SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA

TESI DI LAUREA MAGISTRALE
IN
ARTIFICIAL INTELLIGENCE AND ROBOTICS

Learning of manipulation capabilities in a humanoid robot

Relatore interno:
Prof. Giuseppe Oriolo

Relatore esterno:
Dr. Stefano Nolfi

Controrelatore:
Prof. Daniele Nardi

Laureando:
Manlio Valenti
Matricola 1485188

A. A. 2012-2013

Contents

Nomenclature

D	number of inputs
K	number of hiddens
M	number of outputs
N	number of weights
P	number of patters
$\mathbf{p}_i$	i-th stored pattern
$\mathbf{t}_i$	desired outputs when applying $\mathbf{p}_i$ to the network
x_{ji}	input from neuron i to neuron j
w_{ji}	weight from neuron i to neuron j
t_{mp}	desired output of neuron m when applying the pth pattern
o_{mp}	actual output of neuron m when applying the pth pattern
$net_j = \sum_i w_{ji} x_i$	weighted sum of inputs for unit j
τ	the anticipation term during the training

Introduction

Over the last decades, scientific research showed an increasing interests over bio-inspired approaches. Considering the great effectiveness of biological systems in solving a wide variety of different problems as well as their intrinsic capability of self-adaptation, it is not surprising that attempts in reproducing such kind of systems have been driven by a strong motivation. However, design and formalize such kind of architecture is an extremely complex task; taking inspiration from the "working" example of living beings, the feeling that the control system should be the result of a *simulated evolution* started to spread, rapidly becoming an important field of machine learning. Most of evolutionary experiments exploit genetic algorithms to develop a control system based on artificial neural networks, both for the same biological propensity and for their effectiveness in solving a wide variety of different problems.

Despite the huge potential of this approach and the robustness of obtained solutions, it has a potentially severe drawback: time. As natural evolution is a slow process, so also genetic algorithms tend to require a lot of time before converging. This is also due to the type of informations exploited during the evolution: individuals are, in fact, selected or discarded according to their *fitness*, i.e. a numerical evaluation of their overall performance. It is not trivial, however, to understand the reason of the failure and operate a "focused" correction. In natural world there are many evidences that show how offsprings tend to exploit, at the same time, a wide variety of different feedback: a typical example is the case of a kid that tries to grasp an object. The outcome of the action provides a feedback on whether the movement was correct or not, but no hints about the needed correction; additional informations

usually come from other sources, like corrections made from the parents, that can guide the child and identify the errors as well as providing successful examples.

These observations led to the idea that exploiting different kind of informations can significantly reduce the learning time and increase the effectiveness of the results. However, current trial-and-error based learning algorithms (like genetic algorithms or reinforcement learning) usually consider a single overall evaluation of the performances, being highly inefficient when applied to real world scenarios. Combining different algorithms (each based on a different type of feedback) can, instead, be the key to develop more efficient forms of learning.

In this work we chose to provide the system with some successful examples of how the task should be accomplished. This was realized by applying a preliminary phase where the system is trained by demonstration: programming by demonstration is a very natural and powerful way to train a given system (typically a robot) in order to solve a particular task. It is a central and very active topic in robotic researches, not only because it would simplify the problem of hand-coding a program to perform the task, but also because it will make the training extremely flexible and well-suited for users without experience in programming or machine learning.

Although this solution may constrain the evolution towards a local minimum (and, for this reason, might attract criticism from those who suggests a pure evolutionary approach), we claim that, for complex tasks, searching the entire hypotheses space is not feasible with the current technology as it would require too much time. Of course, we need to rely on the fact that the demonstration will provide a good-enough starting point; this is not, however, a far-fetched idea: there are a lot of examples in natural world where "students" learn from "teachers" by combining a phase of "imitation" with a phase of refinement through trial and error. A classical example is the one of the tennis teacher, who initially performs a sequence of movements to hit the ball. The beginner student will find those movements unusual and will be able to reproduce the movement as precisely only after trying many times.

The aim of this work is to train the humanoid robot iCub (Metta et al. (2008)) to accomplish manipulation tasks. The robot will first be trained using a kinesthetic

teaching phase (i.e. a demonstration phase where the robot is physically moved by the teacher) with a successive phase where a genetic algorithm is applied to improve the robot's ability to achieve the desired goal. Experiments have been conducted in a simulated scenario in which the robot has to acquire and show the capability of reaching and touching a ball, of grasping it and of moving it in a different location.

Chapter 1 presents an overview of the *learning* problem, describing more in details the field of learning by demonstration and evolutionary robotics. Chapters 2 and 3 will be dedicated to a more formal and detailed description of artificial neural networks and genetic algorithms respectively. Chapter 4 will be fully dedicated to the description of the experiments and the obtained results.

Chapter 1

Learning

Computer science is no more about
computers than astronomy is about telescopes
Edsger Dijkstra

One of the most fascinating and interesting problem from the birth of computers is how to make them learn and improve the performance exploiting past experience. Over the years, the topic attracted a lot of scientists and researchers from different fields, and now many different approaches have been developed, showing significative performances in solving a wide variety of problems, from object recognition to information retrieval and text analysis, as well as path planning and game playing. Still computer programs are far from demonstrating the same learning effectiveness and efficiency of human beings, remaining often context-dependent.

Machine Learning is the branch of computer science that is aimed at building and studying algorithms able to identify and detect general patterns in data, so to use them when dealing with future problems, even if concerning unexperienced data. There are two main categories of machine learning approaches, which will be briefly described in the following.

1.1 Unsupervised learning

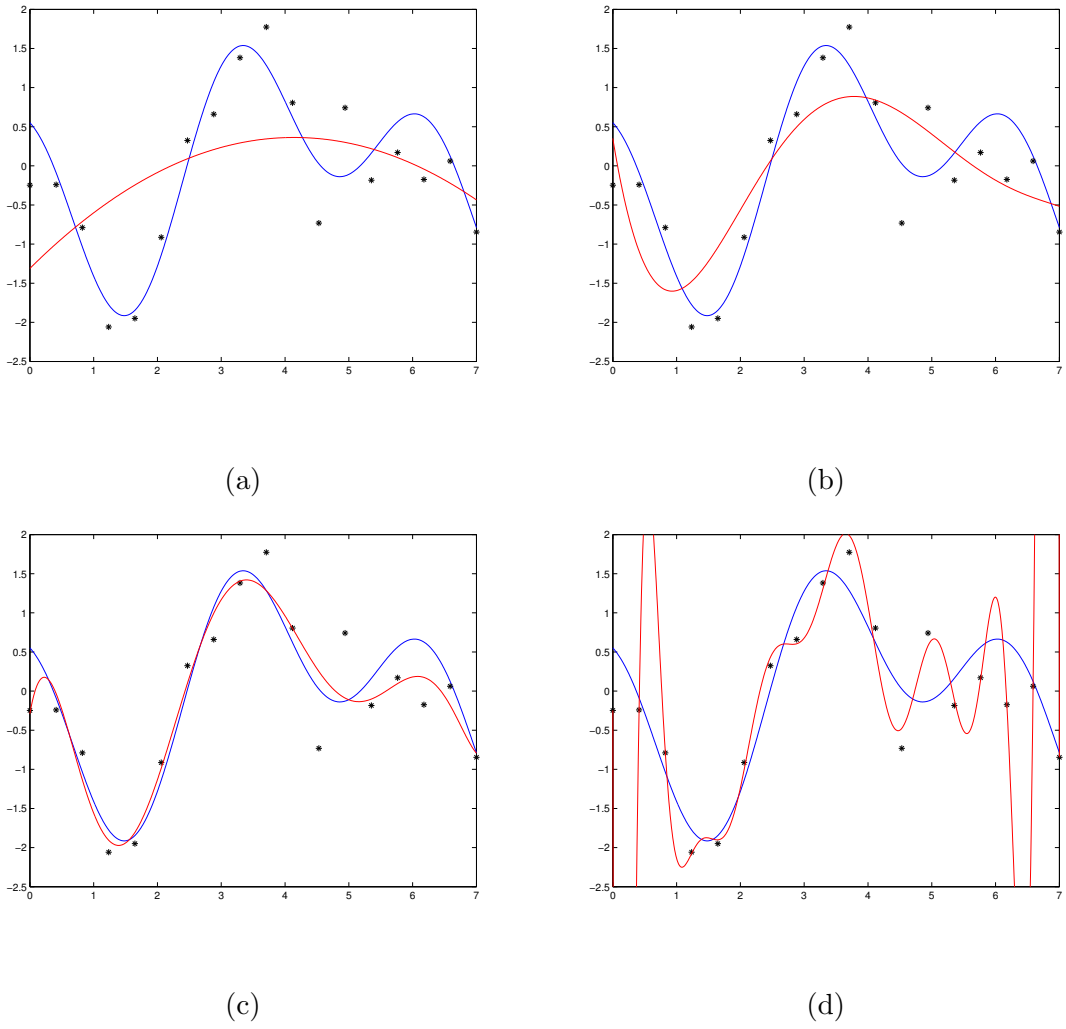
Unsupervised learning approaches are a class of algorithms aimed at analyze and classify input data without receiving explicit feedback from the environment. This type of learning is typical of living being, which have the capability to extract information from sensory inputs even if lacking the presence of a teacher. These algorithms are aimed at discovering the underlying pattern of the input data and, by that, give a characterization of the process rather than approximate a particular transfer function.

Typical examples of unsupervised learning algorithms are *factor analysis*, where the goal is to find lower-dimensional set of parameters able to characterize the process (such as *Principal Component Analysis*), or *clustering*, where similar examples of input data are grouped together, in order to differentiate different classes of patterns (such as *K-means*).

1.2 Supervised learning

Supervised learning algorithms are a class of learning approaches that try to estimate a mapping function between a series of training *patterns* and the corresponding desired outputs. The set of patterns and outputs, named *training set*, is provided by an external teacher (that is often the human designer but may also be a computer program). For example, in order to train a system to discriminate face to non-face images, a typical training set is a list of images paired with a label that marks them as "face" or "non-face".

In practical applications, however, it is more convenient to build a statistical model of the process that generates the dataset, rather than an exact approximation of the set itself. This can be easily explained if we consider the problem of approximating a function given a set of noisy samples. Using a polynomial basis we can try to minimize the mean squared error by finding a proper set of weights. A critical choice, in this example, is the degree of the polynomial. Figure 1.1 shows the different approximation we can obtain by using higher terms polynomials. Clearly, if the degree is too low



(a) (b)

(c) (d)

Figure 1.1: Examples of regression with a polynomial basis with maximum degree 2 (a), 5 (b), 8 (c) and 15(d). The blue line is the hidden process, while the black asterisks are the noisy samples and the red line is the reconstructed function

we obtain a poor representation of our process. However, if we increase too much the degree, our reconstruction of the function will be very distant from the original one: this is clear as minimizing the error with respect to the dataset our approximation will end up fitting the noise, obtaining large oscillations and poor overall reconstruction. This is a typical issue in machine learning applications, known as *overfitting*. A common strategy to minimize the problem of overfitting is to use two different data sets: after the system have been trained using the first dataset, the second, usually indicated as *test set*, is used to evaluate the overall performance. During the firsts

learning phases, the error will, in fact, decrease in both the training and the test set but, in case of overfitting, the error on the test set will increase (conversely with respect to the error on the training set, which always tend to decrease). This can be used as a stopping criteria during learning.

The design of a supervised learning procedure requires, besides, the preparation of a proper training set, which is required to be representative of the function to estimate. If the training set is not rich enough, the system will not be able to generalize well and to provide acceptable answers to unexperienced patterns. There is a trade off, however, with the computational time, as, in general, it is required more time to learn from a wide dataset.

1.3 Reinforcement learning

Reinforcement learning approaches are particular machine learning strategies used to address situations in which the system have to maximize its performances in a given environment. They are often referred to as learning by *trial and error*, since the agent tries to learn a policy for choosing action by exploiting the feedback (or *reward*) provided by the environment. Despite the presence of a feedback, which makes the scenario more similar to supervised rather than unsupervised learning (where the role of the teacher is played by the environment itself), the system usually lacks the explicit presence of a training set that points out the best response to different states (sometimes RL approaches are used in combination with a small and non comprehensive training set that is aimed only at providing an initial guess). Besides, rewards are usually not provided after each action but only after a whole attempt; the problem of "understanding" which action is the cause of the success/failure of the trial is often called *temporal credit assignment*.

A crucial aspect of RL is the necessary trade-off between *exploration* and *exploitation*: due to the lack of a training set, the agent has the role to decide whether try to *explore*, in order to experience new states and test novel solutions, or *exploit* the known situations, in order to refine the current solution and obtain a higher reward.

This underlines another key difference with respect to most of supervised learning scenarios: regression or classification problems have no "interactive" components as the correct (or preferred) output is provided with the training set, while in RL the best choice should arise from the interaction with the environment.
The choice of the reward function is, indeed, particularly important: a proper definition of it may provide a meaningful boost to the algorithm, while a bad choice may also prevent the convergency (see section 3.3).

1.4 Learning by demonstration

Learning by demonstration (*LbD*, often addressed as *imitation learning*) is a particular type of supervised learning where the training set is generated by a teacher that *demonstrate* the task to the learner. It provides a very intuitive and flexible way to train a system, both because even people with poor experience in machine learning can play the role of the teacher and because it can be adapted to many different scenarios (almost) without any modification, leading to a very natural form of human-robot interaction. Over the years, the problems of learning by demonstration have been synthesized in four key points: *how to imitate*, *what to imitate*, *who to imitate* and *when to imitate*. Up to now, almost no effort have been put in answering the lasts two questions and researchers explored almost only various approaches to address the problem of how to imitate, that concerns the different ways to encode the task or the skill showed by the demonstrator, and what to imitate, that is related to the problem of generalizing the learned skill in order to react coherently to changes in the environment. There are, however, a series of design choices that the developer should take into account:

1.4.1 Demonstration approach

There is not a single way the teacher can use to make a demonstration of the task: historically LbD was accomplished by teloperating the robot and storing some key parameters, as the position of the end effector, the forces applied on the objects and

additional constraints, like the position of the target and of the obstacles. This is not always the best choice, however, as in robots with many degrees of freedom (as a humanoid robots) complex movements are hard to achieve. More recent approaches are based on vision systems, where the motion of the teacher is monitored by (stereo) cameras (Kuniyoshi et al. (1994)), data gloves and wearable interfaces, where the execution is recorded by particular sensory suits (Tung and Kak (1995); Kim et al. (2010); Calinon and Billard (2007)) or kinesthetic teaching, where the teacher physically guide the learner through the execution of the task (Kormushev et al. (2010, 2011); Wrede et al. (2013)).

1.4.2 Problem space continuity

The way the world is represented is a key property of the LbD procedure and may significantly change the structure of the learning process. A first choice is to define whether the world is continuous or can be discretized by a set of different features that encode the current state. For example, encoding the state of the target as its 3-dimensional position vector or by subdividing the world in different main areas and storing the one in which the target is in. We can likewise distinguish between continuous or discrete actions. These choices clearly depends on the particular type of task the robot should reproduce and, consequently, affects the learning procedure.

1.4.3 Policy learning

The problem of action/state space continuity is closely related to the goal of the learning procedure; in particular, there are three main policy categories: a possible approach is to use demonstration data to learn the relation between pre/post conditions and actions (Kuniyoshi et al. (1994)). It is useful to combine this approach with the use of a planner, so to be able to better generalize and apply the learned rules even in situation pretty different from those showed by the teacher. Sometimes the demonstration comes along with an explicit labeling of user intentions (Friedrich and Dillmann (1995)).

There are examples in literature where, instead, the robot is trained to learn some of the rules of the world's dynamics and derive a policy to maximize the expected reward (most of the times manually specified by the user). In many practical applications, however, the reward tend to be sparse (i.e. it is zero for most of the attempts and non-zero for only few of them); in these cases the demonstration is aimed at boosting the learning phase, simplifying the initial exploration phase (Smart and Kaelbling (2002)). Besides, since the learning phase tend to be not just extremely long but also potentially dangerous, this is often accomplished in virtual reality or simulated environments (Aleotti et al. (2004)).

Another approach is, instead, use the demonstration to approximate a function that maps the current observation with the corresponding action. The particular mathematical model used to approximate the function highly depends on whether the action/state space is continuous and on the aim of the learning procedure. When the action space is discontinuous actions usually represents high level behaviors of the robot (sometimes indicated as *symbolic encoding*); in these situations the goal can be usually assimilated to a clustering problem, where patterns close in the input space are mapped to the same action. This may be useful since allows a sort of hierarchical learning, where the problem of choosing the correct action to do is decoupled with the problem of providing the correct low level commands. A possible approach is, for example, to use classification algorithms as *k-means* (Saunders et al. (2006)). Whenever the action space is, instead, continuous, the learning problem is more similar to a regression problem, where the learned function has the role to provide low level actions to complete the task.

This is sometimes addressed as *trajectory encoding*; it should not be confused, however, with the goal of the task: according to the particular learning scenario, the task may concern only the execution of a particular movement, like the navigation through a series of keypoints, or the reaching of a particular goal, like a manipulation task where an object should be moved from its starting position to its destination. The former scenario is often called *action-level imitation* and its typical of industrial scenarios, while the latter is usually named *functional imitation*, to underline the

goal-oriented aim of the task (Billing and Thomas (2010)).

1.4.4 Demonstration set

Like all the supervised learning approaches, the performances of the trained system are highly affected by the quality of the training set. As introduced before, a proper dataset needs to be comprehensive of a variety of different situations, so that the system may coherently learn what to do in each of them. If, instead, all the demonstrations are relative to a particular situation (for example, a particular starting configuration) the system will probably be unable to react to a relevant change in the environment. In practical applications it is not possible, however, to provide a demonstration for every possible state; in these situations the system may try to generalize the learned behavior and coherently apply it to the new scenario (there are learning approaches that are effective in this, like artificial neural networks or k-Nearest Neighbours). Alternatively, the robot may start an autonomous learning approach (like RL). This is especially useful when even the teacher was not able to provide a correct reproduction of the motion but just sub-obtimal solutions (Grollman and Billard (2011)). Another possible solution is to make the system evaluate a degree of confidence in the learned behavior and ask for a new teaching sequence in case the confidence is below a given threshold.

1.5 Evolutionary methods

Genetic algorithms are a class of approaches inspired by the natural selection of the fittest which are regarded with increasing interest from the scientific community. They are often referred to as *evolutionary* methods, due to their closeness to natural evolution. Although conceptually different, these kind of approaches are actually close to RL methods: the main difference is that, while RL operates on a single individual, GAs operate on population of individuals. Selection of individuals is achieved by means of an evaluation function, which measures the performances of the solution.

Evolutionary methods have been proved very effective when applied to robotics

(this gave birth to the name *evolutionary robotics*). Indeed, designing a control system for an autonomous robot, able to deal with a wide variety of different sensory stimuli and act coherently to all the possible situations that arise in real world environment, is an extremely complex task, especially when the robot is made of many different interdependent parts (Cliff et al. (1993)). On the other side, testing the system in a simplified world often give birth to implicit dependencies on the properties of the "virtual" environment (Brooks (1991)). A possible solution is suggested by evolutionary robotics, which aims at obtaining control systems by means of adaptation rather than analytic design. Taking inspiration from natural evolution, this kind of approach treats robots as autonomous artificial organisms which are let free to act according to their controllers' rules. Darwinian principle of selective reproduction of the fittest will then select the individuals that better solve the task, pruning the others. Genetic algorithms are described in more detail in chapter 3.

The problem to choose "what to evolve" (i.e. choose a proper representation for the control system) has not a unique solution and, over the years, different answers have been proposed: Brooks (Brooks (1992)) suggests the use of a high-level language to control behavior primitives. The use of genetic algorithms to learn a program is called *genetic programming*. Other approaches are based on fuzzy logic controllers (Hoffmann and Pfister (1996)); Dorigo and Schnepf (1993) suggests a kind of "classifier system" where the system select rules according to their usefulness. By far, the most commonly used control system is based on neural networks. On one side, it can be explained by considering the biological inspiration of ANNs, which makes them the ideal candidate for ethological/behavioral studies; from a computational perspective they appear more promising for various reasons: first of all, a wide variety of different learning algorithms have been developed (they can be used to improve the final result) and ANNs are naturally robust with respect to noise (Nolfi et al. (1995)). Besides, a desirable property of the evolutionary approach is to work on low-level components of the control system (as the connection weights) since high-level semantics tend to constrain the evolution to particular "paths" (Cliff et al. (1993)).

Another meaningful aspect of an evolutionary design is the choice of "how to

evolve". There are two main approaches: a first hypothesis is based on the idea that, not just the weights, but the whole architecture of the network should be submitted to evolution. This is supported by the idea that defining a fixed-length genotype will constrain the evolution, while using variable-length ones will not put any restriction in the search space (Cliff et al. (1993)). In most of the cases, however, a fixed-length genotype showed good performances and the problem of search space reduction seems not so severe.

It is very rare that evolutionary experiments are carried on using real robots. The main reason is clearly time: not differently from natural evolution, the drawback of genetic algorithms is that experiments are very slow (it can take from hours - for the simplest cases - to days or even weeks to converge). Furthermore, for some of the experiments, restoring the initial conditions may not be feasible without the intervention of the developer. Another important reason is security: since during the evolution not all the individuals behave as they should (most are poorly performant or even unforeseeable) there is the risk that the robot may damage itself (for example by hitting a wall). It is clearly necessary, thus, to conduct evolutionary experiments using a software simulator. However, the gap between the virtual and the real environment tend to be significant (sometimes even crucial) in experiments where the robot must rely on the environment and its response. Although modern simulators reached a very high quality (being able to simulate physical phenomena as friction or sensory noise) there are some strategies that can be used in order to minimize the difference between the behavior of a particular phenotype in both the simulated/real environment, increasing the portability of the obtained solution. Urzelai and Floreano (Urzelai and Floreano (2001)) suggest to evolve the rules for parameter self-organization rather than the parameters themselves. This way the parameters will be adapted on-line, using the evolved rules, taking into account the particular environment in which the robot is (being, thus, independent on whether it is real or simulated). Another alternative is to combine the evolution of the robot genotype and of the simulator parameters (Bongard and Lipson (2005)): the main idea is to minimize the difference between the simulated and the real environment by compar-

ing the sensory inputs of an individual (typically the best individual after a certain number of generations) tested on both the scenarios. This leads to a sort of alternate evolution where the data collected by the robot in the real world are used to improve the quality of the simulation.

1.6 Combining solutions

As explained, the huge amount of time required by evolutionary methods (as well as by RL-based methods) dramatically reduce the possibility of using them to solve very complex tasks. On the other hand, it is very hard to use LbD for problems with high intrinsic variability as well as problems that require fine interaction with the environment (due to the difficulty of generating a comprehensive training set). However, these two strategies seems not incompatible: in many natural systems (especially primates) ontogenesis includes, to some extent, an imitation learning phase, which is complementary to the phylogenetical adaptation of the species.

In literature there are different attempts to use LbD to obtain a first raw solution and then refine it through reinforcement learning: Kober and Peters (2012) successfully combined kinesthetic teaching with RL in a *Expectation-Maximization* based algorithm used for solving the *ball-in-a-cup* game; similar results are obtained by Kormushev et al. (2010) using a *Dynamic Movement Primitive* based approach. Differently from the cited works, which starts the training from a successful demonstration, Grollman and Billard (2011) proposed an effective framework for using RL even in case the teacher was not able to provide the robot with a successful demonstration (it may sounds odd, but if the task requires fine control it shall not be surprising if a human teacher is not always able to reproduce it, especially considering that he is not using his body but the robot's one). All cited works show how the sole LbD phase is not effective in many practical tasks but it becomes useful (and almost fundamental) when combined with a successive trial-and-error phase aimed at refining the solution.

In order to combine kinesthetic teaching with evolutionary robotics, a natural

choice is to use a backpropagation based approach to train the neural network with the training set generated after the teacher's demonstration and, then, let the neural controller evolve using genetic algorithms.

Chapter 2

Artificial Neural Network

If you wire it randomly, it will still have preconceptions of how to play. But you just won't know what those preconceptions are
Marvin Minsky

An artificial neural network (ANN) is a biologically inspired mathematical model for approximation of functions, made of the interconnection of independent computational units, called *neurons*. ANNs have been widely used in machine learning for their efficiency and robustness, as well as their effectiveness in dealing with a wide variety of different problems, from pattern recognition (face, handwriting, speech...) to decision making and control problems. In this chapter we will introduce the mathematical notation used as well as the formal description of the learning algorithm used during the experiments.

2.1 Artificial Neuron

Formally an artificial neuron is a computational unit that, given as input a vector $\tilde{\mathbf{x}} \in \Re^D$ return the value

$$o(\tilde{\mathbf{x}}) = \phi(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}} + w_0) \tag{2.1}$$

where $\tilde{\mathbf{w}} \in \Re^D$ is the *weight* vector and $\phi(\cdot) \in \Re \to \Re$ is called *activation function*. We shall refer to w_0 with the term *bias* (not to be confused with the concept of statistical bias). It is useful to redefine the weight vector as $\mathbf{w} = \begin{bmatrix} w_0 \\ \tilde{\mathbf{w}} \end{bmatrix} \in \Re^{D+1}$ and

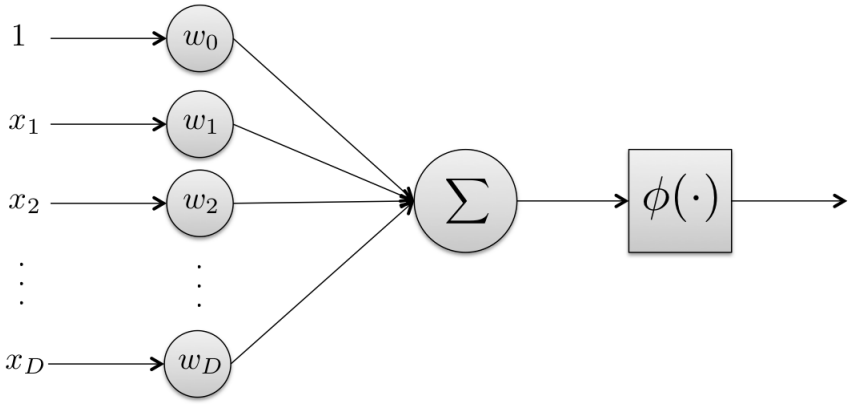


Figure 2.1: A graphical representation of an artificial neuron

the input vector as $\mathbf{x} = \begin{bmatrix} 1 \\ \tilde{\mathbf{x}} \end{bmatrix} \in \Re^{D+1}$. This allows us to rewrite equation (2.1) in the more compact form

$$o(\mathbf{x}) = \phi(\mathbf{w}^T \mathbf{x}) \tag{2.2}$$

In the following, unless explicitly said, the bias will be considered a weight as the others.

Since the inspiration of neural networks has biological roots, historically the activation function was chosen to resemble the "all-or-none" behavior of physical neurons, adopting, thus, the Heaviside step function (or sometimes the $sign(x)$ function) (McCulloch and Pitts (1943); Rosenblatt (1962); Widrow and Hoff (1960)); a discontinuous function, however, is unsuitable for gradient-based learning approaches, hence, from a machine learning perspective, a common alternative is to use *sigmoid* unit. Formally the sigmoid (sometimes called *logistic* function) is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Besides being continuous and differentiable, the sigmoid function has the useful property that its derivative can be expressed as a function of it output:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \tag{2.4}$$

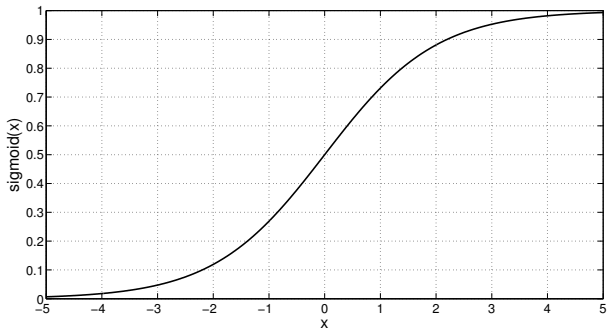


Figure 2.2: Plot of the sigmoid activation function

2.2 Multi layer perceptron

Networks made of a single neuron have clear limitations in terms of representational power. A significative improvement can be obtained by connecting different neurons (or better, different different *layers* of neurons) in a feedforward structure. They are generically called *multi-layer perceptrons*, regardless of the activation function (although this can be misleading, since *perceptrons* is the name originally given by Rosenblatt to single-layer network with thresholded units). Units whose outputs are used as input to neurons in successive layers are called *hidden*. In general, networks with arbitrary number of hidden units can be defined. However, most of practical applications do not go beyond the second level (that means, three-layers network with 2 levels of hidden units and a layer of output units). It can be shown, indeed, that every decision boundary can be approximated with an arbitrary small error by a net of three layers of sigmoidal units. It can be proved, besides, that any *continuous* function can be approximated using a layer with a single layer of hidden sigmoid units

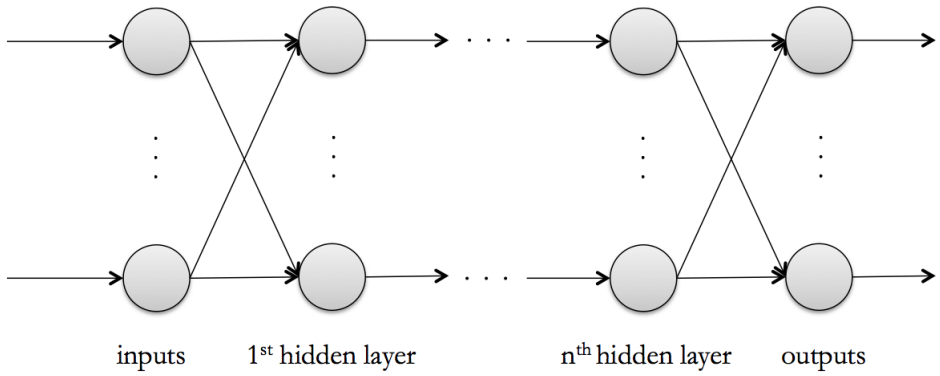


Figure 2.3: Schematic representation of a multi layer perceptron with n layers of hidden units

and a layer of output sigmoid units, provided that the number of hidden neurons is sufficiently large.

In the following we will consider, without loss of generality, feedforward networks with a single layer of hidden units. Formally, let D be the number of inputs, K the number of hidden units and M the number of output units. Besides, let w_{ji} the weight connecting the jth neuron with the ith one. For each neuron in the hidden layer it clearly holds

$$o_h = \sigma\left(\sum_{i=0}^{D} w_{hi} x_i\right) \tag{2.5}$$

Analogously, for each output unit

$$o_m = \sigma\left(\sum_{h=0}^{K} w_{mh} o_h\right) \tag{2.6}$$

2.3 Learning in ANNs

The problem of learning in an artificial neural network consists in determining a set of weights such that the system transfer function will map a set of given inputs to the corresponding desired outputs. There are different approaches to solve this problem; in this section we will consider only gradient based methods (for an evolutionary approach see section 3.5).

2.3.1 Error back-propagation

Historically the most significative approach is the ERROR BACK-PROPAGATION (usually called simply BACKPROPAGATION, sometimes indicated also as STEEPEST DESCENT, Rumelhart et al. (1986)): it is an application of the gradient descent method to minimize a sum-of-squares error function. In order to formally derive the formulation of the EBP it is useful to introduce a couple of symbols: let the training set be the set of pairs $\{(\mathbf{p}_i, \mathbf{t}_i)\}_{i=1}^{P}$ where $\mathbf{p}_i \in \Re^D$ is a known pattern (i.e. a set of inputs) and $\mathbf{t}_i \in \Re^M$ is the vector of the desired output values when we apply the ith pattern to the network. We shall use the notation t_{mp} to indicate the desired mth network output when applying pattern p (analogously, o_{mp} indicates the actual output). Besides, let

$$net_j = \sum_i w_{ji} x_i \tag{2.7}$$

be the weighted sum of inputs for neuron j (whether it is a hidden or an output unit). We can now define our error function as

$$E = \frac{1}{2} \sum_{p=1}^{P} E_p = \frac{1}{2} \sum_{p=1}^{P} \sum_{m=1}^{M} e_{mp}^2 \tag{2.8}$$

where

$$e_{mp} = (t_{mp} - o_{mp}) \tag{2.9}$$

To apply a gradient descent step we need to obtain the derivative of the error (when applying a particular pattern) with respect to every weight in the net. We can the apply the chain rule to derive a more convenient form for the gradient. In particular, noticing that w_{ji} can influence the output only through net_j and net_j can influence the output only through o_j (that, for sigmoidal units coincide with $\sigma(net_j)$), we can write

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{jp}} \frac{\partial o_{jp}}{\partial net_{jp}} \frac{\partial net_{jp}}{\partial w_{ji}} \tag{2.10}$$

Using (2.4) and (2.7) we can write

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{jp}}\ o_{jp}(1 - o_{jp})\ x_{ji} \tag{2.11}$$

The analytic expression of $\frac{\partial E_p}{\partial o_j}$ is dependent on whether j is a output or a hidden unit. We will thus consider separately the two cases.

Output

$$\frac{\partial E_p}{\partial o_{jp}} = \frac{1}{2}\frac{\partial}{\partial o_{jp}}\sum_{m=1}^{M}(t_{mp} - o_{mp})^2 = -(t_{jp} - o_{jp}) \tag{2.12}$$

It is useful now to define the quantity

$$\delta_j = \frac{\partial E_p}{\partial net_{jp}} = \frac{\partial E_p}{\partial o_{jp}}\frac{\partial o_{jp}}{\partial net_{jp}} = -(t_{jp} - o_{jp})\ o_{jp}(1 - o_{jp}) \tag{2.13}$$

Replacing this in (2.11) we obtain

$$\frac{\partial E_p}{\partial w_{ji}} = \delta_j x_{ji} = -(t_{jp} - o_{jp})\ o_{jp}(1 - o_{jp})\ x_{ji} \tag{2.14}$$

Hidden

In case j is a hidden unit the computation must take into account that the error is affected by o_j only through the output units

$$\begin{aligned}\frac{\partial E_p}{\partial o_{jp}} &= \frac{\partial}{\partial o_{jp}}\sum_{m=1}^{M} e_{mp}^2 = \sum_{m=1}^{M}\frac{\partial}{\partial o_{jp}}e_{mp}^2 = \sum_{m=1}^{M}\frac{\partial e_{mp}^2}{\partial net_{mp}}\frac{\partial net_{mp}}{\partial o_{jp}} \\ &= \sum_{m=1}^{M}\frac{\partial e_{mp}^2}{\partial net_{mp}}w_{mj} = \sum_{m=1}^{M}\frac{\partial E_p}{\partial net_{mp}}w_{mj} = \sum_{m=1}^{M}\delta_m w_{mj}\end{aligned} \tag{2.15}$$

In this case the quantity δ_j will be equal to

$$\delta_j = \frac{\partial E_p}{\partial net_{jp}} = o_{jp}(1 - o_{jp}) \sum_{m=1}^{M} \delta_m w_{mj} \tag{2.16}$$

Recalling equation (2.11) we can write

$$\frac{\partial E_p}{\partial w_{ji}} = \delta_j x_{ji} \tag{2.17}$$

Notice how equation (2.18) has the same form of equation (2.14). This is a key property of neural networks since allows to write the derivative of the error with respect to hidden neurons as a function only of the component of the error relative to the neurons in the immediately successive layer (although this advantage seems not so important in a network with a single hidden layer, this holds for networks with arbitrary number of layers). We can finally write the update rule for each weight:

$$\Delta w_{ji} = -\eta \delta_j x_{ji} \tag{2.18}$$

where η is a constant called *learning rate*. It is a critical choice when training a neural network: the learning rate scales the step size of the gradient descent method. If the learning rate is too big it will slow down convergence (in the worst case it may even prevent the algorithm to converge) as the method will not follow carefully the gradient direction. On the other side, a learning rate too small will result in a very slow convergence, especially in case the gradient is not steep.

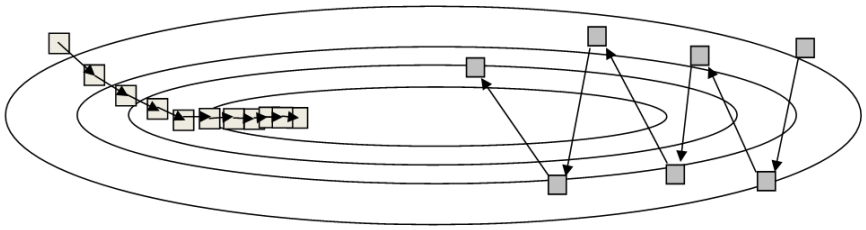


Figure 2.4: EBP with a small learning rate (on the left) and with a high one (on the right)

2.3.2 Newton's method

An improvement of EBP is provided by Newton's algorithm. The main idea is to determine the step size by evaluating the curvature of the surface (i.e. computing the second derivative of the error function). The derivation of the Newton's method starts by approximating the gradient vector with a Taylor series. Formally, being $\mathbf{g} \in \Re^N$ the gradient vector, where N is the number of weights, we can write

$$\mathbf{g} = \mathbf{\Gamma}(\mathbf{w}) \tag{2.19}$$

where $\mathbf{\Gamma} : \Re^D \rightarrow \Re^D$ represent the nonlinear relation between the gradient and the network weights. Using Taylor series we can write

$$g_i \approx g_{i,0} + \sum_{j=1}^{N} \frac{\partial g_i}{\partial w_j} \Delta w_j = g_{i,0} + \sum_{j=1}^{N} \frac{\partial^2 E}{\partial w_i \partial w_j} \Delta w_j \qquad i = 1...N \tag{2.20}$$

Since we want to minimize the error we can set g_i to 0, thus obtaining

$$-\frac{\partial E}{\partial w_i} = -g_{i,0} \approx \sum_{j=1}^{N} \frac{\partial^2 E}{\partial w_i \partial w_j} \Delta w_j \qquad i = 1...N \tag{2.21}$$

which can be written in matrix form as

$$-\mathbf{g} = \mathbf{H}\mathbf{\Delta}\mathbf{w} \tag{2.22}$$

thus obtaining the weight update rule

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{H}_k^{-1}\mathbf{g}_k \tag{2.23}$$

where k indicates the iteration.While this approach has the advantage of computing a proper step size, it has two weak points: it requires the computation of the Hessian matrix (and it could be not that easy) and it is not defined where the Hessian matrix is singular (very often in practical applications). We discuss the first problem in the next section and the second in section 2.3.4.

2.3.3 Quasi-Newton methods

Since, as anticipated, computing the Hessian matrix and invert it in real time is computationally expensive, there are alternatives approaches, knowns as QUASI-NEWTON methods, which instead build an approximation of the inverse of the Hessian matrix, in order to use equation (2.23). An example is GAUSS-NEWTON algorithm, which approximate the Hessian using the Jacobian.

Formally, being $\mathbf{e} = [e_{1,1}, ..., e_{1,M}, ..., e_{P,M}]^T \in \Re^{PM}$ the error vector, the Jacobian is defined as

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial \mathbf{w}} \\ . \\ . \\ \frac{\partial e_{1,M}}{\partial \mathbf{w}} \\ . \\ . \\ \frac{\partial e_{P,M}}{\partial \mathbf{w}} \end{bmatrix} \tag{2.24}$$

Knowing that g_i can be written as

$$g_i = \frac{\partial E}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{mp}^2 = \sum_{p=1}^{P}\sum_{m=1}^{M}\frac{\partial e_{mp}}{\partial w_i}e_{mp} \tag{2.25}$$

we can use the Jacobian to write

$$\mathbf{g} = \mathbf{J}^T\mathbf{e} \tag{2.26}$$

Besides, the generic element of the Hessian matrix can be written as

$$h_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M}\frac{\partial^2 e_{mp}^2}{\partial w_i \partial w_j} = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M}\left(\frac{\partial e_{mp}}{\partial w_i}\frac{\partial e_{mp}}{\partial w_j} + \frac{\partial^2 e_{mp}}{\partial w_i \partial w_j}e_{mp}\right) \tag{2.27}$$

Neglecting the second term inside the sum (Bishop (1995); Wilamowski and Irwin (2011)) we can approximate the Hessian matrix as

$$\mathbf{H} \approx \mathbf{J}^T\mathbf{J} \tag{2.28}$$

and, thus, the update rule becomes

$$\mathbf{w}_{k+1} = \mathbf{w}_k - (\mathbf{J}_k^T \mathbf{J}_k)^{-1} \mathbf{J}_k^T \mathbf{e}_k = \mathbf{w}_k - \mathbf{J}_k^{\#} \mathbf{e}_k \tag{2.29}$$

2.3.4 Levenberg-Marquardt algorithm

In many practical applications, the Gauss-Newton approach still suffers of instability in the regions of the weight space where the matrix $\mathbf{J}^T\mathbf{J}$ is close to singularity. To avoid this problem we consider a modified error function

$$\tilde{E} = \frac{1}{2}\|\mathbf{e}_{k+1} + \mathbf{J}(\mathbf{w}_{k+1} - \mathbf{w}_k)\|^2 + \lambda\|\mathbf{w}_{k+1} - \mathbf{w}_k\|^2 \tag{2.30}$$

that, minimized with respect of $\mathbf{w}_{k+1}$ leads to the new update rule

$$\mathbf{w}_{k+1} = \mathbf{w}_k - (\mathbf{J}_k^T \mathbf{J}_k + \lambda \mathbf{I})^{-1} \mathbf{J}_k^T \mathbf{e}_k \tag{2.31}$$

The parameter λ is inversely proportional to the step size: in particular, with small values of λ we get close to the Newton's update rule, while for large values we recover the classic gradient descent. A common heuristic to set properly the value of λ is to start with a relatively small value (typically 0.1) and change it every iteration according to the following rule: after each step, consider the error E' obtained by applying the weight update according to (2.31). If $E' < E$, apply the new weight vector, discount λ by a factor of 10 and go to the next iteration. If, otherwise, the overall error increases, the change is discarded, λ increased by a factor of 10 (which leads to a smaller step) and recompute the update.

Notice that the Levenberg-Marquardt can be seen as a "back-propagation" algorithm, since errors on internal units can be expressed as a function only of the component of the error relative to the neurons in the successive layer. In this case, however, we do not compute the derivative of the whole error E with respect to the weight but the derivative of the error on a particular output (for a particular pattern). Formally

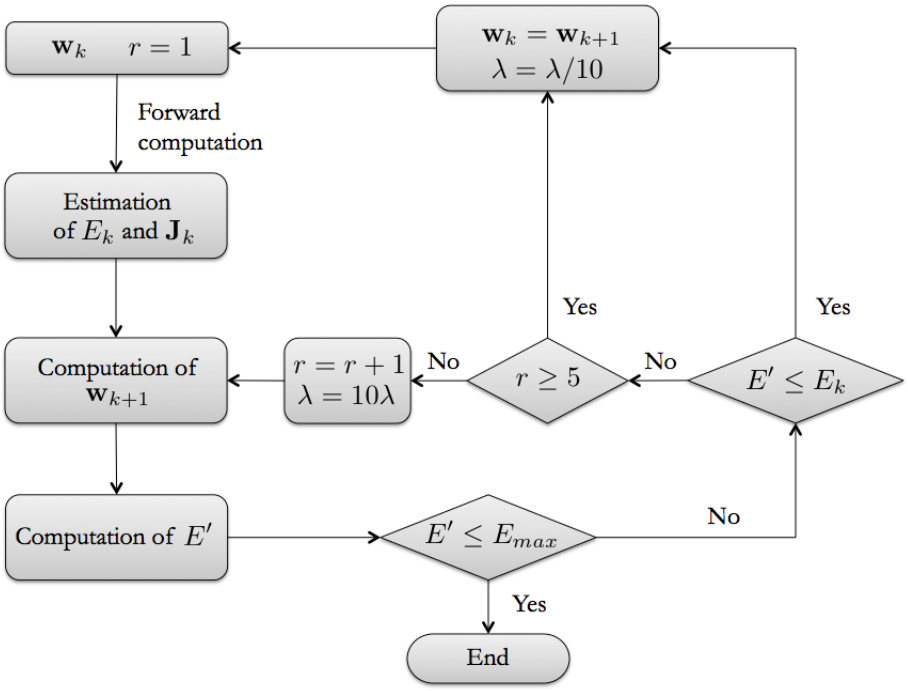


Figure 2.5: Diagram representation of the Levenberg-Marquardt algorithm

$$\frac{\partial e_{mp}}{\partial w_{ji}} = \frac{\partial e_{mp}}{\partial o_{jp}} \, o_{jp}(1 - o_{jp}) \, x_{ji} \tag{2.32}$$

Again we need to consider separately the case in which j is an output or a hidden unit:

Output

$$\frac{\partial e_{mp}}{\partial o_{jp}} = \frac{\partial (t_{mp} - o_{mp})}{\partial o_{jp}} = \begin{cases} 0 & j \neq m \\ -1 & j = m \end{cases} \tag{2.33}$$

$$\delta_{jp} = \frac{\partial e_{mp}}{\partial net_{jp}} = \begin{cases} 0 & j \neq m \\ -o_{jp}(1 - o_{jp}) & j = m \end{cases} \tag{2.34}$$

Hidden

$$\frac{\partial e_{mp}}{\partial o_{jp}} = -\frac{\partial o_{mp}}{\partial o_{jp}} = -\frac{\partial o_{mp}}{\partial net_{mp}} \frac{\partial net_{mp}}{\partial o_{jp}} = \delta_{mp} w_{mj} \tag{2.35}$$

Note that in case of a two layer network it is not necessary to explicitly compute the

δ_{jp} for a hidden unit. This reasoning can, however, be effortlessly extended to the case of a generic feedforward network.

2.3.5 Weight inizialization

It is meaningful to discuss about the problem of initialization of the weight vector. A common suggestion is to fill it with small random numbers. This is (in general) a good strategy, since weights fall in the region where the gradient of the sigmoid is maximum (see figure 2.2), allowing a faster convergence.

Nguyen and Widrow (1990) proposed an heuristic approach to speed up the algorithm convergence by properly initialize the weights of the hidden neurons: practically, they noticed that the weights tend to rearrange so that the region of interest is divided in small intervals. If the initialization phase may separate the weights, so that each of them falls in a proper "region", that would speed up the convergence since the deviation from the starting value will be smaller. In practice this is accomplished by multiplying each weight that connects the hidden layer to the input one by the constant

$$k = \frac{\beta}{\|\mathbf{w}_H\|} \tag{2.36}$$

where $\mathbf{w}_H$ is the vector of the weights from the hidden layer to the input one and β is defined as

$$\beta = K^{\frac{1}{D}} \tag{2.37}$$

Chapter 3

Genetic Algorithms

Nothing in biology makes sense except in the light of evolution

Theodosius Dobzhansky

Genetic algorithms (GAs) are optimization procedures inspired by biological evolution. They are the object of increasing attention for their effectiveness in exploring the hypotheses space and finding solution even in case the cost function is highly non-linear or discontinuous. The search starts with the initialization of a *population* (names are clearly inspired by their natural correspondences) that is let to *live* and *reproduce*; eventually the population give birth to offsprings that better solve the specified task.

3.1 Overview

Although it may looks complex, GAs all share a simple architecture: after the initialization of a population (section 3.2), each individual is let to "live" and its performance is measured by means of a *fitness function* (section 3.3). Each of the individual is then let to reproduce (secton 3.4) in order to generate new individuals that will determine the starting population for the next generation. It can be seen as a modular structure where these three components can be customized without changing the overall approach.

Before describing how these steps are meant to work, we should describe how the individuals should be represented. One of the most accredited theories on natural evolution is that the set of genetic informations inherited by the offspring is the result of integration of the genetic informations coming from the parents. The genetic material is coded in the DNA, which is a long sequence of chemical molecules (known as nucleotides). The particular chain of nucleotides is called *genotype*, whereas its manifestation in the living being is called *phenotype*. It is then necessary to define a proper representation for the genotype and a function that maps the genotype in the phenotype (sometimes the genotype does not encode directly the properties of the individual but the rules to generate them). In GAs, genotypes have often the structure of arrays, since this allows an easy manipulation and modification of the genes, neglecting most of biological complexities.

3.2 Initial population

There are a couple of critical aspect one should take care of when initializing the population: first of all the initial population should (somehow) permit the evolution to eventually give birth to a "winning" individual. In other words, the initial hypotheses should be chose so that the search space can be efficiently explored (this, of course, depend also on the reproduction and mutation phase) in order to reduce the risk of falling into local minima. If the initial population is initialized randomly, a large initial population is required if the search space is very wide. On the other side, a wider population requires a higher computational cost, which may be unfeasible, especially if the evaluation of a single individual is already time-consuming. If, instead, some initial informations are available (such as a sub-optimal solution) it is possible to reduce the size of the population, choosing initial individuals on the basis of the prior knowledge. These choices are, however, domain dependent and there is not a general rule to address this problem.

3.3 Fitness

After the initialization phase, the evolution proceeds iteratively from generation to generation, until a stopping criteria is met. For each generation, each individual is let to "live" and its performance is evaluated by means of a fitness function. In practice the fitness function is an application from the space of hypotheses to $\Re$ that, as a generalized distance, provides an criterion to selectively distinguish a good hypothesis from a bad one.

Choosing a proper fitness function is the core of the design of an evolutionary learning procedure and may allow a fast convergence as long as preventing the algorithm to reach a solution at all. Obviously it should somehow encode the efficiency and effectiveness of the particular individual in solving the task (for example, in a classification problem a typical function may be the accuracy rate). Unfortunately, this is not always easy, as in many situations the evolved individual should be able to accomplish different objectives.

Again, no standard rule allows a easy choice for the fitness; a common heuristic is to choose the fitness so as to be as simple and general as possible. A strength point of GAs is, in fact, its effectiveness in exploring the search space, often reaching solution than, even though they are pretty far from a "human" perspective, they result very effective in practical applications as they can overcome system intrinsic limitations (Nolfi and Marocco (2002); Nolfi (2002)). Choosing a complex fitness function with different parameters, may constrain the evolution on a particular "path", preventing the algorithm to reach a more robust solution. On the other side, however, if the fitness function is "too general", the algorithm may not be able to converge at all. A simple example of this scenario is the situation in which a robot need to accomplish a sequence of tasks (like reaching different points in the space). It is very unlikely that, sampling random individuals all over the search space, one will eventually find the one who accomplish the task. Thus, if the fitness function gives a bonus only to the individual who accomplish the task and no reward at all to the others, then there will be no criteria to distinguish an individual that is moving in the right direction

(even though in an inaccurate way) with respect to another that is moving in the opposite direction. A better approach is, instead, giving additive rewards, so that "better" individuals can be selected and the others discarded. Weighting properly the components of the fitness is, again, not trivial, and often accomplished through trial and error procedure.

3.4 Reproduction

The evaluation of the fitness is the most time-consuming phase in a evolutionary approach (in fact, the "living" step is only functional to fitness evaluation). It is then necessary to select the best individuals of the current generation and discard the worst. There are a lot of different ways to apply this step and it is not clear whether there is a significative advantage in using an approach instead of another. In general, however, GAs tend to identify the individuals with higher performance and combine or "mutate" them in order to obtain slightly different individuals that (hopefully) will receive an higher fitness (performing, in others words, a local random sampling in the hypotheses space). This is clearly a critical step: if only a small percentage of current population is selected for the next generation (high selection pressure) the fitness tend to rapidly increase but easily converge to a local minimum (as the diversity and heterogeneity of individuals is decreased, resulting in a variant of hill-climbing algorithm). The opposite situation tend, instead, to starvation, as the population will almost remain unchanged over generation without, thus, any increase in the overall performance. A common approach, sometimes called *proportionate selection*, is selecting individuals with a probability proportional to their fitness. In symbols:

$$p(i) = \frac{f(i)}{\sum_j f(j)} \tag{3.1}$$

where $f(i)$ is the fitness of the i-th individual. Other possible selection criteria are *tournament selection*, where pairs of hypotheses are randomly selected and the fittest

of the two is selected with probability p, or *rank selection*, where hypotheses are sorted by their fitness and selected with a probability proportional to their rank. Many other variations are possible.

After some of the individuals are selected, they are often used to generate offsprings. The way this happen is, again, design-dependent and may significantly vary among GAs. A common strategy is to use *crossover*: named after the biological crossover, this technique is based on the combination of the genotype of two individuals. According to the way the parents' genotypes are combined we talk of *one point*, *multipoint*, *uniform* or *arithmetic* crossover. This approach tend to be effective when the resulting offsprings may combine subsolutions obtained by their parents, i.e. when the genotype tend to encode almost independent properties of the phenotype, as the crossover may "take the best" from the parents. Another possibility (often used in combination with crossover) is mutation: it consists in randomly change one of the values encoded in the genotype. It is clear that the magnitude of mutation can be tuned according to the needs (see section 4.6.4).

In general, generated offspring are used to replace the entire population (this is called *generational replacement*). Despite its similarity with natural selection, this kind of strategy may replace performant parents with worse individuals (especially if the search space is very complex). *Elitism* is an alternative approach that consists in select for the next generation only the best individuals, whether they are parents or offsprings. This solution allows to replace individuals only when better ones are found.

3.5 Genetic Algorithms and Neural Networks

As anticipated in section 2.3, GAs can be used to learn the weights of an ANN. Being this approach independent from the activation function and/or from the particular structure of the net, the very same procedure can be applied to train both thresholded (in general discontinuous) units as well as recursive networks. It is meaningful to underline that, however, an evolutionary approach is not always trivial, since each

ANN has an high intrinsic symmetry. This is clearly noticeable if we consider two neural networks where the weights leading into and out of the first hidden unit are changed with the corresponding weights of the last hidden unit (the same reasoning can be applied to every pairs of hidden unit): the transfer function of the net will remain unchanged but the weight vector will be very different. In general, a network with a single layer of hidden units will have $K!2^K$ number of equivalent weight vectors (Bishop (1995)), each of them generating a local peak in the fitness surface. Careless reproduction phase may produce "half-way" offsprings with low fitness. Although this is not a serious risk in practical applications (as individuals with low fitness are soon discarded on behalf of better ones), it may still worth the effort to take this effect into account.

In order to design an evolutionary procedure first we need to define the genotype and how this is mapped into the phenotype, then we must formally establish the three phases the GA is made of.

3.5.1 Genotype

Our goal is to find a set of weights in order to minimize the quantity (2.8). For simplicity the same quantity is reported here:

$$E = \frac{1}{2}\sum_{p=1}^{P} E_p = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M}(t_{mp} - o_{mp})^2 \tag{3.2}$$

The simplest solution is to define the genotype as a N-dimensional vector $\hat{\mathbf{w}}$ where $\hat{w}_i$ encodes the value of the ith weight. Notice that more complex alternatives are possible, like generate each of the weights of the network as a non-linear function of the values encoded in the genotype (this may have the advantage of reducing the search space, but the nonlinear function should be properly defined so that it does not prevent the exploration of regions of the weights space).
There is no real difference, in this case, between the genotype and the phenotype, as the latter is obtained by applying $\hat{\mathbf{w}}$ to the ANN.

3.5.2 Initial population

The dimension of the initial population is clearly dependent on the dimension of the ANN. As said before, no general rule is available to address this problem. Since each individual performs a local search in the weight space, having a wider population increases the possibility that one of the individuals falls close enough to the global minimum. On the other side, estimating the fitness for a large population requires a higher computational time. In practice, this problem is solved by trial and error, finding the size that allows a wide-enough search without being too computationally expensive.

3.5.3 Fitness

The choice of the fitness function is, in this case, obvious, as we want to minimize the error with respect to the training set. Notice that, in this case, the fittest individual will be the one with the lowest fitness.

3.5.4 Reproduction

There is not a single possible definition for the reproduction phase. Every alternative described in section 3.4 is valid and should allow the algorithm to converge. As a simple example, a possible strategy is to generate an offspring by computing a (weighted) average of the weights of its parents (parents may be probabilistically selected on the basis of their fitness) and then applying a mutation phase that alters the value of some of the weights.

Chapter 4

Experiments

As already introduced, the aim of the work is to develop effective forms of learning by exploiting feedbacks coming from different sources. To achieve this goal we tried to combine a teaching informations that provides hints on the way the task should be solved with a goal-oriented feedback, which provides informations on whether the goal is actually reached by the system. The first one is obtained by means of a learning by demonstration phase: the robot is kinesthetically guided through the resolution of the task, in order to obtain an example of successful trial. This information is used to provide the robot with a raw initial movement; this will, in turn, affect the efficiency of the evolutionary training, as the exploration phase will be highly reduced.

The chapter is divided in different sections: initially we will describe the software simulator used for the experiments, as well as the typical scenario (section 4.1). The chapter continues with a general description of the control system (section 4.2) and of the training algorithm (section 4.3). Then we will analyze in detail the various aspects of the conducted experiments: a preliminary experiment for *reaching/touching* is described in section 4.4, while sections 4.6 and 4.7 will describe the problem of *grasping* and *moving* respectively.

4.1 Simulation and scenario

Experiments have been carried out using the simulator developed at the Laboratory of Autonomous Robotics and Artificial Life (LARAL) at the Institute of Cognitive Science and Technology of the National Research Council (ISTC-CNR) by Gianluca Massera, Tomassino Ferrauto et al (see laral.istc.cnr.it/farsa).

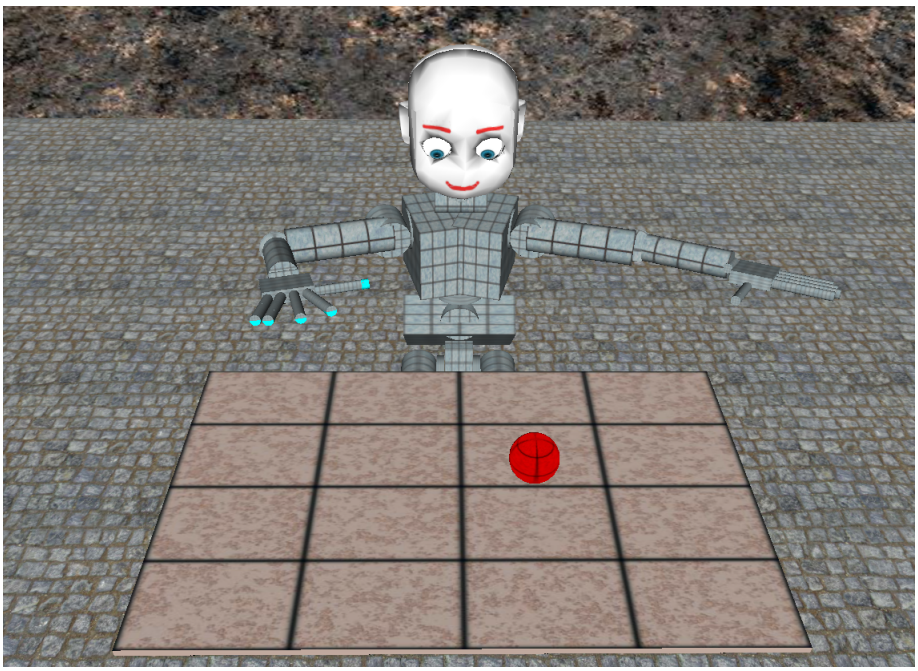


Figure 4.1: The scenario of a typical simulation. The left arm is not used in the experiments, so it is kept fixed. The robot should acquire an ability to reach an manipulate an object located over the table in a varying position.

The simulator allows the reproduction of an iCub robot, designed so to be as similar as possible to the real one (except for the esthetic). The virtual robot (as the real one) is equipped with two cameras, gyroscopes and accelerometers and microphones. It is moreover provided with force-torque sensors, as well as tactile sensors on the palm and on the finger tips. Proprioception is possible thanks to encoders (one per each joint). It was not necessary, however, to use all the available sensors: in particular, the experiments involve only the encoders and the motors of the neck, the torso and the right arm, plus tactile informations of the right hand and a camera. Legs and left arm are kept static.

The typical scenario sees the robot standing before a table. The target object is a red ball with a diameter of 7 cm and a mass of 250 grams. In order to obtain a more robust solution, the position of the ball is not the same in every experiments: the table is divided in 4 different "areas" and, per each trial, the ball is randomly placed in one of those areas (i.e. per each trial not only the area will not always be the same, but also the position inside that area will be different, see figure 4.2). All the

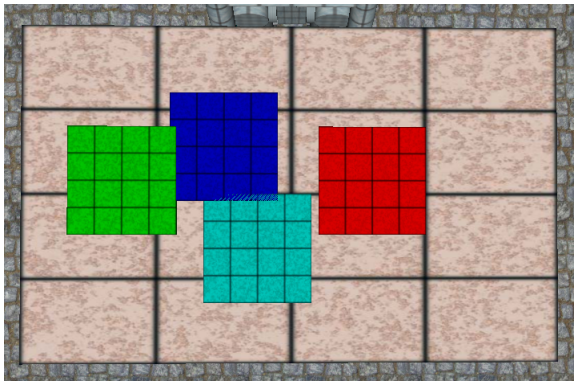


Figure 4.2: The four possible areas in which the ball can be placed. The areas are concentrated on the left part of the image (corresponding to the right of the robot): since the task should be accomplished only with the right arm, reaching a ball placed on the extreme left will be pretty hard and, also, not meaningful for the experiments.

simulations are "dynamic", i.e. they take into account dynamical effects as collisions, accelerations, inertia. The physical simulation is based on Newton Game Dynamics, an open-source physics engine.

4.2 Control system

As anticipated, the control system is based on a feedforward ANN. In particular, the adopted model is a modular neural network where inputs are defined on the basis of the sensors (proprioceptive and exteroceptive) used and output units are defined on the basis of motors to control. There is not a 1:1 mapping from sensors to input units and from motors to outputs: data coming from sensor are, in fact, preprocessed before being sent to the network as inputs. This means that a sensor can be associated to more than one input unit (this is what actually happens for

camera sensors). Analogously, output units are not used directly to control motors but a higher level interface is used: each joint is controlled by a low-level controller that is responsible for applying the required torque (last three fingers are indeed actuated together). In our experiments, the controller will work in position mode. The reference values used by the controller are the outputs of the network, i.e. the network has the role to provide the controller the reference positions for each joint, while the controller has the role to actually move the joint by applying the required torque.

Hidden units are divided in two groups: in order to combine kinesthetic teaching with genetic algorithms, we chose to train half of the network by demonstration (keeping fixed the other half). After the kinesthetic phase, the pre-trained weights will be kept fixed while the other will undergo the evolutionary process. The ANNs used will be described in detail for each experiment.

4.3 Training

Each experiment is divided in two phases: a kinesthetic and an evolutionary one.

4.3.1 Kinesthetic teaching

In the first phase the robot is moved by the teacher in a successful demonstration of how the task should be accomplished. The robot should be able to record, step by step, the sensorial state and the position of the joints that should be controlled.

Although moving a real robot is a relatively intuitive task for a human, obtaining the same results in a software simulator is not completely trivial. The proposed solution is to disable the controller of the arm/torso of the robot and approximate the guidance of a teacher by applying at each instant a force on the wrist directed towards the goal. The resulting trajectory was pretty natural as the robot is automatically reconfigured, as if an invisible teacher was pushing it in the right direction. This solution, however, is much harder to apply in situations where it is necessary to move different objects at the same time, especially if precision is needed. The kinesthetic

phase of grasping, namely, is achieved by temporarily re-enabling the controller of the wrist and setting a desired position for the joints of the fingers (i.e. close the fingers). However, the fact that the robot is moved by its own controller does not affect the recording of the training set: indeed, as the robot is only storing the *position* of the joints, it is not relevant the cause that moves the joints (whether it is an external teacher or the controller itself) but only the values returned by the encoders.

Once the dataset was generated, it can be used to train the neural network. It is important to underline that a naive sensory input - joint position association cannot be successful: according to the way the training set is generated, it contains a list of sensory inputs paired with the position of the joints *at a particular time step*. Training the ANN in order to learn this kind of mapping will result in a sensorial loop where the current inputs will push the robot to take the current posture, while the current posture will produce the same sensory inputs. It is necessary, thus, to associate the sensorial stimulus at time t with the posture experienced at time $t + \tau$, so that, moving towards the goal, the robot will perceive different sensory inputs and will avoid a stalemate. This problem will be resumed in the following sections.

In order to train the network while keeping distinct the two modules (the "genetic" from the "kinesthetic" one) it was necessary to implement a modified version of the Levenberg-Marquardt algorithm: in particular, the "classic" version computes the error of *all* the outputs with respect to *all* the weights. In our scenario, however, this computation was simplified by reducing the derivation only with respect to the "kinesthetic" weights.

4.3.2 Evolutionary phase

The second training phase is based on an evolutionary approach. The initial population consists in 20 individuals initialized on the base of the pre-trained individual. The genotype consists in an array of integers of size equal to the number of connections. The weights of the phenotype are generate by mapping the corresponding genotype element from $[0, 255]$ to $[-5, 5]$. All the individuals share the same "kinesthetic" weights, while the "genetic" ones are initialized at the beginning of the evolution

according to the mutation phase (details will be provided with the description of the experiments). The "lifetime" is divided in *trials*, whereas each trial is divided in *steps*. A trial is the single attempt of complete the task in a given number of steps (i.e. time units). The performances of each individuals are evaluated as the average fitness over the number of initial trials: in order to speed up the computation, in the initial phase of the evolution (where, likely, most of the individuals have a low performance) the fitness is computed on a small number of trials (typically 4). As the performances of the individuals increase, the number of trials is increased, so to make a "deeper test" on performing individuals and reduce the influence of luck: in the initial phase of the evolution, in fact, most performing phenotypes are the luckiest ones, i.e. those that manage to come closer to the goal only because the ball is in a particularly convenient position. While, at the beginning, those individuals must be encouraged, as time goes by it is necessary to refine the solution, making it robust with respect to environment changes.

The reproduction phase takes place by allowing each of the 20 individuals to reproduce, creating a mutated offspring (mutations are explained in deep in section 4.6) which will then be evaluated. The most performing 20 individuals (among 20 parents and 20 offsprings) are retained for the next generation. Due to computational cost of evolutions (simulations that take into account dynamical effects between objects are very expensive), all the experiments have been executed on the cluster of the ISTC.

4.4 Reaching/Touching

In a first series of experiment we trained the robot for the ability to reach the spherical object and to touch it with the palm of the hand. We selected this task since it is one of the easiest manipulation behavior we could study within this scenario that include a physical interaction between the robot and the environment. As we will see this ability can be acquired through the use of kinesthetic teaching only, i.e. the combination of kinesthetic and trial and error learning is not necessary in this case. The neural network used for this task is in figure 4.3.

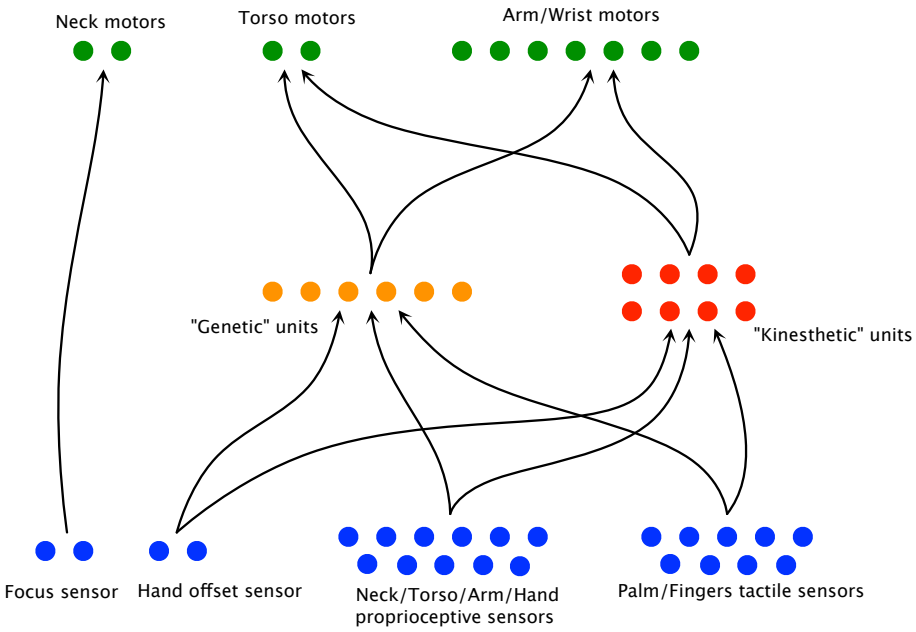


Figure 4.3: The architecture of the net for the reaching problem. Arrows between two groups of neurons indicates full connectivity (i.e. each neuron in the first group is connected to each neuron of the second group).

The leftmost part of the net is the one that controls the neck motors. The focus sensor is a high-level sensor that encodes the position of the ball in the image plane. These 4 connections are hand-tuned so that the robot moves its head in order to keep the ball at the center of the image plane. This also affects the values received as inputs from the "hand offset" sensor: the third and the fourth input units encode the distance of the hand with respect to the ball in the image plane. Keeping the focus on the ball, thus, affects also the perception of how far is the hand from the goal, while these inputs would have been misleading if the robot was looking somewhere else. Proprioceptive sensors (2 for the neck, 2 for the torso, 3 for the shoulder, 1 for the elbow and 3 for the wrist) provide the values of the joint encoders (properly scaled in the range $[0, 1]$). Neck and torso are constrained to avoid roll movement (only pan and tilt are allowed); in general, all DOFs are free to move in the range of possible movements of the real iCub. Tactile sensors (4 on the palm and 1 on each finger tip) activates if they are in contact with any object (except the robot itself). Output units provide the desired joint position for the neck, torso and arm motors

respectively.

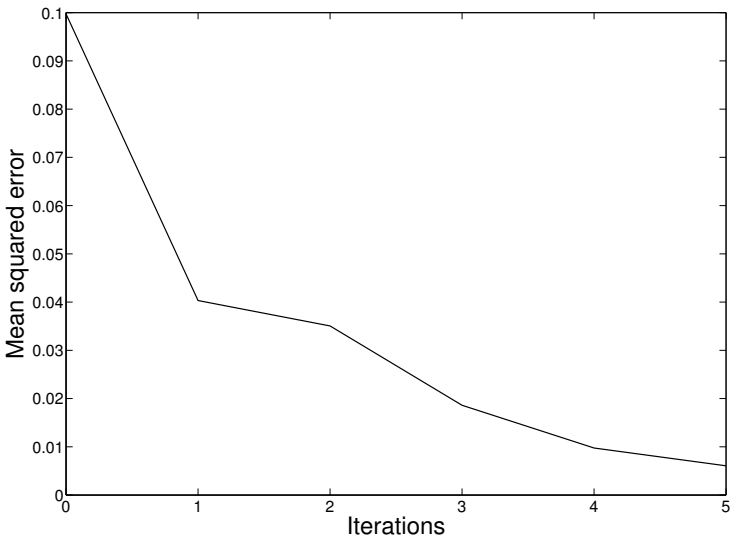
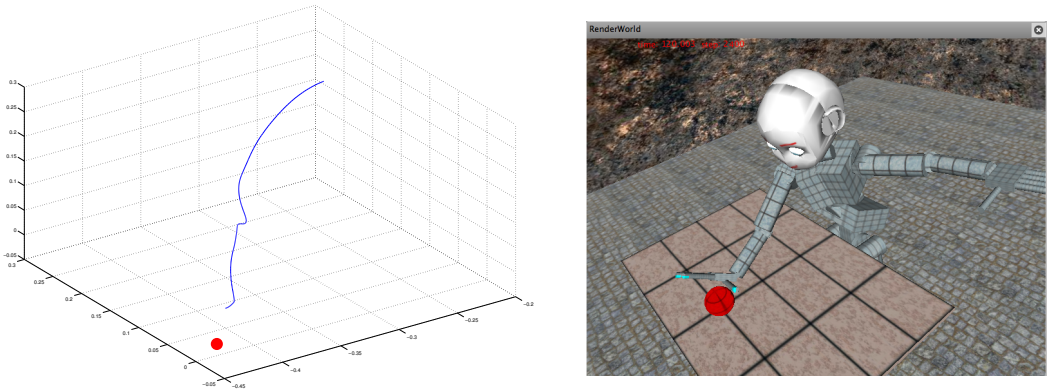


Figure 4.4: The plot of the mean squared error over the number of iterations of the Levenberg-Marquardt algorithm.

The training set for the kinestetic teaching has beeen generated by applying a force on the palm proportional to the distance of the palm with respect to the ball. This allows to go faster when the hand is far away, and gradually slow down the movement as the palm is approaching the ball. The training set is generated over 4 trials of 300 steps each. During the demonstrations the robot stores the values of all input units and the positions of the torso/arm joints. Notice that no desired output is stored for the neck motors: the neck is indeed guided by the hand-coded reflex and there is no need to store its reference position. For each trial, the sensory input at time t is associated with the position at time $t + 50$; for a deeper explanation of the anticipation term see section 4.5.

The kinestetic part of the neural network has been trained using the Levenberg-Marquardt algorithm; this approach showed extremely good performances as it was able to converge in only 5 iterations (see figure 4.4). It is interesting to notice that the weights of the connections from the tactile sensors are (almost) zeroed: this means that the robot does not rely on the tactile sensors in order to complete the task. This is, in fact, coherent, as the tactile informations do not play any role in the "reaching"



(a) (b)

Figure 4.5: The behavior of the robot immediately after the kinesthetic phase. On the left, the 3D plot shows the trajectory of the palm during the trial, while the red dot marks the position of the centroid of the ball. On the right, the screenshot of the simulator window shows the final posture of the robot.

phase, while fire only when the task is, in fact, completed.

Figure 4.5 shows the behavior of the robot immediately after the kinesthetic teaching phase. The plot shows that the robot is able to reproduce very well the demonstrated trajectory, reaching the ball with a very natural movement. It actually does not touch the ball, but this may be due to the low number of demonstrations (only 4 trials) as well as a too slow movement (since each trial is stopped after 300 steps, it is easy that a too slow movement prevent the robot to fully complete the task).

4.5 Why anticipation?

As pointed out before, an anticipation term τ is necessary in order to avoid deadlock situations (associating the sensory input at time t with the posture at the same time will produce a sensory loop from which it is hard to get out). However, with a simple mapping there would not be the desired outputs for the last τ stored inputs (see figure 4.6). In case of the reaching problem, this may cause the robot to come close to the ball but eventually drift and move away from the goal instead of reaching it.

To avoid this problem, lasts patterns are mapped to the last position: this is also coherent with the aim of the training, as when it is "close enough" to the goal it is driven to head for it .

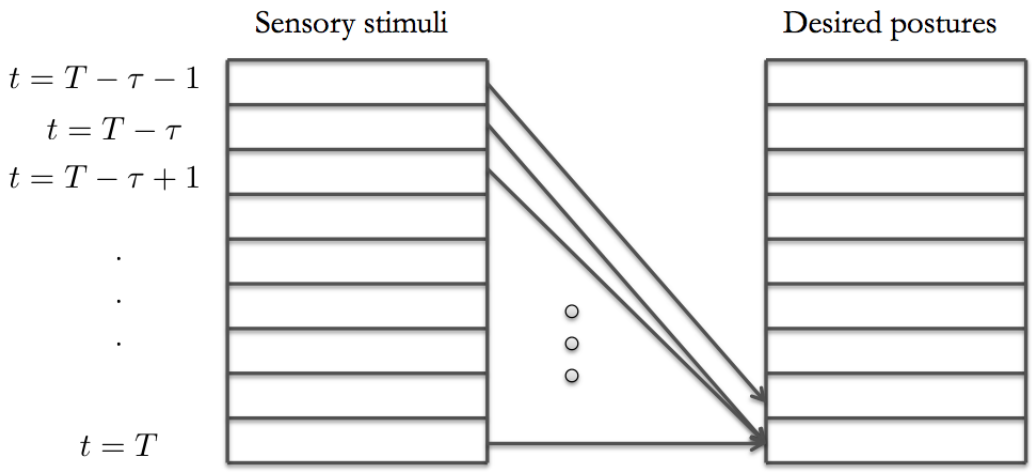


Figure 4.6: The mapping between sensory stimuli and desired postures. In the diagram, T is the total number of steps in the current trial and τ is the anticipation.

Typically the value of τ has been set equal to 50. This was chosen after different attempts: too low values will prevent the robot from moving (or, better, will make it move so slowly that the trial will be over before it is able to make any significative movement). On the other side, a too high anticipation will lead to undesirable results. In case of the reaching problem, indeed, a very high anticipation will produce equally good results but this, of course, does not hold in general. A very high anticipation will in fact push the robot directly to the final state; in case of the reaching problem this behavior may be considered acceptable, as the final state is actually the only goal of the task. In general, however, the task may have multiple goals (as in the grasping or moving experiments) and, thus, heading directly to the final position will make the robot skip meaningful parts of the task.

Figure 4.7 shows a plot of different trajectories obtained varying τ. The force applied on the palm during this example is computed as

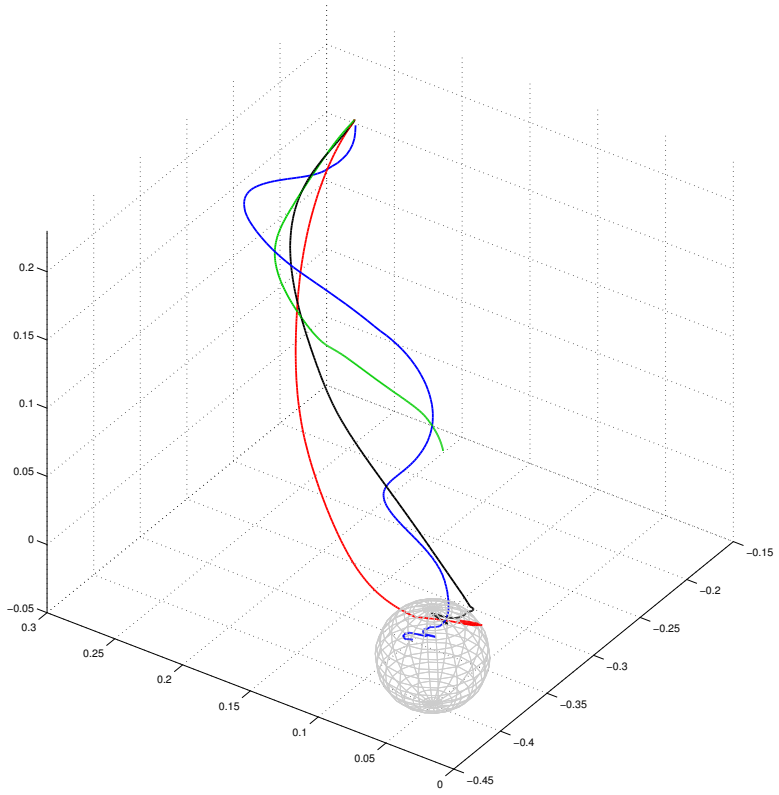


Figure 4.7: The trajectories obtained by varying the anticipation term. The blue line is the one showed by the demonstrator, while the green, black and red lines plot the behavior of the robot after training the network using an anticipation term of 15, 50 and 140 respectively. Differences of the trajectories in the neighborhood of the ball are due to different interactions with the ball itself: when the robot touches the ball it tends to roll away. The particular way the palm and the ball collide results in a different rolling and, thus, in a different behavior for the robot.

$$f = k\frac{b-p}{\|b-p\|} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} sin\left(\frac{\pi t}{40}\right) \tag{4.1}$$

where b is the position of the ball, p is the position of the palm (both expressed in the world frame), k is a gain factor and t is the current time step. The sinusoidal term is added only to simulate a trajectory with different changes of directions.

Notice how the green line, corresponding to the behavior obtained after the training with $\tau = 15$, is close to the teacher's demonstration (blue line). However, the movement is too slow and the trial ends before the palm is able to reach the ball. On the other hand, using an anticipation of 140 steps (red line) the robot tend to ignore the particular trajectory showed by the instructor, relying only on very distant reference positions. Again, this may sound reasonable in case of the reaching problem, but in case of task with multiple via-points the training may results in a behavior distant from what expected.

The anticipation step affects also the combination of tasks. This is not clear when dealing with the reaching problem but becomes meaningful in more complex examples, like the grasping one. During the demonstration, the teacher initially moves the palm of the robot towards the goal and, only when the palm is close enough to the ball, he starts to close the fingers. The demonstration is, in fact, segmented: reach $\rightarrow$ grasp. This is clearly not the way humans commonly perform grasping. The anticipation term suggests a natural solution for this issue: using a reference position that is τ steps forward, the robot will start closing the fingers before coming in the very neighborhood of the ball. This combination of the two tasks (reach and grasp) arise spontaneously when using a (proper) anticipation step: again, if τ is too small the robot may not close the fingers even if the palm is in contact with the ball, if it is too high it may reach the ball with the fingers already closed.

4.6 Grasping

The problem of grasping is far more complex than the reaching (and touching) one. As one may guess, the task involves a "reaching" phase, where the robot approaches the palm to the ball, and a "grasping" phase, where the robot has to close its fingers around the ball so to hold it and prevent the ball from falling.

The neural network used has the very same structure of the one used for the reaching problem but, in this case, it was necessary to add an additional motor (that receives inputs from both the genetic and the kinesthetic units) to control the opening

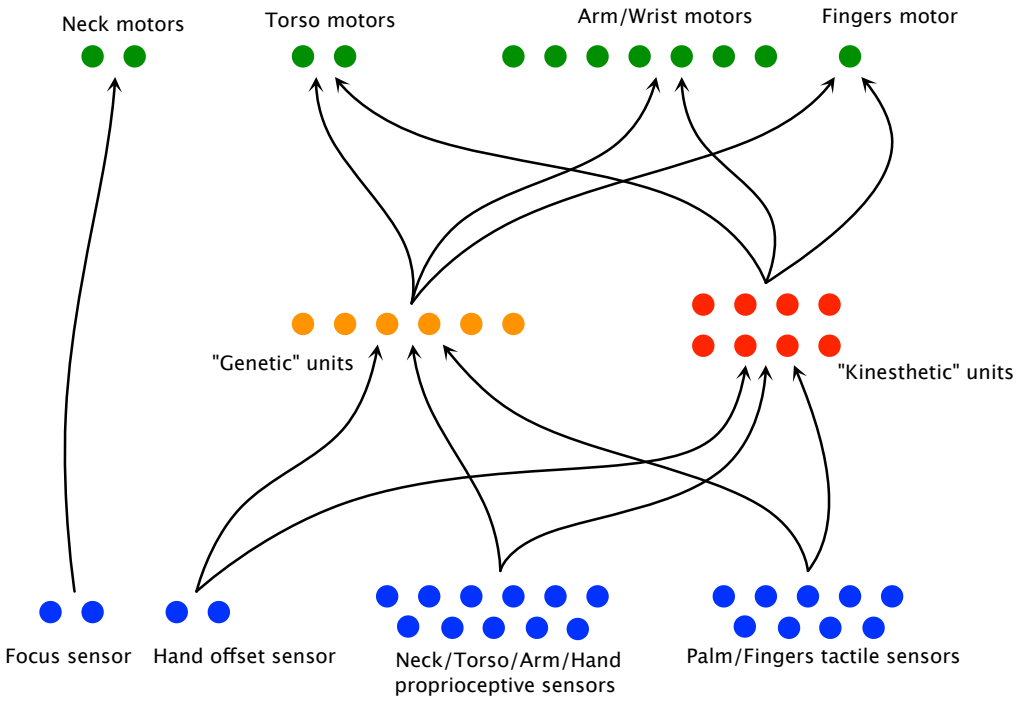


Figure 4.8: The architecture of the network for the grasping problem.

and closing of the fingers.

4.6.1 Kinesthetic teaching

The kinesthetic demonstration consists of an initial phase where the robot is pushed towards the ball (equivalently to the case of reaching); when the robot is close enough to the ball a second kinesthetic phase guides it in closing the fingers around the ball (while moving towards it, in order to remove the small gap that may be present). As introduced before, closing the fingers of the robot by applying forces on them is not easy in the simulator (with the real robot, the teacher may close his hands above those of the robot in a pretty natural way). The adopted workaround consists in temporarily re-enabling the controller of the fingers and manually setting a desired position for all of them (so that the low-level controller will gradually close them). The desired position is not set to the fingers' joint limits: proper grasping is a very dynamic task where the necessary forces to apply should be modulated according to the "response" of the object (is it heavy? is it smooth? is it rigid?). Setting the

reference values for the fingers at the respective joint limit would result in a over-tight closure, making the ball slipping away, with a resulting failure of the overall task. In order to avoid this problem, desired positions have been set to $\frac{2}{3}$ of the joint range; a more general solution would surely be creating a compliant controller that modulates the forces according to the response of the environment. However, this is not necessary since, as explained before, the kinesthetic phase is much easier with a human teacher guiding a real robot. The demonstration includes a final phase where the robot, after the completion of the grasping, is taught to lift the palm holding the ball. Although this is not strictly needed, it is aimed at creating a sort of "grasping confirmation" phase, so that the robot can show a different behavior when it recognize that the grasping have been successful.

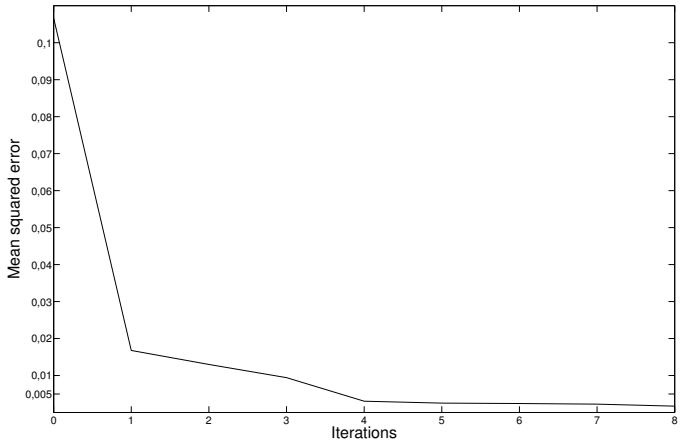


Figure 4.9: The plot of the mean squared error over the number of iterations for the grasping problem.

The dataset was generated over 10 trials of 320 step each. During the training phase the anticipation τ has been set equal to 50 step. Figure 4.9 shows how the error decreases with the iterations. Notice that, even though the number of iterations required to converge is not meaningfully higher than the one required for the reaching problem, in this case each iteration is much more computationally expensive: the jacobian is, in fact, a 32000×274 matrix (while for the reaching problem its dimensions were 10800×265). Each step requires first the computation of the jacobian and then the inversion of the matrix $J^T J + \lambda I$, which is a 274×274 square matrix.

Figure 4.10 shows a test performed immediately after the kinesthetic teaching phase. The overall results (considering also other trials) are very good: the reaching phase is (almost) every time correct and sometimes the robot is able to correctly close the fingers on the ball. The grasping phase, however, is prone to errors: sometimes the robot reaches the ball with the fingers already closed or, conversely, does not close the fingers at all. Although, at this stage, the problem cannot be considered solved, it is interesting to notice that, since the kinesthetic phase, the use of neural networks allows to remove the segmentation of the task: the initial position of the hand sees the fingers completely extended. However, since the firsts time steps, the robot slightly flexes them. This is a very natural behavior, as when humans perform grasping the bending of the fingers is concurrent to the reaching phase.

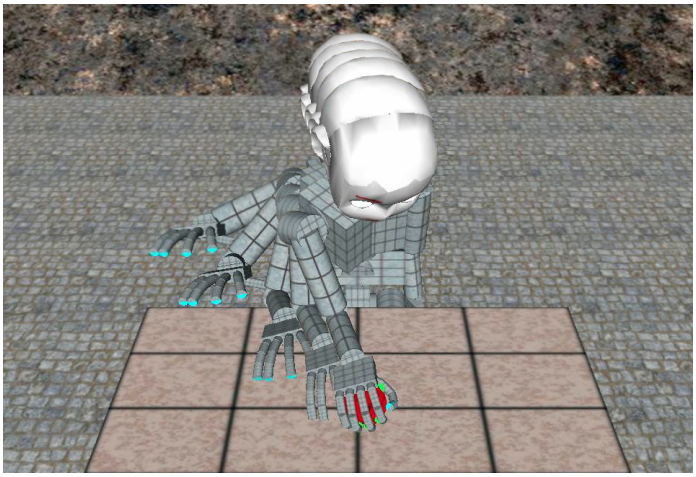


Figure 4.10: The image shows the behavior learned from just the kinesthetic phase.

The result of the kinesthetic phase is a neural network where all the connections relative to the genetic weights are set to 0, while the kinesthetic ones have just been initialized by the Levenberg-Marquardt algorithm. We would like now to use genetic algorithms to improve the robustness and the performances of the system. Before analyzing the adopted solution, it is important to underline the role that this evolutionary phase plays in the training of the robot: a pure evolutionary approach is aimed at finding a solution in the genotype space that maximize the fitness of the phenotype. In other words, the goal is to find a set of weights for our ANN so that the robot is able to grasp a ball in most of the configurations. In this case, however,

the scenario is slightly different: the kinesthetic phase provides an initial sub-optimal solution of the problem so, rather than start blindly with an *exploration* phase, it is more convenient to *exploit* the obtained solution in order to refine it.

Different evolutionary experiments have been conducted: section 4.6.3 shows the results obtained using a bit-flipping mutation phase while in section 4.6.4 we will show the results obtained using a different mutation phase where mutations are modulated to small values. Section 4.6.5 will describe the differences obtained when trying to minimize the applied forces. Results will be compared with a pure evolutionary approach. All the experiments share the same population size and reproduction phase, their difference is in the fitness and/or in the mutation.

4.6.2 Pure evolutionary experiment

First of all, a pure evolutionary experiment has been carried out. This is useful to have a measure of how much the suggested approach is increasing the learning speed and the performance of the system.

For a complex task as the grasping the choice of a good fitness function is not completely trivial: first of all it is necessary to define a function that gives a good indication on how "good" is the grasp. This is not enough, though: if the fitness keeps into account only the grasp quality it means that, for all the individuals that do not even reach the ball (most of them, especially in the firsts generations) the fitness will be identically 0 and there would be no way to distinguish a promising individual (i.e. one that starts moving towards the ball) from a failing one (i.e. one that is moving in the opposite direction). The suggested fitness is defined as

$$\begin{aligned} f(\Theta) = 0.1 \cdot dist(\Theta) + 0.2 \cdot touch(\Theta) + 0.1 \cdot (openFar(\Theta) \cdot closeNear(\Theta)) \\ +0.5 \cdot grasp(\Theta) + 0.2 \cdot grasp_{succ}(\Theta) \end{aligned} \tag{4.2}$$

where Θ is the current trial. In particular

$$dist(\Theta) = e^{-4\|b-p\|^2} \tag{4.3}$$

gives an indication of how much the robot has come close to the ball at the end of the trial: this value is 1 only if the centroid of the palm coincide with the centroid of the ball. This is, in fact, an ideal value; however, being a monotonically decreasing function, the more the robot comes close to the ball, the higher $dist(\Theta)$ will be. This components is useful to discriminate individuals that move towards the goal from individuals that don't and, thus, is critical in the first phase of the evolution.

The second term of the fitness is defined as

$$touch(\Theta) = \begin{cases} 1 & p_c \geq 10 \\ \sqrt{\frac{p_c}{10}} & p_c < 10 \end{cases} \tag{4.4}$$

where p_c is a variable that stores the number of time steps in which the palm touches the ball. A possible alternative is to return a binary value to encode whether the robot touched the ball or not; however, this solution has the advantage of encouraging the robot to stay close to the ball instead of touching it and recede.

Two additional components have been defined to guide the evolution: $openFar(\Theta)$ and $closeNear(\Theta)$. The first one is defined as

$$openFar(\Theta) = \begin{cases} \frac{f_{ext}}{n_{ext}} & n_{ext} > 0 \\ 0 & n_{ext} = 0 \end{cases} \tag{4.5}$$

where f_{ext} is the average extension of the fingers (scaled between 0 and 1) and n_{ext} the number of time steps in which the fingers should be extended. These two quantities are updated at every time step until the robot touches the ball for the first time. The

second one is analogously defined as

$$closeNear(\Theta) = \begin{cases} \frac{f_{cl}}{n_{cl}} & n_{cl} > 0 \\ 0 & n_{cl} = 0 \end{cases} \tag{4.6}$$

where f_{cl} is the average closure of the fingers (clearly measured as $f_{cl} = 1 - f_{ext}$) and n_{cl} is the number of time steps in which the fingers should be closed. These two quantities are updated at each time step from the one in which $p_c = 5$. Although $openFar(\Theta)$ and $closeNear(\Theta)$ are not strictly needed (the evolution should eventually reach a solution even without these components) they can speed up convergency as the robot will be encouraged to close the fingers only when it is close enough to the ball, keeping them extended for the first part of the trial, obtaining a more natural movement.

The most significative component of the fitness is the one that measure the grasp quality: reach and touch is a relatively simple problem that the robot can quickly solve. It is not trivial, however, to provide a comprehensive evaluation of the grasp quality as this would require the analysis of the contact points classification (which can be either *frictionless*, *hard-finger* or *soft-finger* according to the forces we assume it is possible to apply) and position (which is also dependent on the shape of the object), closure properties and grasp stability (the capability to withstand external disturbances), force equilibrium (a grasp tend to be more stable when the applied forces have similar magnitude) (Bicchi and Kumar (2000); Chella et al. (2007)). In this context, however, an extremely formal qualification of the grasping phase may not be really meaningful, as our focus is not on the perfect grasping but on the learning and combination of different behaviors. The proposed grasp quality function is

$$grasp(\Theta) = \frac{1}{25} \max_t n_f^2(t) \cdot \frac{1}{T} \sum_{t \in T} d_c(t) h(d_c(t)) \tag{4.7}$$

where T is the number of steps in the current trial, $n_f(t)$ is the number of fingers that concurrently touch the ball at time step t, $h(x)$ is the Heaviside step function and d_c

is a function that encodes how much the current grasp can be considered solid. In symbols, let $F(t)$ be the set of the fingers that are in contact with the ball at time step t and $f_i(t)$ the position (in the world frame) of the i-th finger tip, we can write

$$d_c(t) = \left(1 - \left\| b(t) - \frac{1}{|F(t)+1|}(p + \sum_{i \in F(t)} f_i(t)) \right\| \right)^3 h(|F(t)|) \tag{4.8}$$

In spoken words, if the centroid of the tips of the fingers that are in contact with the ball is close to the centroid of the ball itself we can assume that the grasp is solid. However, the position of the palm plays a crucial role in the evaluation of the grasping, so it is necessary to keep into account also its position. $d_c(t)$ provides an indication of how much the centroid of the finger tips and of the palm is close to the centroid of the ball, given that there is at least one finger in contact (see figure 4.11). This quantity is then averaged over all time steps. Besides, in order to encourage the robot to perform grasping with all of the 5 fingers, the sum of $d_c(t)$ is weighted by the square of maximum number of concurrent fingers (the square is useful so that the robot prefers to perform grasping with all of the five fingers rather than for long time intervals). The maximum value $grasp(\Theta)$ can assume is 1; however this is a theoretical limit that requires that, from the first time step, all of the five fingers are in contact with the ball and their centroid coincide with the one of the ball.

The last component of the fitness is a binary value that encodes whether the grasp can be considered successful or not. There is a subtle problem in this evaluation: while in the real world an intelligent teacher has the possibility to physically interact with the robot, in the simulator it may happen that the robot manages to come very close to the ball and close the fingers but, in the moment when the robot tries to lift the ball it falls. Considering only the final posture of the hand, thus, may be misleading. The simplest choice to avoid this is to temporarily disable the collisions with the table: this way the ball will remain close to the robot's hand only if the robot grasped it correctly. It is necessary, however, to keep into account another aspect: in order to perform grasping the robot exploits not only the interactions with the ball but also

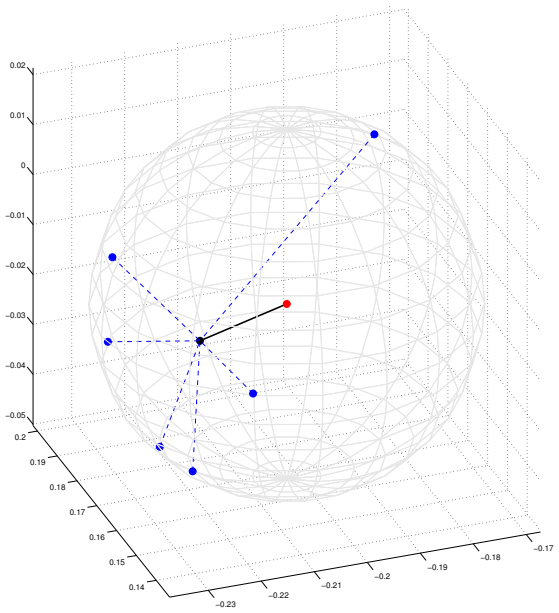


Figure 4.11: A graphical indication of how $d_c(t)$ is computed: blue circles are the contact points between the finger tips and the ball. The blue point that is farthest from the others is the position of the palm. The aim is to measure the distance between the centroid of the ball (red point) with respect to the centoid of the blue points (the black one).

with the table (i.e. it pushes the ball downwards and exploit the reaction force of the table to keep the ball still). By disabling the table it is possible that the robot falls below it (due to the sudden lack of its reaction force), and this cannot be considered as a success, even though the robot keeps holding the ball. $grasp_{succ}(\Theta)$ is then evaluated by enforcing the constraint that both the ball and the palm are above the table and their distance is below a threshold.

The maximum possible value the fitness function can assume is 1.1 (as the image of all the components is $[0, 1]$). However, as for some of the components 1 is an ideal value, we cannot expect individuals to obtain such an high fitness; practical examples showed that a fitness value around 0.8 is indicative of a proper and robust enough movement.

First generation is made of individuals with weights randomly initialized in the range $[0, 255]$. In order to avoid ambiguity, we will talk about the genotype weights,

remembering that the phenotype is obtained by scaling them in the range $[-5, 5]$. Notice that, in this case, there is no conceptual difference between the genetic and the kinesthetic weights (as there is no kinesthetic phase). The neural controller, thus, can be considered as made of two different modules: the neck reflex (hand-coded) and a two-layer net with 14 hidden neurons (responsible for the movement).

The mutation phase consists in randomly flipping the firsts 8 bit, each with a certain probability p. This type of mutation guarantees that each of the weights lies in the range $[0, 255]$. The probability of flipping a bit is initialized to 0.5 and decreases with the number of generations (until a minimum mutation rate, usually set to 0.025). The use of a high initial mutation rate results in meaningful changes in the weights: this is useful to explore the search space and compensate the fact that, in general, the initial population will be very far from a local minimum. As the evolution goes by the possibility of flipping a bit decreases, so most of the weights will remain unchanged. This encodes the fact that, after a sufficiently high number of generation, the population should converge towards a local optimum, thus it is better to make small changes to find a better solution in the neighborhood.

Figure 4.12 shows the trend of the fitness over the generations. Obviously, in the firsts generations most of the individuals will produce non-appropriate behaviors, receiving thus a very low fitness. Some of them, however, show a coherent behavior from the beginning of the evolution (the fitness of the best individual during the firsts generations is around 0.25, which means that the robot managed to reach the ball and, at least in some trials, touch it) and this allows to quickly prune most of the individual that move in the opposite direction with respect to the ball.

The reaching behavior is, actually, the first to appear during the evolutionary process. Nevertheless it is not immediate to acquire a robust reaching and, in general, individuals tend to complete it only in some of the trials (those in which the ball is in a particularly convenient position). It is interesting to note that the learning of the reaching behavior encourage, in turn, the closure of the fingers: since the robot starts with the fingers completely extended, the firsts generations gather an high $openFar(\Theta)$ component. However, as $openFar(\Theta)$ is multiplied by $closeNear(\Theta)$,

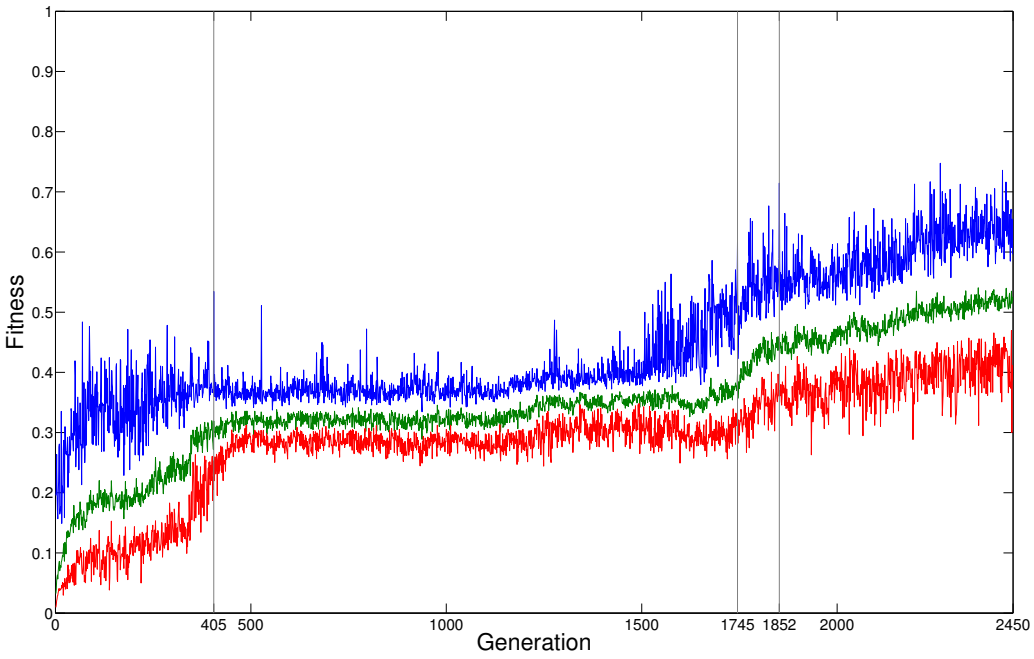


Figure 4.12: The trend of the fitness over the generations in the pure evolutionary experiments. The blue plot shows the fitness of the best individual in the particual generation, the red line represent the worst one and the green shows the average fitness over the population. Vertical lines represent the moments in which the number of trials increased for all individuals: the starting number of trials is 4 and each vertical line correspond to an increase of 4 trials.

only the individuals that eventually close the fingers receive an higher fitness. As some of them soon managed to reach (and, in particular, touch) the ball, the closing of the fingers is quickly identified as a correct behavior. Within generation 40, in fact, all of the individuals show the tendency of closing the fingers (more or less slowly) when in proximity of the table. This may sound odd, but in the first phase the reaching task is very dependent on the ball position; on the other side, the individuals are not yet able to identify the exact moment in which it is necessary to close the fingers, so they tend to associate a particular arm position with the fact that the fingers should be closed.

After the firsts 100 generations, almost all individuals show an approximate reaching behavior: the reproduced trajectory is not always linear and, often, individuals manage to receive an high $dist(\Theta)$ component by making a wide movement with the

arm that allows them to traverse a wide area of the workspace (so that the robot is actually "reaching" almost everything that is on the table). With this kind of behavior, however, the robot tend to hit the ball and make it roll away. Yet, this movement is the starting point for the birth of a particular strategy that allowed the individuals to bypass the grasping problem while obtaining the bonus for the grasp success: the robot first places the hand in a far enough position, then, with a wide movement of the arm, pushes the ball towards its body and locks it with the forearm. This solution has the advantage of being simple and robust with respect to the position of the ball and of guaranteeing the grasp success bonus, as the ball will not fall when the table gets disabled. Besides, in most of the trials the robot will still receive the $dist(\Theta)$ and $touch(\Theta)$ components as the palm is actually very close to the ball (see figure 4.13).

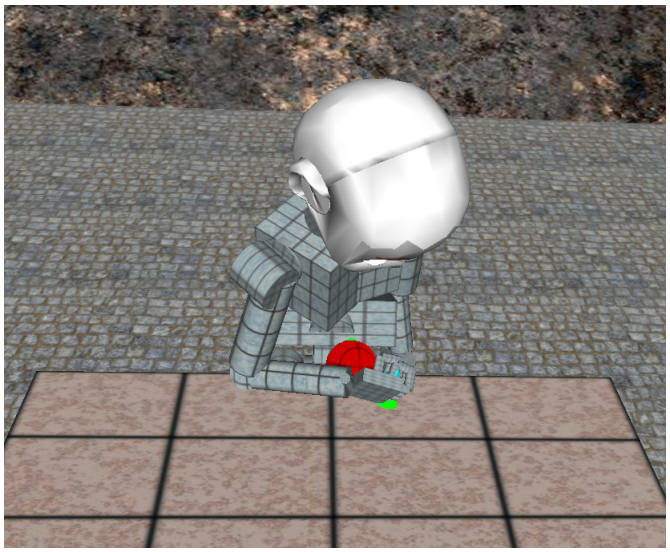


Figure 4.13: The image shows the final posture of the robot when bypassing the grasping phase.

Due to the effectiveness of this approach, there is a phase in the evolution in which the population tend to adopt this new behavior: we see in fact that, while the fitness of the best individual is (on average) around 0.35, the average fitness of all the individuals tend to increase and reach a similar value.

This kind of strategy, however, works only when the ball is in the central part of the table (areas blue and light blue if we refer to figure 4.2), while it fails if the ball is on the left or on the right. Typically, best individuals tend to receive a pretty

high fitness value (around 0.6) when they manage to "lock" the ball (2 trials over 4), while a very low value (0.06) when they don't (the other 2 trials). We observe, however, different peaks in the fitness: if the ball is randomly placed in a convenient position, individuals may manage to "lock" the ball in 3 trials over 4, obtaining a significantly higher fitness. As said before, the evolutionary experiments is designed so that best individuals are tested on more trials, so to reduce the influence of luck. In our experiment, the first individual that goes beyond the threshold of 0.5 appears in generation 405; it is probable that similar individuals had the capacities to obtain the same fitness before but they have been "unlucky".

After generation 405 individuals are tested on 8 trials rather than on 4 and the next threshold is set to 0.6. We can coherently notice that the fitness has smaller fluctuations: this is, as explained, the result of reducing the effect of luck in the trials and proves that all individuals have, more or less, the same capabilities.

It then follows a (long) evolutionary phase in which the robots tend to reinforce the learned behavior. The grassroots movement is the same as described; improvements are made in the position of the wrist and in the finger closure speed. In practice, the population is trying to find the best strategy to obtain also the $grasp(\Theta)$ component, without losing the other fitness' bonuses. Indeed, individuals show the tendency of approaching the ball with the palm placed to the side of the ball (instead of above) and slowly closing the finger while pushing it towards the body. At a later stage we can observe also the tendency of rotating the wrist so that the palm is pointing upward. This way they sometimes manage to obtain some fitness component for the grasping. As the evolution goes by, the movement becomes also wider, so that the robot is able to capture the ball also when it is in the left/right part of the table.

Significative improvements in the behavior are however obtained only when the individuals start to have a less rapid and dynamic movement: since around generation 1500, the robot tend to reproduce a slower and more precise gesture, while improve the timing of the closure of the fingers. This strategy, combined with the final rotation of the wrist, is a much better approach and, sometimes, results in a correct grasping. It is, indeed, meaningful to note that the population overcame the two fitness thresholds

(0.6 and 0.7) in around 100 generations.

There are no qualitative changes until the end of the evolutionary experiment: individuals tend to refine this strategy, trying to close the fingers around the ball and then pushing it against the body while rotating the wrist. However, the learned behavior is not very robust and often the ball tend to slip away due to a non perfect finger closure timing or to a too rapid movement.

This experiment took over 1 month of simulation. As we will see in the next section, the training time can be reduced significantly through the combined use of supervised and trial-and-error learning.

4.6.3 Combined evolution

The first attempt of combining the kinesthetic teaching with the genetic approach has been conducted by using a pre-trained individual as the starting point for the evolution. The initial population was generated by preserving the kinesthetic weights from mutation (maintaining, thus, the value they obtained after the kinesthetic phase) and randomly initializing the genetic weights in the range $[0, 255]$. We adopted, besides, the same mutation and reproduction phase of the pure evolutionary experiment.

Figure 4.14 shows the trend of the fitness for this experiment. As expected, the robot demonstrate a coherent behavior since the very firsts generations and manages to overcome the firsts three fitness thresholds before the first 100 generations (it took more than 1800 generations to reach this level with the pure evolutionary approach). It is interesting to underline, however, that the average fitness of the population (and, especially, the fitness of the worst individual) is much lower than the one of the best individual: this is due to the fact that a random initialization of the genetic weights likely results in significative changes in the overall movement. Considering that the initial movement is close to the desired one, this often results in a worsening rather than in an improvement. The population is actually made of few promising individuals and lot of poor performant ones.

The evolution goes then through a phase of reinforce of the current movement until around generation 500, where a qualitative difference is introduced: after the

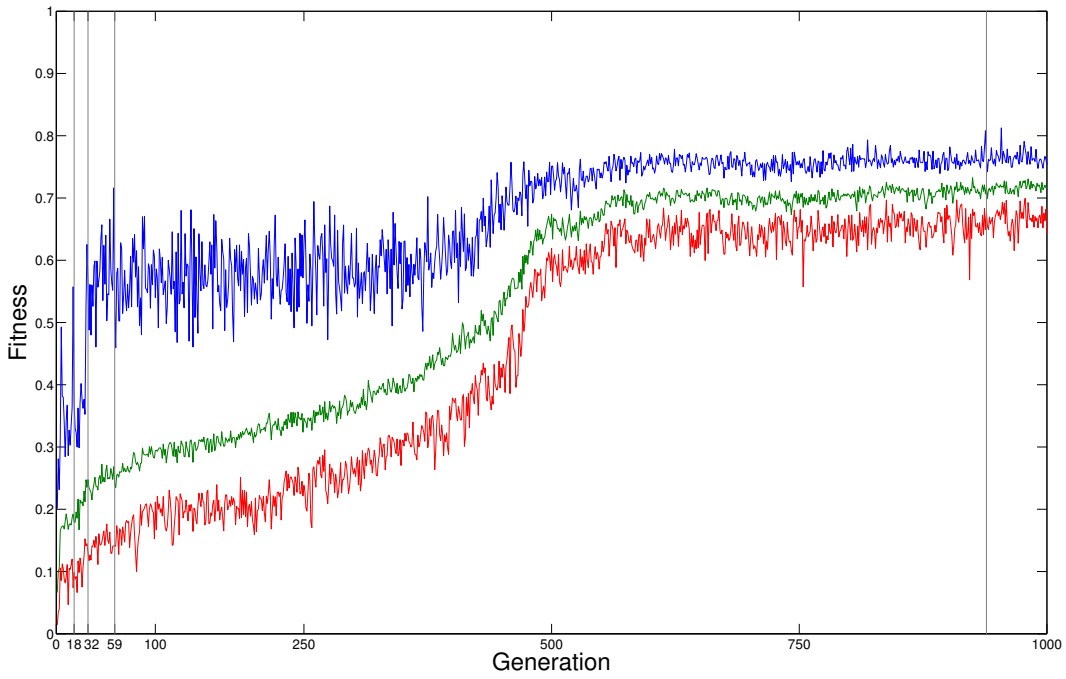


Figure 4.14: The trend of the fitness in the combined evolutionary experiment. The colors have the same meaning of previous plot: blue line represent the best individual, red line the worst one and green line the average fitness over all the individuals of the population.

reaching movement the robot starts to close the fingers and, contextually, to rotate the wrist counterclockwise (pointing the palm towards the right of the robot, see figure 4.15). To explain the reason of this movement it is necessary to consider its physical limitations: the simulated robot (as well as the real one) has a problem in the opposition of the thumb, so that, when the robot close the fingers, the thumb is actually parallel to the index, not opposite to it (see figure 4.15). This type of limitation, added to the fact that the palm is actually a rigid parallelogram (does not allow any kind of deformation or compliancy with respect to the grasped object), makes very hard for the robot to perform grasping as humans do. The tendency of rotating the wrist counterclockwise has the effect of providing a support base for the ball, obtaining a much more stable grasping. It is particularly relevant to underline the relation of this problem with the kinesthetic phase: the movement of the teacher did not include any kind of strategy to solve the "thumb problem". This is deliberate

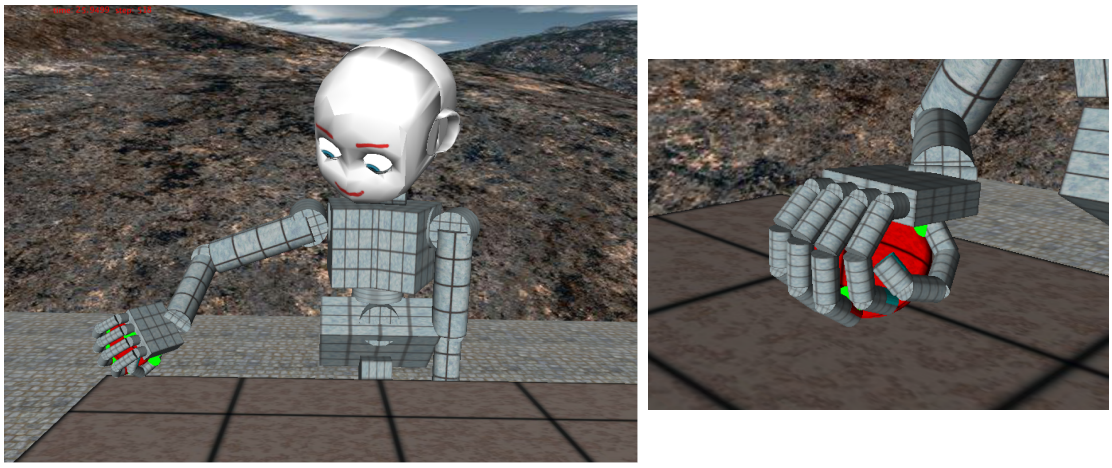


Figure 4.15: On the left, the robot tend to rotate the wrist counterclockwise. On the right, a close view of the thumb opposition.

choice: in general it is very unlikely that the teacher knows every limitation or strength point of the student (they are very particular and hard to foresee). Rather he just limits himself to a formal demonstration of the task; it is then role of the student to refine the movement in order to adapt it to his particular needs. In this case we chose to ignore the robot's limitations and provide a demonstration of how a human would perform grasping, leaving to the evolutionary experiment the role to develop a strategy to overcome them. It is likewise interesting to notice that the need of a base for the ball spontaneously arise from the interaction with the environment: a conceptually similar behavior was present also in the pure evolutionary approach, as the robot tried to rotate the wrist clockwise, pointing the palm upward. Indeed, through a trial and error procedure, the robot is able to identify movements in which the ball has a support base as solutions with a higher success rate.

This adopted solution is then reinforced until the end of the evolution, obtaining a faster and more precise (even if non extremely robust) movement. We can consider the evolution as (almost) stable since around generation 700 (the plot of the fitness have been truncated because there was not any meaningful change in the successive generations). It took around 15 days to complete the firsts 700 generations, that is less than half the time needed for the pure evolutionary experiment (we must remember that, although the number of generations is less than 1/3 of the amount needed in

the pure genetic approach, in that case in most of the generations individuals were tested for 4 or 8 trials, while now almost all of the tests lasted 16 trials).

4.6.4 Combined evolution: small mutations

Despite the clear improvement in the performances, one may wonder why there is still the need of 15 days of training if the result of kinesthetic phase is close enough to the desired behavior. There are two different aspect that should be underlined: on the one hand, refine a behavior is a complex task that requires the learning of the proper motor commands that shall be used to refine the movement by adding small modifications; this problem is most often addressed by trial and error (this is also the way humans train and improve their performances) and usually requires time before obtaining meaningful results. On the other hand, an improvement may be obtained by analyzing the mutation phase: in previous experiments, after each generation, the weights (all of them in the pure evolutionary approach, the genetic ones in the combined experiment) go through a mutation phase that tend to apply significative changes in their value (especially in the early generation, where the probability of flipping a bit is 0.5). Although this is reasonable if the evolution starts from scratch, as it allows to explore very different regions of the search space, applying this kind of modification to pre-trained individuals results in a deep change of the overall behavior (as the kinesthetic weights have been trained with zeroed genetic weights, so the network can produce the learned behavior only in this condition). Our suggestion is to use, indeed, a different mutation phase, where each (genetic) weight is altered by adding a random quantity uniformly distributed in the range $[-10, 10]$. This means that each weight undergoes a "small" mutation, resulting in a form of exploitation and refinement of the current movement.

Figure 4.16 shows the trend of the fitness function when applying this alternative type of mutation. It is clear from the plot how this solution has a much faster convergency. Besides, it is meaningful to underline that a proper grasping arise from the very firsts generations: they actually do not receive a very high fitness value only because the learned behavior is coarse and inaccurate, but it requires very few

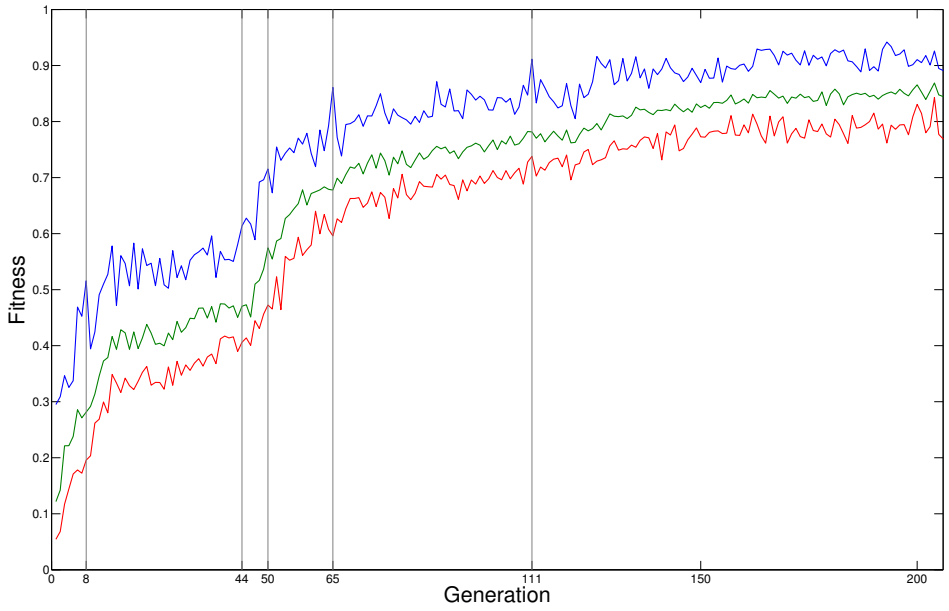


Figure 4.16: The trend of the fitness when evolving the system using modulated mutations.

generations to reach a more robust and precise solution. Notice how, in just 65 iterations of the GA, the best individual receives the same fitness reached only after hundreds of generations in the first combined experiment.

It is pairwise relevant to analyze the results from a qualitative point of view: the evolved system shows a behavior very similar to the one demonstrated during the kinesthetic teaching phase. This highlights two different key aspects: on the one hand, small mutations results in a very fast learning (in case the initial behavior is close enough to the desired one). On the other hand, poor exploration of the search space results in a local minimum (remember that, due to the thumb opposition problem, human-like grasping, with the palm above the ball, is not a very solid solution for the robot). However, if we can assume that the demonstration of the teacher is the "proper" movement and if the behavior learned after the kinesthetic phase is sufficiently good, then applying small perturbations on the solution results in a significantly faster convergency. In this particular case, the firsts 120 generations required less than 3 days to complete: more than 10 times faster than the pure evolutionary

approach. Note besides that the system obtained a much higher performance: using only GAs the system reached a fitness of approximately 0.65. The same level has been reached after only around 50 generations and less than 14 hours of computation.

4.6.5 Combined evolution: forces minimization

An additional experiment has been carried out in order to minimize the forces applied by the robot: it is in fact clear that, given the fitness used, there are no reasons to obtain a "delicate" grasping and, typically, individuals learn to apply high pressures on the ball in order to keep it attached to the palm (as this guarantees a higher success rate). It is interesting to notice how the evolution changes when modifying the fitness function in order to minimize the applied forces.

In particular, we defined an additional fitness component as

$$force(\Theta) = \frac{1}{100T} \sum_{t \in T} \sum_{o \in Obj} \|\mathbf{F}_{N,t}(p, o)\| \tag{4.9}$$

where $\mathbf{F}_{N,t}$ is the normal force between two objects at time t and Obj is the set of all objects in the world (minimized contact forces are not only those between the palm and the ball). The new fitness function is then defined considering a malus for the contact forces:

$$f'(\Theta) = f(\Theta) - 0.05 force(\Theta) \tag{4.10}$$

The problem of finding a proper weight for the forces' malus shall not be underestimated: contact forces are necessary for doing solid grasping. If we put too much importance in minimizing the force, the robot will fail in learning the desired task (of course the solution that minimizes the forces is not to touch anything), while if we consider a too small weight the malus may be absorbed by natural oscillations of the fitness function. We chose this weight so to make the force malus smaller than the smallest fitness bonus: this way, in case some individuals manage to complete

a phase of the task (i.e. lift the ball), the gained bonus should be higher than the forces malus, thus the overall fitness should be greater than the one received from individuals that do not complete it (i.e. do not lift the ball).

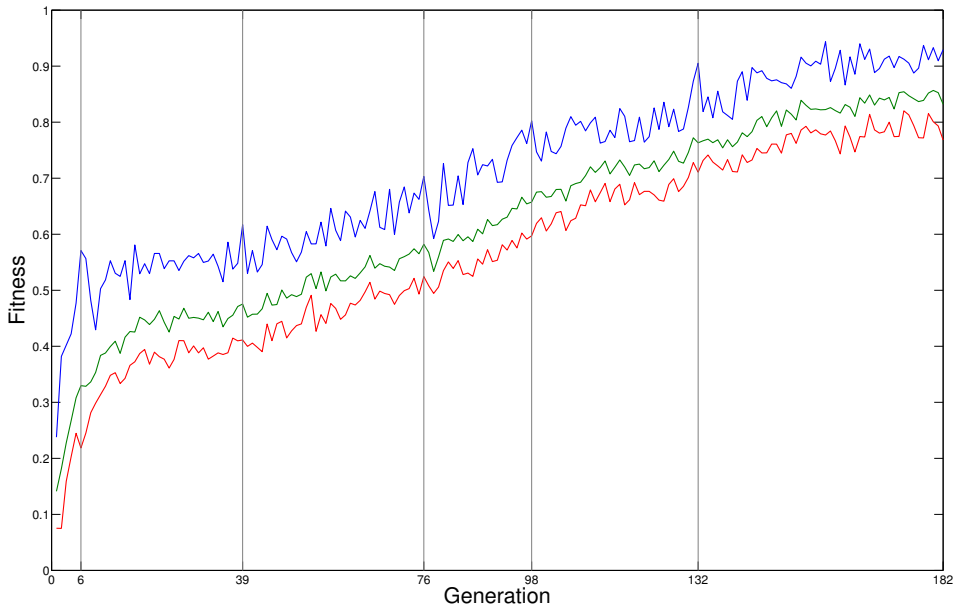


Figure 4.17: The trend of the fitness when using a fitness function that minimizes the contact forces.

Figure 4.17 shows the trend of the fitness during the evolution. As one might expect, the learning is (slightly) slower with respect to the evolution without the force's malus: at early stage, in fact, the presence of a malus for the contact forces discourages from touching the ball, pushing towards a weaker (and less reliable) grasping.

The difference in terms of speed is not meaningful; it is, indeed, interesting to underline the qualitative differences in the two evolutions. While during the firsts generations both experiments show the tendency to acquire a grasping behavior similar to the one demonstrated during the kinesthetic teaching phase, as the evolution goes by the forces minimization experiment tend to deviate from it in favor of the one learned in the first combined experiment (i.e. the one with the rotated wrist). This is reasonable since, as explained before, the kinesthetic demonstration makes no assumptions on the required forces; besides, if we do not consider any malus for

applying too much pressure on the ball, there is no incentive to deviate from a solution that actually guarantees high success rates. When, instead, applied forces play a role in the behavior evaluation, the teacher's demonstration is not good anymore, as high forces correspond to great malus and, thus, to lower fitness, while, at the same time, the reduction of the forces makes the grasping unreliable. It is interesting how though a trial and error procedure (and, especially, through the interaction with the environment) the robot is able to modify its behavior, adopting an equally effective and more "natural" (at least for the robot) strategy.

These differences point out how the learned behavior is significantly affected by the particular choice of the fitness function: it was not obvious that a malus for the forces would result in a *qualitatively* different behavior. Choosing a proper fitness function is, indeed, a very critical aspect of the evolution as the consequences are often hard to foresee.

4.7 Moving

In a last series of experiment described in this section we tried to study a more complex problem in which the robot has to move the object toward the centre of the table after having grasped it. Clearly, the phases of reaching and grasping are the same as those described above but, while before the final goal was just to hold the ball (no matter how and where), now the robot has to move the arm while holding the ball and reach a final position.

It is important to underline that the problem addressed in this and in the previous sections are relatively complex with respect to those investigated in related literature. Typically, in fact, experiments about the problem of grasping an object or moving it are carried out on small robots with poor manipulation capabilities (Ito et al. (2006); Calinon and Billard (2007); Guenter et al. (2007)). This is a key aspect for this kind of experiments: the workspace of these robots is, actually, very limited. Moreover, their fingers are not well suited for grasping (in some cases they are not even provided with fingers but just small pliers), so usually the robot is taught to hold and move relatively

big (with respect to the robot itself) objects. This implies that small variations on the robot behavior do not affect significantly the outcome of the action and, thus, learning the correct behavior is much simpler.

Another important aspect is about the problem of *how to imitate*: complex tasks are often encoded at the action-level, i.e. the robot learns to reproduce the demonstrated trajectory. Although very sophisticated approaches have been developed for this kind of problems (Kormushev et al. (2010)), they are not very well suited for the problem we deal with as the correct trajectory is highly dependent on the position of the ball.

Besides the moving problem presents an additional level of complexity with respect to the reaching/grasping ones: the reaching task is actually pretty simple, as it is monolithic. The grasping, instead, is made of two different phases. As explained before, difficulties arises when trying to associate similar sensory inputs with different motor outputs: this happens in proximity of the ball, where reaching and grasping overlaps. This problem is particularly meaningful for the moving task, where the robot, after the grasping is successful, must be able to lift the ball and move it to the final target. In this case, in fact, there are three phases that overlaps in the vicinity of the ball. Besides, the final position (the center of the table) is often close to the initial position of the ball, inducing another source of ambiguity. Problems with different overlapping phases are often addressed by mean of recurrent neural networks (Yamashita and Tani (2008); Ito et al. (2006); Massera et al. (2010)): they differ from feedforward ANN in that the outputs do not depend only on the current inputs but also on the current state (i.e. previous inputs). RNN, thus, do not approximate functions but dynamical systems. In the cited works, recursion is obtained by defining additional output units aimed at identify the phase in which the robot is in (a sort of internal conceptualization) and using the output value of this neurons as additional input for the network itself (sometimes as input for the hidden neurons). This has been proven to be an effective strategy for discriminating similar sensory inputs on the base of what the robot is currently doing.

Due to the complexity of this task, a lot of different experiments have been carried

out. In the following we will describe the results of the best ones. For most of them the neural network used is the same of the one used for the grasping problem (reported here again for simplicity).

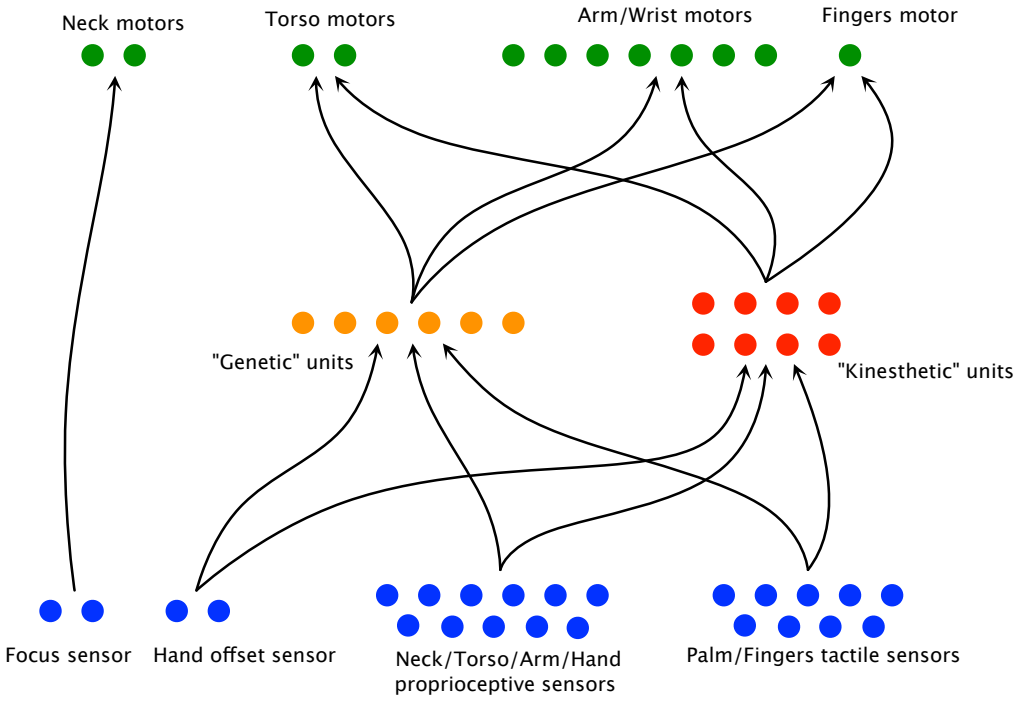


Figure 4.18: The architecture of the network for the grasping/moving problem.

4.7.1 Kinesthetic teaching

The kinesthetic demonstration consists in 5 different phases: the firsts three are exactly the same of the grasping (reach $\rightarrow$ grasp $\rightarrow$ lift). After lifting the palm holding the ball, the robot is horizontally (keeping the same height) pushed towards the center of the table (phase 4) and finally the robot is guided to lay the ball on the table (phase 5). During the final phase, the reference position for the fingers is reset so to open the hand.

Different training sets have been generated, with differences in the length and number of the trials as well as on the anticipation term. Sufficiently good results have been obtained using a set generated over 16 trial of 370 steps each and $\tau = 40$. The anticipation term has been slightly reduced as its influence is particularly relevant

in this context: due to the difficulties explained above, the risk of mixing different behaviors and/or skip parts of them is very high. Reducing the anticipation results in a more precise movement (even though slower).

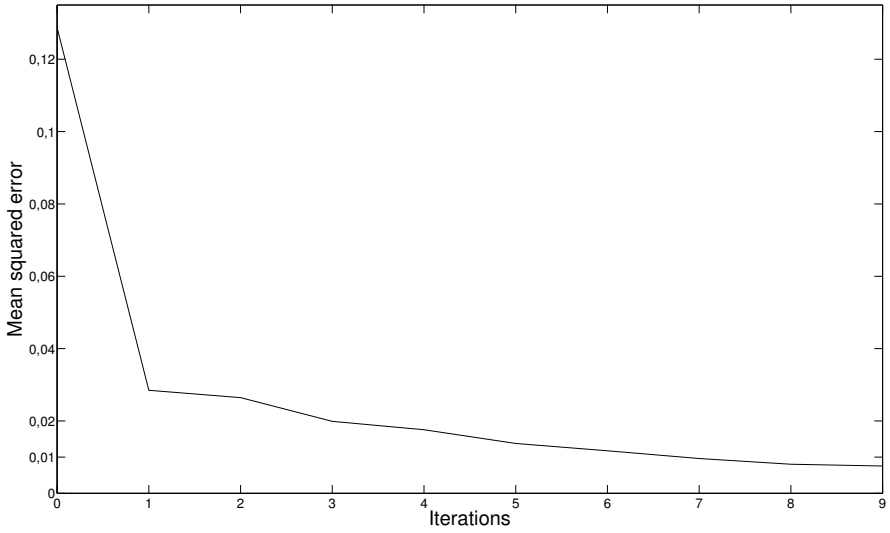


Figure 4.19: The plot of the mean squared error over the number of iterations for the moving problem.

Figure 4.19 shows the plot of the error during the Levenberg-Marquardt training. Notice, however, that a very low error do not correspond to a perfect movement: it just means that, given some particular sensory input, the ANN responds with the demonstrated output. In practice, in fact, the reproduction of the task is not correct: due to a drift phenomena, the experienced stimuli are often slightly different from the one demonstrated, which, in turn, correspond to slightly different motor commands. Besides, the movement is not as fast as the one demonstrated, often resulting in an overall failure.

Testing the robot immediately after the kinesthetic phase we see that the reaching is correctly learned but the robot often fails in reproducing the grasping, sometimes skipping that phase and pointing directly towards the final goal. As a counter-proof, some tests have been performed by rendering, at the same time, a virtual robot that instantaneously reach the desired positions set by the neural controller. In practice, this is useful to have a visual indication of where the robot is heading (see figure 4.20).

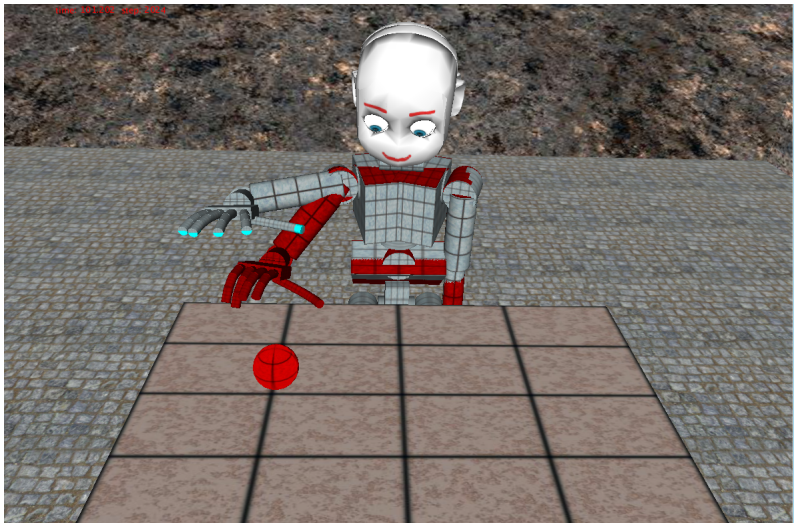


Figure 4.20: A screenshot of the simulator, showing the real robot (gray) and, overlayed, the virtual robot (red). Notice how the fingers of the virtual robot are not fully extended (indicative of the fact that the robot overcome the phases segmentation.

Tests show how the virtual robot is correctly going through all the phases; however, the behavior actually shown by the robot is slightly different and tend to go towards the center of the table before reaching the ball. As explained, this is a consequence of the great ambiguity that is present at the center of the table.

4.7.2 Evolutionary experiments

Before describing the various evolutionary experiments carried out it is necessary to introduce another fitness component, designed to give an reward for the completion of the moving phase:

$$move(\Theta) = e^{-4\|c-b\|} \tag{4.11}$$

where c is the center of the table. A lot of different fitness functions have been used (by combining in a different ways the described components); the exact formulation will be reported along with comments of the obtained results. In general, however, it has been simplified: the components $touch(\Theta)$, $openFar(\Theta)$ and $closeNear(\Theta)$ have been removed in all the evolutions. This was motivated by the fact that the fitness

was becoming too complex and establish the proper weight for all of its components was not trivial (a bad choice can easily result in an overall failure, as it would mean to give too much importance to some aspect and too few to others); besides, as explained before, it is desirable to have a fitness function as general as possible, in order not to put constraints on the evolution. Finally, the kinesthetic phase already provides "hints" for closing the finger when in the vicinity of the ball and the touching is an essential aspect of the grasping (its bonus can be then integrated in the grasping one).

The trial has a slightly different structure with respect to the grasping experiment: in the previous case, the robot was let free to "live" for almost all the trial duration and, few steps before the end of it, the table was disabled in order to check the successfulness of the grasping. In this case, however, we cannot apply the same strategy as, after the grasping phase, the robot will need to interact with the table in order to put down the ball. The simplest solution, in this case, was just to disable the table after a certain number of time steps (typically 200) and re-enable it after a while (typically after 20 steps). This choice has two effects: first of all, it allows to discriminate trials in which the grasping succeeded from trials in which doesn't. On the other side, individuals that do not manage to grasp the ball have no possibility to complete the moving, thus failing the grasp implies failing the trial. Then, if, when the table is re-enabled, the ball is below the table (it means that it didn't fall and the only possibility is that the robot is holding it), the whole trial can be considered failed and, thus, stopped. This allows to gain a lot of computation time, as in early phases of the evolution, most of the individuals will fail the grasping.

4.7.3 Escaping from local minimum

For the firsts experiments the fitness was defined as

$$f_{m,1}(\Theta) = 0.1 \cdot dist(\Theta) + 0.3 \cdot grasp(\Theta) + 0.2 \cdot grasp_{succ}(\Theta) + 0.5 \cdot move(\Theta) \quad (4.12)$$

One may argue that the $dist(\Theta)$ component is not really needed as the grasping implies the reaching, so there is no need to explicitly provide a reward for coming

closer to the ball. Although it is actually not necessary (as, after the kinesthetic phase, individuals are usually able to reach the ball), it helps during the firsts generations, providing an instrument to discriminate individuals that do not even touch the ball. The $dist(\Theta)$ is evaluated once per trial, just before disabling the table (clearly, evaluating it in a successive moment will not be meaningful, as, in case the grasping is not complete, the ball will fall and the individual will receive a low fitness even though it came very close to the ball). Analogously, the $move(\Theta)$ component is evaluated once at the end of the trial.

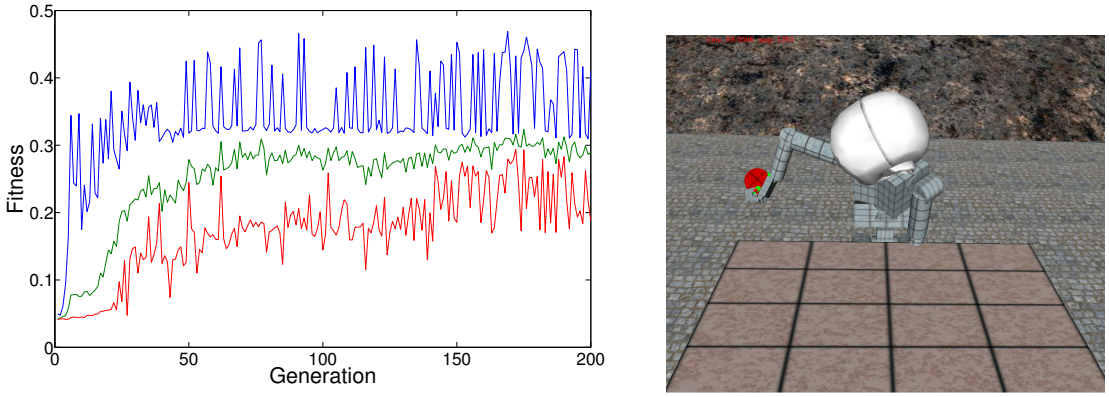


Figure 4.21: On the left, the trend of the fitness during the evolution. On the right, a screenshot of the robot while demonstrating the "local minimum" behavior.

It is interesting to notice that the grasping behavior emerges from the very firsts generations. This confirms the fact that, although the initial behavior is not correct, the robot correctly learned how to perform grasping and is able to react coherently when coming in the vicinity of the ball. However, the acquired behavior is very far from the desired one: it consists in closing the fingers around the ball and then pull the ball towards itself while moving the elbow upwards (see figure 4.21). This is a variation of the "solid grasping" behavior learned in the experiments on the sole grasping: while before the wrist was rotated so that the thumb provided a base for the ball, now the base is provided by the other four fingers. Experiments show that this position, even though very good for the grasping, prevent the robot from reconfigure itself in order to place the ball at the goal position. There are, indeed, different possible moving that the robot may do to complete the task in spite of its position;

however, they typically require a significative reconfiguration so they are not easy to achieve, especially considering that little deviations often results in a lower fitness, as the ball may fall during reconfiguration (exactly the meaning of local minimum).

A first explanation for this (unexpected) behavior can be found by analyzing the reward the robot receives: being defined in this way, the higher the number of steps in which the robot is holding the ball the higher the whole fitness. Essentially, the robot tends to maximize the grasping quality while (almost) ignoring the $move(\Theta)$ component. In other words, the found solution is a local minimum, as the grasping quality is actually very high and pretty reliable (it manages to grasp the ball almost in every step), but, of course, the moving task is completely failed.

In order to solve this problem, two modifications have been applied to the fitness function: first of all, the robot does not receives grasping bonuses during all the trial but only until the table is disabled. After that moment, either the grasp is successful (and so the successive increments of the fitness are only thanks to the $move(\Theta)$ component) or it is not (and the trial is interrupted). The second modification consists in integrating the value of $move(\Theta)$ from the time step in which the table is re-enabled. As a result, individuals are encouraged to head directly towards the goal (the sooner they reach the target, the higher the overall fitness).

Despite these modifications, the successive experiments show no qualitative difference with respect to the one described above: a sufficiently reliable grasping is soon acquired but the robot falls in the "local minimum" behavior and is not able (even after long evolutions) to successfully escape from it.

Lots of other variations have been tried: some evolutionary experiments were launched using a fitness where the $grasp(\Theta)$ component was multiplied by the $move(\Theta)$ one. This way individuals that perform grasping bringing away the ball with respect to the center of the table were discouraged in favor of those that grasp the ball remaining in the vicinity of the goal. Another attempt consists in constraining the grasping to be "local" with respect the initial position of the ball (by resetting the bonus if it happens outside of a relatively small neighborhood). Even though this is, to some extents, against the philosophy of GAs, the adopted solution consisted

in violently pushing the ball towards the table (so that friction actually prevents the ball and the arm to move away from this position). Consequently, the sudden disable of the table makes the robot falls below it.

4.7.4 Launching the ball

The failure of a so wide variety of different fitness functions suggests the need to explore different regions of the solution space. In practice, the strategy to apply small mutations was thought because the solution provided by the kinesthetic phase was already a good approximation of the optimal one. If we compare the results obtained for the reaching/grasping with the ones obtained for the moving, it should be clear that in the formers there was just a quantitative error (too slow/fast movement, sometimes inaccurate), while in this case the error is qualitative (the robot head directly towards the center of the table, skipping the grasping). Small mutations tend, thus, to be ineffective, as they do not encourage the exploration of different solutions. On the other side, we must remember that the fact that the showed behavior is distant from the desired one doesn't mean that the network didn't learn anything (as already explained). These considerations led to the adoption of a novel mutation strategy, consisting in randomly (with probability 0.5) choosing between the "bit-flipping" mutation phase (the one described in section 4.6.3) and the "small mutations" one. Intuitively, in the early stages of the evolution, bit-flipping mutation may results in meaningfully different behaviors; however, (in case they obtain very poor performances) they are soon discarded in favor of those that, being mutated of a small amount, tend to maintain the learned behavior.

An evolutionary experiment with this kind of "mixed" mutation phase has been launched. The fitness function used, in this case, is

$$\begin{aligned} f_{m,2}(\Theta) = 0.1 \cdot dist(\Theta) + 0.3 \cdot grasp(\Theta) + 0.1 \cdot grasp_{succ}(\Theta) \\ +0.5 \cdot move(\Theta) - 0.05 \cdot force(\Theta) \end{aligned} \tag{4.13}$$

where at he first lifting step (i.e. the first step in which the ball is lifted of a certain - very small - amount) the $grasp(\Theta)$ component is interrupted and the $move(\Theta)$ one is enabled (both are evaluated at every time step).

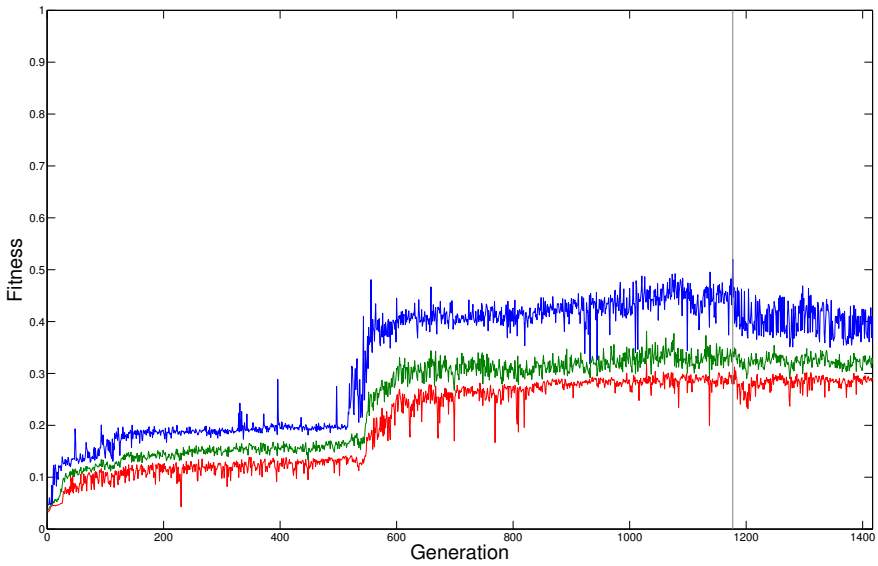


Figure 4.22: The trend of the fitness function in the evolutionary experiment with "mixed" type mutation.

Figure 4.22 shows the trend of the fitness during the evolution. The step around generation 500 highlights a qualitative change in the behavior: in the initial phases most of the individuals tend to perform grasping by pushing the ball downwards (against the table). This behavior is discouraged, at the same time, by the force component (discouraging the pressure) and by the lack of the bonus for the grasp successful (as when the palm falls below the table the task in considered failed). However, while the grasp successful bonus is binary, the force malus is continuous, and this guides the robot through a gradual reduction of the pressure until the moment in which, after the grasping, the robot starts to lift the ball (instead of pushing it downwards).

Still missing is, however, the final phase of the task: after the grasping the robot shall move the ball towards the center of the table. Although it may seems a short step towards the completion of the task, this would require a sharp inversion of the

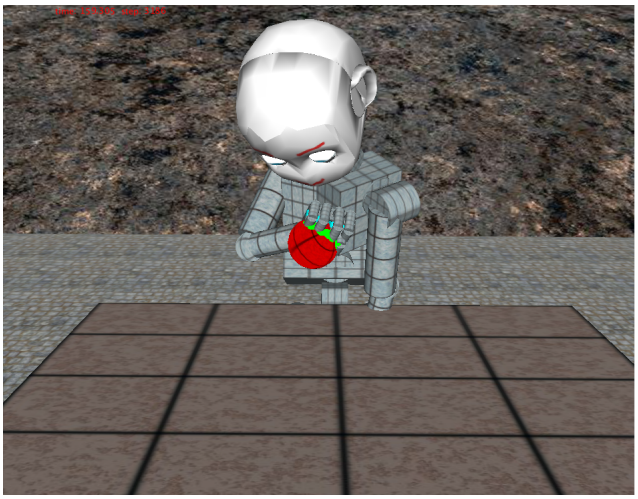


Figure 4.23: The position of the robot before "launching" the ball.

direction of the movement without any significant change in the sensory stimuli. The adopted strategy consists, instead, in continuing the lifting movement of the palm while slightly rotating it so to make the palm vertical (see figure 4.23). From this position the robot tends to slightly open the fingers so to let the ball fall. This has two main consequences: first of all, by leaving the ball the robot eliminate the malus for the contact forces (clearly it is not touching anything anymore), secondly the distance between the ball and the center of the table decreases (thus the robot obtains a higher $move(\Theta)$ component).

The evolution continues, then, to reinforce this "launching" behavior, essentially by refining the timing and/or the velocity of the movement. The slight decrease of the fitness at the end of the evolution shall not be seen as a decrease of the performances but, since around generation 1200 a lucky individual managed to throw the ball close to the center of the table, successive generations were tested on 8 trials.

Chapter 5

Conclusions and future work

The aim of this work was to show how it is possible to combine different learning paradigms in order to exploit different types of informations, reducing the learning time and improving the quality of the results. In particular, the system is initially trained using a demonstrative approach (in particular kinesthetic teaching), strategy widely used for its efficiency and intuitiveness. This first learning phase is useful to quickly acquire the grassroots of the correct movement, which is then refined using a trial-and-error method (in particular an evolutionary approach) exploiting a goal-oriented feedback. To test the applicability of this strategy we carried out a series of experiments in which a simulated iCub robot have been trained for the ability to manipulate object (i.e. for the ability to display structured action formed by sequence of elementary behaviors such as reach, touch, grasp, lift, move).

Experiments showed how it is possible to exploit kinesthetic teaching to easily generate a training set by storing experienced sensory stimuli and correspondents joints positions. Choosing a proper anticipation term, the use of the Levenberg-Marquardt back-propagation yields to very good results, both in terms of convergency speed and quality of approximation. Simple problems (as the reaching one) can actually be solved just by using the teacher's demonstration.

When dealing with more complex problems, the kinesthetic phase is often not sufficient, especially for task with a high variability: the scenarios we considered are more complex than the ones commonly analyzed in literature as, usually, grasping

tasks are performed in small workspaces with big (with respect to the robot) objects (clearly inducing a simplification). In our experiments, instead, fine tuning of motion is necessary to achieve the desired goal.

The combination of programming by demonstration and trial and error learning realized thorugh an evolutionary robotics method allowed to obtain a much more precise and robust behavior (with respect to the one learned after the sole kinesthetic phase) while drastically reducing the time necessary to converge (with respect to a learning approach based only on genetic algorithms). Besides the refinement of the movement through a trial and error procedure allowed to overcome the intrinsic limitations of the teacher's demonstration: the robot has been, in fact, able to automatically adopt novel (undemonstrated) strategies to maximize its performance, exploiting its weak/strong points (typically not known in advance by the teacher, which is asked to provide only a "formal" solution of the problem). This underlines the critical role that the teacher's demonstration plays during the learning: the behavior acquired after the kinesthetic phase highly affects the effectiveness of the trial-and-error phase as, clearly, the more it is close to the "optimal" one the shorter will be the time needed for the refinement. However, the need to adapt the demonstration to the physical characteristics of the robot (obtaining a qualitatively different movement) reduces the advantages obtained by combining different learning phases (a good teacher is, in fact, considered very valuable, not only in the context of programming by demonstration!).

The use of a two-layer feedforward network was a deliberate choice: simplifying a system is the best way to understand its limitations. Indeed results suggest that the critical problem of the learning is the discrimination of similar sensory inputs. The moving problem, in fact, is the one that required greater effort: thanks to the kinesthetic phase the robot demonstrated the capability of grasping an object from the very firsts generations, however move the ball to the center of the table would require a sharp change in the current movement. A sudden inversion is very hard to achieve, especially if we consider that there is almost no difference between the sensory stimuli perceived during or immediately after the grasping.

One the possible approaches to overcome this problem is to use a recurrent neural

network where a group of output units are responsible to encode the phase of the movement in which the robot is. This results in a sort of semantic segmentation of the task: this output units can be used in a feedback loop, so that the robot is able to discriminate the phase in which it is in, and coordinate its movement coherently. It is besides interesting to underline that, per se, this solution do not prevent the use of the methods described in chapter 2: using a strategy similar to Ito et al. (2006), the recurrent connection can be ignored during the kinesthetic phase (assuming that proper task segmentation is provided by the teacher) and enabled during the evolutionary experiment (in general, when testing the performances of the system).

Another possibility is to use a fully recurrent neural network and train it with BACK-PROPAGATION THROUGH TIME, a variation of the classic back-propagation that consists in unfolding the recursion and keep into account the influence that previous inputs have on the current error (in order to compute the update for each weight). This strategy have been proved effective when training RNN, although it is notoriously highly dependent on the particular choices of the parameters.

In general, other network architectures can be also taken into account: from ANNs with more than two layers to more sophisticated solutions (for example nets in which the kinesthetic neurons are not directly connected to the output units but only through the genetic ones, that will have the role of modulation). It may be interesting to consider the possibility of attentional neurons: when the grasp is completed it is more useful to look toward the goal rather than keep looking the hand.

There are some strategies that may be interesting to analyze in order to improve the effectiveness of the kinesthetic phase (and, thus, provide the GAs with a better initial guess): a possibility is to generate the training set without disabling the low-level controllers of the robot. In practice, this means that the robot will try to execute the task *while* the teacher is guiding it. The difference between the direction of the movement imposed by the teacher and the one learned by the robot generates an error which can be used to train the net. This means, besides, that the robot is trained only for the phases of the task that the robot didn't learn correctly: the intelligent

teacher may choose not to constrain the movement on a particular path during the reaching phase (allowing the robot to slightly deviate from the straight line) while applying higher forces when the robot tend to drift. This kind of teaching may be approximated using a simulated system by dividing the whole task in subtasks and letting the robot be guided by its neural network during the correctly learned phases while switching to "teacher mode" (i.e. disabling the controller and move the robot kinesthetically) in the phases that the robot is not able to complete.

An alternative approach may consists in alternating kinesthetic and genetic training phases: instead of using a single (relatively) long evolution, one may consider using a shorter evolutionary phase followed by successive kinesthetic phases, possibly focused on unlearned subtasks. This recall the way a teacher, after making a first demonstration, observes the student during its attempts in reproducing the movement, eventually providing him with successive (more focused) demonstrations aimed at highlighting the errors.

A variation of this strategy consists in considering an additional fitness component that encodes the error with respect the kinesthetic teaching phase. Essentially this should constrain the evolution so to remain as close as possible to the demonstration of the task. The critical and non-trivial aspect of this approach is how this error should be defined in order to provide a meaningful evaluation of the distance with respect to the demonstration (after the kinesthetic phase the error is already very low and clearly it tends to increase during the evolution).

In conclusion, the combination of kinesthetic teaching and evolutionary methods yields to drastic improvements in term of performances, but still a lot of different aspects should be analyzed in order to make it effective for a wide variety of different tasks.

Bibliography

Aleotti, J., Caselli, S. and Reggiani, M. (2004), 'Leveraging on a virtual environment for robot programming by demonstration'.

Argall, B. D., Chernova, S., Veloso, M. and Browning, B. (2009), 'A survey of robot learning from demonstration', *Robot. Auton. Syst.* **57**(5), 469–483.

Bicchi, A. and Kumar, V. (2000), Robotic grasping and contact: A review, pp. 348–353.

Billard, A., Calinon, S., Dillmann, R. and Schaal, S. (2008), Robot programming by demonstration, *in* 'Springer Handbook of Robotics', pp. 1371–1394.

Billing, E. and Thomas, H. (2010), 'A formalism for learning from demonstration', **1**, 1–13.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, Inc., New York, NY, USA.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bongard, J. and Lipson, H. (2005), 'Nonlinear system identification using coevolution of models and tests', *Evolutionary Computation, IEEE Transactions on* **9**(4), 361–384.

Brooks, R. (1991), 'Intelligence without representation', *Artificial Intelligence* **47**, 139–159.

Brooks, R. (1992), Artificial life and real robots, *in* 'Proceedings of the First European Conference on Artificial Life', MIT Press, pp. 3–10.

Calinon, S. and Billard, A. (2007), Active teaching in robot programming by demonstration, *in* 'Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on', pp. 702–707.

Chella, A., Dindo, H., Matraxia, F. and Pirrone, R. (2007), A neuro-genetic approach to real-time visual grasp synthesis, *in* 'Neural Networks, 2007. IJCNN 2007. International Joint Conference on', IEEE, pp. 1619–1626.

Cliff, D., Husbands, P. and Harvey, I. (1993), 'Explorations in evolutionary robotics', *Adapt. Behav.* **2**(1), 73–110.

Dorigo, M. and Schnepf, U. (1993), 'Genetics-based machine learning and behavior-based robotics: a new synthesis', *Systems, Man and Cybernetics, IEEE Transactions on* **23**(1), 141–154.

Floreano, D., Husbands, P. and Nolfi, S. (2008), Evolutionary robotics, *in* 'Springer Handbook of Robotics', pp. 1423–1451.

Floreano, D. and Mattiussi, C. (2008), *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*, Intelligent Robotics and Autonomous Agents Series, Mit Press.

Friedrich, H. and Dillmann, R. (1995), 'Robot Proghramming Based on a Single Demonstration and User Intentions'.

Grollman, D. and Billard, A. (2011), 'Donut as i do: Learning from failed demonstrations', In Proceedings of IEEE International Conference on Robotics and Automation.

Guenter, F., Hersch, M., Calinon, S. and Billard, A. (2007), 'Reinforcement learning for imitating constrained reaching movements', *RSJ Advanced Robotics* pp. 1521–1544.

Hoffmann, F. and Pfister, G. (1996), 'Evolutionary learning of a fuzzy control rule base for an autonomous vehicle'.

Ikeuchi, K. and Suchiro, T. (1992), Towards an assembly plan from observation. i. assembly task recognition using face-contact relations (polyhedral objects), *in* 'Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on', pp. 2171–2177 vol.3.

Ito, M., Noda, K., Hoshino, Y. and Tani, J. (2006), 'Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model'.

Kim, S., Gribovskaya, E. and Billard, A. (2010), Learning Motion Dynamics to Catch a Moving Object, *in* '10th IEEE-RAS International Conference on Humanoid Robots'.

Kober, J. and Peters, J. (2012), Reinforcement learning in robotics: a survey, *in* 'Reinforcement Learning', Springer Berlin Heidelberg, pp. 579–610.

Kormushev, P., Calinon, S. and Caldwell, D. (2010), Robot motor skill coordination with em-based reinforcement learning, *in* 'Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on', pp. 3232–3237.

Kormushev, P., Nenchev, D., Calinon, S. and Caldwell, D. (2011), Upper-body kinesthetic teaching of a free-standing humanoid robot, *in* 'Robotics and Automation (ICRA), 2011 IEEE International Conference on', pp. 3970–3975.

Kuniyoshi, Y., Inaba, M. and Inoue, H. (1994), 'Learning by watching: Extracting reusable task knowledge from visual observation of human performance', *IEEE Transactions on Robotics and Automation* **10**, 799–822.

Leugger, T. and Nolfi, S. (2012), Action development and integration in a humanoid icub robot how language exposure and self-talk facilitate action development, *in* 'COGNITIVE 2012, The Fourth International Conference on Advanced Cognitive Technologies and Applications', pp. 24–30.

Massera, G., Cangelosi, A. and Nolfi, S. (2007), 'Evolution of prehension ability in an anthropomorphic neurorobotic arm', *Frontiers in Neurorobotics* **1**(4).

Massera, G., Tuci, E., Ferrauto, T. and Nolfi, S. (2010), 'The facilitatory role of linguistic instructions on developing manipulation skills', *Computational Intelligence Magazine, IEEE* **5**(3), 33–42.

McCulloch, W. S. and Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *Bulletin of Mathematical Biology* **5**(4), 115–133.

Metta, G., Sandini, G., Vernon, D., Natale, L. and Nori, F. (2008), The icub humanoid robot: an open platform for research in embodied cognition, *in* 'Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems', PerMIS '08, ACM, New York, NY, USA, pp. 50–56.

Mitchell, T. (1997), *Machine Learning*, McGraw-Hill.

Nguyen, D. and Widrow, B. (1990), Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, *in* 'Neural Networks, 1990., 1990 IJCNN International Joint Conference on', pp. 21–26 vol.3.

Nolfi, S. (2002), 'Power and the limits of reactive agents', *Neurocomputing* **42**(1), 119–145.

Nolfi, S. and Floreano, D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, A Bradford book, MIT Press.

Nolfi, S., Floreano, D., Mondada, F. and Miglino, O. (1995), 'Robotica evolutiva: metodologia e prospettive.', *Sistemi Intelligenti* **7**(2), 203–221.

Nolfi, S. and Marocco, D. (2002), Active perception: A sensorimotor account of object categorization, MIT Press, pp. 266–271.

Rosenblatt, F. (1962), *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*, Report (Cornell Aeronautical Laboratory), Spartan Books.

Rosenstein, M. T. and Barto, A. G. (2004), 'Supervised actor-critic reinforcement learning'.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986), Parallel distributed processing: explorations in the microstructure of cognition, vol. 1, MIT Press, Cambridge, MA, USA, chapter Learning internal representations by error propagation, pp. 318–362.

Saunders, J., Nehaniv, C. L. and Dautenhahn, K. (2006), Teaching robots by moulding behavior and scaffolding the environment, *in* 'IN HUMAN-ROBOT INTERACTION', ACM Press, pp. 118–125.

Siciliano, B., Sciavicco, L., Villani, L. and Oriolo, G. (2011), *Robotics: Modelling, Planning and Control*, Advanced Textbooks in Control and Signal Processing, Springer.

Smart, W. and Kaelbling, L. (2002), Effective reinforcement learning for mobile robots, *in* 'Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on', Vol. 4, pp. 3404–3410 vol.4.

Tung, C. and Kak, A. (1995), Automatic learning of assembly tasks using a dataglove system, *in* 'Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on', Vol. 1, pp. 1–8 vol.1.

Urzelai, J. and Floreano, D. (2001), 'Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments', *Evolutionary Computation* pp. 495–524.

Widrow, B. and Hoff, M. E. (1960), Adaptive Switching Circuits, *in* '1960 IRE WESCON Convention Record, Part 4', IRE, New York, pp. 96–104.

Wilamowski, B. and Irwin, J. (2011), *The Industrial Electronics Handbook: Intelligent systems*, Industrial Electronics, Taylor & Francis Group.

Wrede, S., Emmerich, C., Grünberg, R., Nordmann, A., Swadzba, A. and Steil, J. (2013), 'A user study on kinesthetic teaching and learning for efficient reconfiguration of redundant robots', *Journal of Human-Robot Interaction* **2**.

Yamashita, Y. and Tani, J. (2008), 'Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment', *PLoS computational biology* **4**(11).