Aalto University
School of Science and Technology
Dep. of Biomedical Engineering and Computational Science

María Isabel Cordero Marcos

Learning Sensorimotor Abstractions

Final project submitted in partial fulfillment of the requirements for the double degree in Informatics and Telecommunications Engineering.

Espoo, 24.11.2010

Supervisor: Professor Jouko Lampinen

Instructor: Dr. Tech. Harri Valpola

Aalto University School of Science and Technology Faculty of Information and Natural Sciences	ABSTRACT OF THE FINAL PROJECT

Author: María Isabel Cordero Marcos

Title: Learning sensorimotor abstractions

Title in Finnish: --

Degree Programme: Erasmus - Computer Science and Engineering

Major subject: Telecommunications and Informatics	Minor subject: --

Chair (code):

Supervisor: Jouko Lampinen	Instructor: Harri Valpola

Abstract:
In order to interact with real environments, performing daily tasks, autonomous agents (as machines or robots) cannot be hard-coded. Given all the possible scenarios and, in each scenario, all the possible variations, it is impossible to take into account every single situation that the autonomous agent may encounter. Humans are able to interact with the changing world using as a guidance the sensory input perceived.

Thus, autonomous agents need to be able to adapt to a changing environment. This work proposes a biologically inspired solution that allows the agent to learn representations and skills autonomously that prepare the agent for future learning tasks.

The biologically inspired solution proposed here, called a cognitive architecture, follows the hierarchical architecture found in the cerebral cortex. This model permits the autonomous agent to extract useful information from the sensory input data it receives. The information is coded in abstractions, which are invariant features found within the input patterns. The cognitive architecture uses slowness as a principle for extracting features. In principle, unsupervised learning algorithms based on slowness try to find relevant and slowly changing data. This information could be useful for self evaluation.

The agent tries to learn how to manipulate the sensory abstractions, by linking those to the motor ones. This allows the robot to find the mapping between the motor actions it is taking and the changes it is able to produce in the surrounding environment.

Using the cognitive architecture, an example will be implemented. An agent, who knows nothing about the environment it is placed on, will be able to learn how to move towards different places in space in an efficient (not random) way. Starting from random movements and capturing the sensory input data, it is able to learn concepts such as place and distance, which permits it to learn how to move towards a target efficiently.

Date: November 2010	Language: English	Number of pages: 67

Keywords: Machine learning, autonomous agents, cognitive architecture, slowness principle, sensorimotor abstractions, cerebral cortex.

Universidad Politècnica de Catalunya Facultat d'Informàtica de Barcelona	RESUMEN DEL PROYECTO FINAL DE CARRERA

Autora: María Isabel Cordero Marcos

Título: Learning sensorimotor abstractions

Título en español: Aprendizaje de abstracciones sensorimotoras

Programa: Doble titulación en Ingeniería Superior en Informática e Ingeniería Superior en Telecomunicaciones

Centro: Centro de Formació Interdisciplinària Superior (CFIS)

Supervisor : Jouko Lampinen	Instructor: Harri Valpola

Resumen:
Los agentes autónomos (máquinas o robots) cuya finalidad sea interactuar en ambientes reales desarrollando tareas cotidianas no pueden ser programados de forma determinista. Dada la gran diversidad de escenarios y, dentro del mismo escenario, multitud de variables, es imposible tener en cuenta cada situación con la que el agente autónomo pueda encontrarse. Estos agentes no poseen la misma facilidad de adaptación que tienen los humanos, capaces de interactuar con los cambios del entorno utilizando la información que reciben a través de los sentidos.

Los agentes autónomos necesitan poder adaptarse a situaciones de cambio constante. Este proyecto propone una solución basada en fundamentos bilógicos, que consiste en conseguir que los agentes autónomos sean capaces de aprender habilidades y de adquirir capacidades para desarrollar tareas posteriores de mayor complejidad.

Esta solución propuesta, fundada en aspectos biológicos, y conocida como arquitectura cognitiva, posee la misma estructura jerárquica que la corteza cerebral de los humanos. Gracias a esta arquitectura cognitiva, el agente autónomo es capaz de extraer información útil de los datos que recibe a través de los sensores. La información es codificada en forma de abstracciones (sensoriales y motoras) y constituyen rasgos invariables encontrados entre los datos. La arquitectura cognitiva utiliza el principio de lentitud para obtener las abstracciones. En principio, los algoritmos de aprendizaje no supervisado que utilizan el criterio de lentitud, tratan de encontrar rasgos que cambien lentamente y sean relevantes entre los datos de entrada. Esta información retenida puede resultar útil para autoevaluación.

Mediante la conexión entre las abstracciones sensoriales y motoras, el agente autónomo trata de aprender cómo manipular las abstracciones sensoriales. Esto le permite encontrar la relación existente entre las acciones motoras que realice y los cambios que es capaz de producir en el ambiente que le rodea.

En este trabajo se implementa un ejemplo que utiliza la arquitectura cognitiva. Un agente autónomo, que desconoce todo acerca del ambiente que le rodea, será capaz de aprender a moverse de manera no completamente aleatoria entre diferentes puntos en el espacio. Empezando con movimientos aleatorios y procesando la información sensorial que percibe, será capaz de aprender conceptos como lugar y distancia, lo que le permitirá aprender a moverse hacia un lugar concreto de manera eficiente.

Fecha: Noviembre 2010	Idioma: Inglés	Número de páginas: 67

Palabras clave: Aprendizaje de máquinas, agentes autónomos, arquitectura cognitiva, principio de lentitud, abstracciones sensorimotoras, corteza cerebral.

Acknowledgements

I am heartily thankful to my instructor Harri Valpola, who made this project possible. I am grateful for his explanations, guidance and support during the completion of this project. I have truly enjoyed and learnt a lot about the topics concerning this thematic.

I would also like to thank my family, both in Finland and in Spain, who have been always supporting and encouraging me.

María Isabel Cordero Marcos

Contents

Chapter 1

Introduction

In our society, machines play an important role. From laundry machines to vehicles and industrial robots, machines facilitate human life, providing higher-levels of comfort and happiness. Nevertheless, some tasks such as serving a cup of tea or walking from a place to another, which may seem fairly simple to a human, are extremely difficult, if not impossible, to carry out by a hard-coded machine. The reason lies in the uncertainty about the environment: the location of the teacup, unexpected obstacles in the walking path or abrupt changes in the terrain, to name but a few.

A newborn human baby knows nothing about the world and yet, with time, the infant will be able to interact with the environment. Through the interpretation of the data provided by different receptors, the baby, autonomously, will be able to learn new abilities and get to know the environment. It is well known that humans are able to learn, solve complex unknown problems and adapt themselves to changing environments.

A machine or a robot, from now on, an agent, with different sensors could be simile of a living being. At the beginning, as the baby, the agent knows nothing about the world. However, without proper tools for interpreting the signals it perceives, it will not be able to interact with the environment. Creating these kind of agents that autonomously acquire new skills, refine the ones previously acquired and continually adapt to changes in their environment is nowadays an essential problem in robotics.

In this work, a part of the *sensorimotor system* for the agent will be implemented following the working principles found in the brain, specifically in the *cerebral cortex*. As the word suggests, *sensorimotor* makes reference to both sensory (what is perceived by the receptors) and motor functions (muscle or joint movement). Thus, taking into account the structure found in the *cerebral cortex*, a hierarchy of *sensorimotor abstractions* will be formed, using *machine learning* algorithms to determine the abstractions. The abstractions make reference to the

invariant features found within the data, where the data could be referred to sensory or motor functions. Using the abstractions found, the agent will be able to acquire new motor skills, which will allow it to interact with the environment.

The goal of this work is to implement a part of the *sensorimotor hierarchy* found in the human cortex, so that an agent will be able to acquire a certain skill in an unknown environment without any supervision. Those autonomously learnt representations and skills prepare the agent for future learning tasks. The method is not intended to substitute any other learning technique, but rather to complement them. In that sense, one could apply other learning methods using as basis the abstractions found with the cognitive architecture.

The abstractions are found using *slowness* as a criterion. The slowness criterion is defined as extracting signals from the data whose temporal difference is small compared to the raw inputs. The slowly changing abstractions can serve the *critic* to evaluate how well a certain task is being performed.

This work is organized as follows. In the second chapter, a biological view of the sensorimotor algorithm will be presented, that is, how the brain processes the sensory stimuli and learns to behave appropriately as an answer to those stimuli. In the third chapter, different machine learning techniques which can be used as an imitation of the processes occurring in the brain, will be introduced. In the fourth chapter, the model to be used for learning sensorimotor abstractions will be presented, as well as reviewing some justifications and previous work related to these themes. Finally, an implemented example of the model will be described in chapter five.

Chapter 2

Biological background

When a living being comes to the world it knows nothing about it. A newborn baby has no previous experience about how to interact with the environment, yet with time, it will be able to learn many new things. Nevertheless, it receives information from the environment, coded as an large range of sensory stimuli. The sensory stimuli that a human can experience is very large: one may encounter visual receptors in the eyes, tactile, pressure and temperature receptors in the skin or smell receptors in the nose, to name but a few. This will provide the needed information for the brain in order to learn to interact adequately in the environment.

The main function of the nervous system is to be able to control the body. Through the processing and interpretation of massive amount of sensory information that the central nervous system receives from other systems in the body such as the skin, ears, eyes, etc., the central nervous system must determine the appropriate motor response. This response can be quite variate, for instance, picking a book from a shelf or answering to a certain question that somebody asked. The muscles to move are very different depending on the situation, context that the nervous system must find from the sensory data that receives. However, not all the signals will provide useful information. In fact, it seems that the human brain ignores more than 99 per cent of the sensory information [Guyt 05].

If analysed in general terms what the human nervous system does, one will realize that it is not much different what a computer does, that is, transforming some inputs (sensory stimuli) into a certain set of outputs (motor responses). Such is the basic function of any algorithm. In this context, it will be called sensorimotor algorithm, algorithms that transforms data received from different sensors into motor commands. The sensorimotor algorithm is then equivalent to the integrative function of the nervous system. A visual comparison is shown in Fig. 2.1.

As in the case of a human being, an agent may have many kinds of sensory receptors: distance, magnetic, electric,...; those that imitate the human ones such

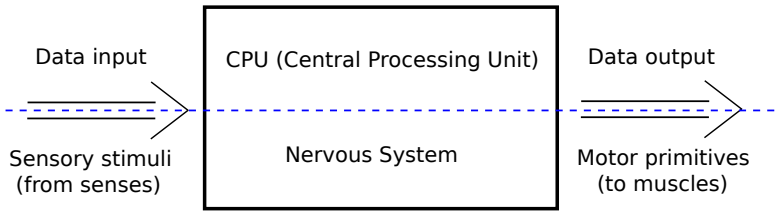


Figure 2.1: *The figure above shows the simile between the nervous system and a computer's CPU. The inputs to the nervous system are the sensory stimuli originated from the different sensors, while the outputs are the motor primitives that cause muscle movement.*

as the cameras which provide the visual sensory input, or microphones which permits hearing; or any other kind of gadget, imitating something from the living world or not, that it may occur to someone. All that large variety of receptors will provide the rich sensory input to the system. As it happens in the human brain, many of the sensory stimuli that the computer receives as input will be not useful. Thus, the sensorimotor algorithms must learn to extract the useful information hidden within all the sensory input data to provide the needed motor response.

What are the mechanisms that the central nervous system uses in order to learn these sensorimotor relationships? How could they be translated into machine terms? The following subsections analyse the information processes taking part in the brain, that is, taking into account the simile, the algorithm or program they implement. Firstly, the different sensory stimuli receptors and pathways previously to their processing will be introduced. After that, the cortex and the sensory data processes occurring within it will be studied and with that formation of a hierarchy of abstractions will be presented. Following, the formation of motor responses is analysed. Finally, the dorsal and ventral pathways, that will be represented in the cognitive architecture, will be described.

2.1 Sensorimotor data flow

The different sensory receptors in the body provide the sensory input to the nervous system. There are five different types of receptors: *mechanoreceptors* (compression and stretching receptors), *thermoreceptors* (temperature receptors), *nociceptors* (pain receptors), *electromagnetic receptors* and *chemoreceptors* (chemical receptors).

Each kind of receptor is sensitive to only one kind of stimulus. Following different pathways from the different sensors along the nerves, the signals arrive to the *central nervous system* where they will be treated. In the central system there

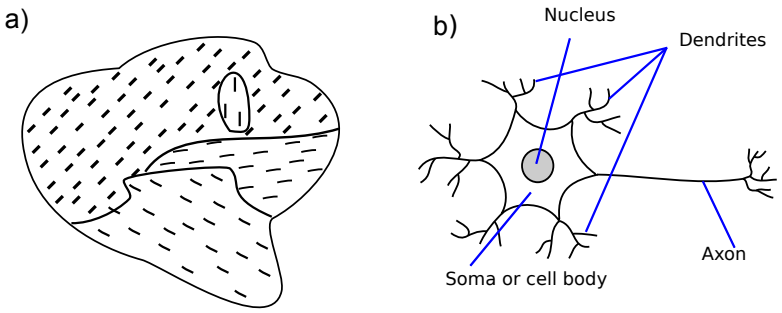


Figure 2.2: (a) *In order to understand the idea of a pool, imagine that the vectors' orientation in this figure represent the neurons configuration. Then, as it can be appreciated, in the figure there are four different pools, each one of them containing a different number and density of vectors (neurons).* (b) *Schematic representation of a neuron.*

are a huge amount of areas which presents an unique internal organization of *neurons* (Fig. 2.2(b)). Each of these areas is called *pool*. A pool can be composed from few neurons to millions of them. Some examples of pools are the *thalamus*, the *cerebral cortex* or the *cerebellum*. In Fig. 2.2(a) a schematic explanation of a pool can be seen.

The circuitry, patterns of neural organization, differs for each pool. Remembering that as the cables and transistors of a computer are connected in an specific way in order to implement different logic gates (and with that functions), the specific organization of neurons allow these areas to process the input signals in an unique way. As it happens in the computer processor, this fact allows the existence of multitude of functions in the whole nervous system. Thus, each pool, represents an operation (algorithm, function) in machine terms over the set of inputs that it receives.

Nevertheless, the aim of this work is not to focus on the circuitry of the nervous system and implement it by mechanical ways (transistors and cables), but to learn things by analysing and implement those algorithms that occur in the brain. However, by analysing the connectivity patterns, one can get closer to finding the algorithms implemented by the brain, so they should not be completely ignored. More extensive information about sensors, sensory data, neural connections and pathways can be found in Ref. [Guyt 05].

As mentioned previously, the information coming from the sensors goes to the *central nervous system*, ending in the cerebral cortex after visiting maybe other pools in their path, like for example, the *thalamus* as pain signals do. In the *cerebral cortex*, the data is processed in order to extract, and posteriorly use, the useful information carried among all the sensory input. In the *cerebral cortex* different processing areas corresponding to different types of sensation can be found. For example, the primary task of *visual cortex* is, as the word suggests, to

process the data received in the eyes; while the *auditory cortex* mainly processes auditory signals; or the *somatosensory cortex* which mostly takes care of tactile data.

Each of those *cortical areas* is divided in different layers, which are connected among them following a hierarchical structure. The hierarchy is defined by the different levels of processing of the data. This idea comes from the early 60's and was originally proposed by Hubel and Wiesel, who studied the structure of a cat's visual cortex [Hube 62, Hube 65]. Although the scheme they proposed has changed, the hierarchical vision of it still remains nowadays.

It was mentioned before that a cortical area mainly process data that corresponds to a certain kind of sensation, but not exclusively. For example, in the *visual cortex*, the majority of the areas are implicated in visual processing, but there are a few of them process non-visual data from other modalities of sensation. In the case of the visual areas, this non-visual data mainly consist of auditory and somatosensory stimuli or related to motor activity, such as eyes movement in this case. These areas are useful in finding correlations among different types of data.

As a result of the sensory data processing, different *representations* will be formed in the brain. The nervous system will interpret the *context* embedded in those sensory stimuli and together with the solutions obtained from the data processing, it will generate the adequate responses. That is, it will generate the appropriate signals that will carry out a certain action, in this sensorimotor context, a motor action, i.e., movement of muscles. For that, the neurons in the primary motor cortex go to the spinal cord. The neurons in the spinal cord are directly connected to the muscles and will cause them to contract or stretch, to move.

Those sensorimotor processes take part in different regions of the cortex. That is why it will be analysed in more detail in the following subsections.

2.2 Information processes in the cerebral cortex

The *cerebral cortex* ("cortex" being the Latin word for "cap") is a thin layer of folded tissue, that covers the surface of each cerebral hemisphere. The cortex itself is divided in six thinner layers of cells, called *cortical layers*. The sheets of neurons are the grey matter, while the wiring corresponds to the white matter (inputs and outputs). As the task of the cerebral cortex could be defined as to provide useful inputs for the rest of the brain areas: the cortex commands and the rest brain areas obey, always after learning. Nevertheless, only some mammals have cerebral cortex and among them, humans posses the largest and the most complex one. What is it function then? How is it structured? Which processes are computed in the cerebral cortex?

There are two major types of inputs in the cortex. One of them is *bottom-up input* that typically originates from the senses. They are the *primary inputs* and they end in *cortical layer IV*. It seems that the IV cortical layer is involved in feature expansion for bottom-up inputs, that is, sensory stimuli. More numerous are the other kind of inputs, the *feedback* (*lateral* and *top-down*) input connections which originate in higher-level cortical areas.

A notable characteristic of the cerebral cortex is it columnar architecture, which is formed during development, when neurons climb to the cortex following vertical strings. Looking at the architecture form by the neural connections, two conclusions can be extracted. First of all, it seems that the cortex, as a whole, works as a kind of hierarchy associative memory. Secondly, it appears that in the cortex there are increasing levels of abstraction processing, a hierarchy, defined by the outputs and inputs between the different cortical parts. In fact, it is the only place in the central system where a hierarchy of abstract representations can be found. There is also a hierarchy not only of sensory data, but also of motor actions. By using primitive motor actions, complex patterns of movements can be learnt and executed. More information about the hierarchy, and justifications such as connections between different areas can be found in Ref. [Fell 91].

As said in the previous section, sensory signals coming from the sensors end up eventually in the cerebral cortex for further processing. Also there are areas destined to control movement, that is, parts of the cortex send signals out to the muscles. Moreover, based on scientific observations of people with brain damage, and on animal studies, it seems that activity in the cerebral cortex produces:

- Meaningful experience of the world: the cortex can represent not only the state of the outside world, but also the uncertainty about that state, which is important for decision making.
- Simulation: it is possible to simulate and plan within the cerebral cortex. It plays an important role about the dynamics of the world, the rules of the world and what would happen if.
- Segmentation: the cerebral cortex is very important in segmenting objects. It is possible to think about relations between objects.
- Keep track of long term goals, the goal of the task that is being done
- Abstract thinking in general
- Language in general
- Selective attention
- Coordinate transformations
- And many others.

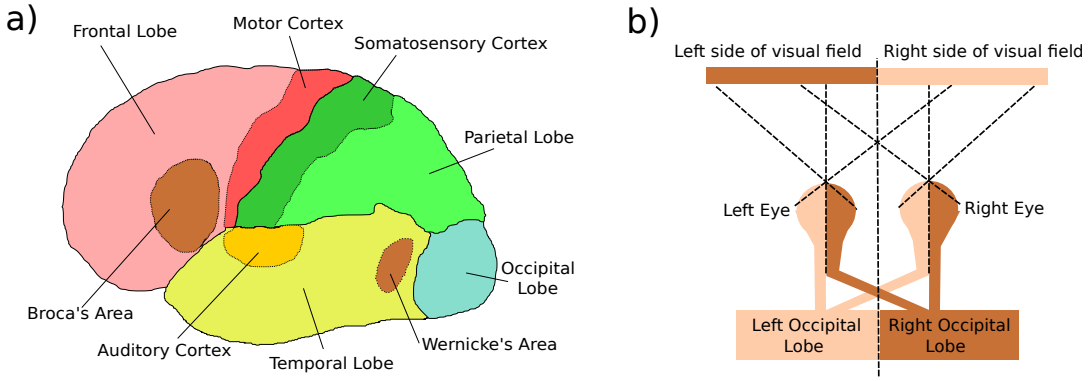


Figure 2.3: (a) *Lobes in the cerebral cortex and some of the main areas within them.* (b) *Schematic figure of how the visual field is processed in each of the occipital lobes.*

Even so, the cortex is not everything. If the cortex is completely removed on a mammal the animal can function in a surprisingly normal way.

The cerebral cortex is usually divided in several subcortical sections in each of the hemispheres. The different areas can be seen in Fig. 2.3(a). Their main functions are:

- The *frontal lobes* are located behind the forehead. This area of the brain is associated with higher-level thinking, such as problem solving, reasoning and some aspects of speech (*Broca's area*). It also contains the motor cortex, which controls voluntary movement. The forward part of the frontal lobe, *prefrontal cortex*, helps control the highest-levels of thinking, such as planning, reasoning and imagination; in conscious functions like empathy, self-perception and the ability to interact appropriately with others. This part of the brain is especially well-developed in humans.

- The *temporal lobes*, above the ears, are involved in hearing (auditory cortex), identifying objects, understanding language (*Wernicke's area*), and storing memories. They also play a role in emotions.

- The *parietal lobes* are located on the top of the head. They process senses such as touch, pain and temperature (all of them in the *somatosensory cortex*). They are also related to voluntary movement, spatial awareness, attention, language and some mathematical abilities.

- The *occipital lobes* at the back of the brain interpret visual information (visual cortex) such as movement, shape, color and light. An interesting fact is how the data is received and processed. It is well know that the left hemisphere is in charge of the right part of the body and vice-versa. In case of the occipital lobes, each one of them interprets the sensory visual data

from the opposite halves of each eye, that is, the left occipital lobe receives sensory visual signals from the left half of the retinas in both the right and left eyes which corresponds to the opposite (right) halves of the visual field (the images captured in the retina is rotated); that can be schematically appreciated in Fig. 2.3b. Idem for the right occipital lobe. The two lobes are connected, so the information is combined to produce a single image.

Another division of the cortex is the division in *gyrus*. According to the dictionary definition, a gyrus is a ridge, a convex fold or elevation in the surface of the brain. Each one of the previous lobes contain several of these gyrus.

Now that the different parts and functions have been presented, new questions appear: how does the cortex learn those tasks? How does it work? In the next sections the underlying processes taking place in the cortex will be introduced.

2.2.1 Bottom-up processing and learning: from the senses to first level of abstraction

This kind of processing and learning is related to the bottom-up inputs. As said before, the bottom-up inputs make reference to the inputs coming from the senses. These signals go through thalamus and end in in cortical layer IV. The bottom-up inputs combine and expand the primitive sensory stimuli into what it would be the first and most basic level of hierarchy of abstractions present in the cortex, the *low-level abstract features*. A feature is a representation that characterizes something, in this case it will be the sensory stimuli.

Even though there are several cortical areas and each one of them has its primary function, the cortex shows tremendous plasticity. That implies that the location of the areas does not determine what they are doing, but it is the structure of the inputs, the wiring, which determines their function. This means that if visual stimuli were feed to the skin, the *primary somatosensory area* would become a *visual cortex*. It can be said then that the cerebral cortical learning is highly relaying on the type of structure that the input has.

Even though, the variety of sensory input is quite large and there are many functions that the cerebral cortex implements, it seems that, surprisingly, the cortex is implementing only one general algorithm. There are different facts supporting this idea, such as evolutionary reasons, when compared the cerebral cortex among different mammals, and the development of the cerebral cortex, which point to a refinement of the algorithm in time and no a complete new algorithm. For example, the cortex of a newborn looks quite homogeneous, but with time it starts differentiating and for example in adulthood, layer IV in the motor cortex disappears. If some parts of the cortex is removed in early stages of life, in adulthood, that person will no seem to behave any different to any other with the whole cortex.

It seems that the cortex is a specialized organism for unsupervised learning [Doya 99]. Some of the reasons that support this notion and possible algorithms that could be used in the cognitive architecture are introduced below.

Competitive learning is one kind of *unsupervised learning* that seems to be implemented in the cortex due to reasons such as neuronal connections. It is a way of learning *abstractions*. An abstraction is a representation. Those are coded in the activations of the neurons.

If there were only one neuron, that neuron should learn the average activation or the direction of most energy (squared activation). The learning rule is simple: when the neuron is activated it will increase those synapses that were active. That is the way to learn an average activation pattern. This is called *Hebbian learning*. Hebbian learning is defined as changing those synapses which are active at the same time that the neuron is activated.

The problem appears in an environment with multiple neurons that use the same rule. This would cause that all of them would learn the same feature. However, adding *competition* among them, so that the neurons have a rule that if there are some highly active neurons (receiving inputs from which they get a high-level of activation), then those neurons will tend to inhibit all the other neurons. This is know as competition for the right to activate.

Combining both competition for the right to be active and Hebbian learning, *competitive learning* is obtained. With this competitive learning process, small initial differences at the beginning will be exaggerated.

The competitive learning is a very natural way of learning *sparse feature codes*, which seems to be the way neurons store information [Olsh 04]. Because of the competition, if there are many inputs that look quite similar, the coding patterns (activation pattern that the neurons present) will be different. Certain neurons become specialize in one type of input, the one that are best at representing, and they will inhibit the others neurons which are pushed away and moved towards representing the other inputs. The system is automatically driven to solutions where only a few neurons are representing the inputs, which is know as *sparse* (not dense) code. Sparseness may be a requisite in some algorithms. In Sec. 5.1.2, competition will be used for finding sparse abstractions.

The solutions obtained for representing a feature, will depend on the kind of competition. If the competition were very harsh and global, this would actually result in local coding, in which one neuron would be representing everything. If the competition were not that harsh and localized, it means that in different areas there would be active neurons, and overall the whole representation would be sparse. The activation of the different neurons will indicate unequivocally the actual input.

The competing mechanism described previously is not the only possible way of competition. In that case, it was done by choosing a winner, that is, one

active neuron which inhibits all the others. Another alternative way to implement competition is to do a more specific inhibition. This is something that is used in many *statistical unsupervised learning algorithms* and it requires the activations of each of the neurons to become statistically uncorrelated to each other.

Independently of how competition is implemented, the results are similar. The first case of competition, choosing a winner, corresponds in what is called *self-organizing map* (SOM, [Koho 95]). A SOM is a kind of unsupervised learning based on a *neural network* that finds a map, a representation of the inputs coded as activation levels of the neurons in the network. The other kind of competition would have as corresponding algorithm *independent component analysis* (ICA), which will be explained in more detail in Sec. 3.2.4. Basically ICA tries to find statistically independent components within the data. With those components found, it is possible to linearly reconstruct the data.

Similar results to those obtained with the ICA algorithm when processing natural scenes, are found in the area V1 (*primary visual cortex*), the entering point of nearly all visual information [Hate 98]. This does not show that the algorithm implemented in V1 is the ICA algorithm, but rather it shows that the cortex is implementing unsupervised learning when processing the data.

The first levels of the cortical hierarchy in the visual cortex, V1, respond to orientation. When going to upper levels in the cortical hierarchy, the activation patterns become more and more complex. For example, there is an area that responds to faces, to different features of faces. It is a fact that the cortex learns more complex sensory features and more complex motor categories. The question lays now in how to find those more complex, more abstract patterns.

In principle more complex abstractions can be implemented, until a certain degree, by grouping together different basic level features. There are different criteria for grouping abstractions. In the *bottom-up stage*, there is something similar to *subspace ICA* which could be used. The working principle of subspace ICA consist of grouping together features that are active in a particular temporal window. Another method in machine learning that only uses bottom-up information, is *slow feature analysis* (SFA). This method will be also described in a future section. Basically SFA looks within the data features that change as slowly as possible.

Nevertheless, it has been demonstrated that the bottom-up criteria, grouping all the way to the higher-level, is not enough for learning complex abstractions from real data. There is too much structure in the data, which makes it necessary to select which higher-level abstractions, obtained by grouping of elementary feature, are meaningful for the task. As said before, bottom-up is based on unsupervised learning and it is hard for this kind of method to decide which information to keep and which to thrown away, information that will be lost. In order to select, to decide, is necessary to have some kind of supervision.

In fact, when a human is born, the cortex first uses bottom-up learning, that is, just samples inputs and uses the statistical structure to figure out what is a useful vocabulary for representing these inputs, what are the primitive features by which these inputs are best represented, such as lines in area V1. After a critical period, top-down supervision starts affecting learning.

2.2.2 Top-down processing and learning: Towards more complex abstractions

Previously it was said that it is really hard, even impossible, to learn hierarchies just by using the data obtained from bottom-up connections. In order to find higher-level abstractions, it is necessary to select information. In the cortex, *feedback inputs* accomplish this purpose. There are plenty more feedback connections than bottom-up connections. Feedback connections can be either *top-down* or *lateral* (*horizontal*). The procedures described below could be used for building up more complex abstractions in the cognitive architecture.

Top-down connections have many roles. Among then, segmentation of objects, integration of information from the senses, interpretation of data inside a context, modelling transformations, predictive dynamics and selective attention can be found [Andr 01].

Lateral connections are learnt through experience and they encode the regularities of the world. Top-down and lateral connections select information [Gilb 07] to group together simple features creating, that way, more complex ones. This selection happens in both *learning time-scale*, when the cortex is deciding which kind of representations it has to learn; and in *behavioural time-scale*, when the cortex is trying to figure out what to represent, what kind of neurons should activate and what kind of information should be sent to the other areas.

There is a hypothesis that indicates that neurons encoding the same object (grouping segments together) synchronize their activations [Andr 01]. Lateral and top-down connections take an important role in synchronization: without them it will be impossible to do segmentation. The connections between neurons will not just be topographical, but also based on learnt associations. This can affect perceptual processes [Gold 07].

In machine learning, there are different strategies for grouping things together. One of them is *mutual information maximization*. Mutual information maximization is machine learning technique that corresponds to this grouping quite closely, and it is very similar to an old statistical technique called *canonical correlation analysis* (CCA). Nevertheless, those techniques are far easier than the ones supposedly used by the cortex.

Canonical correlation analysis is a simple linear analysis. It works with two or more data sets. Since it is a linear technique, it can find as many correlations

as the smallest dimensionality of the data sets. The goal of learning is to find projections (one from each dataset) such that their correlation is maximized. As an example one could think of the correlation between the lips movement (visual signal) and the sounds heard (auditory signal). In fact, different cortical areas influence and support each other [Skip 07].

An important detail to take into account when selecting information is the *context*. It may happen that two identical inputs patterns may have different meanings in different contexts. That is known as sensory *homonym*. There are also different input patterns may have identical meanings given a context, which is known as sensory *synonym*. As a part of information selection, the context is coded in the top-down and parallel connections.

Another kind of selection of information is *attention* and learning is strongly modulated by it. In adults, perceptual learning is very strongly dependent on attention. Attention is useful for learning since it already provides some candidate inputs for being grouped together. Even given the same bottom-up inputs while performing different tasks, the learnt features will be different. The *biased competition model of attention* (competition between cortical areas [Deco 05]) and the competitive learning introduced in the previous section, can explain how attention can help guide learning features on learning time scale. Lateral and top-down information and attention can influence what kind of representation a cortical area will learn. Some researches have been trying to apply this property in robotics, as in Ref. [Roa 09, Valp 08, Yli 07]. In the last one, curiosity and 'something interesting' can be interpreted as something that captivates the attention.

This biased competition model of attention is based on this kind of distributive selection: each area add local competition, and decides which bottom-up information should be passed to the next processing stages either laterally or top-down. The competition in one area affects the competition in other areas and so on. Selective attention emerges from the dynamics. This system tends to settle to one set of activations. Each of this set of activations is like an *attractor* in the system. An attractor is a group of neurons that support each other. The others would be *distractors*. The system is completely parallel, but at times the time to convert is linear with respect the distractors. This model was first proposed as a model for selective attention, but seems that the same mechanisms biasing local competition, serves other purposes such as interpreting the inputs in context.

The cerebral cortex can be seen as an *attractor network*. The *Hopfield network* is a kind of attractor network which was propose in machine learning communities as a model for the brain. This is a different approach from the one concerning this work. The Hopfield network has binary neurons connected with symmetric weights. The activation rule is simple: all the inputs activations are multiplied to the weights, sum together and then compared with a threshold. If the threshold is surpassed, then the neuron becomes active, other way not active. This network

converges to one of usually many stable states. Those states are called attractor. The network has an energy function and the attractors are minimums of the energy function.

Even if the cerebral cortex can be seen as an attractor network, there are different degree of connection between the cortical areas, which allows them to be in different states. Different cortical areas can be considered as making distributive decisions all by themselves, that is, the cortex seems to implement distributive selection of information, both in learning and behavioural time scale. Attention and learning are part of this distributive selection of information.

One important fact to take into account is that the motor representations used in the cognitive architecture are goal-oriented. Thus, in order to select information, one has to pay attention no only to the context, but to the final goal. With that purpose, prefrontal neurons bias the lower-levels. The prefrontal area implements a kind of working memory, and is there where the tasks being carried out are stored. Prefrontal areas are part of the net attractor as the rest of the cortex, but their time constant is very different. More information about the decision making in the frontal cortex as well as their relation to emotion and memory can be found in Ref. [Bech 00].

An attractor network, as the cortex is, can solve problems. Thinking, planning or consciousness are also few examples of dynamic phenomena occurring in the cortex. Humans can plan and solve situations they have never seen before. Moreover, cortex is a hierarchy. The higher-levels are watching the lower-levels and setting the connection strengths of the network, so the higher-level can set up the situation to be solved by a lower one. The high-level can control the constrains and the low-level can find the solution subject to those constrains. Once the problem has been solved the cortex commands other areas how to proceed according to the answer found.

2.2.3 Motor actions: Cortex cooperating with other areas

The main purpose of the brain is to carry out actions in order to achieve a certain goal. The *motor cortex* is the most involved part in the *cerebrum* controlling voluntary movement. For that purpose the motor cortex receives information from other cortical areas such as the parietal lobe, where there is information about the position of the body in space; the temporal lobe, where lays memories about past strategies; the prefrontal lobe, where the ultimate goal is keep in mind; and so on.

Not only cortical areas are involved, but also other parts of the brain. For example, the *basal ganglia* control the couplings between the neurons in the motor cortex and the prefrontal neurons. Basal ganglia are regathering some important signals coming from the cortex. There are some cortical areas that project to

thalamus and back to themselves so they form a *cortico-thalamo-cortical* positive feedback loop and basal ganglia are inhibiting all of these.

This way basal ganglia are able to allow or disallow these loops to activate and, with that, allow or not the motor action to be carried out. Basal ganglia learn through *reinforcement learning* (at least *dopamine*-based learning) and they learn in which circumstances the action should be or not done. The basal ganglia can modulate these decisions according to *reward*. Basal ganglia only needs to consider those motor programs that the cortex considers: many motor actions will not be considered by the cortex.

The motor representations of the cognitive architecture presented in this work are, as the actions in the brain, goal-oriented. In order to know if a goal was successfully achieved or not it is necessary some kind of evaluation. Receiving some kind of reward signal, (or, in its absence, an estimation of it) that indicates how good or bad an action was in a particular context, will serve decision-making in future steps (as the basal ganglia do) and planing processes. Planning processes will try to favour those actions that are more likely to achieve the goal of the motor representations while avoiding those that are more likely to fail.

In the brain, planning any movement is mainly done in the frontal lobe. The frontal lobe receives information about the body position from other areas in order to plan the sequence to carry out. When a decision to do something is made in the frontal cortex, this causes an increment in neural activity. Those neurons, at the same time, activate the ones in the motor cortex. In order to plan the best movement path and correct the movements, the motor cortex uses the information perceived by the sensors, such as the visual cortex. It also initiates communication with other brain areas, such as the cerebellum, which will help to activate and coordinate the muscles in the correct order. After that, the primary motor cortex sends the signals towards the rest of the body which will cause muscle movement. This closes a cycle of the sensorimotor algorithm, algorithm that has been explained through this chapter.

2.3 Ventral and dorsal pathways

In the cognitive architecture model presented in this work, there is a subdivision of the high-level sensory abstractions. This subdivision is related to the separate processing streams found in the cerebral cortex. These streams are know as *what* and *where pathways*. Both pathways present the hierarchical architecture typical of the cortex. The pathways were proposed by Ungerleider and Mishkin [Unge 82]. The function of each of the streams has been analysed by several researches, who investigated monkeys with lesions in either the temporal lobes or the parietal ones.

The *what pathway*, which originates in the occipital cortex and goes to the temporal cortex, is also called *ventral stream* or *ventral pathway*. The neurons in this pathway respond to visual features of objects such as color and texture. It seems that the ventral pathway is related to object recognition, that is, *what* an object is.

The *where pathway*, which originates in the occipital cortex and goes to the parietal cortex, is also called *dorsal stream* or *dorsal pathway*. The neurons in this pathway respond to spatial features of objects such the motion of the object, as, for example, neurons in area V5 are coding the orientation of the movement, in the same way that neurons in area V1 code line orientation. First studies indicated that the dorsal stream was related to spatial vision, that is *where* an object is.

Goodale and Milner [Good 92] suggested another interpretation for both dorsal and ventral streams. They proposed that the ventral stream is more closely related to perception of objects, whereas the dorsal stream helps guiding motor actions in real time to achieve a certain manipulation over the object. That is, the ventral pathway is a perception pathway, whereas the dorsal pathway could be described as an action pathway. With that, the distinction of the pathways proposed would be *what* and *how*, nomenclature that will be taken in this work. Thus, the approach in this work is the one proposed by Goodale and Milner, where the dorsal stream will serve the high-level motor abstractions to actively provoke changes in the high-level sensory abstractions.

Recent studies seem to support the interpretation proposed by Goodale and Milner. Moreover, it seems that the distinction of dorsal and ventral pathways is not only due to their function, but also due to the consciousness. It appears that the processing done in the dorsal pathway is unconscious (patients with lesions on the parietal lobes are often unaware of such), while the processing done on ventral pathways is conscious.

Before closing this section, it is important to remark that neurons in the ventral pathways need to code invariant representation of objects [Quir 05, Gema 06]. In order to be able to recognize an object, these neurons must be invariant to position, size, movement or any other kind of information that is not necessary or relevant for identifying the essence of an object. This principle of invariance is nowadays an approach used in computer vision for objects' recognition. Typically, something is invariant with respect to space or time or both at the same time. Thus, invariance is linked to slowness, to a slowly changing signal. The principle of slowness will be the approach used in this work in order to find invariant features, abstractions.

Chapter 3

Technological context: Machine learning

Machine learning consists of programming machines to improve their performance by means of taking intelligent decisions using previous experience. Among its many applications, one can find pattern recognition, data mining, bioinformatics, knowledge extraction, compression, and many others. Machine learning applications can be found in signal processing, medical diagnosis, signal processing, banking or robotics, to name but a few.

There are three main types of machine learning algorithms: *supervised learning*, *unsupervised learning* and *reinforcement learning*. Supervised learning requires the presence of a *teacher*, that is, an entity that will guide the learning process by giving, for each problem and moment, the correct solution. At first glance, it would appear the easiest and fastest method for learning, but this is far from the truth. There are certain problems with complex solutions, in which supervised methods prove to be the slowest and least efficient. In general, it is not always possible to teach how to do something, or think how could it be possible to teach something by describing it to an agent or an infant that cannot understand what one is saying.

In comparison, the unsupervised learning algorithms do not require a *teacher*. Instead, they try to find hidden patterns or any kind of structure from the data. This will turn out to be useful in order to capture abstractions.

The last of the three, reinforcement learning, has been proved a useful way of learning in novel environments, a thing that supervised learning does not allow. It does not require the presence of a *teacher*, but *rewarding signals*. The *rewarding signals* are usually related to emotions such as the level of pleasure (good reward) or pain (bad reward) or happiness or hunger or any other emotion. In a baby, these signals are innate and, in an agent, they can be more or less easily pre-programmed. Of course, more complex rewarding signals could be

obtained in later stages of the learning process. The main problem is that usually reinforcement learning is very slow.

Some machine learning algorithms are introduced below. The ones presented are those that will be used in the *cognitive architecture*, that is, the model implemented in chapter 4.

3.1 Reinforcement Learning

Reinforcement learning aims to maximize the performance (total reward) of a sequence of actions, thus, giving importance to the actions as a total: immediate reward is not that important. An illustrative example, where one can find this kind of scenario, is a go or a chess match where, at times, sacrificing some pieces (action that could appear to be bad in that moment) can turn out to be the best strategy.

In reinforcement learning there is no teacher that guides the learning process by giving the correct answers, i.e. an action that should have been taken, but rather a critic, or evaluation function, that measures how good a taken action is according to previous experience. Thanks to this feature, it is possible to face and find new solutions in novel situations, thing that is not possible in supervised learning methods. Nevertheless, the information given by the critic is often delayed, leading to the problem of associating the correct reward to the action, following as a consequence, not being able to find the optimal solution in some cases.

The context of the reinforcement learning technique is defined by the environment and the agent (see Fig. 3.1). In the case of a game of chess, the environment

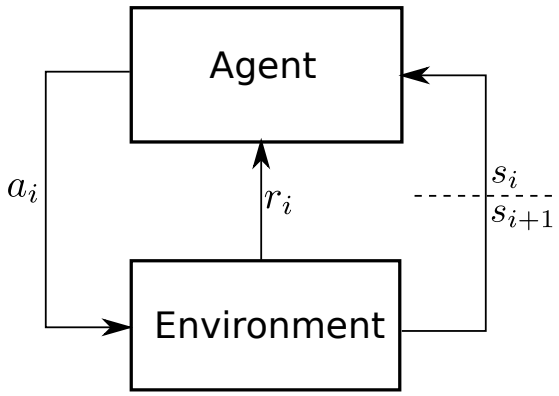


Figure 3.1: *Schema of the interaction of the reinforcement learning elements. The agent decides to carry out an action in the present environment, which can cause a change of state in the environment. The agent will receive a certain reward that will evaluate how good or bad was the action taken.*

would be the board. The environment can be in different states. In chess the state could be defined by the position of the different pieces in the board. In each state the agent, decision maker (player, intelligent machine,...), selects one among the valid actions. The set of possible actions, in the chess match, is different valid movements that could be made with the pieces.

Thus, in every step i the environment is in a certain state, $s_i \in S$, in which different actions, $A(s_i)$, can be executed. The agent chooses the action to execute, $a_i \in A(s_i)$, which may cause a change of state in the environment, $s_{i+1} \in S$. $A(s_i)$ represents the set of valid actions that can be taken when the state is s_i. After every action, or after a set of actions, the agent receives at step i a reward, r_i, from the environment. By optimizing the cumulative reward, the agent tries to find the best policy to solve the problem.

3.1.1 Elements of the reinforcement learning problem

The main elements that define the system are the policy, the reward function, the value function and optionally the model of the environment.

In reinforcement learning terms, the policy, π, is the sequence of actions to follow in order to arrive to the goal. Since each action taken in a particular state, s_i, leads to another state, s_{i+1}, the policy can be seen as a function between actions and states, $\pi : S \rightarrow A$, where S is the set of all possible states the environment can be and A represents the conjunct of all valid actions in all the states, that is, $A = \bigcup A(s_i) \forall s_i \in S$. There can be many mappings from states to actions, that is, policies, which will be labelled as π_k. Thus, for any state s_i in the environment, the policy π_k indicates the action $a_i \in A(s_i)$ to take: $\pi_k(s_i) = a_i$. The optimal policy, π^*, is the sequence of actions that maximizes the total (cumulative) reward.

As previously said, the aim of the reinforcement learning problem is to maximize the total reward, that is, to find the optimal policy. The reward comes from the environment and it measures how good or bad the action taken, a_i, was. An important feature of the reward, and one of the main characteristics of reinforcement learning, is that the reward is delayed. The agent does not usually obtain the reward immediately and it may not always be one reward per action taken. For instance in chess, the reward would be the result of the match. Because of this delay, the agent must observe the environment all the time in order to predict the consequences of the actions and use its past experience to improve over time.

Reward, as described above, indicates how good or bad a state-action pair is, but it does not give any idea of how good that pair is in the long run, whether the complete path will serve to maximize the total reward. That is why a value function, V, is introduced. The value function of a state is evaluated according to a certain policy π_k such that $V_k^{\pi}(s_i)$, indicates the expected total reward that

will be obtained starting from s_i and following policy π_k. Mathematically, the value function looks like:

$$V^{\pi_k}(s_i) = E\{r_{i+1} + r_{i+2} + r_{i+3} + r_{i+4} + ... + r_{i+T}\} = E\left\{\sum_{l=1}^{T} r_{i+l}\right\}, \qquad (3.1)$$

where r_i is the reward obtained in a state s_i. The reward obtained when executing the same action going from state s' to s'' may be different each time, that is why the expected value is calculated. There are cases in which there is not a terminal state. Then it is necessary to introduce a discount factor, γ, leaving the previous equation as:

$$V^{\pi_k}(s_i) = E\{r_{i+1} + \gamma r_{i+2} + \gamma^2 r_{i+3} + \gamma^3 r_{i+4} + ...\} = E\left\{\sum_{l=1}^{\infty} \gamma^{l-1} r_{i+l}\right\}, \qquad (3.2)$$

where the values of γ belong to the interval $[0, 1)$. As γ tends to 1, the more the future states affect the expected value of the policy in the current state. Values are in this sense predictions of the cumulative rewards, an elaborated good from them, whereas rewards are the raw material obtained from the environment.

According to the definition of the value function, the optimal policy, π^*, can be defined as:

$$V^{\pi^*}(s_i) = \max_{\pi_k} V^{\pi_k}(s_i), \qquad \forall s_i, \qquad (3.3)$$

and this optimal value function is unique.

The last element of the reinforcement learning problem is the model of the environment, which is not always available. Mathematically, not only it describes the function in which given the current state the environment, s_i, and the action taken by the actor, a_i, computes as outputs the reward to receive, r_{i+1}, and the next state, s_{i+1}, $f : S \times A \rightarrow S \times R$, but it includes all the probability distributions of the environment parameters, that is, $p(r_{i+1}|s_i, a_i)$ and $P(s_{i+1}|s_i, a_i)$. The environment can be used for planning, since the actor could predict the consequences of the actions, both in short and long term, before taking any actual action. In many cases the environment is completely unknown; in these situations the actor should follow a trial-and-error method.

3.1.2 Exploration-Explotaition dilemma

In reinforcement learning the objective of the learning agent is to take the action at each step that will maximize the cumulative (total) reward. As it has been pointed out above, the action that maximizes the local reward may not be the best action to take in terms of the total reward. The agent needs to select those

actions that maximizes the cumulative reward (exploit), but in order to be able to select those, it also needs to discover them (explore). This is what is know as the exploration-exploitation dilemma.

In order to find better actions, the agent must explore all the possible actions at all steps. By exploration it is understood that the agent will choose a valid action randomly in a certain state and observe the result. This allows to check the different possibilities and analyse which action could belong to the optimal policy.

Exploration is needed in order to choose better actions in a future time. This is the second main characteristic of reinforcement learning learning: trial and error. In order to determine if an action is good or bad, the agent selects it and compares the reward obtained to rewards of the other actions. A common used exploration strategy consists of the $\epsilon - greedy$ algorithm, in which the agent selects a random action with ϵ probability, where all actions have the same probability of being selected, and the best action with $1 - \epsilon$ probability.

Once the exploration has been used to determine the best actions for each state, the exploitation feature can be used. By best action it is understood the action, taken in a certain state, that will maximize the cumulative reward, that is, the action that belongs to the optimal policy or seems to belong to it in at the moment. The exploitation mode in reinforcement learning consists of selecting always the best action given the current state.

Using exploration first and exploitation second will be enough for deterministic environments, which do not change with time. In other kind of situations one may encounter that an action that was the best in a past time, will not be the best at the present or in a future time. Thus exploration is needed continuously, but exploration alone will not allow to get good results in terms of reward, so also exploitation is needed.

Some environments present such large state-spaces that the exploration of the full system is practically impossible, even if there is a model of the environment available. In these cases, the more exploration done (the more freedom the agent has), the better the results are, but at the cost of slowing down the learning process. In order to speed it up, it is very useful to have a good exploring strategy.

3.1.3 Implementation of reinforcement learning algorithms

The three main methods for implementing reinforcement learning algorithms are dynamic programming, Monte Carlo methods and temporal difference methods. There are also more efficient algorithms that are a mixture of the three basic ones. The aim is to implement an evaluation function (critic) that will provide with the best agent, that is, the agent that maximizes the cumulative reward.

The first method, dynamic programming, requires a model of the environment, which, as mentioned above, is not usually available. Some well known algorithms of this kind are policy iteration and value iteration. For more details, check Ref. [Sutt 98, Alpa 04].

The second method does not need a model of the environment. It works by averaging values of samples of episodes. An episode is understood as the sequence from an initial state s_i to one of the terminal states s_{T_k}. Thus, this method takes as feedback for changing the values estimates, the total reward. The main problem with this method is that the length of the episode, i.e. the number of steps, can be excessively large.

The third method, temporal difference method, differs from the Monte Carlo method such that instead of being 'episode-wise' it re-estimates the parameters after every action, that is, it is 'step-wise'. They are called temporal difference methods because the main parameter for re-estimating the new values is mainly based on the estimation error between the current state estimate and the next state estimate. Since the most interesting and realistic part of reinforcement learning is that in which the model is unknown (novel situation) and due to the Monte Carlo methods disadvantage explained above, only some of the temporal difference methods will be introduced in the below.

As an intermediate hybrid between Monte Carlo and temporal difference methods, there is a variant that takes into account the last k states in order to update the new estimates of the parameters, that is 'k-step-wise'. This variant uses *elegibility traces*, which are records of the previously visited states. In general terms, in each step they are updated by multiplying the previous value by a discount factor λ, which belongs to the interval $[0, 1]$. As λ tends to 1, the future states affect the expected value of the parameters more. In conclusion, $\lambda = 1$ is equivalent to Monte Carlo and $\lambda = 0$ is equivalent to temporal difference methods.

Temporal difference (TD) algorithms

Before introducing those algorithms, it should be mention that in some applications instead of using the value function for guiding the learning process, state-action pairs are used as it is equivalent. As explained above, the value function, $V^{\pi_k}(s_i)$, describes the estimate of the cumulative reward expected to achieve following the policy π_k starting in the state s_i. In this context, it describes how good or bad is to be in or visit a certain state. Meanwhile, the state-action pairs, $Q^{\pi_k}(s_i, a_i)$, describe the estimate of the cumulative reward expected to achieve following the policy π_k starting in the state s_i and taking action a_i. In other words, they indicate how good or bad is to take a certain action, a_i, when the current state is s_i.

Working with the optimal policy and according to this definition of the state-action, the value function of state s_i, $V^{\pi^*}(s_i)$, is equivalent to the value of the

state-action pair in state s_i when taking the best possible action a_i^*. By definition, the best possible action is the action that belongs to the optimal policy. Thus,

$$V^{\pi^*}(s_i) = Q^{\pi^*}(s_i, a_i^*) = \max_{a_{i,j}} Q^{\pi^*}(s_i, a_{i,j}) \tag{3.4}$$

Moreover, given the definition of state-action pair, the optimal policy could be defined as:

$$\pi^*(s_i) = a_i^* \tag{3.5}$$

with a_i^* being the action that $\max_{a_i} Q^{\pi^*}(s_i, a_i)$

In brief, TD learning algorithms try to find a stable estimate of the future cumulative reward.

SARSA: It is a kind of TD on-policy algorithm. It is based on the estimation and update of the $Q^{\pi_k}(s_i, a_i)$ map in order to find the best actions through experience. As an on-policy method, the convergence depends on the policy followed and the policy continuously adapts with Q^{π_k}. The algorithm is described in Alg. 1.

Algorithm 1 SARSA algorithm

Arbitrarily initialize $Q(s_i, a_i)$, $\forall i,\ \forall a_i \in A(s_i)$
for all $s \in E$ **do**
  $s \leftarrow s_i$ Initialize state $s \leftarrow s_i$
  Choose a_i using policy derived from Q, such as ϵ-greedy
  repeat
    $a \leftarrow a_i$ Take action
    $r \leftarrow r_{i+1}$ Observe reward
    $s' \leftarrow s_{i+1}$ Observe next state
    $a' \leftarrow a_{i+1}$ Choose $a_i + 1$ using policy derived from Q, such as ϵ-greedy
    Update the $Q(s_i, a_i)$ **value:**
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r_{i+1} + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$ Update the state for the next iteration
    $a \leftarrow a'$ Update the action for the next iteration
  until s is terminal
end for

Q-learning: It is a kind of TD off-policy algorithm. It is based on the estimation and update of the $Q^{\pi_k}(s_i, a_i)$ map in order to find the best actions through experience. As an off-policy method, the convergence of Q^{π_k} to the optimal values, Q^{π^*}, is independent of the policy being followed. The policy affects the order of the action pairs visited and updated. The algorithm is described in Alg. 2.

Algorithm 2 Q-learning algorithm

Arbitrarily initialize $Q(s_i, a_i), \quad \forall i, \ \forall a_i \in A(s_i)$
for all $s \in E$ **do**
  $s \leftarrow s_i$ Initialize state $s \leftarrow s_i$
  repeat
    Choose a_i using policy derived from Q, such as ϵ-greedy
    $a \leftarrow a_i$ Take action
    $r \leftarrow r_{i+1}$ Observe reward
    $s' \leftarrow s_{i+1}$ Observe next state
    Update the $Q(s_i, a_i)$ **value:**
      $$Q(s,a) \leftarrow Q(s,a) + \alpha[r_{i+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
    $s \leftarrow s'$ Update the state for the next iteration
  until s is terminal
end for

Actor-critic methods: In this on-policy method, the actor corresponds to the policy, that is the mapping between states and action, whereas the critic is the value function. This pair represents the essence of reinforcement learning such as is has been discussed above: it is the actor (agent) who chooses the actions to take based on the feedback obtained from the value function (critic). After each action, the critic evaluates how good the action taken was. This evaluative feedback serves for changing both the policy and value function (actor and critic) and guiding them towards the optimal solution. In order to update those values it also uses the TD-error.

3.2 Unsupervised Learning

The aim of unsupervised learning is to find the hidden patterns in the input data. As opposite to supervised learning, there is not an external agent that provides information about the correctness of the solution, that is, it is not guided.

Clustering is one kind of unsupervised learning method. It tries to group different instances of the data that share similar characteristics, that is, classify them into clusters. This is useful in compression algorithms, reducing the total amount of data.

Other useful compression algorithms that belong to unsupervised learning are those used for dimension reduction such as principal component analysis (PCA). The algorithm will be described in detail below. The aim of the algorithm is to find a mapping from the original data space of dimensionality o to a new space of dimensionality n, such that $n < o$, with the minimum loss of information.

More examples of unsupervised learning algorithms are slow feature analysis

(SFA) denoising source separation (DSS) and independent component analysis (ICA), which will be introduced next.

3.2.1 Principal Component Analysis (PCA)

As explained above, PCA is a kind of dimensionality reduction algorithm in the unsupervised learning context. In other fields it is known with other names, for example, Karhunen–Loève transform (KLT) or the Hotelling transform, both terms commonly used in signal processing.

Mathematically, PCA is based on an orthogonal linear transformation, transforming the data into a new coordinate system such that the components are related to the variance of the data. The first or principal component will be the one with the maximal variability (statistically speaking variance, power in terms of signal processing), the second will be associated with the second most variability and so on. Again, depending on the application, the components of interest will be the ones with the least variance, as major interest in slow feature analysis (SFA) algorithms; or the ones with most variance, as for selecting the most important components of a signal. The ones with the most variance are the ones that contain the most power of the whole signal. Selecting those and discarding the others one could compress the data keeping the most information the possible and later recomposing it with the least square error.

The basis functions of this transformation are the eigenvectors of the data. In order to prove this, the description of PCA in [Alpa 04] is followed. The variance of a signal is defined as the expected value of the squared value of the signal minus its mean, that is:

$$\mathbf{\Sigma_x}^2 = E\{(\mathbf{x} - \mu_x \mathbf{1})^2\}. \tag{3.6}$$

The principal component, $\mathbf{w_1}$, is that in which the input data will have the highest variance once it is projected to the new space. Let $\mathbf{x}$ be the input data and $\mathbf{z}$ the input data projected over the principal component, i.e., $\mathbf{z} = \mathbf{w_1}^T\mathbf{x}$. The principal components are orthonormal, thus $\|w_1\| = 1$. The variance of the projected signal can be calculated as

$$\mathbf{\Sigma_z}^2 = E\{(\mathbf{z} - \mu_z \mathbf{1})^2\} = E\{(\mathbf{w_1}^T\mathbf{x} - \mu_z \mathbf{1})^2\}, \tag{3.7}$$

where $\mu_z = E\{z\} = E\{w_1^T x\} = w_1^T E\{x\} = w_1^T \mu_x$, since w_1 is a deterministic vector. Substituting this result into Eq. (3.7) gives

$$\begin{aligned}
\mathbf{\Sigma_z}^2 &= E\{(\mathbf{w_1}^T\mathbf{x} - \mathbf{w_1}^T\mu_x)^2\} \\
&= E\{(\mathbf{w_1}^T\mathbf{x} - \mathbf{w_1}^T\mu_x)(\mathbf{w_1}^T\mathbf{x} - \mathbf{w_1}^T\mu_x)^T\} \\
&= E\{\mathbf{w_1}^T(\mathbf{x} - \mu_x\mathbf{1})(\mathbf{x} - \mu_x\mathbf{1})^T\mathbf{w_1}\} \\
&= \mathbf{w_1}^T E\{(\mathbf{x} - \mu_x\mathbf{1})(\mathbf{x} - \mu_x\mathbf{1})^T\}\mathbf{w_1} \\
&= \mathbf{w_1}^T E\{(\mathbf{x} - \mu_x\mathbf{1})^2\}\mathbf{w_1} = \mathbf{w_1}^T\mathbf{\Sigma_x}\mathbf{w_1}.
\end{aligned} \tag{3.8}$$

In order to find the $\mathbf{w_1}$ that maximizes the variance and fulfills the orthonormal condition, Lagrange multipliers are used. The general equation of a Lagrange multiplier is

$$\Lambda(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda[g(\mathbf{x}) - c], \tag{3.9}$$

where $f(\mathbf{x})$ is the function to be maximized or minimized and $g(\mathbf{x}) = c$ is the constrain the function $f(\mathbf{x})$ must obey. Setting $f(\mathbf{x})$ equal to Eq. (3.8) and the constrain of orthonormal vectors to $g(\mathbf{x}) = c$ and substituting all into Eq. (3.9) gives

$$\Lambda(\mathbf{x}, \lambda) = \mathbf{w_1}^T \mathbf{\Sigma_x} \mathbf{w_1} - \lambda[\mathbf{w_1}^T \mathbf{w_1} - 1], \tag{3.10}$$

taking the derivative with respect $\mathbf{w_1}$ for obtaining the maximum,

$$\begin{aligned}
\mathbf{0} &= 2\mathbf{\Sigma_x} \mathbf{w_1} - 2\lambda \mathbf{w_1} \\
&\Rightarrow \mathbf{\Sigma_x} \mathbf{w_1} = \lambda \mathbf{w_1} \\
&\Rightarrow \mathbf{w_1}^T \mathbf{\Sigma_x} \mathbf{w_1} = \mathbf{w_1}^T \lambda \mathbf{w_1} \\
&\Rightarrow \mathbf{w_1}^T \mathbf{\Sigma_x} \mathbf{w_1} = \lambda \mathbf{w_1}^T \mathbf{w_1} \Rightarrow \{\mathbf{w_1}^T \mathbf{w_1} = 1\} \\
&\Rightarrow \mathbf{w_1}^T \mathbf{\Sigma_x} \mathbf{w_1} = \lambda \Rightarrow \mathbf{\Sigma_x} \mathbf{w_1} = \lambda \mathbf{w_1}.
\end{aligned} \tag{3.11}$$

By observing the last expression, it can be deduced that $\mathbf{w_1}$ is the eigenvector of the matrix $\mathbf{\Sigma_x}$ and the constant λ is the eigenvalue associated to that vector. In order to make the variance the largest the possible, the eigenvector $\mathbf{w_1}$ should be related to the largest eigenvalue λ.

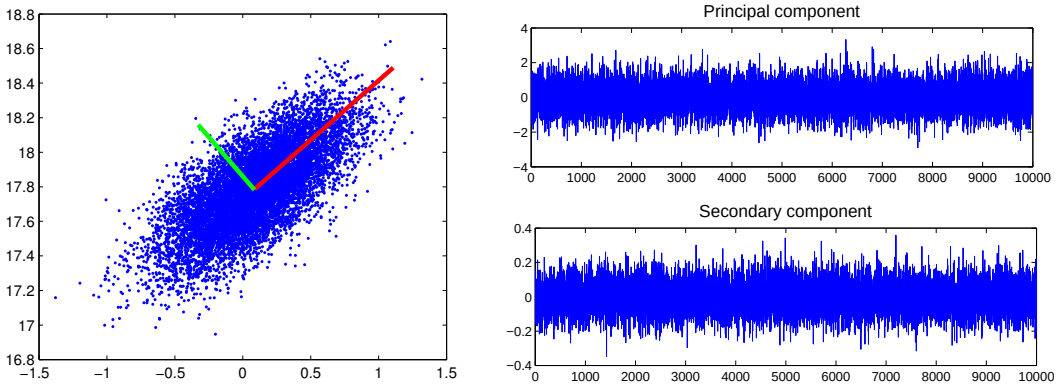
The rest of the principal components can be calculated in the same way, using multiple constrains instead of only one, where the other constrains make reference to the orthogonality of the vectors. For example, when calculating the third principal component, the Lagrange multiplier would look like

$$\Lambda(\mathbf{x}, \lambda) = \mathbf{w_1}^T \mathbf{\Sigma_x} \mathbf{w_1} - \lambda[\mathbf{w_3}^T \mathbf{w_3} - 1] - \alpha[\mathbf{w_3}^T \mathbf{w_1} - 0] - \beta[\mathbf{w_3}^T \mathbf{w_2} - 0]. \tag{3.12}$$

Taking the derivative with respect to $\mathbf{w_3}$

$$\begin{aligned}
\mathbf{0} &= 2\mathbf{\Sigma_x} \mathbf{w_3} - 2\lambda \mathbf{w_3} - \alpha \mathbf{w_1} - \beta \mathbf{w_2} \\
&\Rightarrow 2\mathbf{\Sigma_x} \mathbf{w_3} = 2\lambda \mathbf{w_3} + \alpha \mathbf{w_1} + \beta \mathbf{w_2} \\
&\Rightarrow 2\mathbf{w_3}^T \mathbf{\Sigma_x} \mathbf{w_3} = 2\mathbf{w_3}^T \lambda \mathbf{w_3} + \mathbf{w_3}^T \alpha \mathbf{w_1} + \mathbf{w_3}^T \beta \mathbf{w_2} \\
&\Rightarrow 2w_3^T \Sigma_x w_3 = 2\lambda w_3^T w_3 + \alpha w_1 w_3 + \beta w_2 w_3 \\
&\Rightarrow \{\mathbf{w_3}^T \mathbf{w_3} = 1, \mathbf{w_3}^T \mathbf{w_1} = 0, \mathbf{w_3}^T \mathbf{w_1} = 0\} \\
&\Rightarrow \mathbf{w_3}^T \mathbf{\Sigma_x} \mathbf{w_3} = \lambda \Rightarrow \mathbf{\Sigma_x} \mathbf{w_3} = \lambda \mathbf{w_3},
\end{aligned} \tag{3.13}$$

which gives again an eigenvector as $\mathbf{w_3}$, in this case it should be associated to the third largest eigenvalue, since the two largest ones are associated to the first and the second principal components.



(a) (b)

Figure 3.2: (a) *The raw two dimensional random data sample. As it can be seen, the direction of more energy corresponds to the red line (principal component), while the direction of less energy corresponds to the secondary component.* (b) *Principal and secondary component extracted from the original data. As it can be appreciated, the principal component represents about 90% of the total energy.*

Proceeding in the same way with all the components, the solution obtained is that the principal components are the eigenvectors of the covariance matrix sorted in descending order.

In an example with 2 dimensional data, as shown in Fig. 3.2, the principal component would be the one in the direction of the red line (maximum variance) while the other one would be the orthogonal to the first one, represented with green line.

In order to apply the dimensionality reduction, the first n principal components are selected. Those components contain most of the variance (power) of the original signal since the smaller the order or the vector the bigger the variance (eigenvalue associated). Depending on the precision desired and the total allowed error, the number of components to select will change.

As mentioned before, PCA is used for dimension reduction, which is useful in fields like telecommunications in order to reduce the complexity of the signal before transmitting it, that is, data compression. Many other applications in other scopes can be found, such as informatics [Labi 04] or even weather applications [Iyen 91, Gold 04].

3.2.2 Slow Feature Analysis (SFA)

Slow feature analysis (SFA) algorithms are another kind of unsupervised learning algorithms, especially good at extraction of invariants in the data. It is a method

for finding, in the data mixture of signals, those features (signals) that change slowly in time and contain useful information, that is, SFA is an implementation of the *slowness principle* (Sec. 4.3). Another way of obtaining slowly varying signals would be using a low pass filter, but they are often uninformative.

In order to find the slow and informative signals, the algorithm consists of applying PCA over the time derivative of the data. As said before, PCA extracts the components in the data in order of variance amount (energy), being the one with more variance the principal component, the second with more variance the second component and so on. The variance of a signal indicates how much the values of the signal vary from the mean. On the other hand, the derivative of a function indicates how much the value of the outputs (image) changes with respect to a change in the input (domain), in this case, being time the domain in this case. Thus, the faster the signal varies, the more variance the derivative will have, since small changes in the input provoke large changes in the output. The first slowest feature would correspond to the last component in PCA.

The objective is to find a slow varying signal in the data mixture, avoiding the obvious answer, that is, the constant function. There are I time varying input signals, $\mathbf{X}(t)$, where $\mathbf{x}_i(t)$ denotes the input signal i. The output signals will be identified as $\mathbf{Y}(t)$, and they will be ordered according to their slowness. Moreover, the output signals are subject to the following constraints

$$\begin{aligned} \text{Zero mean} \quad &: \quad E_t\{\mathbf{y}_j\} = 0 \\ \text{Unit variance} \quad &: \quad E_t\{\mathbf{y}_j^2\} = 1 \\ \text{Decorrelation} \quad &: \quad E_t\{\mathbf{y}_i \mathbf{y}_j\} = 0 \quad \forall i < j, \end{aligned} \tag{3.14}$$

where E_t is the expected value with respect to time. The first two equations avoid the constant solution, while the third one assures that the signals contain (mutually) different information. It also makes the solutions to appear in the correct order.

The problem and algorithm explanations in more detail and some applications examples can be found in [Wisk 02]. A more mathematical and complete approach is available in [Spre 08]. In order to obtain the slowly varying signals, the SFA algorithm follows the next steps.

1. Normalization of the input signal, $\mathbf{X}(t)$:

$$\tilde{\mathbf{x}}_i(t) = \frac{\mathbf{x}_i(t) - E_t\{\mathbf{x}_i\}}{\sqrt{E_t\{(\mathbf{x}_i - E_t\{\mathbf{x}_i\})^2\}}}. \tag{3.15}$$

2. Non-linear expansion of the normalized input signal, $\tilde{\mathbf{X}}(t)$. This step consists of applying different non-linear functions with the purpose of generating an expanded signal from the input data. This adds a higher dimensionality to the data. A way of applying this expansion is, for example, adding

second degree terms to the original signal, by taking into account all the possible products between each pair of signals including the square of the signals, that is

$$\mathbf{Z}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), ..., \mathbf{x}_I(t), \mathbf{x}_1(t)\mathbf{x}_1(t), \mathbf{x}_1(t)\mathbf{x}_2(t), ..., \mathbf{x}_I(t)\mathbf{x}_I(t)]^T. \quad (3.16)$$

3. Sphering of the expanded signal, $\mathbf{Z}(t)$, or whitening, that is, having zero mean and identity matrix as a covariance. For that, first the mean is taken away, centring the signal. For whitening the data, one could use the eigenvalue decomposition (EVD) of the covariance matrix of the centred data. Mathematically, any covariance matrix $\mathbf{C}$ of a real signal $\mathbf{S}$ can be decomposed into their eigenvalues and eigenvectors according to the next equation:

$$\mathbf{C} = E\{\mathbf{S}\mathbf{S}^T\} = \mathbf{D}\mathbf{\Lambda}\mathbf{D}, \quad (3.17)$$

where $\mathbf{D}$ is an orthogonal matrix with the eigenvectors in its columns, and $\mathbf{\Lambda}$ is the diagonal matrix containing the corresponding eigenvalues.

The data whitened, $\tilde{\mathbf{S}}$, now can be expressed as:

$$\tilde{\mathbf{S}} = \mathbf{D}\mathbf{\Lambda}^{-1/2}\mathbf{D}^T\mathbf{S}. \quad (3.18)$$

Since $\mathbf{\Lambda}$ matrix is diagonal, the inverse of the square root operation is element-wise. By whitening it is understood the removal of the correlations in the data, that is, making the components orthogonal, which makes it easier to treat in posterior steps. As it can be seen in the equations, it just implies a linear transformation, a change of base.

4. Taking the time derivative of the whitened expanded signal, $\tilde{\mathbf{Z}}(t)$, one obtains the signal $\dot{\tilde{\mathbf{Z}}}(t)$.

5. PCA of the derivative data. As said in the introduction of the SFA algorithm, the signal with the least variance would be the slowest varying signal, while the one with the most variance will be the fastest varying one. Thus applying PCA over the derivative data will give the slow features, but appearing in the opposite order.

SFA has been used in object recognition in order to extract invariant representations from data.

An abstraction could be an invariant feature found in the data. That is why this algorithm can be useful for the purpose of looking for sensorimotor abstractions. Moreover, SFA could be used in a hierarchical way, allowing that way to extract more complex features from the previous ones.

SFA is also useful due to the criterion of slowness. It tries to find slowly changing signals that contain useful information. Those signals could be used as evaluation functions, critics, in order to indicate the agent how well is progressing a task that is being performed.

3.2.3 Denoising Source Separation (DSS)

Denoising source separation (DSS) algorithms are a family of algorithms for separating different sources that are based on denoising procedures. The model of the data is defined as $\mathbf{X} = \mathbf{AS} + \mathbf{n}$, where $\mathbf{X}$ is the noisy data matrix, $\mathbf{S}$ is the matrix of different sources, $\mathbf{A}$ is the mixing matrix and $\mathbf{n}$ is the added noise. The algorithm calculates the sources one by one. Its basic structure is

$$\mathbf{s}_i = \mathbf{w}_i^T \mathbf{X} \tag{3.19}$$

$$\mathbf{s}_i^+ = f(\mathbf{s}_i) \tag{3.20}$$

$$\mathbf{w}_i^+ = \mathbf{X}\mathbf{s}_i^{+T} \tag{3.21}$$

$$\mathbf{w}_i = \frac{\mathbf{w}_i^+}{\|\mathbf{w}_i^+\|}, \tag{3.22}$$

where $\mathbf{w}_i$ is the weight vector that extracts the source $\mathbf{s}_i$ from the noisy data $\mathbf{X}$. In the first step, Eq. 3.19, the source signal $\mathbf{s}_i$ is estimated from the data $\mathbf{X}$ and the current weight vector $\mathbf{w}_i$.

In the second step, Eq. 3.20, the source is recalculated using the previous estimation and the denoising function f, which could be a linear filter, a shrinking function, etc.. In general any linear and easy or non-linear and complex functions can be used. This step is known as the denoising step. Depending on the function applied, different kind of algorithms can emerge.

In the third step, Eq. 3.21, the new weight vector, $\mathbf{w}_i^+$, is estimated from the new source estimation, $\mathbf{s}_i^+$, and the noisy input data. In the last step, Eq. 3.22, the normalization of the weight vector is applied. The algorithm must be run until the convergence of the weight vector is achieved.

The structure of this algorithm, although simple, allows one to easily implement a huge variety of algorithms by only changing the denoising function f applied in Eq. (3.20). Some of the possible functions and their interpretations are discussed in [Sare 05]. Moreover, DSS is also highly interesting due to its accuracy in selection of useful information.

3.2.4 Independent Component Analysis (ICA)

Falling into unsupervised learning techniques, the goal of independent component analysis (ICA) [Hyva 00, Ston 05] is to find a linear transformation that transforms the original non-Gaussian data into components as independent as possible. Among its applications one can find signal source separation [East 07], image processing [Yu 06, Scar 05], sparse code [Chen 01] and feature extraction.

In order to describe the algorithm, the problem of the cocktail party will be used. Suppose you have n microphones (observed signals) in a room with k people (source signals). Let $\mathbf{x}_i$ be the signal obtained from the microphone i, with $i = 1, 2, ..., n$. One can express the observed signals as a mixture of the k source signals. Calling $\mathbf{s}_j$ to the source signal j, that is:

$$\mathbf{x}_i(t) = a_{i1}\mathbf{s}_1(t) + a_{i2}\mathbf{s}_2(t) + ... + a_{ik}\mathbf{s}_k(t) + \mathbf{n}_i(t) \qquad \forall i, \tag{3.23}$$

where $\mathbf{n}$ is added noise. The model presented here is linear. Moreover, if the model does not have added noise, $\mathbf{n} = \mathbf{0}$, it is called noiseless. This linear model can be expressed in the matrix form as $\mathbf{X} = \mathbf{AS} + \mathbf{n}$, being the same model as the one defined for DSS, where the observations are explained as the mixture of independent sources (plus noise). Since the mixing matrix, $\mathbf{A}$, and the sources, $\mathbf{S}$, are unknown, the variance of the sources will be also unknown, and thus, without loss of generality, unitary variance will be assumed.

The two necessary conditions the sources must follow are that they are statistically independent and that they do not follow a Gaussian distribution, although the distribution is unknown. The reason why they should not follow a gaussian distribution is related to the working principle of ICA. For finding the independent source signals, the ICA algorithm tries to maximize the non-gaussianity of the signals. This is equivalent to minimizing the mutual information.

Before applying ICA to the data, it is useful to do some preprocessing to it. Sphering the data, i.e. subtracting the mean and whitening it, makes the sources to have zero mean and the mixing matrix orthogonal, since the process of whitening is equivalent to removing the correlations in the data. This saves the number of parameters to calculate, thus making the algorithm faster.

The ICA algorithm for one cell is a particularization of the DSS one. In this case, the algorithm tries to find the weight vector $\mathbf{w}$ that maximizes the non-gaussianity of the source signal $\mathbf{s}$, whose estimation is $\mathbf{w}^T\mathbf{X}$, and it will use a denoising function f that accomplishes that purpose. The algorithm starts with a random weight vector which is iterated until convergence of $\mathbf{w}$

$$\mathbf{s} = \mathbf{w}^T\mathbf{X} \tag{3.24}$$

$$\mathbf{s}^+ = f(\mathbf{s}) \tag{3.25}$$

$$\mathbf{w}^+ = \mathbf{X}\mathbf{s}^{+T} \tag{3.26}$$

$$\mathbf{w} = \frac{\mathbf{w}^+}{\|\mathbf{w}^+\|}. \tag{3.27}$$

In [Hyva 00], the function $f(\mathbf{s})$ used is the derivative of a function that measures the non-gaussianity. Some approximations for this measure are the negentropy

and the kurtosis. The first three steps in the algorithm can be written together as

$$\mathbf{w}^{+} = \mathbf{X} f(\mathbf{w}^{T}\mathbf{X}). \tag{3.28}$$

In the fastICA algorithm [Hyva 99], this step is replaced by

$$\mathbf{w}^{+} = \mathbf{X} f(\mathbf{w}^{T}\mathbf{X}) - f'(\mathbf{w}^{T}\mathbf{X})\mathbf{w}, \tag{3.29}$$

where f' denotes the derivative of the function f. This is based on a fixed-point iteration schema.

In case that multiple sources are to be estimated, their weight vectors must be decorrelated after each iteration. There are several methods to achieve that, see Ref. [Hyva 00].

After the convergence of the weight vectors, several independent sources will be obtained, as the result of projecting the data over the obtained weight vectors, that is, $\mathbf{S} = \mathbf{W}^{T}\mathbf{X}$.

Chapter 4

Model: The cognitive architecture

In this section the model used for learning sensorimotor abstractions, called a *cognitive architecture*, will be described. The model is biologically inspired over the architecture schema present in the cerebral cortex. In that sense it is not a new idea, since many authors have been previously trying to merge the best of biology and engineering approaches to the learning problem in order to design artificial intelligent agents [Dean 05,Alic 08]. Many robotics projects nowadays are focusing in learning sensorimotor abstractions in order to interact in an efficient way with the environment such as SENSOPAC [Arle] in the European framework or DOMO [Edsi 04] at the MIT.

In the next sections, it will be justified why it is important to use a learning system instead of a hard coded (completely programmed) one and why the abstractions are important. After that, there will be an explanation about the principle used for learning abstractions, *slowness*. Following, the cognitive architecture will be described. Some possible scenarios where this model could be used will also be presented. Last, as a comparison, there is a discussion about previous works related to this theme, which contain different approaches for solving the problem.

4.1 The importance of learning algorithms

When an algorithm is manually programmed in a computer, the computer will execute this algorithm step by step, following the instructions. It will always do exactly the same with the same performance. But that performance may not be the best that could be achieved. There may be other approaches or solutions to the problem that offer better performance. By using learning algorithms, there

is an open path for exploration, that is, trying to find new ways of getting the correct result, but in a more efficient way.

A clear example of how the approach taken to solve a problem can affect greatly the performance, can be found in athletics. Imagine the case of high jump. There, the athletes have to jump over a bar placed at a certain height without any tool except their own bodies. There are no other rules. Whoever invented the sport in the 19th century, took an approach in order to solve the problem. The name of this first approach was called scissors technique. This is the technique that the coaches taught their pupils as time passed. That could be the same as the manually programmed algorithm. However, as time passed and with trial-and-error of different approaches (exploration) to solve the problem, new techniques (solutions) appeared, such as the Eastern cut-off, the Western Roll and the Straddle technique, which not only solved the problem, but also gave better results (performance). A new revolving technique, called Fosbury flop, appeared in 1968, which greatly improved performance. This not only happens in athletics, but in almost any field one can think of. Intelligent reasoning, exploration, and with that, learning, is what has made it possible to have nowadays the luxury and comfort humans enjoy.

On the other hand, it is not always possible to program a certain type of behaviour, due to either the complexity of the algorithm or the problem to be solved, or even because in each different context, time or place, the solution changes. In this case it may be possible to program a certain amount of guidelines that will allow the system to explore different possibilities, finding the correct solution in each possible environment or situation that it may encounter.

Imagine an agent that picks up things. One could teach the guidelines or fundamentals of grasping, such as opening the hand or gripper and closing it, or approaching to the object to pick. However, the way of approaching cannot always be the same since it is highly context dependent: it is not the same to pick a book from a shelf as from a table or from the floor, not to mention the huge variety of objects that can be picked, and for each one of them there is an optimal way of picking that cannot be taught or hard coded case by case. In fact, evolution has provided humans with the tools that are needed to learn things, but there are only few things that are genetically coded. Moreover, the environments and solutions change with time, especially fast within the last few years, whereas evolution is a really slow process. Hard coding everything would not only prove useless, but counter-productive.

In brief, learning allows to use, in the most efficient way, the tools that one can use to solve each particular task.

4.2 The importance of abstractions

An abstraction is a simplified model of something, a representation of that something, a group of *features*. For example in order to paint a representation of a person, a typical abstraction is a long vertical stick (body), two other tilted sticks in the lower part (legs), other two almost in the most upper part (arms) and a circle in the top (head). An abstraction of a house, could be represented as a square and a triangle, maybe adding two small squares inside the big one (windows) and a rectangle (door).

There are not only abstractions of things related to abstract drawings. For example, an abstraction of a fixed line in $\mathbb{R}^2$ can be represented mathematically as $y = ax + b$, where $a, b \in \mathbb{R}$ are the parameters that characterize, define and unequivocally identify the line, the features. One important characteristic of the abstractions is that an abstract representation of something is not unique. For example, the equation $(y, x) = \mathbf{p}(1-t) + \mathbf{q}t$ is another mathematical abstraction of the same line, where now the features are $\mathbf{p}$ and $\mathbf{q}$, which are points in the 2D space that belong to the line in question. It may also happen that some abstractions are better than others in certain cases, as it happens here, since with the first one, $y = ax + b$, it would be impossible to represent the line $x = const$ if the a and b features were fixed.

According to the previous examples, one could think that an abstraction of a thing consist of the essential features that describe it. If one needed to describe a glass for example, one would not make reference to the color, the size or the form (topology). Those are not the essential features as there are glasses of many colours and forms, but one would maybe describe it as a recipient where one can pour liquids and which is used for drinking them. In this case the abstraction is made based on the object's function.

Many times, when talking about abstractions or features, one could understand them as an invariant characteristic of something. Typically, something is invariant with respect to space or time or both at the same time.

So an abstraction is a simplified model of something, but why are abstractions useful? It is impossible to work with every detail of every thing; every object is different: as there are no two identical people, there are no identical objects. Think of an apple, for example. There are apples of many different colours and sizes, with different forms within the same varieties: some are almost perfect spheres, others are flat over a small area, the colours are not exactly the same neither the "birth-marks". One would need infinite amount of space and time in order to classify and "label" every single object one encounters. Another example are the police databases. They use abstractions, most distinguishable features, of fingerprints or faces in order to analyse them. Having to store every single detail of everything is practically impossible.

In the same way, in the case that concerns this work, it would be impossible to program every single task or every single action to take for each possible situation that may occur, like in the example explained before about grasping an object. Just try to imagine the infinite number of possible scenarios.

4.3 Slowness as a criterion

An important remark about the cognitive architecture is that it uses slowness as a criterion for extracting features. Using the *slowness principle* [Spre 09], that is, finding useful signals whose temporal difference is small compared to the raw inputs, the agent will be able to represent, in an autonomous manner, the current state of its surrounding environment. The slow signals could be interpreted as invariant representations, that is, representations that remain stable over time or space. Then, the slowness principle is useful for learning invariant features.

It is believed that slow changing features are more likely to carry relevant information than fast varying signals. The reason is that the entity of something does not change as fast as the sensory input perceived. In order to illustrate this, imagine any object in motion, such as a spinning coin or a tree blown by the wind. The raw sensory input perceived by the visual receptors vary quickly while some relevant features of the objects or their environment do not, such as the entity of the object (coin or tree) or their position in space. Those invariant signals are much more informative than the raw sensory signals.

The slowness principle is related to the critic in TD learning. The critic evaluation is expected to be stable over time, and unsupervised learning algorithms, based on the slowness principle, try to find relevant and slowly changing data. This information found by the unsupervised algorithms could be useful for evaluation, that is, serve the critic in the cognitive architecture.

Some unsupervised learning algorithms used in the model follow this principle. One particular and popular implementation of the slowness principle is carried out by SFA, which tries to minimize the variance of the temporal derivative of the data.

Applying slowness as a criterion is not something new. Many authors have been using slowness as a way to recognize objects [Wall 97, Spre 07] and as other kind of learning invariant features such as patterns found in complex cortical or hippocampal neurons [Kör 04, Berk 05, Fran 07] or new skills [Beck 92].

4.4 The model

In this work, a structure for learning sensorimotor abstractions is presented. It is called a cognitive architecture, defined and explained below. This method is

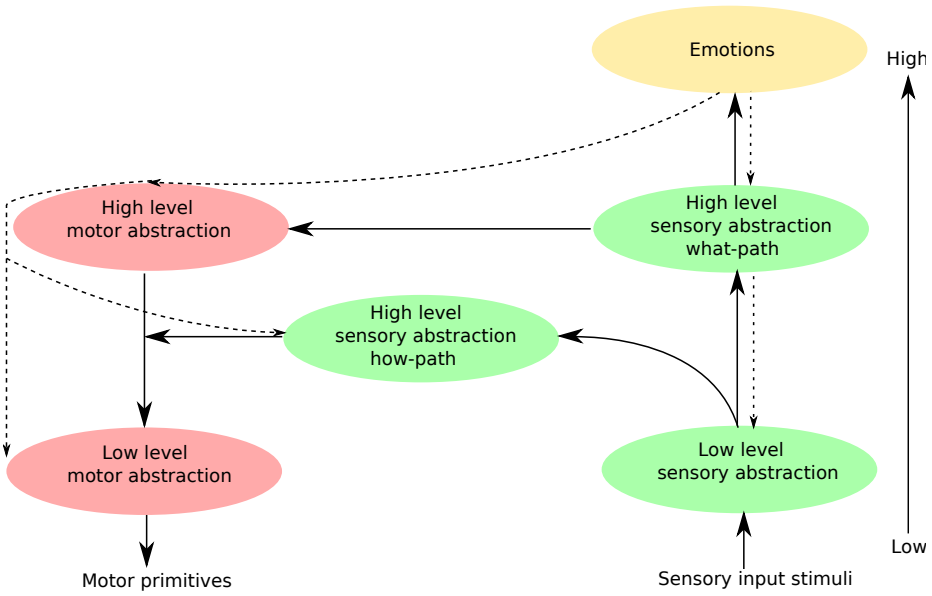


Figure 4.1: *(Colour on-line) Schema of the cognitive architecture with one level of hierarchy. The light red ellipses, on the left side in the figure, represent the motor modules while the light green ones represent the sensory modules. The arrows represent the flow of information during learning. In discontinuous lines, the flow of the emotion (reward) signal can be appreciated.*

not intended to substitute other learning techniques, but to complement them. It could be used as a basis for learning more complicated skills using different learning techniques.

An architecture describes a structure, the organization and relations of different parts. In this context, the brain architecture describes how the different parts of the brain, such as the cerebellum, the basal ganglia, the hypothalamus, cerebral cortex, etc. are connected with respect to each other. Similarly, the cognitive architecture describes how different modules that correspond to the sensor and motor parts in the brain are interconnected in order to learn and process data.

There are two main modules that together define this cognitive architecture, indicated in Fig. 4.1 with different colours: *sensory modules*, that analyse and process data that originates from the sensors; and *motor modules*, that give different commands to other parts of the body (i.e., motors in an agent and muscles in humans) in order to achieve a movement.

Before starting the discussion about the cognitive architecture in more detail, it should be emphasized that motor and sensory modules are related. When making a movement, that is, executing a motor command, there is a change observed in the flow of sensory data. For example, now writing this text, the different motor commands sent to the fingers can be reflected in the visual sensors (since more text appears on the screen) and in tactile or pressure sensors located in the fingers.

That is, every motor action taken has an effect over different sensors. Then, by observing the changes in sensory data and which motor commands caused those changes, it could be possible to learn to use the motor outputs in order to achieve a certain goal that could be measured with some sensors.

Nevertheless, it is not always obvious how to map low-level sensory data with low-level motor commands. It is usually easier to relate higher-level concepts of sensory and motor actions than low-level ones. That is why inside those two main modules (sensory or motor) there are several submodules, which represent different levels of abstraction in the data: some will correspond to basic concepts (sensory or motor depending on the type of submodule) while others will correspond to complex ones.

According to the previous statement, and taking into account the representation of the cognitive architecture, the lower the submodules are the more basic they are. In fact, the submodules at the bottom just process input raw data or send the most basic motor commands. The ones over them select and process the data that has been obtained in previous levels computing, in that way, more abstract data, and so on, as it happened in the cortex, see Sec. 2.2.2. It is possible to define several layers of processing, and with that, different levels of abstraction. Thus, the model is hierarchical and it permits to learn how to perform difficult, complex tasks based on easier subtasks.

This hierarchical structure is based on the architecture found in the cerebral cortex (see Chapter 2). Different layers work with different levels of complexity. It has been demonstrated by several studies that there is a hierarchy of processing in the cortex. The first of those studies, which proved that certain areas process higher-level of information, were carried out in the 80's, and have been reinforced over the past decades [Fell 91]. Forward (ascending) and feedback (descending) neural connections have been used to identify these hierarchical relationships.

In Fig. 4.1 one can observe a single level hierarchy of both motor and sensory modules. One level of hierarchy in the sensory part does not need to correspond to one level of hierarchy in the motor part or vice versa: it could correspond to several levels in the other part. Also, higher-level modules could receive data from many low-level ones, not only one and not from only the ones right under them. A more complete architecture could look like the one shown in Fig. 4.2.

As explained before, an abstraction is a simplified representation of something, for example a task or an object. The task could be grasping an object, serving a cup of something, sitting, etc. Such actions require many other low-level tasks to be taken in order to be able to carry them out, like moving the hand up or down or closing or opening the hand or gripper. It is this hierarchy of increasingly abstractions of the model that allows the system to learn and solve advanced and highly complex tasks.

In the rightmost part of Fig. 4.1, one can observe the sensory modules. As mentioned before, these modules process all the data relative to sensory informa-

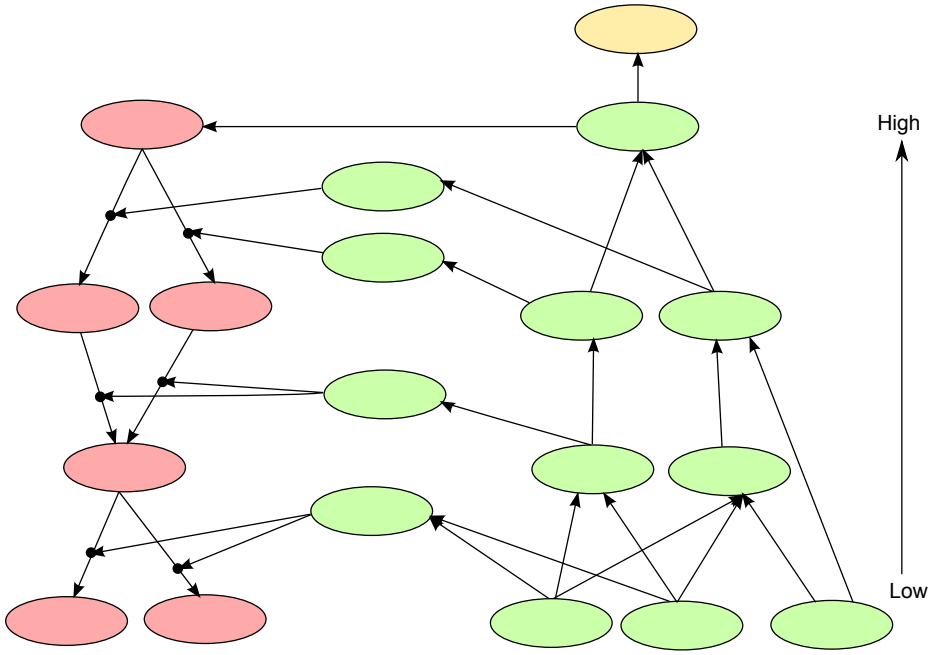


Figure 4.2: *(Colour on-line) Schema of the cognitive architecture with several levels of hierarchy. The light red ellipses, on the left side in the figure, represent the motor modules while the light green ones represent the sensory modules, and the yellow the emotions. There could be more emotion modules related to lower sensory modules.*

tion. The bottom module, called the low-level sensory abstraction, will process the raw sensory input data that originates in the sensors. Its main purpose is to find useful information among the sensory data that can be used in more complex sensory modules. In many real-world problems, most of the data perceived by the different sensors is useless, and only a small part of it contains the essential, important information. That is why it is needed to extract the relevant information from the data. As noted previously, abstractions are usually related to invariant features, and that is why the slowness principle is used (see Sec. 4.3).

In order to find invariant information, that is, the higher-level sensory representations or abstractions, machine learning algorithms are used. As discussed in Sec. 3.2, unsupervised learning has been proven useful in extracting invariant (slowly changing) features from data. For instance, slow feature analysis (SFA), which tries to find structures in the data that change the slowest; or principal component analysis (PCA), which localizes the signals in the data that provide the most information (those with highest variance); or independent component analysis (ICA), or its generic version denoising source separation (DSS), which looks for statistically independent sources within the data; or, in general, any other unsupervised learning algorithm.

Thus, in general, all extracted features or higher-level representations of the

information hidden in the raw data are used to predict the emotions or rewards, that is, a critic that evaluates how well or bad a certain task was performed, for example, if while writing with the laptop the expected key was pressed or not by observing the change in the screen (abstract data obtained by processing the visual sensory input). The emotion, or critic signal, flows downwards in the architecture towards the different sensorimotor modules.

These higher-level features, extracted from the low-level sensory modules and used to predict the emotions, answer the question of *what* is the useful information found in the sensory data, and that is why they are called "high-level sensory abstractions what-path" or shorter, "what sensory abstractions", in order to distinguish them form other sensory abstractions that will be introduced below. The "what sensory abstractions" make reference to the *ventral pathway* found in the cerebral cortex (see Sec. 2.3).

As explained, the sensory modules are related to motor modules. That way one could describe how changes in a high-level motor module affect changes in a sensory module and vice versa. The way that high-level sensory abstractions map into high-level motor ones may vary depending on the scenario.

Once these high-level motor abstractions, which are goal oriented, have been obtained the question would be *how* to pass from these high-level to low-level ones. This step is not obvious without the help of some latent variables that can be obtained from the sensory input. Those are what it would be called "high-level sensory abstractions how-path" or again, shortening, "how sensory abstractions". As it happened with the other kind of sensory abstractions, the "how sensory abstractions" make reference to the *dorsal pathway* found in the cerebral cortex (see Sec. 2.3). Motor commands (low-level motor abstraction) will cause changes in the sensory input data received (low-level sensory abstraction), that is why it is useful and necessary to have those two in order to correlate them to a higher-level motor abstraction. The two arrows crossing in the left part of Fig. 4.1 represent the correlation between the three modules. For learning, it is necessary to have the three of them. Once the mapping between them has been learnt, by having two of them, one could determine the third one.

Then, once the mapping between different motor levels and sensory ones has been determined, it will be possible to generate the proper motor commands in order to maximize a certain reward. This reward (emotion) is related to a certain pattern of changes in high-levels of sensory abstraction and with that to changes in basic sensory data. The criterion the motor abstractions follow is to be able to manipulate the slowly varying sensory abstractions.

More complex architectures could be implemented using the one introduced here as a building block. By combining those blocks, one could obtain higher-level of abstract features, and with that higher-level of complexity. In case of combination, the input that higher sensory levels receive would be the outputs obtained by one or several of any of the lower-levels. In that way there could

also be several blocks working in parallel (same level of complexity) that feed higher-level blocks. In the motor modules, the outputs of the higher-levels would go to the lower-levels until the motor primitives are obtained.

Moreover, the cognitive architecture that only takes into account motor and sensory hierarchies can be expanded to a more complete brain architecture like the one present in Ref. [Valp 05].

4.5 Examples of sensorimotor modules

One sensorimotor module can have several layers of complexity in both the motor and the sensory modules.

An example of a layered sensory hierarchy, could be based on the one present in the visual cortex. In the low-level, the primary visual cortex, the neurons represent orientation of lines. In the next level, area V2, they represent contours and figures. Proceeding that way, it would be possible to arrive at higher-levels of abstraction that represent different objects such as a *house*, a *tree* or a *car* to name but a few.

An example of a layered motor hierarchy could be having a high-level concept such as *write something*. Obviously this is an abstraction since depending on the context (i.e., using a computer keyboard or pen and paper or a mobile phone keyboard) the sequence of actions (muscles or motor movements) changes. Also, depending on the desired text to write, the commands sent to the muscles will change. A lower-level concept, that is, more specifically, could be to *write with a computer keyboard*. That would be followed by a lower-level that could be, for example, *pressing a keyboard key* and so on until everything is translated into basic signals that will cause the movement of different muscles.

Many of the situations that a human being encounters daily could be solved by an agent using this cognitive architecture model as a basis. Some examples are such tasks as *picking an object* (a book from a shelf, a glass from a cupboard), *pouring coffee into a mug* or *boiling something*. For example, in the case of *boiling something*, the final task could be divided into *picking the kettle*, *pouring water to the kettle*, *turning on the fire*, etc.

Following the principle of a sensorimotor hierarchy, one could think of dividing the tasks into much simpler ones. One unique sensorimotor module may not be enough to solve a particular task, but a combination of them could. In that sense, the sensorimotor modules would be the bricks of which the final task is composed. For example in the case of boiling, in both the motor and the sensory modules.

The example implemented in Chapter 5 corresponds to one level of hierarchy of the cognitive architecture. In the example, an agent learns to move inside a room.

In the lower-level of sensory modules there would be the sensory stimuli coming from the sensors, preprocessed and expanded. The higher sensory modules will represent concepts such as *distance*. The objective is to *move towards a target* or to *move further from* it. That would be the high-level motor abstraction (*getting close* or *getting further*) while the signals sent to the motors will be the motor primitives found in the low-level motor module.

4.6 Previous work and approaches

There have been several researchers that have also studied how to implement the sensorimotor abstractions, but they have taken another approach than the one presented here. More and more researches are using the advantages of learning over a complete hard-coded architecture, learning from interaction with the environment, such as Ref. [Smag 97, Olss 06, Koni 08]. It will be impossible to mention all the works that have been and are using such models. Nevertheless, some of themt, that can represent several parts or ideas contained in the cognitive architecture are mentioned and briefly described in the next paragraphs.

For example, in both Ref. [Cohe 97, Muga 07], they observe how the sensory input data changes through the execution of different motor actions. They look for associations among the different streams of input data, observing which set of data change in the same moment or just right after another has changed. With those observations, they learn simple rules that associate motor actions with changes produced in sensory data. The new rules will create a new level of abstraction that will make it possible to learn more complex rules by seeking and finding associations between these higher abstractions.

Taking into account the comparison between the nervous system and a general purpose computer, the same can be extend between an automaton and a newborn child. That is, an agent, as the baby, knows nothing about how everything works. It will receive a lot of sensory input, from different receptors. In the beginning this information will make, probably, no sense at all. By exploring through the execution of different actions and perceiving the changes that those actions (motor commands) make over the the sensory inputs, it will be able to find correlations and "hidden" structures within the input data. Once it learns about the motor actions that causes the inputs to change, it will be able to learn what motor actions needs to execute in order to get to a certain sensory state from the given sensory input. This is the principle followed in the sensorimotor abstractions. By learning these basic structures, it will be able, by selecting, combining them and extracting more features from them, to learn more complex tasks as it is demonstrated in Ref. [Koni 09]. It is an example of extracting high-level abstract categorises from selection of low-level abstractions.

A model similar to the sensorimotor cognitive architecture proposed here, can be found in Ref. [Prov 05]. They also use high-level sensory and motor abstrac-

tions. The sensory features are obtained using unsupervised learning, like it happens in this work and in the cortex. More specifically, SOM networks are used. Moreover, they show that the agent that tries to navigate in a certain environment, performs faster when having high-level actions. In a sense they demonstrate empirically one of the advantages of a hierarchy. Nevertheless, the approach taken in Ref. [Prov 05] differs from the one taken with the cognitive architecture.

Ref. [Butz 08] uses an architecture in which an agent can learn cognitive maps of the environment that will be useful to guide it through a maze finding optimal paths between two different points. They use sensors to take data and find abstractions via time-growing neural gas (TGNG) algorithm that guides the motor control. The agent has no previous knowledge about its environment as it happens here.

In Ref. [Lege 10], they use a hierarchy of unsupervised learning algorithms, specifically SFA, in order to reduce the state-space dimensionality of the problem and being able to apply reinforcement learning algorithms. In that context, the abstractions or features found via SFA are the basis for other learning algorithms allowing to execute more complex tasks, which is the aim intended with this cognitive architecture. In general, approaches that use slowness as a criterion in order to learn invariant representations could be combined with the proposal presented in this work, since they share the same principle. That is the case of the research projects mentioned in Sec. 4.3 and many others.

Moreover, nowadays there is a tendency towards autonomous learning in robotics applications, letting the systems learn by themselves, acquiring new skills and perfecting old ones via machine learning and biological inspired techniques.

Chapter 5

Implemented example: The waiter agent

Using the model of the cognitive architecture, an example of a waiter agent has been implemented. In this example an agent will learn how to take orders from a customer to the bar and deliver it again to the costumer. A highly abstracted version of that scenario will be simulated below. Mathematically, both the customer and the bar represents a point in a space, in this case, a 2D one, and the sensory information perceived will not be an image, but a random vector.

In the beginning, the agent knows nothing about its environment. It receives different sensory input from the cameras. As a baby, the only thing that it can do is to move randomly in the space. Then, how can it learn to move from a certain point in space to another one? How can it know the concept of a distance? How can it know the concept of a place?

Note that this example has been chosen to illustrate the working principle of the model explained in the previous chapter. With a different motor set-up, using for example the speed and angle, this particular problem could have been easily solved using reinforcement learning since the state space is not that large. However, the main aim of this work is demonstrating that it is possible for an intelligent being to find useful representations about the environment and its current state, and acquire new skills, using non-trivial motor configurations, by means of unsupervised learning based on the slowness principle. This principle is very useful in environments with a large state (or action) space or in those that reward signals are non-existent.

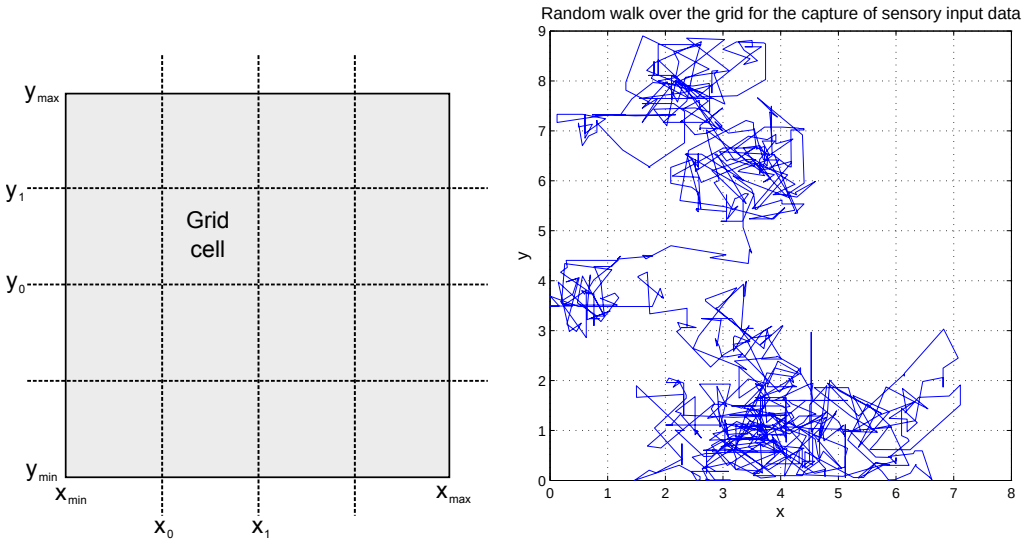
Note also that this example does not correspond to how the brain learns about space or walking, since evolution has provided animals with specialized mechanisms for that purpose. Nevertheless, the results obtained are easy to visualize, which simplifies the understanding of the principle being used. The model can be applied to other general problems.

5.1 From raw sensory input data to sensory abstraction

Solving the problem at hand, the first step is not planning how to go from one point to another, not even planning how to move, but trying to understand the vast amount of sensory input data perceived.

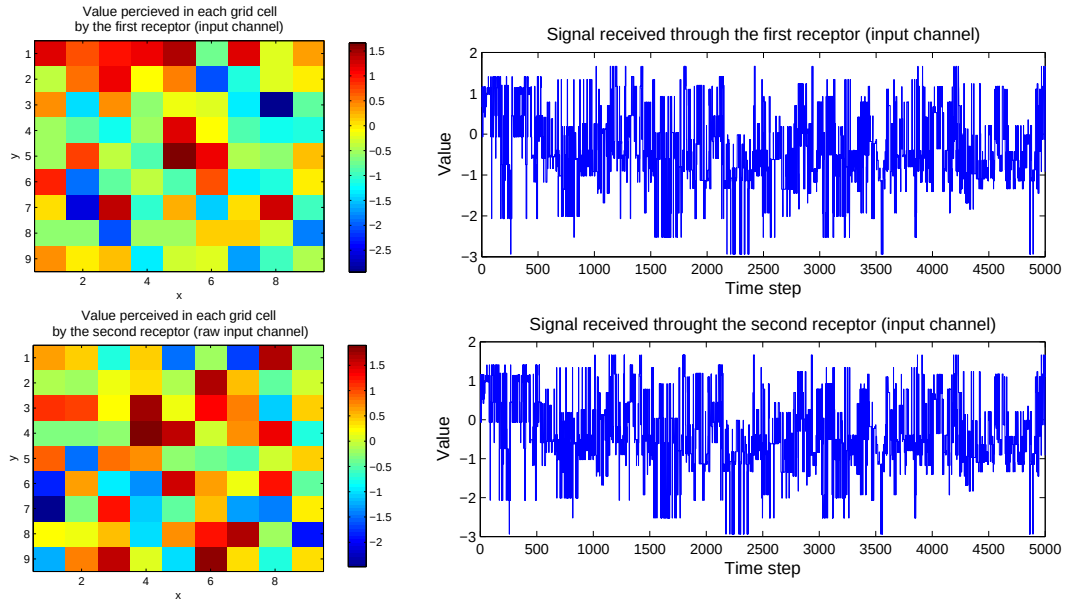
As mentioned before, the agent will be able to move, but in an uncoordinated way, which will result in random movements. By using random walking, different sensory input data will be collected and finally processed. From the processed sensory input data, sensory abstractions will be found.

The agent is confined to move within a limited 2D space, called grid (see Fig. 5.1(a)). Even thought the agent movements are continuous in the 2D space, in order to simplify the problem, the grid is divided into *grid cells* (a position within the grid, see Fig. 5.1(a), not to be confused with the hippocampal grid cells). In each grid cell, the agent perceives the same sensory input signal, see Fig. 5.2(a). This will not only simplify things, but it will be a way to recognize the grid cells, remember a position within the grid, although not enough to know their relative position in space. The sensory signal consists of a n-dimensional random vector, where each dimension is the data perceived by a receptor (input channel). The data perceived by a receptor in time will be denoted as *signal*. In the results



(a) (b)

Figure 5.1: (a) *The agent will be able to move in the box* $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$. *The grid is divided in smaller grids called grid cells. In each grid cell the agent perceives the same sensory stimuli.* (b) *A path followed by the agent during the first 1000 steps of random walk.*



(a) Raw input channel

(b) Random walk

Figure 5.2: (a) *Raw sensory input perceived by the first and second receptors (input channels) in each of the grid cells. The value perceived is a random number.* (b) *Signal perceived in the first and second input channels during 5000 steps of the random walk.*

presented in the figures $n = 15$ was used. Moreover the grid had a dimensionality of 9×9 grid cells.

The aim of the agent is to learn to walk from one point to an specific one. In order to do that, it needs some mechanism that would allow to identify points in the 2D space. One way of identifying points, quite mathematical one, would be to use coordinates. These can unequivocally represent every single position in space. Therefore, the first aim is trying to find a *coordinate system* as a higher-level sensory abstraction, by processing the sensory data perceived.

5.1.1 First approach: slow features

As said previously, the agent will move randomly in the 2D grid. One can notice that by moving in a random and uniform (same probability) way in any possible direction, the feature that changes the slowest is the position. Mathematically, this can be proved by observing the probability distribution of finding the agent in a particular grid cell at a time t_1. This is know as the *random walk*. If analysed in detail the random walk, one would obtain a Gaussian distribution as the probability to find the moving object in a position p_i in at any time t_j. The Gaussian distribution is centred in the initial position. Moreover, if the space is unlimited,

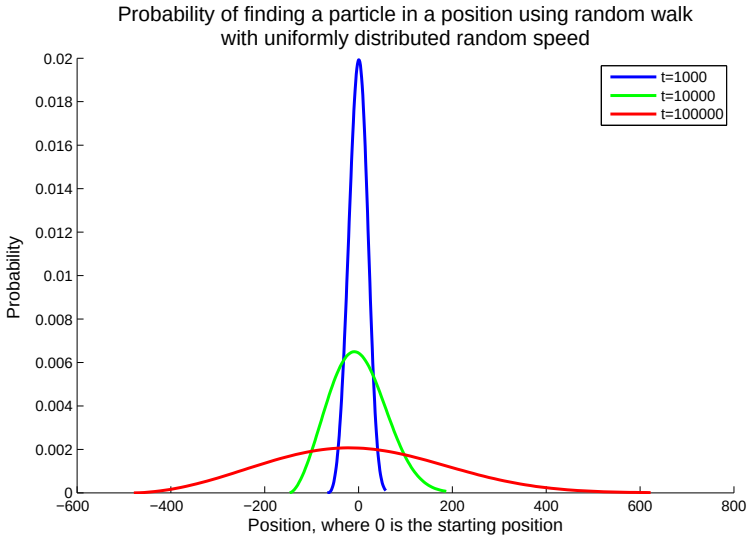


Figure 5.3: *Probability distribution of the random walk in different times. It indicates the probability of finding the agent in a certain position, being 0 the position at t=0. The movement made* $m \in [-1, 1]$

the Gaussian bell would look more spread as the time grows. That is, the object could arrive to a further position. In conclusion, it is more probable to find a moving object close to the initial position, or said in another way, it is hard to advance somewhere when the direction of the movement changes constantly. Due to these characteristics, a slow feature algorithm was applied.

The data analysed with the slow feature analysis was collected in the following way: in each time step, the agent gave the motor primitives a random value, which caused a movement in a random direction. All the the directions in the 2π circle had the same probability of being selected. The agent moved a random length and collected, after the movement, the sensory input vector, formed by the values of the n receptors (see Fig. 5.2(b)). It is worth mentioning that even if the movements are not the same length (the length is a uniform random variable between [0,1]), the probability distribution is still a Gaussian distribution (see Fig. 5.3 for an example in 1D). The only difference is that is less spread. The first samples of the random walk over the grid can be visualized in Fig. 5.1(b).

After collecting the data, this was normalized so that each of the n signals had zero mean and unit variance. In order to have higher dimensionality, the data was expanded using the original signals and the second order terms that their combinations generated, that is, $Z = \{s_1, s_2, ..., s_n, s_1s_1, s_1s_2, ..., s_ns_n\}$, where Z is the data expanded, and s_i is the signal i.

After the expansion of the input data, the data was whitened (also called sphered), that is, it was forced to zero mean and identity matrix as a covari-

ance one, using Eq. (3.17) and (3.18). Whitening makes it easier to treat the data in later steps.

It is known that there is some whitening over the data happening in the brain before they reach the cortex, as it happens with the sensory stimuli from the retina [Grah 06]. Whitening facilitates learning, since it simplifies the task of competition in the bottom-up stage. Without whitening, neurons tend to activate at the same time and learning pushes the weights (neural activations) in the same direction, which is a tough problem for inhibition.

Following the SFA algorithm, after whitening PCA is applied over the correlation matrix of the derivative of the expanded and sphered data $\tilde{\mathbf{Z}}$. The derivative of a discretely sampled (digitalized) signal is defined as the rate of change of the quantity in the *image* of a function, in this case $\tilde{\mathbf{X}}$ with respect its *domain*, in this case time $\mathbf{T}$, that is, $d\tilde{\mathbf{X}}/d\mathbf{T}$:

$$\dot{\tilde{\mathbf{x}}}_{\mathbf{j}} = \frac{\tilde{\mathbf{x}}_{\mathbf{j+1}} - \tilde{\mathbf{x}}_{\mathbf{j}}}{\Delta t}\ , \qquad 1 < j < n-1, \tag{5.1}$$

where Δt is the time different between consecutive samples of $\tilde{\mathbf{X}}$. This is equivalent to a high-pass filter since it attenuates the more the lower the frequency of the signal.

Thus the derivative step could be changed into a filtering step. Using high-pass filtering will order the eigenvectors and eigenvalues such that the first one corresponds to the fastest feature. Using low-pass filtering will produce the equivalent results (eigenvectors and eigenvalues) but in the opposite order.

The results obtained are illustrated in Fig. 5.4. The two slowest features always correspond to the first order terms of the coordinates, being these x, y, $x+y$ or $x-y$ (or with the opposite sign). The third slowest feature corresponds to the product xy. The fourth and fifth to any of the following second order combinations: x^2, y^2, x^2+y^2 or x^2-y^2, or with the opposite sign. And so on, always faster varying features corresponding with faster order terms of the combinations of x and y even if the initial data did not contain any information relative to the coordinates. The information was 'hidden' in the structure of the data collected by the movements of the agent.

Going back to the problem of learning how to move, the slow features could be used to guide the direction of the movement. A specific place (grid cell) is distinguished by the sensory input data received, since for each particular grid cell, the same sensory input is received. Then, by calculating the distance to the target from the actual position of the agent and trying to minimize it while moving, the target could be reached.

The distance from any grid cell to another one can be calculated using the first two slowest features. One should calculate the square error between the actual

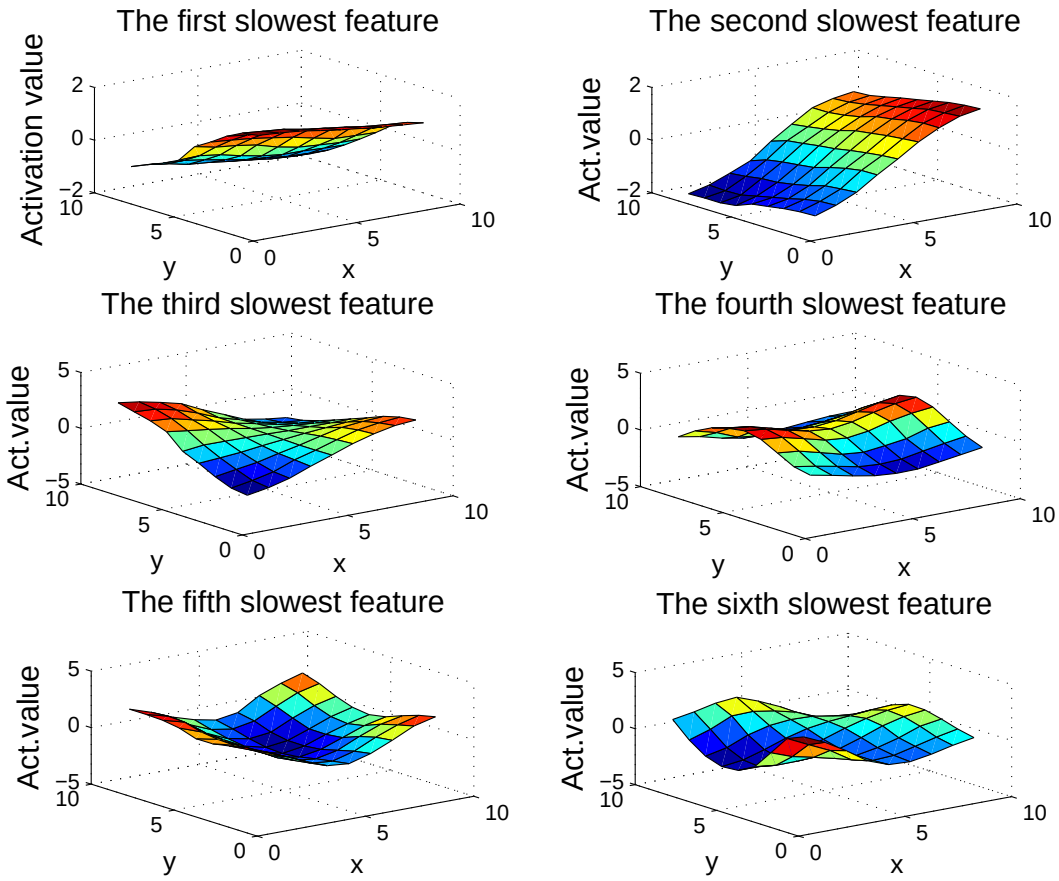


Figure 5.4: *The first six slow features visualized as a 3D image.*

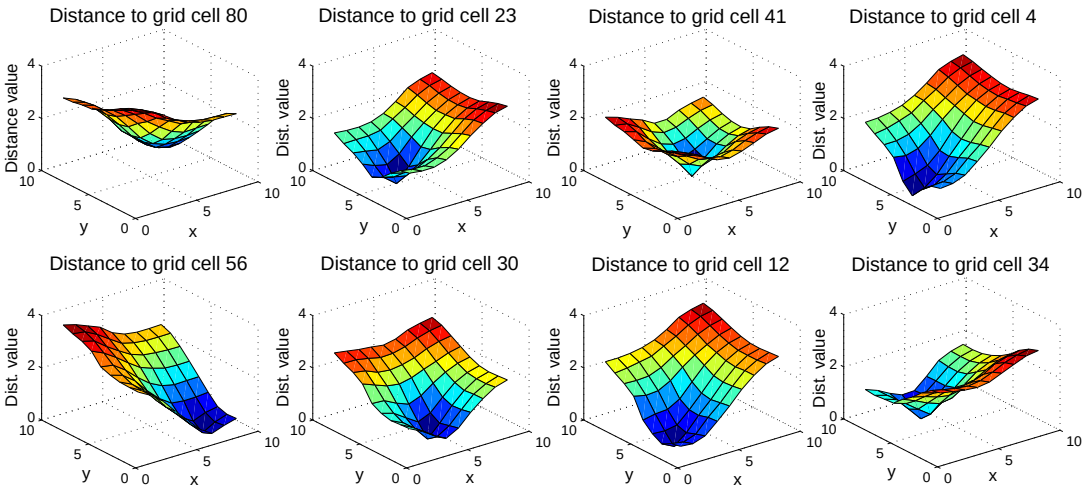


Figure 5.5: *Distance function from every grid cell towards eight different grid cells, represented in 3D. The total number of grid cells is 81. The numbers in the graphics indicate the number of the target grid cell.*

grid cell the agent is and the desired destiny point. In equations, if $\mathbf{S_1}$ corresponds to the first slow feature and $\mathbf{S_2}$ to the second slow feature the distance d to a point p from a point q would be defined as:

$$d_p(q)_{sfa} = \sqrt{(\mathbf{S_1}(q) - \mathbf{S_1}(p))^2 + (\mathbf{S_2}(q) - \mathbf{S_2}(p))^2}, \tag{5.2}$$

where $\mathbf{S_i}(j)$ represents the value of the slow feature i in the grid cell j. One can see the defined distance from every point q in the grid to eight particular ones in Fig. 5.5.

In this configuration the principle used was slowness. Even if is good enough for finding invariant features from the environment, their higher-order terms are not sparse. Sparseness is a property that can be found in biological beings. Some neurons use sparse code for representing abstractions.

A new approach that not only uses the slowness principle, but also the sparseness one, will be implemented next.

5.1.2 Second approach: slow and sparse features

It is known that some neurons in the hippocampus, called *place cells*, fire (activate) when an animal moves in a specific location [Mose 08]. There are other kinds of cells in the hippocampus that represent abstractions related to place (*grid cells*) and direction (*head cells*), but only *place cells* will be used in this approach. It would have also been interesting to implement *head cells* in order to recognize and use information about the direction of the movement, but in the approach used in this work the agent head always points to the same direction and not in the direction of the movement.

Then the aim of this approach is to mimic the *hippocampal place cells* in order to determine a place and how to go from one place to another. In general, what any neuron should achieve is to maximize the amount of information stored and represented. Thus, it is desirable that the neurons present independent patterns of activations among them. In brief, the aim is to cover all the space with different neurons whose activation patterns overlap the minimum possible. Therefore, a place, in this case a grid cell, would be represented by the activation levels indicated by the all neurons.

One can see that using few neurons for representing the whole space (in this case, grid) will provoke that different places (the grid cells) will share the same activation level, that is, they will be indistinguishable even if they are different cells in the original grid, even if the sensory input data is different. In fact, that would happen as long as the conjunct of neurons would represent with the same activation levels more than one of the cells in the grid. This is the same principle

as GPS triangulation with satellites, in which a point $\mathbf{p} \in \mathbb{R}^3$ is located according to the distance from each of the satellites. In that context, each satellite would be a neuron, and the distance to the satellite would be the activation level that the neuron presents in a place. In general, for triangulating a position with distances in a 3D space, four satellites are needed. For triangulating more than one position simultaneously more satellites will be needed. The same holds with neurons.

Then, in order to be able to identify all the places (grid cells) uniquely, one could think of using as many neurons as cell grids are. With this configuration every place would be determined without any confusion, but it raises the question of how to go from a certain place to another, since the activation pattern in the rest of the grid of that particular place cell would be null or almost null, since the activation patterns try to overlap as little as possible. Then, with that many neurons, the agent would know only when it is in the correct destination, but the only way of arriving there would be by random walk, which is not the desired solution.

The previous problem is not present if there is a small number of neurons. If only one neuron represented the whole space, and the activation pattern over the space looked like a hill topographically, there would be information about how to go from one place to another by trying to move towards the activation level (high in the hill) that the target has. That could be easily achieved by moving towards places with higher activation level, if the current place presents a smaller than the one of the target, or vice-versa. Nevertheless, there is indeterminacy when several places share the same activation level. This indetermination could be solved if, once the desired level has been reached, there was more information where to proceed next, moving within the activation level indicated by that first neuron. Following this idea, and using several neurons, one could proceed in the same manner until reaching the target.

The previous concept is implemented by using different *scales* (see Fig. 5.6) to represent the whole space. Scale in this sense is a representation of space by a different amount of neurons. In this way, the advantages of the scales with many neurons or the ones with few are combined.

Summarizing, if the scale is represented by many neurons, since their activations are independent, only certain places are represented by a particular neuron. That is, a neuron only covers a part of the whole space, whereas the group of neurons cover every place in the space. The more neurons there are, the more spiky the representation of their activation patterns are. If a scale is represented by few neurons, a neuron would contain information about a big part of the space. The activation patterns of those neurons in space would be like as smooth hill.

Note that, with this configuration, the representation of a place differs from a coordinate system. In this scenario, a place would be described by the different activation levels that all the neurons have in that position and in each one of the scales. For example, if there are three scales, and in the first scale there are two

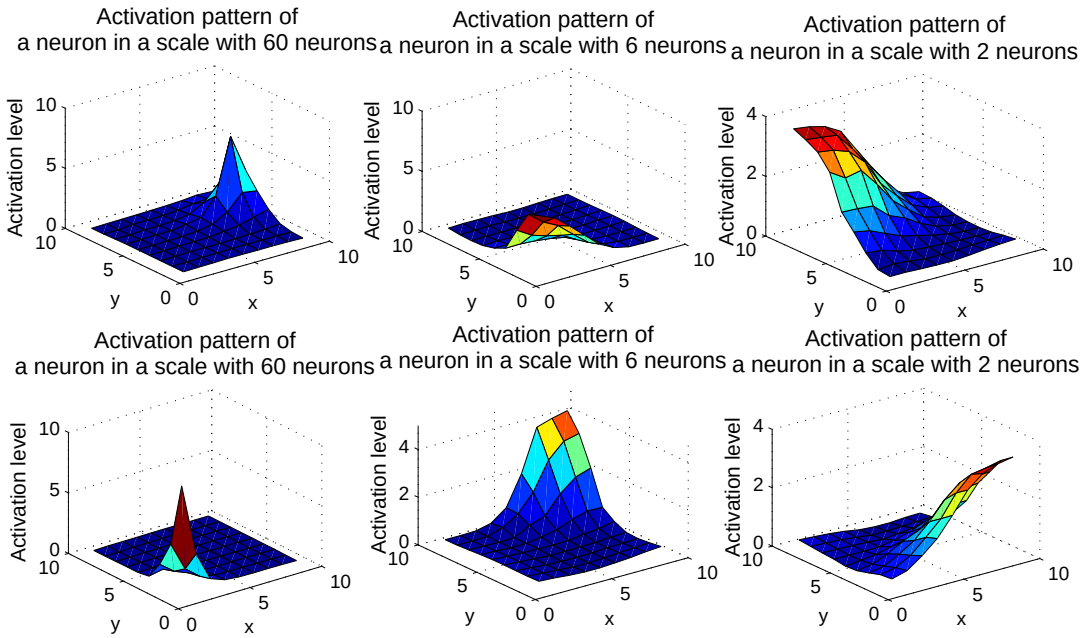


Figure 5.6: *Neural activations of neurons are illustrated in different scales. The leftmost figures, belong to a scale with sixty. The ones in the center belong to a scale with six neurons, and the rightmost figures to a scale with two neurons.*

neurons, in the second scale five neurons and in the third scale eight neurons, the activation level of any place p would be described with $2 + 5 + 8$ values, corresponding to the total number of neurons among all the scales.

Now, what is the algorithm that will find the scales? The activations of the neurons in a scale should be independent. Thus, it seems proper to use an ICA algorithm. The problem with ICA is that the activations do not cover the whole space in the grid, which is another desired requirement. In order to achieve this, a DSS algorithm is used. As said in Sec. 3.2.4, ICA is a particular case of DSS.

The desired independent components are obtained by applying an expansion of the activation patterns of each neuron, a smoothing and activation competition among the neurons in each scale. The first one, expansion, assures that all the space grid will be covered. The second one, smoothing, assures that not abrupt changes will occur. And last, the competition process will assure independence.

The data used in this algorithm is obtained the same way that in the previous approach (Sec. 5.1.1). The agent will move randomly around the grid and in each time step it will capture the sensory input data. It will receive the same sensory data in the same grid cell.

The raw sensory data will also be expanded, using as sensory data for the main algorithm all the first and second order terms. Before applying the DSS schema,

the data will be sphered. These two processes have the same purpose as described in Sec. 5.1.1.

The DSS schema is now applied over the preprocessed data $\mathbf{X}$. Each row of the matrix $\mathbf{X}$, in future references denoted as $\mathbf{X}(t)$, is related to the sensory input vector obtained in time step t. Each file, is understood as an input from a sensor (or due to the expansion the product of the data obtained in two of the sensors). The projection vectors $\mathbf{w_j}$, being the columns of the $\mathbf{W}$ matrix, are initialized to random vectors. There are as many $\mathbf{w}$ vectors as neurons desired to describe the full space. When the input sensory vector in a grid cell p is projected over the $\mathbf{w_j}$, the value obtained is the activation level of neuron j in the grid cell p.

According to Eq. (3.19), the estimated *source signal* $\mathbf{s_i}$ is obtained by projecting the preprocessed data $\mathbf{X}$ over the vectors $\mathbf{w_i}$. In this case, since it is not desired to give more preference (weight) to some neurons than to others, all the $\mathbf{w}$ vectors will be calculated in parallel and not one after another. Then the Eq. (3.19), would change into $\mathbf{S} = \mathbf{W}^T\mathbf{X}$.

After obtaining the estimated source signals, the $f()$ function is applied. It should include smoothing, expansion and competition:

- The first two can be accomplished by using a filter. If the filter is linear, applying it over the source signal $\mathbf{S}$ is equivalent to applying it to the data $\mathbf{X}$. This will save considerably the number of operations in the loop, so instead of applying it to $\mathbf{S}$ it will be applied to $\mathbf{X}$. This is equivalent because, calling $\mathbf{S}_f$ to the source signal filtered:

$$\mathbf{S}_f = \mathbf{SF} = \mathbf{W}^T\mathbf{XF} = \{\mathbf{X}_f = \mathbf{XF}\} = \mathbf{W}^T\mathbf{X}_f. \tag{5.3}$$

  Actually, since $\mathbf{X}$ is used twice inside the loop, it would be equivalent to apply the filter two times per each loop, one in the forward and the other backwards. Due to Eq. (3.21) and taking into account that $\mathbf{X}$ is sphered:

$$\begin{aligned} \mathbf{S}_f &= \mathbf{W}^T\mathbf{X}_f = (\mathbf{X}_f\mathbf{S}^T)^T\mathbf{X}_f \\ &= \mathbf{S}\mathbf{X}_f^T\mathbf{X}_f = \mathbf{S}(\mathbf{XF})^T\mathbf{XF} \\ &= \mathbf{SF}^T\mathbf{X}^T\mathbf{XF} = \mathbf{SF}^T\mathbf{F}. \end{aligned} \tag{5.4}$$

  The filter used was a *leaky integrator*. This provided enough expansion and smoothening needed. The spreading was done in time (steps). This is related to expansion in position (grid space) due to the configuration of the samples. In order to avoid errors, the previous values of samples (the original sample, a zero) and the future samples (queue of the leaky integrator) should be taken into account. The formula of the leaky integrator used was:

$$Y(i,t) = \gamma Y(i,t-1) + (1-\gamma)X(i,t), \tag{5.5}$$

where $Y(i,t)$ is the output of the leaky integrator in time t and sensory input i, and γ is the parameter of the leaky integrator with $\gamma \in [0,1]$. The bigger the constant γ, the more the previous samples affect over the next ones.

- Because of the nature of the data, there are certain values that are negative. The activation patterns are desired to be greater than or equal to zero in every moment. Due to that, before applying the competition process between activation of neurons, all the negative values are removed. For that, the next function is used:

$$S(j,i) = \ln(1 + e^{S(j,i)}). \tag{5.6}$$

  As it can be appreciated, when $S(j,i)$ is big enough, the function works as the identity function, while if the value of $S(j,i)$ is small, it is compressed to zero value. The image of the function is greater than zero for all the domain.

- The competition process consists of assuring that certain areas are not covered at the same time by different neurons and that other areas that are almost uncovered, start having the presence of activation levels of at least one neuron. Competition also happens in the cortex (see Sec. 2.2.1) and it is one of the keys for obtaining sparse features.

  Taking that into account, the small activation levels were multiplied by a bigger value than the big ones. A continuous inverse function was used, such that it multiplied the biggest value by an unitary factor and the smallest by a factor of two.

  After that, the competition of the activation levels between neurons was computed. Always the neuron with more activation level in a certain sample was favoured with respect the others. The function used for that was:

$$S(j,i) = \frac{S(j,i)^2}{\sum_{k=1}^{N} S(k,i)}, \tag{5.7}$$

  where $S(j,i)$ is the activation level of neuron j in sample i and N is the total number of neurons.

After the $f()$ function only remains to apply the Eqs. (steps) (3.21) and (3.22) for finalizing the loop. The loop should be executed until convergence of the W vectors. The complete algorithm is described in Alg. 3.

Applying this algorithm over the same expanded data vector $\mathbf{X}$, for different number of neurons, different scales are obtained.

Due to the nature of the algorithm, the spreading of the sources, the competition process of the activation and the symmetric decorrelation of the $\mathbf{W}$ vectors,

Algorithm 3 Sensory abstractions: scale calculation

Arbitrarily initialize $w_j \forall j$
Symmetric decorrelation of the w_j ($\|w_j\| = 1$, $w_j w_i = 0$, $\forall i, j \;\; j \neq i$)
Expansion with 0's of the sphered data signal X
$Y(t) \leftarrow cY(t-1) + (1-c)X(t)$ Leaky integrator: spreading the signal
repeat
    $s_j \leftarrow w_j^T Y \; \forall j$ Estimate sources
    $s_j \leftarrow f(s_j)$ Shrinking function (in this case $f(s_j) = ln(1 + e^{s_j})$)
    Competition process between sources
    $w_j \leftarrow Y s_j^T$ Re-estimation of the sources
    Symmetric decorrelation of the w_j ($\|w_j\| = 1$, $w_j w_i = 0$, $\forall i, j \;\; j \neq i$)
until All w_j have converged
$s_j \leftarrow w_j^T Y \; \forall j$ Estimate sources
$s_j \leftarrow f(s_j)$ Shrinking function (in this case $f(s_j) = ln(1 + e^{s_j})$)
Competition process between sources
$w_j \leftarrow Y s_j^T$ Re-estimation of the sources

which are also normalized, the activation patterns of the neurons are similar to each other and moreover they are evenly distributed around the grid space. Nevertheless, due to the random nature of the algorithm, not all the places were visited the same number of times, which caused a bit of noise when calculating the activations in the grid space. For correcting this undesired situation, the neural activations at each position were normalized scale-wise. Since the brain can decide how to represent things, normalizing the activations just makes posterior steps easier, but it does not change the essence of the abstract representation: it only takes noise away.

If now, for all the scales, the distance function is calculated like in Eq. (5.2), the next equation is obtained.

$$\begin{aligned} d_p(q)_{sfa} &= \sqrt{\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q) - S_k(j,p))^2} \\ &= \sqrt{\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q)^2 - 2S_k(j,p)S_k(j,q) + S_k(j,p)^2)} \\ &= \sqrt{\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q)^2 + S_k(j,p)^2) - 2\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,p)S_k(j,q))}, \end{aligned} \tag{5.8}$$

where M is the number of scales and N_k is the number of neurons in scale k. Because of the reasons stated above about the normalization of neural activations for each scale, the $\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q)^2)$ for all q is the same, in particular, equal to the number of scales, and so, those terms could be ignored. Redefining the new

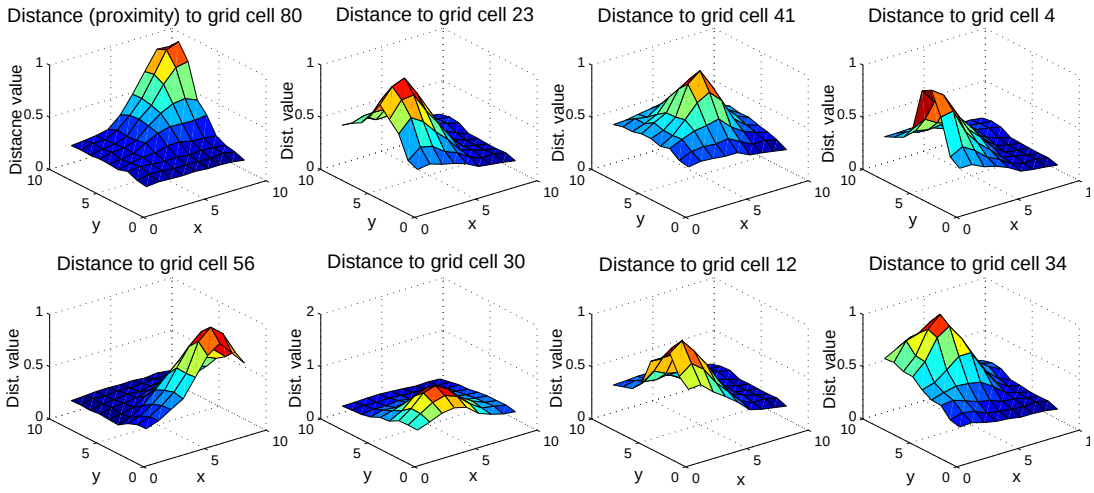


Figure 5.7: *Distance function to a eight different grid cells, represented in 3D.*

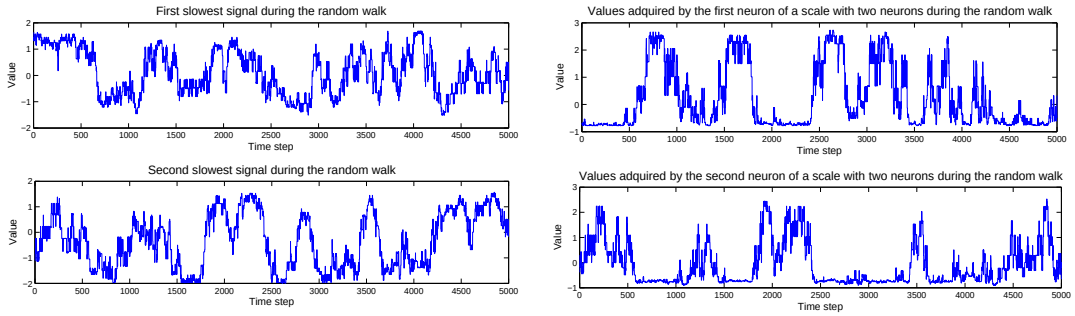
distance function as the square of the one obtained in the previous section and ignoring the constant terms, it gives:

$$\begin{aligned} d_p(q) = d_p(q)^2_{sfa} &= \sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q)^2 + S_k(j,p)^2) - 2\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,p)S_k(j,q)) \\ &\equiv -2\sum_{k=0}^{M}\sum_{j=0}^{N_k}(S_k(j,q)S_k(j,p)) \\ &= -2S(q)^T S(p), \end{aligned} \tag{5.9}$$

where $S(q)$, called activation vector of q, is a vector with dimensionality $\sum_{k=0}^{M} N_k$, that contains the values of the activation levels that the neurons acquire in the scales at the grid position q.

Leaving the square root away, it can be seen that the new concept of distance can be defined by just the projection of the activation vector in the target location p, over the activation vector of the current location q. This is just a linear operation, fairly easier than computing the squares of a difference and the square root of their sum, as it happened in the previous approach. The features found in this case are non-linear, but the operations needed for finding the distance are linear.

Note that now the distance function is not a function with a global minimum (see Fig. 5.7), but on the contrary a function with a global maximum. In both cases, the global minimum and maximum, happen when the current position equals the targeted one, that is, $q = p$. In this case the distance function could be understood as a proximity function.



(a) Slow features

(b) Slow and sparse features

Figure 5.8: (a) *The value that the first and the second slowest features (signals) take during the first 5000 steps of the random walk when only slowness principle is used.* (b) *The value that the two neurons (in a scale with two neurons) take during the first 5000 steps of the random walk when both slowness and sparseness principles are used.*

In this approach the abstractions found are slow and sparse, whereas in the previous approach the abstractions are just slow (see Fig. 5.8). There are multiple ways of obtaining slow or slow and sparse invariant features, which can be useful as sensory abstractions. In fact the cortex uses unsupervised learning, and there are numerous algorithms in this kind of learning. For example, in [Fran 07] they used slow feature analysis in a hierarchical way, in order to find the place, head and spatial-view cells using simulated movements of a rat.

5.2 From sensory abstraction to motor commands

The aim of this part is for the agent to learn to move on its own. Using the previously learnt sensory abstractions, the objective is to find the motor primitives that will allow the agent to move willingly. There are three simple motor primitives, which will be denoted as a, b and c. The values of these three variables, which vary within a certain interval, will control the amount and direction of movement. Then, the goal is to find the appropriate values for a, b and c.

The way the motor system of the agent works is based on the way of walking of any legged agent, such as a human. In order to be able to walk, a human must first move a foot up, the forward and finally down. Repeating, the same with the other foot. This process allows a human to walk forward. Having that image in mind and simplifying the process, one could observe that it is similar to the motion of a fixed point on a wheel rolling around a surface.

Making the motion discrete one could describe the motion of the point as a periodic sequence of $up, forward, down, backward$ movements. With this, it is

possible to move along a line, say north-south direction. Combining this movement with another, in west-east direction, a motion over a 2D plane is possible. That is the motor system the agent has, although the agent ignores the working principles of its motor primitives. The three motor parameters represent the different directions of movement, a being the movement in the north-south direction, b west-east one and c up-down. Random walk in this a-b-c corresponds to random walk in x-y space on the grid. Even with this mapping, the results obtained in Sec. 5.1 still hold.

Mathematically, the motor system works as follows: in each time step t the agent moves an amount $(c(t)/2 + c(t-1)/2)(a(t-1) - a(t)) = c^*(t)\Delta a(t)$ in its north-south relative direction and an amount equal to $(c(t)/2 + c(t-1)/2)(b(t-1) - b(t)) = c^*(t)\Delta b(t)$ in its west-east relative direction, north being the direction the head is pointing to. $c(t)$ denotes the value that the motor primitive c acquires at the time step t. The same applies to $b(t)$ and $a(t)$. $a(t-1)$, $b(t-1)$ and $c(t-1)$ make reference to the value that the motor primitives a, b and c had in the previous time step. The values that a, b and c have are limited to a certain intervals $[a_{min}, a_{max}]$, $[b_{min}, b_{max}]$ and $[c_{min}, c_{max}]$. Moreover, $\Delta a(t)$, $\Delta b(t)$ and $\Delta c(t)$ are also limited. Since the ranges of the variables are limited, the agent should learn how to cycle the activation of the variables in order to move longer distances towards a specific target in a direct way.

Even if the agent has been able to find the 'distance' to the target, the agent still ignores which values it should give to the simple motor primitives in order to move itself towards the desired target. That is why, at first, the agent would give random values (uniformly distributed) to those primitives, which will cause random movements. The data collected would be the distance value to the target $d_p(q)$ as well as the value given to the motor primitives a, b and c. The result of the data collected is a multi-dimensional temporal signal, which contains the 'distance' value in one of the components, and the values given to the motor primitives in the other dimensions.

In order to learn the optimal movements, it would be desired to find a mapping between the high and low-level abstractions in the motor part. The high-level abstraction in this case would correspond to 'get close' or 'get further' from a point, which can be measured with the 'distance function', whereas the low-level abstractions would be represented by the motor primitives. In mathematical terms this will correspond to finding the function $d_p(q) \rightarrow [a_{min}, a_{max}] \times [b_{min}, b_{max}] \times [c_{min}, c_{max}] \quad \forall p, q$.

Considering the first component of the signal (distance values, d_p) and the other three (motor primitives, a, b and c) it is not hard to observe that there is not a fixed relationship between them. The reasons are that different values assigned to motor primitives can produce the same result (the agent can arrive to the same position from different ones), and that identical values in motor primitives can cause very different results (depending on their previous values and/or the actual

grid cell). That is, the previous function is neither injective nor surjective. One has to think that the distance values depend on the grid position relative to the target, while the motor primitives indicate only the movement.

Nevertheless, there is a certain relationship between the differential distance value, $\Delta d_p(t) = d_p(g(t)) - d_p(g(t-1))$ and the movement made, where $g(t)$ is a function that returns the particular grid cell where the agent is located at time t. Again, it is not a fixed relationship, because of similar reasons. Going towards a target using the same movement, that is, the same Δa, Δb and c^*, may cause opposite results if the movement was taken in different grid cells: one movement could imply getting closer, positive Δd_p, and the other movement getting further, negative Δd_p. That is, on average the correlation between $\Delta d_p(t)$ and the movement made is null. Because of that, another abstraction is needed to link the two.

One could use the slow features obtained as the high-level sensory abstraction in Sec. 5.1.1, that is, the features which made reference to coordinates. Thus, knowing the coordinates (or activation values of the neurons) of the target and the actual position, as well as the distance function, one could analyse how good or bad the movement made was, and moreover, a fixed map between the three abstractions could be found. The problem is that, knowing that the coordinates values the agent should use are the target position and the actual position (even if those values are the only data the agent has at any moment) may signify to make a huge assumption. That is why, this approach will not be used.

A new approach, can be found by analysing, in more detail, the relationship between the differential distance value and the movement made. Before, it was discussed that, in general, it is not a fixed relationship. Nevertheless, locally, which translates in a small time frame on the random walk, a fixed relationship can be seen. This happens because, in a small time interval, the actual position, and with that, relative position with respect the target, does not change much. Then it is possible to find temporal correlations between the signals $\mathbf{\Delta d_p}$ and the motor primitives including $\mathbf{\Delta a}$, $\mathbf{\Delta b}$ and $\mathbf{\Delta c}$. This is the approach that will be used and developed.

In order to do find temporal correlations in the signals, consider the data matrix $\mathbf{D}$, which is a $n \times k$ dimensional matrix, where n is the number of data signals and k is the temporal length of the data signals. As data signals, there are all the values that the motor primitives a, b and c acquired in time (the samples) as well as their differential values Δa, Δb and Δc and all the second order terms that could be obtained from their combinations. In this example, second order terms are enough, since the movement is described by $c^*(t)\Delta a(t)$ and $c^*(t)\Delta b(t)$, which are second order terms. Nevertheless, one could also use higher order terms in order to add uncertainty, but in order to find the solution faster, only first and second order terms are used here. Each column of this data matrix will be called expanded data vector $v(t)$.

After the data expansion, any residual correlation between the signals in $\mathbf{D}$ and the $\mathbf{\Delta d_p}$ signal is removed, $\mathbf{\Delta d_p^* = \Delta d_p - MD}$, so that in later steps the temporal correlation will not be masked by the residual correlation. The matrix $\mathbf{M}$ is the filter obtained as a result of minimizing the square error of $E\{(\mathbf{\Delta d_p - MD})^2\}$. In this case, the filter $\mathbf{M}$ has all its elements almost equal to zero, as expected, since over the whole data, the correlation between $\mathbf{\Delta d_p}$ and the rest of the signals is null.

As a measure of the correlation or grade of dependence between the data signals and $\mathbf{\Delta d_p^*}$, each of the data signals is multiplied by $\mathbf{\Delta d_p^*}$. The reason of this multiplication lays in the formula of the correlation. For example, taking the *Pearson correlation coefficient*, the *correlation coefficient* between two signals $\mathbf{X}$ and $\mathbf{Y}$ is defined as their covariance divided by the product of their variances, $\frac{\text{cov}(\mathbf{X},\mathbf{Y})}{\sigma_X \sigma_Y}$, where the covariance is defined as $\text{cov}(\mathbf{X},\mathbf{Y}) = E\{(\mathbf{X} - \mu_\mathbf{X})(\mathbf{Y} - \mu_\mathbf{Y})\}$, μ_i representing the mean of the variable i. Before the multiplication takes place, both the data signals and the $\mathbf{\Delta d_p^*}$ signal are sphered since it may simplify posterior steps. Among its advantages, it makes the mean of the signals zero, which reduces the formula of the covariance to $\text{cov}(\mathbf{X},\mathbf{Y}) = E\{\mathbf{XY}\}$.

Once the multiplications are computed, the next step is to filter the data using a low pass filter, since the objective is to find temporal correlations in small time frames. In any given time frame, correlated variables will present a non-zero bias (value in frequency zero when the signal is transformed into a frequency domain) while uncorrelated variables will present a zero value. If PCA is applied now, the resulting projection vectors, $\mathbf{W}$, would be ordered in grade of similarity between the data signal and the distance function. Note that filtering together with PCA is equivalent to SFA as discussed in Sec. 5.1.1.

The filtering implementation is crucial since one should determine the cutting frequency without too much error, or the results obtained would not be too good. Of course, for avoiding this, many filters could be implemented and used in posterior steps until analysing which one is the one that in future steps gives more accurate results. This should be the approach taken for avoiding adopting unnecessary assumptions. The agent should be able to find the solution on its own.

Analysing the results after PCA, one can appreciate that the projection vectors, which correspond to those that have strongest similarities with $\Delta d_p(t)$, are the ones related to $c(t)\Delta a(t)$, $c(t-1)\Delta a(t)$, $c(t)\Delta b(t)$ and $c(t-1)\Delta b(t)$. That is, without giving any information about how the movement was calculated, the agent is able to figure out the principal signals among all the possible combinations analysed. Those values will be referred as *state* or *state variables*.

Once the state variables have been calculated, and knowing the values that the motor primitives acquire, an estimator of the increment of the distance, $\Delta dist = \Delta d_p(t)$, covered by each movement was implemented. The main aim is not only to determine which of the implemented filters is the best, if any, but if good

predictions could be made from the states obtained. If the agent would be able to predict how good a movement is, it will allow to modulate the values of the motor primitives in posterior steps. For that reason the following algorithm (Alg. 4) was implemented.

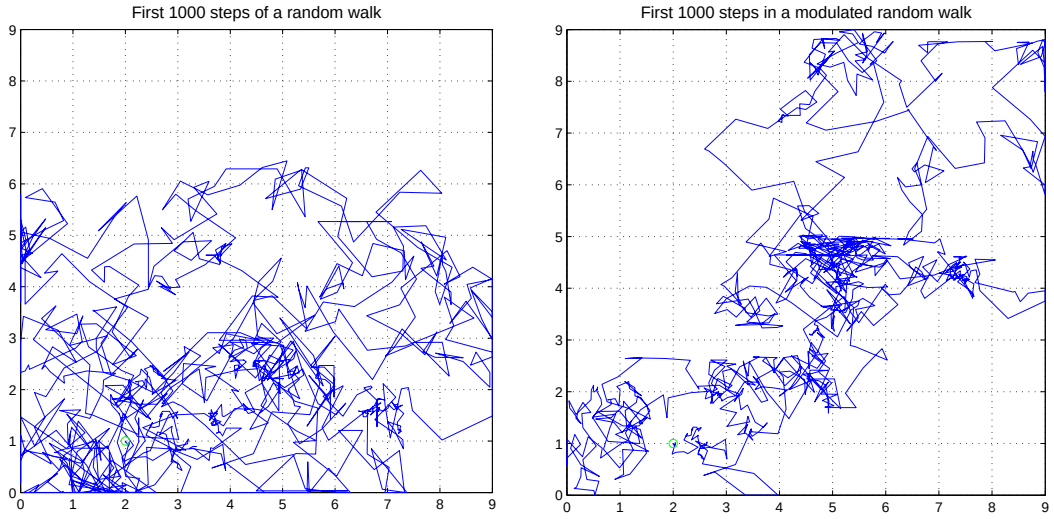
Algorithm 4 Motor abstraction: quality of the a, b, c motor parameters

1: Set time t to zero
2: Initialize to zero $a(t)$, $b(t)$ and $c(t)$
3: Initialize state variables $S(t)$ to zero
4: **repeat**
5: $t \leftarrow t + 1$
6: Randomly select $\Delta a(t)$, $\Delta b(t)$ and $\Delta c(t)$
7: Calculate $a(t)$, $b(t)$ and $c(t)$
8: Calculate the expanded data vector v and whiten it
9: $m_h(t) \leftarrow W\tilde{v}^T$ Calculate hypothetical movement v_1
10: $S(t) \leftarrow S(t-1)$ Assume the state does not change
11: $d_e(t) \leftarrow m_h(t)S(t)$ Estimate the $\Delta dist$ covered by $m_h(t)$
12: Make movement and observe the real $\Delta dist$, $d_r(t)$
13: $S(t) \leftarrow filter(m_h(t)d_r(t))$ Update the $S(t)$ value
14: **until** All the desired samples have been taken

The assumption that the state does not change between the actual iteration and the previous one is made because of lack of data. Knowing the movement and the state variables, namely correlation values between the movement and the $\Delta dist$, an estimate about the $\Delta dist$ can be made by multiplying the movement and the state variables. In fact, this structure allows to guess the missing value by using the other two. Nevertheless, when computing the state variables (correlation), some filtering is needed, since the multiplication of the variables alone is a very crude estimate of the correlation. In the filter implemented, the state variables obtained in previous time steps $t-1, ..., t-n$ are considered, with n small, since it is only within a small time frame where the situation (state) does not change and the correlation between the $\Delta dist$ and the movement would be non zero.

By running several simulations of this algorithm, it was found that, in the best situation, the $\Delta dist$ estimate can decide if the movement was good (positive value) or bad (negative value) with about 90-93% accuracy. Curiously, this accuracy was the same that could be obtained by using the SFA coordinates that were mentioned above. That is, without any extra information, the agent was able to figure out a solution with the same performance that could be obtained with the additional information.

Having found the best filters for obtaining a good accuracy in the $\Delta dist$ estimation, the next step is proceeding to modulate the $\Delta a(t)$, $\Delta b(t)$ and $\Delta c(t)$ values. One possible modulation would be to multiply the Δ values of the motor primitives by α when the $\Delta dist$ estimate is positive and by $\beta > 0$ when it is negative, with $\alpha > \beta$, or any other kind of function dependent of the $\Delta dist$ estimate.



(a) (b)

Figure 5.9: (a) *Random walk on the grid.* (b) *Modulated random walk, with α=2 and β=0.5. Arrival to a particular target takes approximately one fourth of the time it takes with the random walk.*

As said before, a positive $\Delta dist$ means a good movement, or movement towards the target, while a negative $\Delta dist$ means a bad movement, or movement further from the target. It makes sense, trying to take bigger steps when estimating that the step to take is going to good and small ones when estimating that the step is not going to be good. The reason of not using zero (no movement) or negative value (movement in the opposite direction) for β, resides in the nature of the estimation. The estimate could be wrong and the agent could get stuck. Think, for example, that in a corner of the grid, the agent thinks that the estimates that go outside the grid are good, while the ones that go inside the grid are bad. If β were zero or negative and with α positive, it will only try to move outside the boundaries of the grid, in other words, through walls, which is highly improbable, so the agent will get stuck. On the other hand, if $\beta > 0$, after a certain number of steps, the agent may be able to get outside the problematic situation. In Fig. 5.9, one can observe the difference between random walk with and without modulation.

Modulation is not the last step of the motor part. The modulation biases the movement towards the target, so that the agent moves, on average, towards it. Nevertheless, the movement has still a high random component, which makes the movement slow and not too efficient. That is why, after the modulation, new simulations were run and new data was taken. The goal is to obtain a function that estimates the correct value of $\Delta a(t)$, $\Delta b(t)$ and $\Delta c(t)$ based on the previous values of the motor primitives ($a(t-1)$, $b(t-1)$ and $c(t-1)$) and the state variables. A mapping that used first and second order terms of the previous

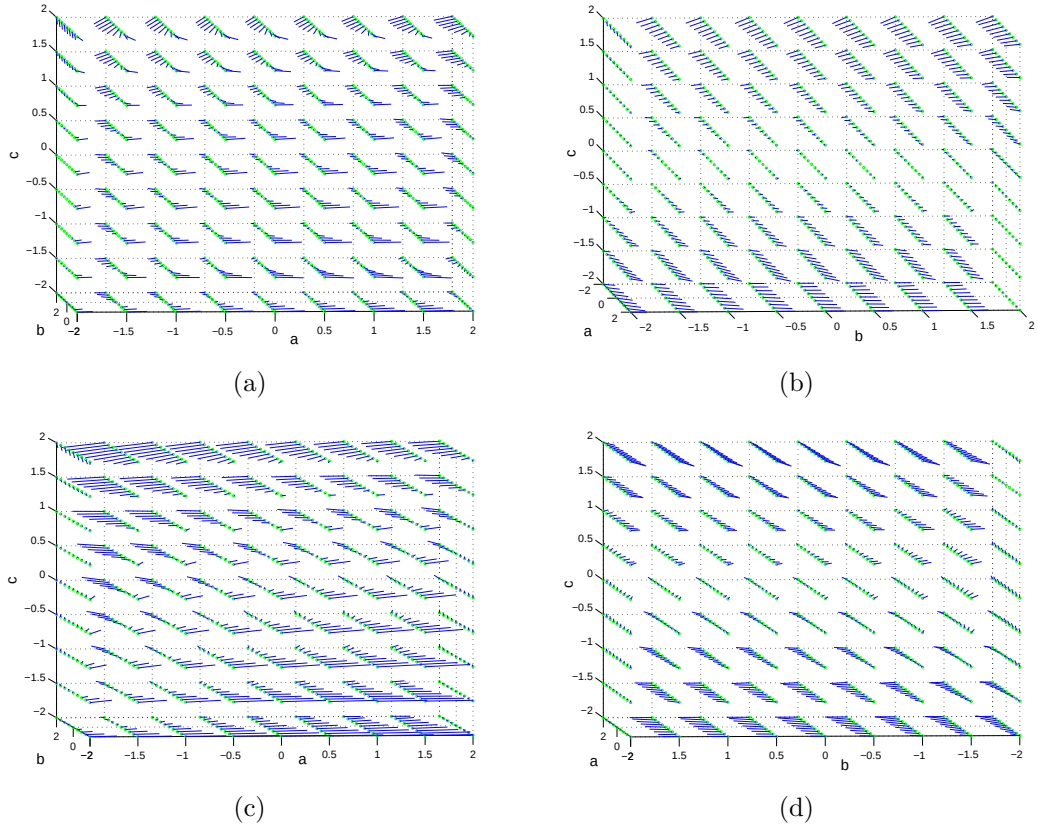


Figure 5.10: (a) *View of the a and c motor primitives in a certain state. The vectors indicate towards where the values of a and c should proceed.* (b) *View of the b and c motor primitives in the same state than in Fig. a). The vectors indicate towards where the values of b and c should proceed.* (c) and (d) *Idem than Fig. a) and b) respectively for a different state.*

values of the motor primitives and the state values for calculating the Δ-values was computed.

As it can be appreciated in Fig. 5.10, there is rotation in the a and b motor primitives depending on the state and the value of the c motor primitive. Mathematically, this can be seen in the weight that the different first and second order terms of the previous values of the motor primitives and the state values have over the total value for the Δ. In the case of a and b motor primitives, the main contribution is made by the second order terms that are the result of the multiplication of the states and the c motor primitive. The agent is able to determine the direction of the rotation according to the state of the environment that previously found. In the c motor primitive there is no visible rotation. This is expected since the prediction of the $\Delta dist$ is insensitive in changes of c when Δa and Δb are null, which happens in the limits.

Once the solution is found, the real values given to $\Delta a(t)^*$, $\Delta b(t)^*$ and $\Delta c(t)^*$

are those calculated from the previous values of the motor primitives and the state, plus added noise, that is, $\Delta\gamma(t)^* = \Delta\gamma(t) + n(t)$, where $\gamma \in \{a, b, c\}$. The reason to the added noise (randomness) is that without it, realizing that the situation has changed (for example a new target has been decided) may be impossible. It is believed that, for a biological being, learning may be impossible without that element of randomness [Tume 07]. Playing darts, kicking a ball or making a signature, will produce more or less the same movements, but they will never be identical due to that randomness in the movements.

5.3 A more complex task

The previous abstractions found can be the basis for more complex tasks. One could try to find higher-level complexity abstractions using the cognitive architecture, that is unsupervised learning, but in principle it is not possible to execute every possible task by using unsupervised learning exclusively. The representations (abstractions) found with the cognitive architecture can be refined by other learning techniques, or other learning algorithms could be used for learning more complex tasks. Remember that not everything is done by the cerebral cortex, but rather by the cortex in cooperation with other areas. Below, there is an example of how the agent can learn to serve clients, in a simplified manner, by using reinforcement learning (which appears to be implemented somehow by the basal ganglia) and having as a basis the previously learnt sensorimotor abstractions.

Thus, up till now, the agent has learnt to move form one place to another according to its desire. One possible following step is to learn how to serve customers. Serving will consist of going to a place (table) taking an order and heading to another place (bar). After that, the agent will return to the table. This sequence of actions will be the procedure. Having or not an order is indicated by sensors. In brief, the new task consists of moving between places following a particular order. Knowing how to move from a place to another was learnt in the previous sections of this chapter, which simplifies greatly the new problem.

The algorithm for learning the new task can be easily implemented using temporal difference learning. Using the SARSA algorithm, the states could be identified with the sensory signal that indicates if there is an order or not, for example, if the tray the agent has, is heavy or not according to a certain threshold. If only the positions of the bar and the table are given, the actions, in a first and simplistic approach, could be resumed as getting close to one or the other position. Receiving positive reward when a task is performed adequately, the agent can easily learn the sequence of actions. Each episode could consist of serving a certain number of clients.

This is a simplistic example of how to complicate the task carried out by the agent using as basis the previous learnt abstractions. The example can be extended by using objects that block the agent path, by making the agent to pick

things, even making the agent to recognize the bar and the tables, as well as the tables that have unattended clients and so on. Nevertheless the main aim of this work was to show a hierarchy of the cognitive architecture and show how it is possible to learn useful sensory abstractions from the sensory input data that will allow to understand the environment and accomplish certain tasks on it.

Chapter 6

Discussion

Up until now, many artificial agents have been used in high precision tasks, being completely programmed for that purpose. Nevertheless, there are new approaches in the use of agents, as assistants in a train crash, for example. In those unknown environments it is impossible to pre-program the agents. That is why, solutions like the one presented here are needed. The approach tries to combine the efforts of both neuroscientists and engineers towards the goal of creating intelligent and autonomous agents.

For autonomous agents it is fundamental to be able to acquire meaningful data that will allow them to act at will in any environment. In order to do that, it is important to explore and extract relevant data from the environment. Following that purpose, in this work, a cognitive architecture, which extracts invariant representations (abstractions) from the raw sensory input data, and uses the representations with the purpose of training new motor skills, have been presented.

The goal of this work was to show that it is possible for an agent to acquire new motor skills in a novel environment using just the sensory input data perceived, without any kind of supervision required. With that purpose, a biologically inspired solution, calle a cognitive architecture was implemented.

The cognitive architecture model implemented is based on the processes taking place in the brain, in particular, on the sensorimotor system found in the cerebral cortex. The sensorimotor system in the cortex is based on a hierarchy of abstractions, where each of the hierarchical levels presents a different level of complexity. The motor representations in the architecture, which are goal-oriented, try to change (manipulate) the sensory abstractions. The existent link between motor and sensory abstractions allows the agent to find the mapping between the motor actions it is taking and the changes it is able to produce in the surrounding environment.

The abstractions are obtained using the slowness principle. The slowness principle allows an agent to autonomously learn invariant features within the raw

data. The features found prepare the agent for future, and possibly more complex, learning tasks. The cognitive architecture is not intended to replace other learning techniques, but to complement them.

The cortical model was not implemented fully, but a fraction, in particular, one hierarchical level of the cognitive architecture. In the example the agent learnt concepts such as *place* and *distance*. The results obtained also show how the agent learnt to move on its own towards any place desired in its environment. Before finding the abstractions, the agent knew nothing about the environment, nor its motor functions. Thus, as it has been proved, it is possible to use the cognitive architecture presented here in order to learn new things, how to perform different tasks, without previous knowledge and only based on the signals received as sensory input.

By using the same cognitive structures and different input data it may be possible to obtain different kinds of abstractions, as it happens in the brain. That would allow to use the same modules in multiple scenarios, being able to obtain different abstractions from different sensory input data types. Thus, the same algorithms could be used for learning different tasks. Moreover, in the example implemented, the emotions module described in the cognitive architecture was not used, but in other scenarios it could be used for obtaining other abstractions.

The cognitive architecture could be used as a brick, allowing to build up a more complex architecture. In principle, proceeding that way, it may be possible to find higher-level sensory and motor abstractions using more basic ones, like it happens in the cortex. The representations found not only serve as basis to other learning methods, but they can also be refined by other techniques.

Steps towards the future would extend the functionality of the now implemented architecture, by adding more levels of complexity and learning new motor skills. Other learning methods could be used with the purpose of refining the abstractions already learnt or using the ones found as a basis to find more complex abstractions.

Bibliography

[Alic 08] B. Alicea. "Performance augmentation in hybrid bionic systems: techniques and experiment". *arXiv:0810.4629v3 [q-bio.NC]*, 2008.

[Alpa 04] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[Andr 01] P. F. Andreas K. Engel and W. Singer. "Dynamic predictions: Oscillations and synchrony in top–down processing". *Nature Reviews Neuroscience*, Vol. 2, pp. 704 – 716, 2001.

[Arle] A. Arleo, M. Arnold, D. Cohen, E. D'Angelo, C. D. Zeeuw, R. Johansson, H. Jörntell, E. Ros, P. van der Smagt, S. Vijayakumar, and D. Wolpert. "SENSOPAC: SENSOrimotor structuring of Perception and Action for emerging Cognition". *EU Framework 6 IST Cognitive Systems*.

[Bech 00] A. Bechara, H. Damasio, and A. R. Damasio. "Emotion, Decision Making and the Orbitofrontal Cortex". *Oxford Journals - Cerebral Cortex*, Vol. 10, No. 3, pp. 295–307, Mar 2000.

[Beck 92] S. Becker and G. E. Hinton. "Self-organizing neural network that discovers surfaces in random-dot stereograms". *Nature*, Vol. 355, pp. 161 – 163, Jan 1992.

[Berk 05] P. Berkes and L. Wiskott. "Slow feature analysis yields a rich repertoire of complex cell properties". *Journal of Vision*, Vol. 5, pp. 579 – 602, Jul 2005.

[Butz 08] M. V. Butz, K. Reif, and O. Herbort. "Bridging the Gap: Learning Sensorimotor-Linked Population Codes for Planning and Motor Control". *Proceedings of the 2008 International Conference on Cognitive Systems*, Apr 2008.

[Chen 01] C. Chennubhotla and A. Jepson. "Sparse Coding in Practice". *SCTV*, 2001.

[Cohe 97] P. R. Cohen, M. S. Atkin, and T. Oates. "Neo: Learning Conceptual Knowledge by Sensorimotor Interaction with an Environment".

In: *Proceedings of the First International Conference on Autonomous Agents*, pp. 170–177, ACM, 1997.

[Dean 05] T. Dean. "A Computational Model of the Cerebral Cortex". In: *Proceedings of AAAI-05*, pp. 938 – 943, The MIT Press, 2005.

[Deco 05] G. Deco and E. T. Rolls. "Neurodynamics of biased competition and cooperation for attention: a model with spiking neurons". *Journal of neurophysiology*, Vol. 94, No. 1, pp. 295 – 313, Jul 2005.

[Doya 99] K. Doya. "What are the computations of the cerebellum, the basal ganglia and the cerebral cortex?". *Neural networks*, Vol. 12, pp. 961 – 1009, 1999.

[East 07] M. Eastaway, J. Steinbauer, A. Qian, and A. Dayal. "Blind Source Separation Via ICA: Signal Processing Applications". *The Connexions Project*, 2007.

[Edsi 04] A. Edsinger-Gonzales and J. Weber. "Domo: A Force Sensing Humanoid Robot for Manipulation Research". *International Journal of Humanoid Robotics*, 2004.

[Fell 91] D. J. Felleman and D. C. Van Essen. "Distributed Hierarchical Processing in the Primate Cerebral Cortex". *Oxford Journals - Cerebral Cortex*, Vol. 1, No. 1, Jan/Feb 1991.

[Fran 07] M. Franzius, H. Sprekeler, and L. Wiskott. "Slowness and Sparseness Lead to Place, Head-Direction, and Spatial-View Cells". 2007.

[Gema 06] S. Geman. "Invariance and selectivity in the ventral visual pathway". *Journal of Physiology*, Vol. 100, pp. 212 – 224, 2006.

[Gilb 07] C. D. Gilbert and M. Sigman. "Brain States: Top-Down Influences in Sensory Processing". *Neuron*, Vol. 54, pp. 677 – 696, 2007.

[Gold 04] M. D. Goldberg, L. Zhou, and W. Wolf. "Applications of Principal Component Analysis (PCA) to AIRS Data". *NOAA/NESDIS/Office of Research and Applications*, 2004.

[Gold 07] E. B. Goldstein. *Sensation and perception.* Wadsworth, 2007.

[Good 92] M. A. Goodale and A. D. Milner. "Separate visual pathways for perception and action". *Trends of Neuroscience*, Vol. 15, pp. 20 – 25, 1992.

[Grah 06] D. J. Graham, D. M. Chandler, and D. J. Field. "Can the theory of "whitening" explain the center-surround properties of retinal ganglion cell receptive fields?". *Vision Research*, Vol. 46, pp. 2901 – 2913, Sep 2006.

[Guyt 05] A. C. Guyton and J. E. Hall. *Textbook Of Medical Physiology 11th ed.* Saunders, 2005.

[Hate 98] J. H. van Hateren and D. L. Ruderman. "Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex". *The Royal Society*, Vol. 265, pp. 2315 – 2320, Aug 1998.

[Hube 62] D. H. Hubel and T. N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". *Physiology*, Vol. 160, pp. 106 – 154, 1962.

[Hube 65] D. H. Hubel and T. N. Wiesel. "Receptive fields and functional architecture in two non-striate visual areas (18 and 19) of the cat". *Neurophysiology*, Vol. 18, pp. 229 – 289, 1965.

[Hyva 00] A. Hyvärinen and E. Oja. "Independent Component Analysis: Algorithms and applications". *Neural Networks*, Vol. 13, pp. 411–430, 2000.

[Hyva 99] A. Hyvärinen. "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis". *Neural Networks*, Vol. 10, pp. 626–634, 1999.

[Iyen 91] R. N. Iyengar. "Application of principal component analysis to understand variability of rainfall". In: *The Proceeding of the Indian Academy of Sciences: Earth and Planetary Sciences*, pp. 105–126, 1991.

[Koho 95] T. Kohonen. *Self-Organizing Maps.* Springer, 1995.

[Koni 08] G. Konidaris and A. Barto. "Sensorimotor abstraction selection for efficient, autonomous robot skill acquisition". In: *Proceedings of the 7th IEEE International Conference on Development and Learning*, 2008.

[Koni 09] G. Konidaris and A. Barto. "Efficient Skill Learning using Abstraction Selection". *Twenty-First International Joint Conference on Artificial Intelligence*, Jun 2009.

[Kör 04] K. P. Körding, C. Kayser, W. Einhäuser, and P. König. "How are complex cell properties adapted to the statistics of natural stimuli?". *Journal of Neurophysiology*, Vol. 91, pp. 206 – 242, Jan 2004.

[Labi 04] K. Labib and V. R. Vemuri. "An Application of Principal Component Analysis to the Detection and Visualization of Computer Network Attacks". In: *The Proceeding of SAR*, 2004.

[Lege 10] R. Legenstein, N. Wilbert, and L. Wiskott. “Reinforcement Learning on Slow Features of High-Dimensional Input Streams”. *PLoS Computational Biology*, Vol. 6, Aug 2010.

[Mose 08] E. I. Moser, E. Kropff, and M.-B. Moser. “Place Cells, Grid Cells, and the Brain’s Spatial Representation System”. *Annual Review of Neuroscience*, 2008.

[Muga 07] J. Mugan and B. Kuipers. “Learning to predict the effects of actions: Synergy between rules and landmarks”. In: *Proceedings of the 6th (IEEE) International Conference on Development and Learning*, 2007.

[Olsh 04] B. A. Olshausen and D. J. Field. “Sparse coding of sensory inputs”. *Current Opinion in Neurobiology*, Vol. 14, pp. 481 – 487, 2004.

[Olss 06] L. Olsson, C. L. Nehaniv, and D. Polani. “From Unknown Sensors and Actuators to Actions Grounded in Sensorimotor Perceptions”. *Connection Science*, Vol. 18, No. 2, pp. 121 – 144, Jun 2006.

[Prov 05] R. Provost, B. J. Kuipers, and R. Miikkulainen. “Self-Organizing Distinctive-State Abstraction For Learning Robot Navigation”. *Artificial Intelligence Lab Technical Report AI-TR-05-319*, Jul 2005.

[Quir 05] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried. “Invariant visual representation by single neurons in the human brain”. *Nature*, Vol. 345, pp. 1102 – 1107, Jun 2005.

[Roa 09] S. Roa, G. M. Kruijff, and H. Jacobsson. “Curiosity-Driven Acquisition of Sensorimotor Concepts Using Memory-Based Active Learning”. *International Conference on Robotics and Biomimetics*, Feb 2009.

[Sare 05] J. Särelä and H. Valpola. “Denoising source separation”. *Journal of Machine Learning Research*, Vol. 6, pp. 233 – 272, 2005.

[Scar 05] D. Scaramuzza and M. Vetterli. “Sparse Codes for Natural Images”. Ecole Polytechnique Federal de Lausanne, EPFL, Lausanne, Switzerland, Oct 2005.

[Skip 07] J. I. Skipper, V. van Wassenhove, H. C. Nusbaum, and S. L. Small. “Hearing Lips and Seeing Voices: How Cortical Areas Supporting Speech Production Mediate Audiovisual Speech Perception”. *Oxford Journals - Cerebral Cortex*, Vol. 17, No. 10, pp. 2387 – 2399, 2007.

[Smag 97] P. van der Smagt. “Teaching a robot to see how it moves”. *Neural Network Perspectives on Cognition and Adaptive Robotics*, pp. 195 – 219, 1997.

[Spre 07] H. Sprekeler, C. Michaelis, and L. Wiskott. "Slowness: An Objective for Spike-Timing–Dependent Plasticity?". *PLoS Computational Biology*, Vol. 3, pp. 1136 – 1148, Jun 2007.

[Spre 08] H. Sprekeler and L. Wiskott. "Understanding Slow Feature Analysis: A Mathematical Framework". *Cognitive Sciences*, Vol. EPrint Archive (CogPrints) 6223, 2008.

[Spre 09] H. Sprekeler. *Slowness learning: mathematical approaches and synaptic mechanisms*. Dissertation, University of Berlin, 2009.

[Ston 05] J. Stone. "Independent Component Analysis". In: *Encyclopedia of Statistics in Behavioral Science*, pp. 907 – 912, 2005.

[Sutt 98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. The MIT Press, 1998.

[Tume 07] E. C. Tumer and M. S. Brainard. "Performance variability enables adaptive plasticity of 'crystallized' adult birdsong". *Nature*, Vol. 450, pp. 1240 – 1244, Dec 2007.

[Unge 82] L. G. Ungerleider and M. Mishkin. *Analysis of Visual Behavior*. The MIT Press, 1982.

[Valp 05] H. Valpola. "Development of representations, categories and concepts–a hypothesis". In: *Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 593 – 599, 2005.

[Valp 08] H. Valpola. "The Engine of Thought — a Bio-Inspired Mechanism for Distributed Selection of Useful Information". *Nokia Workshop on Machine Consciousness*, pp. 27–31, 2008.

[Wall 97] G. Wallis and E. T. Rolls. "Invariant face and object recognition in the visual system". *Progress in neurobiology*, Vol. 51, pp. 167 – 194, Feb 1997.

[Wisk 02] L. Wiskott and T. Sejnowski. "Slow Feature Analysis: Unsupervised Learning of Invariances". *Neural Computation*, Vol. 14, No. 4, pp. 715–770, 2002.

[Yli 07] A. Yli-Krekola and H. Valpola. "Computational model of co-operating covert attention and learning". *Fifth Nordic Neuroinformatics Workshop*, p. 34, 2007.

[Yu 06] Y. Yu and J. Yang. "A New Method of Image Feature Extraction and Denoising Based on Independent Component Analysis". In: *Proceedings of the 2006 IEEE - International Conference on Robotics and Biomimetics*, 2006.