driven by what happens in humans. Indeed, children often see their parents or other adults perform a specific behaviour and try to reproduce it. Both adults and children may learn a particular task by repeatedly executing it until a certain level of performance is reached. These two forms of learning might be used together in order to accelerate the learning process. Many researchers proposed different approaches, each one addressing the issue of whether and how learning by experience and learning by demonstration can be effectively combined. A detailed description is provided in chapter 3.

1.3. Scientific objective of the thesis

The first objective of this thesis is to verify the efficacy of some of the most promising neuro-evolutionary methods, including two new methods that I elaborated.

As in previous comparative studies (Moriarty and Miikkulainen, 1996; Gomez and Miikkulainen, 1997; Stanley and Miikkulainen, 2002; Gomez, Schmidhuber and Miikkulainen, 2008; Wierstra, Schaul, Glasmachers, Sun, Peters and Schmidhuber, 2014), I decided to use the double-pole problem (Wieland, 1991) as a testbed in consideration of the fact that it involves fundamental aspects of agent's control (e.g., situatedness, non-linearity, temporal credit assignment [Sutton, 1984]), it is intuitive and easy to understand, it requires a low computational cost, and it has become an universally recognized benchmark in the area of continuous control. However, given that the classic version of this problem is too limited to study the evolution of solutions that are robust with respect to environmental variations and to study the capacity of alternative methods to scale-up to more complex problems, I designed and used also three complexified versions of the problem that involve the capability to balance the pole from different initial conditions, the capability to solve a complexified version of the problem in which the response of the sensors include a time delay, the capability to solve a harder version of the problem in which the length and the mass of the second pole are increased.

The evolutionary methods chosen for the comparisons are: NeuroEvolution of Augmenting Topologies (NEAT, see Stanley and Miikkulainen, 2002), Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE, see Gomez, Schmidhuber and Miikkulainen, 2008), Cartesian Genetic Programming of Artificial Neural Network (CGPANN, see Miller and Thomson, 2000; Khan, Khan and Miller, 2010; Khan, Armad, Khan, and Miller, 2013; Turner and Miller, 2014), Exponential Natural Evolution Strategies (xNES, see Wierstra, Schaul, Peters and Schmidhuber, 2008; Glasmachers, Schaul, Sun, Wierstra and Schmidhuber, 2010; Wierstra, Glasmachers, Schaul, Sun and Schmidhuber, 2014). In addition to these methods, I tested a $(\mu + \mu)$ evolutionary strategy named Stochastic Steady State (SSS) which includes a parameter that can be used to regulate the selective pressure and a more complex evolutionary strategy named Parallel Stochastic Hill Climber (PSHC) which is constituted by a combination of a (μ + μ) and a (1 + 1) evolutionary strategy.

The results indicate that, in the extended version of the problem, the best results are obtained by using relatively simple evolutionary strategies and extended versions of evolutionary strategy that estimate the local gradient and use this information to direct the evolutionary search toward the most promising direction. Widely used methods such as NEAT (Stanley and Miikkulainen, 2002) are not really competitive with the best methods identified by my analysis with respect to all criteria considered (i.e. robustness to environmental variations, speed, and capability to scale up to more complex problems).

Moreover, the analysis performed indicates that the methods proposed to date that permit to adapt also the topology of the network have a lower adaptive power than the methods that operate with fixed topology, at least in the considered problem domain.

This study is reported in Chapter 2.

The second objective of the thesis has been that to investigate the possibility to combine complementary evolutionary and learning processes. Namely, a standard evolutionary process that performs a form of adaption based on trials and errors and a learning by demonstration that represents a form of supervised learning.

From a machine learning perspective, the combined use of learning by demonstration (Argall, Chernova, Veloso, and Browning, 2009; Billard, Callinon, Dillmann, and Schaal, 2008) and learning by experience (Kober, Bagnell and Peters, 2013; Nolfi and Floreano, 2000; Sutton and Barto, 1998) provides important potential advantages. Indeed, it potentially allows the adaptive agent to exploit a richer training feedback constituted by both a scalar performance objective (reinforcement signal or fitness measure) and a detailed description of a suitable behaviour (demonstration). Moreover, it potentially permits to combine the complementary strengths of the two different learning modes. Indeed, the possibility to observe proper behaviours frees the agent learning by demonstration from the need to find out the behavioural strategy that can be used to solve the task and, consequently, narrows down the goal of the learning process only to the discovery of how the demonstrated behaviour can be re-produced.

To investigate this issue, I devised a new algorithm that operates by estimating the local gradient of the current candidate solution with respect to an objective performance measure and by exploring preferentially variations that reduce the differences between the robot and the demonstrated behaviour.

The results obtained on two qualitatively different tasks (a robotic exploration problem and a robotic navigation problem) demonstrate how the algorithm is able to synthesize effective solutions. Such solutions are qualitatively similar to the demonstrated behaviour with respect to the characteristics that are functionally appropriate, while deviate from the demonstration with respect to other characteristics.

This study is reported in Chapter 3.

Finally, in Chapter 4 I draw my conclusions.

Chapter 2. A Systematic Comparison of Promising Evolutionary Methods

Introduction

After the pioneering research carried out in the early 90s, research in neuroevolution kept expanding over the years (for reviews see Yao, 1999; Floreano, Dürr, and Mattiussi, 2008; Lehman and Miikkulainen, 2013). This has led to the development of many alternative methods that I briefly reviewed in the first chapter. The analysis and the data reported in the literature to date, however, do not enable to evaluate in an objective manner which are the most effective methods and which are the strengths and weaknesses of alternative methods. Progressing the understanding of these aspects is essential for successfully applying this methodology to concrete problems and for developing better methods.

In this chapter I carry on a systematic comparison of the adaptive power of some of the most promising methods described in the literature including two new methods designed by myself and by my colleagues. With the term adaptive power I mean the ability to discover solutions that are robust to variations of the environment and the ability to scale-up to more complex versions of the problem. This is made by: (i) using in a consistent manner a widely recognized benchmark problem known as the double-pole balancing problem (Wieland, 1991), (ii) identifying and using extended versions of the problem that highlight differences in both the performance and the ability to scale up to harder problems, and (iii) verifying the possibility to evolve agents able to effectively operate in varying environmental conditions.

The results obtained indicate that widely used methods such as NEAT (Stanley and Miikkulainen, 2002) are not really competitive with the best methods identified by this analysis with respect to all considered criteria (i.e. robustness to environmental variations, speed, and capability to scale up to more complex problems).

Moreover, the performed analysis indicates that the methods proposed to date that permit to adapt also the topology of the network have a lower adaptive power than the methods that operate with fixed topology, at least in the considered problem domain.

Finally, I demonstrate how environmental variations and the addition of a moderate level of noise in the fitness function facilitate the evolution of better solutions.

2.1. Method

In this section the classic double-pole balancing problem and the complexified versions of the problem used to carry out this analysis are presented. Then, there is a description of the six neuroevolutionary methods that have been compared, namely: NeuroEvolution of Augmenting Topologies (NEAT, see Stanley and Miikkulainen, 2002), Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE, see Gomez, Schmidhuber and Miikkulainen, 2008), Cartesian Genetic Programming of Artificial Neural Network (CGPANN, see Miller and Thomson, 2000; Khan, Khan and Miller, 2010; Khan, Armad, Khan, and Miller, 2013; Turner and Miller, 2014), Exponential Natural Evolutionary Strategy (xNES, see Wierstra, Schaul, Peters and Schmidhuber, 2008; Glasmachers, Schaul, Sun, Wierstra and Schmidhuber, 2010; Wierstra, Glasmachers, Schaul, Sun and Schmidhuber, 2014), and two novel developed methods. The former four methods have been selected based on results of previous comparative analyses.

2.1.1. The double-pole balancing problem

The pole balancing problem, introduced by Wieland (1991), consists in controlling a mobile cart with one or two poles attached through passive hinge joints on the top of the cart for the ability to keep the poles balanced (Figure 2.1).

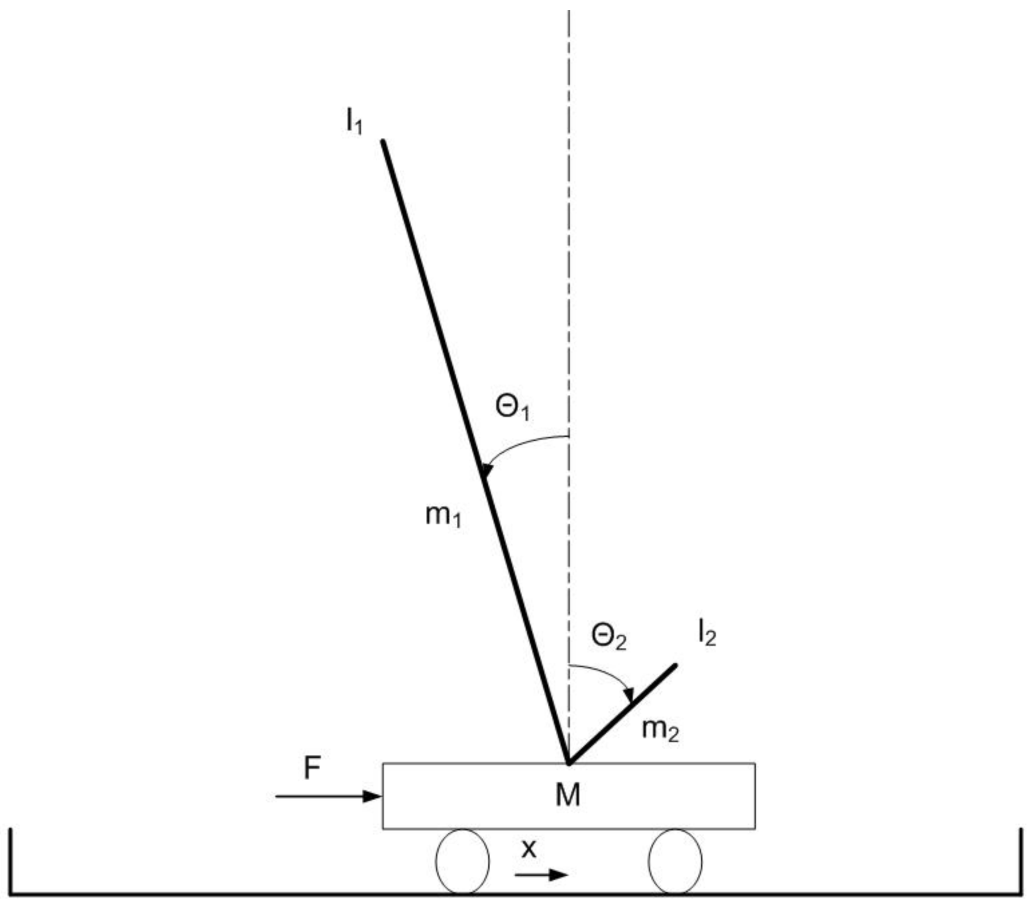

Figure 2.1. The double pole balancing problem

The cart has a mass of 1 Kg. The long pole and the short pole have a mass of 0.5 and 0.05 Kg and a length of 1.0 and 0.1 m, respectively. The cart can move along one dimension within a track of 4.8 m. In this thesis only the non-Markovian version of the problem has been considered, in which the cart is provided with three sensors encoding the current position of the cart on the track (x), and the current angle of the two poles (θ_1 and θ_2). The motor controls the force applied to the cart along the x axis. The goal is to control the force applied to the cart so to maintain the angle of the poles and the position of the cart within a viable range (details provided below).

The following equations (Wieland, 1991) are used to compute: the effective mass of the poles (1), the acceleration of the poles (2), the acceleration of the cart (3), the effective force on each pole (4), the next angle of poles (5), the next velocity of the poles (6), the next position of the cart (7), and the next velocity of the cart (8), respectively:

$$\widehat{m}_i = m_i\left(1 - \frac{3}{4}cos^2\theta_i\right) \tag{1}$$

$$\ddot{\theta}_i =)\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{2}$$

$$\ddot{x} = \frac{F + \sum_{i=0}^{N}\widehat{F}_i}{M + \sum_{i=0}^{N}\widehat{m}_i} \tag{3}$$

$$\widehat{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4}m_i\cos\theta_i\left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\dot{\theta}_i\right) \tag{4}$$

$$x[t+1] = x[t] + \tau\dot{x}[t] \tag{5}$$

$$\dot{x}[t+1] = \dot{x}[t] + \tau\ddot{x}[t] \tag{6}$$

$$\theta[t+1] = \theta[t] + \tau\dot{\theta}[t] \tag{7}$$

$$\dot{\theta}[t+1] = \dot{\theta}[t] + \tau\ddot{\theta}[t] \tag{8}$$

where x is the position of the cart on the track that varies in the range [-2.4, 2.4] m, $\dot{x}$ is the velocity of the cart, θ_1 and θ_2 are the angular positions of the poles in rad, $\dot{\theta}_1$ and $\dot{\theta}_2$ are the angular velocities of the poles in rad/s. The dynamics of the system was simulated by using the Runge-Kutta fourth-order method and a step size of 0.01 s.

The controller of the agent is constituted by a neural network with three sensory neurons and one motor neuron. The sensory neurons encode the position of the cart (x), and the angular positions of the two poles (θ_1 and θ_2). The state of the x, θ_1 and θ_2 sensors are normalized in the [-0.5, 0.5] m, [-$\frac{5\pi}{13.5}$, $\frac{5\pi}{13.5}$] rad and [-$\frac{5\pi}{13.5}$, $\frac{5\pi}{13.5}$] rad ranges, respectively. The activation state of the motor neuron is normalized in the range [-10.0, 10.0] N and is used to set the force applied to the cart. The state of the sensors, the activation of the neural network, the force applied to the cart, and the position and velocity of the cart and of the poles are updated every 0.02 s.

To promote the evolution of solutions that are robust with respect to the initial position and velocity of the cart and of the poles, I evaluated each controller for 8 trials that varied with respect to the initial state of the system. In a first experimental condition (Fixed Initial States condition) I used the initial states reported in Table 2.1. In a second experimental condition (Randomly Varying Initial States condition) the initial states were set randomly during each trial in the range described in Table 2.2.

Trial	x	$\dot{x}$	θ_1	θ_2	$\dot{\theta}_1$	$\dot{\theta}_2$
1	-1.944	0	0	0	0	0
2	1.944	0	0	0	0	0
3	0	-1.215	0	0	0	0
4	0	1.215	0	0	0	0
5	0	0	-0.10472	0	0	0
6	0	0	0.10472	0	0	0
7	0	0	0	-0.135088	0	0
8	0	0	0	0.135088	0	0

Table 2.1. Initial states used during different trials carried out in the Fixed Initial States condition.

	min	max
x	-1.944	1.944
$\dot{x}$	-1.215	1.215
θ_1	-0.10472	0.10472
θ_2	-0.135088	0.135088
$\dot{\theta}_1$	-0.10472	0.10472
$\dot{\theta}_2$	-0.135088	0.135088

Table 2.2. Range of the states used to set the initial state in the Randomly Varying Initial States condition.

Trials terminate after 1000 steps or when the angular position of one of the two poles exceeded the range [$-\frac{\pi}{5}, \frac{\pi}{5}$] rad or the position of the cart exceed the range [-2.4, 2.4] m. In the case of the classic double-pole problem, trials terminate after 100,000 steps.

The fitness of the agent corresponds to the fraction of time steps in which the agent maintains the cart and the poles within the allowed position and orientation ranges and is calculated on the basis of the following equations:

$$f_i = \frac{t}{1000} \tag{9}$$

$$F = \frac{\sum_{i=1}^{8} f_i}{8} \tag{10}$$

where *t* is the time step in which the cart or the pole exceeded the allowed range or 1000 in case they are maintained in the range until the end of the trial, f_i is the fitness of a trial, and F is the total fitness.

In the majority of the previous studies, the networks were evaluated for a single trial. I decided to use multiple trials since this forces the evolutionary process to find solutions that are robust with respect to variations of the initial state. Furthermore, some of the previous studies were carried out by using the fitness function devised by Gruau, Whitley, and Pyeatt (1996), usually referred with the term *damping fitness*, which includes an additional fitness component that is maximized by minimizing the value of x, $\dot{x}$, θ_1, and $\dot{\theta}_1$. This additional component has been introduced to facilitate the evolution of networks capable of estimating the current velocity of the cart on the basis of the available sensor information. I did not use this additional fitness component since I did not want to reduce the complexity of the problem. Some authors claimed that this additional component prevents the discovery of brittle solutions exploiting rapid oscillations of the cart. This type of solutions, however, never occurs in experiments in which the agents are evaluated for multiple trials that vary with respect to the initial state of the cart (see S1-S6 figures at http://laral.istc.cnr.it/res/neuroevolution/). Finally, in previous studies, the generalization ability of the evolved networks have been tested by post-evaluating them for 625 trials during which the initial state of the x, $\dot{x}$, θ_1 and $\dot{\theta}_1$variables was varied systematically within the following values (-1.944, -1.08, 0.0, 1.08, 1.944) m, (-1.215, -0.675, 0.0, 0.675, 1.215) m/s, (0.056548, -0.031416, 0.0, 0.031416, 0.056548) rad, (-0.135088, -0.075049, 0.0, 0.075049, 0.135088) rad/s, respectively. The state of θ_2 and $\dot{\theta}_2$ variables was initialized to 0.0. I rather decided to post-evaluate the network on 1000 trials with initial state randomly generated in the interval described in Table 2.2. This permits to verify the generalization ability of the network in a wider range of conditions.

Finally, in order to increase the complexity of the problem even further, I also used two complexified versions of the problem. In the delayed double-pole balancing problem, the agent needs to determine the state of the motor on the basis of the state that the cart had 0.02 s before (i.e., I take into account the fact that, in hardware, sensors would take time to extract information from the environment). In the long double-pole balancing problem, instead, the length and the mass of the second pole is set to 0.5 m and 0.25 Kg, respectively, instead than to 0.1m and 0.05 Kg. The elongation of the short pole makes the problem significantly more challenging. Indeed, as pointed out by Wieland (1991), the longer the second pole is, the higher the complexity of the problem becomes.

In the next subsections I describe the neuroevolutionary methods that I tested. Some of the methods operate on fixed network topologies while others permit to evolve both the connection weights and the architecture of the neural controller. In all cases the neural network controller includes 3 sensory neurons and 1 motor neuron, as described above.

2.1.2. The NeuroEvolution of Augmenting Topologies (NEAT) method

Neuroevolution of augmenting topologies (NEAT) is a method that permits the evolution of both the weights and the topology of the neural network (Stanley and

Miikkulainen, 2002). As described in section 1.2.2, it is characterized by (i) the development of progressively more complex topologies from simple networks including only sensory and motor neurons, (ii) the utilization of innovation numbers associated to genes that permit to cross over networks with varying topologies, and (iii) the utilization of speciation and fitness sharing (Goldberg and Richardson, 1987) to preserve innovations.

The initial population is composed of a vector of μ genotypes of equal length. In the case of the double-pole problem each genotype includes four genes that encode four corresponding connections from the three sensory neurons and the bias to the motor neuron.

The description of the operation of the algorithm (i.e., the information encoded in the genes, the genetic operators, etc.) is provided in section 1.2.2. For more details, see Stanley and Miikkulainen (2002). The authors later proposed new extended methods such as HyperNEAT (Stanley, D'Ambrosio and Gauci, 2009; D'Ambrosio, Gauci, Stanley, 2014) that, however, is targeted to the evolution of large neural networks and thus has not been tested on the double-pole balancing problem.

NEAT includes many parameters: crossing-over rate, add-neuron rate, add-connection rate, perturbe-weight rate, interspecies migration rate, etc. (see Stanley and Miikkulainen, 2002). The data reported in this thesis have been collected by using the author's source code freely available from http://nn.cs.utexas.edu/keyword?neat-c and the parameters used from the authors to solve the double-pole balancing problem. The experiments were repeated by varying the size of the population in the range [20, 100, 500, 1000] and the add-neuron rate in the range [1%, 10%]. The modifications introduced in the source code to evaluate the evolving agents for 8 trials on the basis of the fitness function described in subsection 2.1.1 are described in http://laral.istc.cnr.it/res/neuroevolution/NEATmodifications.txt.

2.1.3. Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE)

The neural evolution through cooperatively coevolved synapses (CoSyNE) is a method that operates by evolving solution components instead of full solutions. More precisely, instead of a population of complete neural networks, a population of network fragments is evolved (Gomez, Schmidhuber and Miikkulainen, 2008). Each fragment (connection weight) is evaluated as one part of a full network. The fitness of a fragment indicates how well it cooperates with other fragments of the solutions it contributes to compose.

The method operates by using θ sub-populations each containing m real numbers, where θ is equal to the number of connection weights and biases of the neural network and m is the number of alternative candidate solutions for each weight or bias. The population therefore consists of a matrix with θ rows and m columns, where each row corresponds to a sub-population. During each generation the algorithm: (i) generates m complete networks on the basis of the value contained in each column of the matrix, (ii) evaluates the networks, (iii) uses the top quarter networks to generate a pool of offspring that are sorted by fitness and used to replace the least fit weights of the corresponding sub-populations, and (iv) permutes the positions of the weight values in each sub-population so to ensure that the complete networks formed during the next generation are constituted by different combination of weights. For more details, see Gomez, Schmidhuber and Miikkulainen (2008).

The data reported in this thesis have been collected by using the authors' source code freely available from http://www.idsia.ch/~tino/CoSyNE-1.2.tar.gz (Gomez, Schmidhuber and Miikkulainen, 2008). I systematically varied the mutation rate and the number of sub-populations in the following ranges [1%, 2%, 5%, 10%, 20%, 40%] and [15, 30, 60], respectively. The modifications of the source code introduced to evaluate evolving agents for 8 trials on the basis of the fitness function described in Section 2.1.1 are described on http://laral.istc.cnr.it/res/neuroevolution/CoSyNEmodifications.txt.

2.1.4. Cartesian Genetic Programming of Artificial Neural Network (CGPANN)

Cartesian Genetic Programming is an evolutionary method developed by Miller and Thomson (2000) which has been widely used for the evolution of digital circuits and that has been recently applied also to the evolution of neural networks (see Khan, Khan and Miller 2010; Khan, Armad, Khan, and Miller, 2013; Turner and Miller, 2014). It is a graph-based form of Genetic Programming that can be used to evolve the weights and the topology of neural networks.

The genotype of individuals is constituted by a list of blocks encoding the property of the internal neurons of the network. More specifically, each block encodes, with integer numbers, the ID index of the input and/or internal neurons from which the current neuron receives connections and, with real numbers, the corresponding connection weight. Moreover, the genotype includes additional blocks that encode the ID indexes of the internal neurons that are used to produce the output of the network. The length of the genotype and consequently the number of neurons and the number of incoming connections per neuron is fixed. In the initial population the integer and the real numbers contained into blocks are generated randomly within the appropriate range.

The evolutionary process is realized on the basis of $(1 + \lambda)$ evolutionary strategy that uses a population composed of a single individual. During each generation the algorithm: generates λ offspring, evaluates the parent and the offspring, and selects the best individual between the parent and the offspring as the new parent.

Offspring are generated by creating a copy of the genotype of the parent and by subjecting each number contained in each block to mutations with a *MutRate* probability. Mutations are performed by replacing the real numbers, encoding the connection weights, with a new randomly generated real number on the same range or by replacing integer numbers, encoding the index of neurons projecting and receiving connections, with a new integer number generated randomly in the appropriate range. The *RecurrentConnectionsProbability* parameter is used to determine the probability with which the index of the incoming connection is replaced with a number selected in the range that includes either sensory neurons or internal neurons.

As suggested by the authors (Khan, Armad, Khan, and Miller, 2013; Turner and Miller, 2014), the number of offspring was set to 4 and the probability to generate recurrent connections was set to 20%. The number of internal neurons was set to 10. The mutation rate was varied systematically within the interval [1%, 3%, 5%, 7%, 10% and 20%]. The number of incoming connections was varied within the interval [4-12].

In one of their studies, the authors have applied this method to the double-pole balancing task (see Khan, Armad, Khan, and Miller, 2013). In this work, however, they introduced a series of modifications that facilitate the problem. More specifically: (i) they added a new input unit that provides a function of the state of the motor neuron at time *t-1*, (ii) they used a motor neuron that encodes only two alternative torque forces (i.e. -10 N or 10 N) instead of a force varying continuously within these two values, and (iii) they used a restricted version of the algorithm that is able to synthesize feed-forward network topologies only. For these reasons I decided to use the CGPANN model described in Turner and Miller (2014) that allows the evolution of recurrent neural network and I maintained the characteristics of the problem consistent with those used in other studies.

The experiments performed with this method can be replicated by downloading and installing FARSA (Massera, Ferrauto, Gigliotta and Nolfi, 2013, 2014) from "https://sourceforge.net/projects/farsa/" and the http://laral.istc.cnr.it/res/neuroevolution/CGPANN.zip experimental plugin.

2.1.5. The Exponential Natural Evolution Strategies (xNES) method

The Exponential Natural Evolution Strategies (xNES) was proposed by Wierstra, Schaul, Peters and Schmidhuber (2008) as an extension of the Evolutionary Strategy method originally proposed by Rechenberg (1973) and Schwefel (1977), see also Igel (2003). xNES operates on the basis of a single candidate solution, which consists of a vector of θ real numbers, and on the basis of a θ x θ co-variance matrix encoding the correlation between the variation of the fitness and the variation of the θ parameters. The genes of the candidate solutions are initialized randomly. The co-variance matrix is initialized with all zero values.

During each generation the algorithm: (i) generates the variation vectors that are constituted by λ vectors of θ real numbers generated randomly with a Gaussian distribution, (ii) generates λ varied candidate solutions that are obtained by adding to the candidate solution the variation vectors multiplied by the exponential of the co-variance matrix, (iii) evaluates the varied candidate solutions, (iv) estimates the local gradient of the candidate solutions on the basis of the fitness of the varied candidate solutions and uses the gradient to update the covariance matrix, and (v) modifies the current candidate solution in the direction of the estimated local gradient for a given length or learning rate. The number of candidate solutions is a constant proportional to the number of parameters to be adapted and is set to 19 in the case of the presented experiments which involve 151 parameters. For more details, see Wierstra, Schaul, Peters and Schmidhuber (2008).

The only parameter of the method is the learning rate that, according to the authors, should be set to 1.0 (see Wierstra, Schaul, Peters and Schmidhuber, 2008). By systematically varying the learning rate with the following values [0.1, 0.2, 0.25, 0.5, 1.0], however, I observed that the best results are obtained with values included in the following range [0.1, 0.5], see below.

The experiments performed with this method can be replicated by downloading and installing FARSA from “https://sourceforge.net/projects/farsa/” and the http://laral.istc.cnr.it/res/neuroevolution/xNES.zip experimental plugin.

2.1.6. The Stochastic Steady State (SSS) method

The Stochastic Steady State (SSS) is a (μ + μ) evolutionary strategy (Rechenberg, 1973) that operates on the basis of populations formed by μ parents. During each generation each parent generates one offspring, the parent and the offspring are evaluated, and the best μ individuals are selected as new parents (see the pseudo-code below). It is a method that I developed that belongs to the class of methods proposed by Harvey (2001) and Whitley (2001). The novelty with respect to previous related methods consists in the introduction of the *Stochasticity* parameter that permits to regulate the selective pressure. This is obtained by adding to the fitness of individuals a value randomly selected in the range [-*Stochasticity*MaxFitness*, *Stochasticity*MaxFitness*] with a uniform distribution. When this parameter is set to 0.0, only the best μ individuals are allowed to reproduce. The higher the value of the parameter is, the higher the probability that other individuals reproduce is. For a discussion of evolutionary optimization in uncertain environments and in the presence of a noisy fitness function, see Jin and Branke (2005).

SSS Method:

```
1: NEvaluations<- 0
// the genotype of the parents of the first generation is initialized randomly
2: for p <- 0 to NParents do
3:    for g <- 0 to NGenes do
4:       genome[p][g] <- rand(-8.0, 8.0)
5:    Fitness[p] <- evaluate (p)
6:    NEvaluations <- NEvaluations + NTrials
   // evolution is continued until a maximum number of evaluation trials is performed
7: while (NEvaluations < MaxEvaluations) do
8:    for p <- 0 to NParents do
         // in the randomly varying experimental condition parents are re-evaluated
9:       if (RandomlyVaryingInitialCondition) then
10:          Fitness[p] <- evaluate (p)
11:          NEvaluations <- NEvaluations +NTrials
12:      genome[p + NParents] <- genome[p]        // create a copy of parents' genome
13:      mutate-genome(p + NParents)               // mutate the genotype of the offspring
14:      Fitness[p + Nparents] <- evaluate[p + NParents]
15:      NEvaluations <- NEvaluations + NTrials
         // noise ensures that parents are replaced by less fit offspring with a low probability
16:     NoiseFitness[p] <- Fitness[p] * (1.0 + rand(-Stochasticity * MaxFit, Stochasticity * MaxFit))
17:      NoiseFitness[p + NParents] <- Fitness[p + NParents] * (1.0 + rand(-Stochasticity * MaxFit, Stochasticity * MaxFit))
        // the best among parents and offspring become the new parents
18:   rank genome[NParents * 2] for NoiseFitness[NParents* 2]
```

In the experiment reported in this thesis the connection weights are evolved and the topology of the network is kept fixed. The initial population is encoded in a matrix of μ x θ real numbers that are initialized randomly with a uniform distribution in the range [-8.0, 8.0], where μ corresponds to the number of parents and θ corresponds to the total number of weights and biases. Offspring are generated by creating a copy of the genotype of the parent and by subjecting each gene to mutation with a *MutRate* probability. Mutations are performed by replacing or perturbing a gene value with equal probability. Replacements consist in substituting the gene with a new real number randomly generated within the range [-8.0, 8.0] with a uniform distribution. Perturbations are performed by adding to the current value of a gene a real number randomly selected with a Gaussian distribution with a standard deviation of 1.0 and a mean of 0.0. Values outside the [-8.0, 8.0] range after perturbations are truncated in the range.

This method requires setting two parameters: *MutRate* and *Stochasticity*. To identify the optimal values of the parameters I systematically varied *MutRate* in the range [1%, 3%, 5%, 7%, 10%, 15% and 20%] and *Stochasticity* in the range [0%, 10%, 20%, 30%, 40%, 50%, 70%, and 100%]. Population size was set to 20. To enhance the ability of the method to refine the evolved candidate solutions, I reduced the mutation rate to 1% and I set the Stochasticity to 0% during the last 1/20 period of the evolutionary process.

The experiments performed with this method can be replicated by downloading and installing FARSA from “https://sourceforge.net/projects/farsa/” and the http://laral.istc.cnr.it/res/neuroevolution/SSS.zip experimental plugin.

2.1.7. Parallel Stochastic Hill Climber (PSHC and PSHC*)

The Parallel Stochastic Hill Climber (PSHC) is a new method that I developed that consists of a combination of a (μ + μ) and a (1 + 1) evolutionary strategy. The novelty of this method consists in the utilization of a stochastic hill climber method operating on single individuals that maximizes exploration at the risk of retaining maladaptive variations combined with an evolutionary strategy operating on a population of stochastically hill climbing individuals which ensures the retention of acquired adaptive traits.

The initial population is composed by a matrix of μ x θ real numbers that are initialized randomly with a uniform distribution in the range [-8.0, 8.0], where μ corresponds to the number of parents and θ corresponds to the total number of weights and biases. Varied individuals are generated by creating a copy of the genotype of the original individual and by subjecting each gene to mutation with a *MutRate* probability. Mutations are performed by replacing or perturbing a real number with equal probability. Replacements consist in substituting the gene with a new real number randomly generated within the range [-8.0, 8.0] with a uniform distribution. Perturbations are performed by adding to the gene a real number randomly selected with a Gaussian distribution with a standard deviation of 1.0 and a mean of 0.0. Values outside the [-8.0, 8.0] range after perturbations are truncated in the range.

Each generation involves an adaptation phase, during which each individual is adapted for a certain number of *Variations* through a (1 + 1) evolutionary strategy and a selection phase in which part of the adapted individuals are used to compose the new population.

During the adaptation phase, for each individual of the population, a sequence of N varied individuals is created and evaluated during N corresponding variation steps. The first varied individual is generated by creating a mutated copy of the original individual. The other varied individuals are generated by creating a mutated copy of the last varied individual that matched or outperformed the original individual (or by creating a mutated copy of the original individual when none of the previous varied individuals matched or outperformed the original individual). During this phase, the fitness of the individuals is perturbed through the addition of a value randomly selected in the range [-*Stochasticity*MaxFitness*, *Stochasticity*MaxFitness*] with a uniform distribution.

During the selection phase, for each individual of the population, the best of its variations is selected for the new population. Moreover, the best selected variation of all individuals is used to replace the worst selected variation with an *Interbreeding* probability. Contrary to the adaptation phase, the selection phase is performed on the basis of the actual fitness (i.e. the fitness without the addition of noise). The pseudo-code of the method is included below.

PSHC Method:

```
1:  NEvaluations<- 0
     // the genotype of the parents of the first generation is initialized randomly
2:  for p <- 0 to NParents do
3:     for g <- 0 to NGenes do
4:         Genome[p][g] <- rand(-8.0, 8.0)
5:     Fitness[p] <- evaluate (p)
6:     NEvaluations <- NEvaluations + NTrials
     // evolution is continued until a maximum number of evaluation trials is performed
7:  while (NEvaluations < MaxEvaluations) do
8:     for p <- 0 to NParents do
          // each individual is adapted through a (1 + 1) ES
9:        VariedGenome[0] <- Genome[p]
10:       VFitness[0] <- evaluate (VariedGenome[0])
11:       NEvaluations <- NEvaluations + NTrials
12:       for v <- 0 to NVariations do
13:          VariedGenome[v + 1] <- VariedGenome[v]
14:          mutate-genome(VariedGenome[v + 1]);
15:          VFitness[v + 1] <- evaluate(VariedGenome[v + 1])
16:          NEvaluations<- NEvaluations + NTrials
             // maladaptive variations are discarded
             // noise ensure that occasionally maladaptive variations are retained
17:          if (Vfitness[v + 1] * (1.0 + rand(-Stochasticity * MaxFit, Stochasticity * MaxFit)) < VFitness[v] * (1.0 + rand(-Stochasticity * MaxFit, Stochasticity * MaxFit)) then
18:             VariedGenome[v + 1] <- VariedGenome[v]
          // the best varied genotype is selected
20:       rank VariedGenotype[NVariations] for fitness[NVariations]
          // the original genotype is replaced with its best variation
21:       Genome[p] <- VariedGenotype[p]
```

```
22:        Fitness[p] <- VFitness[p]
           // the worse individual of the population is occasionally replaced with the best individual
23:    **rank** Genome[NParents] **for** Fitness[NParents]
24:     **if** (rand(0.0,1.0) > Interbreeding) **and** (Fitness[0] >= Fitness[Nparents - 1]) **then**
25:       Genome[NParents - 1] <- Genome[0]
```

The PSHC* method is a variation of the PSHC method that permits to evolve also the topology of the network. The difference only concerns the mutation operator. In the case of the PSHC* method mutations are performed by setting connection weights to a null value (i.e., by eliminating weights) with *EliminateConnection* probability and by replacing or perturbing the connection weight as usual with the remaining probability. Mutations of connections with null weight value produce the creation of new connections or the re-establishment of previously eliminated connections.

The PSHC method requires setting three parameters: *MutRate*, *Stochasticity* and *Interbreeding*. To identify the optimal values of the parameters I systematically varied *MutRate* in the range [1%, 3%, 5%, 7%, and 10%] and *Stochasticity* in the range [0%, 10%, 20%, 30%, 40%, 50%, 70%, and 100%]. The *Interbreeding* parameter was set to 10%.

To enhance the ability of the method to refine the evolved candidate solutions, I reduced the mutation rate to 1% and I set the Stochasticity to 0% during the last 1/20 period of the evolutionary process. The population size was set to 20.

The experiments performed with this method can be replicated by downloading and installing FARSA from "https://sourceforge.net/projects/farsa/" and the experimental plugins from http://laral.istc.cnr.it/res/neuroevolution/PSHC.zip experimental plugin.

2.2. Results

Table 2.3 displays the results obtained on the classic non-markovian version of the double-pole balancing problem used in previous comparisons. In these experiments agents were evaluated for a single trial and the initial state of the cart and of the poles was set to $[x = 0, \dot{x} = 0, \theta_1 = 4°, \theta_2 = 0, \dot{\theta}_1 = 0, \dot{\theta}_2 = 0]$. The table displays the results obtained with the best parameters, i.e. the parameters that enable to minimize the number of evaluations required to solve the problem and to find a solution in all replications (see the caption of Table 2.3).

As can be seen, all methods manage to solve the problem in all replications. The results obtained with NEAT, CoSyNE and xNES are in line with those published in (Gomez, Schmidhuber and Miikkulainen, 2008; Wierstra, Schaul, Peters and Schmidhuber, 2008). The CGPANN method was tested previously only on a simplified version of the problem (see subsection 2.1.4). The performance statistically differs in all cases (Mann-Whitney U test, p-value < 0.05 with Bonferroni correction) with the exception of the performance of SSS versus CoSyNE (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction) and of NEAT versus CGPANN (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction) that do not differ among them. Consequently, xNES results the fastest method followed by PSHC, followed by CoSyNE and SSS, followed by CGPANN and NEAT.

Classic Double-Pole	**Evaluations**
xNES	395
PSHC	563
CoSyNE	1257
SSS	1557
CGPANN	6885
NEAT	7743

Table 2.3. Evaluations required to solve the problem averaged over 30 replications of the experiments and percentage of replications that successfully solved the problem (i.e.

that found a network capable of balancing the pole until the end of the trial). In the methods with fixed topologies, I used full recurrent architectures. The data reported in the table are those obtained with the best combination of parameters: xNES (fixed positions: LearningRate 0.5, NumHiddens 0), PSHC (MutRate 50%, Stochasticity 10%, Interbreeding 10%, NumHiddens 1), CoSyNE (MutRate 90%, SubPopulations 5, NumHiddens 1), SSS (fixed positions: MutRate 50%, Stochasticity 20%, NunHiddens 1), CGPANN (MutRate 3%; NumIncomingConnections 8, NumHiddens 2), and NEAT (popSize 100, NumHiddens 1.8, see Stanley and Miikkulainen [2002] for the other parameters). In the case of NEAT, the number of hidden neurons indicates the average number of internal neurons possessed by first solution obtained in each replication. The network of the fastest replication included 1 hidden neuron.

I now report the results obtained in the three extended versions of the double pole balancing problem (i.e. the standard, the delayed, and the long double-pole balancing problems). For each experiment I report the results obtained in the Fixed and Randomly Varying Initial States conditions. In the case of the long double-pole balancing problem I tested only the methods that produced good performance on the first two problems.

In the case of the methods that operate with fixed topology, the agents are provided with a three layers neural network with 10 internal neurons and recurrent connections. In all cases evolution is terminated after 16 million of evaluations (i.e. after a total of 16 million of trials were performed).

The performance and generalization abilities of the evolved agents are reported in Table 2.4 and Table 2.5. The Fixed Initial States condition refers to the experiments in which the state of the cart at the beginning of each trial is set in a deterministic manner (see Table 2.1). The Randomly Varying Initial States condition, instead, refers to the experiments in which the initial state of the cart is set randomly within the range indicated in Table 2.2. Performance indicates the fitness obtained by the best evolved agents evaluated from the same initial states that they experienced during evolution. Generalization indicates the fitness obtained by the best evolved agents post-evaluated for 1000 trials during which the initial state of the cart was set randomly within the ranges indicated in Table 2.2. The way in which performance varies during the course of the evolutionary process is reported in Figure 2.2 and Figure 2.3. In the case of NEAT, the neural network of evolved agents include 6.6 and 6.4 internal neurons, on the average in the case of the standard double-pole (fixed and randomly varying initial states) and 9.5 and 8.9 neurons, on the average in the case of the delayed double-pole.

Standard Double-Pole	**Fixed Initial States**		**Randomly Varying Initial States**	
	Performance	Generalization	Performance	Generalization
NEAT	0.879	0.397 [322]	0.696	0.710 [656]
CoSyNE	0.971	0.701 [609]	0.911	0.699 [594]
CGPANN	0.967	0.693 [622]	0.824	0.613 [506]
xNES	**1.0**	0.717 [696]	0.889	0.897 [884]
SSS	**1.0**	0.821 [791]	**1.0**	0.906 [893]
PSHC	**1.0**	0.785 [746]	**1.0**	0.807 [788]

Table 2.4. Performance and generalization ability of neural network controllers evolved with different methods on the double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. The data reported in the table are those obtained with the best combination of parameters: NEAT (fixed positions: Population 1000; varying positions: Population 1000, for the other parameters see Stanley and Miikkulainen, 2002), CoSyNE (fixed positions: MutRate 5%, SubPopulation 60; varying positions: MutRate 10%, SubPopulation 60), CGPANN (fixed positions: MutRate 20%, Incoming Connections 8); varying positions: MutRate 1%, Incoming Connections 4), xNES (fixed positions: LearningRate 0.2; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 7%, Stochasticity 10%; varying positions: MutRate 3%, Stochasticity 10%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%, Interbreeding 10%; varying positions: MutRate 7%, Stochasticity 0%, Interbreeding 10%).

Delayed Double-Pole	**Fixed Initial States**		**Randomly Varying Initial States**	
	Performance	Generalization	Performance	Generalization
NEAT	0.292	0.038 [0]	0.300	0.037 [0]
CoSyNE	0.589	0.453 [380]	0.642	0.434 [360]
CGPANN	0.206	0.077 [40]	0.367	0.284 [171]
xNES	0.943	0.692 [679]	0.992	0.911 [903]
SSS	0.996	0.756 [731]	**1.0**	0.873 [854]
PSHC	**1.0**	0.685 [635]	**1.0**	0.755 [722]

Table 2.5. Performance and generalization ability of neural network controllers evolved with different methods on the delayed double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test. The best performance is indicated in bold. The data reported in the table are those obtained with the best combination of parameters: NEAT (fixed positions: Population 1000; varying positions, Population 1000, for the other parameters see Stanley and Miikkulainen, 2002), CoSyNE (fixed positions: MutRate 10%, Subpopulations 60; varying positions: MutRate 5%, Subpopulations 60), CGPANN (fixed positions: MutRate 20%, Incoming Connections 8; varying positions: MutRate 1%, Incoming Connections 5), xNES (fixed positions: LearningRate 0.2; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 3%, Stochasticity 100%; varying positions: MutRate 7%, Stochasticity 70%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%; Interbreeding10%, varying positions: MutRate 3%, Stochasticity 0%, Interbreeding 10%).

As can be seen the SSS, xNES and PSHC methods outperform the NEAT, CGPANN and CoSyNE methods in the case of the standard and delayed double pole problems (Table 2.4 and Table 2.5). The performance of the former three methods are significantly better than the performance of the latter three methods (Mann-Whitney U test, p-value < 0.05 with Bonferroni correction in all cases) with the exception of the performance of xNES and CoSyNE methods in the random initial states condition which do not differ statistically among themselves (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction). Given the clear superiority of the former methods, I did not test the latter methods on the long double-pole balancing problem.

The agents evolved with the SSS and PSHC methods also generalize better than the agents evolved with the NEAT, CGPANN and CoSyNE methods in all cases (Table 2.4 and Table 2.5, Mann-Whitney U test, p-value < 0.05 with Bonferroni correction). The xNES method generalizes better than the NEAT, CGPANN and CoSyNE methods in all cases with the exception of the fixed initial states condition (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction).

To analyse the speed with which the different methods find a close-to-optimal solution, I compared the number of evaluations required to reach a fitness equal or greater than 0.9 for methods that reached this threshold in at least 15 out of 30 replications (see Table 2.6 and Figure 2.2). In the case of the Standard Double-Pole problem, the SSS method is significantly faster than xNES and CGPANN methods (Median test, p-value < 0.05 with Bonferroni correction) and does not differ from the NEAT, CoSyNE and PSHC methods (Median test, p-value > 0.05 with Bonferroni correction). NEAT is significantly faster than xNES, CGPANN and PSHC methods (Median test, p-value < 0.05 with Bonferroni correction) and does not statistically differ from CoSyNE (Median test, p-value > 0.05 with Bonferroni correction). The CGPANN, NEAT, CoSyNE and PSHC methods do not differ among themselves (Median test, p-value > 0.05 with Bonferroni correction). In the case of the Delayed Double-Pole problem, only the agents evolved with the xNES, PSHC and SSS method reach a fitness of 0.9 in 15 out of 30 replications. The speed with which they reach sufficiently good performance does not statistically differ (Median test, p-value > 0.05 with Bonferroni correction in all cases).

Fixed Initial States condition	**Standard Double-Pole**	**Delayed Double-Pole**
NEAT	234,666.6	*
CoSyNE	256,000.0	*
CGPANN	2,389,333.3	*
xNES	490,666.6	1,450,666.6
SSS	192,000.0	3,989,333.3
PSHC	405,333.3	1,902,933.3

Table 2.6. Average number of evaluations required to reach a fitness equal or greater than 0.9 for the first time. Asterisks indicate the conditions that failed to reach this threshold in more than 15 out of 30 replications.

An additional advantage of the SSS, xNES and PSHC methods is constituted by the fact that they require to set few parameters and tend to produce good solutions for a wide range of parameters' values, especially in the Randomly Varying Initial State conditions. The SSS method, for example, requires setting only two parameters (the mutation rate and the stochasticity range) and displays rather high performance for most combinations of parameters' value (see Table 2.7). Data for the other methods and for the other experimental conditions are available in the supporting information (see S1-S5 Tables).

Double-Pole	**MutRate1%**	**MutRate3%**	**MutRate5%**	**MutRate7%**	**MutRate10%**
Stochasticity 0%	1.0 [0.901]	1.0 [0.899]	1.0 [0.894]	1.0 [0.890]	1.0 [0.888]
Stochasticity 10%	1.0 [0.897]	1.0 [0.906]	1.0 [0.897]	1.0 [0.894]	1.0 [0.888]

Stochasticity 20%	0.996 [0.896]	1.0 [0.896]	1.0 [0.895]	0.998 [0.881]	1.0 [0.880]
Stochasticity 30%	1.0 [0.894]	1.0 [0.895]	1.0 [0.881]	1.0 [0.883]	1.0 [0.881]
Stochasticity 40%	0.989 [0.870]	1.0 [0.893]	1.0 [0.875]	1.0 [0.874]	1.0 [0.875]
Stochasticity 50%	1.0 [0.887]	0.998 [0.880]	0.999 [0.870]	1.0 [0.882]	1.0 [0.872]
Delayed Double-pole	**MutRate1%**	**MutRate3%**	**MutRate5%**	**MutRate7%**	**MutRate10%**
Stochasticity 0%	0.917 [0.744]	0.956 [0.806]	0.957 [0.818]	0.962 [0.809]	0.972 [0.805]
Stochasticity 10%	0.894 [0.743]	0.909 [0.748]	0.977 [0.820]	0.960 [0.836]	0.988 [0.847]
Stochasticity 20%	0.952 [0.780]	0.993 [0.829]	0.994 [0.844]	0.969 [0.818]	0.971 [0.827]
Stochasticity 30%	0.965 [0.812]	0.983 [0.839]	0.969 [0.831]	0.963 [0.826]	0.966 [0.822]
Stochasticity 40%	0.942 [0.789]	0.989 [0.852]	0.983 [0.852]	1.0 [0.862]	0.994 [0.863]
Stochasticity 50%	0.982 [0.835]	0.986 [0.836]	0.993 [0.856]	0.993 [0.856]	0.980 [0.823]
Stochasticity 70%	1.0 [0.847]	1.0 [0.862]	0.994 [0.862]	1.0 [0.873]	1.0 [0.869]
Stochasticity 100%	0.988 [0.834]	0.996 [0.845]	1.0 [0.867]	0.818 [0.700]	0.350 [0.250]

Table 2.7. Performance and generalization ability of neural network controllers evolved withthe SSS method on the standard and delayed double-pole balancing problem in the Varying Initial States experimental condition in experiment carried out with different combination of parameters. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during

which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test.

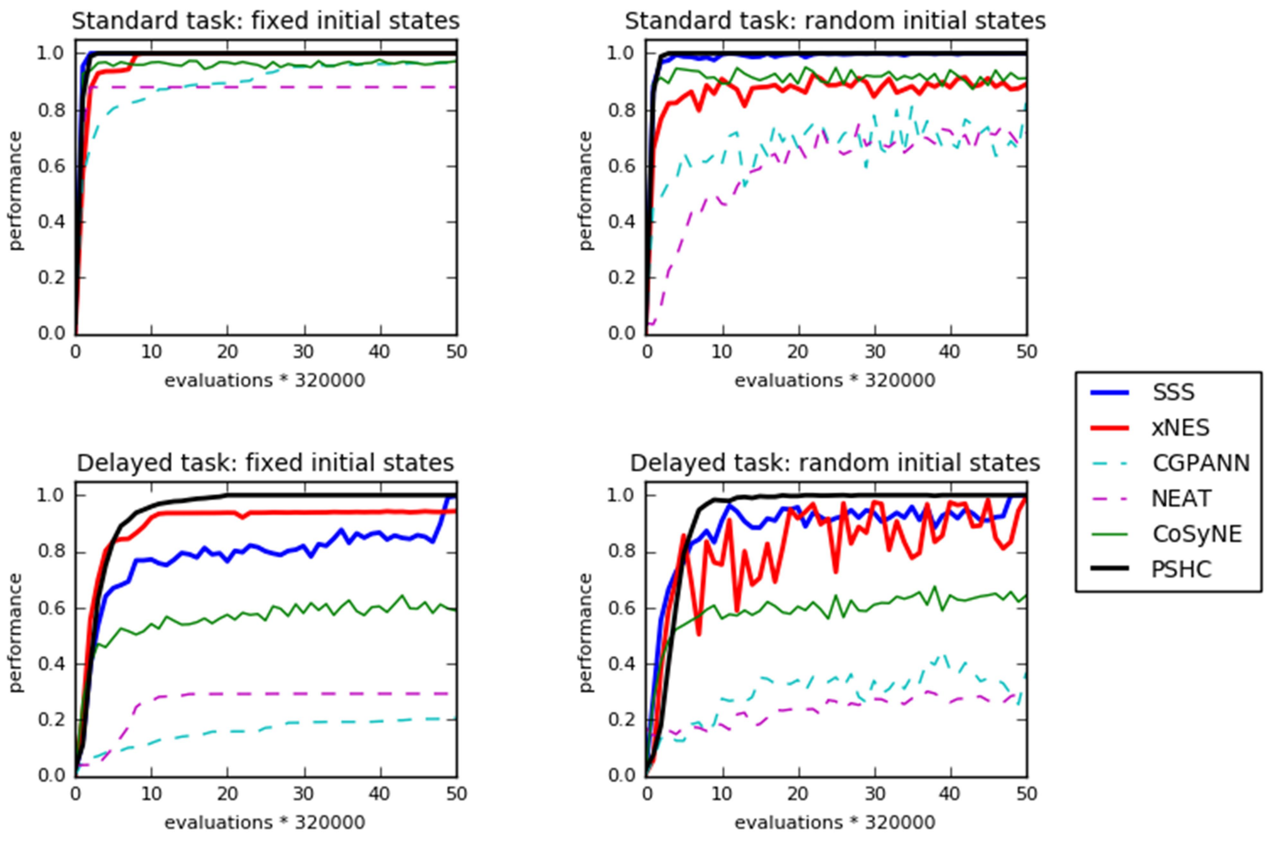

Figure 2.2. Performance of the best agents evolved through different methods during the evolutionary process. The top and bottom pictures display that data of the standard and delayed double-pole problems, respectively. The left and right pictures display the data of the experiment carried out in the Fixed and Randomly varying initial conditions, respectively. Each curve displays the average result of the best 30 networks evolved in the 30 corresponding replications of each experiment. Data refer to the best individual of the population.

The differences in performance and generalization among the SSS, xNES, and PSHC methods are less marked. In the case of the long double-pole balancing problem (see Table 2.8 and Figure 2.3), that turned out to be much harder than the other two problems, the best performance in the Fixed Initial States condition are achieved by the xNES method (Mann-Whitney U test, p-value < 0.05 with Bonferroni correction). The performance of the SSS and the PSHC methods do not vary statistically among themselves (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction). In the case of the Randomly Varying Initial States condition, the best performance is achieved by the SSS and PSHC methods that differ statistically from the xNES method (Mann-Whitney U test, p-value < 0.05 with Bonferroni correction) and do not differ among themselves (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction). The xNES method generalizes significantly better than the SSS and PSHC method in the case of the Randomly Varying Initial States condition (Mann-Whitney U test, p-value < 0.05 with Bonferroni correction). In the case of the Fixed Initial States condition, instead, the generalization performance of the xNES method does not statistically differ from those of the SSS method (Mann-Whitney U test, p-value > 0.05 with Bonferroni correction).

Long double-pole	**Fixed Initial States**		**Randomly Varying Initial States**	
	Performance	Generalization	Performance	Generalization
xNES	**0.842**	0.220 [190]	0.502	0.520 [493]
SSS	0.802	0.247 [197]	0.664	0.431 [394]
PSHC	0.799	0.209 [142]	**0.693**	0.312 [245]
PSHC*	0.715	0.238[179]	0.602	0.335 [261]

Table 2.8. Performance and generalization ability of neural network controllers evolved with different methods on the long double-pole balancing problem. Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart has been set randomly. The best performances are indicated in bold. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced

for the entire duration of the trial during the post-evaluation test. The data reported in the table are those obtained with the best combination of parameters: xNES (fixed positions: LearningRate 0.5; varying positions: LearningRate 0.1), SSS (fixed positions: MutRate 20%, Stochasticity 30%; varying positions: MutRate 3%, Stochasticity 0%), PSHC (fixed positions: MutRate 7%, Stochasticity 50%, Interbreeding10%; varying positions: MutRate 3%, Stochasticity 0%, Interbreeding 10%), PHSC* (fixed positions: MutRate 7%, EliminateConnection 40%, Stochasticity 70%, Interbreeding 10%; varying positions: MutRate 1%, EliminateConnection 15%, Stochasticity 50%, Interbreeding 10%).

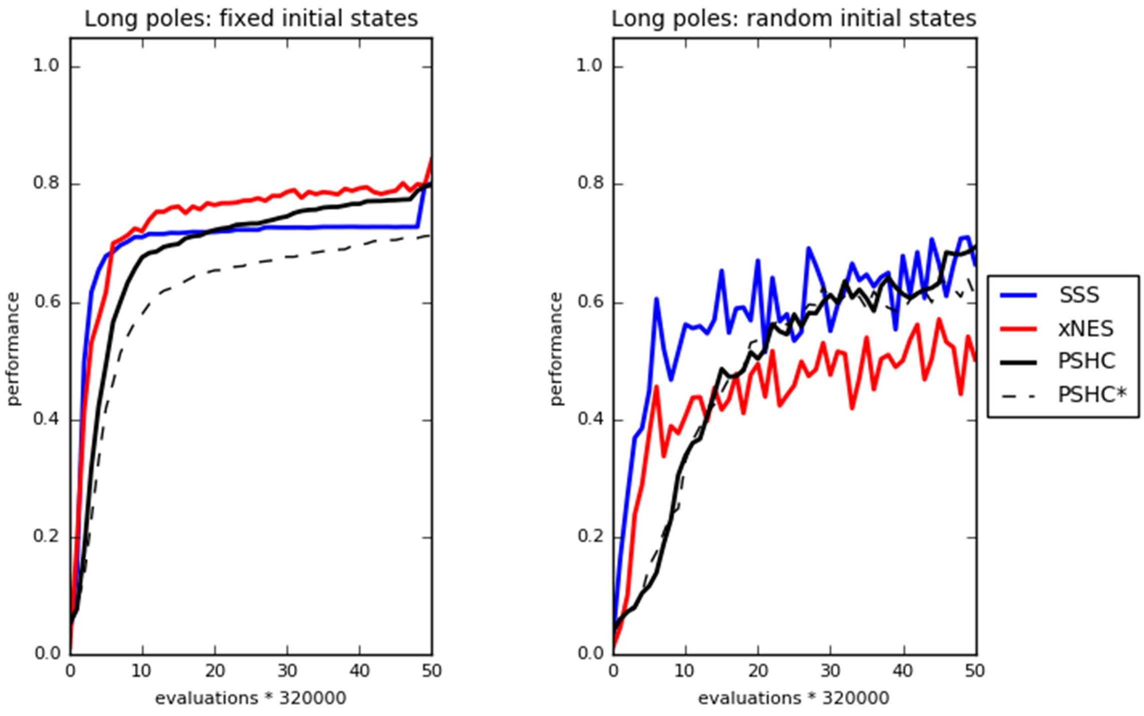

Figure 2.3. Long double-pole problem. Performance of the best agents during the course of the evolutionary process. The left and right pictures display the data of the experiment carried out in the Fixed and Randomly varying initial conditions,

respectively. Each curve displays the average result of the best 30 networks evolved in the 30 corresponding replications of each experiment. Data refer to the best individual of the population.

The results of the PSHC* reported in Table 2.8 and Figure 2.3 refer to the experiments carried out with the variation of the Parallel Stochastic Hill Climber method that permits to adapt also the topology of the neural network. In consideration of the fact that the networks produced by the PSHC* method are not fully connected, I set the number of internal neurons to 15 instead of 10, as in the other experiments. In the case of the Randomly Varying Initial States condition, the performance and generalization capabilities obtained with this method do not differ statistically from those obtained with the standard PSHC method that operates with a fixed network topology (Mann-Whitney U test, p-value > 0.05). In the case of the Fixed Initial States condition, instead, the PSHC* method produces worse performance (Mann-Whitney U test, p-value < 0.05) and better generalization (Mann-Whitney U test, p-value > 0.05) than the standard PSHC method. The percentage of connections present in the evolved networks matches very closely the value of the *EliminateConnection* parameter. In other words, the average rate of connectivity of evolved networks is determined by setting the parameter while the specific connections that are present or absent are determined by the evolutionary process. By systematically varying the *EliminateConnection* parameter I observed that the best results are obtained by setting this parameter to 40% and 15% in the case of the experiments carried out in the Fixed and Randomly Varying Initial States experimental conditions, respectively.

I now discuss the relationship between the variability of the environmental conditions in which the agents are evaluated during the course of the evolutionary process and the performance and generalization capabilities of the evolved agents.

An advantage of evaluating the agents in randomly varying environmental conditions is that it fosters the selection of solutions that are robust with respect to these variations, i.e. solutions that generalize. The fact that the environmental conditions vary, on the other hand, implies that the fitness measure of individuals can be over or under-estimated. The fitness is over-estimated when the individual happens to be evaluated in easier than average environmental conditions and under-estimated when the individual happens to be evaluated in more complex than average conditions. The effect of this over or under-estimation is analogous to the effect of adding noise to the fitness. The addition of noise to the fitness (within limits) does not reduce the efficacy of evolution but rather promotes the evolution of better solutions (see Table 2.9). Consequently, the utilization of randomly varying conditions for the evaluation of the evolving agents is expected to produce better results than the utilization of fixed environmental condition.

Standard double-pole	**Stochasticity = 0.0**	**Stochasticity>0**
SSS	1.0	1.0
PSHC	1.0	1.0
Delayed double-pole	**Stochasticity = 0.0**	**Stochasticity>0**
SSS	0.867	0.996
PSHC	0.823	1.0
Long double-pole	**Stochasticity = 0.0**	**Stochasticity>0**
SSS	0.760	0.802
PSHC	0.479	0.799

Table 2.9. Performance and generalization ability of the agents evolved with the PSHC and the SSS method in the Fixed Initial States condition in the experiments carried out with or without stochasticity. The numbers refer to the data obtained with the best combination of parameters, i.e. with the best mutation rate in the case of the data reported in the second column, and with the best mutation rate and stochasticity level in the case of the data reported in the third column.

This hypothesis is confirmed by the comparison of the results obtained in the Fixed and in the Randomly Varying Experimental condition (see Table 2.10). As expected, in fact, the agents evolved in the Randomly Varying Initial States condition generalize better than the agents evolved in the Fixed State Initial States condition in all cases. The performance in the two conditions are not comparable since the agents evolved in fixed environmental conditions tend to select solutions that are particularly effective for those conditions while agents evolved in randomly varying initial conditions cannot produce this type of over-fitting.

	Fixed Initial States	**Randomly Varying Initial States**	**Mann-Whitney U test, p-value**
Standard double-pole			
SSS	0.821 [791]	0.906 [893]	< 0.05
xNES	0.717 [696]	0.897 [884]	< 0.05
PSHC	0.785 [746]	0.807 [788]	< 0.05
Delayed double-pole			
SSS	0.756 [731]	0.873 [854]	< 0.05
xNES	0.692 [679]	0.911 [904]	< 0.05
PSHC	0.685 [635]	0.755 [722]	< 0.05
Long double-pole			
SSS	0.247 [197]	0.431 [394]	< 0.05
xNES	0.220 [190]	0.520 [493]	< 0.05
PSHC	0.209 [142]	0.312 [245]	< 0.05

Table 2.10. Statistical difference between the generalization performance obtained in the Fixed and Randomly Varying Initial States conditions for the experiment carried with the xNES, SSS and PSHC methods. Data collected on the experiments carried with the best parameters (see Table 2.4, Table 2.5 and Table 2.8). Each number indicates the average performance obtained during 30 replications of the experiment. Generalization refers to the average performance obtained by post-evaluating the evolved networks on 1000 trials during which the initial states of the cart have been set randomly. The numbers in square brackets indicate the number of trials in which the agents manage to maintain the poles balanced for the entire duration of the trial during the post-evaluation test.

The other aspect that strongly affects the performance of the evolving agents is the number of the trials used, i.e. the number of different conditions in which the agents are evaluated (see Table 2.11). The comparison of the results obtained by varying systematically the number of trials indicates that the best performance and generalization abilities are achieved by using a relatively small number of trials (i.e. 4-8 trials).

This can be explained by considering that increasing the number of trials has both costs and benefits. The computational cost increases linearly with the number of trials. Consequently when the computational cost is maintained constant, as in the case of my experiments, the utilization of a larger number of trials leads to a reduction of the number of generations that impact negatively on performance. The utilization of additional trials, on the other hand, facilitates the selection of robust solutions and permits to reduce the level of noise in the fitness estimation (which constitutes an advantage when the fitness is too noisy as a consequence of the fact that the fitness of the agent is evaluated on few trials).

N. Trials	**Double-Pole**	**Delayed Double-Pole**	**Long Double-Pole**
1	1.0 [0.826]	1.0 [0.818]	0.180 [0.075]
2	1.0 [0.863]	1.0 [0.870]	0.458 [0.182]
4	1.0 [0.882]	1.0 [0.865]	**0.779** [0.384]
8	**1.0** [0.906]	**1.0** [0.873]	0.664 [0.431]
16	0.990[0.907]	0.969 [0.867]	0.683 [0.421]
32	0.968 [0.905]	0.903 [0.821]	0.493 [0.353]
64	0.917 [0.885]	0.855 [0.810]	0.325 [0.261]

Table 2.11. Performance and generalization capabilities of neural network controllers evolved with the SSS method in the Randomly Varying Initial States condition by using a variable number of trials. Data collected by using the best parameters. The numbers in square brackets indicate the average performance obtained by post-evaluating the

corresponding 30 best evolved agents for 1000 trials during which the initial state of the cart have been set randomly within the range shown in Table 2.2. Each number indicates the average performance obtained during 30 replications of the corresponding experiment.

2.3. Discussion

Neuroevolution represents a convenient technique that can be potentially applied to any type of problem, since it requires only a scalar value indicating the ability of the neural network to deal with the given task. Nonetheless, its efficacy depends on the possibility to improve candidate solutions for prolonged periods of time without entering in stagnation phases caused by the preliminary convergence into sub-optimal solutions or by the inability to keep generating and/or retaining better solutions.

The interest in this technique has led to the development of several alternative methods that I briefly reviewed. In this chapter, I analysed the adaptive power of some of the most promising methods described in the literature, including two new methods that I developed, in the context of the double-pole balancing problem. By adaptive power, I mean the capacity of a method to discover solutions that are robust to variations of the environment and the capacity to scale-up to more complex versions of the problem. Consequently, differently from previous related works (Stanley and Miikkulainen, 2002; Igel, 2003; Durr, Mattiussi and Floreano, 2006; Gomez, Schmidhuber and Miikkulainen, 2008; Wierstra, Schaul, Peters and Schmidhuber, 2008; Khan, Khan and Miller, 2010) that investigated the speed with which alternative method discover sufficiently good solutions, I compared alternative methods for the ability to: (i) generate high performing solutions, (ii) generate solutions that generalize in varying environmental conditions, (iii) scale up to complexified versions of the double-pole problem (i.e. a problem in which the agent needs to act on the basis of the state of the world at time t-1 and a problem in which the length and the mass of the shorter pole is increased from $\frac{1}{10}$ to $\frac{1}{2}$ the length and the mass of the long pole).

The obtained results indicate that the two new introduced methods, namely the Stochastic Steady State (SSS) and the Parallel Stochastic Hill Climber (PSHC) methods, and the Exponential Natural Evolution Strategies (xNES) method proposed by Wierstra, Schaul, Peters and Schmidhuber (2008) (see also Wierstra, Schaul, Glasmachers, Sun, Peters and Schmidhuber, 2014) largely outperform the other methods considered, namely the NeuroEvolution of Augmenting Topologies method (NEAT), the Cartesian Genetic Programming of Artificial Neural Network (CGPANN), and the Neural Evolution through Cooperatively Coevolved Synapses (CoSyNE) methods with respect to all criteria. Indeed, the first three methods are able to generate solutions displaying significantly better performance and generalization capabilities than the other three methods. The fact that the offset in performance between the former and the latter methods is greater in the delayed double-pole balancing problem than in the standard double-pole balancing problem demonstrates how the former method also scale-up better to more complex problems. Finally, the fact that best three methods quickly outperform the other methods indicates that the advantage in term of adaptive power is not gained at a cost of a reduced adaptive speed.

I minimized the effect of parameter setting by varying systematically the most important parameters and by considering the performance achieved with the best combination of parameters. This was possible only in part in the case of NEAT, since it requires setting many parameters. Consequently, clearly NEAT could produce better performance with different parameter values.

I hypothesise that the efficacy of the SSS and PSHC methods is due to their capacity to effectively retain adaptive traits, thanks to the utilization of a steady state selection, and to keep exploring new portions of the search space, thanks to the utilization of an effective method for regulating the selective pressure. The ability to avoid losing previously discovered adaptive traits is probably particularly important in hard problems that involve a high-dimensional search space and that require the accumulation and the preservation of many adaptive changes.

The high performance of the xNES could be explained by its ability to estimate and use local gradient information and by its tendency to explore promising directions of the search space. This could represent a disadvantage in problems affected by strong local minima. For an analysis of the performance of this method on the optimization of standard benchmark functions, see Wierstra, Schaul, Peters and Schmidhuber (2008).

The comparison among the three best methods does not highlight a clear winner. The fact that the SSS and the xNES methods outperform the PSHC method in the case of the long double-pole problem indicates that they have a greater ability to scale-up to more complex problems than the PSHC. An advantage of the SSS method with respect to the xNES method consists in the fact that it is simpler. On the other hand, the ability to estimate the local fitness gradient might enable the xNES method to outperform the SSS method in certain domains. This aspect that should be investigated in future research.

I also demonstrated that the utilization of randomly varying environmental conditions promotes the evolution of robust solutions and that the performance and the generalization abilities of the evolved agents are strongly influenced by the number of trials used for evaluating evolving individuals.

Finally, my results demonstrate that the methods that permit to evolve the topology and the connection weights of the network (i.e. NEAT and CGPANN) are outperformed by methods that operate with fixed topologies (i.e. PSHC, SSS, and xNES), at least in the case of the double-pole balancing problem domain. The PSHC* method introduced in this thesis, instead, achieved performance that are comparable with those obtained by the best methods operating with fixed topology (i.e. PSHC, SSS, and xNES). Overall this indicates that whether or not the possibility to co-adapt the topology of the network can increase the adaptive power of neuroevolution still represents an open question. Future research in this area should clarify the characteristics of the domains in which the possibility to co-adapt the topology of the network can represent an advantage and should identify the way in which such advantage can be maximized.

Other areas that deserve farther research include the study of the characteristics of the evolving networks and the role of the environmental variations. The possibility to evolve network with heterogeneous neurons and topology (see for example Stanley, 2007; Turner and Miller, 2014) and/or networks provided with regulatory mechanisms (see for example Philippides, Husbands, Smith and O'Shea, 2005) represent promising research lines. Further research is required to verify whether these techniques can be embedded in methods that are competitive with respect to the state of the art. With regard to the role of environmental variations, I demonstrated the advantage of evaluating evolving agents in randomly varying environmental conditions and the advantage of exposing evolving agents to a limited number of variable conditions. Further research should investigate the effect of the rate of variations of the environmental conditions throughout generations and the effect of spatial versus temporal variations of the environmental conditions.

Chapter 3. Integrating Learning by Experience and Demonstration in Autonomous Robots

Introduction

Humans frequently learn by using a combination of demonstration and trial and error. For example, children often learn tasks by first observing other adults executing them and then attempting to re-produce the demonstrated behaviours several times until a certain level of performance is achieved. Similarly, students usually observe their teachers/masters performing actions and then try to replicate what they saw for many hours before obtaining a similar degree of precision/quality (as in the example of learning to play tennis provided in Kober, Bagnell and Peters, 2013).

From a machine learning perspective, the combined use of learning by demonstration (Billard, Callinon, Dillmann, and Schaal, 2008; Argall, Chernova, Veloso, and Browning, 2009) and learning by experience (Sutton and Barto, 1998; Nolfi and Floreano, 2000; Kober, Bagnell and Peters, 2013) provides advantages and disadvantages. As already debated in section 1.3, benefits include the exploitation of a richer/larger training feedback (i.e., a quantitative reward of the obtained performance and a qualitative measure in the form of a demonstration) and the possibility to observe suitable behaviours, which reduces the search space to those behaviours that re-produce the demonstration. Another advantage concerns the fact that the possibility to rely on the objective scalar measure evaluating the functionality of the overall robot's behaviour enables the learning robots to both select variations that represent real progresses and exploit properties that emerge from the agent/environmental interactions.

On the other hand, the two learning modes have also drawbacks. Indeed, in learning by demonstration, the attempt to solve the task by approximating a demonstrated solution, rather than directly optimising performance, exposes learning agents to the problems caused by the fact that minor differences between the demonstrated and reproduced behaviours might cumulate over time eventually causing huge undesired effects. In learning by experience, the selection of variations that represent genuine advance with respect to the current capabilities of the agent might drive the learning process toward local minima that might then prevent the discovery of better solutions. Furthermore, the combination of these two learning modes has potential shortcomings as well. In effect, channelling the learning process through demonstrated solutions is likely to be beneficial only when the demonstrated behaviour does represent a solution that is effective and learnable from the point of view of the learning agent. This is not necessarily the case when the learning agent is a robot and the demonstrator is a human, i.e. when the demonstrator and the learning agent have different body structures and different cognitive capabilities. Finally, the combined effects of two different learning methods can drive the learning process toward the synthesis of intermediate solutions, representing a sort of compromise between the changes driven by the two learning modes, which are not necessarily functionally effective.

Over the last few years, several researchers have proposed hybrid learning by demonstration and experience methods for robot training. In particular, Rosenstein and Barto (2004) extended an actor critic reinforcement learning algorithm (Sutton and Barto, 1998) with a supervisor agent that co-determines, together with the learning agent, the actions to be executed at each time step. These actions are generated by performing a weighted sum of the actions proposed by the supervisor and the learning agent. The parameter that trades off between the two actions is varied by taking into account the estimated efficacy of both the supervisor and the agent in specific circumstances and the need to progressively increase the autonomy of the agent during the learning process. Overall, the results obtained in a series of case studies demonstrate that the combination of the two learning modes can provide advantages. Nonetheless, the strategy of averaging the actions suggested by the supervisor and the agent might limit the efficacy of the algorithm to domains in which the negative effects caused by the summation of incongruent actions are negligible. In a related work Judah, Roy, Fern and Dietterich (2010) incorporated user advice into a Reinforcement Learning algorithm. More specifically, learners alternate between practice (i.e., experience) and end-user critique, where advice is gathered. The user analyses the behaviour of the learners and marks subset of actions as good or bad. The policy is optimised through a linear combination of practice and critique data. Results obtained in a series of experiments demonstrate that the approach significantly outperformed pure reinforcement learning. However, the study revealed usability issues related to the amount of practice and advice needed to achieve successful performance.

Kober and Peters (2009) investigated how a special kind of pre-structured parameterised policies called motor primitives (Ijspeert, Nakanishi and Schaal, 2003; Schaal, Mohajerian and Ijspeert, 2007) can be subjected to a combined learning by demonstration and experience training (see also Daniel, Neumann and Peters, 2012; Paraschos, Daniel, Peters and Neumann, 2013). Motor primitives are constituted by two coupled parametrical differential equations. The equations are hand-designed, while the equation parameters are learned. Learning occurs in two phases: first, the parameters are subjected to a learning by demonstration process; then, the parameters are refined on the basis of a reinforcement learning process. This method has been successfully applied to a series of distal rewarded tasks that could not be solved through learning by demonstration by itself. However, it operates appropriately only when the amount of deviation from the demonstrated trajectories is actively limited (Kober and Peters, 2009; Kober, Bagnell and Peters, 2013; Valenti, 2013). As a result, the method can only be successful when the learning by demonstration phase enables the learning robot to acquire a close-to-optimal strategy that only requires subsequent fine-tuning.

Argall, Browning and Veloso (2008) proposed a method for enriching the demonstration data set with advice produced by the human demonstrator during the observation of the behaviour exhibited by the learning robot. Advice consists in required modifications such as “turn less/more tightly” or “use a smoother translational speed” that can be easily identified by a human observer and can be used to automatically generate additional demonstration data obtained through a corrected version of the behaviour displayed by the learning robot. The collected results indicate that this technique can outperform standard learning by demonstration methods. Consequently, as hypothesised, personalising the demonstration depending on the characteristics of the learning robot can improve the outcome of the learning process. Nevertheless, this method only operates by attempting to reduce the discrepancy between the actual and the demonstrated behaviour and does not combine this with a learning by experience method driven by a direct measure of performance. Therefore, it does not provide a way to overcome the problem caused by the fact that minor inevitable differences between the demonstrated and the reproduced behaviour can lead to large negative consequences over time.

Other related works are those of Abbeel and Ng (2005), Cetina (2007), Chernova and Veloso (2007) and Knox and Stone (2009, 2010). Abbeel and Ng (2005) demonstrated the advantage of initialising the state-action pairs of a Reinforcement Learning algorithm based on a learning by demonstration method. Cetina (2007) implemented an algorithm in which the advice of the user is used to restrict the set of possible actions explored by the Reinforcement Learning algorithm. Chernova and Veloso (2007) proposed a Learning by Demonstration approach in which the learning agent can progressively increase its autonomy by reducing the number of demonstrations required from the teacher. Knox and Stone (2009) proposed a Reinforcement Learning method that operates on the basis of short-term rewards provided by human trainers that predict the expected long-term effect of single actions or short sequence of actions.

A still unanswered question of all the presented approaches concerns how different training feedbacks can be effectively integrated. Indeed, the different learning modes (i.e., learning by experience and learning by demonstration) often drive the learning agents toward different outcomes, whose effect is not necessarily additive. For example, the combination of the two processes in sequence, that can be obtained by first subjecting the robot to a learning by demonstration phase and then to a learning by experience phase, might cause the washing out of the capabilities acquired during the first phase at the beginning of the second phase (Kober, Bagnell and Peters, 2013; Valenti, 2013). Similarly, a combination achieved by summing up the effects of the two processes might lead to un-desirable consequences.

In this chapter, I introduce a new method that integrates the learning by demonstration and by experience in a single algorithm. This algorithm operates by estimating the local gradient of the current candidate solution with respect to an objective performance measure and exploring preferentially variations that reduce the differences between the robot and the demonstrated behaviour.

The results obtained on two qualitatively different tasks demonstrate how the algorithm is able to synthesize effective solutions. Such solutions are qualitatively similar to the demonstrated behaviour with respect to the characteristics that are functionally appropriate, while deviate from the demonstration with respect to other characteristics.

In section 3 I illustrate the first experimental setting and the learning algorithm. In section 3.2, the obtained results will be described. The second experimental setting and the corresponding obtained results are described in sections 3.3 and 3.4. Finally, in section 3.5 I discuss the implications of the results obtained.

3.1. Exploration experiment

In a first experiment a simulated Khepera robot (Mondada, Frenzi and Ienne, 1994) has been trained for the ability to explore an arena surrounded by walls that contains cylindrical obstacles located in randomly varying positions (Figure 3.1). With the term explore I mean “to visit the highest possible number of environmental locations”. To verify the advantages and disadvantages of different learning modes and of their combination I carried out three series of experiments in which the robot has been trained through a learning by experience, a learning by demonstration, and an integrated learning by experience and demonstration algorithm. The three algorithms are described in subsections 3.1.1, 3.1.2 and 3.1.3.

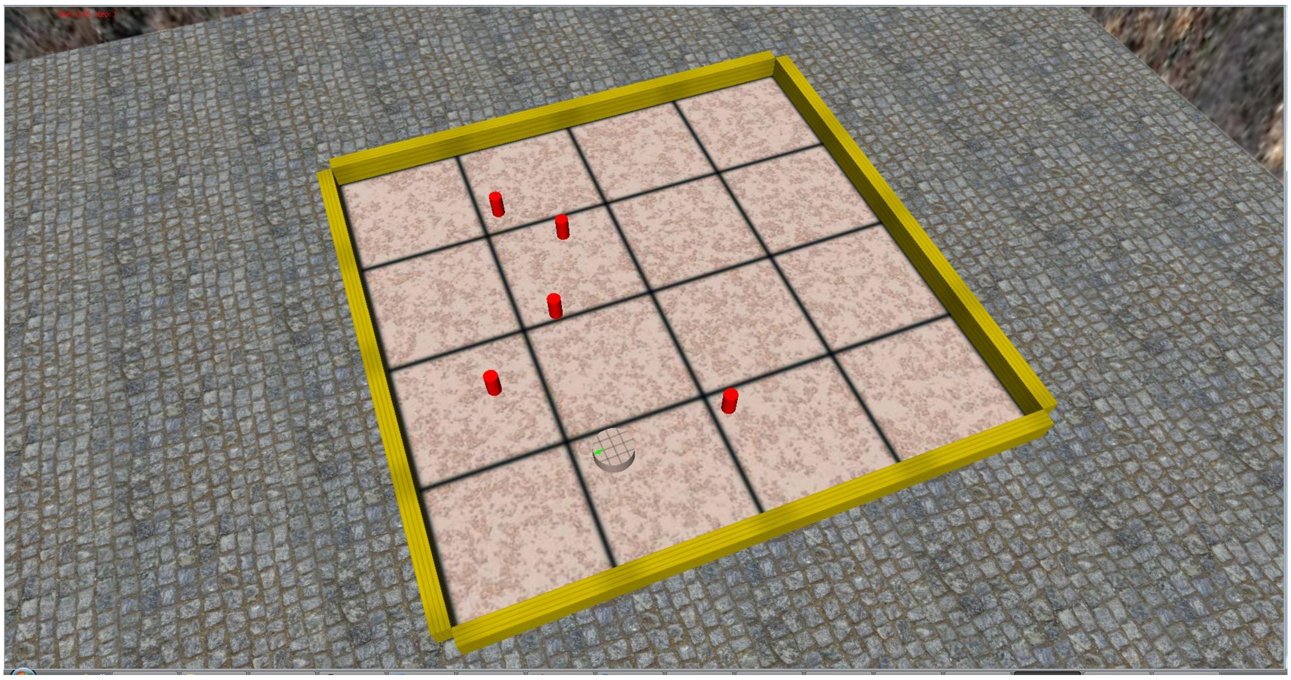

Figure 3.1. Exploration Experiment. The robot and the environment.

The robot is provided with 6 infrared sensors located in its front side that allow it to detect nearby obstacles and two motors controlling the desired speed of the two wheels. The robot's controller consists of a three-layer feed-forward neural network with 6 sensory neurons, which encode the state of the six corresponding infrared sensors, 6 internal neurons, and 2 motor neurons, which encode the desired speed of the two wheels. The activation of the infrared sensors is normalised in the range [0.0, 1.0]. The activation of the internal and motor neurons is computed based on the logistic function. The activation of the motor neurons is normalised in the range [-10.4, 10.4] cm/s. Internal and motor neurons are provided with biases.

The environment consists of a flat arena of 1.0x1.0m, surrounded by walls, containing five cylinders with a diameter of 2.5cm located in randomly varying positions. The robot and the environment have been simulated by using FARSA (Massera, Ferrauto, Gigliotta and Nolfi, 2013 and 2014), an open software tool that has been used to successfully transfer results obtained in simulation to hardware for several similar experimental settings (e.g. Sperati, Trianni and Nolfi, 2008; De Greef and Nolfi, 2010).

The performance of the robot, i.e. the ability to visit all environmental locations, is computed by dividing the environment into 100 cells of 10x10cm and counting the number of cells that have been visited at least once over a period of 75s during which the robot is allowed to interact with the environment. To select robots able to carry out the task in variable environmental conditions, the robot's performance is evaluated during 25 trials, each lasting 75s. The position and the orientation of the robot and the position of the obstacles are randomly initialised at the beginning of each trial. The overall performance is calculated by averaging the performances obtained during the different trials. It is worth noting that the optimal performance is unknown since the time for exploration is limited and some of the cells are occupied by obstacles.

The 48 connection weights of the neural network and the 8 biases are the free parameters that are initially set randomly and then learned by using the algorithms described in the next sections. The values of these parameters determine the control policy of the robot, i.e. how the robot reacts to the experienced sensory states.

3.1.1. Learning by experience

In learning by experience methods the learning robot discovers either the behaviour through which the task can be solved, or the control rules that, in interaction with the environment, lead to the production of the selected behaviour. The learning process is driven by a scalar measure of performance that rates the extent to which the robot is able to accomplish the task. The utilisation of a performance measure that does not specify how the problem should be solved potentially enables the learning robot to find effective solutions that are often simpler and more robust than those identifiable by human designers (Argall, Browning and Veloso, 2010; Coates, Abbeel and Ng, 2008; Taylor, Suey and Chernova, 2011). On the other hand, the use of such an implicit training feedback, that constraints only loosely the course of the adaptive process, can initially drive the learning process toward the synthesis of sub-optimal behavioural solutions that constitutes local minima (i.e., that cannot be progressively transformed into better solutions without producing performance drops).

A first way to achieve a learning of this kind consists in using a Stochastic Hill Climber algorithm (Russell and Norvig, 2009), in which the free parameters of the robot's controller are randomly modified and the variations are retained or discarded depending on whether or not they produce an improvement of the robot's performance.

A second option consists in using a Reinforcement Learning algorithm (Sutton and Barto, 1998) that operates by calculating the estimated utility of possible alternative actions on the basis of received rewards and using this acquired information to preferentially select the actions with the highest expected utilities.

A third way entails the use of an evolutionary algorithm (Holland, 1975; Schwefel, 1995; Nolfi and Floreano, 2000) that operates on a population of candidate solutions. These solutions are selected based on their relative performance and varied through the production of offspring, i.e. copies modified through genetic operators such as mutation and crossing over.

Finally, a fourth option is provided by Natural Evolution Strategies (NES) (Wierstra, Schaul, Peters and Schmidhuber, 2008; see also Rechenberg, 1973; Schwefel, 1977) that operate by sampling the local gradient of the candidate solution (i.e., the correlation between variations of the free parameters and variation of performance) and modifying the candidate solution toward the most promising directions of the multi-dimensional parameter space. Samples are generated by creating varied copies of the candidate solution.

In my experiments I used the xNES algorithm (Glasmachers, Schaul, Sun, Wierstra and Schimdhuber, 2010), i.e. an algorithm of the latter type since, as I will see, it can be easily extended to incorporate also learning-by-demonstration feedbacks. The xNES algorithm has already been successfully tested on a series of domains including autonomous robot learning (Glasmachers, Schaul, Sun, Wierstra and Schimdhuber, 2010).

More specifically, the xNES algorithm works as described in Algorithm 1. The notation used differs from (Glasmachers, Schaul, Sun, Wierstra and Schimdhuber, 2010) in order to allow the reader to easily understand all the steps of the algorithm with plenty of details.

Algorithm 1: the xNES algorithm

```
initialise:
    the candidate solution (ind) by selecting the best over 100 parameters' sets
        generated by using eq. 1
    the number of offspring ($\lambda$), see eq. 2
    the covariance matrix (covMat) with zeroed values. The covariance matrix
        encodes the correlation between free parameters of high utility samples. The
        utility of a sample is the relative performance advantage with respect to the
        other generated samples.
    the learning rate used to update the individual ($\eta_{ind}$). I use $\eta_{ind}$=1.0.
    the update rate of the covariance matrix ($\eta_{covM}$), see eq. 3
  repeat
    for $k = 1 \dots \lambda$ do
        generate random variation vectors with a Gaussian distribution
            $s_k \square \pi(\bullet \mid \theta)$
        generate samples by adding the product of the variation vectors by the
            exponential of the covariance matrix to the candidate solution
            $z_k = ind + e^{covMat} \square s_k$
        evaluate the fitness of the samples
            $f_k = F(z_k)$
    end
    sort $z_k$ with respect to their performance $f_k$
        $ranking = z_{1,} \dots, z_\lambda, F(z_1) \square \dots \square F(z_\lambda)$
        calculate the utility of the samples $u_k$
                $u_k$ , see eq. 4
        sum-up variations weighted by utilities
                $\nabla_{ind} = \sum_k u_k \cdot s_k$
        calculate the estimated local gradient
                $\nabla_{covM} = \sum_k u_k \cdot (s_k \cdot s_k^T - I)$ , where $I$ is the identity matrix, and $T$
                superscript represents the matrix transposition operation
        move the candidate solution depending on the local gradient with the given
learning rate
                $ind = ind + \eta_{ind} \cdot e^{covMat} \cdot \nabla_{ind}$
        update the covariance matrix based on the estimated local gradient with the
            given update rate
                $covMat = covMat + \eta_{covM} \cdot \nabla_{covM}$
for 500 iterations
```

$$ind \sim \pi(\bullet \mid \theta) \qquad (1)$$

where π is a Gaussian distribution with parameters $\theta = (\mu,\sigma)$. μ represents the mean of the Gaussian distribution and it is set to 0.0. σ stands for the standard deviation of the Gaussian distribution and it is set to 1.0.

$$\lambda = 4 + floor(3 \cdot log(p)) \qquad (2)$$

where p is the number of free parameters, and *floor* indicates the inferior integer part of the number.

$$\eta_{covM} = 0.5 \cdot max\left(0.25, \frac{1}{\lambda}\right) \tag{3}$$

where *λ* is the number of offspring.

$$u_k = \frac{max\left(log\left(\frac{\lambda}{2}+1\right) - log(k)\right)}{\sum max\left(log\left(\frac{\lambda}{2}+1\right) - log(j)\right)} \tag{4}$$

where *λ* is the number of offspring and *k* is the index of the k-th offspring.

3.1.2. Learning by demonstration

In learning by demonstration the control policy is learned from examples or demonstrations provided by a teacher (Argall, Chernova, Veloso and Browning, 2009; Schaal, 1997). The objective of the learning process is therefore to reproduce the demonstrated behaviour. In particular, the learning algorithm should allow the discovery of the control mechanisms that, in interaction with the environment, yield the demonstrated behaviour (Billard, Calinon and Guenter, 2006). The examples can be defined either as the sequences of sensory-motor pairs that are extracted from the teacher demonstration (offline demonstration learning) (Hayes and Demiris, 1995), or as the sequences of the desired action specifications that the teacher provides to the learning robot while it is acting (online demonstration learning) (Rosenstein and Barto, 2004).

In general terms, the availability of the demonstration reduces the complexity of the learning task and permits the use of potentially more powerful supervised learning techniques. On the other hand, the behaviour demonstrated by the teacher might be impossible to learn from the robot's point of view (i.e., the demonstrator might be unable to identify behaviours learnable by the robot). Besides, the acquisition of an ability to produce a behaviour that is very similar but not identical to the demonstration does not guarantee the achievement of good performances since small differences might cumulate over time leading to significant differences.

In my experiment I generated the demonstrations automatically through a demonstration robot provided with a hand-designed controller. This allowed to generate a large number of demonstrations and to use exactly the same kind of demonstration in all experiments. The hand-designed controller of the demonstrator: (i) moves the robot straight at the highest possible speed when the infrared sensors are not activated, (ii) makes the robot turn left or right when the two right or the two left infrared sensors are activated respectively, and (iii) makes the robot turn left when both the two left and the two right infrared sensors are activated. This is made by setting by default the desired speed of the two wheels to the highest positive value and by proportionally reducing the speed of one wheel according to the average activation of the two opposite infrared sensors. It is worth noting that the optimal relation between the activation of the infrared sensors and the speed of the wheels, with respect to the ability of the robot to explore the arena, is not necessarily proportional and homogeneous for the two sensors located on either side. Nevertheless, since there is no clear way for determining this relation, I used a simple proportional and homogeneous relation.

The fact that certain characteristics are hard to optimise from the point of view of an external designer constitutes one of the reasons that explain why combining learning by demonstration and experience can be beneficial. Indeed, as I will see, the integrated learning by experience and demonstration algorithm can find solutions that are better optimised in this respect.

In the offline demonstration mode, the sequence of sensory-motor pairs experienced by the demonstrator robot, while it operates in the environment for several trials, are used to train the neural network controller of the learning robot. The training is performed by using the demonstrations as training set and learning the weights of the robot's neural controller through back-propagation (Rumelhart, Hinton and Williams, 1986). More specifically, the sensory states experienced by the demonstrator robot and the motor actions produced by the demonstrator robot every 50 ms are used to set the sensory state and the teaching input of the learning robot. Conversely, in the online demonstration mode, the learning robot is situated in the environment and can operate based on its own motor neurons. To generate the teaching input, the demonstrator robot is virtually placed in the same location (i.e., position and orientation) of the learning robot at each time step. The output actions produced by the demonstrator robot in this situation are used to set the teaching input of the learning robot.

In both cases, a learning rate of 0.2 was used and the learning process was continued for $1.875 * 10^7$ time steps of 50 ms. This number was chosen in order to keep the overall period of evaluation constant with respect to the previous algorithm.

3.1.3. Integrated learning by experience and by demonstration

Here I propose a new algorithm that enables the realisation of an integrated learning by experience and demonstration. The algorithm is designed so that the two learning modes can operate simultaneously by driving the learning process toward solutions that are functionally effective and similar to the demonstrated behaviour. The combination of the effects of the two learning modes is not realised at the level of the actions, as in the case of Rosenstein and Barto (2004), but rather at the level of the variations of the free parameters. In other words, the algorithm operates by combining variations that are expected to produce progress in objective performance with variations that are likely to increase the similarity with the demonstrated behaviour. As a whole, this implies that the algorithm tries to steer the learning by experience process toward solutions that resemble the demonstrated behaviour.

To obtain such an integrated algorithm, I designed an extended version of the xNES algorithm, described in section 3.1.1, that operates by iteratively estimating and exploiting both the local performance and the demonstration landscape. This has been made by training the current candidate solution for a limited time (i.e. 25 trials as in the case of the learning by experience algorithm) based on the demonstrated behaviour. More precisely, the integrated algorithm works as described in Algorithm 2:

Algorithm 2: the integrated learning by experience and demonstration algorithm

```
initialise:
    the candidate solution (ind) by selecting the best over 100 parameters' sets
        generated by using eq. 1
    the number of offspring ($\lambda$), see eq. 2
    the covariance matrix (covMat) with zeroed values
    the learning rate used to vary the individual ($\eta_{ind}$). I use $\eta_{ind}$=1.0.
    the update rate of the covariance matrix ($\eta_{covM}$), see eq. 3
repeat
    for $k = 1 \ldots \lambda$ do
        generate random variation vectors with a Gaussian distribution
            $s_k \sim \pi(\bullet \mid \theta)$
        generate a special sample <$S_d$> by training the candidate solution ind for
    25 trials
            for 25 trials do
                $w = w + backprop(ind)$
            end
            $\langle S_d \rangle = ind + w$
        where the update of the free parameters (w) is computed by means of the
    back-propagation algorithm. In particular, the batch mode described in section
    2.2 has been used
        generate samples by adding the product of the variation vectors by the
    exponential of the covariance matrix to the candidate solution
            $z_k = ind + e^{covMat} \cdot s_k$
```

evaluate the fitness of the samples

$$f_k = F(z_k)$$

end

sort z_k with respect to their performance f_k

$$ranking = z_{1,} \dots , z_\lambda , F(z_1) \geq \dots \geq F(z_\lambda)$$

calculate the utility of the samples u_k, $<S_d>$ is always ranked as second (the aim is at taking into consideration both the performance and the demonstration feedback)

u_k see eq. 4

sum-up variations weighted by utilities

$$\nabla_{ind} = \sum_k u_k \cdot s_k$$

calculate the estimated local gradient

$\nabla_{covM} = \sum_k u_k \cdot (s_k \cdot s_k^T - I)$, where I is the identity matrix, and T superscript represents the matrix transposition operation

move the candidate solution depending on the local gradient with the given learning rate

$$ind = ind + \eta_{ind} \cdot e^{covMat} \cdot \nabla_{ind}$$

update the covariance matrix based on the estimated local gradient with the given update rate

$$covMat = covMat + \eta_{covM} \cdot \nabla_{covM}$$

for *500 iterations*

Readers might replicate these experiments and the experiments described in section 4 by downloading and installing FARSA from https://sourceforge.net/projects/farsa/ and using the experimental plugins named KheperaExplorationExperiment e RobotBallApproachExperiment, respectively.

3.2. Results for the exploration experiment

In this section I report the results obtained in three series of experiments performed by using the three algorithms described above. For each experiment, I ran 10 replications starting with different randomly generated candidate solutions.

The performance displayed by the best robots at the end of the training process (Figure 3.2) shows that the robots trained with the integrated learning by experience and demonstration algorithm (E&D) outperform both the robots trained the learning by demonstration (D) algorithm (Mann-Whitney test, df = 10, $p < 0.01$) and the robots trained with the learning by experience (E) algorithm (Mann-Whitney test, df = 10, $p < 0.01$).

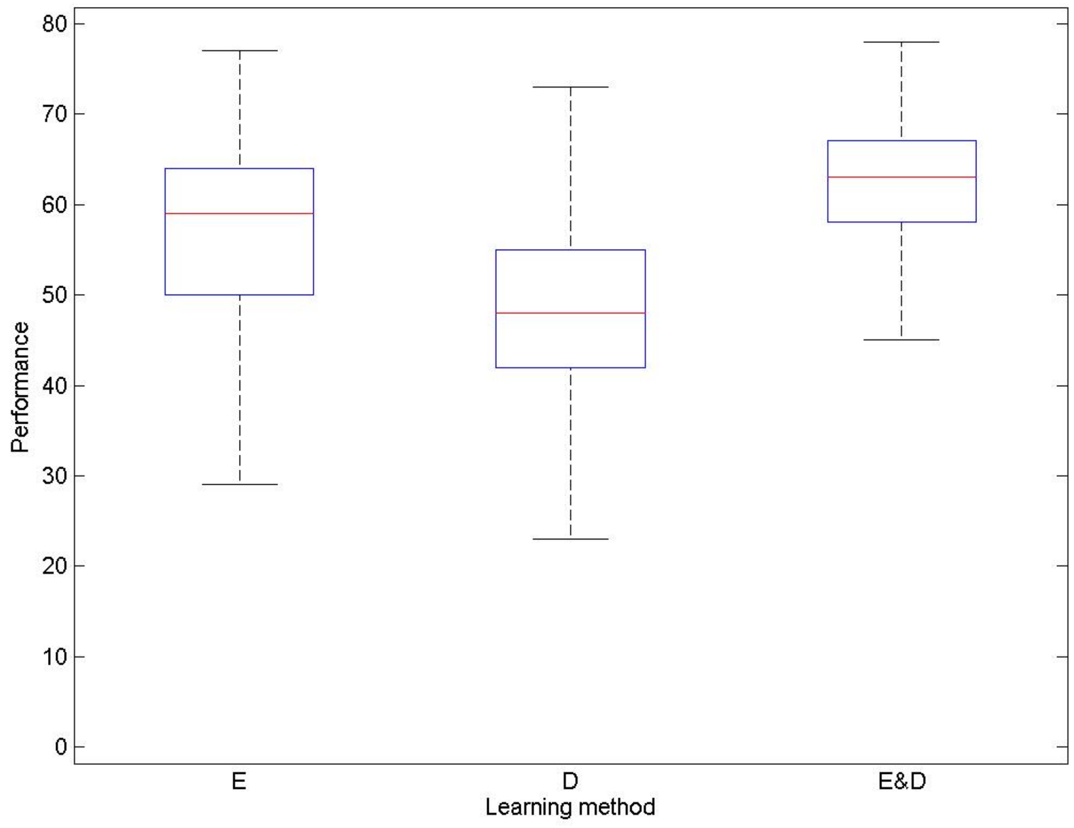

Figure 3.2. Boxplots of performance obtained, in the learning by experience (E), learning by demonstration (D) and integrated learning by experience and demonstration (E&D) experimental conditions. Each boxplot displays the performance obtained by post-evaluating the best 10 robots of each replication for 500 trials in environments including 5 obstacles. Boxes represent the inter-quartile range of the data, while the horizontal lines inside the boxes mark the median values. The whiskers extend to the most extreme data points within 1.5 times the inter-quartile range from the box. For the (D) condition I only report the result obtained in the offline mode that yielded better results than the online mode.

By analysing the behaviour displayed by the robots trained through the three learning algorithms (Figure 3.3) and the percentage of times these robots avoid obstacles by turning toward they preferred direction (Table 3.1) I can see that, as expected, the robot trained in the (D) and (E&D) conditions display a behaviour similar to the demonstration. The robots trained in the (E) condition, instead, exhibit a behaviour that differs qualitatively from the demonstration. In particular, they avoid the obstacles by turning always, or almost always, in the same direction. This qualitative difference plays a functional role with respect to the exploration task: indeed, avoiding obstacles by mostly turning in the same direction increases the probability to end up in previously explored locations of the environment compared to avoiding obstacles by turning in both directions.

The fact that the robots trained in the (E) experimental condition initially develop the ability to avoid obstacles by always turning in the same direction is not surprising. This simple strategy permits the robots to improve their performance with regard to the initial phase in which they are still unable to avoid obstacles. Yet, the acquisition of this strategy and its further refinement then blocks the learning process into a local minimum, i.e. a situation in which it is not possible to change the obstacle avoidance strategy without impacting negatively on performance.

The integrated learning by experience and demonstration algorithm (E&D) overcomes this local minima problem thanks to its ability to drive the learning process toward solutions that are similar to the demonstration, i.e. toward solutions that avoid obstacles by turning either left, or right, depending on the relative position of the obstacle.

The behaviour of the robot trained in the (E&D) condition is similar to the demonstrated behaviour concerning the direction, but not with regard to the trajectory followed to avoid obstacles (Figure 3.3). This trajectory depends on the relative reduction of the speed of the left or right wheel when the right or left infrared sensors are activated. In the case of the demonstrator robot, this relative reduction is proportional to the activation of the two left or of the two right infrared sensors. However, as you might remember, this relation is not optimised. It has been chosen simply because I, as the demonstrator designer, was unable to identify the best relation. The robots trained in the (E&D) experimental condition perform better than the demonstration in this respect (i.e. avoid the obstacles with a trajectory that maximise the exploration ability). Consequently, not only do they outperform the robots trained in the (E) condition, but they also have a more effective behaviour than the robots trained in the (D) condition (Figure 3.2).

Overall, this implies that the (E&D) algorithm leads the learning robots toward solutions that incorporate the functionally effective characteristics of the demonstrated behaviour, while deviate from the characteristics of the demonstrated behaviour that are not functional.

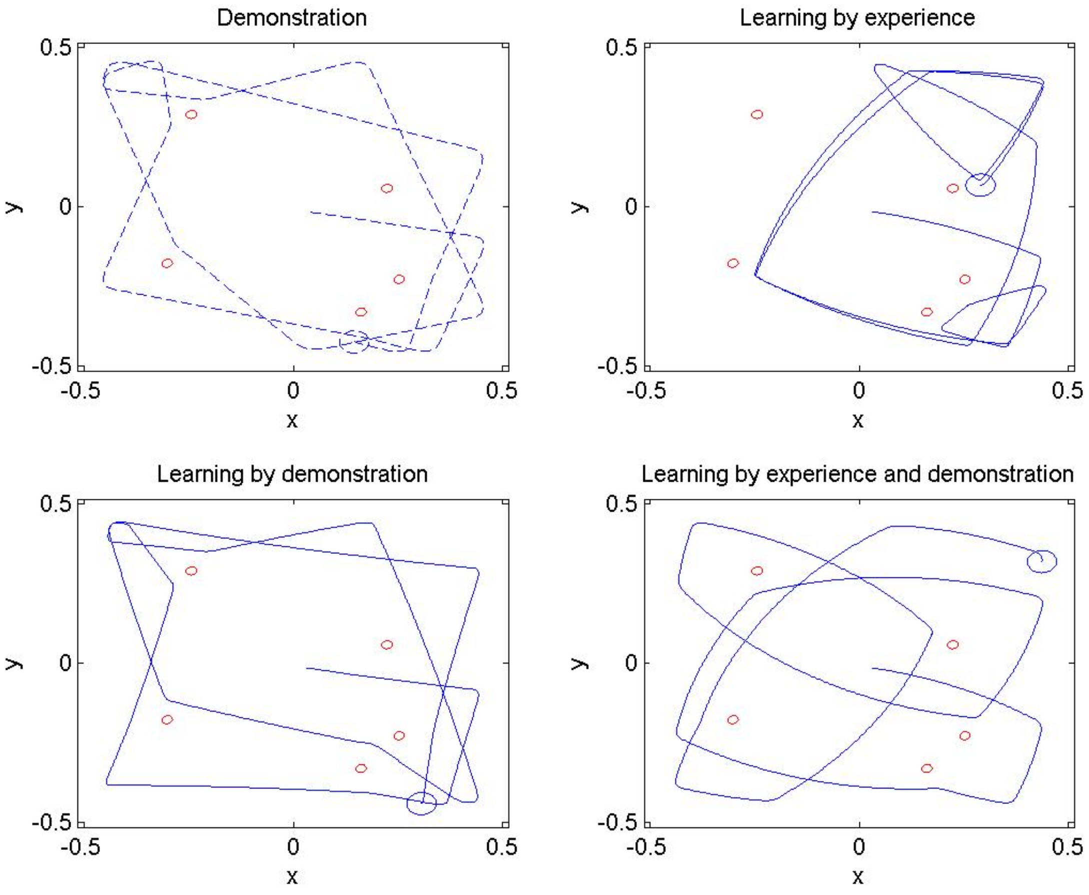

Figure 3.3. Trajectories displayed by the best robots of the three experimental conditions and by the demonstrator robot during a typical trial. To ensure that the trials shown are representative, they have been selected among the trials that produced a performance similar to the average performance obtained in each condition. The blue circle indicates the position of the robot at the end of the trial. The red circles indicate the position of the obstacles.

Best robot	**Performance**	**Turns in the preferred direction (%)**
E	60.65	100.0
D	51.36	2.4
E&D	61.90	0.8
Best 10 robots	**Performance**	**Turns in the preferred direction (%)**
E	55.09	90.94
D	44.25	11.72
E&D	61.28	0.88

Table 3.1. Performance and percentage of times the best robots turn in their preferred direction in the three experimental conditions. Some trained robots turn preferentially left, others preferentially right. Others turn both left and right, depending on the circumstances. These latter robots do not have a preferred direction, they turn left and right with similar probabilities. Top part: data for the best robot of all replications. Bottom part: average results of the 10 best robots of each replication.

3.3. Spatial positioning with heading experiment

In this section I report the results of a series of experiments carried out in a more complex scenario in which a robot should reach a 2D planar target position with a given target orientation (i.e. a position and orientation from which the red cylinder could be pushed toward the blue cylinder by simply moving forward, see Figure 3.4).

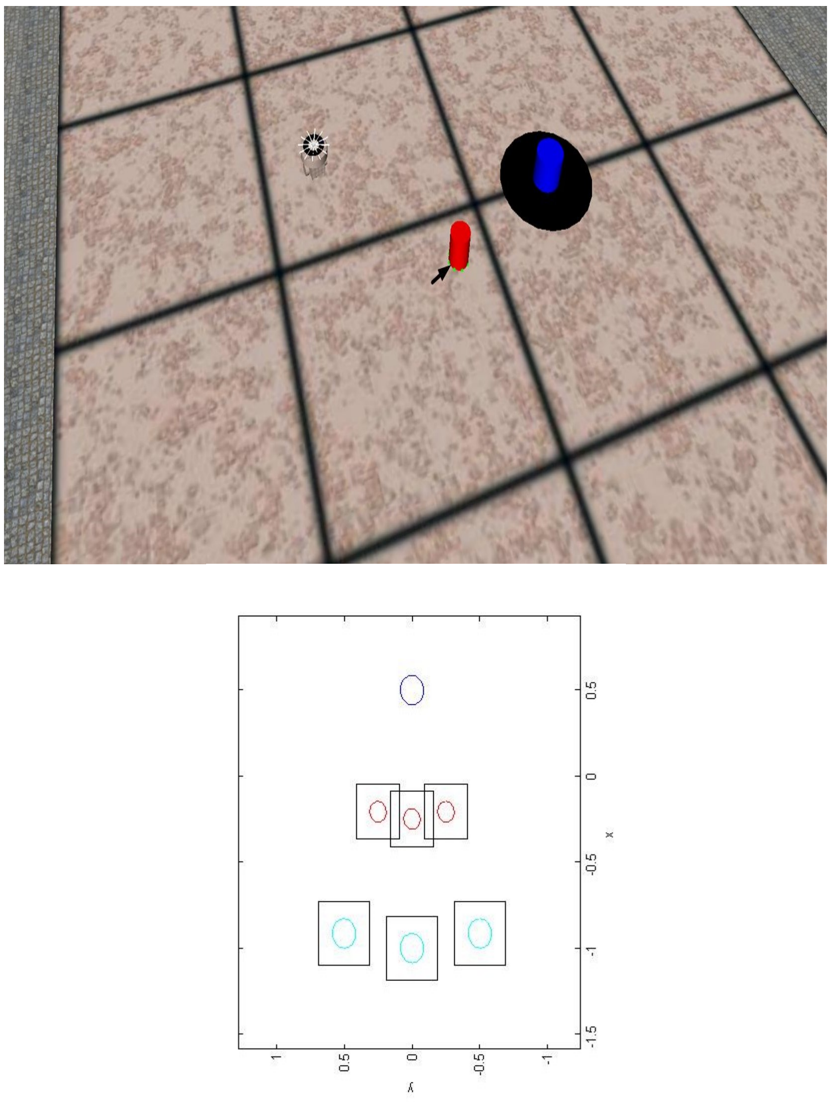

Figure 3.4. Spatial positioning with heading experiment. Top: The robot and the environment. The black arrow indicates the relative position and orientation that the robot should reach (i.e. the robot should reach the target with a relative position and orientation that could enable it to push the red object toward the blue object). **Bottom:** Exemplification of how the position of both the robot and the red cylinder varies among trials. The top circle indicates the blue cylinder. The intermediate and bottom rectangles indicate the areas in which the red cylinder and the robot are placed, respectively. The exact position of the red cylinder and of the robot is randomly chosen inside the selected rectangular area.

More specifically, the experiment concerns a simulated marXbot robot (see Bonani, Longchamp, Magnenat, Rétornaz, Burnier, Roulet and Mondada, 2010) placed in an arena containing a red and a blue cylindrical object. I used a different robotic platform in this second set of experiments since the marXbot is provided with an omnidirectional colour camera that enables it to detect the relative position of the two cylinders.

The robot is equipped with 24 infrared sensors evenly spaced around the robot body, an omni-directional camera, and two motors controlling the desired speed of the two wheels. The camera image is pre-processed by calculating the fraction of red and blue pixels present in each of the 12 30^{o}-sectors of a circular stripe portion of the image.

The robot's controller consists of a three-layer feed-forward neural network with 32 sensory neurons, 6 internal neurons, and 2 motor neurons. In particular, 8 sensory neurons encode the average activation state of 8 groups of 3 adjacent infrared sensors and 24 sensory neurons encode the percentage of red and blue pixels detected in the 12 sectors of the perceived image. The 2 motor neurons encode the desired speed of the two wheels. The activation of the sensors is normalised in the range [0.0, 1.0]. The activation of internal and motor neurons is computed on the basis of the logistic function. The activation of the motor neurons, i.e. the desired speed of the two wheels, is normalised in the range [-27, 27] cm/s. Internal and motor neurons are provided with biases.

The environment consists of a flat arena of 5.0x5.0m containing a red and a blue cylinder with a diameter of 12 and 17 cm respectively, randomly placed within 3 x 3 rectangular portion of the arena shown in Figure 3.4, bottom. In particular, the robot is initially placed within one of three possible rectangular portions (Figure 3.4 bottom, circles in the bottom rectangular areas). Similarly, the red cylinder occupies one of three possible rectangular portions (Figure 3.4 bottom, circles in the central rectangular areas). The initial location of the blue cylinder has been kept fixed over all the simulations (Figure 3.4 bottom, top circle). As in the case of the previous experiment, the robot and the environment are simulated by using FARSA (Massera, Ferrauto, Gigliotta and Nolfi, 2013 and 2014).

The performance of the robot is calculated by taking into account the offset between the target position and orientation of the robot and the actual position and orientation of the robot when it first touches the cylinder, or at the end of the trial. The target spatial parameters are defined as follows:

- the target position is located on the straight line passing over the centres of the bases of red and blue objects, outside the segment connecting these two points on the side of the red object, at a distance equal to the sum of the robot and red cylinder radius;
- the target orientation is given by the relative angle between the red and the blue cylinder.

More precisely, the fitness is calculated by averaging the product of the inverse of the position offset and the inverse of the angular offset, normalised in the range [0.0, 1.0], over trials:

$$Fitness = \frac{\sum_{t=1\ldots n_t} \left[(1 - \Delta P_t)(1 - \Delta O_t) \right]}{n_t}$$

where ΔP_t is the position offset normalised in the range [0.0, 1.0], ΔO_t is the angular offset normalised in the range [0.0, 1.0], t is the trial index, and n_t is the number of trials.

To select robots able to carry out the task in varying environmental conditions, I evaluated each robot for 9 trials each lasting up to 50 s (trials end either when the robot touches the red cylinder, or after 50 s). In the case of the Integrated Learning by Experience and Demonstration Algorithm (E&D), the back-propagation training has been also carried out for 9 trials during each iteration. At the beginning of each trial the red cylinder and the robot are in random locations selected within one of the 3x3 rectangular areas shown in (Figure 3.4, bottom), that are respectively at a distance of approximately 0.75 and 1.5 m from the blue cylinder.

As for the previous experiment, a demonstrator robot that operates on the basis of a hand-designed controller is used to generate demonstration data. The controller works by calculating, at each time step, a destination point situated along the straight line (*l*) connecting red and blue cylinders. The distance between the destination point and the red cylinder is directly proportional to the offset between the current robot position and *l*:

$$distance = K \cdot d_l$$

where *K* is a constant set to 2.5 and d_l is the distance in metres between the robot and *l* line. This is due to the fact that the time required for the robot to reach the target location depends on the initial angular offset between the robot orientation and the target orientation.

In particular, the coordinates of the destination point over planar surface are calculated according to the following equation:

$$destination = \left(distance \cdot \cos(\alpha), distance \cdot \sin(\alpha)\right)$$

where *α* represents the relative angle between the red and the blue cylinder.

The action to be performed by the demonstration robot at each time step is selected in order to reduce both the distance between the robot and the destination point, and the offset between the current and the target orientation of the robot. In more detail, the desired speed of the robots' wheels is set on the basis of the following equations:

$$speedLeftWheel = \left\{ \begin{array}{c} \frac{v_{max}}{2} \; if - 180 \leq \theta \leq 0 \\ \frac{v_{min}}{2} \cdot \left(\frac{\theta}{180}\right) + \frac{v_{max}}{2} \cdot \left(1 - \left(\frac{\theta}{180}\right)\right) \; if \; 0 < \theta \leq 180 \end{array} \right\}$$

$$speedRightWheel = \begin{cases} \frac{v_{max}}{2} \cdot \left(\frac{\theta}{180}\right) - \frac{v_{min}}{2} \cdot \left(1 + \left(\frac{\theta}{180}\right)\right) & if\ -180 \leq \theta < 0 \\ \frac{v_{max}}{2} & if\ 0 \leq \theta \leq 180 \end{cases}$$

where θ is the turning angle normalised in the range [-180°,180°]. θ is positive or negative depending on whether the destination point is located on the right or on the left of the robot's front.

This strategy enables the demonstrator robot to solve the task with a relative good performance. However, as for the previous experiment, the demonstration strategy is non-optimal. Indeed, in certain cases the robot might reach the destination point with a partially wrong orientation. Furthermore, in other cases it might spend an unnecessary amount of energy and time to reach the target position and orientation. Besides, I should consider that the learning robot might be unable to produce behaviours that match perfectly, or almost perfectly, the demonstrations as a consequence of limited precision of the robot sensory system. The control system of the demonstrator robot is not affected by this limitation since it operates on the basis of precise distance and angular measures extracted from the robot/environmental simulation.

The 212 connection weights and biases of the neural network are initially set randomly and then learned by means of the algorithms described above.

3.4. Results for the spatial positioning with heading experiment

Here I report the results obtained in three series of experiments in which the robots have been trained by using the algorithms described in subsections 3.1.1-3.1.3. Each experiment was replicated 10 times.

By analysing the performance displayed at the end of the training process I can see how, as for the exploration experiment, the robots trained in the integrated learning by experience and demonstration condition (E&D) outperform the robots trained in the learning by experience (E) and learning by demonstration (D) conditions (Mann-Whitney test, df = 10, $p < 0.01$). The difference between (E) and (D) conditions is also statistically significant (Mann-Whitney test, df = 10, $p < 0.01$). To verify robots' generalisation capabilities, I post-evaluated the best robots for 180 trials during which the distance between the robot and the blue cylinder was systematically varied in the range [62.5, 300] cm (see Figure 3.5). Also in this case the robots trained in the integrated learning by experience and demonstration condition (E&D) outperform the robots trained in the learning by experience (E) and learning by demonstration (D) conditions (Mann-Whitney test, df = 10, $p < 0.01$). The robots trained in the learning by experience (E) outperform the robot trained in the learning by demonstration (D) condition (Mann-Whitney test, df = 10, $p < 0.05$). The best (E&D) robots outperform the hand-designed demonstrator robot as well (Mann-Whitney test, df = 10, $p < 0.01$). The fact that the difference in performance between the (E&D) condition and the (D) and (E) conditions is more marked in the case of this second and more complex experiment might indicate that the (E&D) algorithm scale better with respect to the complexity of the task.

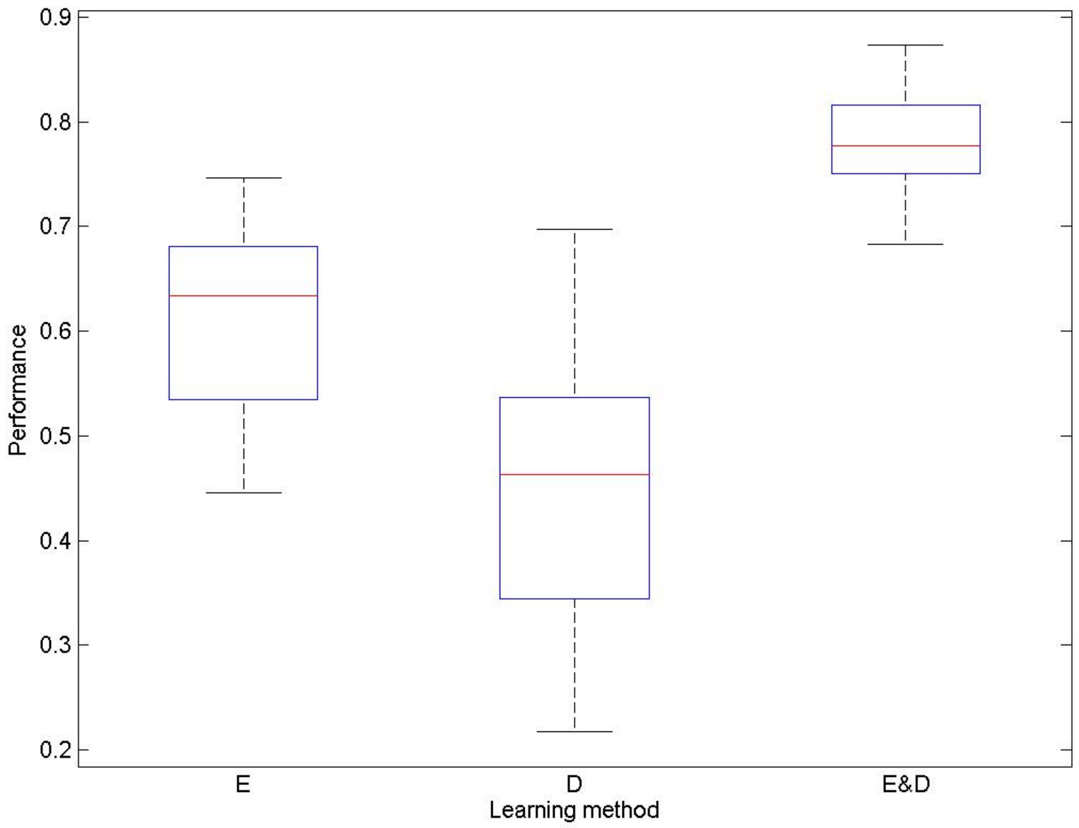

Figure 3.5. The boxplots display the performance carried out by the robots at the end of the training process in the learning by experience (E), learning by demonstration (D) and integrated learning by experience and demonstration (E&D). Boxes represent the inter-quartile range of the data, while the horizontal lines inside the boxes mark the median values. The whiskers extend to the most extreme data points within 1.5 times the inter-quartile range from the box. Data obtained by post-evaluating the best 10 robots of the 10 replications of each experimental condition for 180 trials. For the D condition I only report the result obtained in the online mode that yielded better results with respect to the offline mode.

To analyse the similarity between the behaviour displayed by trained robots and by the hand-designed demonstrator robot, I calculated the summed square error between the actions suggested by the demonstrator and the actions actually produced by the best robots trained in the three experimental conditions (see Figure 3.6). As expected, the difference between the behaviour of the robots and the demonstration is larger in the (E) condition than in the (D) and (E&D) conditions. Surprisingly, the differences with respect to the demonstration behaviour are greater in the (D) condition, in which the robots are trained for the ability to reproduce the demonstrated behaviour only, than in the (E&D) condition, in which the robots are also trained for the ability to maximise performance. This could be explained by the fact that the local minima affecting the outcome of the demonstration learning, that operates by minimising the offset between the demonstrated and the reproduced actions, play a minor role when the learning process is driven primarily by the attempt to maximise the long term effect of actions (i.e. the ability to reach the target with the appropriate position and orientation).

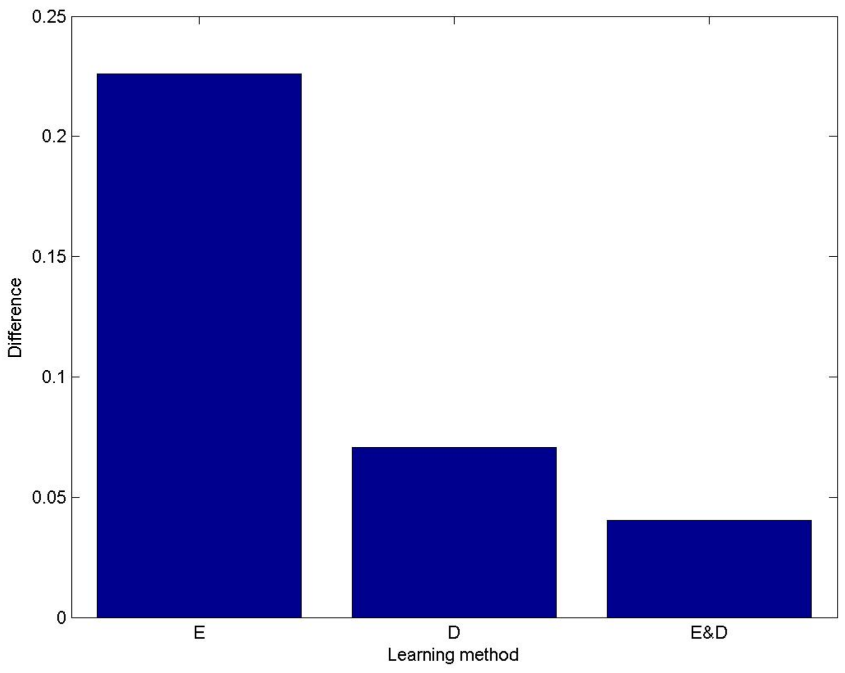

Figure 3.6. Average difference between the actions produced by the best robot trained in the three experimental conditions (E, D, and E&D) and the actions that would be produced by the demonstrator robot in the same robot/environmental conditions. Data are calculated by averaging the summed square difference between the actual and desired speeds of the two wheels, normalised in the range [0.0, 1.0], during 180 trials.

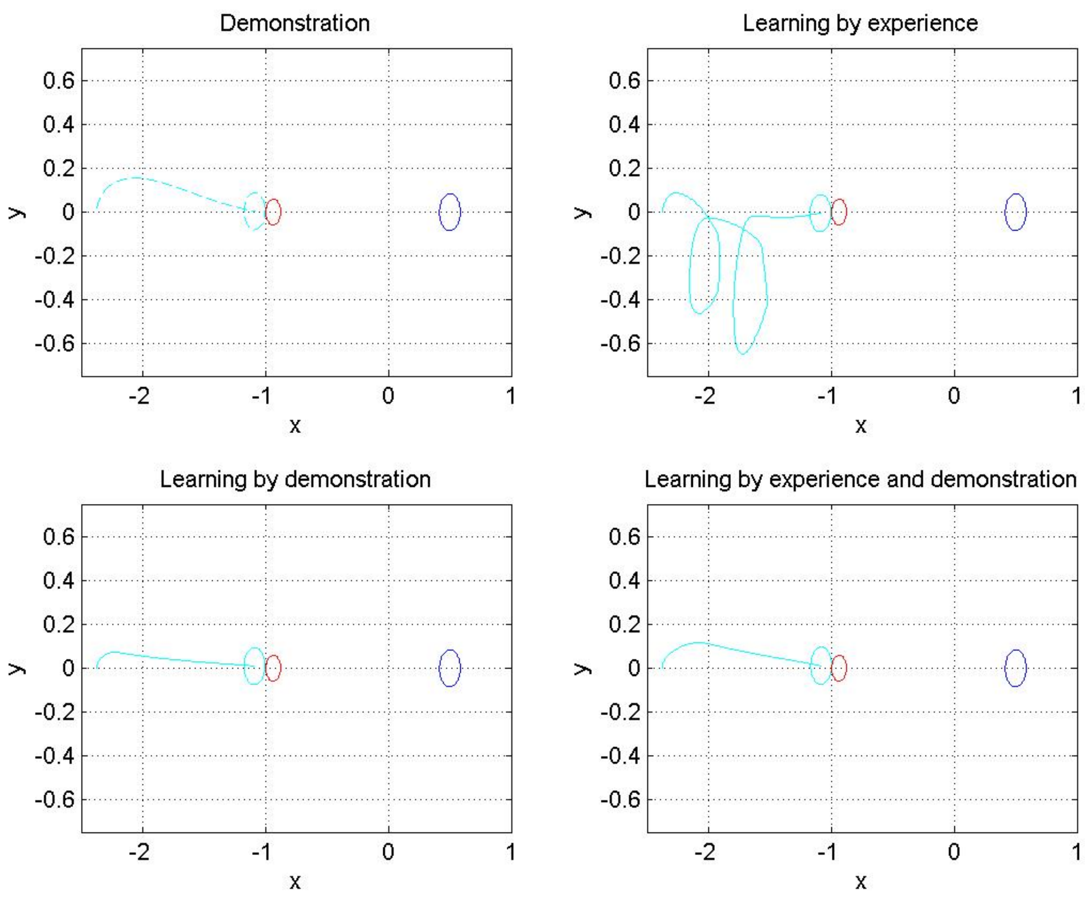

Figure 3.7. Trajectories displayed by the best robots obtained in the learning by experience (top-right), learning by demonstration (bottom-left) and learning by experience and demonstration condition (bottom-right) in a trial in which they start from the same initial position and orientation and in which the red cylinder is placed in the same position. The top-left picture shows the trajectory produced by the demonstrator robot. The red and blue circles represent the position of red and blue cylinder.

Figure 3.7 shows the typical trajectories displayed by the three best robots obtained in the three corresponding experimental conditions.

Not surprisingly the best (D) robot displays a behaviour that is qualitatively similar to the demonstration: the robot always approaches the red cylinder from the appropriate direction by producing a single curvilinear trajectory (Figure 3.7, left). Yet, the curvature of the trajectory of the (D) robot often differs significantly with respect to the curvature produced by the demonstrator robot, despite the learning error at the end of the training is very low (0.0042). This difference can be explained by considering that, as I mentioned above, small differences between the produced and the demonstrated behaviour tend to cumulate by producing significant differences over time. The other robots obtained in the remaining nine replications of the experiment display qualitatively similar behaviours (results not shown). The fact that the performance of the (D) robots are relatively low (Figure 3.5) can thus be explained by considering that the significant differences, that originate from the cumulative effect of small differences over time, have an effect on both the similarity with respect to the demonstrated behaviour and the performance. In some replications the functional effect of the difference is small, while in other cases it is large (Figure 3.5). The analysis of the relation between learning error and performance demonstrates that these two measures are lowly correlated. In other words, the robots with the highest performance are not necessarily those with the lower residual learning by demonstration error. This can be explained by considering that, in learning by demonstration, performance does not play any role. The minimisation of the error with respect to the demonstration, therefore, does not guarantee a maximisation of performance.

The behaviour produced by the best (E) robot is better than the one produced by the best (D) robot and differs widely from the demonstration (see Figure 3.6 and Figure 3.7). Indeed, it produces a series of curvilinear trajectories followed by sudden changes of directions until the robot manages to reach a relative position and orientation from which it is able to consistently move toward the red cylinder. Since the number of back-and-forth movements is highly variable and dependent on the circumstances, in some cases the robot is unable to reach the target destination in time. The robots of the other replications of the experiment show qualitatively similar behaviours (results not shown).

The best (E&D) robots approach the target by producing single curvilinear trajectories that are qualitatively similar to that displayed by the demonstrator robot (see

Figure 3.6 and Figure 3.7). Furthermore, their behaviours are even more similar to the demonstration than those shown by the (D) robots (Figure 3.6). Nonetheless, they manage to control the curvature of the trajectory in a way that enables them to achieve high performance and even outperform the demonstrator robot (the average performance of the demonstrator robot calculated over 500 trials is 0.79). Apparently, this is achieved by producing longer trajectories, with respect to the demonstrator, that allow the robot to reach the target destination with an orientation that matches better the target orientation. Overall, this implies that, also in this case, the (E&D) robots manage to find solutions that are qualitatively similar to the demonstrated behaviour, but deviate from the demonstration with respect to the aspects that are functionally sub-optimal.

3.5. Discussion

Standard learning methods rely on a single type of learning feedback and simply discard all other relevant information. For example, reinforcement learning methods are based on reward signals that indicate how well the agent is performing, but neglect other relevant cues such as: (i) perceived states that provide indications on why the current exhibited behaviour had only partial success (e.g. the kicked ball went too far or too left with respect to the target), (ii) advice provided by other agents (e.g. "more slowly"), and (iii) demonstrations of effective behaviours. Since the quality of the training feedback strongly affects the outcome of the learning process, the development of new learning methods capable of exploiting richer training feedbacks can potentially lead to significantly more powerful training techniques.

Although this research direction has started to be explored in the last few years, how different training feedbacks can be integrated in an effective manner still largely represents an open question.

As I stated at the beginning of this chapter, in order to appreciate the complexity of the problem, it is worth noting that different learning modes (e.g. learning by experience and learning by demonstration) often drive the learning agents toward different outcomes and that their effect is not necessarily additive. This entails that identifying an effective way to combine heterogeneous learning modes is far from trivial.

In this thesis I proposed a new method that allows to combine learning by demonstration and learning by experience feedbacks. The algorithm is shaped in a way ensuring that the learning process is constantly driven by both an objective performance measure, which estimates the overall ability of the robot to perform the task, and a learning by demonstration feedback used to steer the learning process toward solutions similar to the demonstration.

The analysis of the results obtained by testing this method on two qualitatively different problems indicates that my new integrated algorithm is indeed able to synthesise solutions that are (i) functionally better than those obtainable by using a single mode only, and (ii) qualitatively similar to the demonstrated behaviour. Besides, the integrated learning algorithm succeeds in synthesising solutions that incorporate the aspects of the demonstration that are functionally useful, whereas deviate with respect to other non-functional aspects.

In future works I plan to investigate the efficacy of the proposed method in task involving significant larger search space (i.e. a greater number of parameters to be set) that are particularly hard to tackle through learning by experience only.

Chapter 4. Conclusions

Conclusions

The overall objective of this thesis is to identify efficient ways to evolve neural network controllers. This general objective has been articulated in two sub-objectives concerning: (i) the comparison of alternative evolutionary algorithms, and (ii) the development of a hybrid algorithm that combines evolution and learning by demonstration. The state-of-the-art of research in these sub-areas has been reviewed in section 1.2 and 1.2.3, respectively. The research work carried out has been reported in Chapter 2 and 3, respectively.

The first objective of the work described in this thesis concerned the analysis of the efficacy of some of the most promising evolutionary methods in the context of neuroevolution, including two new algorithms that I developed. The selected techniques have been evaluated on the double-pole balancing task, a widely recognized benchmark problem in this area, and on extended versions of the problem. Previous comparative studies focused the analysis on the time required by evolution to find out sufficiently good solutions for the standard version of the problem. In this thesis, instead, I analysed what I called the adaptive power of alternative methods, namely the maximum fitness achieved by alternative methods, the ability to scale up to more complex versions of the problem, and the ability to generate solutions that are robust with respect to variations of the environmental conditions. For these reasons, instead of comparing the number of evaluations required to reach a given fitness threshold, I compared the fitness of the best solutions. Moreover, I designed two extended versions of the problem and I evaluated the evolving agents to solve the balancing problem from randomly different environmental conditions during multiple evaluation trials in which the initial conditions of the agents varied randomly.

The second objective of the thesis was to investigate the possibility to integrate two complementary processes, i.e. evolution and learning, so to exploit the combined effect of different kinds of feedbacks to drive the adaptive process. In particular, I considered a standard evolutionary process that realizes a form of adaptation through a trial-and-error mechanism and a learning by demonstration technique that represents a form of supervised learning. More specifically, I designed and implemented an integrated evolutionary algorithm that operates by estimating the local gradient of the current candidate solution with respect to an objective performance measure and by exploring preferentially variations that reduce the differences between the robot and the demonstrated behaviour. The combination of these two processes is appealing since it potentially allows the adaptive agent to exploit a richer training feedback constituted by both a scalar performance objective (reinforcement signal or fitness measure) and a detailed description of a suitable behaviour (demonstration). Another potential advantage consists in the possibility of observing proper behaviours (the demonstration), which frees the learning agent from the burden of identifying the behavioural strategy suitable for the task. Consequently, the aim of the learning process is limited to the discovery of how the demonstrated behaviour can be re-produced. The method proposed has been successfully evaluated on two qualitatively different robotic problems.

Overall, the results of the comparison of state-of-the-art evolutionary methods indicate that the xNES (Wierstra, Schaul, Peters and Schmidhuber, 2008; Wierstra, Schaul, Glasmachers, Sun, Peters and Schmidhuber, 2014) and the SSS methods outperform the other methods in the considered domain with respect to all the considered criteria, i.e. performance, robustness to environmental variations and capability to scale-up to more complex problems. The higher performance is not gained at cost of reduced speed, i.e. the best methods require few evaluations to discover sufficient good solutions. Furthermore, the higher the complexity of the task is, the higher the offset in performance between the different methods becomes. This implies that techniques that proved to be more effective are also more scalable.

The major innovation of the SSS method introduced in this thesis consists in the application of a variable quantity of noise to the performance measure that increases the probability of retaining less fit solutions. This, in turn, reduces the sensitivity to the issue of local minima. The analysis confirms my hypothesis about the importance of regulating the selective pressure for the evolution of efficient neural networks. Nonetheless, these results are limited to the family of pole balancing problems. In future research, these algorithms should be applied to different, and even more challenging, domains like evolutionary robotics.

With respect to the combination of evolution and learning, the algorithm that I proposed permits to achieve better performance than the two constituting algorithms used in isolation. By looking at the characteristics of the discovered solutions, the integrated algorithm finds out strategies that are (i) functionally better than those obtainable by using a single algorithm only, and (ii) qualitatively similar to the demonstrated behaviour. In other words, the algorithm is able to synthesise solutions that incorporate functionally useful aspects of the demonstrated behaviour but that do not necessarily replicate all aspects of it, hence leaving the learning agents free to discover their preferred way to solve the problem. This is important since demonstrations could be sub-optimal and/or the demonstrator might produce behaviours that cannot be replicated exactly by the learning agent (as a consequence of physical differences and/or limitations). Nonetheless, future research should clarify the importance of exposing agents to ineffective demonstrations (e.g., behaviours that do not allow solving the task or lead to poor results). Besides, the integrated algorithm should be evaluated on different domains, which require larger search space and are difficult to solve with a learning by experience approach.

4.1. Contribution to knowledge

Summarising the contributions to knowledge provided by this thesis, it is possible to underline the following points:

- design of new variations of the double-pole problems that can be used to more properly benchmark alternative algorithms;
- development of new methods for neuroevolution displaying state-of-the-art performance;
- discussion of suitable criteria for benchmarking in the context of neuro-evolution;
- detailed comparison of some of the most promising neuroevolutionary methods including the new methods introduced in this thesis;

- development of a new hybrid method which combines evolution and learning by demonstration;
- validation of such hybrid method on two qualitatively different robotic problems.

References

Abbeel, P., & Ng, A. Y. (2005). Exploration and Apprenticeship Learning in Reinforcement Learning. In Proceedings of the 22Nd International Conference on Machine Learning (pp. 1–8). New York, NY, USA: ACM.

Ackley, D., & Littman, M. (1991). Interactions between learning and evolution. Artificial Life II, 10, 487–509.

Ancel, L. W. (2000). Undermining the Baldwin Expediting Effect: Does Phenotypic Plasticity Accelerate Evolution? Theoretical Population Biology, 58(4), 307–319.

Anderson, R. W. (1995). Learning and evolution: A quantitative genetics approach. Journal of Theoretical Biology, 175(1), 89–101.

Argall, B. D., Browning, B., & Veloso, M. (2008). Learning robot motion control with demonstration and advice-operators. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 399–404).

Argall, Brenna D., Browning, B., & Veloso, M. M. (2010). Mobile Robot Motion Control from Demonstration and Corrective Feedback. In O. Sigaud & J. Peters (Eds.), From Motor Learning to Interaction Learning in Robots (pp. 431–450). Berlin, Heidelberg: Springer Berlin Heidelberg.

Argall, Brenna D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. Robotics and Autonomous Systems, 57(5), 469–483.

Beer, R. D., & Gallagher, J. C. (1992). Evolving Dynamical Neural Networks for Adaptive Behavior. Adaptive Behavior, 1(1), 91–122.

Belew, R. K., Mcinerney, J., & Schraudolph, N. N. (1990). Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In In (pp. 511–547). Addison-Wesley.

Beyer, H. (2007). Evolution strategies. Scholarpedia, 2(8), 1965.

Bianco, R., & Nolfi, S. (2004). Evolving the Neural Controller for a Robotic Arm Able to Grasp Objects on the Basis of Tactile Sensors. Adaptive Behavior, 12(1), 37–45.

Billard, A., Calinon, S., Dillmann, R., & Schaal, S. (2008). Robot Programming by Demonstration. In B. Siciliano & O. Khatib (Eds.), Springer Handbook of Robotics (pp. 1371–1394). Berlin, Heidelberg: Springer Berlin Heidelberg.

Billard, A. G., Calinon, S., & Guenter, F. (2006). Discriminative and adaptive imitation in uni-manual and bi-manual tasks. Robotics and Autonomous Systems, 54(5), 370–384.

Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., … Mondada, F. (2010). The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 4187–4193).

Bongard, J. C. (2002). Evolving modular genetic regulatory networks. In Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on (Vol. 2, pp. 1872–1877).

Bongard, J. C. (2013). Evolutionary Robotics. Commun. ACM, 56(8), 74–83.

Bongard, J. C., & Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (pp. 829–836). Morgan Kaufmann Publishers Inc.

Bongard, J. C. (2011). Morphological change in machines accelerates the evolution of robust behavior. Proceedings of the National Academy of Sciences of the United States of America, 108(4), 1234–1239.

Borenstein, E, Meilijson, I., & Ruppin, E. (2006). The effect of phenotypic plasticity on evolution in multipeaked fitness landscapes. Journal of Evolutionary Biology, 19(5), 1555—1570.

Borenstein, Elhanan, & Ruppin, E. (2003). Enhancing autonomous agents evolution with learning by imitation. Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour, 1(4), 335–348.

Calabretta, R., Nolfi, S., Parisi, D., & Wagner, G. P. (2000). Duplication of modules facilitates the evolution of functional specialization. Artificial Life, 6(1), 69–84.

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2009a). Evolving competitive car controllers for racing games with neuroevolution. In Proceedings of the 11th Annual conference on Genetic and evolutionary computation (pp. 1179–1186).

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2009b). Learning drivers for TORCS through imitation using supervised methods. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on (pp. 148–155).

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2010a). Applying cooperative coevolution to compete in the 2009 torcs endurance world championship. In Evolutionary Computation (CEC), 2010 IEEE Congress on (pp. 1–8).

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2010b). Learning to drive in the open racing car simulator using online neuroevolution. IEEE Transactions on Computational Intelligence and AI in Games, 2(3), 176–190.

Castilho, O., & Trujilo, L. (2005). Multiple Objective Optimization Genetic Algorithms For Path Planning In Autonomous Mobile Robots. Int. J. Comput. Syst. Signal, 6(1), 48–63.

Castillo, O., Trujillo, L., & Melin, P. (2007). Multiple Objective Genetic Algorithms for Path-planning Optimization in Autonomous Mobile Robots. Soft Computing, 11(3), 269–279.

Cerecedo-Cordoba, J. A., Barbosa, J. J. G., Terán-Villanueva, D., Frausto-Solís, J., & Flores, J. A. M. (2017). Use of Neuroevolution to Estimate the Melting Point of Ionic Liquids. International Journal of Combinatorial Optimization Problems and Informatics, 8(2), 2–9.

Chandra, A., & Yao, X. (2006). Ensemble learning using multi-objective evolutionary algorithms. Journal of Mathematical Modelling and Algorithms, 5(4), 417–445.

Chellapilla, K., & Fogel, D. B. (1999). Evolving neural networks to play checkers without relying on expert knowledge. IEEE Transactions on Neural Networks, 10(6), 1382–1391.

Chen, L., & Alahakoon, D. (2006). NeuroEvolution of augmenting topologies with learning for data classification. In Information and Automation, 2006. ICIA 2006. International Conference on (pp. 367–371).

Chernova, S., & Veloso, M. (2007). Confidence-based Policy Learning from Demonstration Using Gaussian Mixture Models. In Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (pp. 233:1–233:8). New York, NY, USA: ACM.

Chong, S. Y., Tan, M. K., & White, J. D. (2005). Observing the evolution of neural networks learning to play the game of Othello. IEEE Transactions on Evolutionary Computation, 9(3), 240–251.

Cliff, D., Husbands, P., & Harvey, I. (1993). Explorations in evolutionary robotics. Adaptive Behavior, 2(1), 73–110.

Coates, A., Abbeel, P., & Ng, A. Y. (2008). Learning for Control from Multiple Demonstrations. In Proceedings of the 25th International Conference on Machine Learning (pp. 144–151). New York, NY, USA: ACM.

Coello, C. A. C. (2006). Evolutionary multi-objective optimization: a historical view of the field. IEEE Computational Intelligence Magazine, 1(1), 28–36.

Coello Coello, C. A. (1999). A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. Knowledge and Information Systems, 1(3), 269–308.

Dale, M. (2018). Neuroevolution of hierarchical reservoir computers. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 410–417).

D'Ambrosio, D. B., Gauci, J., & Stanley, K. O. (2014). HyperNEAT: The First Five Years. In T. Kowaliw, N. Bredeche, & R. Doursat (Eds.), Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks (pp. 159–185). Berlin, Heidelberg: Springer Berlin Heidelberg.

Daniel, C., Neumann, G., & Peters, J. (2012). Hierarchical Relative Entropy Policy Search. In N. D. Lawrence & M. Girolami (Eds.), Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (Vol. 22, pp. 273–281). La Palma, Canary Islands: PMLR.

DasGupta, B., & Schnitger, G. (1992). Efficient approximation with neural networks: A comparison of gate functions. Pennsylvania State University, Department of Computer Science.

de Greeff, J., & Nolfi, S. (2010). Evolution of Implicit and Explicit Communication in Mobile Robots. In Stefano Nolfi & M. Mirolli (Eds.), Evolution of Communication and Language in Embodied Agents (pp. 179–214). Berlin, Heidelberg: Springer Berlin Heidelberg.

Deb, K. (2011). Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction.

DeWitt, T. J., Sih, A., & Wilson, D. S. (1998). Costs and limits of phenotypic plasticity. Trends in Ecology & Evolution, 13(2), 77–81.

Dima, C., Hebert, M., & Stentz, A. (2004). Enabling learning from large datasets: applying active learning to mobile robotics. In Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on (Vol. 1, pp. 108-114 Vol.1).

Doncieux, Stephane, & Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. Evolutionary Intelligence, 7(2), 71–93.

Doncieux, Stéphane, Mouret, J.-B., Bredeche, N., & Padois, V. (2011). Evolutionary Robotics: Exploring New Horizons. In Stéphane Doncieux, N. Bredèche, & J.-B. Mouret (Eds.), New Horizons in Evolutionary Robotics (pp. 3–25). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dopazo, H., Gordon, M. B., Perazzo, R., & Risau-Gusman, S. (2001). A model for the interaction of learning and evolution. Bulletin of Mathematical Biology, 63(1), 117–134.

Dürr, P., Mattiussi, C., & Floreano, D. (2006). Neuroevolution with Analog Genetic Encoding. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, & X. Yao (Eds.), Parallel Problem Solving from Nature - PPSN IX (pp. 671–680). Berlin, Heidelberg: Springer Berlin Heidelberg.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on (pp. 39–43).

Edgeworth, F. Y. (1881). Mathematical Psychics : An Essay on the Application of Mathematics to the Moral Sciences. London : Kegan Paul.

Endo, K., Yamasaki, F., Maeno, T., & Kitano, H. (2002). A method for co-evolving morphology and walking pattern of biped humanoid robot. In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on (Vol. 3, pp. 2775–2780).

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. Evolutionary Intelligence, 1(1), 47–62.

Floreano, D., Husbands, P., & Nolfi, S. (2008). Evolutionary Robotics. In B. Siciliano & O. Khatib (Eds.), Springer Handbook of Robotics (pp. 1423–1451). Berlin, Heidelberg: Springer Berlin Heidelberg.

Floreano, D., & Nolfi, S. (1997). Adaptive Behavior in Competing Co-Evolving Species. In Proceedings of the fourth European Conference on Artificial Life (pp. 378–387). MIT Press.

Floreano, D., & Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. Neural Networks, 13(4–5), 431–443.

Floreano, D., & Urzelai, J. (2001). Evolution of Plastic Control Networks. Autonomous Robots, 11(3), 311–317.

Fogel, D. B. (2001). Blondie24: Playing at the Edge of AI. Elsevier.

Fogel, D. B, Hays, T. J., Hahn, S. L., & Quon, J. (2004). A self-learning evolutionary chess program. Proceedings of the IEEE, 92(12), 1947–1954.

Fonseca, C. M., & Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation, 3(1), 1–16.

Fontanari, J. F., & Meir, R. (1991). Evolving a learning algorithm for the binary perceptron. Network: Computation in Neural Systems, 2(4), 353–359.

Gajer, M. (2009). Examining the Impact of Positive and Negative Constant Learning on the Evolution Rate. TASK Quarterly : Scientific Bulletin of Academic Computer Centre in Gdansk, Vol. 13, No 4, 355–362.

Gajer, M. (2012). Decelerating the rate of evolution with constant learning. Pomiary Automatyka Robotyka, R. 16, nr 11, 50–53.

Gauci, J., & Stanley, K. (2007). Generating Large-scale Neural Networks Through Discovering Geometric Regularities. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (pp. 997–1004). New York, NY, USA: ACM.

Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., & Schmidhuber, J. (2010). Exponential Natural Evolution Strategies. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (pp. 393–400). New York, NY, USA: ACM.

Goldberg, D. E., & Richardson, J. (1987). Genetic Algorithms with Sharing for Multimodal Function Optimization. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application (pp. 41–49). Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Gomes, J., Mariano, P., & Christensen, A. L. (2015). Devising Effective Novelty Search Algorithms: A Comprehensive Empirical Study. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (pp. 943–950). New York, NY, USA: ACM.

Gomes, J., Urbano, P., & Christensen, A. L. (2013). Evolution of swarm robotics systems with novelty search. Swarm Intelligence, 7(2–3), 115–144.

Gomez, F., & Mikkulainen, R. (1997). Incremental Evolution of Complex General Behavior. Adapt. Behav., 5(3–4), 317–342.

Gomez, F., & Schmidhuber, J. (2005). Evolving Modular Fast-Weight Networks for Control. In W. Duch, J. Kacprzyk, E. Oja, & S. Zadro\.zny (Eds.), Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005 (pp. 383–389). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gomez, F., Schmidhuber, J., & Miikkulainen, R. (2008). Accelerated Neural Evolution through Cooperatively Coevolved Synapses. Journal of Machine Learning Research, 937–965.

Gong, D.-W., Zhang, J., & Zhang, Y. (2011). Multi-objective Particle Swarm Optimization for Robot Path Planning in Environment with Danger Sources. JCP, 6, 1554–1561.

Gruau, F. (1994). Automatic Definition of Modular Neural Networks. Adapt. Behav., 3(2), 151–183.

Gruau, F., & Whitley, D. (1993). Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect. Evol. Comput., 1(3), 213–233.

Gruau, F., Whitley, D., & Pyeatt, L. (1996). A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In Proceedings of the 1st Annual Conference on Genetic Programming (pp. 81–89). Cambridge, MA, USA: MIT Press.

Handzel, Z. (2013). Modifying the rate of evolution using the learning process. Przegląd Elektrotechniczny, R. 89, nr 4, 50–52.

Hansen, N., & Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation, 9(2), 159–195.

Harvey, I. (2001). Artificial Evolution: A Continuing SAGA. In T. Gomi (Ed.), Evolutionary Robotics. From Intelligent Robotics to Artificial Life (pp. 94–109). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hayes, G., & Demiris, Y. (1995). A Robot Controller Using Learning by Imitation. Citeseer, 676.

Hinton, G. E., & Nowlan, S. J. (1987). How Learning Can Guide Evolution. Complex Systems, 1, 495–502.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press.

Hornby, G. S., & Pollack, J. B. (2002). Creating High-Level Components with a Generative Representation for Body-Brain Evolution. Artificial Life, 8(3), 223–246.

Hughes, E. J. (2003). Piece difference: Simple to evolve? In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 4, pp. 2470–2473).

Husbands, P., Harvey, I., Cliff, D., & Miller, G. (1994). The use of genetic algorithms for the development of sensorimotor control systems. In From Perception to Action Conference, 1994., Proceedings (pp. 110–121).

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies. In Evolutionary Computation, 2003. CEC '03. The 2003 Congress on (Vol. 4, pp. 2588-2595 Vol.4).

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2003). Learning Attractor Landscapes for Learning Motor Primitives. In S. Becker, S. Thrun, & K. Obermayer (Eds.), Advances in Neural Information Processing Systems 15 (pp. 1547–1554). MIT Press.

Israel, S., & Moshaiov, A. (2012). Bootstrapping Aggregate Fitness Selection with Evolutionary Multi-Objective Optimization. In Carlos A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, & M. Pavone (Eds.), Parallel Problem Solving from Nature - PPSN XII (pp. 52–61). Berlin, Heidelberg: Springer Berlin Heidelberg.

Jakobi, N. (1997). Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. Adaptive Behavior, 6(2), 325–368.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Morán, A. Moreno, J. J. Merelo, & P. Chacón (Eds.), Advances in Artificial Life (pp. 704–720). Berlin, Heidelberg: Springer Berlin Heidelberg.

Jin, Y., & Branke, J. (2005). Evolutionary Optimization in Uncertain Environments-a Survey. Trans. Evol. Comp, 9(3), 303–317.

Judah, K., Roy, S., Fern, A., & Dietterich, T. G. (2010). Reinforcement Learning via Practice and Critique Advice. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (pp. 481–486). Atlanta, Georgia: AAAI Press.

Khan, M. M., Khan, G. M., & Miller, J. F. (2010). Evolution of Optimal ANNs for Non-linear Control problems using Cartesian Genetic Programming. In In Proceedings of IEEE International Conference in Artificial Intelligence.

Kistemaker, S., & Whiteson, S. (2011). Critical Factors in the Performance of Novelty Search. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (pp. 965–972). New York, NY, USA: ACM. https://doi.org/10.1145/2001576.2001708

Knox, W. B., & Stone, P. (2009). Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. In Proceedings of the Fifth International Conference on Knowledge Capture (pp. 9–16). New York, NY, USA: ACM.

Knox, W. B., & Stone, P. (2010). Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. In Proceedings of the 9th International

Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1 (pp. 5–12). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238–1274.

Kober, J., & Peters, J. R. (2009). Policy Search for Motor Primitives in Robotics. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), Advances in Neural Information Processing Systems 21 (pp. 849–856). Curran Associates, Inc.

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9), 992–1007.

Koos S., Mouret J. B., and Doncieux S. (2010). Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In Proceedings of GECCO, pages 560–567.

Koos S., Mouret J. B., and Doncieux S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. In IEEE Transactions on Evolutionary Computation 17.1 (2013), pp. 122–145.

Koutník, J., Cuccu, G., Schmidhuber, J., & Gomez, F. (2013). Evolving large-scale neural networks for vision-based torcs.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press.

Lande, R. (2009). Adaptation to an extraordinary environment by evolution of phenotypic plasticity and genetic assimilation. Journal of Evolutionary Biology, 22(7), 1435—1446.

Lehman, J., & Miikkulainen, R. (2013). Neuroevolution. Scholarpedia, 8(6), 30977. https://doi.org/10.4249/scholarpedia.30977

Lehman, Joel, & Stanley, K. O. (2008). Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In Proc. of the Eleventh Intl. Conf. on Artificial Life (ALIFE XI). Cambridge, MA: MIT Press.

Lehman, Joel, & Stanley, K. O. (2010). Efficiently Evolving Programs Through the Search for Novelty. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (pp. 837–844). New York, NY, USA: ACM.

Lehman, Joel, & Stanley, K. O. (2011). Abandoning Objectives: Evolution Through the Search for Novelty Alone. Evol. Comput., 19(2), 189–223.

Leitner, J., Frank, M., Förster, A., & Schmidhuber, J. (2014). Reactive reaching and grasping on a humanoid: Towards closing the action-perception loop on the iCub. In Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on (Vol. 1, pp. 102–109).

Lima, J. A., Gracias, N., Pereira, H., & Rosa, A. (1996). Fitness function design for genetic algorithms in cost evaluation based problems. In Proceedings of IEEE International Conference on Evolutionary Computation (pp. 207–212).

Liu, S., & Iba, H. (2011). A Study on Computational Efficiency and Plasticity in Baldwinian Learning. JACIII, 15(9), 1300–1309.

Liu, Y., & Yao, X. (1996). Evolutionary design of artificial neural networks with different nodes. In Proceedings of IEEE International Conference on Evolutionary Computation (pp. 670–675).

Lucas, S. M., & Runarsson, T. P. (2006). Temporal difference learning versus co-evolution for acquiring othello position evaluation. In Computational Intelligence and Games, 2006 IEEE Symposium on (pp. 52–59).

Lund, H. H., & Miglino, O. (1996). From simulated to real robots. In Proceedings of IEEE International Conference on Evolutionary Computation (pp. 362–365).

Khan, M. M., Ahmad, M. A., Khan, M. G., & Miller, J. F. (2013). Fast Learning Neural Networks Using Cartesian Genetic Programming. Neurocomput., 121, 274–289.

Massera, G., Ferrauto, T., Gigliotta, O., & Nolfi, S. (2013). FARSA: An Open Software Tool for Embodied Cognitive Science. In ECAL.

Massera, G., Ferrauto, T., Gigliotta, O., & Nolfi, S. (2014). Designing adaptive humanoid robots through the FARSA open-source framework. Adaptive Behavior, 22(4), 255–265.

Mayley, G. (1996a). Landscapes, Learning Costs, and Genetic Assimilation. Evol. Comput., 4(3), 213–234.

Mayley, G. (1996b). The evolutionary cost of learning. In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (pp. 458–467).

Mayley, G. (1997). Guiding or hiding: Explorations into the effects of learning on the rate of evolution. In Proceedings of the Fourth European Conference on Artificial Life (Vol. 97, pp. 135–144).

Mery, F., & Kawecki, T. J. (2004). The effect of learning on experimental evolution of resource preference in Drosophila melanogaster. Evolution; International Journal of Organic Evolution, 58(4), 757–767.

Miikkulainen, R. (2010). Neuroevolution. In Encyclopedia of Machine Learning. New York: Springer. Retrieved from http://nn.cs.utexas.edu/?miikkulainen:encyclopedia10-ne

Miller, J. F., & Thomson, P. (2010). Cartesian Genetic Programming. In Proceedings of the Third European Conference on Genetic Programming (2000), vol. 1802, p. 121-132

Mitchell, M. (1998). An Introduction to Genetic Algorithms. Cambridge, MA, USA: MIT Press.

Mondada, F., Franzi, E., & Ienne, P. (1994). Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms. In The 3rd International Symposium on Experimental Robotics III (pp. 501–513). London, UK, UK: Springer-Verlag.

Moriarty, D. E., & Miikkulainen, R. (1994). Evolving neural networks to focus minimax search. In AAAI (pp. 1371–1377).

Moriarty, D. E., & Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. Connection Science, 7(3), 195–210.

Moriarty, D. E., & Miikkulainen, R. (1997). Forming Neural Networks Through Efficient and Adaptive Coevolution. Evol. Comput., 5(4), 373–399.

Moriarty, D. E., & Mikkulainen, R. (1996). Efficient Reinforcement Learning Through Symbiotic Evolution. Mach. Learn., 22(1–3), 11–32.

Mouret, J. B., & Doncieux, S. (2009). Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In 2009 IEEE Congress on Evolutionary Computation (pp. 1161–1168).

Mouret, J. B., & Doncieux, S. (2012). Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. Evolutionary Computation, 20(1), 91–133.

Mouret, J.-B., & Doncieux, S. (2008). Incremental Evolution of Animats' Behaviors as a Multi-objective Optimization. In M. Asada, J. C. T. Hallam, J.-A. Meyer, & J. Tani (Eds.), From Animals to Animats 10 (pp. 210–219). Berlin, Heidelberg: Springer Berlin Heidelberg.

Muñoz, J., Gutierrez, G., & Sanchis, A. (2009). Controller for torcs created by imitation. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on (pp. 271–278).

Muñoz, J., Gutierrez, G., & Sanchis, A. (2010). A human-like TORCS controller for the Simulated Car Racing Championship. In Computational Intelligence and Games (CIG), 2010 IEEE Symposium on (pp. 473–480).

Nelson, A. L., Grant, E., Barlow, G., & White, M. (2003). Evolution of complex autonomous robot behaviors using competitive fitness. In Integration of Knowledge Intensive Multi-Agent Systems, 2003. International Conference on (pp. 145–150).

Nelson, Andrew L., Barlow, G. J., & Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. Robotics and Autonomous Systems, 57(4), 345–370.

Nelson, Andrew L., & Grant, E. (2006). Using direct competition to select for competent controllers in evolutionary robotics. Robotics and Autonomous Systems, 54(10), 840–857.

Nelson, Andrew L, Grant, E., & Lee, G. K. (2002). Using Genetic Algorithms to Capture Behavioral Traits Exhibited by Knowledge Based Robot Agents. In CAINE (pp. 92–97).

Nolfi, S. (1998). Evolutionary robotics: exploiting the full power of self-organization. IET Conference Proceedings, 3-3(1).

Nolfi, S., Miglino, O., & Parisi, D. (1994). Phenotypic plasticity in evolving neural networks. In From Perception to Action Conference, 1994., Proceedings (pp. 146–157).

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: A garbage collecting robot. Robotics and Autonomous Systems, 22(3–4), 187–198.

Nolfi, S. (2001). Learning and Evolution in Neural Networks.

Nolfi, S., & Floreano, D. (1998). Coevolving predator and prey robots: Do “arms races” arise in artificial evolution? Artificial Life, 4(4), 311–335.

Nolfi, S., & Floreano, D. (1999). Learning and Evolution. Autonomous Robots, 7(1), 89–113.

Nolfi, S., & Floreano, D. (2000). Evolutionary Robotics: The Biology,Intelligence,and Technology. Cambridge, MA, USA: MIT Press.

Nolfi, S., & Parisi, D. (1996). Learning to Adapt to Changing Environments in Evolving Neural Networks. Adaptive Behavior, 5(1), 75–98.

Nolfi, S., Parisi, D., & Elman, J. L. (1994). Learning and Evolution in Neural Networks. Adapt. Behav., 3(1), 5–28.

Paenke, I., Jin, Y., & Branke, J. (2009). Balancing Population- and Individual-Level Adaptation in Changing Environments. Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems, 17(2), 153–174.

Paenke, I., Kawecki, T. J., & Sendhoff, B. (2009). The Influence of Learning on Evolution: A Mathematical Framework. Artif. Life, 15(2), 227–245.

Paenke, I., Sendhoff, B., & Kawecki, T. J. (2007). Influence of plasticity and learning on evolution under directional selection. The American Naturalist, 170(2), E47—58.

Paenke, I., Sendhoff, B., Rowe, J., & Fernando, C. (2007). On the Adaptive Disadvantage of Lamarckianism in Rapidly Changing Environments. In F. Almeida e Costa, L. M. Rocha, E. Costa, I. Harvey, & A. Coutinho (Eds.), Advances in Artificial Life (pp. 355–364). Berlin, Heidelberg: Springer Berlin Heidelberg.

Paraschos, A., Daniel, C., Peters, J. R., & Neumann, G. (2013). Probabilistic Movement Primitives. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q.

Weinberger (Eds.), Advances in Neural Information Processing Systems 26 (pp. 2616–2624). Curran Associates, Inc.

Pareto, V. (1896). Cours D'Economie Politique, vol. I and II, F. Rouge, Lausanne, 1776–1960.

Parisi, D., Cecconi, F., & Nolfi, S. (1990). Econets: Neural networks that learn in an environment. Network: Computation in Neural Systems, 1(2), 149–168.

Petrosino, G., Parisi, D., & Nolfi, S. (2013). Selective attention enables action selection: evidence from evolutionary robotics experiments. Adaptive Behavior, 21(5), 356–370.

Philippides, A., Husbands, P., Smith, T., & O'Shea, M. (2005). Flexible Couplings: Diffusing Neuromodulators and Adaptive Robotics. Artif. Life, 11(1–2), 139–160.

Pigliucci, M. (2005). Evolution of phenotypic plasticity: where are we going now? Trends in Ecology & Evolution, 20(9), 481–486.

Price, T. D., Qvarnström, A., & Irwin, D. E. (2003). The role of phenotypic plasticity in driving genetic evolution. Proceedings of the Royal Society of London B: Biological Sciences, 270(1523), 1433–1440.

Rechenberg, I. (1973). Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart-Bad Cannstatt: Frommann-Holzboog.

Richards, M. (2008). Does learning accelerate evolution? Work, 34.

Risi, S., & Togelius, J. (2017). Neuroevolution in Games: State of the Art and Open Challenges. IEEE Transactions on Computational Intelligence and AI in Games, 9(1), 25–41.

Risi, S., Hughes, C. E., & Stanley, K. O. (2010). Evolving plastic neural networks with novelty search. Adaptive Behavior, 18(6), 470–491.

Risi, S., Vanderbleek, S. D., Hughes, C. E., & Stanley, K. O. (2009). How Novelty Search Escapes the Deceptive Trap of Learning to Learn. In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (pp. 153–160). New York, NY, USA: ACM.

Rosenstein, M. T., Barto, A. G., Si, J., Barto, A., Powell, W., & Wunsch, D. (2004). Supervised Actor-Critic Reinforcement Learning. Handbook of Learning and Approximate Dynamic Programming, 359–380.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. In David E. Rumelhart, J. L. McClelland, & C. PDP Research Group (Eds.) (pp. 318–362). Cambridge, MA, USA: MIT Press.

Runarsson, T. P., & Lucas, S. M. (2014). Preference learning for move prediction and evaluation function approximation in Othello. IEEE Transactions on Computational Intelligence and AI in Games, 6(3), 300–313.

Russell, S., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Saito, N., Ishihara, S., & Kaneko, K. (2013). Baldwin effect under multipeaked fitness landscapes: phenotypic fluctuation accelerates evolutionary rate. Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics, 87(5), 052701.

Saravanan, N., & Fogel, D. B. (1995). Evolving Neural Control Systems. IEEE Expert: Intelligent Systems and Their Applications, 10(3), 23–27.

Sasaki, T., & Tokoro, M. (1997). Adaptation toward changing environments: Why darwinian in nature. In Fourth European conference on artificial life (Vol. 4, p. 145). MIT Press.

Savastano, P., & Nolfi, S. (2012). Incremental learning in a 14 dof simulated icub robot: Modeling infant reach/grasp development. In Conference on Biomimetic and Biohybrid Systems (pp. 250–261).

Schaal, S. (1997). Learning from demonstration. In Advances in Neural Information Processing Systems 9 (pp. 1040–1046). Cambridge, MA: MIT Press.

Schaal, S., Mohajerian, P., & Ijspeert, A. (2007). Dynamics systems vs. optimal control--a unifying view. Progress in Brain Research, 165, 425–445.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu P. and Sutphen, S. (2007). Checkers is solved. Science, 317(5844), 1518–1522.

Schembri, M., Mirolli, M., & Baldassarre, G. (2007). Evolution and Learning in an Intrinsically Motivated Reinforcement Learning Robot. In F. Almeida e Costa, L. M. Rocha, E. Costa, I. Harvey, & A. Coutinho (Eds.), Advances in Artificial Life (pp. 294–303). Berlin, Heidelberg: Springer Berlin Heidelberg.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by evolino. Neural Computation, 19(3), 757–779.

Schmidhuber, J., Wierstra, D., & Gomez, F. J. (2005). Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction. In Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI).

Schwefel, H. P. (1995). Evolution and Optimum Seeking. Wiley Interscience.

Schwefel, H. P. (1977). Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Vol. 26). Basel/Stuttgart: Birkhaeuser.

Silva, F., Duarte, M., Correia, L., Oliveira, S. M., & Christensen, A. L. (2016). Open Issues in Evolutionary Robotics. Evolutionary Computation, 24(2), 205–236.

Sims, K. (1994). Evolving 3D Morphology and Behavior by Competition. Artificial Life, 1(4), 353–372.

Solteiro Pires, E. J., Tenreiro Machado, J. A., & de Moura Oliveira, P. B. (2004). Robot Trajectory Planning Using Multi-objective Genetic Algorithm Optimization. In K. Deb (Ed.), Genetic and Evolutionary Computation – GECCO 2004 (pp. 615–626). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sperati, V., Trianni, V., & Nolfi, S. (2008). Evolving coordinated group behaviours through maximisation of mean mutual information. Swarm Intelligence, 2(2–4), 73–95.

Stanley, K. O. (2007). Compositional Pattern Producing Networks: A Novel Abstraction of Development. Genetic Programming and Evolvable Machines, 8(2), 131–162.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2003). Evolving adaptive neural networks with and without adaptive synapses. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 4, pp. 2557–2564).

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A Hypercube-based Encoding for Evolving Large-scale Neural Networks. Artif. Life, 15(2), 185–212.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies. Evol. Comput., 10(2), 99–127.

Stanley, K. O., & Miikkulainen, R. (2004). Competitive Coevolution Through Evolutionary Complexification. J. Artif. Int. Res., 21(1), 63–100.

Sutton, R. S., & Barto, A. G. (1998). Introduction to Reinforcement Learning (1st ed.). Cambridge, MA, USA: MIT Press.

Sutton, R. S. (1984). Temporal Credit Assignment in Reinforcement Learning (PhD Thesis). University of Massachusetts Amherst.

Suzuki, R., & Arita, T. (2007). The Dynamic Changes in Roles of Learning through the Baldwin Effect. Artificial Life, 13(1), 31–43.

Suzuki, R., & Arita, T. (2013). A simple computational model of the evolution of a communicative trait and its phenotypic plasticity. Journal of Theoretical Biology, 330, 37–44.

Sznajder, B., Sabelis, M. W., & Egas, M. (2012). How Adaptive Learning Affects Evolution: Reviewing Theory on the Baldwin Effect. Evolutionary Biology, 39(3), 301–310.

Taylor, M. E., Suay, H. B., & Chernova, S. (2011). Integrating Reinforcement Learning with Human Demonstrations of Varying Ability. In The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2 (pp. 617–624). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Tikhanoff, V., Pattacini, U., Natale, L., & Metta, G. (2013). Exploring affordances and tool use on the iCub. In Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on (pp. 130–137).

Trianni, V., & López-Ibánez, M. (2014). Advantages of multi-objective optimisation in evolutionary robotics: Survey and case studies. Technical report TR/IRIDIA/2014-014, IRIDIA, Université Libre de Bruxelles, Belgium.

Tuci, E., Quinn, M., & Harvey, I. (2002). An Evolutionary Ecological Approach to the Study of Learning Behavior Using a Robot-Based Model. Adaptive Behaviour, 10, 201–221.

Turner, A. J., & Miller, J. F. (2013). The importance of topology evolution in neuroevolution: a case study using cartesian genetic programming of artificial neural networks. In Research and Development in Intelligent Systems XXX (pp. 213–226).,

Turner, A. J., & Miller, J. F. (2014). NeuroEvolution: Evolving Heterogeneous Artificial Neural Networks. Evolutionary Intelligence, 7, 135–154.

Uc-Cetina, V. (2007). Supervised reinforcement learning using behavior models. In Sixth International Conference on Machine Learning and Applications (ICMLA 2007) (pp. 336–341).

Valenti, M. (2013). Learning of Manipulation Capabilities in a Humanoid Robot. Master Thesis, School of Engineering, Università degli Studi di Roma, La Sapienza, Italy.

Valsalam, V. K., & Miikkulainen, R. (2008). Modular neuroevolution for multilegged locomotion. In Proceedings of the 10th annual conference on Genetic and evolutionary computation (pp. 265–272).

Van Veldhuizen, D. A., & Lamont, G. B. (2000). Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. Evolutionary Computation, 8(2), 125–147.

Verbancsics, P., & Harguess, J. (2015). Image classification using generative neuro evolution for deep learning. In Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on (pp. 488–493).

Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Computing, 14(3), 347–361.

Whitley, Darrell. (2001). An overview of evolutionary algorithms: practical issues and common pitfalls. Information and Software Technology, 43(14), 817–831.

Whitley, Darrell, Gordon, V. S., & Mathias, K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. In Y. Davidor, H.-P. Schwefel, & R. Männer (Eds.), Parallel Problem Solving from Nature — PPSN III (pp. 5–15). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wieland, A. P. (1991). Evolving neural network controllers for unstable systems. In IJCNN-91-Seattle International Joint Conference on Neural Networks (Vol. ii, pp. 667–673 vol.2).

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural Evolution Strategies. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) (pp. 3381–3387).

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., & Schmidhuber, J. (2014). Natural Evolution Strategies. J. Mach. Learn. Res., 15(1), 949–980.

Wiles, J., Watson, J., Tonkes, B., & Deacon, T. (2005). Transient Phenomena in Learning and Evolution: Genetic Assimilation and Genetic Redistribution. Artificial Life, 11(1–2), 177–188.

Wund, M. A. (2012). Assessing the impacts of phenotypic plasticity on evolution. Integrative and Comparative Biology, 52(1), 5—15.

Yamauchi, B. M., & Beer, R. D. (1994). Sequential Behavior and Learning in Evolved Dynamical Neural Networks. Adaptive Behavior, 2(3), 219–246.

Yao, X. (1994). The Evolution of Connectionist Networks. In T. Dartnall (Ed.), Artificial Intelligence and Creativity: An Interdisciplinary Approach (pp. 233–243). Dordrecht: Springer Netherlands.

Yao, X. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423–1447.

You, Y., Pan, X., Wang, Z., & Lu, C. (2017). Virtual to real reinforcement learning for autonomous driving. ArXiv Preprint ArXiv:1704.03952.

Zagal, J.C., Ruiz-del-Solar, J., Vallejos, P. (2004). Back to reality: crossing the reality gap in evolutionary robotics. In Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles, IAV 2004. Elsevier, Amsterdam, The Netherlands (2005)

Zitzler, E., Laumanns, M., & Bleuler, S. (2004). A Tutorial on Evolutionary Multiobjective Optimization. In X. Gandibleux, M. Sevaux, K. Sörensen, & V. T'kindt (Eds.), Metaheuristics for Multiobjective Optimisation (pp. 3–37). Berlin, Heidelberg: Springer Berlin Heidelberg.