Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Corso di Laurea in Fisica

Non-evolutive pattern recognition techniques. An aplication in medical image diagnostics.

Relatore:
Chiar.mo Prof.
Fabio Ortolani

Presentata da:
Matteo Monti

Correlatore:
Chiar.mo Prof.
Renato Campanini

Sessione II

Anno Accademico 2012-2013

بس الحقيقة المدهشة

إن الملايكه

و اللأبالسه

بيعملوا لنا لسه

ميت حساب

(Sayed Hegab, 1971)

Abstract

Lo studio dell'intelligenza artificiale si pone come obiettivo la risoluzione di una classe di problemi che richiedono processi cognitivi difficilmente codificabili in un algoritmo per essere risolti. Il riconoscimento visivo di forme e figure, l'interpretazione di suoni, i giochi a conoscenza incompleta, fanno capo alla capacità umana di interpretare input parziali come se fossero completi, e di agire di conseguenza.

Nel **primo capitolo** della presente tesi sarà costruito un semplice formalismo matematico per descrivere l'atto di compiere scelte. Il processo di "apprendimento" verrà descritto in termini della massimizzazione di una funzione di prestazione su di uno spazio di parametri per un ansatz di una funzione da uno spazio vettoriale ad un insieme finito e discreto di scelte, tramite un set di addestramento che descrive degli esempi di scelte corrette da riprodurre. Saranno analizzate, alla luce di questo formalismo, alcune delle più diffuse tecniche di artificial intelligence, e saranno evidenziate alcune problematiche derivanti dall'uso di queste tecniche.

Nel **secondo capitolo** lo stesso formalismo verrà applicato ad una ridefinizione meno intuitiva ma più funzionale di funzione di prestazione che permetterà, per un ansatz lineare, la formulazione esplicita di un set di equazioni nelle componenti del vettore nello spazio dei parametri che individua il massimo *assoluto* della funzione di prestazione. La soluzione di questo set di equazioni sarà trattata grazie al teorema delle contrazioni. Una naturale generalizzazione polinomiale verrà inoltre mostrata.

Nel **terzo capitolo** verranno studiati più nel dettaglio alcuni esempi a cui quanto ricavato nel secondo capitolo può essere applicato. Verrà introdotto il concetto di grado intrinseco di un problema. Verranno inoltre discusse alcuni accorgimenti prestazionali, quali l'eliminazione degli zeri, la precomputazione analitica, il fingerprinting e il riordino delle componenti per lo sviluppo parziale di prodotti scalari ad alta dimensionalità. Verranno infine introdotti i problemi a scelta unica, ossia quella classe di problemi per cui è possibile disporre di un set di addestramento solo per una scelta.

Nel **quarto capitolo** verrà discusso più in dettaglio un esempio di applicazione nel campo della diagnostica medica per immagini, in particolare verrà trattato il problema della *computer aided detection* per il rilevamento di microcalcificazioni nelle mammografie.

Contents

Chapter 1

Mathematical aspects of artificial intelligence

1.1 Introduction

A wide variety of cognitive processes can be described as the act of making choices. The interaction of an agent, be it an animal, a human being or a simulated chess player, with an environment that provides inputs and can affected by the agent's actions, is to a large extent representable by the set of the choices that agent makes during its existence.

In the real world, such interaction implicitly determines the fitness of a biological agent to its environment: fit agents are those able to live long enough to reproduce. The resemblance of the offspring to the parent, along with random mutations introduced by reproduction, determines the evolution of a set of agents towards the highest possible suitedness to their environment.

The high-level cognition human beings display is usually attributed to such a process. While there is plenty of evidence that evolution is the main factor in human intelligence, some distinction must be made to explain the subject of this thesis.

Instinct will be referred to as the factor determining the agent's choices *a priori* from its interaction with the environment. Instinct can be viewed as a simple answer to determined stimuli that the agent inherits from its parent. Experience does not significantly affect instinct during the agent's life span. Instinct may therefore be seen as an intrinsic factor that evolves thanks to the supremacy of the fittest agents.

Intelligence will be referred to as the factor that, first, determines those choices that can be affected even by a single experience, and, second, makes them change during the agent's lifespan, depending on that agent's prior set of interactions with the environment.

Note the difference between these paradigms: a human being does not need to reproduce for thousands of generations to learn the alphabet!

A wide variety of problems that human intelligence has evolved to deal with is very difficult to describe analytically. In particular, the ability to extract relevant information from a noisy background, and to interpolate partial inputs as if they were complete, is one generally not trivial to reproduce by means of an algorithm. This, however, is a limitation of the human mind: computers have grown more and more powerful and would have the computing capability to deal with many of those problems.

The factors described as leading to evolution, are in no way linked with the agent's biological nature. A common approach to the problem of describing such analytically complex algorithms is to create a virtual environment that provides inputs to a "black-box procedure", lets this procedure interact with the environment through its choices, and defines an evolutionary pressure in terms of the ability to solve a specific problem. This, along with the definition of a (nearly) continuous set of mutations that can be introduced at the time of reproduction, makes the system evolve towards an increasingly accurate solution for the problem.

The process just described is strongly linked to the definition of instinct given above. The evolutionary pressure is completely determined by the choices the agent makes during its existence, so the instinct is the only factor affected by evolution.

The subject of this work is the description of a simple way to build algorithms that reach their best (absolute) solution to a given problem without the need for a long and computationally intensive iterative procedure like evolution.

1.2 Mathematical paradigm

The research for the best solution to a problem that is analytically complicated to describe, can be described as the exploration of a parameters space over which a fitness function is empirically defined. The problem is reduced, then, to the search for the absolute maximum of the fitness function in that space. We will assume that the fitness function is defined in terms of a *training set*, that is, a set of "example correct choices" that will be provided for training purposes.

Note that all the following definitions are based on the assumption of a discrete choice problem, i.e. a problem that can be fully solved by a choice function.

Definition 1. *A* choice function *is a function from a vector space* $\mathbb{R}^N$ *(we will also refer to it as* environment space*) to a finite set of choices:*

$$f : \mathbb{R}^N \to S = \{\sigma_1, \sigma_2 \dots \sigma_s\} \tag{1.1}$$

The definition of the environment and the set of choices is general; some examples will be given in the next section. Note that the definition of a choice function can be easily extended on any function $p : \Upsilon \subset \mathbb{R}^N \to S'$ by adding a dump choice:

$$\begin{aligned} f &: (R)^N \to S = S' \cup \{d\} \\ f(\mathbf{x}) &= \begin{cases} p(\mathbf{x}), & \text{if } \mathbf{x} \in \Upsilon \\ d, & \text{if } \mathbf{x} \notin \Upsilon \end{cases} \end{aligned} \tag{1.2}$$

Since the space of all possible functions is obviously impossible to explore, an ansatz will be needed for a choice function to reduce the problem to the exploration of a finite dimensional space.

Definition 2. *An* ansatz choice function *is a choice function completely determined by a finite set of parameters, that is a vector in a K-dimensional* parameters space*:*

$$f = f_{\mathbf{k}}, \mathbf{k} \in \mathbb{R}^K \tag{1.3}$$

Definition 3. *A* training set *is a set*

$$T = \{t = (\mathbf{x}, \sigma), \mathbf{x} \in \mathbb{R}^N, \sigma \in S\} \tag{1.4}$$

of couples of environment vectors associated with "correct choices". From here on we will also refer to t_σ as the choice component for the training set element, and $t_\mathbf{x}$ as its environment vector component:

A fitness function will be now defined on the parameters space, based on the ability of the parametrized ansatz choice function to reproduce the choices for the elements in the training set.

Definition 4. *A* fitness function *for a parameters space $\mathbb{R}^K$ is a function*

$$\Phi_T : \mathbb{R}^K \to \mathbb{R} \tag{1.5}$$

depending on a training set.

Example 1. *The most simple definition for $\Phi_T(\mathbf{k})$ is given by the fraction of correct answers given by the function parametrized by $\mathbf{k}$:*

$$\Phi_T^{(r)}(\mathbf{k}) = \frac{|\{t \in T : f_\mathbf{k}(t_\mathbf{x}) = t_\sigma\}|}{|T|} \tag{1.6}$$

Where $|T|$ is the cardinality of T.

Example 2. *Given a positive error function*

$$\epsilon : S \times S \to \mathbb{R}^+, \epsilon(\sigma, \sigma) = 0 \forall \sigma \in S \tag{1.7}$$

assessing the error done by a function, Φ can be defined in terms of the average error:

$$\Phi_T^{(\epsilon)}(\mathbf{k}) = -\frac{\sum_{t \in T} \left(\epsilon(t_{\sigma_i}, f_\mathbf{k}(t_\mathbf{x}))\right)}{|T|} \tag{1.8}$$

That reduces to (1.6) with $\epsilon(\sigma, \tau) = 1 - \delta_{\sigma,\tau}$.

To sum up: a choice agent can be described as a function from an environment to a set of choices. Given a training set of couples of environment vectors and corresponding correct choices, and given an ansatz for the choice function in a finite number of parameters, one needs to define a fitness function over the parameters space based on the ability of the choice function parametrized to reproduce the training set. The fitness function therefore depends on the training set, and is usually defined as the fraction of correct choices made by the choice function, or the average error made.

Finding the best ansatz choice function is now explicitly expressed in terms of the fitness function: the best ansatz choice function is obviously the one parametrized by the absolute maximum of $\mathbf{k}$ on the parameters space. Since the fitness function is usually statistically defined in terms of the training set, it is impossible to give a simple explicit definition for it. Furthermore, since the choice function maps on a finite set, no notion of continuity nor derivability can be used to determine local maxima on the fitness function. However, the values of the fitness function can be computed at any point, so a finite difference gradient can be computed. We will briefly review the main fitness maximization techniques in chapter (?).

1.3 Examples of choice functions

A wide variety of problems can be described in terms of a choice function mapping an environment vector in a choice. We will hereby give some examples to show the generality of this description.

Application 1. Optical character recognition
An image representing a character from the alphabet is a set of pixels that can be codified in three components (red, green and blue, or alternatively hue, saturation and brightness). An image can therefore be codified in three matrices (one for each component) with values ranging in $[0, 1]$*, or simply in a vector* $\mathbf{I} \in [0, 1]^{3p}$*,* p *being the number of pixels in the image. The set of choices here is simply represented by the alphabet or, more generally, by the set of symbols represented.*
A training set for this problem can be easily generated by randomizing a symbol, representing it into an image, then adding noise.

Application 2. String recognition
By adding a dump choice to the previous example (a choice that is returned for any image that doesn't represent any symbol in the set), an OCR choice function can be used to scan an image made of many characters in different positions by associating with each pixel a vector of choices for sub-images of varying size (rescaled if necessary), centered on that pixel. This multi-variate scan will ideally result in a matrix of choices all composed by the dump choice, except for the pixels centered on each character.

Application 3. Sound fingerprinting
By sampling a sound and representing it as a vector of fixed size (one sample per component), or by operating a discrete Fourier transform over it (one frequency per component), it is possible to distinguish a sound within a set, that will form the set of choices.

Application 4. Spoken language recognition
By adding a dump choice to the previous example and scanning the sample in a way similar to Application 2, single sounds can build up to form words and sentences.

Application 5. Stock prevision
The values of a cluster of interconnected stocks can be sampled over time and grouped in an environment vector. A choice agent can be trained to predict the evolution of those stocks, the choices being the trend for the next time step of a specific stock (for example up, down, stable*). Note that this can be highly non trivial: since many factors can affect the evolution of stocks, different evolutions can be found in the training set for the same environment vectors. Since a choice function is singlevalued, the training set cannot be completely reproduced even by ideal choice functions.*

Application 6. Closed games (imitation)
In a closed game (like Poker) players have only a partial knowledge of the state of the game, and the best choice usually cannot be determined by means of exploration of a choice tree as in open games (like Chess). As long as the game status can be codified in an environment vector, and there is a finite number of moves that can be done at each turn, however, the game history of an human player can be recorded to form a training set for a choice agent.

Application 7. Closed games (tree exploration)
Since a choice function is singlevalued, the behavior of a choice agent is completely determined by the environment vector provided to it. Even in a closed game, the full tree of choices can be analytically explored while playing against a choice agent. Environment vectors and choices along the path to the best solution can be used as a training set for a new choice agent, that will be effective against the previous.

In the second part of the present work several applications of AI techniques to image analysis will be analyzed in greater depth.

1.4 Genetic algorithms

Among the AI techniques in current use are genetic algorithms. These algorithms mimic the process of natural selection to solve non-trivial optimization problems. At the beginning, a number of agents is randomly created (vectors in the parameters space are randomly chosen to represent such agents, given an ansatz to make that space finite-dimensional). The initial parameter values can be bonded if necessary. The process goes under several iterations of the following steps:

- Evaluation: each agent undergoes an evaluation process to test its suitedness to the environment (that is, its capability to solve the given problem). Mathematically, the fitness function is numerically evaluated on each parameters vector.
- Selection: a fraction of the less fit agents gets deleted. This step realizes the actual evolutionary pressure on the agents.
- Reproduction: fittest agents reproduce introducing random mutations to their genetic pool. Crossing over can also be simulated. The offspring of the fittest replaces the least fit in the population, and evolution takes place.

We will briefly focus on some mathematical aspects of reproduction and mutation, and will try to discern those aspects given by theory, and those given by the trainer's experience.

Definition 5. *Let* $\mathbf{k}$ *be a parameters vector for an ansatz choice function. The* asexual reproduction process *simply adds a white noise vector to* $\mathbf{k}$*:*

$$R(\mathbf{k}) = \mathbf{k} + \rho \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{pmatrix} \tag{1.9}$$

Where $w_1, \cdots, w_K$ *are random real values satisfying the relation* $|(w_1 \cdots w_K)| = 1$*. We will refer to* ρ *as* evolutionary step.

This process is designed to mimic random mutation introduced by reproduction in the set of genes of the offspring. Since the parameters vector uniquely determines the agent's response to an environment vector, such resemblance with the evolution of biological systems should imply a form of continuity on the parameters space for ansatz choice functions, so that agents can *behave* in a way more and more like to their parents as the evolutionary step tends to vanish.

This would be mathematically expressed by requiring that ansatz choice functions parametrized by close parameters vectors map the same environment vector to close choices. Since choices form a discrete set, no notion of continuity can be defined on it.

We will try to show that a weaker notion of continuity can however be defined by asking that, for every environment vector, the choices corresponding to each parameters vector in a bonded open set in the parameters space be grouped in a finite number of maximal choice areas. Let's see this in more detail.

Recall that every environment vector maps to a choice through an ansatz choice function parametrized by a vector in a parameters space $\mathbb{R}^K$.

Definition 6. *Be* $\mathbf{x}$ *an environment vector,* $\mathbb{R}^K$*, be* M *a bonded open set in a parameters space* $\mathbf{R}^K$*.* M *is a* choice area *for* $\mathbf{x}$ *if it is connected and every ansatz choice function parametrized by an environment vector in* M *maps* $\mathbf{x}$ *to the same choice:*

$$\exists \sigma \in S : f_{\mathbf{k}}\left(\mathbf{x}\right) = \sigma \, \forall \mathbf{k} \in M \tag{1.10}$$

Definition 7. *Be* M *a choice area for* $\mathbf{x}$*.* M *is a* maximal choice area *if it cannot be included in any other choice area:*

$$\nexists N : N is\ a\ choice\ area, M \subset N \tag{1.11}$$

Maximal choice areas can be intuitively seen as states on a geographical map: each mapping the same environment vector to the same choice. Note that there can exist more than one distinct maximal choice area mapping the same environment vector to the same choice. The weak form of continuity we are going to pose requests that for a bonded open set in the parameters space there exists only a finite number of distinct maximal choice areas.

Definition 8. *An environment space is* weakly continuous *if, for every bonded open set* A *in* $\mathbb{R}^K$ *and for every environment vector* $\mathbf{x}$ *in* $\mathbb{R}^N$*, there is only a finite number of maximal choice areas included in* A*.*

This request assures that there isn't a configuration vector around which the choice corresponding to the same environment vector varies an infinite number of times. Note that for problems of practical interest this request is almost always satisfied.

Genetic algorithms mostly make use of fitness functions statistically based on answers given by the agent to the environment vectors in the training set, compared with the actual results that should be reproduced by the agent. The most common examples of this were shown in Examples 1 and 2. An explicit definition should be made for this kind of fitness functions.

Definition 9. *A* comparison fitness function *is a fitness function which value is uniquely determined by the values of the corresponding ansatz choice function, evaluated on the environment vectors from the training set:*

$$\Phi_T(\mathbf{k}) = \Phi_T^c(\{f_{\mathbf{k}}\left(t_{\mathbf{x}}\right), t \in T\}) \tag{1.12}$$

An asexual reproduction and comparison based genetic algorithm iterates the process described above to select those agents that randomly get to be more and more fit to the evolutionary pressure determined by the comparison fitness function.

Step by step, parameters vectors will get closer and closer to local maxima of the fitness function. The speed and the precision of the training is determined by the evolutionary step: bigger steps introduce bigger changes on every new generation of agents, but when the population gets close to the local maxima they will likely oscillate around them with random local trajectories approximately of the size of the evolutionary step.

This kind of algorithms pose several issues:

- Locality of maxima: there is, in general, no explcit information on the shape of the fitness function. Therefore, even if this process is probabilistically designed to asymptotically approximate a local maximum of the fitness function, there is no guarantee for this to be its absolute maximum. Once a local maximum is reached, there is no chance for a trajectory to get out of it, unless there is a better local maximum in the range of an evolutionary step, which is very unlikely for evolutionary steps small enough to guarantee a decent approximation of the maximum.

- Genetic poverty: the algorithm mathematically selects brownian discrete trajectories in an highly dimensional space on which a fitness function is defined. Since explicit evalutation of that function in an adequately dense set of points is computationally unfeasible, trajectories are selected at each step on their *current* fitness. There is no guarantee that initially better trajectories lead to higher local maximum, therefore potentially better trajectories could be cut.

- Plateaux: even if it could intuitively seem very unlikely, comparison fitness functions generally *have* plateaux, on which size no assumption can be made. As we already discussed, the discrete nature of choices imply that different ansatz choice functions could yeld the same results not only on a specific environment vector, but also on *all* environment vectors in the training set. If we consider the request of weak continuity that has been made, we can see that for finite training sets the fitness function is completely made up of plateaux on whom sizes no assumptions can be made.

  This can indeed be proved considering that for a finite training set a multiple choice area can be defined.

  Definition 10. *Be T a training set, and M a bonded open set in a parameters space. M is a multiple choice area if it is connected and every environment vector in the training set is mapped to a constant choice:*

  $$\exists \sigma_1 \dots \sigma_s \in S : f_{\mathbf{k}}\left(\mathbf{x}_i\right) = \sigma_i \forall \mathbf{k} \in M \tag{1.13}$$

  Every multiple choice area is therefore an intersection of s choice areas (one for each environment vector in the training set). If the training set is finite and the parameters space is weakly continuous, it is completely made up of multiple choice areas of finite size. From the definition of comparison based fitness functions it is therefore obvious that such functions unavoidably have plateaux.

  Therefore if evolutionary pressure selects a population of agents whom parameters vectors lie in a plateau of the fitness function, there will be no evolutionary pressure on it any more, and brownian-like trajectories will generally randomly drift out of the plateau in a potentially very high number of steps.

- Efficiency: at each step, the best agents reproduce. Not only there is no guarantee that each son will be the *best* son for each agent, but sons can even have a fitness worse than their parent. Therefore this kind of algorithm is not really efficient, and could be trivially substituted by a local finite-difference evaluation of the gradient of the fitness function at the time of reproduction.

We will show in chapter 2 that the choice of a particular non comparison based fitness function solves many of the issues just described.

1.5 Neural networks

Neural networks are a particular type of ansatz choice functions designed to resemble the behaviour of a set of biological neurons. They are widely used in pattern recognition and non trivial data analysis, and usually treated as genetic algorithms with comparison based fitness functions as evolutionary pressure.

Neurons are organized in layers (usually only three layers are used, and they are called *input*, *hidden* and *output* layer). Each neuron is described by a real activation function, and is "connected" to all neurons in the previous and in the following layer by synapsis. Each synapsis has a "weight" describing how strongly the signal generated by a neuron affects the input of another neuron.

The number of neurons in the layer is equal to the number of dimensions of the environment space, while the number of neurons in the output layer is equal to the number of choices. The number of neurons in the hidden layer is not given by theory, but there are many practical rules in literature experimentally adjusted on the efficiency of the neural network, and should approximately be between the number of neurons in the other layers. We will refer to I, H and O as the number of neurons in the input, hidden and output layer respectively.

The output is computed as follows:

- A linear endomorfism (represented by an $I \times I$ matrix) is operated on the environment vector before reaching the input layer of neurons.
- Each component of the input vector is then passed through a real activation function. Many kind of activation functions exist in literature. The most common is the sigmoid function:

$$S(x) = \frac{1}{1+e^{-x}} \tag{1.14}$$

- Synapsis between the input and the hidden layer operate as a linear operator (represented by an $I \times H$ matrix).
- Signal reaching each hidden neuron is passed through the same activation function.
- Synapsis connect the hidden layer and the output layer (an $H \times O$ matrix is used to represent them), and the signal is checked for activation one last time.
- A final $O \times O$ linear transorm generates an output vector. The choice is generally represented by the output component with the highest value.

The number of parameters given by this ansatz is given by the total number of synapsis, each component of the parameters vector will define the strength of a link between two neurons. Since every link is an element of the matrices representing the linear transorms between the layers, the number of dimensions of the parameters space will be given by:

$$K = I^2 + IH + HO + O^2 \tag{1.15}$$

Note that since activation functions are monotonic, and upon continuous variations of values on the matrices vectors mapped vary continuously, this parameters space complies with the request of weak continuity described in section 1.4.

Chapter 2

Non-evolutive techniques

2.1 Non-evolutive AI training

Let us summarize what has been explained in the previous chapters. We defined an artificial intelligence as an ansatz function mapping from an environment space (vectors numerically describing an input) to a finite set of choices. We have shown how general such definition is, and how wide the variety of problems it can be applied to is. We assumed that there exists a training set, that is a set of associations between environment vectors and corresponding correct choices. We also defined a fitness function as a real function defined on the parameters space, and described the training process as the problem of finding the parameters vector maximixing the fitness function.

Note that no explicit definition was given for the form of a fitness function, except that it should practically describe the capability of the agent to solve a problem, given the training set. Next, an explicit definition for comparison based fitness functions has been given to emphasize the fact that usually fitness functions treat ansatz choice functions as black boxes: an input gets in, an output comes out, and the solution is evaluated comparing the answers for a set of inputs for which the answers are known. We also highlighted that one of the main problems posed by this kind of functions is, for finite training sets, the step nature of the fitness function, which not only can't have any analytical definition, but is also flat everywhere, except for gap discontinuities. Therefore, if two parameters vectors lie in the same multiple choice area (which is very likely to happen, if one wants to define some sort of gradient for the fitness function), no distinction is done by comparison based fitness functions between them, even if they could be intuitively distinguished by their proximity to the closest correct choice area.

Even if this is a very common paradigm, this is *not* however the only way to define a fitness function. We will give hereby a new ansatz for a choice function, and a new definition of a non comparison based, *continuous* fitness function.

Definition 11. *A* vector choice function *is a continuous function from an environment space $\mathbb{R}^N$ to a choice space $\mathbb{R}^{|S|}$, which dimension is equal to the number of possible choices:*

$$\tilde{f} : \mathbb{R}^N \to \mathbb{R}^{|S|} \tag{2.1}$$

Definition 12. *A* selection function *is a function that maps a choice vector in* *a choice by selecting its component with the highest value, and yielding the corresponding choice:*

$$\tilde{\sigma} : \mathbb{R}^{|S|} \to S : \tilde{\sigma}(\mathbf{c}) = \sigma_i \Leftrightarrow \mathbf{c}_i = max\left(\{\mathbf{c}_j, j = 1, 2 \ldots, |S|\}\right) \tag{2.2}$$

We will now define an ansaz choice function in terms of the composition of a vector choice function and a selection function. The vector choice function is responsible for the actual choice, we can interpret its output as some sort of *likelyhood* for each choice, as extimated by the vector choice function. The actual choice is then given by the selection function, that simply selects the most likely choice from the choice vector. This decomposition has the great advantage of making it possible to define non comparison based fitness functions on the parameters space. We will indeed define our fitness function *directly* on the output of the vector choice function, easily gaining not only the advantage of continuity on the parameters, but also an analytical definition for the fitness function, that will be nicely easy to maximize.

We will start by using a simple ansatz for vector choice functions, and will generalize it later. For now, we will use only linear functions as vector choice functions:

$$\tilde{f}^{(l)} \in \mathcal{L}\left(\mathbb{R}^N, \mathbb{R}^{|S|}\right) \tag{2.3}$$

The parameters vector will be naturally defined by the elements of the matrix representing $\tilde{f}^{(l)}$. Note that obviously each row of this matrix represents a vector in the environment space, and each component of the choice vector is given by a scalar product between the environment vector and the corresponding row of the matrix. We will explictly refer to each of these parameters vectors in the environment space labelling them $\mathbf{y}_i$, with $i = 1, 2, \ldots, |S|$.

$$\tilde{f}^{(l)}_{\mathbf{y}_1,\ldots,\mathbf{y}_{|S|}} : \mathbb{R}^N \to \mathbb{R}^{|S|} : \left(\tilde{f}^{(l)}(\mathbf{x})\right)_i = \mathbf{x} \cdot \mathbf{y}_i \tag{2.4}$$

By observing that the selection function is scale independent:

$$\tilde{\sigma}(\lambda \mathbf{c}) = \tilde{\sigma}(\mathbf{c}) \text{ for} \lambda > 0 \tag{2.5}$$

we make the ansatz of $|\mathbf{y}_i| = 1$. We will also refer to $\mathbf{y}_i$ as choice versors. Moreover, each vector in $\mathbf{x}$ in the training set T can be normalized as well.

There is now need for a fitness function, defiend on choice vectors, capable of somehow representing the quality of an answer before the selection function is used.

If we interpret the choice vector as the likelyhood of each choice, a good vector choice function should output a neat vector with a high likelyhood corresponding to the correct choice, and no likelyhood at all for all the other choices.

The capability of the agent of correctly making a specific choice can then be assessed by the average ratio on the training set restricted to that choice between the component corresponding to that choice and the module of the choice vector. That is, expressing the training set as:

$$T = \bigcup_{i=1}^{|S|} \left(T_i = \{(\mathbf{x}, \sigma_i) \in T\}\right) \tag{2.6}$$

we can define a choice-specific fitness function Φ_i as:

$$\Phi_i = \frac{\displaystyle\sum_{(\mathbf{x},\sigma_i)\in T_i} \left(\frac{(\mathbf{x}\cdot\mathbf{y}_i)}{\sqrt{\sum_{j=0}^{|S|}(\mathbf{x}\cdot\mathbf{y}_j)^2}} \right)}{|T_i|} \tag{2.7}$$

Each of these fitness functions should be maximized in respect to the constraint given to $\mathbf{y}_i$.

Since a scalar product is obviously maximized when its two operand vectors are parallel, recalling the normalization condition for $\mathbf{y}_i$, Φ_i is maximum when

$$\mathbf{y}_i = \frac{\displaystyle\sum_{(\mathbf{x},\sigma_i)\in T_i} \left(\frac{(\mathbf{x})}{\sqrt{\sum_{j=0}^{|S|}(\mathbf{x}\cdot\mathbf{y}_j)^2}} \right)}{\left|\displaystyle\sum_{(\mathbf{x},\sigma_i)\in T_i} \left(\frac{(\mathbf{x})}{\sqrt{\sum_{j=0}^{|S|}(\mathbf{x}\cdot\mathbf{y}_j)^2}} \right)\right|} \tag{2.8}$$

The solution of this equation is highly non trivial. A solution for it can however be approximated in a computationally efficient way.

Recall that if $\Xi(\mathbf{x})$ is a contraction, that is:

$$|\Xi(\mathbf{y}) - \Xi(\mathbf{p})| < |\mathbf{y} - \mathbf{p}| \ \forall \mathbf{y}, \mathbf{p} \tag{2.9}$$

then Ξ has a fixed point, which is solution for $\Xi(\mathbf{x}) = \mathbf{y}$. Moreover, the sequence defined by

$$\mathbf{y}_n = \Xi^{(n)}(\mathbf{y}_0) \tag{2.10}$$

converges to the fixed point for all $\mathbf{y}_0$.

This property is particularly useful when trying to solve big systems of equations with such a complex analytical expression. Note that the *whole* system of equations must define a contraction. In other words, the vector $\mathbf{y}$ must represent the whole set of parameters that are to be determined by the system. In this case, we can define:

$$\mathbf{y} = \begin{pmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \vdots \\ \mathbf{y_{|S|}} \end{pmatrix} = \begin{pmatrix} (\mathbf{y}_1)_1 \\ (\mathbf{y}_1)_2 \\ \vdots \\ (\mathbf{y}_1)_N \\ (\mathbf{y}_2)_1 \\ \vdots \\ \left(\mathbf{y}_{|S|}\right)_N \end{pmatrix} \tag{2.11}$$

Ξ will be defined by (2.8) for each group of N components, that is for each $\mathbf{y}_i$. We now have to prove that the map hereby defined is a contraction.

Let's start by noting that $\mathbf{y}$ lies in $\Omega^{|S|,N}$, where

$$\Omega^{P,N} = \left\{ \mathbf{v} \in \mathbb{R}^{P\cdot N}, \sum_{i=kN+1}^{(k+1)N} (\mathbf{v}_i)^2 = 1,\ k = 0, 1, \ldots, (P-1) \right\} \tag{2.12}$$

is the set of points that lie in P N-dimensional circles in $\mathbb{R}^{P\cdot N}$. Note that each vector in $\Omega^{P,N}$ obviously satisfies:

$$|\mathbf{v}| = \sum_{i=0}^{N\cdot P} \mathbf{v}_i^2 = P\ \forall \mathbf{v} \in \Omega^{P,N} \tag{2.13}$$

That is, every vector defined as in (2.11) also lies in the $|S|\,N$ dimensional sphere of radius $P^{1/2}$.

Next, we note that the distance between two points lying on a sphere is a monotonically decreasing function of their scalar product. Therefore:

$$\forall \mathbf{x},\mathbf{y},\mathbf{z},\mathbf{w} \in \Omega^{|S|,N}, |\mathbf{x}-\mathbf{y}| < |\mathbf{z}-\mathbf{w}| \implies \mathbf{x}\cdot\mathbf{y} > \mathbf{z}\cdot\mathbf{w} \tag{2.14}$$

So that condition (2.9) can be rewritten on the sphere as:

$$\Xi(\mathbf{y})\cdot\Xi(\mathbf{p}) > \mathbf{y}\cdot\mathbf{p} \tag{2.15}$$

Which, in terms of $\mathbf{y}$ as defined in (2.11) takes the following form:

$$\sum_n^{|S|}\left(\frac{\sum_i^{|T_n|}\left(\frac{\mathbf{x}_i^{(n)}}{\sqrt{\sum_l^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\right)}}\right)}{\left|\sum_i^{|T_n|}\left(\frac{\mathbf{x}_i^{(n)}}{\sqrt{\sum_l^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\right)}}\right)\right|}\cdot\frac{\sum_j^{|T_n|}\left(\frac{\mathbf{x}_j^{(n)}}{\sqrt{\sum_k^{|S|}\left(\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right)}{\left|\sum_j^{|T_n|}\left(\frac{\mathbf{x}_j^{(n)}}{\sqrt{\sum_k^{|S|}\left(\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right)\right|}\right) > \sum_n^{|S|}\left(\mathbf{y}_i\cdot\mathbf{p}_i\right) \tag{2.16}$$

This can be treated with some algebra. We need a chain of inequalities to prove (2.16). We can use the triangular inequality on the denominator D of each term of the sum in n:

$$D \le \sum_i^{|T_n|}\left(\frac{\left|\mathbf{x}_i^{(n)}\right|}{\sqrt{\sum_l^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\right)}}\right)\sum_j^{|T_n|}\left(\frac{\left|\mathbf{x}_j^{(n)}\right|}{\sqrt{\sum_k^{|S|}\left(\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right) = \tag{2.17}$$

$$= \sum_{i,j}^{|T_n|}\left(\frac{1}{\sqrt{\sum_{l,k}^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right) \tag{2.18}$$

While each numerator can be easily expressed as:

$$\sum_{i,j}\left(\frac{\mathbf{x}_i^{(n)}\cdot x_j^{(n)}}{\sqrt{\sum_{l,k}^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right) \tag{2.19}$$

Therefore (2.16) is bigger than:

$$\sum_{n}^{|S|}\left(\frac{\sum_{i,j}\left(\frac{\mathbf{x}_i^{(n)}\cdot x_j^{(n)}}{\sqrt{\sum_{l,k}^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right)}{\sum_{i,j}^{|T_n|}\left(\frac{1}{\sqrt{\sum_{l,k}^{|S|}\left(\left(\mathbf{x}_i^{(n)}\cdot\mathbf{p}_l\right)^2\left(\mathbf{x}_j^{(n)}\cdot\mathbf{y}_k\right)^2\right)}}\right)}\right) \tag{2.20}$$

So (2.8) is a contraction if:

$$\sum_{n}^{|S|}\sum_{i,j}^{|T_n|}\mathbf{x}_i^{(n)}\cdot\mathbf{x}_j^{(n)} > \sum_{n}^{|T_n|}\mathbf{p}_n\cdot\mathbf{y}_n\ \forall\,\mathbf{p},\mathbf{y} \tag{2.21}$$

That is:

$$\sum_{n}^{|S|}\sum_{i,j}^{|T_n|}\mathbf{x}_i^{(n)}\cdot\mathbf{x}_j^{(n)} > S \tag{2.22}$$

Note that this request expresses a need for locality of the vectors in the training subset corresponding to each choice: no linear vector choice function can interpolate points located at opposite sides of a sphere. This requirement, however, is for practical purposes always fulfilled, since each vector in the training set contributes to the sum in i and j with a scalar product with itself. We will not try to build pathological examples of training sets *not* fullfilling (2.22).

Note that the only practical use where this inequality could be falsified is when there is only one vector in the training subset corresponding to each choice, but in this case no contraction is needed to solve (2.8): each choice versor equals each training vector, normalized:

$$\mathbf{y}_n = \mathbf{x}_1^{(n)} \tag{2.23}$$

In all cases compatible with the contraction condition, an arbitrarily accurate solution for (2.8) can be computed as the limit of the succession defined as follows:

$$\begin{cases}\mathbf{y}_i^0 = \dfrac{\sum_j^{|T_i|}\mathbf{x}_j^{(i)}}{\left|\sum_j^{|T_i|}\mathbf{x}_j^{(i)}\right|}\\[2ex] \mathbf{y}_i^{(n+1)} = \dfrac{\sum_j^{|T_i|}\frac{\mathbf{x}_j^{(i)}}{\sqrt{\sum_l^{|S|}\left(\left(\mathbf{x}_j^{(i)}\cdot\mathbf{y}_\mathbf{l}^{(\mathbf{n-1})}\right)^2\right)}}}{\left|\sum_j^{|T_i|}\frac{\mathbf{x}_j^{(i)}}{\sqrt{\sum_l^{|S|}\left(\left(\mathbf{x}_j^{(i)}\cdot\mathbf{y}_\mathbf{l}^{(\mathbf{n-1})}\right)^2\right)}}\right|}\end{cases} \tag{2.24}$$

2.2 Polynomial generalization

To sum up: we attributed many of the problems connected with some existing AI techniques to a non efficient definition of their ansatz and fitness functions. In particular, we have shown how comparison based fitness functions are intrinsically made up of plateaux, so that the very gradient of the function is everywhere null of undefined. Through a new ansatz for the choice function, that expresses it as the composition of a vector choice function and a selection service function, and by defining the fitness function directly on the answers of the vector choice function instead of doing the comparison on the output of the selection function, we have built a fitness function that is continuous on the parameters space.

We then used a linear ansatz for the vector choice function itself. This made the fitness function not only continuous, but also analytically defined in a simple form on the parameters. A linear function can be expressed as a set of scalar products. A fitness function was then separately defined on each choice as the ratio between the component of the choice vector corresponding to each choice and the module of the choice vector. The maximization of each fitness function therefore reduces to the determination of the corresponding choice vector.

The functional form of Φ_i is in terms of a scalar product between $\mathbf{y}_i$ and a function of $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{|S|}$. We then defined a vector $\mathbf{y}$ grouping all the choice vectors. The set of equations in $\mathbf{y}_1, \ldots, \mathbf{y}_{|S|}$ is now expressed as a vectorial equation in the form $\mathbf{y} = \Xi(\mathbf{y})$. Since this equation has a complex form, it cannot be easily solved analytically. But, if Ξ can be proven to be a contraction, the solution of $\mathbf{y} = \Xi(\mathbf{y})$ can be expressed as $\lim_{n\to\infty} \Xi^{(n)}(\mathbf{y_0})$ for any $\mathbf{y_0}$.

To prove that Xi is a contraction, we observed that each vector in y lies in $\Omega^{|S|,N}$, which is a subset of the $|S|N$-dimensional sphere $\{\mathbf{y}, |\mathbf{y}| = \sqrt{|S|}\}$. Since on the sphere the scalar product between two vectors is a monotonically decreasing of their distance, we could put the contraction condition for Ξ in a nice form, that could be easily reduced to (2.22).

Now, in first approximation, the i-th choice versor for the ansatz described above is just the average of the environment vectors in T_i. Since this average must lie on the sphere, there are obvious cases for which this is impossible.

In a three dimensional environment space, on example, it is possible to build this pathologic training subset T_1^p:

$$T_1^p = \{(0,0,1), (0,0,-1))\} \tag{2.25}$$

Which obviously doesn't comply with (2.22). In general a succession defined as in (2.24) will oscillate without converging.

Practical examples of cases where this could happen actually exist. On example, there could be need to train an agent to discern the axis of $\mathbb{R}^3$ to which a point is closest. In this case, the training set for each axis would be in a form similar to (2.25). Linear functions therefore are not capable of interpolating any kind of training set, and a generalization to other kind of functions is needed. On example, a good vector choice function for (2.25) could be given by:

$$\left(\tilde{f}((x,y,z))\right)_1 = z^2 \tag{2.26}$$

Recall that a general vectorial polynomial function $p : \mathbb{R}^n \to \mathbb{R}$ of degree P

takes the form:

$$p(\mathbf{x}) = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}^T + \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1N} \\ y_{21} & y_{22} & \cdots & y_{1N} \\ \vdots & \ddots & & \vdots \\ y_{N1} & y_{N2} & \cdots & y_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}^T + \\ + \ldots + \sum_{(a_1,a_2,\ldots,a_P)\in\{1,2,\ldots,N\}^P} \left(y_{a_1 a_2 \ldots a_P} \prod_{i=1}^{P} x_{a_i} \right) \tag{2.27}$$

Where each variable identified by y is a parameter for the ansatz. This in turn can be put in the equivalent form:

$$p(\mathbf{x}) = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \\ y_{11} \\ y_{12} \\ \vdots \\ y_{1N} \\ y_{21} \\ \vdots \\ y_{NN} \\ y_{111} \\ \vdots \\ y_{NNN} \\ \vdots \\ y_{\underbrace{N\ldots N}_{\text{P times}}} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ x_1^2 \\ x_1 x_2 \\ \vdots \\ x_1 x_N \\ x_1 x_2 \\ \vdots \\ x_N^2 \\ x_1^3 \\ \vdots \\ x_N^3 \\ \vdots \\ x_N^P \end{pmatrix} \tag{2.28}$$

Where we explicited the polynomial function in $\mathbf{x}$ as a scalar product of a coefficients vector and a polynomial vector function on $\mathbf{x}$.

Note that the polynomial function that maps a vector in a vector in an higher dimensional space making explicit all of its internal products up to a given degree does not depend on any parameter.

Definition 13. *A* polynomial explicitation map up to a degree d *is a function $\tilde{p}_d : \mathbb{R}^N \to \mathbb{R}^L$, with $L = \sum_{i=1}^{d} N^i$ defined by:*

$$\tilde{p}_d((x_1, x_2 \ldots x_N)) = \frac{\left(x_1, x_2, \ldots, x_N, x_1^2, x_1 x_2 \ldots x_1 x_N, x_1 x_2 \ldots x_N^2, x_1^3 \ldots x_N^3 \ldots x_N^P\right)}{\left|\left(x_1, x_2, \ldots, x_N, x_1^2, x_1 x_2 \ldots x_1 x_N, x_1 x_2 \ldots x_N^2, x_1^3 \ldots x_N^3 \ldots x_N^P\right)\right|} \tag{2.29}$$

There is therefore no difference between using a polynomial ansatz for vector choice functions on an environment space, and using a linear ansatz instead, but on a higher dimensional environment space, in which every training vector is the polynomial explicitation map p_d of the vectors in the original environment.

It is then easily possible to generalize the findings from the previous section to any polynomial degree, by adding $\tilde{p}_d$ to the composition:

$$f = \tilde{\sigma} \circ \tilde{f} \circ \tilde{p}_d \tag{2.30}$$

Where $\tilde{f}$ is a linear function from $\mathbb{R}^L$ to $\mathbb{R}^S$, and the versors determined by the maximization of Φ_i represent the set of polynomial coefficients for the most efficient polynomial ansatz choice function for the given training set.

Chapter 3

Examples and optimization

3.1 Polynomial requirements

It is known from basic calculus that every continuous function defined on a compact space can be arbitrarily approximated by a polynomial function. It is then obvious that the technique described above offers an arbitrary precision solution for nearly any problem of practical use. Since the dimension of the polynomial space on which $\tilde{p}_d$ maps every environment vector exponentially increases in d, it is not however computationally feasible to handle $\tilde{p}$ over some low degree. It is therefore important to discern if a given problem *requires* a polynomial choice function of a minumum degree, i.e. it cannot be solved by a lower degree polynomial choice function.

Two practical examples will be given to emphasize the simplicity and the efficiency of this non evolutive training paradigm.

3.1.1 Linear captcha recognizer

The CAPTCHA (**c**ompletely **a**utomated **T**uring test to tell **c**omputers and **h**umans **a**part) is a widely used test designed to prevent automation on many procedures on the internet. Even if bots (software agents built to automize process like form-filling, crawling, etc.) can emulate a user's interaction with almost any service, a Turing test validated request ensures that a human is specifically making such request. Since the number of requests to be handled must be high enough to filter even a huge number of automated requests, no human must be needed to create the Turing problem.

The most common type of CAPTCHA problem is an image representing a string. Thanks to high efficiency pseudo-random algorithms, an arbitrary quantity of noise can be introduced. The noised image is shown to the user, and the interaction is validated only if the user (which is human, and its perception is very resilient to noise) enters the correct string.

In this example, one-char CAPTCHAs will be used for simplicity. 26 bitmap images (50×50 pixels) were created to represent each letter in the latin alphabet using Calibri (a commonly used font).

An image of a given size can be easily codified in a vector through a bitmap. On a grayscale, each pixel of the image is represented by an integer in a given range (usually $[0, 255]$, 0 representing black and 255 white). Images were loaded in vectors

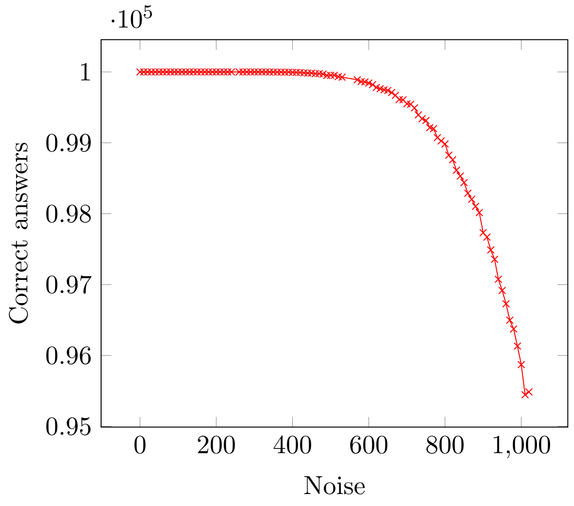

Figure 3.1: Efficiency vs. noise with linear captcha recognizer

(the environment space can be seen as a compact in $\mathbb{R}^2500$). Each image vector has been normalized and was used as training set corresponding to each image (26 choices), so minimal size training sets have been used. Each image was then dimed so that each pixel value ranged in $[64, 192]$. Once the parameters for each *linear* ansatz vector choice function were determined, test were run as follows.

- A random character was chosen
- Its image vector was copied, and white noise in $[-\xi, +\xi]$ was added, where ξ was a parameter used to set the intensity of noise.
- The vector was normalized.
- A scalar product was computed with each choice versor.
- The choice corresponding to the highest scalar product was selected, and it was compared with the actual character.

Efficiency was computed in terms of the number of correct answers over 10^5 tests (figure 3.1). Note that the amplitude of the signal itself was up to 128. This algorithm shows an outstanding resilience for noise, with a **95% efficiency for a noise 10 times bigger than the signal itself!**

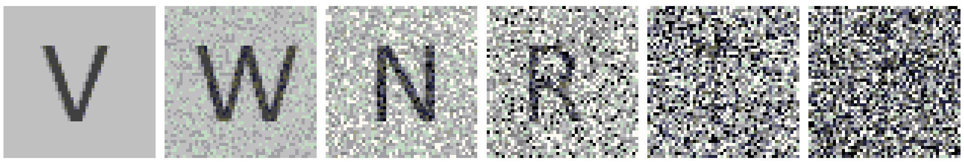

Figure 3.2: Characters with increasing noise: (V, 0), (W, 50), (N, 100), (R, 150), (T, 200), (Q, 250)

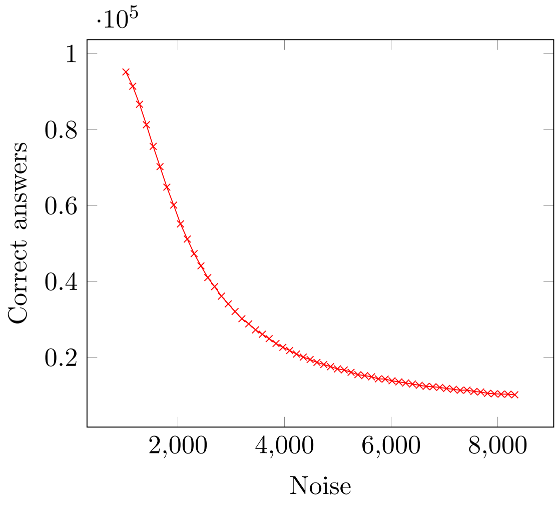

Figure 3.3: Efficiency vs. noise with linear captcha recognizer, higher noise

Figure 3.2 shows five examples of captchas generated with increasing noise. The algorithm gets a **100% efficiency even on characters like the last in the figure**.

Figure 3.3 shows the results of an efficiency test on more noisy signals (from 10 to 80 times the signal). This Turing test is efficiently solved even for images that a human isn't able to recognize.

This efficiency, however, is due to a good compatibility between the input and the agent. In other words, what is provided to the agent is an actual vector that has been perturbed with noise. Many other problems in image analysis require the agent, on example, to be resilient to distortion, scale, and rotation. But from a mathematical point of view, a rotation, or a distortion, is a permutation over the components of the input, altering its very structure. In these cases, transforms must be applied to the input before analyzing it. As we will see, these transforms are highly non injective functions on the raw image input, designed to map rotated, scaled or distorted images to the same or close vectors.

3.1.2 Tic Tac Toe

Tic Tac Toe is a popular strategy, turn based, complete information game. Two players take turns in placing their marks (usually a cross and a circle) over a 3×3 grid. Two marks cannot be placed on the same square of the grid. If at any turn there is a full row, coloumn or diagonal marked with the symbol of one of the two players, that player wins the game. Obviously, several configurations of the game grid can lead to a draw.

The small tree complexity of this game makes it a good pedagogical example in the field of artificial intelligence. Since an optimal strategy is known, we will determine the efficiency of an artificial intelligence in terms of the number of games it wins against a random player.

Figure 3.4 shows two different configurations of the game grid and two possible ways to represent them with vectors. Since each square can be either unmarked,

$$\begin{array}{c|c|c} \times & \circ & \\ \hline \times & & \circ \\ \hline \circ & \times & \end{array} \to \begin{pmatrix}1\\-1\\0\\1\\0\\-1\\-1\\1\\0\end{pmatrix} \text{ or } \begin{pmatrix}1\\0\\0\\1\\0\\0\\0\\1\\0\\0\\1\\0\\0\\0\\1\\1\\0\\0\end{pmatrix} \qquad \begin{array}{c|c|c} \circ & \circ & \circ \\ \hline \times & & \times \\ \hline \circ & \times & \times \end{array} \to \begin{pmatrix}-1\\-1\\-1\\1\\0\\1\\-1\\1\\1\end{pmatrix} \text{ or } \begin{pmatrix}0\\0\\0\\1\\0\\1\\0\\1\\1\\1\\1\\1\\0\\0\\0\\1\\0\\0\end{pmatrix}$$

Figure 3.4: Tic Tac Toe configurations, with two vector representations

marked with a circle or marked with a cross, the most compact way to describe the configuration of a game table is using a 9-dimensional vector, each representing a square. An empty square can be represented by a 0, a cross marked square with a 1, and a circle marked square with a -1.

This however makes two vectors representing the same play, only with circles and crosses swapped, as opposite. Another way to represent the table that allows one to not make this assumption is to use 18-dimensional vectors. The first 9 components are to tell if there is a cross in each square, the last 9 if there is a circle. Obviously, there are vectors in this representation that don't resemble any actual configuration.

Using the second representation, a training set was generated based on random games. Each game could be won, lost or drawn by the cross player. If the game was won, each move was recorded, that is each couple (present configuration, move done by the cross player as a random, but winning reaction to that configuration) was added to the training set.

Choice versors were generated from the training set built as described, and the agent obtained was tested against a random player. The selection function was slightly modified to select only the *valid* choice represented by the component with the highest value, so that the agent is forced to follow the rules of the game.

Figures 3.5 and 3.6 show the efficiency of two agents, trained with a growing number of items in the training set, both generated the same way. The first plot relates to a linear ansatz choice agent, while the ansatz for the second was a second degree polynomial.

It is evident from the plots that while the growth of the training set of the first doesn't affect its efficiency, that stays approximately in a random neighborhood

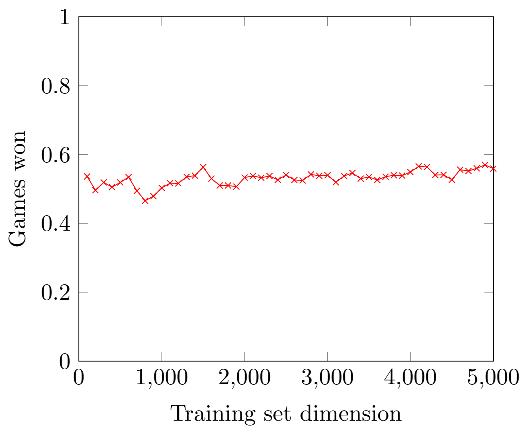

Figure 3.5: Fraction of games won by linear Tic Tac Toe player vs. dimension of the training set

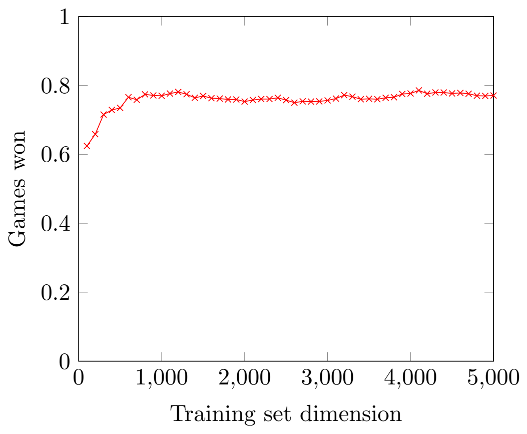

Figure 3.6: Fraction of games won by quadratic Tic Tac Toe player vs. dimension of the training set

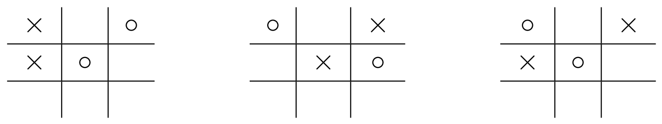

Figure 3.7: These three configurations are equally interpreted by a linear player

of the 0.5 efficiency any random player would have, the efficiency of the second significantly increases for the first hundreds of elements in the training set, until it reaches a plateau.

First, the height of the second plateau (the asymptotic efficiency of the second degree player doesn't reach the 100% efficiency) can be easily justified in terms of the random tree exploration done by the randomly playing opponent. Since there *are* perfect games in Tic Tac Toe for the player starting the game, a perfect path can be chosen at random leading to the defeat of the trained player.

The difference between the two efficiencies, in turn, can be justified by the very *nature* of the Tic Tac Toe problem.

Figure 3.7 shows three configurations of the game table. It is evident that the correct answer to the first two is placing a cross in the bottom left square to win the game, while the correct answer to the last one is placing a cross in the bottom right square, preventing the circle player from winning the game at the next move.

Now, if the first two elements were to use for a linear training set, both would belong to the 7th training subset. If the ansatz for the vector choice function was to be linear, this would mean:

- Squares number 1 and 4 if occupied by the cross player, should lead to the choice of marking the square number 7.
- Squares number 3 and 5, if occupied by the circle player, should lead to the choice of marking the square number 7.
- Squares number 3 and 5 if occupied by the cross player, should lead to the choice of marking the square number 7.
- Squares number 1 and 6, if occupied by the circle player, should lead to the choice of marking the square number 7.

Now, note that the linearity of the ansatz implies that the *uncorrelated* presence of these marks should lead to the said choice. No additional value is given to the *contemporary* presence of two marks at a time in two different squares, which is what actually determines the choice.

Therefore, when shown the third configuration, the agent will react as it would to a superposition of the first two, by putting its mark on the bottom left corner and losing the game.

These two problems were shown to emphasize their intrinsic difference: while the solution to the first problem, if distortion, scaling and rotation are exluded, is given by a superposition of absolute values of each pixel, the solution for the second is based on the relation between components of the environment vector. The intrinsic polynomial requirement for a problem is to be considered as the maximum number of interacting properties to be considered at the same time. This leads to a reasonable upper bound for the polynomial degree of the choice function, which should not be higher than the number of dimensions of the environment space.

3.2 Computational considerations

The exponential growth of the dimension of the parameters space with the degree of the polynomial ansatz for the choice function can lead to three different kind of problems:

- **Space complexity**: since the number of parameters per agent grows exponentially with the polynomial degree of the agent's ansatz, the first problem is obviously space complexity. Once trained, an agent should be practical to transfer and store.
- **Time complexity (training)**: we have seen how the determination of the choice versors is done through the iteration of a succession defined by induction. If the environment space has an $O(exp(p))$ dimensions, each step of the iteration is going to need a proportional amount of time.
- **Time complexity (execution)**: in general, once it has been trained, the agent will be deployed and used massively. Its efficiency is going to be the most crucial problem to deal with.
- **Size of the training set (statistics)**. Unlike what has been done in the CAPTCHA recognition example, training sets are generally constituted by experimental data that *is* subject to error. If, instead of the original, noiseless images we used, on example, the fifth kind of image from figure 3.2, we would not have been able to use just one image, since it would have been significantly different from the actual character that should be used as a term of comparison. A statistical mass of examples is needed in general for the training set, which size must be estimated in terms of the degree of the ansatz.

  Note that even if the set is randomly generated, since it originates from a given signal the contraction condition (2.22) is guaranteed to be statistically satisfied. It is known from basic statistics, however, that the standard deviation associated to the distribution of the average of a set of N randomly distributed variables decreases with the square root of the number of variables:

  $$\sigma \propto \frac{\bar{\sigma}}{\sqrt{N}} \tag{3.1}$$

  where $\bar{\sigma}$ is the standard deviation of the distribution of each variable. Be $\bar{\Delta x}$ the maximum acceptable error for the single component of the choice versor. The probability of the average for a single component being within the error range is:

  $$P(\Delta x \leq \bar{\Delta x}) = \text{erf}\left(\frac{\bar{\Delta x}}{\sigma}\right) = \text{erf}\left(\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}\right) \tag{3.2}$$

  The probability of all the M components of the choice versor being within the error range (assuming them independent and with the same standard deviation) is:

$$P^* = \left(\operatorname{erf}\left(\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}\right)\right)^M \tag{3.3}$$

So, if χ is the confidence level required for P^*, the following must be satisfied:

$$1 - P^* < \chi \tag{3.4}$$

Upon expanding the error function in series in $+\infty$:

$$\operatorname{erf}(x) = e^{-x^2}\left(-\frac{1}{\sqrt{\pi}x} + \frac{1}{2\sqrt{\pi}x^3} - \frac{3}{4\sqrt{\pi}x^5} + \dots\right) + 1 \tag{3.5}$$

We obtain, as a first order valuation:

$$1 - \left(\frac{-\exp\left(-\left(\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}\right)^2\right)}{\sqrt{\pi}\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}} + 1\right)^M < \chi \tag{3.6}$$

Which, upon expanding again in Taylor series for $x \to \infty$, becomes:

$$M\frac{-\exp\left(-\left(\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}\right)^2\right)}{\sqrt{\pi}\frac{\bar{\Delta x}\sqrt{N}}{\bar{\sigma}}} < \chi \tag{3.7}$$

Now, by noting that, for $x \to \infty$:

$$\frac{e^{-x^2}}{x} = e^{-x^2 - ln(x)} \simeq e^{-x^2} \tag{3.8}$$

We must have:

$$\exp\left(-\frac{\bar{\Delta x}}{\sigma}\right) < \frac{\sqrt{\pi}\chi}{M} \tag{3.9}$$

Since, in turn, $M \propto e^d$, where d is the polynomial degree, we obtain:

$$\sqrt{N} \propto d \tag{3.10}$$

Which makes the size of the training set only polinomial in respect to the degree of the ansatz.

There are a few easy expedients one can use to tackle with some of these problems.

3.2.1 Precomputing data

We have proven that it is possible to analytically determine the best polynomial function to reproduce a training set. Each vector in the training set contains data to be analyzed. For sufficiently big training sets (enough to show *each and every* possible example of (input, choice)), and for a sufficiently high polynomials degrees, an efficient function *will* be found. However, this doesn't consider any efficiency issue.

As an example for this, we will focus on one of the applications that have already been discussed: optical character recognition for strings. An image with text in it has to be analyzed to extract such text.

The easiest approach that can be used to tackle with this problem is to deputize every analysis to the training algorithm. The entire image of the page is submitted as environment vector, the choices being *any combination of letters up to the number of characters that can fit within a page.* The training algorithm will need to determine an equal number of choice versors, which in turn need at least an equal number of elements in the training set. We are mathematically assured that a solution for this problem exists, but it is impossible to generate such a training set, and to train so many choice versors. Moreover, no assumptions can be done on the degree of the agent needed for this problem.

Mathematically speaking, however, some assumptions can be made. We know that the value of each letter is independent from the value of the others. There is no need, then, to introduce in the polynomial all the internal products of independent variables that we already know will statistically vanish.

Practically speaking, artificial intelligence algorithms should be used where no simple analytical preprocessing can be done. We have already discussed an efficient implementation for a single character optical recognition algorithm. An efficient, but still quite error resilient technique that can be used is to scan portions of the image trying to detect single characters in different positions.

The following procedure has ben followed:

- On a squared image (1000×1000 pixels), 1000 random characters have been drawn in random positions. The approximate size of each character was 50×50 pixels. The value and coordinate of each character was recorded for comparison.
- The signal was dimed reducing the gap between black and white.
- Noise has been introduced, obtaining an image like the one in Figure 3.8.
- Images representing single characters have been exported without noise to be used as training set.

The scanning program has been then run with the following procedure:

- Images representing single characters where used as single-element training sets, determining choice versors.
- Any 50×50 pixels image can be compared with the choice versors. Since any portion of the test image of that size obviously complies with this requirement, the single character recognition agent was run on each of them, scanning the whole image. To each pixel was associated a character recognized by the choice agent in the 50×50 square centered on it. Since no dump choices as been put in the choices set, even those squares that don't actually represent any character are assigned some choice.
- As a further training, 10 coordinates of known letters on the image were specifically scanned, and the value of the scalar product between each correct choice versor and each square was computed.

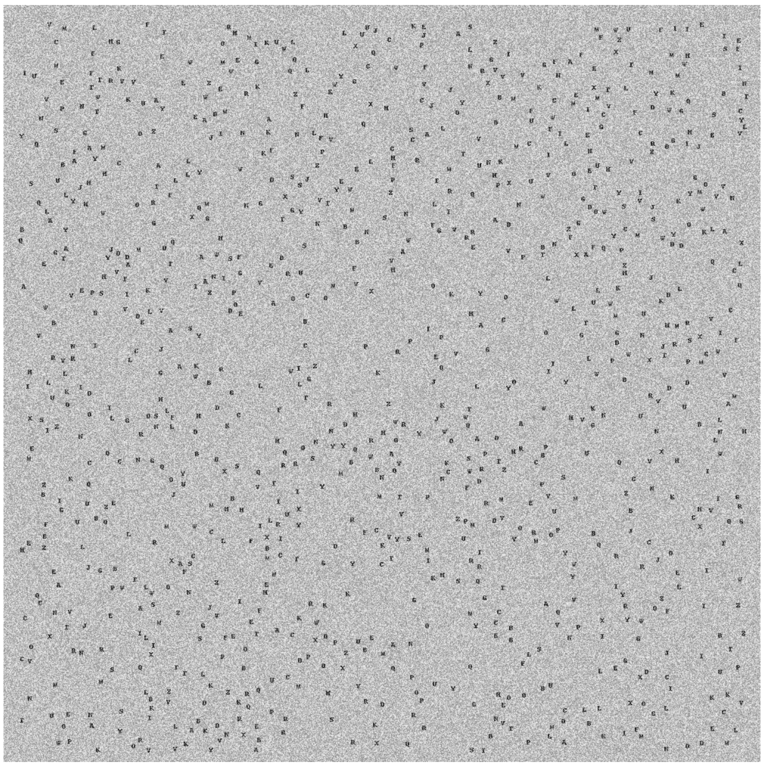

Figure 3.8: Image generated for testing purposes on the OCR string recognizer.

- Based on the values obtained, a threshold has been set on the minimum value of the scalar maximum product to consider the corresponding choice acceptable. We will justify this procedure more in detail in the following chapters. The coordinates whose scalar product complied with the threshold were selected, and compared with the original letters and coordinates recorded.

Five tests were run on randomly generated sets of 1000 images. Characters were considered correctly detected if the letter was correctly chosen and its coordinates matched exactly with the values recorded. The following results were obtained:

Table 3.1: OCR scanning efficiency (noise / signal $\simeq 0.5$)

#	Characters detected	False positives
1	951	24
2	963	17
3	952	17
4	962	30
5	946	26

The scanning procedure took approximately 3.4 seconds on an Intel Core i5, 2.3 GHz processor.

Tests were run also on lower levels of noise, recalibrating thresholds, with the same procedure:

Table 3.2: OCR scanning efficiency (noise / signal $\simeq 0.2$)

#	Characters detected	False positives
1	996	1
2	995	0
3	995	0
4	997	2
5	995	0

On an actual page of text, noise levels are usually lower than in the second test, and characters are placed in a grid-like pattern, making it easier to detect false positives. A further improvement (whose implementation will not be discussed here) could be done if the language of the text was known. On example, if letters could be codified in vectors in a fixed dimensional space (which is easy if one introduces a null character at the end of the word), a dictionary could be used as training set, with one choice versor per word, and a second choice agent could be used to correct errors done by the first, increasing efficiency further.

3.2.2 Zeros deletion

On high degree polynomial choice versors, many zeros are likely to be present. Both runtime and space efficiency can be improved if zeros are not explicitly stored. On example, a vector with many null components like:

$$\mathbf{n} = (0, 0, 0, 1.7, 0, 0, 4.3, 0, 0, 0, 0, 2.2, 0, 0, 9.9, 0, 0) \tag{3.11}$$

Could be stored as:

$$\mathbf{n}_4 = 1.7, \mathbf{n}_7 = 4.3, \mathbf{n}_12 = 2.2, \mathbf{n}_15 = 9.9 \tag{3.12}$$

Obviously, on problems of practical interests, components are unlikely to be exactly null. This issue can be treated, however, noting that increasing the statistical mass of the training set, the value of ideally null components will converge to zero with an increasing confidence level.

3.2.3 Components sorting

Let's take an N-dimensional choice versor $\mathbf{c}$ and compute its scalar product with a normalized environment vector $\mathbf{n}$. Their scalar product is defined by:

$$\mathbf{c} \cdot \mathbf{n} = \sum_{i=0}^{N} \mathbf{c}_i \mathbf{n}_i \tag{3.13}$$

Now, if the scalar product is computed up to a given component Y, one has:

$$\left|\mathbf{c}\cdot\mathbf{n}-\sum_{i=0}^{Y}\mathbf{c}_i\mathbf{n}_i\right| \leq |(\mathbf{n}_{Y+1},\ldots,\mathbf{n}_N)|\max(\{\mathbf{c}_k, Y<k\leq N\}) \tag{3.14}$$

Which in turn can be expressed as:

$$\left|\mathbf{c}\cdot\mathbf{n}-\sum_{i=0}^{Y}\mathbf{c}_i\mathbf{n}_i\right| \leq \left(\sqrt{1-\sum_{j=0}^{Y}\mathbf{n}_j^2}\right)\max(\{\mathbf{c}_k, Y<k\leq N\}) \tag{3.15}$$

This can lead to a significant efficiency improvement for many problems of practical interest. In fact, the components of $\mathbf{c}$ can be sorted and the scalar product can be computed in the new order.

Be $a(\mathbf{c})_i$ the number of the i-th component of $\mathbf{c}$ in decreasing order of magnitude. One can define:

$$\tau(\mathbf{c},\mathbf{n},k)=\left(\sqrt{1-\sum_{j=0}^{Y}\mathbf{n}_{a(\mathbf{c})_j}^2}\right)\mathbf{c}_{a(\mathbf{c})_{j+1}} \tag{3.16}$$

By the triangular inequality one has (be $\mathbf{b}$ a choice versor):

$$\mathbf{c}\cdot\mathbf{n}-\mathbf{b}\cdot\mathbf{n}\leq\left(\sum_{i=1}^{Y}\mathbf{c}_{a(\mathbf{c})_i}\mathbf{n}_{a(\mathbf{c})_i}\right)-\left(\sum_{i=1}^{W}\mathbf{b}_{a(\mathbf{b})_i}\mathbf{n}_{a(\mathbf{b})_i}\right)+|\tau(\mathbf{c},\mathbf{n},Y)|+|\tau(\mathbf{b},\mathbf{n},W)| \tag{3.17}$$

So if

$$\left|\left(\sum_{i=1}^{Y}\mathbf{c}_{a(\mathbf{c})_i}\mathbf{n}_{a(\mathbf{c})_i}\right)-\left(\sum_{i=1}^{W}\mathbf{b}_{a(\mathbf{b})_i}\mathbf{n}_{a(\mathbf{b})_i}\right)\right|>|\tau(\mathbf{c},\mathbf{n},Y)|+|\tau(\mathbf{b},\mathbf{n},W)| \tag{3.18}$$

One has:

$$\left|\left(\sum_{i=1}^{Y}\mathbf{c}_{a(\mathbf{c})_i}\mathbf{n}_{a(\mathbf{c})_i}\right)-\left(\sum_{i=1}^{W}\mathbf{b}_{a(\mathbf{b})_i}\mathbf{n}_{a(\mathbf{b})_i}\right)\right|>0\implies\mathbf{c}\cdot\mathbf{n}-\mathbf{b}\cdot\mathbf{n}>0 \tag{3.19}$$

Since the choice is determined only by the highest scalar product with versors, and not by its value, being Y_i the development of the scalar product of the environment vector with the i-th choice versor one can proceed as follows:

- If a copule of choice versors $\mathbf{c}_i, \mathbf{c}_j$ satisfies condition (3.18), delete the one with the lower partial scalar product: by (3.19) the one with the lower partial scalar product is never going to be the one with the highest scalar product at the end, and there is no need to compute the scalar product any further.

- If no couple of choice versors satisfies (3.18), develop one more term of the i-th scalar product, where i satisfies:

$$(\mathbf{c_i})_{Y_i}=\max\left(\left\{(\mathbf{c_j})_{Y_j}, \mathbf{c_j}\text{ has not yet been deleted}\right\}\right) \tag{3.20}$$

3.2.4 Fingerprinting

An efficient way to significantly improve performace in change of a small chance of random errors not determined by the choice agent is to preprocess data with a fingerprinting function. A fingerprinting function maps with continuity an higher dimensional environment vector in a lower dimensional one. Continuity assures that “similar vectors” are mapped in “similar vectors” by the fingerprinting function. However, the different number of dimensions of domain and codomain spaces for the fingerprinting determine the possibility of also two “significantly different vectors” being mapped to “similar vectors” by the fingerprinting, determining errors in the input of the choice agent.

A good fingerprinting function is the one minimizing the probability of these *fingerprint matches* happening on environment vectors of pratical interest.

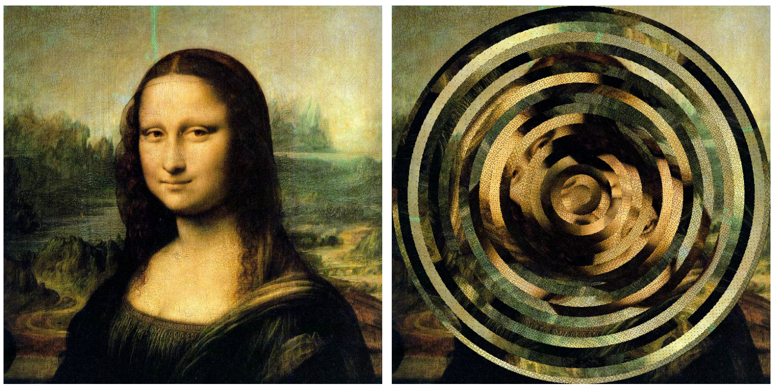

Figure 3.9: These two images will have the same radial fingerprint. However, fingerprint matches are very unlikely to happen for cases of practical interest.

As an example, we will introduce the *radial sampling* as a good fingerprinting function for image analysis. As we have said in the previous chapters, an image can be described by three matrices, one per color component (red, green, blue).

A radial fingerprint with a given bin width b for a given color component is defined as the vector:

$$F_b^{(r)}(M_{r,g,b})_i = \frac{\sum_{(j,k),(i-1)b\leq\sqrt{j^2+k^2}<ib} M_{ij}}{\left|\left\{(j,k),(i-1)b \leq \sqrt{j^2+k^2} < ib\right\}\right|} \tag{3.21}$$

That is, the i-th component represents the average over the i-th circular crown of internal radius radius $(i-1)b$ and external radius ib of the intensity of that component. This has the advantage of being rotation independent: two images that differ only by a rotation will have the same radial sample fingerprint.

3.3 Multipole expansion

Choice agents are mathematical objects that map environment vectors into choices. We have seen that, for a linear ansatz to the choice space, the problem reduces to the determination of a set of choice versors maximizing a fitness function, which we report again:

$$\Phi_i = \frac{\sum_{(\mathbf{x},\sigma_i)\in T_i} \left(\frac{(\mathbf{x}\cdot\mathbf{y}_i)}{\sqrt{\sum_{j=0}^{|S|}(\mathbf{x}\cdot\mathbf{y}_j)^2}} \right)}{|T_i|} \tag{3.22}$$

It has also been proved that this fitness function, defined for the single choice, can be maximized by the limit of the succession (2.24).

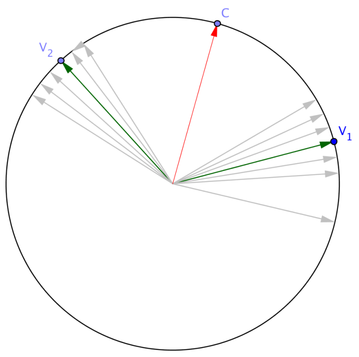

Figure 3.10: Bipole expansion for a bimodal distribution of environment training vectors. Versor **C** would be the single choice versor, $\mathbf{V_1}$ and $\mathbf{V_2}$ are the two virtual choice versors maximizing the fitness function on the splitting.

Once all choice versors have been determined, the answer to each environment vector will be determined by the *closer* choice vector on the sphere.

Now, let's take as an intuitive example a three dimensional environment space, and the training set vectors for a given choice be half localized closely around the $\hat{z}$ axis, and half closely around the $\hat{x}$ axis. Succession (2.24) will likely converge somewhere between the two. Therefore, even if Φ_i is going to be maximized for the single choice by that limit, its value is not going to be very high.

But if one introduces two virtual choices, both with the same meaining when it gets to interpret them, but with two distinct choice versors to be determined separately, and splits the original training in set in two complementary subsets, the first with all and only the elements localized closely around the $\hat{z}$ axis, the second with the ones around the $\hat{x}$ axis, Φ_i will boost significantly.

In a more formal way, we are modifying the ansatz for the choice function as a whole adding a new composition. In its most general expression:

$$f_{\mathbf{k}} = \tilde{\gamma} \circ \tilde{\sigma} \circ \tilde{f}_{\mathbf{k}} \circ \tilde{p}_d \tag{3.23}$$

Where now $\tilde{\sigma}$ maps from a more highly dimensional virtual choice space to a bigger virtual choice set, that will be compressed to the actual choice set by $\tilde{\gamma}$, that will map each virtual choice to an actual choice. $\tilde{\gamma}$ is obviously injective only when no additional virtual choices are introduced, and the virtual choice space reduces to the choice space.

A new fitness function can be defined, still on the actual choices. Being S the set of actual choices and $V_i = \tilde{\gamma}^{-1}(\sigma_i) = \{v_{in}\}$ the set of virtual choices corresponding to the i-th choice, T_{in} the training set for the n-th virtual choice in V_i, and $\mathbf{y}_{in}$ the choice versor corresponding to the virtual choice v_{in} the new fitness function can be expressed as:

$$\Phi_i = \sum_{n=1}^{|V_i|} \frac{\sum_{(\mathbf{x},v_{in})\in T_{in}} \left(\frac{(\mathbf{x}\cdot\mathbf{y_{in}})}{\sqrt{\sum_{j,k}(\mathbf{x}\cdot\mathbf{y_{jk}}}} \right)}{|T_{in}|} \tag{3.24}$$

If we consider again the previous example, introducing two virtual choices and splitting the training sets as described determines two virtual choice vectors, one along the $\hat{z}$ axis and the other along the $\hat{x}$ axis. This significantly increases the fitness function, since the scalar product in the numerator is going to increase while the scalar product between the environment vectors in the first training set and the second virtual choice versor are going to be perpendicular, so the denominator is not going to be significantly affected.

In general, virtual choices should be considered as an alternative solution to increasing the polynomial degree, since both time and space complexity are linear in the number of virtual choices.

The determination of the training subsets for virtual choices, however, are not trivial. The splitting should be done to maximize the total fitness function.

Since the number of ways one can split a training set in two or more subsets is $|V_i|^{|T_i|}$, an explicit exploration of all the combinations is in general computationally unfeasible.

Approximate methods however exist, based on the iteration of a potential minimization procedure that can be described as follows:

- At the beginning, elements in the original training set are randomly split between the virtual training subsets. Virtual choice versors are determined, and Φ_i is computed on the splitting as in (3.24) .
- Be n the number of virtual choices. Each item can be moved in $n-1$ other virtual training subsets. Be t the number of items in the original training set. There are $t(n-1)$ ways of modifying the present configuration by moving one item from its virtual training subset to another. Each of these operations leads to a new configuration, on which the fitness function can be evaluated as in (3.24) upon determining the new choice versors. We will refer to these sets as *reachable configuration*
- Once the fitness function has been evaluated on all reachable configurations, the one with the highest fitness is selected.
- If the best reachable configuration has a lower fitness than the previous, a local maximum has been determined. Else, the best reachable configuration is selected and the process is iterated until completition.

Note that this is an evolutive method, but applied to a discrete problem. The efficiency of this iteration can be improved by considering the act of moving an element from one virtual subset to another as a perturbation of the corresponding virtual choice versors. Therefore the values of the choice versors of the two subsets exchanging an element can be used as first elements of the succession iterating the contraction (2.24).

3.4 Single choice problems

Some problems can be described as two choice problems (a.k.a. emphyes-or-no problems), but no training set can be exhibited for one of the two choices. We will see how the main example in this thesis can be described by this paradigm.

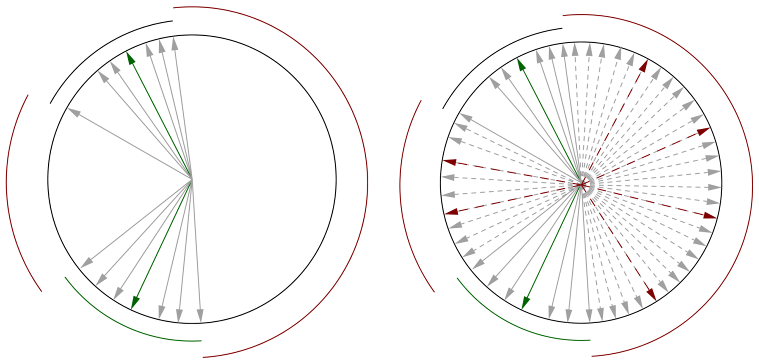

Figure 3.11: In a one choice problem there is a choice with an empty training set. However, training vectors could be generated to cover up all the choice sphere, and then multipole expanded.

Consider, for example, the problem represented in figure 3.11. Training vectors are provided for choice 1 (here, two-pole expanded by the green virtual choice versors), while choice 2 is defined as the other's negative. If training versors are not distributed over the whole sphere, training elements for choice 2 could however be generated to cover up all the the choice sphere's surface that is free from the training versors of choice 1. The newly generated poles could then be multipole expanded, and virtual choice versors would be obtained for choice 2.

An important issue to be considered, however, is the definition of "surface free from training versors of choice 1". In general, training sets do not cover the whole surface of the sphere linked with the choice they belong to, that is, $f(\mathbf{x}) = \sigma_i \not\Rightarrow \mathbf{x} \in T_i$ (if this happened, f would be obviously defined by its training set). It is not trivial, then, to set a threshold, on the sphere, between the choices. Note that that threshold is otherwise implicitly determined by the two training sets.

An approximate solution can be statistically determined as follows:

- Choice 1 is multipole expanded (if necessary) and its (virtual) choice versors are determined.
- For each element in each virtual training subset, the scalar product with the corresponding virtual choice versor is determined.
- Remember that, on a finite dimensional sphere, the scalar product can be interpreted as the cosine of the angle between two versors:

$$\mathbf{x} \cdot \mathbf{y} = \cos(\theta_{\mathbf{xy}}) \tag{3.25}$$

- For each virtual training subset, then, the dispersion beween the training versors can be determined by evaluating the angular standard deviation between the virtual choice versor and the elements of its virtual training subset:

$$\sigma_\theta = \sqrt{\frac{\sum_{i=0}^{|T_{1n}|} \left(\cos^{-1}(\mathbf{x_i} \cdot \mathbf{y_{1n}})\right)^2}{|T_{1n}|}} \tag{3.26}$$

σ_θ can be used as a parameter to evaluate the probability that an environment versor belongs to choice 1. Assuming that versors for each virtual choice versor are normally distributed, one has (be $\mathbf{x}$ an environment versor and $\mathbf{y}$ the virtual choice versor maximizing $\mathbf{x} \cdot \mathbf{y}$):

$$p^{(1)}(\mathbf{x}) = e^{\frac{\left(\cos^{-1}(\mathbf{x}\cdot\mathbf{y})\right)^2}{2\sigma_\theta^2}} \tag{3.27}$$

Where $p^{(1)}$ represents the probability density that $\mathbf{x}$ belongs to choice 1. Note that this is no longer a deterministic solution. We will see a practical usage for this in the next chapter.

Chapter 4

Medical image diagnostics

4.1 Microcalcifications

Breast calcifications are small areas of calcium that can be found in female breast. While macrocalficitations are in general harmless, microcalcifications can be a sign for a pathologic cell replacement rate. Since high cell replacement rates could be determined by cancer, women over a certain age should be screened yearly for microcalcifications.

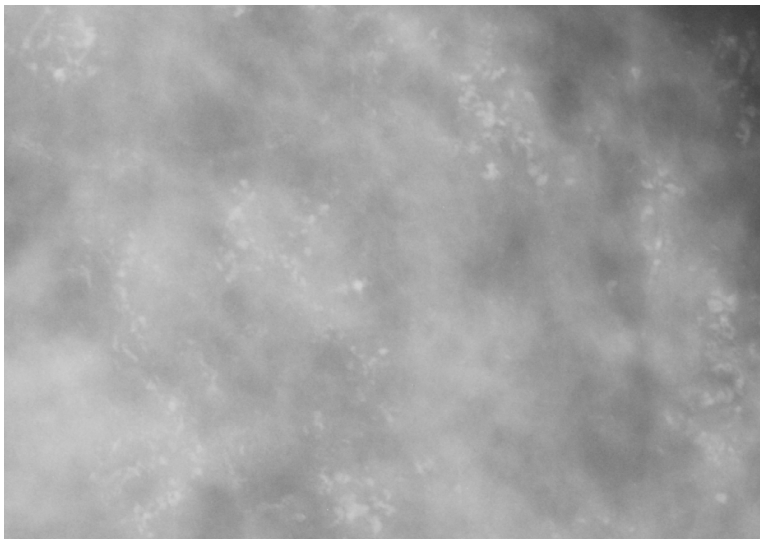

Figure 4.1: A cluster of microcalcifications

Since calcium is radiodense, microcalcifications appear as small white spots on a radiography (figure 4.1). The analysis of radiography is manually done by the radiologist. As an example application for non-evolutive pattern recognition techniques, a simple CAD (computer aided detection) system has been developed.

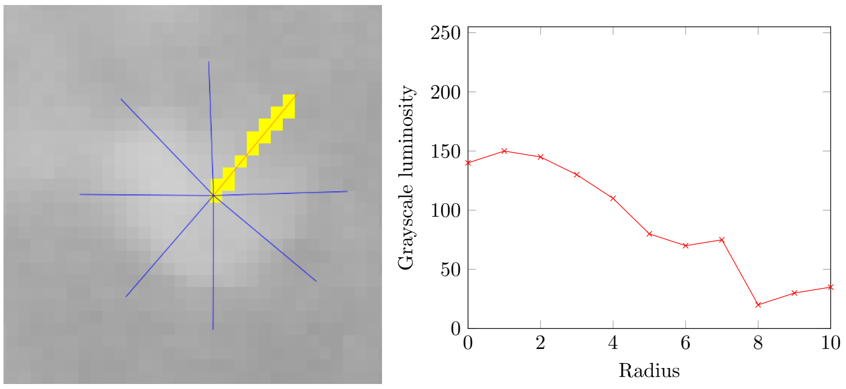

Figure 4.2: Example radial sampling. Pixels marked in yellow will be sampled when sampling along the axis marked in orange. An example set of sampling values is shown on the plot on the right.

4.2 Input selection and training set

All the images processed were obtained from the *Digital Mammography Database* of the University of South Florida. This database contains high-definition images for both normal breasts and breasts with cancer. All the images of breasts with cancer come along a meta file with the coordinates of every zone of lesions.

However, the dimension of lesions significantly varies between different images. A scanning procedure is therefore needed like in the optical character recognition example, to detect single microcalficitations instead of zones containing microcalcifications.

Since a microcalcification appears as a small, brighter dot on the mammography, and since it has an approximate radial simmetry, grayscale radial sampling was chosen as an efficient fingerprinting method. Note that fingerprint matches coincide in this case with actual image matches: two samples with a similar fingerprint will both represent small, white dots.

Due to the high variability of both microcalcification sizes and image luminosity, the radial fingerprinting method was modified to make it also scale independent and intensity independent. Based on the resolution of the radiography machine (information are included in metafiles of each image in the database) an upper bound to a microcalcification radius was empirically set to 20 pixels.

Given an (x, y) integer coordinate of a pixel, a radial sample is a 20-components vector determined as follows:

- An angle step is empirically determined so that every pixel on the most external circular crown of the sample is accounted for.
- For each angle step θ, a raw sample $\rho^{(\theta)}$ is taken as follows:

  $$\left(\rho^{(\theta)}\right)_i = I(x + i\cos\theta, y + i\sin\theta) \tag{4.1}$$

  Where $I(x, y)$ assesses the grayscale luminosity of the image in (x, y).

  Note that there is no need for $x + i\cos\theta$ and $y + i\sin\theta$ being integers, since each pixel (x, y) can be considered as an open set $[x, x + 1[\times[y, y + 1[$ over

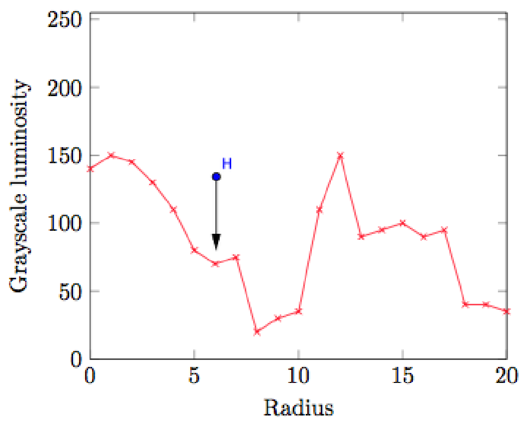

Figure 4.3: An example on how the heuristics method to find the cut position works on a sample. The point marked with "H" is the cut position

which I is constant (figure 4.2).

- A raw round sample is then defined as the sum over all the angle steps of each raw radial sample:

$$\rho = \sum_{\theta} \rho^{(\theta)} \tag{4.2}$$

- The round sample obtained is now the average over circular crowns of increasing size of the grayscale luminosity of the image. However, since 20 pixels is in general an overshoot for a microcalcification radius, even if (x, y) represents the center of a microcalcification the sample will be affected by surrounding image elements on its most external circular crowns. Samples should be cut at the outer border of the microcalcification, i.e. at the first minumum of luminosity from the center. A simple heuristic method has been implemented to determine a cut position c:

$$c = \min\{i \in [1, 19[\} : \rho_i < \rho_{i+1}, \rho_i < \rho_{i-1}, (\rho_0 - \rho_i) > \frac{\max_j \rho_j - \min_j \rho_j}{\delta} \tag{4.3}$$

  Where δ is an empirical parameter efficiently performing when set to 5.

  The cut position (figure 4.3) is therefore defined as the first minimum lower than the first element of the sample by more than one fifth of the amplitude of the sample. Adding the requirement of ρ_i being lower than the first element by more than one fifth of the amplitude makes the method more resilient to noise. Note that such c could not exist, since the condition in (4.3) is not said to be satisfied by any i. If this happens, (x, y) are not centered on a bright dot in the image and further tests can be skipped.

- Due to the fixed dimensionality needed for environment vectors, an interpolation function λ is defined on ρ:

$$\lambda(x, \rho) = \rho_{[x]} + (\rho_{[x]+1} - \rho_{[x]})(x - [x]) \tag{4.4}$$

where $x \in \mathbb{R}$ and $[x]$ represents the integer part of x. This function linearly interpolates the values of the sample between two components. ρ can now be resampled keeping unvaried the number of its components:

$$\rho^*{}_i = \lambda\left(\rho, \frac{ic}{20}\right) \tag{4.5}$$

- The luminosity independence condition can now be satisfied by introducing a normalized round sample defined as follows:

$$(\tilde{\rho})_i = 2\frac{\rho^*{}_i - \min_j \rho^*{}_j}{\max_j \rho^*{}_j - \min_j \rho^*{}_j} - 1 \tag{4.6}$$

  Note that this condition linearly transforms each round sample forcing its components values to lie between -1 and 1.

This procedure generates a 20-components vector that describes in a scale and luminosity independent way the average value of luminosity of a circle centered in (x, y). A training set is now needed. For this purpose, an image has been manually analized, and 50 microcalcifications have been highlighted by placing a red pixel at the center of them. The image has then been scanned for these pixels, and the procedure described above has been run on them to determine training set elements, from which a single choice versor has been computed.

This scanning problem represents a single choice problem, since no training set has been generated for non-calcifications. As described in section 3.4, an angular standard deviation has been determined on the training set elements.

4.3 Image analysis

Now that a choice versor has been defined to represent the "radial luminosity shape" of a disk centered in (x, y), it is possible to scan the image. For each pixel of the image, the disk centered on it has been radial sampled and its angular deviation from the choice versor has been computed. A score has been then associated to every pixel based on the gaussian probability density of three variables: the luminosity shape angular deviation, the size of the calcification (that is, the value of the cut position c) and the luminosity amplitude. The average and standard deviation for the last two variables have been determined on the training set elements.

$$S_{xy} = \exp\left(-\frac{(\theta_s)^2}{2\sigma_\theta^2} - \frac{(c-\bar{c})^2}{2\sigma_c^2} - \frac{(l-\bar{l})^2}{2\sigma_l^2}\right) \tag{4.7}$$

Where θ_s is the luminosity shape angular deviation from the choice versor, σ_θ is the angular standard deviation for the luminosity shape, c and $\bar{c}$ are the local and average cut positions respectively, σ_c the standard deviation for the cut position, l and $\bar{l}$ the local and average luminosity amplitude, and σ_l the luminosity amplitude standard deviation.

This algorithm, however, poses an issue. If only the average luminosity shape factor is considered, filaments of proper thickness can be detected as dots. Remember that the average luminosity shape factor is given by the sum of several radial samples. Now, for a sampling centered on a dot in a filament, most of the raw radial samples will cross the filament, therefore resulting in a luminosity shape very similar to the one of a dot. The few radial samples along the filament are characterized, however, by a nearly constant luminosity, therefore their influence on the round sample is filtered by the normalization procedure.

This can be worked around by introducing a new term to the score. Note that the heuristic procedure, described in the previous section, to determine a cut position is going to fail if applied to a nearly constant luminosity sample. Since that procedure can be run also on single raw radial samples, a new parameter n has been introduced as the fraction of raw radial samples on which the heuristic procedure to determine a cut position fails. Once the average $\bar{n}$ and the standard deviation σ_n have been computed on the training set, the score has been redefined:

$$S_{xy} = \exp\left(-\frac{(\theta_s)^2}{2\sigma_\theta^2} - \frac{(c-\bar{c})^2}{2\sigma_c^2} - \frac{(l-\bar{l})^2}{2\sigma_l^2} - \frac{(n-\bar{n})^2}{2\sigma_n^2}\right) \tag{4.8}$$

This score expresses the probability density of (x, y) being the center of a microcalcification.

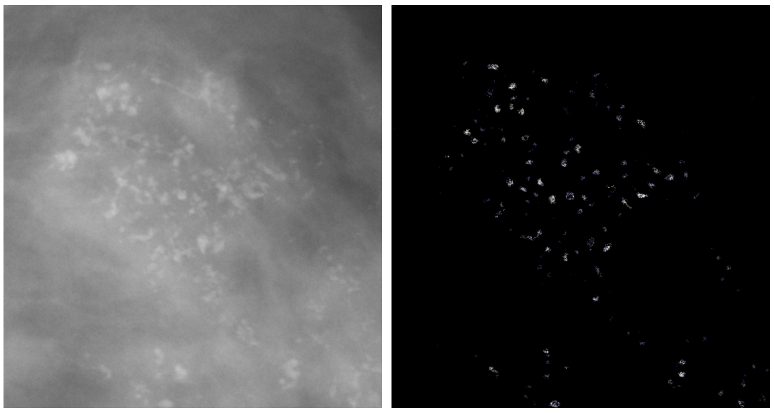

Figure 4.4: Two crops from the same area of a lesioned portion. Note the correspondence between spots on the right (score) and calcifications on the left image.

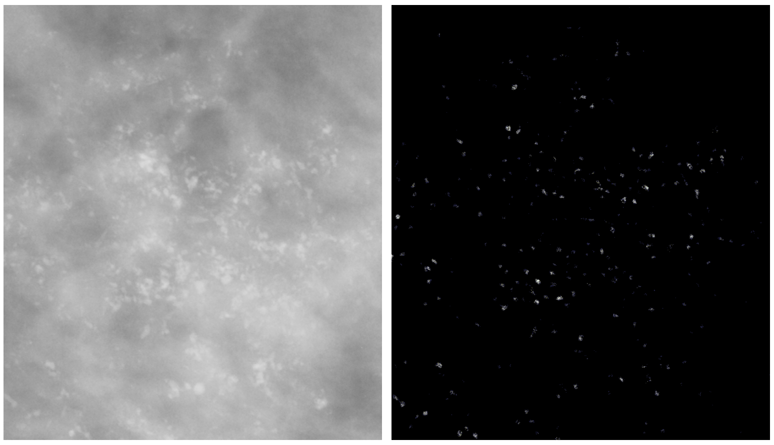

Figure 4.5: Two different, bigger lesions

4.4 Scanning results

The scan described in the previous section has been run over several images. For rendering purposes scores have been renormalized, so that the highest score is always represented as white and the null score is represented as black. Several tests were run both on normal and cancer images.

However, since no information could be extracted from the Digital Mammography database on the exact position of single calcifications, no significant tests could be done to assess the efficiency of this algorithm.

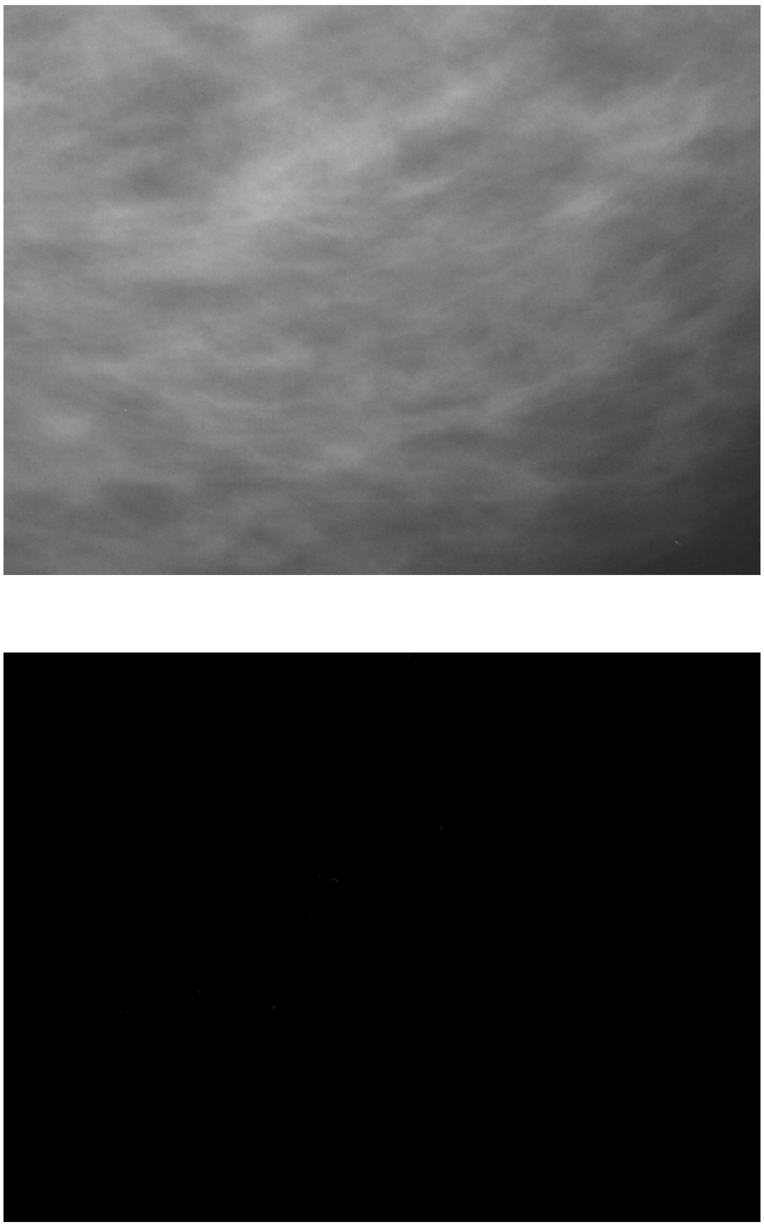

Figure 4.6: A portion of sound tissue. Note how no score is displayed.

4.5 Conclusions

Non-evolutive artificial intelligence training has proved to be an efficient, noise resilient, computationally light procedure to tackle a wide variety of problems. Moreover, its generality, along with the fact that it derives from a solid mathematical base, makes it more trustworthy than techniques based on empirical knowledge of the problem. The simplicity of the various examples shown proved it to be also applicable with nearly no programming effort.

All the research on the subject was brought on by the author on his own, nonetheless some significant results were obtained. This branch of non-evolutive artificial intelligence training has just been opened, but proved itself prolific enough to be explored more in depth.