Elements for Learning and Optimizing Expensive Functions

Philippe Rolet

▶ **To cite this version:**

Philippe Rolet. Elements for Learning and Optimizing Expensive Functions. Computer Science [cs]. Université Paris Sud - Paris XI, 2010. English. NNT : . tel-00551865

HAL Id: tel-00551865
https://theses.hal.science/tel-00551865

Submitted on 6 Jan 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : ?

Thèse
de
L'Université Paris–Sud

présentée en vue de l'obtention du
Doctorat de l'Université Paris–Sud
Spécialité : Informatique

Par
Philippe Rolet

Équipe A&O, Laboratoire de Recherche en Informatique (LRI),
U.M.R. CNRS 8623
Université Paris-Sud, 91405 Orsay Cedex, France

Éléments pour l'Apprentissage et l'Optimisation de Fonctions Chères

sous la direction de Michèle Sebag et Olivier Teytaud

Soutenue le 22 décembre 2010 devant la commission d'examen :

M.	Jean-Yves AUDIBERT	École des Ponts/Paris-Tech	Rapporteur
M.	Guillaume DEFFUANT	CEMAGREF	Président du jury
M.	Damien ERNST	Université de Liège	Examinateur
M.	Claudio GENTILE	Universita' dell'Insubria	Rapporteur
Mme	Michèle SEBAG	CNRS	Directrice de thèse
M.	Olivier TEYTAUD	INRIA	Directeur de thèse

Résumé

Ces travaux de doctorat sont centrés sur l'apprentissage artificiel et l'optimisation, c'est à dire la construction de programmes apprenant à identifier un concept, à approximer une fonction ou à trouver un optimum à partir d'exemples de ce concept (ou de points de la fonction).

Le contexte applicatif est l'apprentissage et l'optimisation de modèles simplifiés en ingénierie numérique, pour des problèmes industriels pour lesquels les exemples sont coûteux à obtenir. Il est nécessaire d'en utiliser le moins possible pour l'apprentissage; c'est le principe de l'apprentissage actif et de l'optimisation de fonction chères.

Mes efforts de recherche ont d'abord porté sur la conception et le développement d'une nouvelle approche de l'apprentissage Actif, fondée sur l'apprentissage par renforcement. Les fondements théoriques de l'approche ont été établis. Parallèlement, l'implémentation d'un logiciel fondé sur cette approche, BAAL, a permis une validation expérimentale (publications: CAP'09, ECML'09). Une extension de cette approche a été réalisée pour l'optimisation de fonction chères (publication: GECCO 2009).

La deuxième partie de mon doctorat s'intéresse aux potentiels et aux limites de l'apprentissage actif et de l'optimisation chère d'un point de vue théorique. Une étude des bornes de complexités de l'apprentissage actif par "paquets" a été réalisée (publication: ECML 2010). Dans le domaine de l'optimisation bruitée, des résultats sur le nombre minimal d'exemples nécessaires pour trouver un optimum ont été obtenus (publications: LION 2010, EvoSTAR 2010).

Remerciements

The work presented in this thesis was built upon a vast amount of financial, academical and emotional support. I believe it would be unfair not to thank the Digiteo research park for the grant that funded this thesis, and more specifically the various institutions belonging to Digiteo that contributed to address the financial needs of this work: the Paris-Sud University, the French Atomic Enery and Alternative Energies Commission (CEA), the National Research Institute for Computer Science (INRIA), the French National Center for Scientific Research (CNRS), the Ile-de-France region and the Pattern Analysis, Statistical Modeling and Computational Learning network of excellence (PASCAL).

I would like to warmly thank my referrees, Jean-Yves Audibert and Claudio Gentile, and the jury members Daniel Ernst and Guillaume Deffuant, for kindly accepting to review this research work and attend the thesis presentation. Since the rest of the thanks I would like to convey are destined to French speakers, I will write them in French.

Je remercie tout d'abord Michèle Sebag, qui a accepté de diriger cette thèse, avec qui j'étais fort heureux de travailler, et qui m'a apporté beaucoup de bons conseils et une aide précieuse durant ces trois ans.

Je remercie très chaleureusement Olivier Teytaud, qui a largement contribué à l'encadrement de cette thèse en tant que co-directeur. C'est un excellent chercheur, de qui j'ai beaucoup appris, et sans qui ce travail aurait indéniablement été de qualité bien moindre.

Merci également à Jean-Marc Martinez, qui a lui aussi participé à la co-direction de ces travaux.

Je remercie aussi Gilles Dowek du LIX, qui a été pour moi un très bon professeur, qui m'a beaucoup aidé à choisir mon orientation durant mon cycle polytechnicien, et qui m'a entre autres dirigé vers Michèle Sebag et l'équipe TAO du LRI.

Merci ensuite aux nombreux membres de l'équipe qui ont contribué à la bonne ambiance dans laquelle j'ai pu effectuer ma thèse. Merci notamment à Romaric, avec qui j'ai beaucoup discuté, et avec qui j'ai passé deux très bonnes semaines à l'île de Ré à ~~nager~~ ~~faire des balades~~ ~~chanter autour du feu~~ étudier l'apprentissage artificiel. Merci à l'éminent représentant des doctorants au conseil de labo, Ludovic, à Fabien, à Jean-Marc et à Pierre pour les trolls du midi (d'ailleurs, la GPL c'est pire qu'Hitler et HADOPI va sauver la création artistique). Merci à Nataliya, qui a su résister à mes tentatives de la convaincre que le MacDo c'est génial. Merci à Cédric pour tous ses mouhahaas. Merci à l'autre Cédric pour ses opinions sur la durée du trajet LRI/Montparnasse (qui valent approximativement un café). Merci à Raymond pour tous ses insights sur les gadgets de Cupertino. Merci aussi à la Mogo team (Hassen, Arpad, JB, Fabien bis), à Julien (plus ou moins Mogo team aussi d'ailleurs), à Xiangliang, à Dimo et Anne à qui je souhaite d'avoir passé un très bon voyage en Thaïlande, à Iliya, à Alvaro et Hassen (encore) pour les bons moments à Venise, à Jacques, à Mohamed, à Miguel, à Marc, à et à tous les autres qui font que la vie à TAO est vraiment sympa.

Merci aux élèves et à l'équipe enseignante de l'IUT d'Orsay (et en particulier à Cédric), j'ai vraiment aimé travailler avec vous.

Merci aussi à tous ceux qui sont venus (parfois de loin) pour assister à ma soutenance.

Enfin, un très grand merci bien sûr à mes amis François, Jean-Noël, Etienne, Rocaille, Ben, Matthieu, Clémence, Émilie, Gilles, Clotilde, Antoine, Vincent, Amélie et tant d'autres, à Anne, François et Antoine, mes frères et soeurs, et surtout à mon père et à ma mère, qui m'ont donné et qui continuent à me donner un soutien absolument indispensable pour accomplir tout projet d'envergure tel que celui-ci.

Elements for Learning and Optimizing Expensive Functions

Philippe Rolet

December 22, 2010

Abstract

This work focuses on learning and optimizing expensive functions, that is constructing algorithms learning to identify a concept, to approximate a function or to find an optimum based on examples of this concept (resp. points of the function).

The motivating application is learning and optimizing simplified models in numerical engineering, for industrial challenges for which obtaining examples is expensive. It is then necessary to use as few examples as possible for learning (resp. optimizing).

The first contribution was the conception and development of a new approach of active learning, based on reinforcement learning. Theoretical foundations for this approach were established. Furthermore, a learning algorithm based on this approach, BAAL, was implemented, and used to provide experimental validation.

The approach, originally focused on machine learning, was also extended to optimization.

The second contribution is focused on the potential and limits of both active learning and expensive optimization, from a theoretical point of view. Sample complexity bounds were derived: 1/ for batch active learning; 2/ for noisy optimization.

Contents

Part I

Preliminaries

Chapter 1

Introduction

"Guess the number" is a classical game that almost anyone played at least once as a kid. The rules are basic: a player chooses a number in-between a given range and the other tries to find the number in a minimal number of yes/no questions. One can easily figure out an optimal strategy for this game: divide the space in which the number lies in two equal parts, ask whether the number is in a given part, and repeat the process.

Expensive problems

This simple game bears the characteristic of *expensive* problems that are going to be studied by the present work: an objective must be reached by querying information from an "oracle", and the solution should be found by using as few queries as possible. Of course, strategies for the problems that will be encountered in the manuscript will not be as straightforward as for "Guess the number".

Queries can be deemed "expensive" for various reasons. Here are the most common:

- obtaining answers requires a lot of time. This is often the case when answers are provided by numerical engineering computational codes. Consider for instance the design of plane wings. The quality of a wing design is often assessed by simulating flights with planes using the wing design. Such simulations rely on fluid mechanics equations that are usually dealt with finite element methods: they may require the simulation software to run for hundreds of hours on high-performance computers to output the result of a single simulation;

- obtaining answers requires human effort (such as in "guess the number"). The primary purpose of computers is to relieve humans from doing tedious tasks; in software programming and web programming, developers aim at minimizing the need for human input, which usually bothers users. A well-known example is movie preference learning, made popular by Netflix

(2006): a solution would obviously not be practical if it needed users to give more that a couple dozens of examples of movies they like or dislike;

- obtaining answers is in fact literally expensive. In the automobile industry, for instance, design parameters that are the safest for a car are ultimately assessed by crash tests, which cost lots of money to set up and perform.

In this work, an expensive problem will be embodied by a function, that we either wish to discover or to optimize. The request of the image of a point of the function domain is what will be called a *query*. Learning a function from point/image pairs is part of a branch of machine learning called *statistical learning*. In many statistical learning settings, the pairs are a given: there is no ability to choose which points will be queried (this is passive learning). Nevertheless, when dealing with expensive learning problems, it is common to turn to *active learning*, and to assume that the learner has the ability to select the points that may be queried. Therefore, the sub-field of machine learning concerned with expensive problems often roughly coincides with active learning. In optimization, however, it is almost always assumed that the algorithm is responsible for choosing the points to query. This manuscript will be focused on active learning and expensive optimization.

Learning and optimization

One might wonder why topics that might at first seem distinct, function approximation and function optimization, are addressed side-by-side in this work. Here are some reasons that suggest to tackle both topics simultaneously.

Problem nature

First of all, in the present manuscript, the information available to solve the problem, be it an optimization problem or a learning problem, will be provided in the same way: by querying the images by the objective function of various points. Either the whole function, or the optimum, must be guessed from a set of pairs of points and their images (the sample). The goal is also the same: perform the task with the minimum *sample complexity*, namely the minimum number of queries.

Solving techniques

Multiple strategies to solve the kind of problem we are interested in can be applied for both learning and optimization. For instance, Gaussian processes have been used for regression tasks (Williams and Rasmussen, 1996) as well as for optimization problems (Jones et al., 1998; Villemonteix et al., 2008). Another example will be provided in this manuscript: the same Monte-Carlo tree search strategy will be used to solve both active learning and optimization problems.

Relationships between learning problems and optimization problems

It is not uncommon that learning techniques are used to solve optimization problems and vice-versa. For instance, a widely used optimization heuristic is "surrogate optimization": approximations of an objective function are learned in the neighborhood of the position where the optimum is believed to be, and optimization is performed on these "surrogates" rather than on the true objective function (Booker et al., 1999). Conversely, at the core of one of the most famous machine learning algorithm, support vector machines, lies an optimization problem.

Purpose statement and structure

The purpose of this contribution is to

- give some theoretical insights regarding the best achievable sample complexity and possible strategies, thus emphasizing the potential of active learning and expensive optimization, as well as their limits;
- propose tractable methods to close up to those limits as much as possible;
- present some experiments validating the quality of those methods.

Chapter 2 will introduce background and notations that are common to both learning and optimization, along with general mathematical tools used throughout the manuscript. Despite the close relationships between those topics that were described above, the topics are not identical; as such, the manuscript is divided in two parts, one dedicated to learning (part II) and the other to optimization (part III).

Part II begins by providing background knowledge specific to learning (chapter 3). Then, chapter 4 presents a formulation of active learning as a *Markov decision process* (this concept is introduced in chapter 2), and proposes an algorithm to approximate the optimal solution yielded by this formulation. Chapter 5 is interested in *batch* active learning, that is active learning when multiple queries can be made simultaneously. This is for instance the case when a computational code can be run on multiple computers at once. The theoretical benefits of such a framework in terms of sample complexity are studied. Finally, experiments validating the claims of chapters 4 and 5 are presented in chapter 6.

Part III also starts by giving the formal background specific to optimization, in chapter 7. The two following chapters, chapters 8 and 9, derive lower and upper bounds in the number of queries necessary to reach a certain precision for optimizing expensive functions in the presence of noise. These bounds rely on a formalism involving multi-armed bandits (introduced in chapter 2) and races (introduced in chapter 7). Chapter 10 concludes the part by confronting experimental behaviors of noisy expensive optimization algorithms to the theoretical bounds, and presenting further empirical evidence of the adequacy of bandits and Monte-Carlo tree search approaches to expensive noisy optimization.

Chapter 2

General Framework and Background

In this chapter, formal background and notations common to both learning and optimization are presented (section 2.1). Two probabilistic frameworks relevant to the following research work are then introduced: Markov decision processes (section 2.2) and multi-armed bandits (section 2.3).

2.1 Learning and Optimization Problems

To introduce the formal background used throughout this manuscript, let us consider a problem that was first presented in the previous chapter, relevant to both learning and optimization goals: the design of good airplane wings. This canonical problem can be formalized as follows. Let $\mathcal{X}$ denote a set of shapes, let quality function f^* be defined on $\mathcal{X}$, mapping a shape to its quality in terms of flight abilities, assuming further that $f^* : \mathcal{X} \to \mathcal{Y} = [0, 1]$. The general function f^* mapping a shape to its value is unknown. But given a particular shape x of $\mathcal{X}$, it is possible to compute its value $f^*(x)$ via a numerical simulator (or by actually building the wing).

A natural engineering problem, known as *optimal design*, would be to find the best wing shape, and hence to find a good approximation of the maximum x^* of this unknown f^*, given some couples $(x, f^*(x))$. The engineering team might also want to get a global overview of how each parameter influences the quality of the wing shape: this requires to try and learn an approximation of f^* on the set of parameters. This would be a statistical learning problem.

Generally speaking, both problems can be defined by the following ingredients:

- a "search space" $\mathcal{X}$;
- a "value space" $\mathcal{Y}$;

- a set $\mathcal{F}$ of mappings from $\mathcal{X}$ to $\mathcal{Y}$ and a "target" mapping f^* from this set;
- a set of pairs $(x, f^*(x))$ of points of $\mathcal{X}$ and their image in[1] $\mathcal{Y}$;

The set $\mathcal{F}$ to which f^* belongs encompasses the prior knowledge available about f^* so that the problem may be solved more easily. For example, fluid mechanics physicists might tell the engineering team that f^* is continuous, and then $\mathcal{F} \subset \mathcal{C}([0,1])$.

f^* may be referred to as *fitness function* or *objective function* in an optimization context, and *target hypothesis* or *target concept* in a statistical learning context.

Besides, given n in $\mathbb{N}$, a set of couples $s = \{(x_1, f^*(x_1)), \ldots, (x_n, f^*(x_n))\}$ generated from $\{x_1, \ldots, x_n\} \subset \mathcal{X}$ by f^* is available. This *sample*, or *training set*, belongs to $\mathcal{S}_n \doteq (\mathcal{X} \times \mathcal{Y})^n$. The *sample space* is the set $\mathcal{S} \doteq \bigcup_{n \in \mathbb{N}} \mathcal{S}_n$.

Useful Notations

In the following, $P_{\mathcal{S}}$ denotes a probability measure on set $\mathcal{S}$, while $p_{\mathcal{S}}$ denotes the corresponding densities. A conditional probability measure on space $\mathcal{X}$ depending on spaces $\mathcal{Y}$ and $\mathcal{Z}$ is written $P_{\mathcal{X}|\mathcal{Y}\mathcal{Z}}$. Unless otherwise stated, density-based formulas used in the manuscript hold for discrete probabilities. It is thus assumed that all probability measures admit a density. *Dirac measures* can act as densities for discrete probability measures when required. Given a point a of a continuous set $\mathcal{S}$, the Dirac measure δ_a is defined for subsets $\mathcal{T}$ of $\mathcal{S}$ by

$$\delta_a : \mathcal{T} \mapsto \begin{cases} 1 & \text{if } a \in \mathcal{T} \\ 0 & \text{otherwise.} \end{cases}$$

As a notation, Dirac measures will be used as densities for discrete probability measures: we will use δ_a as if it were defined on elements s of $\mathcal{S}$ and write

$$\delta_a(\mathcal{T}) = \int_{s \in \mathcal{T}} \delta_a(s) ds.$$

$\mathbb{E}[X]$ is the expectation of random variable X. $\mathbb{E}_{X,Y,\ldots}[.]$ is the expectation operator with respect to random variables $X, Y, \ldots$ applied to an expression containing those random variables. Similarly, $Var_{X,Y,\ldots}[.]$ is the variance of an expression containing those random variables.

The ith element of a vector $\mathbf{x}$ will be written $\mathbf{x}_i$. If $x, y \in \mathbb{R}^d$, then the notation $[x, y]$ refers to $\{a \in \mathbb{R}^d; \forall i, x_i \leq a_i \leq y_i\}$.

The following notations will also be used:

- $\forall (a, b) \in \mathbb{R}^2$,
  - $]a, b] \doteq [a, b] \setminus \{a\}$,

[1] or a mechanism for generating such pairs

  - $[a, b[\doteq [a, b] \setminus \{b\}$,
  - $]a, b[\doteq [a, b] \setminus \{a, b\}$;

- $\forall \mathbb{S} \subset \mathbb{R}$,
  - $\mathbb{S}^+ \doteq \mathbb{S} \bigcap [0, \infty[$,
  - $\mathbb{S}^- \doteq \mathbb{S} \bigcap]-\infty, 0]$.

The cardinal of a finite set $\mathcal{S}$ will be noted $|\mathcal{S}|$. log refers to the logarithm to the base 2.

Let us also remind Landeau notations o, O, Ω, Θ, used on functions of the integer variable n, to describe their asymptotic properties. As $n \to \infty$,

$f(n) = o(g(n))$ indicates that $f(n)/g(n) \to 0$;

$f(n) = O(g(n))$ indicates that there exists a constant $K > 0$ such that $f(n) < Kg(n)$;

$f(n) = \Omega(g(n))$ indicates that there exists a constant $K > 0$ such that $f(n) > Kg(n)$;

$f(n) = \Theta(g(n))$ indicates that there exists two constant $K_1, K_2 > 0$ such that $K_1 g(n) < f(n) < K_2 g(n)$.

The same notations may also be used when f, g depend on a continuous variable ϵ, with the same meaning except that $\epsilon \to 0$ instead of $n \to \infty$. Furthermore, tilded notations $\tilde{o}, \tilde{O}, \tilde{\Omega}, \tilde{\Theta}$ refer to similar asymptotic properties within logarithmic factors (instead of constant factors only).

2.1.1 Algorithms and samplers

Solving a learning problem (respectively an optimization problem) is finding a good approximation of the function f^* (respectively of the optimum x^*) given the sample s. In the following, a learning or optimization algorithm is noted $\mathbf{A}$ ($\mathbf{A} : \mathcal{S} \to \mathcal{F}$ for a learning problem and $\mathbf{A} : \mathcal{S} \to \mathcal{X}$ for an optimization problem). The goal of a computer scientist addressing such problems is to design algorithm $\mathbf{A}$ such that if s has been generated by f^*, then $\mathbf{A}(s)$ is close either to f^* or to x^*. Algorithm $\mathbf{A}$ is here defined deterministically, but it could also be seen as a probability distribution on $\mathcal{F}$ or $\mathcal{X}$—in the learning case, with $\mathbf{A} : \mathcal{S} \times \mathcal{F} \to \mathbb{R}^+$ and $\forall s, \int_{\mathcal{F}} \mathbf{A}(s, f) df = 1$. The set of all learning (or optimization) algorithms is noted $\mathcal{A}$. Formalizations of what it is for $\mathbf{A}$ to be close to its goal will be given in section 3.1 for learning and section 7.2 for optimization.

Furthermore, the sample s used by the algorithm can be a given set, over which the problem solver has no control. It can also be drawn from a probability distribution $P_{\mathcal{X} \times \mathcal{Y}}$, or it can be constructed by the problem solver itself if it has access to an oracle providing $f^*(x)$ when it is queried on the value of x. In general, it can be said that the sample is generated by a *sampler* $\mathbf{S} : \mathcal{S} \times \mathcal{X} \to \mathbb{R}^+$

that[2] maps each sample to a probability density on $\mathcal{X}$, and an *oracle* that takes as input the output x of the sampler and returns $f^*(x)$. As for learners, deterministic samplers directly map $\mathcal{S}$ to $\mathcal{X}$.

Noting X_s the random variable whose law is given by density $\mathbf{S}(s,.)$, this means that random samples S_{n,f^*} of size n are constructed recursively as follows:

$$\begin{cases} S_{0,f^*} & = \emptyset \\ S_{n+1,f^*} & = S_{n,f^*} \bigcup \{(X_{S_{n,f^*}}, f^*(X_{S_{n,f^*}}))\} \text{ for } n > 0. \end{cases} \tag{2.1}$$

Passive settings are settings in which the sampler is a given to the problem solver; *active learning* and *expensive optimization* are settings defined by the assumption that the problem solver is responsible for designing $\mathbf{S}$ in addition to $\mathbf{A}$. Thus, active learners and expensive optimizers are noted as a pair $\mathbf{L} = (\mathbf{A}, \mathbf{S})$. Since the way of sampling the set $\mathcal{X}$ is left to the problem solver, better performance can be hoped for.

2.1.2 Performance, sample complexity and expensive functions

To measure how good a proposed solution to a learning or optimization problem is, a function to measure the proximity of such solution to the actual target is needed. Definitions of such a function, called ρ in the following, are different for learning and for optimization. Therefore actual definitions of ρ are given in chapter 3 for learning and chapter 7 for optimization.

Once ρ is decided upon, the quality of a learning or optimization algorithm can be measured. For instance, for learning problems, ρ would map $\mathcal{F}^2$ to $\mathbb{R}^+$. Given a sample s generated from f, an algorithm $\mathbf{A}$ will perform well if $\rho(\mathbf{A}(s), f)$ is small. Generally, a sample s of size n is constructed by the sampler $\mathbf{S}$ using f: it is a random variable $S_{n,f}$ lying in $\mathcal{S}_n$. Algorithm $\mathbf{A}$'s output is a random variable $\mathbf{A}(S_{n,f})$ depending on this sample. Therefore, the performance of $\mathbf{A}$ and $\mathbf{S}$ is measured probabilistically. Given ϵ in $\mathbb{R}^+$ (a precision) and δ in $]0,1[$ (a confidence), a common measure of the performance of $\mathbf{L}$=($\mathbf{A}$,$\mathbf{S}$) is the minimum required sample size ensuring that with probability 1-δ, $\mathbf{A}$ outputs an approximation whose distance to f^* measured by ρ is less than ϵ, for *any* function of $\mathcal{F}$. Formally, the function *sc* is defined by:

$$sc : (\mathbf{L}, \epsilon, \delta, \mathcal{F}) \mapsto \min\{n_0 \in \mathbb{N} \bigcup \{+\infty\} \mid \\ \forall f \in \mathcal{F}, n > n_0, P_{\mathcal{S}_n}(\rho(\mathbf{A}(S_{n,f}), f) > \epsilon) < \delta\}.$$

sc is the *sample complexity* of $\mathbf{L}$ on $\mathcal{F}$ at precision ϵ and confidence δ. According to the above definition, if $\mathbf{L}$ cannot learn some f of $\mathcal{F}$ at precision ϵ with confidence δ, no matter how big the sample is, then $sc(\mathbf{L}, \epsilon, \delta, \mathcal{F}) = +\infty$. The sample complexity of a function class $\mathcal{F}$ can also be defined by $sc(\epsilon, \delta, \mathcal{F}) = \min_{\mathbf{L}} sc(\mathbf{L}, \epsilon, \delta, \mathcal{F})$. This is a probabilistic definition of *sc*. A more stringent

[2] or equivalently, $\mathbf{S} : \mathcal{S} \to (\mathbb{R}^+)^{\mathcal{X}}$

definition is sometimes used if the sampler is deterministic ($S_{n,f}$ is not a random variable any more):

$$scm : (\epsilon, \mathcal{F}) \mapsto \min_{\mathbf{L}} \min\{n \in \mathbb{N} \bigcup \{+\infty\} \mid \max_{f \in \mathcal{F}} \rho(\mathbf{A}(S_{n,f}), f) < \epsilon\}.$$

This definition will be referred to as *minimax* sample complexity.

The definition also applies to optimization problems, replacing $\rho : \mathcal{F}^2 \to \mathcal{R}^+$ by $\rho : \mathcal{X}^2 \to \mathbb{R}^+$ and so on—although the term "sample complexity" originally stems from computational learning theory. Some other ways to refer to sample complexity are the *number of fitness calls* in optimization and the *number of queries to the oracle* in statistical learning.

Expensive functions

It is often common in computer science to assess an algorithm's performance by measuring its *runtime complexity*: the number of computing steps it takes to solve a problem of size n, as a function of n. In the formalism used here, $\mathbf{L}$ is an algorithm whose runtime complexity depends on the size n of training set s. However, for learning and optimization problems, sample complexity can be as important or more important than runtime complexity.

Many artificial fitness functions used in scientific experiments have a negligible computational cost, whereas real-world applications often involve huge computational cost. This cost can be due, for instance, to finite-element methods or (quasi-)Monte-Carlo sampling in numerical engineering problems. In such settings, one can often neglect the internal cost of the learning/optimization algorithm, and only consider the number of queries (a.k.a. fitness evaluations)—the fact that algorithms may require a few minutes before asking for the value of an example does not matter.

Therefore, the notion of sample complexity is quite relevant to this manuscript since the following work deals with *expensive* problems. An expensive problem is loosely defined as a problem for which, given x, obtaining $f^*(x)$ is costly—be it in time, money or human effort, as said in the introduction. In other words, what is meant by "expensive" throughout all this manuscript is the fact that the cost of obtaining $f^*(x)$ given x is important enough that the focus is set on sample complexity: runtime complexity is not considered at first.

Various parts of this work aim at finding *upper bounds* and *lower bounds* for sample complexities of various function classes. Formally, bounding a function class $\mathcal{F}$ is finding $lb : (\epsilon, \delta) \to \mathbb{N}$ and $ub : (\epsilon, \delta) \to \mathbb{N}$ such that

$$\forall (\epsilon, \delta) \in \mathbb{R}^+ \times]0, 1[, lb(\epsilon, \delta) \leq sc(\epsilon, \delta, \mathcal{F}) \leq ub(\epsilon, \delta).$$

The bounds are said to be tight if when $\delta \to 0$, $ub(\epsilon, \delta) = \Theta(lb(\epsilon, \delta))$.

2.1.3 Noisy problems

In the example described in section 2.1, the wing optimization and learning problems are *noise-free*, since given a wing shape x there is exactly one possible quality value y related to x, which is $f^*(x)$. However, in many real-world problems, it is more appropriate to consider *noisy* problems, where the outcome of f^* on x may vary.

For instance, let us assume that the value of a wing shape x is assessed by actually manufacturing the wing, and performing a flight with a plane using the wing. It is unlikely that for the same wing, two different flights output exactly the same value. Rather, the value observed will depend on some random external parameters and will not be the same twice: the observation is *noisy*. Thus, the wing quality y for a given shape x will not be deterministic ($y = f^*(x)$); instead, it will be a random variable Y^* whose law will be given by a probability density p^* that depends on x. Consequently, in a noisy problem, the fitness f^* and set $\mathcal{F}$ are replaced by a fitness density $p^*_{\mathcal{Y}|\mathcal{X}} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$, and a set $\mathcal{P}_{\mathcal{Y}|\mathcal{X}}$ of densities $p_{\mathcal{Y}|\mathcal{X}} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ that are conditional densities—i.e. every $p_{\mathcal{Y}|\mathcal{X}}$ satisfies $\forall x \in \mathcal{X}, \int_{\mathcal{Y}} p_{\mathcal{Y}|\mathcal{X}}(x, y)dy = 1$; therefore their values are noted $p_{\mathcal{Y}|\mathcal{X}}(y|x)$ (rather than $p_{\mathbb{Y}|\mathcal{X}}(x, y)$).

Every $x \in \mathcal{X}$ is then mapped to a random variable whose density is $p^*_{\mathcal{Y}|\mathcal{X}}(.|x)$: let us write $Y(x)$ the random variable accounting for the measurement of x. Noisy problems are often rephrased so that a deterministic function f^* appears anyway. The *target function* f^* is often defined as

$$\begin{aligned} f^* :& \mathcal{X} \to \mathcal{Y} \\ & x \mapsto \mathbb{E}[Y(x)]. \end{aligned}$$

If the elements of $\mathcal{P}_{\mathcal{Y}|\mathcal{X}}$ are Dirac densities—that is, if for each element of $\mathcal{P}_{\mathcal{Y}|\mathcal{X}}$, all the probability mass of the corresponding distribution is on a singleton—the problem degenerates to a noise-free problem. In this manuscript, part II will consider mostly noise-free learning problems, while part III will deal with both noise-free and noisy optimization problems.

2.2 Markov Decision Processes

A key contribution of the present work is to formalize both active learning and optimization as Markov decision processes. This formalization will further allow us to compute optimal solutions to finite-horizon learning and optimization problems. Therefore, this section will introduce Markov decision processes, and provide an overview of classical tools and results. Interested readers can refer to Sutton and Barto (1998), Bertsekas and Tsitsiklis (1996) for more thorough surveys.

2.2.1 Definition

Markov decision processes (MDPs) are a formalism used to model multistep processes in which an actor takes successive actions (decisions) leading him/her from a state to another, and pays a cost (or earns a reward) for each decision he/she takes.

Definition 1 (Markov decision process)**.** A Markov decision process consists of a quadruplet $(\mathcal{S}t, \mathcal{A}c, p, r)$, involving:

- a *state space* $\mathcal{S}t$;
- an *action space* $\mathcal{A}c$;[3];
- a *transition probability density function* $p : \mathcal{S}t \times \mathcal{A}c \times \mathcal{S}t \to \mathbb{R}^+$ satisfying $\forall s \in \mathcal{S}t, \forall a \in \mathcal{A}c, \int_{\mathcal{S}t} p(s, a, u)du = 1$;
- a reward function $r : \mathcal{S}t \times \mathcal{A}c \times \mathcal{S}t \to \mathbb{R}$.

This models the process of a decision maker in discrete time, starting from an *initial state*. At a given time step, the decision maker is in state s and chooses an action from the set of possible actions from s, which is a subset of $\mathcal{A}c$. At the next time step, he/she will be in a new state s', drawn randomly from the probability density $p(s, a, .)$—noted $p(.|s, a)$ since it refers to the probability density of reaching new states conditionally to previous state s and chosen action a. The decision maker then receives reward $r(s, a, s')$. A graphic representation of a MDP is given in figure 2.2.1.

It is straightforward to see that such a stochastic process satisfies the *Markov property*: the state at time step t only depends on the (state,action) pair at time $t - 1$, and does not depends on states and actions at time $t - 2$ and below.

Example

A typical example well-modeled by a Markov decision process is the problem known as *GridWorld*. This example is illustrated in figure 2.2.2. In its simplest version, the set of states are the possible positions on a $N * N$ grid. There are four possible actions: up, down, left and right—the four possible directions that an agent can take. Transitions are deterministic: given a state and a direction, the next state is the position on the grid obtained by moving one step in this direction, or by not moving if the direction points to an edge. The rewards are null, except when reaching "target" positions that have strictly positive rewards, and when moving in an edge, in which case a negative reward is given.

This is an illustrative example; MDPs are of course suited to model many real-world processes, such as for instance:

Dialogue systems An automated machine must interact with a user to acquire information and perform actions (e.g. interactive voice response for a

[3]some authors sometimes introduce for each s in $\mathcal{S}t$, a related action space $\mathcal{A}c_s$ with $\mathcal{A}c = \bigcup_s \mathcal{A}c_s$

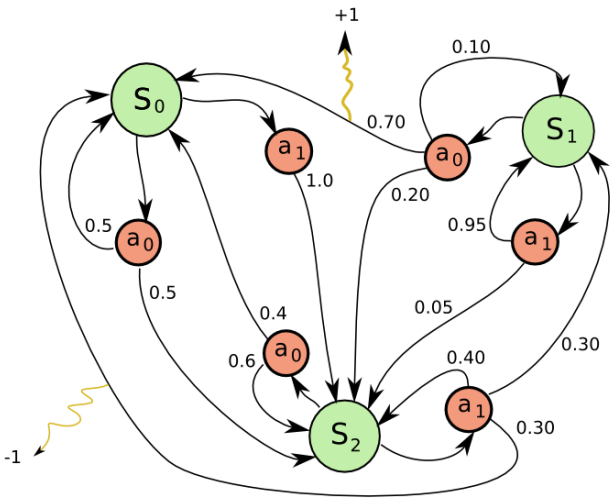


Figure 2.1: Example of a Markov decision process. States are represented in large green/ gray circles and actions in small red/gray circles. The number on an arrow leaving an action is the probability to reach the state to which the arrow goes. A yellow (or light gray) sinusoidal arrow indicates the reward for arriving in a state by taking a given action in a given previous state.

telephone banking system). A state contains the information acquired by the machine, what is left to ask and the current position of the machine in the conversation. An action can for example be an information request (e.g. credit card number) or a transaction (e.g. transferring money from one account to another); see for instance Levin et al. (1998).

Game playing The set of states correspond to different possible game configurations and the set of actions are the moves that the player can make. Markov decision processes have for instance been used for Backgammon by Tesauro (1992): the TD-Gammon algorithm is known as an example of the efficiency of TD (λ) algorithms in particular, and reinforcement learning in general.

2.2.2 Policies

Policies on MDPs can be either be deterministic, or stochastic. A *deterministic policy* for a MDP is a function $\pi : \mathcal{S}t \rightarrow \mathcal{A}c$ that describes which action to take at each state. Thus, a policy embodies the decision maker's behavior. A *stochastic policy* $\pi : \mathcal{S}t \times \mathcal{A}c \rightarrow \mathbb{R}$ gives a probability density to each action (that can be a Dirac density) from each state. Hence, π is a conditional probability,

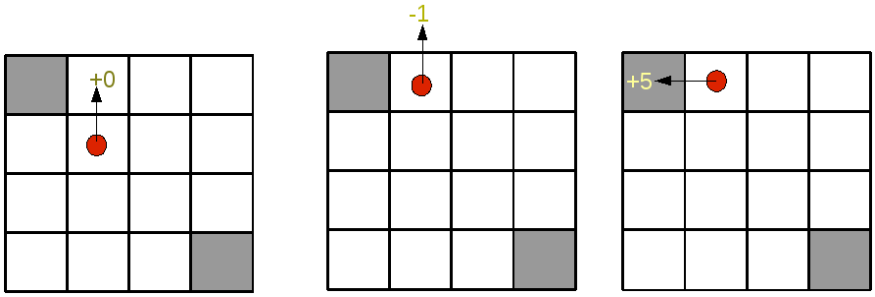


Figure 2.2: The GridWorld example with a 4*4 grid. The agent is in the position marked by the red/gray dot. His/her actions are modeled by black arrows. Rewards for each transition are the numbers pointed at by arrows.

satisfying

$$\forall s, \int_{a \in \mathcal{A}c} \pi(s,a) da = 1,$$

and as such it is written $\pi(a|s)$. Note that deterministic policies are a particular case of stochastic policies; this is why, unless otherwise stated, only stochastic policies will be considered in the following. Given an MDP $(\mathcal{S}t, \mathcal{A}c, p, r)$ and a policy π, let us write

$$\begin{cases} s_{0,\pi} & \text{the initial state,} \\ \forall t \geq 0, a_{t,\pi} & \text{the action randomly drawn from } \pi(.|s_{t,\pi}), \\ \forall t \geq 1, s_{t,\pi} & \text{the state randomly drawn from } p(.|s_{t-1,\pi}, a_{t-1,\pi}). \end{cases} \tag{2.2}$$

The corresponding random variables are noted $A_{t,\pi}$ and $S_{t,\pi}$. The fact that $A_{t,\pi}$ is a conditional random variable depending only on $S_{t,\pi}$ and that $S_{t,\pi}$ is a conditional random variable depending only on $S_{t-1,\pi}, A_{t-1,\pi}$ is a restatement of the Markov property of this process.

Joint probability measures $P^{(t)}$ and densities $p^{(t)}$ on $(\mathcal{A}c \times \mathcal{S}t)^t$ for all t are induced by this process, as follows:

$$p^{(t)}(a_{0,\pi}, s_{1,\pi}, \ldots, a_{t-1,\pi}, s_{t,\pi}) = \prod_{i=0}^{t-1} \pi(a_{i,\pi}|s_{i,\pi}) p(s_{i+1,\pi}|s_{i,\pi}, a_{i,\pi}). \tag{2.3}$$

Thanks to the Markov property, the transition density $p(s'|s,a)$ is obtained by marginalizing any $p^{(t)}$ on three consecutive variables s, a, s' and then conditioning on s and a (the policy density $\pi(a|s)$ is obtained similarly).

Also note that for any t_0, t_1 with $t_1 > t_0$, the density $p^{(t_0)}$ is the marginal density of the first $2t_0$ variables computed from $p^{(t_1)}$; Hence, those probabilities can all be referred to as a single measure, using a single notation (here p, omitting the index).

2.2.3 Value functions and action-value functions

Solving a MDP consists in finding a policy which maximizes some function of all the rewards obtained during the process—the *criterion* of optimality. It is standard to use the *discounted total expected reward* as a criterion:

$$R_\pi(s) = \sum_{t=0}^{\infty} \gamma^t r(S_{t,\pi}, A_{t,\pi}, S_{t+1,\pi}) \text{ given } S_{0,\pi} = s \tag{2.4}$$

where $\gamma \in]0, 1]$ is the *discount factor*. Of course, γ and r must be such that the series is convergent. Naturally, given an initial state s, a policy is then said optimal if it maximizes the *value function* $\mathbf{V}^\pi : \mathcal{S}t \to \mathbb{R}$, that represents how "rewarding" it is to be in a state s knowing that policy π will be used to decide on actions in the future.

Definition 2 (Value function)**.** The *value function* $\mathbf{V}^\pi$ of a policy π is[4]

$$\mathbf{V}^\pi : s \mapsto \mathbb{E}[R_\pi(s)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_{t,\pi}, A_{t,\pi}, S_{t+1,\pi}) | S_{0,\pi} = s\right]. \tag{2.5}$$

Other criteria sometimes appear in the literature (e.g. Auer et al., 2008). In chapter 4, for instance, a criterion suited to finite-time horizons is presented. Nevertheless, in most cases, these other criteria can be viewed as particular cases of equation 2.4 (the criterion used in chapter 4 will actually be rephrased in this respect); this is why only the discounted total expected reward will be considered in the rest of the manuscript.

Given the above definition of a value function, it can be shown that V^π satisfies an induction equation:

$$\begin{aligned}
\mathbf{V}^\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_{t,\pi}, A_{t,\pi}, S_{t+1,\pi}) | S_{0,\pi} = s\right] \\
&= \int_{(a,s') \in \mathcal{A}c \times \mathcal{S}t} \pi(a|s) p(s'|s,a) (r(s,a,s') \\
&\quad + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(S_{t,\pi}, A_{t,\pi}, S_{t+1,\pi}) | S_{1,\pi} = s'\right]) dads' \\
&= \int_{(a,s') \in \mathcal{A}c \times \mathcal{S}t} \pi(a|s) p(s'|s,a) \left(r(s,a,s') + \gamma \mathbf{V}^\pi(s')\right) dads' \\
&= \mathbb{E}_{A_{0,\pi}, S_{1,\pi}}[r(s, A_{0,\pi}, S_{1,\pi}) + \gamma \mathbf{V}^\pi(S_{1,\pi})].
\end{aligned} \tag{2.6}$$

This equation is called the Bellman equation (Bellman, 1957). For a given initial state and a given policy, there exists a unique function $\mathbf{V}^\pi$ satisfying this equation: the Bellman equation can then also be used as a definition of value functions.

[4]The expectation here is taken over all the random variable $A_{O,\pi}, S_{1,\pi}, A_{1,\pi}, S_{2,\pi} \dots$

In the same vein, it is also quite common to rely on an *action-value* function $\mathbf{Q}^\pi : \mathcal{S}t \times \mathcal{A}c \to \mathbb{R}$ whose purpose is the same as the value function, but that fixes an action in addition to a state:

$$\mathbf{Q}^\pi(s,a) = \mathbb{E}[R_\pi(s)|s_{0,\pi} = s, A_{0,\pi} = a].$$

Similarly to value functions, action-value functions $\mathbf{Q}^\pi$ also satisfy a form of the Bellman equation.

The goal is then to find an optimal policy π^* such that given an initial state s, $\mathbf{V}^{\pi^*}$ is maximal. This defines a unique optimal value function (that can however be achieved by more than one policy):

$$\mathbf{V}^*(s) = \max_\pi \mathbb{E}[R_\pi(s)].$$

In the Bellman inductive equation, this leads to

$$\begin{aligned}\mathbf{V}^*(s) &= \max_\pi \mathbb{E}[R_\pi(s)] \\ &= \max_\pi \mathbb{E}_{A,S}[r(s,A,S) + \gamma \mathbf{V}^\pi(S)] \\ &= \max_\pi \mathbb{E}_{A,S}[r(s,A,S) + \gamma \mathbf{V}^*(S)]\end{aligned}$$

where A is drawn randomly from $\pi(.|s)$ and S is drawn randomly from $p(.|s,A)$. In the last line, note that $\mathbf{V}^*(S)$ does not depend on the policy, and $\mathbb{E}_{A,S}[r(s,A,S)]$ only relies on the first action advocated by π from s. This yields the Bellman equation for the optimal value function:

$$\mathbf{V}^*(s) = \max_{a \in \mathcal{A}c} \mathbb{E}_S[r(s,a,S) + \gamma \mathbf{V}^*(S)].$$

Again, a similar equation can be derived for $\mathbf{Q}^*$, the optimal action-value function. This equation is central in reinforcement learning since it allows computation of $\mathbf{V}^*$ and π^* by backward induction, which is the cornerstone of dynamic programming algorithms.

2.2.4 A generalization: partially observable Markov decision processes

In some cases, the agent is not perfectly aware of the current state; rather, he/she partially observes the state. For instance, when using a Markov decision process to model a poker game, a current state of the game consist of the cards on the board, the remaining cards of all players, and for each player, which amounts of money have been bet so far. However, the agent only observes his/her cards, the board cards and the bets of other players, and knows not the cards that other players have.

Such a setting is referred to as a partially observable Markov decision process. partially observable Markov decision processes (POMDPs) involve two more ingredients than classical MDPs:

- a set of *observations* Ω;
- a conditional observation density $\omega : \Omega \times \mathcal{S}t \to \mathbb{R}$.

If the agent takes action a from state s, the state s' where he/she arrives will be drawn from $p(.|s, a)$. The agent will then see an observation o drawn from $\omega(.|s')$. Consequently, a policy in the case of POMDPs describe which action to take conditionally to observation o, rather than to state s as in classical MDPs. The reader interested in POMDPs may refer for instance to Kaelbling et al. (1998) for a more comprehensive introduction.

2.3 Multi-armed Bandits

The typical example for introducing the multi-armed bandit framework (Gittins, 1979; Auer et al., 2002) is the following. A player enters a casino where multiple slot machines (the so-called "one-armed bandits") are available to play. The player plays a machine by pulling its arm and getting a reward—which of course may be negative. Each machine has a fixed mean reward. At each time step, the player chooses a machine and plays it, receiving the reward. Since he/she wishes to maximize his/her gains (or minimize his/her losses), the selection of which machine to play has to be done carefully enough so that over time, machines that yield the best rewards are played more often. This is an example of a multi-armed problem. The following will present formal definitions and some well-known results in the literature, since the presented work partly relies on multi-armed bandit approaches.

2.3.1 Setting

Let us first define the multi-armed bandit framework with a finite number of arms. Consider K arms numbered from 1 to K. To each arm $a \in [[1, K]]$ corresponds a probability distribution P_a on $\mathbb{R}$. Arm a is a machine that outputs a value drawn randomly from P_a each time it is "pulled". Therefore, it is actually a collection of independent identically distributed (i.i.d.) random variables $(X_{a,i})_{i\in\mathbb{N}}$ whose laws are given by P_a, each corresponding to a reward that the arm can yield. When the arm is pulled for the ith time, it returns $X_{a,i}$ as reward. It is then natural to define the expected reward, and the empirical mean reward of a at step t:

$$\mu_a \doteq \mathbb{E}[X_{a,1}] \text{ and } \hat{X}_{a,t} \doteq \frac{1}{t}\sum_{i=1}^{t} X_{a,i},$$

with the obvious property that $\hat{X}_{a,t} \to_{t\to\infty} \mu_a$ almost surely. Similarly, the variance and empirical variance are

$$\sigma_a \doteq Var[X_{a,1}] \text{ and } \hat{V}_{a,t} = \frac{1}{t}\sum_{i=1}^{t}(X_{a,i} - \hat{X}_{a,t})^2, \tag{2.7}$$

The best possible expected reward is $\mu^* = \max_{a \in [[1,K]]} \mu_a$. The margin of an arm is measured by $\Delta_a = \mu^* - \mu_a$. The player is allowed a number of successive trials $N \in [[1, +\infty]]$. He/she decides on an arm-pulling strategy A: a mapping from previous arms pulls and their reward to the next arm to pull. To simplify notations, A is only written as a mapping from the current time step to the next chosen arm $A : i \mapsto A(i)$. It is also handy to define $T_a : i \mapsto T_a(i)$ mapping a time step i to the number of times arm a has been pulled until time i (included). Sometimes, the player is required to ultimately decide on the best arm after N pulls, using a recommendation strategy R. The framework is summarized on algorithm 2.3.1.

Algorithm 1 *Multi-armed bandit framework.*

Parameter Number of arms K>0
Parameter Number of allowed pulls $N \in [[1, \infty]]$
$t = 0$
while t<N **do**
  Pull arm $a = A(i)$
  Receive stochastic reward $X_{a,T_a(i)}$
  t++
end while
Return Arm chosen via recommendation rule R

The multi-armed bandit framework is commonly known for the *exploration-exploitation* tradeoff that it models: the player should pull the most rewarding arms according to what he/she observed (exploitation), but he/she should also pull arms that weren't pulled yet (exploration) or that were only pulled a few times, since those could have a high expected reward that could have been misestimated if the first pulls were unlucky. In other words, both estimates on arm values and confidences on these estimates should be used in an arm pulling strategy.

2.3.2 Problem goal

The strategy chosen by the player is based on previously observed rewards.

The goal is to pull as much as possible "best" arms whose expected rewards are close to μ^*. There can be several optimality criteria:

Cumulative regret

In the above example of a casino player, the obvious goal is to maximize the sum of rewards for all time steps. This is known as the *cumulative regret*.

Definition 3 (Cumulative regret)**.** The cumulative regret in a multi-armed

bandit problem at time step t is

$$R_t \doteq \mathbb{E}[t\mu^* - \sum_{i=1}^{t} X_{A(i),T_a(i)}] = t\mu^* - \sum_{a=1}^{K} T_a(t)\mu_a$$
$$= \sum_{a=1}^{K} T_a(t)\Delta_a.$$

Since the rewards $X_{a,i}$ are random variables, so are $A(i)$ and $T_a(i)$ for every i. The expectation on the first line is taken over the $X_{a,i}$.

Simple regret

There are many situations where at some point, an arm needs to be chosen for good, and only the performance of this arm matters. For instance, a company manufacturing a computer chip may have a trial phase during which various different sets of parameters for the chip (the arms) are tested under various conditions. Afterward, it starts a commercialization phase for which a precise set of parameters has to be decided on. In this case, what matters is the performance of this single arm, rather than the cumulative rewards of the trial phase.

Definition 4 (Simple regret)**.** The simple regret in a multi-armed bandit problem at time step t is

$$r_t \doteq \mu^* - \mathbb{E}[X_{A(t),T_a(t)}]$$
$$= \Delta_{A(t)}.$$

Here again, r_t is a random variable depending on the strategy and the observed rewards until t. It is then natural for the player to try to minimize $\mathbb{E}r_t$.

Yet another possible goal in finite multi-armed bandit problems is to maximize the probability that $r_t = 0$, i.e. that after N steps allowed for trial, an optimal arm is chosen.

2.3.3 Algorithms and theoretical results

As mentioned above, there are two kinds of strategies to consider in the multi-armed bandit framework: arm-pulling strategies and recommendation strategies.

Arm-pulling strategies

Multiple arm-pulling strategies have been devised in the literature, depending on the player's goal.

Random Uniform. A strategy that immediately comes to mind is the random uniform selection of arms. As its name suggests, it consists in picking a random arm in $[[1, K]]$ with identical probability weight on each arm. Although it is obvious that this choice is bad for minimizing cumulative regret, it has been shown to be an optimal strategy for minimizing simple regret asymptotically (Bubeck et al., 2008): the value of $\mathbb{E}r_t$ tends to the best achievable value for any strategy when $t \to \infty$ if the empirical best arm is selected after t rounds of uniform selection. However, Bubeck et al. (2008) point out that the random uniform strategy is empirically outperformed by other strategies (see below) even for t large, despite the fact that these are suboptimal in theory.

ϵ-greedy. The first widely used arm-pulling strategy, in which the tradeoff between exploration and exploitation is clear, is the so-called ϵ-greedy policy (Sutton and Barto, 1998): choosing the arm whose empirical mean is best with probability $1 - \epsilon$, and uniformly randomly picking another arm with probability ϵ. This strategy offers a cumulative regret that is quite better than the random uniform strategy, although only within a constant, since with a fixed probability ϵ it actually behaves like the random uniform strategy. Therefore, in order to improve on the asymptotic expected regret, ϵ_n-greedy strategies have been devised, where $\epsilon_n \to 0$ for $n \to \infty$. By carefully choosing ϵ_n, the cumulative regret is logarithmic in n (Auer et al., 2002). However, these strategies depend on multiple free parameters, some of which relate to properties of distributions of the arms which may be unknown.

Upper Confidence Bound (UCB). UCB is a method devised by Auer et al. (2002). It first requires an evaluation of all the arms once. Then, upper confidence bounds on the reward are computed at each round, and the arm with the largest bound is chosen. The simplest and most well-known policy derived from this idea is the following.

Definition 5 (UCB1)**.** The UCB1 strategy is the strategy that first pulls all arms once, and then select at round $t > K$ an arm $UCB(t)$ such that:

$$UCB(t) \in \arg \max_{a \in [[1,K]]} \left[\hat{X}_{a,T_a(t)} + \sqrt{\frac{2 \ln(t)}{T_a(t)}} \right]. \tag{2.8}$$

The first term in the right-hand side of equation 2.8 is the empirical mean of an arm at the current round: it favors exploitation. The second term decreases as the number of pulls of the arm increases: it favors exploration. The form of this term stems from simple statistical confidence bounds. More sophisticated bounds such as Hoeffding and Bernstein bounds (Hoeffding, 1963; Mnih et al., 2008) lead to better variations of equation 2.8.

It is common to add a multiplicative factor to the exploration term, used as a parameter to manually adjust the tradeoff between exploration and exploitation. However, this implies the necessity of tuning a new parameter, which might not be straightforward. To avoid this downside, the use of empirical variance,

in addition to empirical mean, in UCB formulas has been investigated. Auer et al. (2002) report that a formula named UCB-Tuned using empirical variance experimentally leads to better performance. Let us consider an upper bound on $\hat{V}_{a,T_a(t)}$, the variance of arm a after $T_a(t)$ pulls:

$$\bar{V}_{a,T_a(t)} = \hat{V}_{a,T_a(t)} + \sqrt{\frac{2\ln(t)}{T_a(t)}}.$$

Equation 2.8 becomes

$$UCB-Tuned(t) = \arg\max_{a\in[[1,K]]} \hat{X}_{a,T_a(t)} + \sqrt{\frac{\ln(t)}{T_a(t)}\min\{1/4, \bar{V}_{a,T_a(t)}\}}. \quad (2.9)$$

Even more sophisticated versions are presented by Mnih et al. (2008), with theoretical grounding. These will be detailed in chapter 7 as the part of this work concerned with expensive optimization relies on them.

This manuscript will make use of UCB-like policies for the following reasons:

- good theoretical and empirical behavior with respect to cumulative regret: logarithmic rate (Auer et al., 2002);
- good empirical behavior with respect to simple regret: (Bubeck et al., 2008) point out that although random uniform should theoretically perform better, they observe a faster rate with UCB-like policies;
- the fact that no parameter tuning, nor prior knowledge of the arms and their probability distribution, is required.

Recommendation strategies

This topic is not central in the present manuscript, therefore it is only reviewed briefly. A recommendation strategy is a decision rule for the definite choice of an arm after multiple pulls. It is necessary when one is interested in simple regret, i.e. what matters is the quality of the arm chosen after t pulls, and the sum of mistakes until then is irrelevant. Common strategies include (see Bubeck et al. (2008)):

Empirical best arm which is self explanatory: it selects the arm with best empirical mean reward when the player stops pulling arms;

Most pulled arm which is also self-explanatory. Note that it would make no sense to use this strategy on top of a random uniform arm-pulling strategy. It is however quite efficient when used with UCB-like strategies, since it the outcome will have a good empirical mean (otherwise it would not have been pulled a lot) with a good confidence (since the confidence directly depends on the number of pulls);

Empirical distribution of plays which selects an arm by randomly choosing between the arms, weighing the probability of an arm to be selected by its empirical mean reward.

Recommendation strategies are necessary when the algorithm is given a limit on the number of pulls, or a time limit that can be converted in a number of pulls, as in part II dealing with learning. In part III on expensive optimization, a complementary approach is taken: the algorithm is required to reach a certain confidence, and has to pull arms until this confidence is reached.

Part II

Learning Expensive Functions

Chapter 3

Learning and Active Learning: a Brief Survey

In this chapter, the framework and tools specific to statistical learning are introduced. The presented contribution in machine learning is mostly related to active learning. This chapter will however focus on passive learning (when the learner has no control on the training sample) since this setting, widely studied since the seventies, is the traditional one in computational learning theory.

The state of the art in active learning, investigated since the early nineties, will then be presented.

3.1 Statistical Learning Background

This section introduces a classical formalization of statistical learning known as PAC-learning, as well as major elements of learning theory, developed since the seventies, such as VC-dimension. This completes the definitions and formalization of chapter 2.

Denoting $\mathcal{X}$ the instance space (chapter 2), an element x of $\mathcal{X}$ is called an *input*, an *instance* or an *unlabeled example*. Hence, set $\mathcal{X}$ is also referred to as the input space or unlabeled example space. Instance x can be composed of multiple elements or coordinates—for instance if $\mathcal{X}$ is a subset of $\mathbb{R}^d$—which are usually called its *features*. It is also common to rely on a probability measure defined on $\mathcal{X}$, that will be noted $P_{\mathcal{X}}$ in the following.

The image of x by f^* is the *label* of x, and thus $\mathcal{Y}$ is the set of possible labels. If $\mathcal{Y}$ is finite or discrete, the learning task is called a *classification* task; if $\mathcal{Y}$ is continuous, the task is a *regression* task.

The presented work focuses on classification problems. Besides the basic definition and formalism presented in chapter 2, this section introduces the main two formal frameworks in statistical learning, namely the *probably approximately correct* framework (PAC) and the statistical learning theory based on

the Vapnik-Chervonenkis dimension (VC-dimension, see for instance Vapnik, 1995).

3.1.1 Hypothesis space, version space and realizability

Machine learning frameworks commonly consider a so-called *hypothesis space*[1], i.e. a set of functions $h : \mathcal{X} \to \mathcal{Y}$ considered for approximating target function f^* of $\mathcal{F}$. Elements of $\mathcal{H}$ are referred to as *hypotheses* or *classifiers*, since the focus is on classification tasks.

Definition 6 (Consistency of a hypothesis)**.** Let s be a sample generated from f^*. A hypothesis h is said to be *consistent* with s if

$$\forall (x, y) \in s, h(x) = y.$$

Note that in noisy learning cases, i.e. when the training sample involves non-deterministic noise, the notion of consistency with respect to the training sample is not always relevant.

In error-free scenarios, however, consistency is a more natural requirement, especially in the case where $\mathcal{H}$ equals $\mathcal{F}$, which amounts to assuming that the target concept is also a hypothesis. The *realizable* assumption relaxes the above, stating that for each f in $\mathcal{F}$, for all s generated by f there exists an h in $\mathcal{H}$ that is consistent with s—this might be the case even if $\mathcal{H} \neq \mathcal{F}$. An important concept in realizable situations is the *version space* of a training set s (Mitchell).

Definition 7 (Version Space)**.** The version space of s, written $\mathcal{H}(s)$ is the subset of $\mathcal{H}$ whose elements are the hypotheses consistent with s:

$$\mathcal{H}(s) \doteq \{h \in \mathcal{H} | \forall (x, y) \in s, h(x) = y\}.$$

Non-realizable scenarios appear when there might be some f generating some samples s with which no element of $\mathcal{H}$ is consistent, making the learning task more complex. Of course, for small sample sizes, this is less likely to happen.

3.1.2 Performance and generalization error

In statistical learning, performance function ρ (section 2.1.2) is used to measure the difference between two functions.

The performance function most often relies on some cost function, referred to as the *loss function* $\ell : \mathcal{Y}^2 \to \mathbb{R}^+$, where $\ell(h(x), y)$ gives the cost incurred by la belling some instance x with $h(x)$ instead of its true label x. Common definitions for ℓ can be

$$\begin{aligned} &l : (y, y') \mapsto \mathbf{1}_{\{y'\}}(y) \text{ or} \\ &l : (y, y') \mapsto \|y - y'\| \text{ if } \mathcal{Y} \text{ is hilbertian with norm } \|.\|. \end{aligned} \tag{3.1}$$

[1] sometimes referred to as the *concept class* in the classification literature

The latter is typical of regression problems where for instance $\mathcal{Y} = \mathbb{R}$, while the former, known as the $0-1$ loss, is a usual choice for classification problems.

Given a cost function ℓ, it comes naturally to define the performance ρ as the expectation of ℓ on $\mathcal{X}$: $(h, f^*) \mapsto \int_{\mathcal{X}} l(h(x), f^*(x))dx$, using probability measure $P_{\mathcal{X}}$ on $\mathcal{X}$ to compute the integral.

It is usual in statistical learning to assume the existence of such a probabilistic measure $P_{\mathcal{X}}$ on input space $\mathcal{X}$. Indeed, statistical learning problems are often phrased as *prediction* problems: given the training set s, the aim is to be able to predict the value of the next sample x that will appear.

Example: For instance, an online streaming music site may provide guesses about which music a user wants to listen to, given a record of previous choices $s = \{(x_1, f*(x_1)), \ldots, (x_n, f^*(x_n))\}$ where the x_i represent some user-specific characteristics (age, preferences) and probably some environmental features (time of day, songs playing at the time), and the $f^*(x_i)$ are songs that were chosen accordingly. The learning task is then, for the next x to come, to predict $f^*(x)$ as accurately as possible. Some x will show up more often (young users visit online streaming sites more frequently): it is more important to be accurate on those. From this perspective, the next instance to appear is a random variable that is sampled from probabilistic distribution $P_{\mathcal{X}}$ called the *natural distribution* of the input. This formulation of the learning task is just a rephrasal of what was presented in the last chapter—it just emphasizes the role of $P_{\mathcal{X}}$. □

The learning task might include various assumptions on $P_{\mathcal{X}}$, thereby defining a set $\mathcal{P}_{\mathcal{X}}$ of probability measures. If the goal is to be accurate on a single, definite $P_{\mathcal{X}}$, the task is called *fixed distribution* learning. The sample complexity sc of an algorithm becomes dependent on $P_{\mathcal{X}}$. At the opposite, if it is required that learning is good for every element of $\mathcal{P}_{\mathcal{X}}$, the learning task is said to be *distribution-free*, and sc is the maximum sample complexity taken on $\mathcal{P}_{\mathcal{X}}$. Only fixed distribution learning will be considered in the following.

Definition 8 (Generalization Error)**.** If X is a random variable whose law follows $P_{\mathcal{X}}$, the performance function ρ, called *generalization error* is:

$$\begin{aligned}\rho :& \mathcal{F}^2 \to \mathbb{R}^+ \\ & (h, f^*) \mapsto \mathbb{E}_X[l(h(X), f^*(X))] = \int_{\mathcal{X}} l(h(x), f^*(x))p_{\mathcal{X}}(x)dx. \end{aligned} \tag{3.2}$$

Exact computation of ρ is almost never possible: an *empirical error* is used instead.

Definition 9 (Empirical Error)**.** The empirical error on a finite sample s is:

$$\hat{\rho} : (h, f^*) \mapsto \frac{1}{|s|} \sum_{(x_i, f^*(x_i)) \in s} l(h(x_i), f^*(x_i)).$$

If s can be chosen randomly, the empirical error corresponds to a Monte-Carlo estimation of the generalization error. In a learning setting, the generalization error is estimated from the empirical error on a disjoint sample, called *test set*.

In the following, the considered loss function is the $0-1$ loss (equation 3.1), suited to classification tasks. In this case, ρ is a pseudo-metric on spaces $\mathcal{F}$ and $\mathcal{H}$, since l is symmetric and $l(y, y) = 0$.

Remark 1 (Non-symetric losses). Although symmetric losses are used quite often in computational learning theory, there are some cases in which a non-symmetric loss can be preferred. For instance, in medicine, an automated diagnostic tool may serve to screen patients arriving at an hospital. Diagnosing as sick a healthy patient is not too bad, because the actual doctor will sort it out when consulting. However, dismissing a sick patient may have dire consequences (for the patient and the hospital's reputation). The loss function should therefore account for this and emphasize false negatives.

Some appropriate loss functions and learning approaches have been derived (see for instance Bradley 1997) with some caveats however (see Hand 2010); these considerations are outside the scope of this manuscript.

3.1.3 Categories of learning algorithms and samplers

The probability $P_{\mathcal{X}}$ described above is the natural distribution of *unlabeled data*. A *learning algorithm*, in statistical learning, is an algorithm that is responsible for finding a hypothesis that best fits some *given* data. In the standard learning framework, referred to as passive learning, learning algorithms are not responsible for choosing the data—this task is done by *samplers* (a.k.a. sampling algorithms).

In the previous section a learning algorithm was described as a function $\mathbf{A} : \mathcal{S} \to \mathcal{F}$. Such algorithms pertain to a subfield of statistical learning known as *supervised learning*, that focuses on *labeled* training data.

Two subfields of statistical learning complementary to supervised learning focus on some applications of machine learning for which instance space $\mathcal{X}$, i.e. the space of *unlabeled* examples, may have a particular structure that can be exploited:

Unsupervised learning , where algorithms rely solely on unlabeled inputs to cluster the data (Duda et al., 2000, chap. 10);

Semi-supervised learning , where algorithms make use of unlabeled data along with labeled data, see for instance the work of Chapelle et al. (2006).

This manuscript is focused on learning algorithms that use only labeled data, i.e. supervised algorithms. It must be emphasized that although active learning often browses unlabeled data, active learning algorithms considered here do not fall into unsupervised or semi-supervised learning, since they do not rely on the internal structure of the instance space.

As we saw, a learning algorithm is often paired with a sampler $\mathbf{S}$ that outputs an element of $\mathcal{X}$. A sampler may access $\mathcal{X}$ in various ways:

- if unlabeled data comes as a finite pool—i.e. the support of the sampler's output distribution is this pool—the task is *pool-based*. It is however

often assumed that the pool was generated by an underlying probability distribution $P_{\mathcal{X}}$;

- if unlabeled data can be sampled at will from such a $P_{\mathcal{X}}$, the task is *population-based*;
- the sampler can also construct instances by itself, using for instance *generative models*, if its knowledge of $\mathcal{X}$ is good enough.

The degree of control that the problem solver has over the sampling part may also vary; from the higher to the lower level of control, we distinguish:

Active Learning if the problem solver is responsible for designing $\mathbf{S} : \mathcal{S} \to (\mathcal{X} \to \mathbb{R}^+)$, the sampling algorithm;

Design of Experiments (also known as *blind* active learning) if the problem solver can decide beforehand which points of $\mathcal{X}$ should be labeled, but cannot adapt, that is, it cannot decide to see some label *consequently* to other labeled points it has observed. Formally, the sampler does not depend on the labels and degenerates to a function $\mathbf{S} : \mathcal{X}^* \to (\mathcal{X} \to \mathbb{R}^+)$

Passive Learning if the problem solver has no choice over the points to be labeled: either labeled points come as a fixed pool, or they come from a given distribution. $\mathbf{S}$ is then a constant function that for any sample s returns the same probability distribution.

Sometimes the term "active learning" is used to refer to the first two settings as a whole, and *selective sampling* (or *adaptive learning*) refers to the very first setting itself.

3.1.4 PAC-learning

While a learner was formally defined as any algorithm from $\mathcal{S}$ to $\mathcal{H}$, it is clear that a learner picked randomly in the set of all these algorithms does not fit the machine learning agenda. It is necessary to have criteria to assess if a function qualifies as a viable learning algorithm. A classical requirement for a statistical learning algorithm is to be *probably approximately correct* (PAC). From the sample complexity (section 2.1.2), a *passive* sampler $\mathbf{S}$ returns an example drawn randomly from $P_{\mathcal{X}}$. The sample $S_{n,f}$ is then a set of independently identically distributed points of $\mathcal{X}$ their images.

An algorithm $\mathbf{L}$ built upon the passive sampler $\mathbf{S}$ is *probably approximately correct* on a function class $\mathcal{F}$ when, given an increasing number of random examples, it will eventually output with arbitrary confidence δ a hypothesis which is arbitrarily close to a target hypothesis from $\mathcal{F}$. Formally,

$$\forall \epsilon > 0, \forall \delta > 0, sc(\mathbf{L}, \epsilon, \delta, \mathcal{F}) < \infty. \tag{3.3}$$

Furthermore, a function class $\mathcal{F}$ is said to be *PAC-learnable* when there exists $\mathbf{L}$ that satisfies condition 3.3 for $\mathcal{F}$. Learnability has first been introduced by Valiant (1984); the interested reader is referred to the more comprehensive introduction of Vidyasagar (1997).

Example: Consider the set $\mathcal{C}$ of the binary linear classifiers of the d-dimensional unit cube ($\mathcal{C} \in \{0,1\}^{[-1/2,1/2]^d}$): it is the set $\{x \mapsto (sign(\mathbf{w}.x) + 1)/2 | \mathbf{w} \in \mathbb{R}^d\}$. Set $\mathcal{C}$ is PAC-learnable—informally, this can be understood visually since with an increasing number of random points, any hyperplane (represented by its normal vector $\mathbf{w}$) separating the space in positive and negative inputs is guessable with an increasing precision.

This contrasts with the set of all binary classifiers of $[-1/2, 1/2]^d$, i.e. $\{0,1\}^{[-1/2,1/2]^d}$ itself, which is not learnable. Indeed, for any set of points s, there are many functions that have the same image on s, though they differ significantly. The learner is then bound to return any one of those functions. □

Remark 2 (Alternative PAC definition). The definition of the PAC criterion is sometimes given using $\delta(\epsilon, m, \mathbf{L}, \mathcal{F})$, the confidence to which algorithm $\mathbf{L}$ can learn any function of $\mathcal{F}$ at precision ϵ given m sample—the algorithm is PAC if and only if $\forall \epsilon, \delta(\epsilon, m, \mathbf{L}, \mathcal{F}) \rightarrow_{m\rightarrow\infty} 0$. Both definitions (this one and the one given in equation 3.3) are equivalent.

3.1.5 Hypothesis space complexity

Among the many diverse supervised learning algorithms $\mathbf{A}$, some of the best known are neural networks, decision trees and support vector machines. Each algorithm explores a specific hypothesis space in order to find an approximation of the target concept.

Clearly, the choice of $\mathcal{H}$ and $\mathbf{A}$ should depend on learning tasks, and a wrong choice will affect the quality of the solution. Here is a common dilemma over the choice of $\mathcal{H}$. If $\mathcal{H}$ is "large", then many hypotheses fit the data well—and among those that fit the data well, some will be very different from each other: there is a high risk of picking an approximation h that fits the data sampled via f^*, but does not generalize well (i.e. does not have a good generalization error $\rho(h, f^*)$). On the other hand, if $\mathcal{H}$ is small, then the best hypotheses will probably only loosely fit the training set (high bias), and there will be a limit to how good the precision can be, but the generalization error will not be much worse than the (known) empirical error on the training set (low variance).

Two combinatorial parameters on hypothesis spaces $\mathcal{H}$ will now be described. Details on the results presented in this section can be found in Vidyasagar (1997, chap.2,4,6,7).

Vapnik-Chervonenkis dimension

The Vapnik-Chervonenkis dimension (VC-dimension, see for instance Vapnik, 1995) of a binary classification space is an indicator of how well the training error estimates the true error, the generalization error. Informally, it quantifies

how "complex" the hypotheses of $\mathcal{H}$ can be. Its mathematical definition relies on the concept of *shattering*.

Definition 10 (Shattering)**.** Let $\mathcal{H}$ be a set of binary classifiers $h : \mathcal{X} \to \{0, 1\}$. Let $\mathcal{T} = \{x_1, \ldots, x_n\}$ be a finite set of elements of $\mathcal{X}$ whose cardinal is n. It is said that $\mathcal{H}$ shatters $\mathcal{T}$ if

$$\forall \mathbf{y} \in \{0, 1\}^n, \exists h \in \mathcal{H}, \forall i \in [[1, n]], h(x_i) = (\mathbf{y})_i.$$

This basically means that whatever the labels of the points of the set $\mathcal{T}$ may be, there will be some hypothesis that is consistent will the labeled data set derived from $\mathcal{T}$.

Definition 11 (VC-dimension)**.** The VC-dimension of a hypothesis space $\mathcal{H}$ of binary classifiers is the cardinal of a set shattered by $\mathcal{H}$ of maximal size.

An important result is that any concept class with a finite VC-dimension D is PAC-learnable, with a sample complexity $sc(\epsilon, \delta) = \tilde{\Theta}(D/\epsilon)$: the number of random examples required to learn a concept at precision ϵ with a fixed confidence δ is at least inversely proportional to ϵ (upper bound), and not bigger than $1/\epsilon$ (lower bound) up to logarithmic factors.

A very strong result also states that conversely, in the general case of distribution-free learning, if a concept class is learnable, then it has a finite VC-dimension (this is not true for fixed distribution learning).[2]

Covering numbers

Another combinatorial parameter of a hypothesis space $\mathcal{H}$ is its *covering numbers*. Covering numbers appear in various sample complexity bounds regarding $\mathcal{H}$, in particular in the active learning context (Kulkarni et al., 1993). Intuitively, they quantify how "large" a hypothesis space is, when taking as distance between two hypotheses their generalization error (the function ρ). Remember that ρ is a pseudo-metric on $\mathcal{H}$. Let $\mathbf{B}(h_0, \epsilon) = \{h \in \mathcal{H} |\ \rho(h, h_0) \leq \epsilon\}$ stand for a (closed) ball centered on $h_0 \in \mathcal{H}$ of radius ϵ. ϵ-covers and covering numbers are then defined as follows.

Definition 12 (ϵ-cover)**.** An ϵ-cover of $\mathcal{H}$ is a finite set $\mathcal{T} \subset \mathcal{H}$ such that

$$\mathcal{H} \subseteq \bigcup_{h \in \mathcal{T}} \mathbf{B}(h, \epsilon).$$

Definition 13 (Covering numbers)**.** The ϵ-covering number of $\mathcal{H}$ is the smallest number $\mathbf{N}(\epsilon, \mathcal{H})$ such that there exist an ϵ-cover of cardinal $\mathbf{N}(\epsilon, \mathcal{H})$.

Similarly, the ϵ-packing number of $\mathcal{H}$, $\mathbf{M}(\epsilon, \mathcal{H})$, is the cardinal of a largest set $\mathcal{T}$ such that $\forall (h_1, h_2) \in \mathcal{T}^2, \rho(h_1, h_2) \geq \epsilon$ if $h_1 \neq h_2$. It is easy to show (see for instance Vidyasagar, 1997, chap. 2) that for any ϵ,

$$\mathbf{N}(2\epsilon, \mathcal{F}) \leq \mathbf{M}(\epsilon, \mathcal{F}) \leq \mathbf{N}(\epsilon, \mathcal{F}).$$

[2]The extension of VC-dimension to regression tasks is the Pollard dimension.

It has been shown (by Sauer, 1972, among others) that finite VC-dimension spaces have "small covering numbers", i.e. if the VC-dimension is D, the ϵ-covering number is $\tilde{O}(1/\epsilon^D)$. For fixed distribution learning, the fact that a concept class $\mathcal{F}$ is such that its covering numbers are all finite is equivalent to $\mathcal{F}$ being PAC-learnable.

Bayesian learning

By definition, a learning algorithm outputs hypotheses taken from some hypothesis space $\mathcal{H}$. In some learning settings, some knowledge about which hypotheses are more likely to be good is available a priori, that is before any example has been observed. A way to account for this information is to define a probability distribution $P_{\mathcal{H}}$ over $\mathcal{H}$, called a *prior* on hypotheses. This prior can then be multiplied with the probability of observing the given data for each hypothesis, to provide a *posterior* emphasizing which hypotheses are more likely given the data. This approach is known as *Bayesian learning* (see for instance Mackay, 1992).

Assuming a prior on hypotheses can also be used for other purposes, such as to perform *regularization*, that is to bias towards simpler hypothesis to avoid overfitting[3]. Note that regularization is often equivalently performed via other paradigms such as structural risk minimization (Vapnik, 1995, see,e.g. in support vector machines) or minimum description length (Duda et al., 2000, chap. 9)[4]. Chapter 4 will assume a prior $P_{\mathcal{H}}$ on the set of hypotheses.

3.2 State of the Art in Active Learning

This section reviews the main results obtained in active learning, starting with some formal background.

Let $\mathcal{X}$ be the input space for the learning task, and $\mathcal{H}$ be the hypothesis space.

Definition 14 (Uncertainty Region)**.** The *uncertainty region* $\mathcal{U}(s)$ of a sample s is the subset of $\mathcal{X}$ made of every input point which is mapped to different labels by some hypotheses of the *version space*:

$$\mathcal{U}(s) \doteq \{x \in \mathcal{X} | \exists (h_1, h_2) \in \mathcal{H}(s)^2, h_1(x) \neq h_2(x)\}.$$

A first critical requirement for any good active learning algorithm is that the sampler samples from points of the uncertainty region. Indeed, it would be pointless to try to ask for labels of points on which all hypotheses agree.[5]

[3] Overfitting happens when a learning machine with many parameters (very expressive hypothesis space) tends to adjust itself to the training data excessively. Therefore, it has a very low empirical error on the training set, but generalizes poorly on new instances.

[4] In many situations, these three regularization techniques can be proved equivalent to each other.

[5] In a non-realizable scenario, it might be worthwhile to try and find points on which all hypotheses agree, albeit differ from the target concept. This topic however remains outside the scope of this manuscript.

Measuring the size of the version space Under the Bayesian assumption, a probability $P_{\mathcal{H}}$ on the hypothesis space is available. The size of the version space of s can then naturally be defined as $P_{\mathcal{H}}(\mathcal{H}(s))$. As an alternative, i.e. if no such $P_{\mathcal{H}}$ is available, the version space is measured via its diameter, defined as

$$\Delta(s) = \max_{h_1, h2 \in \mathcal{H}(s)} \rho(h_1, h_2).$$

3.2.1 State of the art: algorithms

Seung et al. (1992) were among the first authors to propose an efficient active learning algorithm, based on a Bayesian prior on the hypothesis space, called *Query-by-committee* (QBC). It relies on the ability to sample hypotheses from the version space using the prior. Given a finite committee of hypotheses consistent with examples seen so far, *QBC* selects the instances for which the committee disagrees most on the label are selected. Originally applied to linear separators, *QBC* was extended to kernel machines in a tractable way, by Gilad-Bachrach et al. (2006). Variants of *QBC* using bagging and boosting were also proposed (Abe and Mamitsuka, 1998).

Another early heuristic for learning a binary classifier, devised by Cohn et al. (1994) in the realizable setting, considers a large pool of unlabeled examples (pool-based adaptive sampling), and uses a neural network to characterize the uncertainty region. The active algorithm proceeds by selecting an instance in the uncertainty region at each time step.

A related research direction focuses on *error reduction*, meant as the expected generalization error improvement brought by an instance. Many criteria reflecting various measures of the expected error reduction have been proposed (Iyengar et al., 2000; Roy and McCallum, 2001; Lindenbaum et al., 2004; Dasgupta et al., 2005), with some encouraging results in, for instance, pharmaceutical industry (Warmuth et al., 2003). Specific algorithms and methods have been developed for active learning in linear and kernel spaces, either heuristic (Schohn and Cohn, 2000) or theoretically grounded (Cesa-Bianchi et al., 2003; Dasgupta et al., 2005; Balcan et al., 2007). Active learning for support vector machines has also yielded promising results in image retrieval and text classification (Tong and Koller, 2001; Chang and Tong, 2001). A recent work by Schein and Ungar (2007), inspired by uncertainty reduction methods, surveys active learning on logistic regression models for classification, and proposes a method to find examples reducing error by looking at a criterion called A-optimality, which is trace of the inverse Fisher matrix of the experiment (Fisher, 1951).

Some of these approaches however happen to face learning instabilities, which might require to mix the active learning procedure with a uniform instance selection (Xiao et al., 2005). Such instabilities suggest that in some cases an optimally efficient active learning system can hardly be based on a greedy selection strategy, at least using the criteria considered so far.

3.2.2 State of the art: theoretical results

Angluin (1988) introduced query learning models that formalized how learning algorithms may directly interact with the oracle. Based on this model, an early work by Kulkarni et al. (1993) established a lower bound on the sample complexity in the general active learning case, logarithmic in the ϵ-covering number of hypothesis space $\mathcal{H}$; the result is actually concerned with *binary queries*, a general model that encompasses querying the oracle on a single input.

The concept of PAC-learnability (section 3.1.4) aims at characterizing how learnable is a concept class when using passive learning. The definition extends to active learning; since active learning is more powerful than passive learning, one could expect that more function classes will be learnable under this less stringent definition. This is actually not true: PAC-learnability is equivalent to active PAC-learnability (see for instance Vidyasagar, 1997, chap. 9).

This being said, even though active learning does not allow one to learn in wider function classes, it can be hoped that the sample complexity of active learning algorithms is smaller than the one of passive algorithms, considering that any passive learner dealing with a space of finite VC-dimension D requires $\tilde{\Theta}(D/\epsilon)$ examples to learn a binary concept with precision ϵ.

Freund et al. (1997) proved that methods such as QBC can lead to an exponentially smaller number of queries than the random querying strategy (passive learning), at least in the case of perceptrons. Freund et al. (1997) related the efficiency of QBC to a statistical criterion called *Information Gain*, measuring how efficiently the VS can be divided. This supported the idea that active learning may sometimes significantly reduce the sample complexity.

The appropriateness of hypothesis spaces to active learning has been studied, showing that active learning improves sample complexity significantly in some spaces, while it does not much better than passive learning in others (Hegedüs, 1995; Dasgupta, 2006; Hanneke, 2007a).

Dasgupta (2006) studied the non-Bayesian setting, deriving a lower complexity bound based on a criterion called *splitting index*. This index is used to determine the cases where active learning enables an exponential reduction of the sample complexity. Dasgupta (2006) proposed an "almost" optimal, though intractable, algorithm coarsely matching the lower bound. Hanneke (2007a) proposed a criterion referred to as *disagreement coefficient*, partially capturing the active learning potential of a concept class, even in presence of bounded noise (see below). Similarly to VC-dimension and covering numbers, aimed at estimating the learnability of a hypothesis space, the splitting index and the disagreement coefficient aim at assessing the reduction of sample complexity provided by active learning.

In the Bayesian setting, Dasgupta (2005) has established the quasi-optimality of greedy version space reduction methods in the realizable classification case (i.e. when h^* belongs to $\mathcal{H}$) for a finite instance pool. More generally, when there exists a probability measure P_H on $\mathcal{H}$, usual active learning strategies are concerned with reducing either the measure of the Version Space, or the variance of labels for hypotheses of the version space. In non-Bayesian settings, the

reduction of the version space can be expressed in terms of its diameter.

3.2.3 Noise

Agnostic active learning considers the case where the training set can include noise, and/or the target concept does not belong to the hypothesis space. It was studied by Balcan et al. (2006) using an (intractable) agnostic active learning algorithm called A^2. Hanneke (2007a) established complexity bounds for A^2 through the disagreement coefficient. Dasgupta et al. (2008) presented an agnostic active algorithm whose sample complexity was also bounded using the disagreement coefficient.

The bounded noise rate setting was investigated further by Kaariainen (2006), showing that it does not significantly differ from the realizable setting. Kaariainen (2006) also studied the context of active learning with unbounded noise rate, along with Castro and Nowak (2007), who focused on the learning rates when instances are close to the decision boundary. Hanneke (2007b) also proposed an analysis based on the *teaching dimension* (see Goldman and Kearns, 1991; Hegedüs, 1995), which is outside the scope of this work.

3.2.4 Regression

The so-called *blind active learning*, also referred to as *Experiment Design* (Fisher, 1951), has been investigated in the regression framework since 1920. Some recent results on experiment design are based on low-discrepancy and low dispersion sequences (Niederreiter, 1988), see in particular the work of Teytaud et al. (2007).

Active regression has been much less studied than active classification. Cohn et al. (1996) proposed an approach based on minimization of the learner variance: using mixtures of gaussians as hypothesis space, they can quantify the expected variance of the model when labeling a point, and choose the one that maximizes this variance.

Castro et al. (2005) analyzed convergence rates in two function spaces: *Hölder-smooth* functions, and piecewise-constant functions[6]. For Hölder-smooth functions, the authors have derived a minimax lower bound assuming independent Gaussian noise on the labels given by the oracle. This disappointing result is blamed on the expressiveness of Hölder-smooth functions, adversely affecting their learnability; one understands in hindsight that active learning does not help much in this case. Under stronger assumptions such as finite VC-dimension, many hypothesis spaces ensure a good learnability. Such spaces include e.g. linear separators, kernel spaces, artificial neural networks, etc. It would be very interesting to investigate further the benefits of active regression in such spaces.

[6] although the latter is actually restricted to the *Boundary fragment* class, namely functions valued in $\{-M, M\}$, with regularity conditions on the frontier in the input space corresponding to the jump from $-M$ to M, which makes it similar to a classification setting

Chapter 4

Optimal Active Learning via MDPs : the *BAAL* Algorithm

In this chapter, active learning under bounded resources is formalized as a finite horizon reinforcement learning problem, where the sampling strategy aims at minimizing the expectation of the generalization error. A tractable approximation of the optimal (intractable) policy is presented, the *Bandit-based Active Learner* (*BAAL*) algorithm. *BAAL* tackles active learning as a one-player game, combining UCT, a tree structured multi-armed bandit algorithm (Kocsis and Szepesvari, 2006), and billiard algorithms. The source code is publicly available (Rolet, 2010). Only the classification case will be considered in the following. The stakes of extending the following approach to regression are discussed in chapter 11. The first version of this work was originally published in the proceedings of the twenty-first European Conference on Machine Learning (Rolet et al., 2009a).

4.1 Finite Horizon Active Learning: a Dynamic Programming Formulation

This section presents a theoretically optimal strategy for active learning in *finite time horizon* T, where T corresponds to the total number of instances to be labeled along the active learning process.

Similarly to Freund et al. (1997) and Dasgupta (2005) (among others), the proposed approach is built on a Bayesian setting (Mackay, 1992; Haussler et al., 1994). Prior knowledge about the target concept is accounted for by a probability distribution $P_{\mathcal{H}}$ on hypothesis space[1] $\mathcal{H}$. Noting $\mathcal{H}(s)$ the version space

[1] $P_{\mathcal{H}}$ is set to the uniform distribution on $\mathcal{H}$ in the absence of prior knowledge.

associated to a training set s, measure $P_{\mathcal{H}}$ naturally induces a conditional measure on any version space $\mathcal{H}(s)$ with a non-null probability mass[2]:

$$\forall A \subset \mathcal{H}(s), P_{\mathcal{H}(s)}(A) = P_{\mathcal{H}}(A)/P_{\mathcal{H}}(\mathcal{H}(s)) \text{ or equivalently}$$
$$p_{\mathcal{H}(s)}(h) = \frac{p_{\mathcal{H}}(h)}{P_{\mathcal{H}}(\mathcal{H}(s))}\mathbf{1}_{\mathcal{H}(s)}(h). \tag{4.1}$$

A random hypothesis of $\mathcal{H}$ drawn from $P_{\mathcal{H}}$ (respectively $P_{\mathcal{H}(s)}$) will be noted H (respectively H_s).

The following relies on the *realizable* assumption, i.e. the target concept f^* to be learned is assumed to be deterministic and to belong to $\mathcal{H}$ (how to relax this assumption will be discussed in section 11.3). In this section, given a training set s, a point x and a hypothesis h, the set $s \bigcup \{x, h(x)\}$ will be referred to as $s + h(x)$.

For the sake of readability, the binary classification case will be considered throughout this chapter, although the approach holds for the multi-class setting either.

4.1.1 Optimal finite-time active learning

The first requirement when looking for an optimal sampling strategy is to define what "optimal" means. Considering finite-horizon T, with same notations as chapters 2 and 3, let $\mathbf{S}$ be the sampling process. Random variable $S_{T,h}$ denotes the training set built by $\mathbf{S}$ after T steps of the process described in equation 2.1 when learning some target concept h.

Distance ρ is the generalization error as defined in section 3.1.2. Therefore, $\rho(\mathbf{A}(S_{T,h}), h)$ is the generalization error of the hypothesis learned by $\mathbf{A}$ on training set $S_{T,h}$. A natural way to define an optimal active learning sampler for finite time horizon T is to consider the minimization of the expectation of $\rho(\mathbf{A}(S_{T,h}), h)$ when h ranges in $\mathcal{H}$.

Let us define precisely the law of $S_{T,h}$, considering the probability laws on training sets that embody the process of equation 2.1. The definitions are recursively based on the joint probabilities (such as in equation 2.3).Let δ_s stands for the Dirac measure in s (section 2.1). Conditionally to the probability distribution on $\mathcal{H}$:

$$\begin{cases} p^{(0)}(s|h) & = \delta_{\emptyset}(s); \\ p(s'|s, x, h) & = \delta_{s+h(x)}(s'); \\ \pi(x|s, h) & = \mathbf{S}(x, s). \end{cases} \tag{4.2}$$

The initial training set, noted s_0, is the empty set. The joints $p^{(t)}$ for $t > 0$,

[2] *Case* $P_{\mathcal{H}}(\mathcal{H}(s)) = 0$ In this case, $P_{\mathcal{H}}(s)$ need not be defined for the purposes of this work: active learning as analyzed below is such that instance selection cannot lead to a training set whose corresponding version space has null measure.

conditioned on h, are then

$$p^{(t)}(x_1, s_1, \ldots, x_t, s_t|h) = \prod_{i=0}^{t-1} \mathbf{S}(x_{i+1}, s_i)\delta_{\{s_i+h(x_{i+1})\}}(s_{i+1}). \tag{4.3}$$

Identically to equation 2.3, $p^{(t)}(s|h)$ is the marginal of training set s understood as the last state variable in density $p^{(t)}$.

Noting $S_{T,H}$ the random training set conditioned to random hypothesis H, the desired optimality criterion for a sampler then reads

$$\mathbf{S}_T^* = \arg\min_{\mathbf{S}} \mathbb{E}_{H,S_{T,H}}[\rho(\mathbf{A}(S_{T,H}), H)]. \tag{4.4}$$

The law of $S_{T,H}$ for the expectation in this equation follows density $p^{(T)}(s|h)$.

From this definition, optimal sampler $\mathbf{S}^*$ is specific to the chosen hypothesis space $\mathcal{H}$, and to the learning algorithm $\mathbf{A}$.

4.1.2 The active learning Markov decision process

The main motivation behind the presented MDP framework is to build an optimal sampling policy in the sense of equation 4.4.

As stated in section 2.2, MDPs are classically described in terms of states, actions, reward, policy and transition functions. In the active learning context, the state space $\mathcal{S}$ consists of all possible training sets s. An action corresponds to the selection of a new instance to be labeled; the set of actions $\mathcal{A}c$ thus coincides with the instance space $\mathcal{X}$ or a subset thereof.

There is a clear analogy between the active learning notations of section 2.1.1, where a sampler $\mathbf{S}(x, s)$ is a probability density for an instance x given training set s, and the MDP notations of section 2.2, where a policy $\pi(a|s)$ is a probability density for an action a given a state s. A sampler is then written as a policy π.

Let us derive the right transition density p_{AL} and reward r_T such that solving the Markov decision process $(\mathcal{S}, \mathcal{X}, p_{AL}, r_T)$ amounts to finding an optimal sampling strategy in the sense of equation 4.4. Note that while the process described by equations 4.2 depend on a target h, p_{AL} and r_T must be defined independently of h.

The following result states that, for the adequate definitions of p_{AL} and r_T, the Bellman optimal policy of the MDP yields an optimal active learning policy in the sense of equation 4.4.

Theorem 1. *Let transition density p_{AL} and reward r_T be defined as*

$$r_T(s, x, s') = \begin{cases} \mathbb{E}_{H_s}[\rho(\boldsymbol{A}(s), H_s)] & \text{if } |s|{=}T \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

$$p_{AL}(s'|s, x) = \int_{h \in \mathcal{H}(s)} \delta_{s+h(x)}(s') p_{\mathcal{H}(s)}(h) dh. \tag{4.6}$$

Then, the finite-time active learning Markov decision process $(\mathcal{S}, \mathcal{X}, p_{AL}, r_T)$ *is such that for any policy* π *for this MDP, the value function* $\boldsymbol{V}^\pi$ *satisfies at initial state* s_0

$$\boldsymbol{V}^\pi(s_0) = \mathbb{E}_{H, S_{T,H}}[\rho(\boldsymbol{A}(S_{T,H}), H)] \tag{4.7}$$

where the law of H *is given by* $P_{\mathcal{H}}$ *and the law of* $S_{T,H}$ *is given by density* $p^{(T)}(s|h)$.

An immediate consequence of this theorem is that the Bellman-optimal value function V^* maps s_0 to the minimum described in equation 4.4: an optimal policy for this MDP is an optimal active learning strategy.

Proof. The proof first explains mathematical expressions of reward function r_T and transition function p_{AL}. Equation 4.7 is thereafter obtained by backward induction.

Regarding the reward function, as the goal is to minimize the generalization error of the hypothesis learned after T steps, the reward is only known at horizon T, when a *terminal state* s (i.e. such that $|s| = T$) is reached. From the realizable assumption, the target concept h is bound to be in the version space of s, noted $\mathcal{H}(s)$. Thus, the reward function is defined as in theorem 1:

$$r_T(s, x, s') = \begin{cases} \mathbb{E}_{H_s}[\rho(\mathbf{A}(s), H_s)] & \text{if |s|=T} \\ 0 & \text{otherwise.} \end{cases}$$

Transition probability density $p_{AL}(s, x, s') = p_{AL}(s'|s, x)$ models the probability of going to state s' conditioned by current state s and current action x (the selected instance). By definition, states accessible from s are $s \bigcup \{(x, y = h(x))\}$, where h denotes a concept that belongs to the version space $\mathcal{H}(s)$ (because of the realizable assumption). From equations 4.2, it is consistent to define p_{AL} as in theorem 1:

$$p_{AL}(s'|s, x) = \int_{h \in \mathcal{H}(s)} \delta_{s+h(x)}(s') p_{\mathcal{H}(s)}(h) dh.$$

To show 4.7, let us first notice that from the probabilities defined in equation 4.2 derive conditional joints $p^{(t_0,t_1)}$ modeling probability of training sets $s_{t_0+1}, \ldots, s_{t_1}$ given that the state at step t_0 is s_{t_0}. Since by construction, the process described by equations 4.3 has the Markov property, those probabilities read

$$p^{(t_0,t_1)}(x_{t_0+1}, s_{t_0+1}, \ldots, x_{t_1}, s_{t_1}|s_{t_0}, h) = \prod_{i=t_0}^{t_1-1} \pi(x_{i+1}|s_i) \delta_{\{s_i+h(x_{i+1})\}}(s_{i+1}). \tag{4.8}$$

The marginal of state s' as the last variable of $p^{(t_0,t_1)}$ conditionally to state s of the t_0th step of the process is written $p^{(t_0,t_1)}(s'|s, h)$. With these notations, observe that the density of random variable $S_{T,H}$ in equation 4.4 is $p^{(0,T)}(.|\emptyset, h)$.

We can now prove equation 4.7 by backward induction, by showing

$$\begin{aligned} &\forall t \in [0,T], \forall s \ s.t. \ |s| = t, \\ &\mathbf{V}^{\pi}(s) = \mathbb{E}_{H_s, S_{T,H_s}|s}[\rho(\mathbf{A}(S_{T,H_s}(H_s)), H_s)] \end{aligned} \tag{4.9}$$

where the law of $S_{T,H_s}|s$ has density $p^{(t,T)}(.|s,h)$. The generalization error will hereafter be noted $\rho(S_{T,H_s}, H_s)$ for short. Although the induction may seem intuitive, it actually requires a few computations to ensure that value functions functions associated to p_{AL} and r_T actually satisfy equation 4.9.

To initialize the induction procedure, the case $t = T$ will be addressed. To this end, we need to give a meaning to $p^{(t,t')}(s'|s,h)$ when $t = t'$:

$$\forall t \in [[0,T]], p^{(t,t)}(s'|s,h) \doteq \delta_s(s'). \tag{4.10}$$

Then, from equation 2.5, by setting the discount factor γ to 1, the value function of a policy π at a terminal state s is exactly equal to $r_T(s,.,.)$ (since r_T in this case does not depend on action x and previous state s').

For the induction step, assuming that equation 4.9 is true for some $t > 0$, it can be shown that it is also true for $t-1$ using the Bellman equation for a value function (equation 2.6). Let s be a state satisfying $|s| = t-1$. Then,

$$\begin{aligned} \mathbf{V}^{\pi}(s) &= \mathbb{E}_{A|s,S|A.s}[r(s,A,S) + \gamma \mathbf{V}^{\pi}(S)] \\ &= \mathbb{E}_{A|s,S|A.s}[\mathbf{V}^{\pi}(S)] \\ &= \int_{\mathcal{X}} \int_{\mathcal{S}} p_{AL}(s'|s,x)\pi(x|s)\mathbb{E}_{H|s',S_{T,H}|s'}[\rho(S_{T,H},H)]dxds' \\ &= \int_{\mathcal{X}} \int_{\mathcal{S}} p_{AL}(s'|s,x)\pi(x|s) \int_{\mathcal{H}} \int_{\mathcal{S}} p_{\mathcal{H}(s')}(h) p^{(t,T)}(s_T|s',h)\rho(s_T,h)dhds_T dxds' \\ &= \int_{\mathcal{H}} \int_{\mathcal{S}} \rho(s_T,h)\mathbf{J}(s_T,x,h)dhds_T \end{aligned} \tag{4.11}$$

with $\mathbf{J}(s,s_T,h) = \int_{\mathcal{X}} \int_{\mathcal{S}} p_{AL}(s'|s,x) p_{\mathcal{H}(s')}(h) p^{(t,T)}(s_T|s',h)\pi(x|s)ds'dx$.

From equations 4.8 and 4.10, and the Markov property of the process, we have

$$\begin{aligned} &\forall t_0 \in [[0,T-1]], \forall t_1 \in [[t_0+1,T]] \\ p^{(t_0,t_1)}(s_{t_1}|s_{t_0},h) &= \int_{\mathcal{X}\times\mathcal{S}} \pi(x_{t_0+1}|s_{t_0})\delta_{s_{t0}+h(x_{t0+1})}(s_{t_0+1}) \\ &\quad p^{(t_0+1,t_1)}(s_{t_1}|s_{t_0+1},h)ds_{t_0+1}dx_{t_0+1} \\ &= \int_{\mathcal{X}} \pi(x_{t_0+1}|s_{t_0}) p^{(t_0+1,t_1)}(s_{t_1}|s_{t_0}+h(x_{t_0+1}),h)dx_{t_0+1}. \end{aligned} \tag{4.12}$$

By including the expression of $p_{AL}(s'|s,x)$ in $\mathbf{J}(s_T,x,h)$,

$$\begin{aligned} \mathbf{J}(s,s_T,h) &= \int_{\mathcal{S}} \int_{\mathcal{X}} \int_{\mathcal{H}} \delta_{s+h'(x)}(s') p_{\mathcal{H}(s)}(h') dh' p^{(t,T)}(s_T|s',h)\pi(x|s) p_{\mathcal{H}(s')}(h) ds'dx \\ &= \int_{\mathcal{X}} \pi(x|s) \int_{\mathcal{H}} p_{\mathcal{H}(s)}(h') p^{(t,T)}(s_T|s+h'(x),h) p_{\mathcal{H}(s+h'(x))}(h) dh'dx \end{aligned}$$

Because of the definition of $p_{\mathcal{H}(s)}(h')$ and $p_{\mathcal{H}(s+h'(x))}(h)$ (and since $s + h'(x)$ is short for the set $s \bigcup \{(x, h'(x))\}$), the right-hand side integrand is null when $s + h'(x) \neq s + h(x)$, therefore

$$\begin{aligned}
\mathbf{J}(s, s_T, h) &= \int_{\mathcal{X}} \pi(x|s) \int_{\mathcal{H}} p_{\mathcal{H}(s+h(x))}(h) p_{\mathcal{H}(s)}(h') \\
&\qquad p^{(t,T)}(s_T|s + h(x), h) \mathbf{1}_{\{h_0 \in \mathcal{H} | h_0(x) = h(x)\}}(h') dh' dx \\
&= \int_{\mathcal{X}} \pi(x|s) p^{(t,T)}(s_T|s + h(x), h) p_{\mathcal{H}(s)}(h) dx \qquad (4.13) \\
&= p_{\mathcal{H}(s)}(h) p^{(t-1,T)}(s_T|s, h). \qquad (4.14)
\end{aligned}$$

Equation 4.13 arises from the fact that

$$\int_{\mathcal{H}} p_{\mathcal{H}(s)}(h') \mathbf{1}_{\{h_0 \in \mathcal{H} | h_0(x) = h(x)\}}(h') dh' = \frac{P_{\mathcal{H}}(\mathcal{H}(s + h(x)))}{P_{\mathcal{H}}(\mathcal{H}(s))}$$

(using definition 4.1), which multiplied by $p_{\mathcal{H}(s+h(x)}(h)$ yields $p_{\mathcal{H}(s)}(h)$. Equation 4.14 is obtained using equation 4.12.

Integrating equation 4.14 in the value function expression 4.11 yields the desired result. □

Remark 3. Note that the active learning MDP may start with a non-empty training set: all statements still hold for $s_0 \neq \emptyset$.

4.1.3 A partially observable Markov decision process

A more general formulation of active learning as a *planning* problem, embracing the formalism presented above, is a *partially observable* MDP(POMDP) formulation. As seen in section 2.2.4, states of a POMDP have a *visible* part and a *hidden* part—classical MDPs are of course a subset of POMDPs. In the active learning case, a state comprises the unobserved target hypothesis h besides the training set. The decision to select an instance x_{t+1} leads to a new state $s_{t+1} = s_t \bigcup (x_{t+1}, h(x_{t+1}))$, where s_t contains h (the transition is deterministic). The observation corresponding to this partially observable state is $h(x_{t+1})$—the observed state, is the training set s_{t+1} *without* hypothesis h. The reward at a terminal state s_T is not an expectation anymore, it is directly $\rho(h, s_T)$.

In practice, embedding the unobserved target h in the state amounts to model the hidden variable within the transition function and the reward function of the MDP: equations 4.2 can be used directly as a definition of the POMDP (while they could not be used directly for the simple MDP). The rewards at the end of the process is the same as for the MDP formulation. However, devising an optimal policy and showing that it converges to equation 4.4 then requires additional effort.

In this respect, a connection between POMDPs and MDPs in the case of discrete state spaces has been studied by Astrom (1965). He notably devised

a theoretical strategy to tackle POMDPs as regular MDPs under certain conditions, although this strategy does not apply straightforwardly, in particular because the active learning problem is a continuous one.

This POMDP formulation implicitly presents active learning as a one-player game. The active learner plays against the (unknown) target hypothesis h, belonging to the version space $\mathcal{H}(s_0)$ of the initial training set[3] s_0. Upon each move (selection of some instance x), oracle h provides the label $y = h(x)$. At the end of the game—that is, after T examples have been picked, defining training set s_T—the reward is the generalization error of the hypothesis $\mathbf{A}(s_T)$ learned from s_T with respect to h. The *real* learning game is played against oracle h.

It is however possible to train the active learning player, and therefore devise a good active learning policy beforehand, by mimicking the above game and playing against a "surrogate" oracle, made of a hypothesis h uniformly selected in the VS. The reward of such a game is computed as in the real game: it is the generalization error of the learned hypothesis w.r.t. the (surrogate) oracle. This reward implements a uniform draw of the random variable $\rho(\mathbf{A}(s), h)$ for h ranging in $\mathcal{H}$.

By construction, the average empirical reward collected by the active learning player after many such games asymptotically converges toward the true expectation of this random variable, that is, the desired reward function.

4.2 Overview of the *BAAL* Algorithm

Next sections will present a consistent and tractable approximate resolution of the above MDP, the *BAAL* algorithm. As already mentioned, for the sake of simplicity, *BAAL* is presented in the binary classification case; it must be emphasized that the publicly available code handles multi-class problems.

BAAL relies on three main ingredients:

- the tree-structured multi-armed bandit algorithm called UCT (Kocsis and Szepesvari, 2006), which is extended to the one-player game of active learning in section 4.3;
- *billiard* algorithms to sample the version space, presented in section 4.4;
- *progressive widening* to apply UCT to continuous state spaces, and allow integration of "expert knowledge" on active learning to *BAAL* (section 4.5).

[3] For the sake of simplicity and when no confusion is to fear, the initial training set $\mathbf{s}_0$ is omitted and $\mathcal{H}$ is used instead of $\mathcal{H}(s_0)$.

4.3 Monte-Carlo Tree Search to Solve the Active Learning Markov decision process

Upper Confidence Trees (UCT) is a Monte-Carlo tree-search algorithm (Kocsis and Szepesvari, 2006; Coulom, 2006) in which the selection of a child node is cast into a multi-armed bandit problem (see section 2.3). UCT notably became famous in the domain of strategic games as it inspired the computer-Go program MoGo, first to ever win over professional human players without handicap (Gelly and Silver, 2007).

UCT is aimed at searching large trees in which each leaf bears a deterministic or stochastic reward. The tree representing a MDP is made of all the states and all the state-action pairs. The root of the tree is the initial state s_0. From each state node s, a branch for each action a available from this state leads to state-action node (s, a). Then, from (s, a) there is a branch leading to each s' such that the transition density $p(s'|s, a)$ is not null. Since a state may be reached by more than one state-action node, the tree is actually a directed graph. In the case of active learning, states (sets) of cardinal t can only be reached from state-action nodes whose state have cardinal $t - 1$: the graph is acyclic (it is a DAG). Fortunately, UCT is suited to search DAGs, and does not depend on the particularity of trees requiring a node to have exactly one father. We refer to the active learning DAG as a "game tree", to pinpoint the analogy with a one-player game.

As an example of game tree, in board games such as Chess or Go, the tree is made of all the possible games: at a given node, a player chooses a move, reaching another node, from which the other player chooses a move until the game ends (leaf node). The reward is usually binary: 0 if the game is lost, 1 if it is won.

UCT will be described through the BAAL algorithm (algorithm 2). Let $\mathcal{T}$ denote the "game" tree of active learning (figure 4.1); its root node is the initial training set s_0.

A path from the root node to a leaf is called a tree walk, also referred to as a game or a simulation. To each simulation is attached a hypothesis h, uniformly extracted from the *version space* of the root node s_0. Each node corresponding to a training set s_t , a *state* node, has a continuous set of child nodes, the state-action nodes (s, x) with $x \in \mathcal{X}$ since for active learning actions correspond to instances. The children of s_t (state-action nodes) correspond to the possible instances that the learner can choose to query—each one is made of the parent node s_t, and a chosen instance x_{t+1}, and written $s_t + x_{t+1}$.

Every state-action node has two children in the binary classification case, one for each possible label of instance x_{t+1}. The one selected during the tree-walk indexed by h obviously corresponds to $h(x_{t+1})$, thus leading to the next state node $s_{t+1} = s_t + h(x_{t+1})$.

More precisely, each tree-walk proceeds as follows (Figure 4.1 and algorithm 2):

1. first, a hypothesis h is drawn in version space $\mathcal{H}(s_0)$ of the initial training

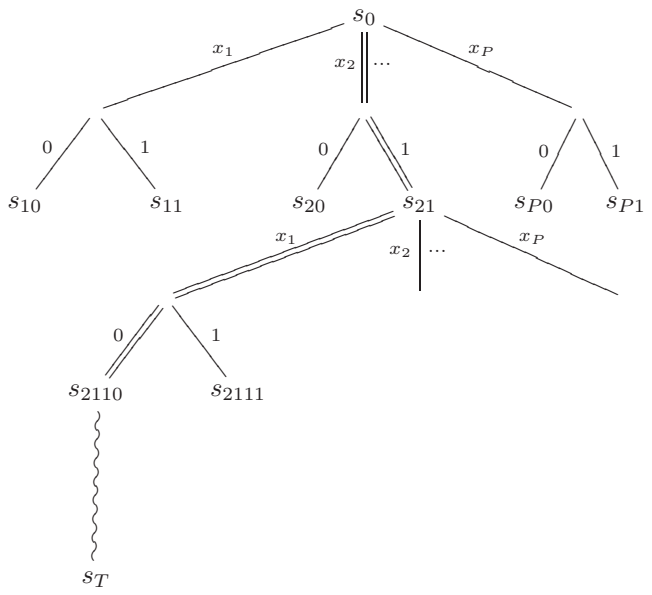


Figure 4.1: Game tree developed by *BAAL*, the Bandit Based Active Learner, in the binary classification case. The root of the tree is the initial training set, s_0. Each tree walk (or simulation) is based on selecting some hypothesis h in the version space of s_0. The tree walk proceeds by iteratively selecting an instance x, whose label is set to $h(x)$. Ultimately, the tree walk is assessed by generalization error $\rho(s_T, h)$ of the learned hypothesis w.r.t. h.

set, s_0, according to $P_{\mathcal{H}}$, referred to as a *surrogate* hypothesis;

2. then, in a given state node (training set s_t), *BAAL* uses the famed Upper Confidence Bounds (UCB) criterion (equation 4.15, see below) to select a state-action node, corresponding to some instance x_{t+1} to be labeled;
3. surrogate hypothesis h is used to set the label to $h(x_{t+1})$, and state node $s_{t+1} = s_t + h(x_{t+1})$ is reached;
4. the tree-walk proceeds until arriving at a state node s_{t_0} not yet visited; at this point, $T - t_0$ additional instances are randomly uniformly selected, labeled using h, and added to the training set s_{t_0}, forming a T-size training set s_T;
5. from training set s_T, a hypothesis $\hat{h}$ is learned by *BAAL*[4];
6. the reward of the tree walk is the generalization error of $\hat{h}$ with respect to the surrogate hypothesis h;

[4]The learning algorithm is actually a parameter of *BAAL*. In the experiments, a uniform sampler of the version space of s_T is used (chapter 6)

7. the reward of the leaf is backpropagated to update the value of every tree node s_t such that $t \leq t_0$.

Algorithm 2 The ***BAAL*** algorithm.

BAAL$(P_{\mathcal{H}}, \mathbf{s}_0, T, N)$
  Parameter Measure $P_{\mathcal{H}}$ on hypothesis space $\mathcal{H}$
  Initial training set $\mathbf{s}_0$
  Time horizon T
  Number N of allowed tree-walks
  Set $n(s) = 0$ for all s //number of times s has been visited, i.e. $T_s(i)$
  for i=1 to N **do**
    $h =$ `DrawHypothesis`$(P_{\mathcal{H}}, \mathbf{s}_0)$ $//h \sim P_{\mathcal{H}}, h \in \mathcal{H}(\mathbf{s}_0)$
    Tree-Walk$(\mathbf{s}_0, T, h)$
  end for
  //select the instance on the most visited branch from the root
  Return $x = \arg\max_{x' \in \mathcal{X}} T_{s_0+x}(N)$
  //instance x to be labeled by the oracle.

routine `Tree-Walk`(s, t, h)
  $n(s)$++
  if t==0 **then**
    Compute $r = \rho(\mathbf{A}(s), h)$
  else
    $\mathcal{Z} =$ `ArmSet`$(s, n(s))$ // discrete set of considered instances
    Select $x^* = UCB - Tuned(s, \mathcal{Z}))$
    // recursively call Tree-Walk
    $r =$ Tree-Walk$(s + h(x^*), t-1, h)$
  end if
  // update reward of node s
  $r(s) = (1 - \frac{1}{n(s)})r(s) + \frac{1}{n(s)}\ r$
  Return r

routine `UCB-Tuned`$(s, \mathcal{Z})$
  for $x \in \mathcal{Z}$ **do**
    if $n(s+x) == 0$ **then**
      Return x
    end if
  end for
  Return $\arg\max_{s' \in \{s+x | x \in \mathcal{X}\}} \hat{X}_{s',n(s')} + \sqrt{\frac{\ln(t)}{n(s')} \min\{1/4, \bar{V}_{s',n(s')}\}}$

Values are attached to each state node and state-action node, and are exploited by a UCB-like criterion (Auer et al., 2002); they support the choice of a state-action node from a state node (in step 2). The value of state node s_t is updated after each tree walk going through s_t. In accordance with notations of section 2.3.3 on multi-armed bandits where UCB is introduced, for the nth tree

walk, let us write $T_{s_t}(n)$ the number of times s_t has been visited. The value of s_t is the average of all $T_{s_t}(n)$ rewards gathered for leaf nodes s_T such that $s_t \subseteq s_T$, noted $\hat{X}_{s_t,T_{s_t}(n)}$. The value of a state-action node $s_t + x$ is computed similarly, as the average of rewards obtained from each tree walk going through $s_t + x$.

The arm selected is the one maximizing the sum of the empirical reward $\hat{X}_{s_t,T_{s_t}(n)}$ (exploitation term), plus an exploration term depending on the number of times that arm s_t has been selected. The exploration-exploitation tradeoff is adjusted via an upper bound on empirical variance $\bar{V}_{s_t,T_{s_t}(n)}$ (section 2.3); the formula for choosing the next state-action node (i.e. choosing an instance) is the UCB-tuned formula (equation 2.9):

$$\arg \max_{s \in children(s_t)} \hat{X}_{s,T_s(n)} + \sqrt{\frac{\ln(n)}{T_s(n)} \min 1/4, \bar{V}_{s,T_s(n)}} \tag{4.15}$$

where $children(s_t) = \{s_t + x | x \in \mathcal{X}\}$ is the set of state-action nodes that are children of s_t. In the multi-armed bandit setting, this formula exhibits good empirical performance (Auer et al., 2002). Furthermore, although there exist some sets of probability distributions for arm rewards such that the formula is suboptimal, slight variations of this formula have been derived (Audibert et al., 2006; Audibert et al., 2008) that were proved optimal in terms of asymptotic cumulative regret (i.e. the regret is logarithmic asymptotically), under the assumption that arm rewards are independent random variables (although clearly this assumption does not hold in the active learning game, no more than in the game of Go, see Wang and Gelly, 2007).

Remark 4. Section 2.3.3 points out that UCB-like criteria require each arm to be evaluated once before the selection equation may be used. This raises an issue in active learning since quite often the instance space $\mathcal{X}$ (which is the set of arms) is continuous. This issue is addressed through the progressive widening heuristic, detailed in section 4.5.

The memory-wise computational tractability of *BAAL* is ensured by gradually developing the tree in memory. It is initially made of the root node and its direct children. During each game simulation, at some point, a node s is encountered that is not yet stored in memory, and from here on the game proceeds randomly. This node is then stored in memory with its child nodes: the next time it is encountered, the selection among its child nodes will rely on the UCB formula. Hence, the tree is asymmetrically grown: the subtrees where nodes have better values will be more developed, since those subtrees will be visited more often.

BAAL implements the UCT scheme with two specific ingredients. The `DrawHypothesis` function selects the surrogate hypothesis h attached to each tree-walk using billiard algorithms (see below). The `ArmSet` function (section 4.5) extends UCT to deal with an infinite set of actions (the continuous instance space), using the so-called progressive widening heuristics. *BAAL* has four inputs:

- the probability distribution $P_{\mathcal{H}}$ on hypothesis space $\mathcal{H}$ (Bayesian setting);
- the initial training set $\mathbf{s}_0$;
- the time horizon T;
- the number N of allowed tree-walks.

Its output is the instance x to be queried next by the active learner.

4.4 Sampling in the Version Space with Billiards

The second ingredient in *BAAL* is a uniform sampler of the *version space*; likewise, as sampler of the instance space is required by *BAAL*(unless the learning problem comes with a predefined pool of instances). This section describes how a fair and frugal billiard-based algorithm (Ruján, 1997; Herbrich et al., 2001) is used to sample the instance space and hypothesis space. As already mentioned, each tree walk in *BAAL* is performed by using a hypothesis h sampled randomly in version space $\mathcal{H}(s_0)$ to label instances. The focus is on uniform random selection (i.e. assuming that every hypothesis of the version space has the same probability of being the target as any other, as often done when we have no prior knowledge).

The most straightforward sampling algorithm is based on *rejection*: hypotheses are uniformly drawn in $\mathcal{H}$ and rejected if they do not belong to the version space (that is, if they are inconsistent with the training set s_0). The rejection algorithm is sound: it is guaranteed that its outcome is exactly what a random uniform draw would output. It can also be adapted to non-uniform measures. Unfortunately, it is hardly tractable.

Alternative algorithms such as Gibbs sampling or more generally Monte-Carlo Markov Chains (MCMC) methods involve quite a few free parameters and might scale poorly with respect to the dimensionality of the search space and the size of the training set s_0.

For this reason a billiard (a.k.a. ray-tracing) algorithm inspired from Ruján (1997), Ruján and Marchand (2000) and Herbrich et al. (2001) has been used; it only assumes that the hypothesis space $\mathcal{H}$ can be parametrized by $\mathbb{R}^d$. The experimental validation (section 6.1) considers linear hypotheses for which such a parametrization is natural; how to go beyond the linear case will be discussed in the perspectives of the presented work (section 11.3).

Let Ω be a connected subset of $\mathbb{R}^d$, defined by a set of constraints $g_1, \ldots, g_n$: $\Omega = \{x \in \mathbb{R}^d \ s.t. \ \forall i \in [[1, n]], \ g_i(x) \geq 0\}$. Consider a point $z \in \mathbb{R}^d$ (not necessarily in Ω) and a direction $\mathbf{v} \in \mathbb{R}^d$ such that $||\mathbf{v}|| = 1$). Point z is seen as a ball in a *billiard*, and moves as follows (4.4):

1. z follows direction $\mathbf{v}$ until it meets an active constraint g, i.e. a constraint that is already satisfied—and that will not be satisfied anymore if the point goes on in the same direction;

2. z "bounces" when it meets such a constraint, i.e. its direction $\mathbf{v}$ is changed to point inside the domain (see below);

3. the trajectory is stopped when the computational resources are exhausted, that is when its total length reaches some user-defined parameter L, and the final point is returned.

Algorithm 3 Billiard algorithm, taking as input a set of constraints g_i, a trajectory length L, and returning a final point.

Randomly select $z \in \mathbb{R}^d$ satisfying at least one (but not necessarily all) constraints, and a direction $\mathbf{v}$.
while $L > 0$ **do**
  //Find the set of satisfied constraints
  $J = \{j; g_j(p) \geq 0\}$
  //Go as far as possible while $g_j \geq 0, j \in J$
  $\lambda^* = \sup\{\lambda \geq 0 \ s.t. \ \forall \ell < \lambda, \forall j \in J, g_j(z + \ell \mathbf{v}) \geq 0\}$
  if $J = \{1, \ldots, n\}$ **then**
    (all constraints satisfied)
    if $L > \lambda^*$ **then**
      //Go until some g_i is saturated
      $p = p + \lambda^* \vec{v}$
      $L = L - \lambda^*$
    else
      Return $p + L\mathbf{v}$ //out of resources
    end if
  end if
  $\mathbf{v} =$ symmetric $(\mathbf{v}, g_i)$ Bounce against g_i
end while

It is clear that the set of constraints satisfied by z does not decrease. Hopefully, it increases and eventually z satisfies all the constraints. In fact, under a few regularity conditions on the constraints, a billiard trajectory is ergodic, i.e. it covers the whole domain $\mathcal{W}$ when L goes to infinity (Comets et al., 2009). Given the law used for rebound (see below), the probability distribution of the final trajectory point converges toward a specific probability distribution on Ω when L gets large.

Billiard algorithms have been successfully used in machine learning, for instance to estimate the Bayes classifier in a (high-dimensional) kernel feature space (Ruján and Marchand, 2000; Herbrich et al., 2001).

Rebound policy

Let g_i be the saturated constraint and z_i the rebound point. The rebound policy sets the new direction $\mathbf{v}$ followed by z when it reaches z_i. This policy determines to which probability distribution the billiard converges—that is, which

probability distribution on $\mathcal{W}$ is approximated by the distribution of point z after it has walked a path of length L in the billiard.

Comets et al. (2009) shows that if the so-called Knudsen law is used to sample the new $\mathbf{v}$ at the rebound point, then the billiard algorithm converges to the uniform probability distribution on $\mathcal{W}$. While the Knudsen law is difficult to compute, an accurate approximation thereof is obtained by setting the direction to a unit vector randomly drawn in the half sphere centered on z_i and defined from the hyperplane tangent to g_i in z_i, as reported by Comets et al. (2009).

4.5 Progressive widening and the integration of existing active learning criteria

A tree search heuristic called progressive widening has been included to the UCT engine of *BAAL* in order to resist over-exploration and deal with continuity of the action space on the one hand, and to take advantage of existing active learning criteria, such as Query-by-Committee (Seung et al., 1992), on the other hand.

UCB requires each arm to be selected at least once for equation 4.15 to be computed. Therefore, when the number of arms is large with respect to the number N of simulations, UCB tends to degenerate into pure exploration. UCT faces the same limitation, and even more so since many arms are considered at each node of the tree, strongly biasing the search towards exploration.

The *progressive widening* (PW) technique has been proposed by Coulom (2007) to handle such cases. The idea is that if $T_s(n)$ is the number of times node s has been visited so far, then the number of arms that can be considered from s should be limited to a fraction m of $T_s(n)$ that grows with $T_s(n)$. Empirical and theoretical studies suggest $m = O(n_s^{1/4})$ (Coulom, 2007; Chaslot et al., 2007; Wang et al., 2009). In practice, the `ArmSet` procedure implementing PW in *BAAL* (algorithm 2), considers a finite set of options for each node (see below), and a new option is added to this set whenever $\lfloor n_s^{1/4} \rfloor$ is incremented by one.

Typically, a single arm will be explored from the root node in the first fifteen tree-walks; an additional arm is considered in the sixteenth random walk; UCB will be used to select among both arms during random walks 16 to 80; a third arm is considered in random walk 81, etc. How to decide which instance to add will be addressed below.

The rationale behind progressive widening is that the better (i.e. the more visited) s, the more careful the investigation of its subtrees should be.

The procedure offers two main advantages (as opposed to e.g. a grid discretization of the domain):

- the growth of the tree in memory is controlled: children are added slowly, one by one (instead of adding a complete pool of state-action nodes when a new state node is stored), and with a focus on more promising parts of the tree (since the number of children depends on the number of visits);

- the consistency of the process is enforced provided that any instance can be selected with non-null probability at any point.

Progressive widening further allows for a seamless integration of existing active learning heuristics in *BAAL*.

Integrating active learning criteria in *BAAL*

The selection of actions (instances) considered at a given time step by `ArmSet` offers some room for the use of prior knowledge. The simplest selection procedure is based on uniform sampling of the instance space. This procedure reflects an agnostic viewpoint, making no assumption about the information held by each instance.

It must be noted however that *BAAL* convergence might be delayed (like other UCT-based algorithms) if the optimal options are considered late during the search: instances introduced later on are clearly disadvantaged compared to the earlier, more investigated, instances.

Furthermore, the active learning literature suggests that some selection criteria—although not optimal—offer generally sound indications in order to select informative instances (section 3.2). Such criteria can seamlessly be integrated in *BAAL* through the progressive widening procedure. Formally, a new instance is added to `ArmSet` when it satisfies the considered criterion, instead of being uniformly selected in the pool of instances. Note that the use of such criteria does not assume or require any hints about the target concept and how it actually labels instances.

4.5.1 *BAAL* with maximal uncertainty

One such criterion is the *maximal uncertainty* (MU) criterion, inspired from Query-by-committee. The *version space* contains functions of $\mathcal{H}$ consistent with the examples observed so far, suited to approximate h^*. Each instance x splits the version space in two: the functions labeling x by 1, and those labeling x by 0. A clever active learning heuristic is to find examples separating the version space the most evenly: the instances for which consistent hypotheses disagree as much as possible, in other words the maximally uncertain instances. This criterion has been studied empirically and theoretically; multiple algorithms enforcing this criterion or related criteria have been proposed (section 3.2).

Query-by-committee (*QBC*, Seung et al., 1992) algorithms were among the first ones to look for uncertain samples. In this respect, maximal uncertainty can be seen as a variation of *QBC*. *QBC* works by randomly sampling hypotheses in the version space, and choosing a given instance only if enough hypotheses disagree about its label, which amounts to looking for "uncertain enough" instances. An analysis of this strategy with a committee of size 2 is for instance presented by Freund et al. (1997). In the case of larger committees, one can either set a threshold of minimum disagreement for an instance to be queried, or rank instances based on the committee disagreement: the top-ranked instances

indeed converge to those with maximal uncertainty if the committee size tends to ∞: this is the approach with which *BAAL* was hybridized, called *QBC-BAAL*.

Maximal uncertainty is integrated within BAAL as follows. At each node (training set s_t) a committee of hypotheses is built by uniformly sampling the version space of s_t (using again a billiard algorithm). Whenever a new instance is to be added to `ArmSet`, the one maximizing the committee disagreement is selected.

Maximal uncertainty is an *aggressive* criterion, thus holding a higher active learning potential than random progressive widening. In counterpart it leads to a less diversified sample; whenever the criterion is under-optimal, the limited exploration will yield poorer performance. It is also more demanding computationally. Of course, any other appropriate active learning criterion can be used in *BAAL* in a similar fashion.

4.6 Chapter Summary

Within this bounded resource setting, active learning was tackled as a reinforcement learning problem: find the sampling strategy aimed at minimizing the overall generalization error for a finite time horizon. This ideal and utterly intractable formalization leads to the proposed *BAAL* algorithm: an approximation of the optimal sampling strategy is learned within a one-player game setting. *BAAL* is inspired from an earlier work devoted to Computer Go (Gelly and Silver, 2007), building upon the tree-structured bandit-based search algorithm UCT (Kocsis and Szepesvari, 2006) and the progressive widening heuristics to deal with a continuous search space (Coulom, 2006; Chaslot et al., 2007; Wang et al., 2009).

Chapter 5

Batch Active Learning Bounds

Batch active learning (Hoi et al., 2006; Sugiyama and Rubens, 2008; Hoi et al., 2009) is the particular case of active learning where the algorithm can query λ examples simultaneously. Parameter λ, referred to as the *batch size*, is the number of simultaneous requests to the oracle. This chapter provides rigorous bounds on the number of iterations before a given precision is reached for batch active learning in binary classification, in particular as a function of λ. The first version of this work was originally published in the proceedings of the twenty-first European Conference on Machine Learning (Rolet and Teytaud, 2010a).

5.1 Motivation

A motivating example of batch active learning setting is when the oracle is a computational code, such as in numerical engineering applications: if λ computing units are available, the oracle code can be launched in parallel on the λ machines. The batch model of complexity only considers the numbers of iterations, which is relevant whenever the computational cost essentially lies in the calls to the oracle function (*expensive* oracle). The underlying assumptions are:

1. the cost of the active learning algorithm is negligible w.r.t. the cost of calling the oracle;
2. at least λ computation units are available.

The quantity of interest when analyzing batch active learning w.r.t. *sequential* active learning (i.e. when $\lambda = 1$), is the *speedup* at λ. To define the speedup, consider a sequential active learning algorithm $\mathbf{L}$, that picks an instance, gets its label and iterates the process until the objective function is learned well enough. The number of iterations that $\mathbf{L}$ needs to learn a function from $\mathcal{F}$ with precision ϵ and confidence δ is its sample complexity, $sc(\mathbf{L}, \epsilon, \delta, \mathcal{F})$. Let us assume

that $\mathbf{L}$ can be extended to a batch active learning algorithm $\mathbf{L}_\lambda$, learning with batches of examples of size λ. The number of iterations required by $\mathbf{L}_\lambda$ is its sample complexity $sc(\mathbf{L}_\lambda, \epsilon, \delta, \mathcal{F})$ *divided by* λ, since $\mathbf{L}_\lambda$ queries λ instances at once (note that in most cases, $sc(\mathbf{L}, \epsilon, \delta, \mathcal{F}) \neq sc(\mathbf{L}_\lambda, \epsilon, \delta, \mathcal{F})$). The *speedup* at λ is the ratio of number of iterations of $\mathbf{L}$ over the number of iterations of $\mathbf{L}_\lambda$, that is $\lambda.sc(\mathbf{L}, \epsilon, \delta, \mathcal{F})/sc(\mathbf{L}_\lambda, \epsilon, \delta, \mathcal{F})$.

With respect to this criterion, *passive* learning—the setting in which instances are selected i.i.d. from the natural input distribution—has a linear speedup: an algorithm querying λ instances at a time in an i.i.d. fashion is λ times faster than the sequential algorithm querying 1 instance at each time. This chapter focuses on a theoretical analysis of the speedup in the case of active learning.

Batch active learning for classification.

Batch active learning has received less attention than sequential active learning. Hoi et al. (2006) assesses the information brought by batches of examples via a criterion based on Fisher information matrix reduction. Guo and Schuurmans (2008) seeks sets of examples with low uncertainty; this goal, cast as an (NP-hard) optimization problem, is handled by an approximation algorithm. Both works provide empirical evidence of the soundness of their strategies. However, they do not provide any formal proof guaranteeing their behavior. Furthermore, we are not aware of any theoretical study of the speedup of batch active learning over sequential active learning, in terms of sample complexity bounds (speedup is in terms of gain with respect to the number of iterations, see section 5.2). Clearly, batch active learning cannot reduce the overall number of evaluations when compared to sequential active learning; the gain relies on the existence of a parallel oracle, simultaneously labeling a batch of λ instances provided by the active learning algorithm.

This work was notably motivated by the task of approximating a computationally heavy numerical code, that require hours or days to compute an outcome given an input. Specifically, this code developed by nuclear physicists from CELIA (Center for Intense Lasers Applications), a research laboratory of the French national institute for nuclear research (the CEA), is concerned with simulating nuclear fusion ignition through lasers, with a low input dimension (5 to 10).

Although the simulation code is not intrinsically parallelized, using multiple processors it can be run simultaneously on multiple inputs, for instance on a single cluster (e.g. a hundred cores), or on a grid (e.g. five thousand cores). In the first case, outputs were given by batches of 100, and in the second by batches of 5000. As can be expected, the second requires much more deployment effort. A main theoretical result of this study is to show that in such a case, it is not worth wasting time and computational power on using the grid, since the speedup of using the grid over using the cluster would be quite low. This result specifically holds for active learning, and would naturally not be true for passive learning scenarios.

The chapter is organized as follows:

- section 5.2 presents the formal background and notations;
- section 5.3 shows lower bounds on the speedup of batch active learning using covering and packing numbers;
- section 5.4 symmetrically presents upper bounds, relying on a "speculative paralellization" algorithm.

The experimental evidence supporting the presented analysis is presented on chapter 6.

5.2 Framework

A sequential active learning algorithm is a sampler-learner pair $\mathbf{L} = (\mathbf{A}, \mathbf{S})$. In the batch case, the sampler noted $\mathbf{S}_\lambda$ should output batches of λ instances; the batch active algorithm is the pair $\mathbf{L}_\lambda = (\mathbf{A}, \mathbf{S}_\lambda)$.

The *speedup* of a parallel algorithm $\mathbf{L}_\lambda$ over its sequential counterpart $\mathbf{L}$ (or equivalently $\mathbf{L}_1$) is the ratio of $\mathbf{L}$'s complexity in terms of number of iterations (i.e. of batch calls to the oracle) on $\mathbf{L}_\lambda$'s complexity.

Only deterministic algorithms are considered here: samplers $\mathbf{S}_\lambda$ map $\mathcal{S}$ to elements of $\mathcal{X}^\lambda$ as opposed to mapping $\mathbf{S}$ on a probability distribution of[1] $(\mathbb{R}^+)^{\mathcal{X}^\lambda}$. The framework of batch active learning is presented in algorithm 4. For the sake of the analysis, we shall use the notion of internal state of algorithm $\mathbf{L}_\lambda$ at iteration n, noted I_n, which reflects the labels associated to examples queried up to n. A function $update_\lambda$ describes how this internal state changes after each batch. Given input domain $\mathcal{X}$, $P_\mathcal{X}$ is a probability measure on this domain.

As in chapter 4, the realizable assumption is made: the target concept $f^* : \mathcal{X} \to \{0, 1\}$ is non-noisy and belongs to hypothesis space $\mathcal{F} \subset \{0, 1\}^\mathcal{X}$. Let us remind the *generalization error* $\rho(f, f^*)$ of an approximation $f \in \mathcal{F}$ of f^* is defined as $P_\mathcal{X}(\{x \in \mathcal{X} | f(x) \neq f^*(x)\})$. Remember that S_{n,f^*} is the set of samples generated by sampler $\mathbf{S}$ after n steps with target function f^*. Similarly, S_{λ,n,f^*} is the set generated by $\mathbf{S}_\lambda$.

We assume that the considered concept class $\mathcal{F}$ has a finite VC-dimension D. It is common to consider finite VC-dimension in active learning settings since the improvement over passive learning is potentially much bigger in this case (Kulkarni et al., 1993). Indeed, for finite VC-dimension concept classes, the number of examples required to learn f^* with precision ϵ, i.e. to find f such that $\rho(f^*, f) \leq \epsilon$, is $N = \Theta(D \log(1/\epsilon))$ in many cases (Balcan et al., 2008).

For a given λ and a given algorithm $\mathbf{L}_\lambda$, the minimum number of iterations for reaching precision ϵ is

$$scm(\mathbf{L}_\lambda, \epsilon, \mathcal{F}) = \min\{n \in [[0, +\infty]]; \max_{f \in \mathcal{F}} \rho(f, \mathbf{A}(S_{\lambda,n,f})) < \epsilon\}$$

[1] The lower bounds can nonetheless be extended to stochastic cases within logarithmic dependencies on the risk δ (i.e. case with confidence $1 - \delta$). Precisely, the sample complexity is increased by a factor $O(\log(1/\delta))$ if we request that the algorithm finds the solution with probability $1 - \delta$. Upper bounds can also be extended to the stochastic case.

Algorithm 4 Batch active learning algorithm $\mathbf{L}_\lambda$.

Given I_0 the initial state *// Global state of the algorithm*
  Parameter Batch size λ, the number of visited points per iteration.
  $n = 0$
  repeat
    $f_n = \mathbf{A}(S_{\lambda,n,f^*})$
    $(x_{n\lambda+1}, \ldots, x_{n\lambda+\lambda}) = \mathbf{S}_\lambda(S_{\lambda,n,f^*})$
    for $i \in [[1, \lambda]]$ **do**
      $y_{n\lambda+i} = f^*(x_{n\lambda+i})$ *// label λ instances at once*
    end for
    $I_{n+1} = update_\lambda(I_n, x_{n\lambda+1}, \ldots, x_{(n+1)\lambda}, y_{n\lambda}, \ldots, y_{(n+1)\lambda})$
    $n++$
  until Computational budget exhausted
  Return $\mathbf{A}(\{(x_1, y_1), \ldots, (x_{n\lambda}, y_{n\lambda})\})$

in accordance with the minimax sample complexity defined in section 2.1.2. The smallest achievable number of iterations for any algorithm is then

$$scm_\lambda(\epsilon, \mathcal{F}) = \inf_{\mathbf{L}_\lambda} scm(\mathbf{L}_\lambda, \epsilon, \mathcal{F})).$$

Although scm_λ depends on the considered function class $\mathcal{F}$, it will be noted $scm_\lambda(\epsilon)$ when no confusion is to fear. Let $\mathbf{M}(\epsilon, \mathcal{F})$ be the ϵ-packing number of $\mathcal{F}$, i.e. the maximum number of hypotheses in $\mathcal{F}$ with pairwise distance ρ at least ϵ, see section 3.1.5. In the sequel, unless stated otherwise, we will assume that there exists constants C and M such that the log-packing numbers of target function class $\mathcal{F}$ are at least $-CD\log(\epsilon/M)$. Formally,

$$\forall \epsilon, \mathbf{M}(\epsilon, \mathcal{F}) \geq (M/\epsilon)^{(C\times D)}. \tag{5.1}$$

The product $C \times D$ in equation 5.1 stems from many results emphasizing some constant C, and the VC-dimension D (Dasgupta, 2006), such as, for example, the well-known case of homogeneous linear separators [2] of the sphere with homogeneous distribution (Freund et al., 1997; Dasgupta et al., 2005; Balcan et al., 2007), in which case D is equal to the dimension of the domain. In this chapter, D always refer to the VC-dimension of the considered function class.

5.3 Lower Bounds

Equation 5.1 leads to (see Kulkarni et al., 1993; Vidyasagar, 1997):

$$scm_1(\epsilon) \geq \lceil CD\log(M/\epsilon)\rceil. \tag{5.2}$$

Equation 5.2 states a lower bound on the sample complexity of active learning, and has the following consequence (which is a lower bound on the sample

[2] that is, linear separators whose value in the null vector is 0

complexity of batch active learning):

$$scm_\lambda(\epsilon) \geq \lceil CD \log(M/\epsilon)/\lambda \rceil \tag{5.3}$$

When λ calls to the oracle are launched in parallel, equation 5.3 provides a straightforward lower bound with linear speedup, which is the ultimate limit for batch active learning (as good as passive learning speedup). A first contribution is to show a more stringent lower bound, log-linear in the speedup, by extending equation 5.2.

Theorem 2 (Lower bound for batch active learning)**.** *If $\mathcal{F}$ has VC-dimension D and packing numbers satisfying*

$$\boldsymbol{M}(\epsilon, \mathcal{F}) \geq (M/\epsilon)^{C \cdot D}, \tag{5.4}$$

then the following holds for $\lambda > 1$:

$$scm_\lambda(\epsilon) \geq CD \log(M/\epsilon)/\log(K) \tag{5.5}$$

where $K = \lambda^D$ if $D \geq 3$, $\lambda^D + 1$ if $D \leq 2$, i.e.

$$scm_\lambda(\epsilon) \geq C \log(M/\epsilon)/(\log(\lambda)). \tag{5.6}$$

Remark. K is an upper bound on the number of possible classifications of λ points, given a class of function with VC-dimension D. $K = \lambda^D$ stems from Sauer (1972). K is exponential in D, but finite, thanks to the finite number of possible classifications of λ points when the VC-dimension is D. Having this upper bound on the number of reachable states for one batch, the number of possible branches in a run of the algorithm can be bounded accordingly.

Proof. Consider an algorithm realizing $scm_\lambda(\epsilon)$.

As the sampling algorithm is deterministic, there is one and only one possible value for the λ-uple $(x_1, \ldots, x_\lambda)$, independently of f:

$$(x_1, \ldots, x_\lambda) = \mathbf{S}_\lambda(\emptyset).$$

Thanks to the finite VC-dimension and to Sauer's lemma, there are at most K possible values for $y_1, \ldots, y_\lambda$; therefore there are at most K possible values for internal state I_1 (since the sampling algorithm is assumed to be deterministic).

Similarly, for each possible value of I_1, there are at most K possible values for I_2; therefore the total number of possible values for I_2 is at most K^2.

By induction, there are at most K^i possible values for I_i. After $scm_\lambda(\epsilon)$ iterations, each possible state $I_{scm_\lambda(\epsilon)}$ corresponds to a function learned with $\lambda scm_\lambda(\epsilon)$ examples. Since the algorithm realizes the bound $scm_\lambda(\epsilon)$, for any 2 oracle functions distant of ϵ or more, the algorithm must have 2 different states. Thus, the final number of states is at least as big as the packing number: $K^{scm_\lambda(\epsilon)} \geq \mathbf{M}(\epsilon, \mathcal{F})$. As a consequence,

$$scm_\lambda(\epsilon) \geq \log(\mathbf{M}(\epsilon, \mathcal{F}))/\log(K). \tag{5.7}$$

Equations 5.7 and 5.4 yield the expected result. □

5.4 Upper Bounds via Speculative Parallelization

The next result shows that this bound is tight, at least asymptotically ($\lambda \to \infty$).

Theorem 3 (Upper bound for batch active learning)**.** *With the notations of algorithm 4, assume $\mathcal{F}$ has VC-dimension D. Let $K = \lambda^D + 1$, $b \geq 1$ and*

$$\lambda' = \lambda \frac{K^b - 1}{K - 1}. \tag{5.8}$$

Then the following holds:

$$scm_{\lambda'}(\epsilon) \leq \lceil scm_\lambda(\epsilon)/b \rceil \tag{5.9}$$

Remark. Equation 5.9 leads to

$$scm_{\lambda'}(\epsilon) = O\left(\lceil scm_\lambda(\epsilon)/\log(\lambda') \rceil\right) \tag{5.10}$$

for fixed b and λ. This is a logarithmic speedup: see for instance that if $\lambda = 1$, then $\forall n, scm_{2^n}(\epsilon) = O(\frac{scm_1(\epsilon)}{n})$

Proof. The proof exhibits an algorithm realizing equation 5.9. Consider an algorithm $\mathbf{L}_\lambda = (\mathbf{A}, \mathbf{S}_\lambda)$ realizing $scm_\lambda(\epsilon)$ and consider some $b \geq 1$. Define $\lambda' = \lambda \frac{K^D - 1}{K - 1}$. Consider, then, another algorithm $\mathbf{L}_{\lambda'} = (\mathbf{A}, \mathbf{S}'_\lambda)$ which generates λ' points by simulating $\mathbf{L}_\lambda$ on b steps; if $\mathbf{L}_\lambda$ has nth internal state I_n, then $\mathbf{L}_{\lambda'}$ has n^{th} internal state $I'_n = I_{Dn}$. At each iteration:

- $\mathbf{S}_{\lambda'}$ simulates the K^b possible internal paths

$$(I_{bn}, I_{bn+1}, \ldots, I_{bn+b})_k \text{ with } k \in [[1, K^b]] \tag{5.11}$$

  and generates for each iteration all the λ' points visited in any of those paths. At state I_{bn}, λ points are to be labeled. Then, there are K possible states I_{bn+1} resulting in Kλ other points. Repeating the process for $I_{bn+2}, \ldots$, one can see that λ' as in equation 5.8 is enough;

- the target f is computed at these λ' points. It is then possible to figure out which path among the K^b possible paths is the one that would actually have happened during b steps of $\mathbf{L}$;

- $update_{\lambda'}$ is the result of $update_\lambda$ for the path selected by $\mathbf{S}_{\lambda'}$[3];

- the output of $\mathbf{A}(S_{\lambda',n,f})$ is the output of $\mathbf{A}(S_{\lambda,n,f})$ on the λ points visited in the selected path.

□

[3]Therefore, only $b\lambda$ points are actually used among the λ' labeled points

Note that most points which the target value has been computed are discarded in $S_{\lambda',n,f}$. While discarding them is useful to establish the tightness of the bounds (equation 5.10), it is clear that in real world applications, we should rather keep all labeled points. Nevertheless, theorem 3 shows that keeping them only improves the asymptotic speedup by a constant factor.

Equations 5.6 and 5.9 show that $\log(\lambda)$ is the optimal speedup when no assumption on λ are made: theorem 3 shows that in all cases, a logarithmic speedup is achievable and Theorem 2 shows that we cannot do much better for λ large. The rate of batch active learning is therefore at least logarithmic in λ, and for λ large enough at most logarithmic. From the passive case, it is clear that even with small λ the speedup is at most linear.

The remaining question is what happens for moderate values of λ, and in particular if the dependency in D can be removed by asking sufficiently many simultaneous queries.

A third result is that $\lambda = D$ leads to a nearly linear speedup for some hypothesis space $\mathcal{F}$. This removes the dependency in D in runtimes—this means that the curse of dimensionality can be broken with a batch size of D, whereas $\lambda > D$ will only provide a logarithmic speedup eventually.

Being reminded that a binary hypothesis on $\mathcal{X}$ can be seen as a subset of $\mathcal{X}$, we map by convention each hypothesis f of $\mathcal{F}$ to a set X of $\mathcal{X}$ such that $X = \{x|\ f(x) = 1\}$.

Theorem 4 (Linear speedup until $\lambda \leq D$)**.** *Given any $D > 0$, consider $\mathcal{F}_D = \{[0, x], x \in [0, 1]^D\}$. Then, for some $M > 0, M' > 0, C > 0, C' > 0$ independent of D,*

$$\exists \epsilon_0, \forall \epsilon < \epsilon_0, scm_1^{\mathcal{F}_D}(\epsilon) \geq CD \log(M/\epsilon) \tag{5.12}$$

and

$$\exists \epsilon_0, \forall \epsilon < \epsilon_0, scm_D^{\mathcal{F}_D}(\epsilon) \leq C' \log(M'/\epsilon). \tag{5.13}$$

One can check that the VC-dimension of $\mathcal{F}_D$ is D. Equations 5.12 and 5.13 state the linear speedup for batch active learning with $\lambda = D$ for this family of functions (within constants C and C').

Proof. We first show that the following holds:

$$\exists C > 0, \forall D, \exists \epsilon_0, \forall \epsilon < \epsilon_0, \mathbf{M}(\epsilon, \mathcal{F}_D) \geq M/\epsilon^{(C \times D)}. \tag{5.14}$$

Equation 5.14 is a version of equation 5.1 modified for considering only ϵ small; it is weaker than equation 5.1 and sufficient for our purpose.

Equation 5.14 is proved as follows:

- Let us give a lower bound on the packing number of $\mathcal{F}_D$. First, notice that there is a one-to-one mapping between sets $[0, x]$ of $\mathcal{F}_D$, and elements x of $[0, 1]^D$: $\mathcal{F}_D$ can be mapped to $[0, 1]^D$. We will now prove a lower bound on the packing numbers of $[\frac{1}{2}, 1]^D$, which is also a lower bound on $[0, 1]^D$ and thus on $\mathcal{F}_D$

- For x and y in $[\frac{1}{2}, 1]^D$, the L^1 distance between $[0, x]$ and $[0, y]$, which is the generalization error between the two corresponding hypotheses, is lower bounded by $||x - y||_1$.

- Consider a grid of edge 2ϵ in $[1/2, 1]^D$: it contains $\Theta(1/\epsilon^D)$ points. Each point of this grid corresponds to a hypothesis of $\mathcal{F}_D$, and any two hypotheses $[0, x]$ and $[0, y]$ of the grid will have a generalization error of at least $\|x - y\|_1 \geq \epsilon$. Thus, the ϵ-packing number for the hypothesis subset corresponding to $[1/2, 1]^D$ is at least $\Theta(1/\epsilon^D)$: this is a lower bound for the packing number of $\mathcal{F}_D$.

- This shows equation 5.14 for $C = 1$.

Then, equation 5.14 classically leads to equation 5.12, using the same proof as that of equation5.2 from equation 5.1 (section 5.3). This result establishes equation 5.12.

Let us now prove equation 5.13, by considering the following algorithm described at iteration n, with $\lambda = D$:

- $\mathbf{S}_\lambda$ prepares the batch $(x_{n\lambda+1}, \ldots, x_{(n+1)\lambda})$ as follows: for each $x_{n\lambda+i}$, all coordinates $j \neq i$ are set to 0. The i^{th} coordinate of $x_{n\lambda+i}$ is chosen by looking at the $n-1$ previous points in position i of each of the $n-1$ previous batches. It is defined as the middle of the segment defined by the lowest previously-observed i^{th} coordinate whose label is 0, and the highest previously observed i^{th} coordinate whose label is 1. More formally: [4]

$$(x_{n\lambda+i})_i = \frac{1}{2}\left(\min_{n' \leq n}\{(x'_{n'\lambda+i})_i | y'_{n'\lambda+i} = 0\} + \max_{n' \leq n}\{(x'_{n'\lambda+i})_i | y'_{n'\lambda+i} = 1\}\right). \quad (5.15)$$

- $\mathbf{A}$ selects any function $f_n \in \mathcal{F}_D$ which is consistent with $x_1, \ldots, x_{n\lambda}$.

At a given iteration n each point $x_{n,i}$ of the batch of size λ makes sure that the domain will be halved along the i^{th} coordinate. Thus, after N iterations, it is known that the target oracle/classifier is in a square of edge size 2^{-N}. As a consequence, precision ϵ is reached in at most $\Theta(\log(1/\epsilon))$ iterations, which shows equation 5.13. □

This theorem shows that, at least for $\mathcal{F}$ as above, we can have a linear speedup until $\lambda \leq D$; this result establishes the tightness of equation 5.3 for $\lambda \leq D$—similarly to the tightness of equation 5.6 (*i.e.* logarithmic speedup) shown by equation 5.10 for λ large.

[4] In equation 5.15, if no point $x_{n'\lambda+i}$ has been labeled as 0, the minimum is set to 1; equivalently, if no point has been labeled as 1, the maximum is set to 0.

5.5 Chapter Summary

It has been shown that batch active learning exhibits:

- a linear speedup until $\lambda = D$ for some families of target functions;
- a speedup at least logarithmic in all cases;
- and a logarithmic speedup at most for λ large.

Please note that the logarithmic speedup is a speculative parallelization result. The point is not to analyze the convergence rate of active learning in general, but to emphasize that *any* active learning algorithm can be transformed into a batch active learning algorithm (with λ computation units) which simulates it with speedup b, where b is logarithmic as a function of λ.

Chapter 6

Experiments

The two claims made in chapters 4 and 5 regarding the learning of expensive functions are confronted to experimental evidence in this chapter.

Chapter 4 proposed a formalization finite horizon active learning as a Markov decision process. The claim is that a sound and theoretically tractable method can approximate the optimal learning strategy with arbitrary precision. A proof of principle of this claim is made in section 6.1 by applying different versions of *BAAL* to linear separators, and comparing it to a variation of the classic *QBC* algorithm.

Chapter 5 establishes two results:

1. the gain provided by parallelizing the oracle (batch active learning) is asymptotically small (logarithmic in λ);
2. the gain can be high (linear in λ) for a low degree of parallelization (same order as the VC-dimension).

Section 6.2 provides some simple experiments with two active learning algorithms that confirm this trend.

6.1 *BAAL* Experiments

This section reports on the experimental study of *BAAL*. After describing the experimental goal and setting, empirical results are discussed comparatively to random active learning (passive learning), and a variation of the Query-by-Committee (*QBC*) approach (Seung et al., 1992; Freund et al., 1997).

6.1.1 Goal of experiments

The main questions investigated in the experiments regard the theoretical and computational performance of *BAAL*. Specifically:
Question 1 (Optimality): Does the stand-alone *BAAL*, where the progressive

widening heuristics involves a uniform instance selection, converge to an optimal strategy; does it match maximal uncertainty results when these are known to be quasi-optimal under certain conditions (Dasgupta, 2005)?
Question 2 (Flexibility): As shown in section 4.5, *BAAL* can embed existing active learning criteria, e.g. *QBC*, within the progressive widening heuristics. Does the use of *QBC* within *BAAL* improve i) on stand-alone *BAAL* ? ii) on stand-alone *QBC*?
Question 3 (Tractability): Does *BAAL* yield good enough results on a reasonable computational budget? In this respect, it is hoped that billiard algorithms allow for better time complexity and scalability than straightforward approaches such as reject.

6.1.2 Experimental setting

Following various works on active learning (Freund et al., 1997; Dasgupta et al., 2005; Balcan et al., 2007), the instance space $\mathcal{X}$ considered in these experiments is the unit sphere of $\mathbb{R}^d$.

The hypothesis space $\mathcal{H}$ is restricted to linear classifiers, a.k.a. separating hyperplanes. The choice of this search space is motivated by the fact that it has been thoroughly studied from a theoretical perspective (Freund et al., 1997; Dasgupta et al., 2005; Balcan et al., 2007) although these results did not lead to experimental studies to the best of the author's knowledge. Accordingly, there are many hints at upper and lower bounds regarding the optimal sample complexity in this case, that will be used to assess *BAAL* performance. The goal of this experimental study is to provide a proof of principle regarding the validity of this new active learning approach; still, it must be emphasized that *BAAL* is not limited to linear hypothesis spaces (a kernelized extension will be discussed in section 11.3).

Two variants of *BAAL* will be considered. *BAAL* stand-alone (or *BAAL* for short), uses a uniform selection of instances within progressive widening, whereas *QBC-BAAL* uses a committee-based selection of instances. More precisely, in each node (training set s_t), a committee of 100 hypotheses uniformly selected in $\mathcal{H}(s_t)$ is built, and a set of $10,000$ instances uniformly drawn from $\mathcal{X}$ is sampled and ordered by the committee disagreement.

Whenever a new action is to be considered (i.e. $\lfloor n(s_t)^{1/4} \rfloor$ is increased by one, according to section 4.5), the first instance not yet considered in the ordered set is returned by `ArmSet`.

A hypothesis h is represented by the unit vector w_h normal to its separating hyperplane ($w_h \in \mathbb{R}^d$, $||w_h||_2 = 1$): $h(x)$ is positive if and only if the dot product $w_h.x$ is positive. The lack of prior knowledge is accounted for by setting distribution $P_{\mathcal{H}}$ to the uniform distribution on the unit sphere of $\mathbb{R}^d$.

A run of *BAAL* proceeds as follows:

1. The target concept h^* is uniformly selected in $\mathcal{H}$.

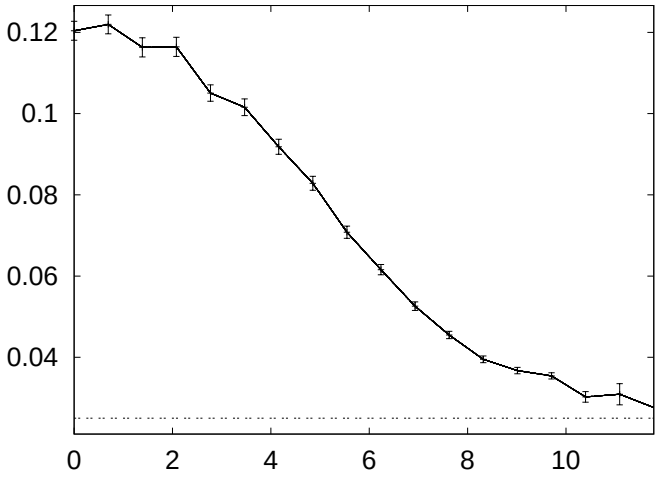
2. For $t = 0$ to T, where T is the horizon time and the initial training set s_0 is empty:
   (a) The game tree rooted on s_t is constructed using N game simulations (tree walks), respectively using an instance pool ordered randomly (for *BAAL*) or via maximal uncertainty (for *QBC-BAAL*) in the progressive widening heuristics;
   (b) The best instance, i.e. the most visited one at the first level of the s_t rooted tree, noted x_{t+1}, is selected and labeled by the target concept (oracle) h^*;
   (c) $s_{t+1} = s_t \cup \{(x_{t+1}, h^*(x_{t+1}))\}$.
3. From s_T, the T-size training set built by *BAAL* (or *QBC-BAAL*) and labeled by the current target concept h^*, a hypothesis h_T is learned.
4. The performance of the run, that is, the generalization error $\rho(h, h^*)$ defined as $P_{\mathcal{X}}(h^*(x) \neq h(x))$ is computed: in this case, it is simply the dot product of vectors w_h and w_{h^*}.

At each run, hypothesis h_T is learned by learning algorithm **A**. Note that this learning algorithm is already called in *BAAL* each time to compute a reward when reaching a leaf node, by learning from a "surrogate hypothesis" associated to the tree walk (section 4.3, algorithm 2, routine **t**Tree-walk). This algorithm was originally set to uniformly pick a hypothesis from version space $\mathcal{H}(s_T)$. It was thereafter modified taking inspiration from Herbrich et al. (2001), to compute and return the center of mass of the version space. This learning algorithm, referred to as *Bayes point machine*, was reported to outperform the canonical SVMs by Herbrich et al. (2001). In the following, the Bayes point machine is used as learning algorithm.

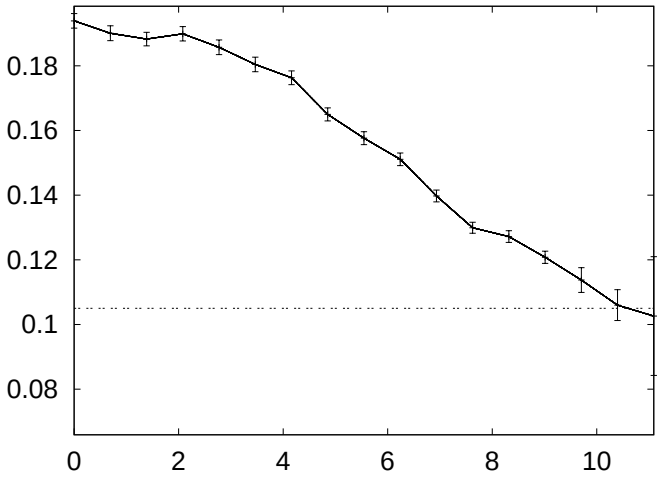
BAAL and *QBC-BAAL* performances are averaged over 400 independent runs for each 3-uple (d, T, N) (dimension, time horizon, simulation number). The number N of simulations, controlling the computational cost, ranges in the powers of 2: $N \in \{1, 2, 4, \ldots 2^{12}\}$. The reported experiments consider $(d = 4, T = 15)$ and $(d = 8, T = 20)$ to assess the scalability of the approach. The performance is plot against the number N of tree-walks on figures 6.1 and 6.2, indicating the average performance (plain line) and the standard deviation (vertical bars). The performance of *BAAL* stand-alone (respectively *QBC-BAAL*) is assessed comparatively to the passive learning (resp. the *QBC* greedy active learning) baseline. The baseline performances correspond to those obtained for $N = 1$. Actually, *BAAL* can be viewed as an algorithm providing an "educated" sampling strategy, where the "educated sampler" is based on a computational budget of N simulations; the baseline, non-educated and uniform sampler, accordingly corresponds to $N = 1$.

6.1.3 Performance and scalability

Figure 6.1 displays the overall performance of stand-alone *BAAL* versus the computational budget N (in log scale).



(a) d = 4, T = 15, stand-alone $BAAL$



(b) d = 8, T = 20, stand-alone $BAAL$

Figure 6.1: Experiments on stand-alone $BAAL$: Generalization error vs the number of simulations N in log scale. Plot (a) reports the results obtained for instance space dimension $d = 4$ and time horizon $T = 15$; plot (b) report the results for $d = 8, T = 20$. The performance of the baseline correspond to the performance obtained for $N = 1$ (at the origin of the curve). The horizontal line reports the results obtained for the stand-alone QBC (with a committee of size 100) strategy.

The competence of $BAAL$ in terms of Active Learning is visible as stand-

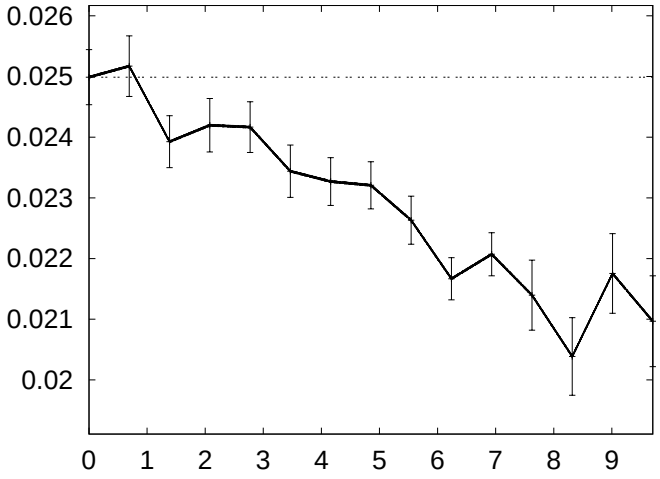
alone *BAAL* strongly outperforms the passive learning baseline ($N = 1$): the generalization error significantly decreases as N increases. For a given computational budget N, the improvement on passive learning is higher in small dimension, as could have been expected. Nevertheless, the improvement is significant both in dimensions 4 and 8 even for a small computational budget ($N \sim 2^6$).

The performance of stand-alone *BAAL* is further assessed by comparison with that of a stand-alone *QBC* using a committee of size 100. Freund et al. (1997) and Dasgupta (2005) show that the the maximal uncertainty heuristics (approximated by a *QBC* selection with large committee) is almost optimal, up to logarithmic terms, in the linear setting. It is thus satisfactory to see that stand-alone *BAAL* steadily approaches the *QBC* reference performance. Since this reference is only quasi-optimal, it is not inconsistent that *BAAL* can even outperform the *QBC* reference in dimension 8 when the number of simulations gets larger: by construction, it is designed to converge to the actual optimum. These results suggest a positive answer to the *Optimality* question (section 6.1.1): stand-alone *BAAL* is a competent, criterion-agnostic Active Learner, which might work well in the absence of prior knowledge about the instance selection, and which matches the known optimal performance in a simple hypothesis space.

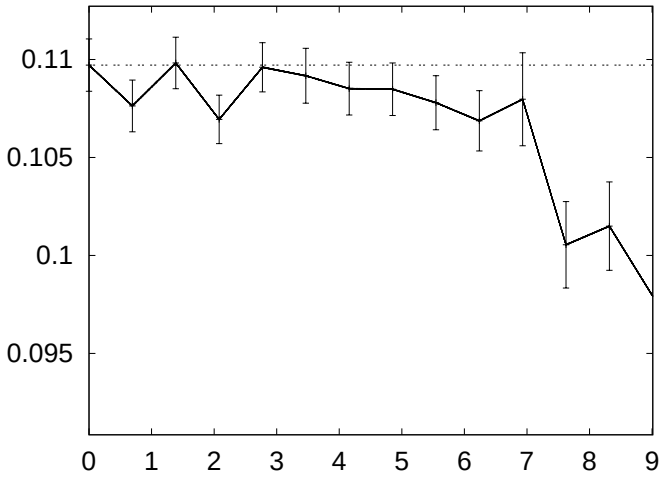
Another picture is provided by the performance of *QBC-BAAL* (figure 6.2), suggesting that *QBC-BAAL* can get the "best of both worlds". In dimensions 4 and 8, *QBC-BAAL* outperforms the *QBC* baseline in a statistically significant way; it overcomes the theoretical limitations of *QBC* with regards to optimality [1]. These results suggest a positive answer to the *Flexibility* question (section 6.1.1): *BAAL* can be efficiently hybridized with the *QBC* criterion, and the hybrid *QBC-BAAL* improves on both stand-alone *BAAL* and stand-alone *QBC* in this simple setting. The significance of this result comes from the fact that maximal uncertainty approaches are among the most widely used criteria in the active learning literature; further research will investigate the incorporation of other active learning criteria within *BAAL*.

The *QBC* baseline presented here is the one with a committee of size 100 embedded in *BAAL*. Since maximal uncertainty is quasi-optimal in the linear separator setting, it is obvious that using a small committee (e.g. of size 2 as in the work of Freund et al., 1997) would have resulted in poorer generalization error for the same sample complexity. On the other hand, simple experiments (not presented here) performed with committee sizes of $10^3, 10^4$ and 10^5 did not provide a significant improvement (relatively to the results presented here) over the committee size of 100, which is why this committee size is referred to as a baseline.

[1] w.r.t. the optimal sample complexity in dimension d, *QBC*'s complexity is approximately $O(\log(1/\epsilon) * d)$ times larger.



(a) d = 4, *QBC-BAAL*



(b) d = 8, *QBC-BAAL*

Figure 6.2: Experiments on *QBC-BAAL*: Generalization error vs the number of simulations N in log scale. The committee size in each node is set to 100. Complementary experiments with a 10000-hypothesis committee do not show significant improvement over a 100-hypothesis committee.

6.1.4 Computational cost and tractability

The main computational cost incurred by *BAAL* comes from the selection of the hypothesis in the version space of the current training set (last algorithm step in *BAAL*, learning a hypothesis from s_T).

Dim. d	Horizon T	Reject	Billiard
4	10	9s	4s
	20	1h10m	14s
8	10	30s	6s
	20	>3h	17s

Table 6.1: Computational cost of Stand-alone *BAAL* (with $N = 16,000$ tree-walks): reject-sampling and billiard-sampling selection of hypotheses for dimension $d = 4$ and $d = 8$, time horizon $T = 10$ and $T = 20$.

The first heuristics used to sample the version space was a rejection-based sampling: uniformly drawing h in $\mathcal{H}$, until h belong to $\mathcal{H}(s_T)$. However, when active learning is successful, the size of the version space decreases exponentially fast, adversely affection the rejection heuristic.

This heuristics has been replaced by a billiard-based sampling of the version space (section 4.4). Without downgrading the performance, the billiard-based sampling reduces the computational cost by several orders of magnitude (table 6.1), and features an outstanding scalability with respect to the dimension d of the search space.

The billiard-based sampling of version spaces thus supports a computationally efficient active learning with *BAAL*. It is expected that, even in larger dimensions, the computational cost will remain negligible compared to the cost needed to label the instances. Let us remind that for targeted application domains such as numerical engineering, the cost of labeling an instance (e.g. checking whether some mechanical design parameters will enforce the device compliance with the desired requirements, using finite element methods) might require several days of computation on high-performance computers.

6.1.5 Conclusion

The experimental validation of *BAAL* investigates three main questions: the convergence towards the optimal performance when it is known; the ability to take advantage of competent active learning criteria, such as maximal uncertainty (Freund et al., 1997), and improve on the greedy use of these criteria; the computational tractability of the overall active learning scheme. The experiments discussed in the paper show that the answer to all three questions is positive in the simple case of a realizable setting with linear hypotheses.

6.2 Batch Active Learning Experiments

As far as the instance labeling step can be parallelized in a simple way, batch active learning has been considered in chapter 5 providing both lower and upper bounds on the iteration gain needed to reach a given precision. This section presents experiments with both a simple and a more sophisticated AL algorithm, supporting the tightness of the established bounds.

6.2.1 Experiments with naive active learning

A simple batch active learning algorithm for $\mathcal{F} = \{[0, x]; x \in [0, 1]^d\}$ (VC-dimension $D = d$) is experimented here. The new instances $x_{n\lambda+1}, \ldots, x_{(n+1)\lambda}$ are uniformly drawn in the uncertainty region. Formally:

$$\mathcal{V} = \{x \in [0, 1]^d; \forall j \in [[1, n\lambda]] y_j = 0 \Rightarrow \neg(x_j \leq x)\}$$

$$\cap\{x \in [[0, 1]^d; \forall j \in [[1, n\lambda]] y_j = 1 \Rightarrow x_j \leq x\}.$$

Note that this parallel algorithm is straightforward to derive from its sequential counterpart that queries one random sample from the version space at each iteration. This is not true of all active learning algorithms: in many cases, it is not clear how to efficiently turn a sequential active learning into a parallel one.

For each batch parameter λ(on the X-axis), figure 6.3 reports the inverse of the number of iterations for reaching precision $0.001d^2$, depending on λ (on the Y-axis), thus displaying *rates* of convergence. Each point is averaged over 33 runs.

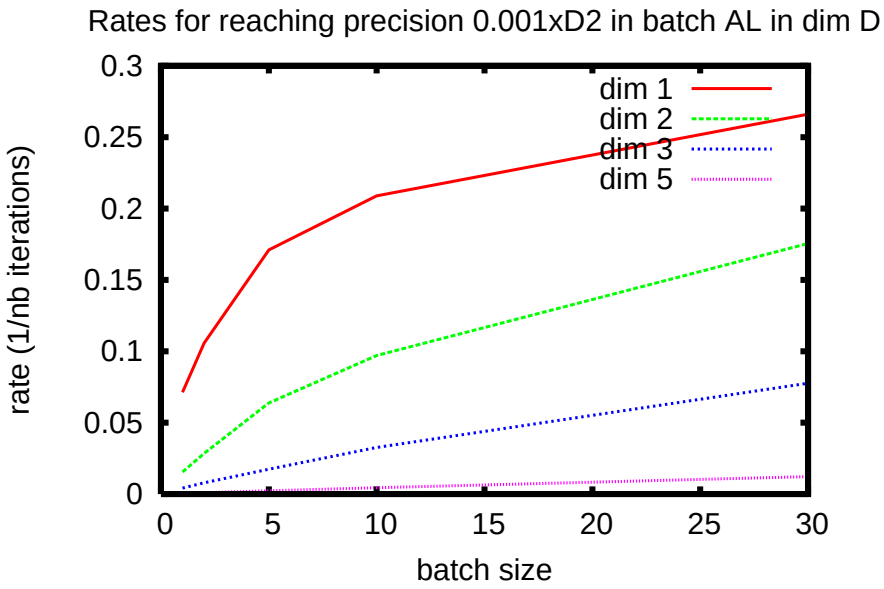


Figure 6.3: speedup of batch active learning for a simple active learning algorithm (see text).

6.2.2 Experiments with maximal uncertainty

This part of the experiments is concerned with a straightforward adaptation of the classical active learning heuristic called *maximal uncertainty* (introduced in

4.5.1) to the batch setting. The idea is to choose the most *uncertain* examples, meaning the ones for which many approximations of the target function that are still good candidates disagree on the label.

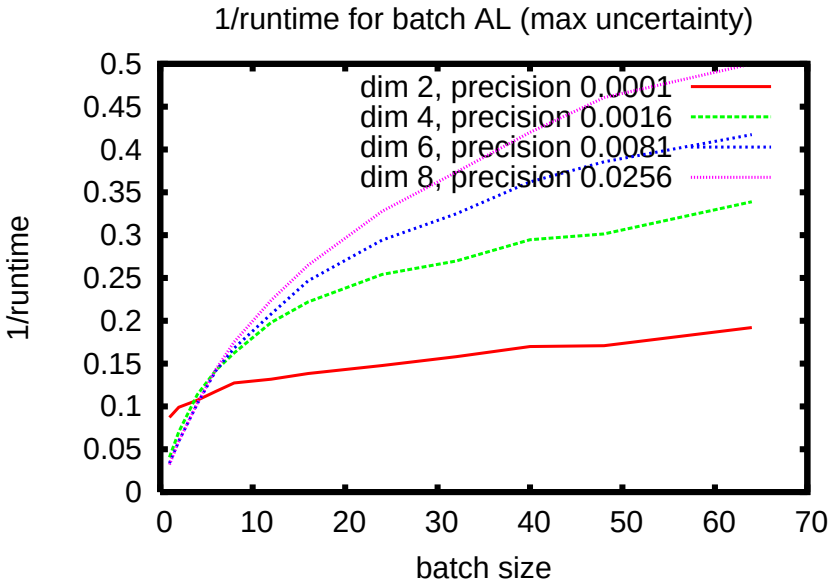


Figure 6.4: speedup of batch maximal uncertainty active learning.

Experiments learn homogeneous linear separators of $\mathbb{R}^d$, where examples lie on the hypersphere $\mathbb{S}^{d-1}$ for dimensions $d = 2, 4, 6, 8$. As mentioned earlier, this setting has been widely studied for sequential active learning (Balcan et al., 2007; Dasgupta et al., 2005; Freund et al., 1997) and is therefore well suited to a speedup analysis for the batch setting.

In such a setting, for $d > 2$, an infinite number of points of the hypersphere maximize the uncertainty given previously witnessed instances (note that there would only be a single point maximizing uncertainty for $d = 2$). Hence, a relevant batch active learning strategy consists in selecting λ of those points maximizing uncertainty, at each iteration. In contrast with the simple batch active learning of section 6.2.1, this strategy is not necessarily the most efficient w.r.t. the batch setting; we shall see however that it still provide significant speedups.

Batch sizes are $\lambda = 1, 2, 4, 6, 8, 12, 16, 20, 24, 32, 40, 48, 64$. Precision is set to $0.0001(d/2)^4$. For each (d, λ), the rate are averaged over 160 runs.

6.2.3 Interpretation

In both cases, the results resemble the expected behavior: a steady (linear) speedup for small batch sizes, when $\lambda < d$ (where d is the dimension), and

a slow, logarithmic-like speedup when λ becomes much bigger than d. These results suggest that the parallel gain can be linear in high dimension and low parallel factor, although it asymptotically goes down to a logarithmic factor.

Remarks

- It was pointed out that the instance maximizing uncertainty is unique in the case $d = 2$, and therefore any batch should consist of the same point λ times. However, figure 6.4 still shows a small improvement when the batch size increases. This is because the computation of the maximal uncertainty relies on stochastic approximations: the batches consist of many points that are slightly different, and thus still bear a little bit more information than only one point.

- The behavior of the speedup for values of λ in-between D and λ large, although depicted in the experiments, was not studied theoretically. It might be that the speedup can remain good even for a while when $\lambda > D$—but asymptotically it will end up in a logarithmic improvement.

- The rates for high dimensions seem low (although linear) on the first figure, while they seem higher in the second. This is due to the fact that the precision to be reached is bigger in the second figure than in the first, in an attempt to be more "fair" to high dimensions—since for a given number of examples, a concept is harder to learn if the dimension of the domain is higher.

Perspectives regarding the work presented in this part will be discussed in the concluding chapter (chapter 11).

Part III

Optimizing Expensive Functions

Chapter 7

Background and framework

This chapter lays down the formal background related to expensive optimization. Optimization, formerly known as mathematical programming, has been a research topic for ages. The first section of this chapter introduces some definitions with respect to the various kinds of optimization approaches, and situates the present work on optimizing expensive noisy functions. Most specifically, sections 7.2, 7.3 and 7.4 respectively introduce estimation of distribution algorithms, noisy optimization and *races* techniques.

7.1 Black-box Optimization

This section aims at situating the presented work within the broad field of optimization, referring the reader to e.g. Boyd and Vandenberghe (2004); Bertsekas and Bertsekas (1999) for a comprehensive presentation of the optimization field.

Vocabulary

Optimization problems are most generally formalized as the search of an optimum of a function $f : \mathcal{X} \to \mathcal{Y} \subset \mathbb{R}$, where f is commonly referred to as *objective function*, *fitness function*, or *fitness landscape*. We can assume that the optimization problem considered in the following is a minimization one without loss of generality. Possible locations of the optimum are called candidate solutions, or simply points. Many optimization problems are *constrained*: although a set $\mathcal{X}$ represents the input space, the optimum is only sought for on points that satisfy a number of constraints. Those points form the *feasible region* (or feasible set, or search space, or solution space). Given a fitness $f : \mathcal{X} \to \mathcal{Y}$, the optimum sought for in $\mathcal{X}$ is often written x^* (although it may not be unique).

Performance measure

The performance of an optimization algorithm in continuous optimization is most often expressed by means of distances defined on domain $\mathcal{X}$ or on values

$\mathcal{Y}$. The most common measure in optimization is the distance to the optimum

$$\rho(x, x^*) = \|x - x^*\|$$

where $\|.\|$ is most commonly the L_2 norm or the L_1 norm on $\mathcal{X} \subset \mathbb{R}^d$. Another classic measure is the distance in terms of fitness value

$$\rho(x, x^*) = \|f(x) - f(x^*)\|$$

where $\|.\|$ is a norm on $\mathcal{Y}$.

Problem categories

One classically distinguishes continuous and discrete optimization depending on the structure of the input space. Discrete optimization deals with objectives for which the optimum is sought among a countable number of candidate solutions. Conversely, continuous optimization is concerned with finding real-valued solutions, where $\mathcal{X} \subset \mathbb{R}^d$.

Discrete optimization consists in *integer programming*, in which the domain is continuous but only integer solutions are considered, and *combinatorial optimization* in which the domain itself is countable, and has a particular structure, such as a partial order—since otherwise only exhaustive search can lead to the optimum.

Only continuous optimization will be considered in the following. One characteristic particularly relevant to continuous optimization algorithms is the convexity of the objective function, since it guarantees the uniqueness of the minimum. The particular case of (twice) differentiable deterministic functions is addressed by many studies (see for instance Boyd and Vandenberghe, 2004), and will not be considered in the following[1]

In this manuscript, though, the focus is on noisy (stochastic) fitness functions, related to a rugged fitness landscape. Optimization algorithms designed for smooth objectives do not apply to noisy objective functions.

Therefore, *derivative-free* techniques are better suited to noisy optimization. While those techniques originally aimed at non-differentiable functions (for instance subgradient methods, which are a generalization of gradient descent), they have also been used for noisy optimization (Beyer, 1998). Many derivative-free algorithms, including genetic algorithms, can be viewed as EDAs.

7.2 Estimation of Distribution Algorithms

Genetic algorithms (resp. evolution strategies) have been designed to handle ill-posed combinatorial problems (resp. continuous problems). Therefore, we

[1] For instance, quadratic convex optimization problems can be solved quite efficiently: there is a well-known analytic form for the solutions that can be approximated even for high dimensional domains. Generally, for finding local optima of twice-differentiable functions, second-order methods (e.g. Newton's method or quasi-Newton methods such as BFGS) are quite efficient. If there are linear constraints, first-order methods (interior point, Frank-Wolfe algorithm) may be used.

will first present evolutionary search algorithms before introducing estimation of distribution algorithms.

An evolution strategy (ES) is defined on $\mathbb{R}^d$. An instance of $\mathcal{X} \subset \mathbb{R}^d$ is referred to as an individual, the coordinates of which are called genes. An ES generates a *population* of solutions of size μ. From the population, λ new individuals, the offspring, are created via variation operators: the *mutation* operator modifies one or several genes of an individual, the *crossover* operator combines some genes of two or more individuals. Some new individuals may also be picked from the domain. From here on, the μ best individuals (ranked according to their fitness values) are selected either from the offspring of size λ (called (μ,λ)-ES) or from the offspring and the parent population of size $\mu + \lambda$ (called (μ+λ)-ES). The process is then repeated with the newly obtained μ-sized population.

In estimation of distribution algorithms, the variation operators are replaced by drawing individuals from a probability distribution. These individuals are evaluated, ranked, and the μ best ones are used to update the probability distribution (algorithm 5).

Note that EDAs generalize evolution strategies: probability distribution $P_{\mathcal{X}}^{(t)}$ that routine *evol* outputs can be any discrete generative model, and can therefore represent any variation operator.

Also note that such algorithms only involve the rank of instances, as opposed to the actual fitness values; such algorithms, invariant through any monotonous transformation of the objective function, are referred to as comparison-based.

7.3 State of the Art in Noisy Expensive Optimization

This section briefly summarizes the state of the art in noisy optimization and expensive optimization, with an emphasis on EDA analysis.

7.3.1 Expensive optimization

In expensive optimization settings, one can usually neglect the internal cost of the optimization algorithm, and can consider only the number of fitness evaluations. In particular, the computational cost of related optimization algorithm might be rather high, though negligible with respect to the fitness computation cost.

Efficient Global Optimization (EGO,Jones et al., 1998) and Informational Approach to Global Optimization (IAGO,Vazquez et al., 2008; Villemonteix et al., 2008) are examples of such algorithms. They assume a Gaussian prior on the fitness function; most specifically, they assume that the fitness function can be approximated accurately by a Gaussian Process. Although no theoretical proof of convergence towards the global optimum is provided, their empirical behavior is quite good in particular with respect to local optima. Their main limitation is a poor scalability with respect to the dimension of the search space.

Algorithm 5 Evolutionary search and estimation of distribution algorithms. $\mathcal{P}$(t) denotes the population (set of points of domain $\mathcal{X}$) at time t.

`(`μ`+`λ`)-ES`

Given Operator $mutation(\mathcal{S})$ generating $m \in [[0, \lambda]]$ instances from set S
Given Operator $crossover(\mathcal{S})$ generating $c \in [[0, \lambda]]$ instances from set S
  Initialize $\mathcal{P}(0)$ with μ points of domain $\mathcal{X}$
  Query fitness value for each element of $\mathcal{P}(0)$
  $t = 0$
  repeat
    $\mathcal{M}(t) = mutation(\mathcal{P}(t))$
    $\mathcal{C}(t) = crossover(\mathcal{P}(t))$
    Generate $\mathcal{R}(t)$ by picking randomly $\lambda - m - c$ points from $\mathcal{X}$
    for$(x \in \mathcal{M}(t) \bigcup \mathcal{C}(t) \bigcup \mathcal{R}(t))$ query $f(x)$
    $\mathcal{P}$(t+1) are the top-ranked μ instances of $\mathcal{M}(t) \bigcup \mathcal{C}(t) \bigcup \mathcal{R}(t) \bigcup \mathcal{P}(t)$
    t++
  until Computational budget is exhausted
  Return $\arg\max_{x \in P(t)} f(x)$

`(`μ`,`λ`)-EDA` // Requires $\lambda \geq \mu$

Given Initial probability distribution $P_{\mathcal{X}}^{(0)}$ on domain $\mathcal{X}$
Given Operator $evol(P, \mathcal{S})$ generating a probability distribution from distribution P and set $\mathcal{S}$
  Form $\mathcal{P}(0)$ by sampling μ points from $P_{\mathcal{X}}^{(0)}$
  repeat
    $P_{\mathcal{X}}^{(t+1)} = evol(P_{\mathcal{X}}^{(t)}, \mathcal{P}(t))$
    Draw λ points from $P_{\mathcal{X}}^{(t+1)}$ to form $\mathcal{E}(t)$
    for$(x \in \mathcal{E}(t))$ query $f(x)$
    $\mathcal{P}(t+1)$ are the top-ranked μ instances of $\mathcal{E}(t)$
    t++
  until Computational budget is exhausted
  Return $\arg\max_{x \in \mathcal{P}(t)} f(x)$

Another approach known as surrogate optimization is based on learning a (cheap) approximation of the objective function, and alternating the learning phase with the use of a standard optimization algorithm to extract the optimum of the surrogate (Booker et al., 1999). As an example, Zhou (2004) applies surrogate optimization in conjunction with evolution strategies.

7.3.2 Noisy optimization

Optimization in noisy environments deals with fitness functions for which multiple evaluations do not necessarily give the same result. Noise occurs in many real world applications (for instance, in many cases, it is due to finite precision of Monte-Carlo sampling) and often the noise is centered to zero: the goal is to

reach the optimal expected value.

Noisy expensive optimization is known to be hard computationally. In particular, handling it with evolution strategies directly may lead to a limit error, termed residual error (Arnold and Beyer, 2000). The formalism of EGO or IAGO is not adapted to noisy optimization either. Specific noisy optimization variants of direct search methods or algorithms with surrogate models have been designed (Conn et al., 1997), but these algorithms have several free parameters and often rely on the assumption that noise decreases to 0 near the optimum. Incidentally, noisy optimization is often the result of heavy Monte-Carlo simulations, and is therefore an expensive optimization problem either.

Noisy optimization is sometimes called "stochastic optimization" (Sengupta, 1972; Kall, 1976; Denton, 2003; Marti, 2005), although "stochastic optimization" rather refers to the optimization of deterministic fitness functions by stochastic algorithms. This is why the term "noisy optimization" is preferred here. Noisy optimization often distinguishes between cases in which the variance of the noise quickly decreases to zero around the optimum, and cases in which the variance of the noise is lower bounded. In the literature, various theoretical analyzes of complexity bounds address the former case. We are not aware of similar studies in the latter case.

The simplest solution for reducing noise consists in evaluating several times the same point: averaging decreases the variance. Unfortunately, depending on the detail of experimental setting, related works differ on whether such averaging is beneficial or adverse to optimization, particularly with respect to evolution strategies: an alternative to averaging in the case of evolution strategies or EDAs is to increase parameter λ (population size).

Various works (Fitzpatrick and Grefenstette, 1988; Hammel and Bäck, 1994; Arnold and Beyer, 2000) have investigated noisy optimization with EDAs from a theoretical point of view; a rigorous mathematical analysis is presented in Jebalia and Auger, 2008. Fitzpatrick and Grefenstette (1988) conclude in the case of genetic algorithms that averaging does not perform well: when considering the tradeoff between the computational overhead of averaging and the convergence rate, convergences are always slower with averaging. In the case of evolution strategies (see Beyer, 2001), Hammel and Bäck (1994) and Arnold and Beyer (2000) draw different conclusions: Hammel and Bäck (1994) conclude that strong averaging is required, whereas Arnold and Beyer (2000) conclude that increasing the population size is better than averaging. According to Arnold and Beyer (2000), results of Hammel and Bäck (1994) are due to a poor mutation strength adaptation schema; however, interestingly, they point out that for various noise models, each usual mutation-strength adaptation schema can lead to poor results: Arnold and Beyer (2000) compare mutative self-adaptation (Rechenberg, 1973; Schwefel, 1981)[2] and cumulative self-adaptation (Hansen and Ostermeier, 2003), and concludes that in both cases, there exist simple noise models adversely affecting any kind of mutation operator. Whether aver-

[2] two variants: one with arithmetic averaging of mutation strength and the other with geometric averaging

aging overcomes these difficulties is an open question.

An improvement of cumulative step-length adaptation is possible with increased population size, but only to a limited extent (Arnold and Beyer, 2008). An analysis using Markov chains has been adapted from the noise-free case (Bienvenue and Francois, 2003; Auger, 2005) to the noisy case (Jebalia and Auger, 2008; Teytaud and Auger, 2007); reevaluating individual fitnesses is suggested, but it is pointed out that in many cases, this would not be sufficient.

Heidrich-Meisner and Igel (2009) proposed the use of "bandits" (Lai and Robbins, 1985; Auer et al., 2002) for choosing the number of function evaluations spent on a given individual (more in section 7.4). This idea, albeit promising, could not be applied as such to the problem[3].

A particular case of noisy optimization setting is when the variance of the noise decreases to zero near the optimum. This has been tackled by Jebalia and Auger (2008), when the noisy fitness values $Y(x)$ involve a multiplicative noise:

$$Y(x) \sim f(x) \times (1 + N)$$

where f is the sphere function (i.e. $f : x \mapsto ||x||^2$) and N is an independent random variable. Assuming that N has a density in a bounded range $[m, M]$ with $-1 < m < M$,Jebalia and Auger (2008) shows that the scale-invariant $(1+1)$-ES converges. However, the proposed algorithm remains useless in practice[4] and its convergence rate could not be determined.

7.4 Hoeffding and Bernstein Bounds and their Application to Races

This section provides some tools related to concentration inequalities, which will be used in chapter 8 to study the convergence of a bandit-based optimization algorithm proposed to prove upper bounds on expensive optimization.

These inequalities aim at quantifying the discrepancy between the empirical average and the expectation of bounded random variables. The well-known Hoeffding bounds (Hoeffding, 1963) were the first ones to generalize bounds on binomial random variables to bounded random variables. Improved versions of these bounds known as Bernstein bounds, accounting for the variance of the random variable (Chernoff, 1952; Bernstein, 1924, 1946) and therefore tighter in some settings, will also be presented; we will focus on their application to races.

A *race* between two or more random variables aims at distinguishing with high confidence random variables with better expectation from those with worse expectation. The intuition behind races, as will be further detailed in chapter 9, is that finding an arm that is better than another one, irrespective of the

[3]The authors had to add several tricks in the implementation in order to get acceptable results, and did not provide theoretical nor experimental supporting evidence.

[4]After footnote 5 in Jebalia and Auger, 2008, it requires some information on each iteration about the distance to the optimum.

quality of all the others, is sufficient to prune the search space in a principled way.

Algorithm 6 Bernstein race between 3 arms. Equation 7.1 is derived from Bernstein's inequality for estimating the precision for empirical estimates (see e.g. Devroye et al., 1997, p124). With notations of section 2.3.1 on multi-armed bandits, $X_{a,i}$ is the ith random draw of arm a, $\hat{X}_{a,i}$ is the empirical mean and $\sqrt{\hat{V_{a,i}}}$ is the empirical standard deviation (evaluating to 0 in the first iteration).

$Bernstein(a_1, a_2, a_3, \eta)$
$T = 0$
repeat
  $T++$
  Pull each arm a_1, a_2, a_3 once, i.e. obtain the realizations of random variables $X_{a_1,T}$, $X_{a_2,T}$ and $X_{a_3,T}$, and update the empirical means
  Evaluate the precision:

$$\epsilon_T = 3\log\left(\frac{3\pi^2T^2}{6\eta}\right)/T + \max_i \sqrt{\hat{V}_{a_i,T}}\sqrt{2\log\left(\frac{3\pi^2T^2}{6\eta}\right)/T}. \tag{7.1}$$

until Two arms (*good* ,*bad*) satisfy $\hat{X}_{bad,T} - \hat{X}_{good,T} \geq 2\epsilon_T$
Return $(good, bad)$

Algorithm 6 presents a *Bernstein race* for 3 arms. For the Bernstein race in the following analysis, arms a_i will be points x_i of domain $\mathcal{X}$, and pulling an arm will be assessing the noisy fitness by sampling $Y(x_i)$. At the end of the race, $3T$ evaluations have been performed; T is accordingly referred to as the halting time in the sequel.

It is crucial in this situation to ensure that there exist i, j such that $\mu_{a_i} \neq \mu_{a_j}$ (which translates in $f(x_i) \neq f(x_j)$ for our purposes). If all $f(x_i)$ are equal, the algorithm may not terminate, and if it does its output will be meaningless. Intuitively, the closer $f(x_i)$ are, the larger T will be. A difference with the usual Bernstein framework is that the goal here is not to find i s.t. $\mu_i = \min\{\mu_1, \mu_2, \mu_3\}$. It is only to find i such that there is a j for which with high confidence, $\mu_i < \mu_j$.

More formally (section 2.3.1),

$$\begin{aligned}\Delta &= \max_{a_i} \Delta_{a_i} \\ &= \sup\{\mu_{a_1}, \mu_{a_2}, \mu_{a_3}\} - \inf\{\mu_{a_1}, \mu_{a_2}, \mu_{a_3}\}\end{aligned}$$

As established by (Mnih et al., 2008), if $\Delta > 0$ then

- with probability at least $1 - \eta$, the Bernstein race is consistent:

$$\mu_{good} < \mu_{bad}. \tag{7.2}$$

- the Bernstein race halts almost surely, and with probability at least $1-\eta$, the halting time T verifies

$$T \leq K \log\left(\frac{1}{\eta\Delta}\right)/\Delta^2 \quad \text{where } K \text{ is a universal constant.} \tag{7.3}$$

- if, in addition,

$$\Delta \geq C \sup\{\mu_{a_1}, \mu_{a_2}, \mu_{a_3}\}, \tag{7.4}$$

  then the Bernstein race halts almost surely, and with probability at least $1-\eta$, the halting time T verifies

$$T \leq K' \log\left(\frac{1}{\eta\Delta}\right)/\Delta \quad \text{where } K' \text{ depends on } C \text{ only.} \tag{7.5}$$

Chapter 8

Lower Bounds for Noisy Expensive Optimization

This chapter focuses on the minimum runtime of any noisy expensive optimization algorithm on a given class of fitness function. The presented contribution is a lower bound on the number of requests needed to yield the optimum with precision ϵ and confidence 1-δ.

Some upper bounds will be presented in the next chapter. The work presented in both chapters has been published in Rolet and Teytaud (2010c,b); Coulom et al. (2011).

8.1 Framework

The optimization framework is described in algorithm 7. The algorithm can request the fitness value at any point of the domain; the cost of the optimization algorithm is measured by its sample complexity. No other information on the fitness function is available, as in any black-box optimization setting. For simplicity, it is assumed that optimization algorithms are deterministic. Consistently with previous notations, an expensive optimization algorithm is written $\mathbf{L}$ and consists of a pair $(\mathbf{A}, \mathbf{S})$. Sampler $\mathbf{S}$ takes as input a sequence of visited points and their measured fitness values, and outputs a new point to be visited. Optimizer $\mathbf{A}$ suggests an optimum given a sequence of visited points. Without loss of generality, it will be assumed that the optimum sought for is a minimum, and that every fitness function considered admits a minimum.

Consider a class of function $\mathcal{F}$ whose elements, fitness functions $f : \mathcal{X} \to \mathbb{R}$, can be parametrized solely by the (unknown) location of their minimum, x^*—i.e. the mapping $f \mapsto \min_{x \in \mathcal{X}} f$ is injective. The goal is to find the minimum x^* of f, by observing noisy measurement of f at x_i.

Since elements of $\mathcal{F}$ can be indexed by their minimum, $\mathcal{F}$ will be characterized by the mapping $\mathbf{f} : \mathcal{X} \times \mathcal{X} \to \mathcal{Y}$, such that the function f of $\mathcal{F}$ whose minimum is x^* is $x \mapsto \mathbf{f}(x, x^*)$.

Following notations of chapter 2 (section 2.1.3), noisy measurements of the fitness at point x are modeled by a random variable $Y(x)$. By construction, $f(x) = \mathbb{E}[Y(x)]$. The probability density function of $Y(x)$ is noted $p(.|x)$. Two remarks should be made on this noise model:

- since few assumptions on the probability density function are made, one cannot consider that the probability mass of $Y(x)$ is centered on $\mathbf{f}(x, x^*)$ (as opposed to the case of a Gaussian noise model);
- the noisy setting does not assume $Y(x) > Y(x^*)$, contrary to some previous analyzes (Jebalia and Auger, 2008) .

For the sake of simplicity, we assume in the following that $Y(x)$ ranges in [0,1]. The proposed contributions (the lower bound and its proof) are however invariant under multiplication or addition of a constant.

The following analysis requires to consider all possible runs of an optimization algorithm. For the purpose of the analysis, it is convenient to model the whole noise as a single random variable Θ which is a sequence of independent random variables $(\Theta)_i$ of uniform law on [0,1]—thus, a realization $\theta = ((\theta)_1, (\theta)_2, \ldots)$ of Θ is an element of $[0,1]^{\mathbb{N}}$—θ can be thought of as a random seed.

During a run of the optimization algorithm, the randomness of a call to fitness $\mathbf{f}(., x^*)$ at step n is then accounted for by $(\theta)_n$: the outcome of the nth fitness call is written $y_n^{(\theta,x^*)}$. Since the points that the algorithm selects at step n for evaluation depend on these outcomes, those also depend on θ and are noted $x_n^{(\theta,x^*)}$.

Note that the use of parameter θ modeling the stochasticity of the process does not imply any assumptions on the noise model. The noise is defined beforehand via random variable $Y(x)$ of probability density $p(.|x)$. It is possible to choose a mapping $\Phi : (x, \tau) \mapsto y$ such that random variable $\Phi(x, \tau)$ has the same law as random variable $Y(x)$. Formally, choosing

$$\begin{aligned}\Phi :&\mathcal{X} \times [0,1] \to [0,1] \\ &(x,t) \mapsto \min\left\{ y_0 | \int_0^{y_0} p(y|x)dy \geq t \right\}\end{aligned}$$

ensures this property. A noisy measurement $y_n^{(\theta,x^*)}$ is given by $\Phi(x_n^{(\theta,x^*)}, (\theta)_n)$.

In the following, minimum x^* is not handled stochastically, i.e. the lower bounds are not computed in expectation w.r.t. all the possible fitness functions of $\mathcal{F}$ yielded by different values of x^*. Rather, the worst case on x^* is considered. Therefore the only random variable in this framework is θ, which governs the trajectory of the search process[1] (algorithm 7). Therefore, the return value of $\mathbf{L}$ is completely determined by number of requests N, random seed θ and fitness function f indexed by its minimum x^*.

[1] Although only deterministic algorithms are considered, the extension to stochastic algorithms is possible by including a random seed for the algorithm in θ.

Given domain $\mathcal{X}$ and mapping $\mathcal{F}$, the distance

$$\rho : (x^{*1}, x^{*2}) \mapsto \sup_{x \in \mathcal{X}} |\mathbf{f}(x, x^{*1}) - \mathbf{f}(x, x^{*2})|$$

for x^{*1} and x^{*2} in $\mathcal{X}$ will be used to measure the error of $\mathbf{L}$. Basically, assuming x^{*1} is the target minimum, this distance is the largest gap in fitness that exist when mistaking minimum x^{*1} with x^{*2}.

Algorithm 7 Black-box noisy optimization framework (optimization algorithm $\mathbf{L}$). s_n is the training set at step n . The performance of $\mathbf{L}$ is measured by $\rho(x^*, x_{N+1}^{(\theta,x^*)})$(details in the text).

Given θ, random seed of $\in [0,1]^N$ whose coordinates $(\theta)_i$ for $i \in \mathbb{N}$ are uniformly distributed in $[0,1]$.
Given x^*, unknown element of $\mathcal{X}$ characterizing the fitness f
  Parameter N, number of fitness evaluations
  $s_0 = \emptyset$, $n = 0$
  while n<N **do**
    $x_{n+1}^{(\theta,x^*)} = \mathbf{S}(s_n)$
    $y_{n+1}^{(\theta,x^*)} = \Phi(x_{n+1}^{(\theta,x^*)}, (\theta)_{n+1})$
    $s_{n+1} = \mathcal{S}_n \bigcup \{(x_{n+1}^{(\theta,x^*)}, y_{n+1}^{(\theta,x^*)})\})$
    $n++$
  end while
  Return $x_{N+1}^{(\theta,x^*)} = \mathbf{A}(s_N) = \mathbf{L}(N, \theta, x^*)$

Remark 5. It is important to notice that in many cases $\rho(\hat{x}, x^*) = \Theta(\|\hat{x} - x^*\|)$ in the neighborhood of x^*, which ensures that the asymptotic convergence rate is the same (up to a constant), whether the distance is $\rho(\hat{x}, x^*)$ or $\|x - x^*\|$. Typically, given a monotonically increasing function g, for $f : x \mapsto g(\|x - x^*\|)$, it is sufficient that g be Lipschitz continuous to derive $\rho(\hat{x}, x^*) = \Theta(\|\hat{x} - x^*\|)$.

8.2 Lower Bound

This section states and prove a theorem allowing derivations of lower bounds in sample complexity for noisy expensive optimization. In this section $B(n, p)$ is a binomial random variable (sum of n independent Bernoulli variables of parameter p).

8.2.1 Theorem statement

Let us consider a Bernoulli noise, for which $Y(x)$ is 1 with probability $1 - f(x)$ and 0 otherwise. It follows that density p is given as

$$p(y|x) = f(x)\delta_0 + (1 - f(x))\delta_1$$

where δ_x stands for the Dirac measure in x. Mapping $\Phi(x, \tau)$ boils down to returning 1 if $f(x) > \tau$ and 0 otherwise. This model fits applications based on highly noisy optimization problems[2].

By construction, a lower bound that holds in this restricted noise setting is valid in the more general case in which no assumption is made on the noise. Besides, as mentioned above, this noise model is adverse, providing a pessimistic estimate of the lower bound, which suggests the generality of this bound.

The proposed result relies on two assumptions, respectively related to the topology of space $\mathcal{X}$, and the convergence of an optimization algorithm $\mathbf{L}$.

The first assumption, referred to as $\mathbf{H}_{dim}(\epsilon_0, D)$, is related to the existence of D points in $\mathcal{X}$ with pairwise distance ϵ for any ϵ lower than some ϵ_0:

$$\begin{aligned}\mathbf{H}_{dim}(\epsilon_0, D): \quad & \forall \epsilon_1 < \epsilon_0 \exists (x^{*1}, \ldots, x^{*D}) \in \mathcal{X}^D, \forall (i,j) \in [[1, D]]^2, \\ & i \neq j \Rightarrow \rho(x^{*i}, x^{*j}) = \epsilon_1.\end{aligned}$$

Note that this assumption makes sense only for $D \geq 2$. It is closely related to the dimension of $\mathcal{X}$. For instance, in many cases (some of them will be presented at the end of this chapter), $\rho(x, x') = \Theta(\|x - x'\|)$, and the maximum number of such equidistant points is $d + 1$ (where $\mathcal{X} \subset \mathbb{R}^d$).

The second assumption, referred to as $\mathbf{H}_{PAC}(\epsilon, N, \delta)$ states that there exists an $\mathbf{L}$ reaching precision ϵ with probability $1 - \delta$ after N visited points:

$$\mathbf{H}_{PAC}(\epsilon, N, \delta): \quad \exists \mathbf{L} \; \forall x^* \in \mathcal{X}, P(\rho(\mathbf{L}(N, \theta, x^*), x^*) < \epsilon) \geq 1 - \delta$$

where P is the probability with respect to θ.

The lower-bound theorem can then be stated as follows:

Theorem 5. *Let ϵ_0 and $D \geq 2$ be such that $\mathcal{X}$ satisfies $\boldsymbol{H}_{dim}(\epsilon_0, D)$. Let precision ϵ be in $]0, \epsilon_0]$, and confidence δ be in $]0, 1/(2D)[$. If a number of requests N is such that $\boldsymbol{H}_{PAC}(\epsilon/2, N, \delta)$ holds for a given algorithm $\boldsymbol{L}$, then N satisfies*

$$P(B(N, \epsilon) \geq \lceil \log(D) \rceil) \geq 1 - \delta D. \tag{8.1}$$

Equation 8.1 is a lower bound on the number of iterations N such that precision ϵ is reached since it implies

$$N = \Omega(\log(D)/\epsilon) \tag{8.2}$$

for a fixed D.

8.2.2 Proof

The proof of theorem 5 will be presented, followed by the proof of equation 8.2.

[2]Games are an example of such problems: let x be a parameter of a game strategy, to be set at its optimal value; one noisy observation is a game against a baseline, resulting either in a win or in a loss; the aim is to find the value of x maximizing the probability of winning.

Useful definitions

Let us now introduce some notations and definitions relevant to the proof.

First, consider $\epsilon_0, D, \epsilon, \delta$ as in the theorem statement, assuming that $\mathbf{H}_{dim}(\epsilon_0, D)$ holds. Let N be a number of requests such that there exists an algorithm for which $\mathbf{H}_{PAC}(\epsilon/2, N, \delta)$ holds, and let $\mathbf{L}$ be this algorithm.

Thanks to assumption $\mathbf{H}_{dim}(\epsilon_0, D)$, there exists D points $x^{*1}, \ldots, x^{*D}$ satisfying

$$\forall i \neq j, \rho(x^{*i}, x^{*j}) = \epsilon. \tag{8.3}$$

These points will correspond to "possible minimums" that $\mathbf{L}$ cannot distinguish if N is not large enough, because the probability that they provide the same answer when querying the noisy fitness on a point x is very high.

Second, given an index $i \in [[1, D]]$, let us define

$$C_i^n(\theta) = \{j \in [[1, D]]; (y_1^{(\theta, x^{*j})}, \ldots, y_n^{(\theta, x^{*j})}) = (y_1^{(\theta, x^{*i})}, \ldots, y_n^{(\theta, x^{*i})})\}.$$

$C_i^n(\theta)$ corresponds to the set of points in $x^{*1}, \ldots, x^{*D}$ such that all answers given by the oracle are the same as on x^{*i}.

Next, define $k_n(\theta)$ as

$$k_n(\theta) = \min\{i \in [[1, D]]; |C_i^n(\theta)| \text{ is maximal}\}.$$

The important point here is that $k_n(\theta)$ is an index i such that $C_i^n(\theta)$ is of maximal cardinal; the min is only here to choose among the possible solutions. In other words, $k_n(\theta)$ is the index of some x^{*i} such that "many" (as many as possible) x^{*j} raise the same labels $y_1^{(\theta, x^{*j})}, \ldots, y_n^{(\theta, x^{*j})}$.

Then, let us define:

$$s_{min,n}(\theta) = \inf_{i \in C_{k_n(\theta)}(\theta)} \mathbf{f}(x_n^{(\theta, x^{*i})}, x^{*i}),$$

$$s_{max,n}(\theta) = \sup_{i \in C_{k_n(\theta)}(\theta)} \mathbf{f}(x_n^{(\theta, x^{*i})}, x^{*i}).$$

Interval $[s_{min,n}(\theta), s_{max,n}(\theta)]$ is "small", thanks to assumption $\mathbf{H}_{dim}(\epsilon_0, D)$:

$$\forall (n, \theta), s_{min,n}(\theta) \geq s_{max,n}(\theta) - \epsilon. \tag{8.4}$$

Lastly, define the property $Error(x^*, \theta) : \rho(\mathbf{L}(N, \theta, x^*), x^*) \geq \epsilon/2$. Define also a "good" set $\mathcal{G}$ by

$$\theta \in \mathcal{G} \text{ iff } \forall i, |C_i^N(\theta)| = 1. \tag{8.5}$$

Then,

$$\begin{aligned} \theta \notin \mathcal{G} &\Rightarrow \exists i \neq j, x_{N+1}^{(\theta, x^{*i})} = x_{N+1}^{(\theta, x^{*j})} \\ &\Rightarrow Error(x^{*i}, \theta) \text{ or } Error(x^{*j}, \theta). \end{aligned} \tag{8.6}$$

thanks to equation 8.3.

Proof of equation 8.1

From those definitions, we can turn to the proof of equation 8.1.

First, let us prove some useful lemmas.

Lemma 6. *With the above notations,*

$$P(\forall i, |C_i^N(\theta)| = 1) \geq 1 - D\delta. \tag{8.7}$$

Proof. First let us show the following equation:

$$\mathbf{H}_{PAC}(\epsilon/2, N, \delta) \Rightarrow P(\mathcal{G}) \geq 1 - D\delta. \tag{8.8}$$

Assume $\mathbf{H}_{PAC}(\epsilon/2, N, \delta)$. If $\theta \notin \mathcal{G}$, then from equation 8.6, there exists i such that $Error(x^{*i}, \theta)$. Therefore,

$$\begin{aligned} P(\neg\mathcal{G}) &\leq \sum_{i \in [[1,D]]} P(Error(x^{*i}, \theta)) \\ &\leq D\delta \text{ by } \mathbf{H}_{PAC}(\epsilon/2, N, \delta). \end{aligned}$$

which proves equation 8.8. Equation 8.8 and the definition of $\mathcal{G}$ (equation 8.5), under assumption $\mathbf{H}_{PAC}(\epsilon/2, N, \delta)$, lead to the result. □

From lemma 6, if $\mathbf{H}_{PAC}$ holds, then $C_i^N(\theta)$ must be small for all i and with high probability on θ.

In order to complete the proof, it will now be shown that $C_i^N(\theta)$ cannot be small unless N is large. More precisely, let us show for all N the following lemma:

Lemma 7. *With the above notations,*

$$P(|C^N_{k_N(\theta)}(\theta)| \leq 1) = P(\forall i, |C_i^N(\theta)| \leq 1) \leq P(B(N, \epsilon) \geq \lceil \log(D) \rceil). \tag{8.9}$$

Proof. First, with $k_n = k_n(\theta)$ for short, notice that the size of the largest set of x^{*i} that have the same outcomes at $n = 0$ is $|C^0_{k_0}(\theta)| = D$.

Second, from step $n-1$ to step n, let us analyze how the size of this largest set evolves. Given $i \in k_{n-1}$, from the definition of θ and $y_n^{(\theta,x^{*i})}$, we have $y_n^{(\theta,x^{*i})} = 1$ iff $(\theta)_n \geq \mathbf{f}(x_n^{(\theta,x^{*i})}, x^{*i})$, and $(\theta)_n$ is a random variable chosen uniformly in $[0,1]$. Therefore, for any $i, j \in k_{n-1}$, having $y_n^{(\theta,x^{*i})} \neq y_n^{(\theta,x^{*j})}$ requires $(\theta)_n \in [s_{min,n}(\theta), s_{max,n}(\theta)]$. Then,

- $|C^n_{k_n}|$ is not reduced with probability $P(|C^n_{k_n}(\theta)| = |C^{n-1}_{k_{n-1}}(\theta)|) \geq 1 - \epsilon$ (by equation 8.4);
- $|C^n_{k_n}|$ is reduced by at least 1 (and halved at most) with probability $1 - P(|C^n_{k_n}(\theta)| = |C^{n-1}_{k_{n-1}}(\theta)|) \leq \epsilon$.

Even in the most optimistic situation, in which $|C^n_{k_n}|$ is halved each time it diminishes, it is required that at least $\lceil \log D \rceil$ reductions occur during the N iterations to ensure the cardinal of $|C^n_{k_n}|$ reaches 1. Since the probability that a single reduction occur is upper-bounded by ϵ, the probability that $\lceil \log D \rceil$ reductions occur during N iterations is bounded by a binomial law of parameters (N, ϵ):

$$P(|C^N_{k_N(\theta)}(\theta)| \leq 1) \leq P(B(N, \epsilon) \geq \lceil \log(D) \rceil).$$

This yields lemma 7. □

Lemmas 6 and 7 together conclude the proof of equation 8.1, which concludes the proof of theorem 5.□

Proof of equation 8.2

Equation 8.2 is derived from a one-sided variant of Chebychev's inequality known as Cantelli's inequality. The expectation of random variable $X = B(N, \epsilon)$ is $N\epsilon$, and its variance is $N\epsilon(1-\epsilon)$. Cantelli's inequality states that for any $k > 0$,

$$P(X - \mathbb{E}[X] \geq k\sqrt{Var(X)}) \leq \frac{1}{1+k^2}.$$

If $k = 1$ then $1/(1+k^2) \leq 1 - \delta D$, since $\delta D \leq 1/2$, and

$$P(X \geq N\epsilon + k\sqrt{N\epsilon(1-\epsilon)}) \leq 1 - \delta D,$$

which combined with equation 8.1 gives

$$\begin{aligned} \lceil \log(D) \rceil &\leq N\epsilon + k\sqrt{N\epsilon(1-\epsilon)} \\ &\leq N\epsilon + \sqrt{N\epsilon}. \end{aligned}$$

Since $\log D \geq 1$, it is necessary that $N\epsilon \geq 1/4$; in this case, $\sqrt{N\epsilon} \leq 2 * N\epsilon$ which yields

$$\log D \leq 3N\epsilon.$$

□

8.3 Discussion

The interpretation of this lower bound result is discussed with respect to a few particular models:

Sphere function $\mathbf{f}(x, x^*) = ||x - x^*||$, a special case in which the variance promptly decreases to 0 around the minimum;

Scaled sphere function $\mathbf{f}(x, x^*) = \lambda||x - x^*||$ for some $\lambda > 0$, a case that might be handled similarly to the one above;

Scaled and translated sphere function (noted S-T sphere from here on). $\mathbf{f}(x, x^*) = \lambda||x - x^*|| + c$, for some $c \in]0, 1[$, fundamentally harder since the variances does *not* decrease to 0 around the minimum;

Polynomial S-T sphere $\mathbf{f}(x, x^*) = \lambda||x - x^*||^p + c$ with $p \in [[2, \infty[[$; in particular, the quadratic case, $p = 2$, is often encountered in practice;

Transformed sphere $\mathbf{f}(x, x^*) = g(||x - x^*||)$ for some increasing mapping g from $[0, \infty[$ onto a subset of $[0, 1]$.

The tightness of the bound for these models will be discussed in the next chapter. Let us now turn to the application of theorem 5 to each of these models.

Sphere model

The simplest interpretation is for the sphere model, with $\mathcal{X} = [0, 1]^d$, and $\mathbf{f}(x, x^*) = ||x - x^*||$ for some norm $\|.\|$. In this case, theorem 5 applies with $D = d + 1$, and distance ρ boils down to the considered norm (i.e. $\rho(x, x^*) = \|x - x^*\|$): reaching a precision $\epsilon = ||x_{N+1}^{(\theta,x^*)} - x^*||$ with confidence $1 - \delta$ for any $\delta < 1/(2D)$ will require at least $\Omega(\log(D)/\epsilon)$ points.

Scaled sphere

The above result can straightforwardly be applied to this setting: a distance ϵ between the minimum and approximation requires at least $\Omega(\log(D)/\epsilon)$ iterations (for any algorithm).

S-T sphere

The fitness at the minimum needs not have a variance of 0 for the above theorem to apply—consider for instance[3] $f(x, t) = \min(1, c + ||x - t||)$. In this case, the bound $N = \Omega(\log(D)/\epsilon)$ is still valid. Let us however present some intuitive arguments pointing to the fact that in the case of a strictly positive variance at the minimum, better lower bounds can be obtained, with a convergence rate in $\Omega(1/\epsilon^2)$.

With the S-T sphere function, the variance can be lower bounded by some positive constant c: $\inf_{x \in \mathcal{X}} Var[Y(x)] > c > 0$. Therefore, evaluating a point n times leads to a confidence interval on its mean whose length is roughly $\sqrt{c/n}$. Hence, the precision of a fitness estimate based on n evaluations is $\Theta(1/\sqrt{n})$. As the precision in the fitness space linearly depends on the precision in the search space, it seems realistic to say that $\|x_n^+ - x_n^-\| = \Theta(1/\sqrt{n})$ is the best achievable rate. Chapters 9 and 10 provide theoretical and empirical evidence showing that this rate can be reached; we are not aware of any theoretical or empirical evidence showing that it can be exceeded.

[3]Note that in our problem setting, the function $g : x \mapsto c + ||x - t||$ should in all generality range in $[0, 1]$, which is not the case here. We can either set the measurement to 0 when $g(x, t) > 1$, or consider $f(x, t) = \min(1, g(x, t))$ instead, which is what will be done from now on.

Quadratic and polynomial S-T sphere

An unexpected result in the polynomial S-T sphere ($f(x,t) = \min(1, ||x-t||^p + c)$ for any $p \in [[2, \infty]]$ and $c \in [0, 1]$) is that the *rate* in the lower bound does not depend on p: it is similar to the case $p = 1$ (although a linear dependence in p exist in constant factors). Indeed, for ϵ_0 sufficiently small, $H(\epsilon_0, D)$ holds for $D = d + 1$ and applying the theorem yields $N = \Omega(\log(D)/\epsilon)$—or equivalently, for any fixed p, the distance between the N^{th} iterate and the minimum that can be guaranteed with confidence $1 - \delta$ for $\delta < 1/(2D)$, is $\Theta(1/N)$ (although there might.

Since in this case $\rho(x, x^*) = \Theta(\|x - x^*\|)$, a somewhat surprising conclusion can be drawn: the number of iterations is lower bounded by $\Theta(log(D)/\epsilon)$ for any exponent p used in the family of fitness functions. Of course, the constant hidden in the Landau symbol Θ might change, but the rate remains ϵ^{-1}.

Monotonically transformed sphere

In the general case of an arbitrary monotonically increasing function g, the problem can be made arbitrarily difficult: the lower bound can be made arbitrarily high by choosing hard function classes. Thus, the only guarantee that any algorithm can provide in this setting is its convergence.

Chapter 9

Upper Bounds for Noisy Expensive Optimization

This chapter, also focusing on noisy optimization, establishes new upper bounds on the number of fitness queries needed to reach the optimum. The contribution lies in the upper bounds, and in the supporting bandit-based algorithm called $R-EDA$.

9.1 Position of the work

As mentioned in section 7.3, Jebalia and Auger (2008) provide an upper bound on the sphere function, based on runtime analysis of the scale-invariant $(1+1)-ES$. The presented work differs from theirs in three respects:

- the noise model is broader, and includes for instance cases in which the noisy measurement can be arbitrarily small with respect to the expected fitness, i.e. $Y(x)/f(x)$ is not lower bounded;
- the algorithm is *explicit*: it does not require information on the position of the optimum, or on the distance towards the optimum;
- the convergence rate is tight for any fixed dimension on the sphere model.

Further, in the general case, while Arnold and Beyer (2000) emphasized that for many algorithms a residual error remains, $R-EDA$ is truly consistent (i.e. $\|x_n - x^*\| \to_\infty 0$) as shown in section 9.3 below.

Finally, the proposed upper bounds are tight for the scaled sphere and quadratic sphere model ($p=1$ or $p=2$) when $c=0$: the algorithm is optimal asymptotically up to logarithmic terms and constant factors. The intuitive arguments presented on the last chapter for the lower bound in the case $(p=1, c>0)$ suggest that the algorithm is optimal in that case too. Hence, a merit of $R-EDA$ is that it adapts to all considered fitness functions, irrespective of the different lower bounds (chapter 8).

Section 9.2 will describe the $R-EDA$ algorithm. Section 9.3 proves convergence rates for the different variations of the sphere model introduced in the previous chapter.

9.2 An Adaptive Noisy Optimization Algorithm: $R-EDA$

The algorithm presented in this section is called *Race*-EDA ($R-EDA$) is based on Bernstein races and inspired by estimation of distribution algorithms (EDAs). $R-EDA$ is a (3,3) evolution strategy: the parent population consists of 3 points, and 3 points are generated from this population and act as the new population. The difference with respect to "standard" EDAs is as follows:

- the algorithm is derandomized: population t is generated deterministically from population $t-1$;
- since $\mu = \lambda$, there is no need for actually ranking all the points (the algorithm still orders two points among the three as will be seen below).

Note that these points do not prevent the algorithm from fitting the EDA definition of chapter 7 (algorithm 5). The algorithm is comparison-based (since fitness values only matter by how they order the population), which is the major trait of EDAs.

Algorithm 8 $R-EDA$: Algorithm for optimizing noisy fitness functions. *Bernstein* denotes a Bernstein race, as defined in algorithm 6. The initial domain is $[x_0^-, x_0^+] \in \mathbb{R}^d$.

Parameter δ, the confidence parameter.
$n = 0$
repeat
  /* Pick the coordinate with highest uncertainty */
  $c = \arg\max_{i \in [[1,d]]} [(x_n^+)_i - (x_n^-)_i]$
  for $i \in [[1,3]]$ **do**
    $x_n^{(i)} \leftarrow \frac{1}{2}(x_n^- + x_n^+)$. // Consider the middle point
    $(x_n^{(i)})_c \leftarrow (x_n^-)_c + \frac{i-1}{2}(x_n^+ - x_n^-)_c$. //except that the c^{th} coordinate may // take 3 different values
  end for
  /* Find a good and a bad point */
  $(good_n, bad_n) = Bernstein(x_n^{(1)}, x_n^{(2)}, x_n^{(3)}, \frac{6\delta}{\pi^2(n+1)^2})$
  Let $\mathcal{H}_n$ be the halfspace $\{x \in \mathbb{R}^d; ||x - good_n|| \leq ||x - bad_n||\}$.
  Split the domain: $[x_{n+1}^-, x_{n+1}^+] = \mathcal{H}_n \cap [x_n^-, x_n^+]$.
  $n++$
until Computational budget exhausted
Return $(x_n^+ + x_n^-)/2$ // middle of the remaining domain

Notations of section 8.1 will be used, except that superscripts (θ, x^*) on points x will be dropped for the sake of clarity. Point $x_n^{(i)}$ stands for the ith point of the n^{th}-generation population, and x_n^+, x_n^- define the boundary of the remaining domain at step n. The optimization domain is $\mathcal{X} = [x_0^-, x_0^+] = [(x_0^-)_1, (x_0^+)_1] \times [(x_0^-)_2, (x_0^+)_2] \times \cdots \times [(x_0^-)_d, (x_0^+)_d]$.

Referring to routine *Bernstein* (algorithm 6, chapter 7), the $x_n^{(i)}$ are seen as arms whose distribution are $p(.|x_n^{(i)})$ centered on $f(x_n^{(i)})$ (associated to random variables $Y(x_n^{(i)})$).

Points x_0^- and x_0^+ are elements of $\mathbb{R}^d$ and therefore $\mathcal{X} \subset \mathbb{R}^d$. It is assumed that $||x_0^- - x_0^+|| \leq 1$.

Overview of $R-EDA$ $R-EDA$ (algorithm 8) proceeds by iteratively splitting the domain in two subspaces, and retaining the one that most probably contains the minimum. At iteration n, from the n_{th} domain $[x_n^-, x_n^+]$, the $(n+1)_{th}$ domain $[x_{n+1}^-, x_{n+1}^+]$ is obtained by:

- finding the coordinate c such that the gap $(x_n^+)_c - (x_n^-)_c$ is maximal;
- selecting 3 regularly spaced points along this coordinate;
- repeatingly assessing those 3 points until gaining some confidence that one point $x_n^{(i)}$ is closer to x^* than another $x_n^{(j)}$ (noting that $x_n^{(i)}$ is not necessarily the best one, nor is $x_n^{(j)}$ necessarily the worst one), which is achieved with confidence $1-\eta$ by the Berstein race (algorithm 6, chapter 7)
- splitting the domain by the hyperplane in the middle of these points and normal to the line they define, and keeping only the side of the domain containing $x_n^{(i)}$.

Visual descriptions of the algorithm are provided by figure 9.1 and 9.2.

Related works. The "domain reduction" step relies on a good point $(x_n^{(i)})$ and a bad point $(x_n^{(j)})$. While the use of good and bad points to narrow down the search space has been used in Arnold and Wart (2008), and in the optimization heuristic Breda by Gelly et al. (2007), $R-EDA$ applies this idea to address the noisy optimization case.

9.3 Convergence and Runtime Analysis

This section will be devoted to the analysis of the runtime of $R-EDA$. The analysis will focus on the *transformed sphere* models[1] introduced in chapter 8. In the following, fitness f is a strictly increasing function of $\|x - x^*\|$ and noisy measurements of f are given via random variables $Y(x)$ of density $p(.|x)$.

The results are concerned with:

[1]The transformed sphere covers in particular models of the scaled sphere, S-T sphere and polynomial S-T sphere.

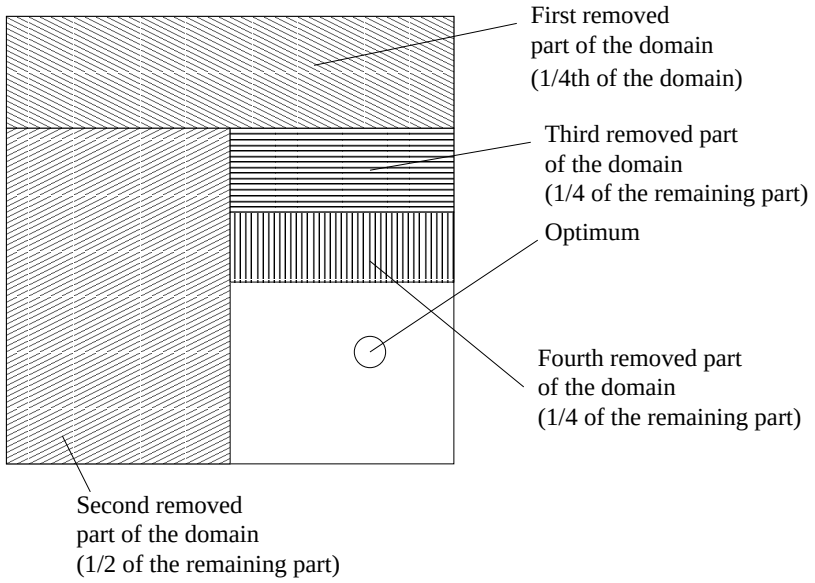


Figure 9.1: Sequence of domain reductions achieved by $R-EDA$ (algorithm 8). At each iteration, the axis with maximal span is selected). Three equally spaced points located on this axis are generated (the offspring). The Bernstein race is used to determine a "good" point and a "bad" point, and the region closest to the bad point is removed, pruning one fourth or one half of the domain (depending on the position of the good arm and of the bad arm—the best case is when the good and the bad arm are diametrically opposed: see figure 9.2).

- the convergence of $R-EDA$: it finds the minimum with precision ϵ and confidence $1-\delta$;
- the convergence rate: for specific sphere models, the number of fitness calls can be upper-bounded by $\tilde{O}(1/\epsilon^{2p})$ and even sometimes $\tilde{O}(1/\epsilon^{p})$.

9.3.1 General case (monotonic transformation of the sphere function)

Proof sketch. The algorithm geometrically shrinks the domain, hence the domain size quickly decreases to ϵ. Based on the assumption that fitnesses are monotonically increasing transformations of the sphere function, the minimum is closer to the "good" arm than to the "bad arm" with high probability: hence the minimum stays within the shrunk domain with high probability. The number of requests to the fitness functions will then be upper bounded by classical Bernstein bounds for variations of the sphere model. This is only possible thanks to the specific mutation operator in algorithm 8: it ensures that arms are all "sufficiently different".

Let us turn to formal proofs, relying on various lemmas.

First, note that assuming that noisy measurements Y(x) belong to [0,1], we have

$$Var\ Y(x) \leq \mathbb{E}[Y(x)^2] \leq \mathbb{E}[Y(x)] = f(x). \tag{9.1}$$

Lemma 8. *Let $\delta > 0$. If the Bernstein race succeeds (i.e. terminates) for steps 1 to n in $R-EDA$, then*

$$\left(\frac{3}{4}\right)^n ||x_0^+ - x_0^-|| \leq ||x_n^+ - x_n^-|| \leq \left(\frac{3}{4}\right)^{\lfloor n/d \rfloor} ||x_0^+ - x_0^-|| \text{ and} \tag{9.2}$$

$$(\forall i < n, f(good_i) \leq f(bad_i)) \Rightarrow x^* \in [x_n^-, x_n^+], \tag{9.3}$$

Proof. At each iteration, at least one fourth of the hyper-rectangle is removed in the direction in which the hyper-rectangle is the longest. This ensures equation 9.2.

$f(good_n) \leq f(bad_n)$ implies that x^* is closer to $good_n$ than to bad_n; therefore $x^* \in \mathcal{H}_n$. This implies equation 9.3. □

The guarantee on $R-EDA$'s convergence follows:

Theorem 9 (Consistency of $R-EDA$ for the transformed sphere)**.** *In the transformed sphere model, algorithm 8 ensures $x_n^- \to x^*$ and $x_n^+ \to x^*$ with probability at least $1-\delta$.*

Proof. From lemma 8 (equation 9.3), $\|x_n^+ - x_n^-\| \to 0$. Furthermore, the Bernstein race halts almost surely, because the fact that f is a *strictly* increasing transformation of the sphere guarantees that there are two arms with a different mean (section 7.4).

We will now show that with probability $1-\delta$, $x^* \in [x_n^-, x_n^+]$ by establishing the left-hand side of equation 9.3 by induction. This will be sufficient to prove theorem 9.

- The induction hypothesis $\mathbf{H}(n)$ is:

$$\text{With probability at least } 1 - \sum_{k=1}^{n+1} \frac{6\delta}{\pi^2 k^2},$$
$$\forall i < n, f(good_i) = \mathbb{E}Y(good_i) \leq \mathbb{E}Y(bad_i) = f(bad_i).$$

- $\mathbf{H}(0)$ clearly holds because the statement $\forall i < 0$, $\mathbb{E}Y(good_i) \leq \mathbb{E}Y(bad_i)$ is always true, since no i satisfies $i < 0$.

- Let us assume $\mathbf{H}(n-1)$ for $n > 0$. For clarity, the statement $\forall i <$

$n, f(good_i) \leq f(bad_i)$ is written $G(n)$.

$$\begin{aligned} P(G(n)) &= P(G(n-1), f(good_n) \leq f(bad_n)) \\ &= P(f(good_n) \leq f(bad_n) \mid G(n-1))P(G(n-1)) \\ &= (1 - \sum_{k=1}^{n} \frac{6\delta}{\pi^2 k^2})(1 - \frac{6\delta}{\pi^2 (n+1)^2}) \qquad (9.4) \\ &\geq 1 - \sum_{k=1}^{n+1} \frac{6\delta}{\pi^2 k^2} \end{aligned}$$

which proves $\mathbf{H}(n)$. The first term of equation 9.4 is the application of $\mathbf{H}(n-1)$. The second term is a property of the Bernstein race described in algorithm 6, and used in algorithm 8.

It only remains to observe that $\sum_{i=1}^{\infty}(6\delta/(\pi i)^2) = \delta$ to conclude. □

9.3.2 Asymptotic rates for polynomial sphere models

While the number of iterations is log-linear ($\log(\|x_n^- - x^*\|)/n$ is upper bounded by a negative constant), the number of evaluations per iteration (during the Bernstein race) might be arbitrarily large in the general case of the transformed sphere model. More precise results for subclasses of the transformed sphere model will now be considered. The results rely on the following lemma lower-bounding the gap between fitness values on individuals of the population (i.e. the arms of the Bernstein race), defined as:

$$\Delta_n \doteq \max_{i \in [[1,3]]} f(x_n^{(i)}) - \min_{i \in [[1,3]]} f(x_n^{(i)}).$$

Lemma 10. *Assume f belongs to the polynomial S-T sphere model, i.e. $f : x \mapsto \gamma\|x - x^*\|^p + c$ for $\gamma \in]0, \infty[$, $p \in [[1, \infty]]$ and $c \in]0, 1[$. Then, for some constant K depending on dimension d of $\mathcal{X}$ and parameters γ, p of f,*

$$x^* \in [x_n^-, x_n^+] \Rightarrow \Delta_n \geq K\|x_n^+ - x_n^-\|^p. \qquad (9.5)$$

Proof. Let us note $g_n^{\max} = (x_n^+)_c - (x_n^-)_c$. Recall that by definition (see algorithm 8),

$$\|x_n^+ - x_n^-\|/d \leq g_n^{\max} \leq \|x_n^+ - x_n^-\| \qquad (9.6)$$

and let $\bar{x_n^*}$ be the projection of x^* on the line on which lie $x_n^{(1)}, x_n^{(2)}, x_n^{(3)}$. The result will now be proved for $(\bar{x_n^*})_c \in [(x_n^{(1)})_c, (x_n^{(2)})_c]$. In this case, $f(x_n^{(3)}) > f(x_n^{(2)})$. The proof for the case $(\bar{x_n^*})_c \in [(x_n^{(2)})_c, (x_n^{(3)})_c]$ is symmetric (see figure 9.2).

First of all, we have

$$\begin{aligned} \Delta_n &\doteq \max_{i,j \in [[1,3]]^2} \gamma(\|x_n^{(i)} - x^*\|^p - \|x_n^{(j)} - x^*\|^p) \\ &\geq \gamma(\|x_n^{(3)} - x^*\|^p - \|x_n^{(2)} - x^*\|^p). \end{aligned}$$

Without loss of generality, constant γ is set to 1. By Pythagora's theorem, $\forall i \in [[1,3]], \|x_n^{(i)} - x^*\|^2 = \|x_n^{(i)} - \bar{x_n^*}\|^2 + \|\bar{x_n^*} - x^*\|^2$. Thus,

$$\Delta_n \geq \left(\sqrt{\|x_n^{(3)} - \bar{x_n^*}\|^2 + \|\bar{x_n^*} - x^*\|^2}\right)^p - \left(\sqrt{\|x_n^{(2)} - \bar{x_n^*}\|^2 + \|\bar{x_n^*} - x^*\|^2}\right)^p$$

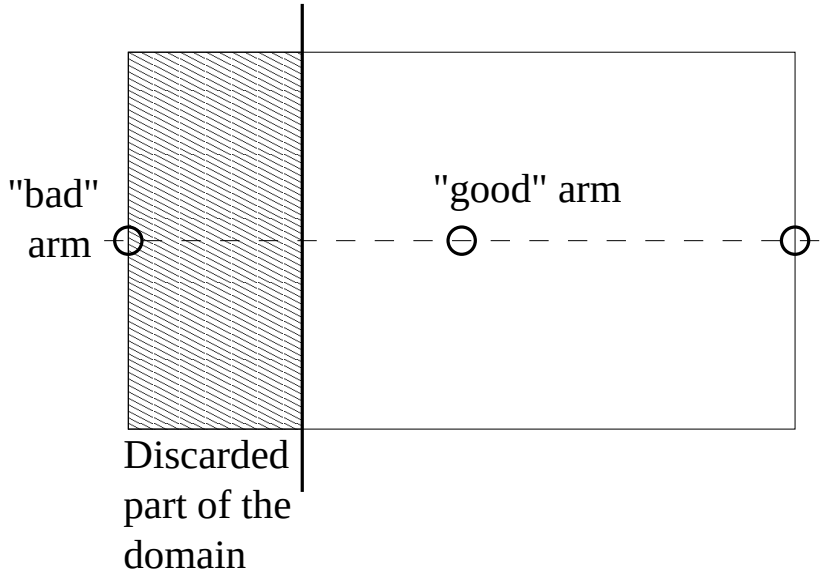


Figure 9.2: The large rectangle is the domain $[x_n^-, x_n^+]$. The three circles are arms $x_n^{(1)}, x_n^{(2)}, x_n^{(3)}$; the left arm is the "bad" arm, whereas the arm in the center is the "good" arm, *i.e.* the one which proved to be closer to the minimum than the left arm, with confidence $1 - 6\delta/(\pi^2 n^2)$.

Note that $\|x_n^{(3)} - \bar{x_n^*}\| = \|x_n^{(2)} - \bar{x_n^*}\| + g_n^{\max}/2$, since $g_n^{\max} = (x_n^+)_c - (x_n^-)_c = \|x_n^{(3)} - x_n^{(1)}\|$. Define $q = \|\bar{x_n^*} - x^*\|^2$ and $a = \|x^{(3)} - \bar{x_n^*}\|$. Then, observing that $g_n^{\max} \geq a \geq g_n^{\max}/2$, we have

$$\begin{aligned}
\Delta_n &\geq \sqrt{a^2 + q}^p - \sqrt{(a - g_n^{\max}/2)^2 + q}^p \\
&\geq a^p \left(\sqrt{1 + \frac{q}{a^2}}^p - \sqrt{\left(1 - \frac{g_n^{\max}}{2a}\right)^2 + \frac{q}{a^2}}^p\right) \\
&\geq \frac{(g_n^{\max})^p}{2^p} \left(\sqrt{1 + \frac{q}{a^2}}^p - \sqrt{\frac{1}{4} + \frac{q}{a^2}}^p\right) \qquad (9.7)
\end{aligned}$$

By setting $u = q/a^2$, it is clear that $\Delta_n 2^p/(g_n^{\max})^p$ is greater than the minimum of $u \mapsto \sqrt{1+u}^p - \sqrt{1/4+u}^p$ on the interval $[0, d]$ (since $\sqrt{q} = \|\bar{x_n^*} - x^*\| \leq$

$\sqrt{d}g_n^{\max}/2$, u ranges in [0,d]). It remains to find the minimum of that function on [0,d]. Its derivative is

$$\frac{p}{2}(1+u)^{p/2-1} - \frac{p}{2}(1/4+u)^{p/2-1}.$$

If $p = 1$, this is always negative, since the left-hand side denominator is always larger than the right-hand side denominator. Therefore, the function is decreasing, and is minimized for $u = d$. Finally, setting $K' = \sqrt{1+d} - \sqrt{1/4+d}$ and injecting in eq. 9.7 yields $\Delta_n \geq K' g_n^{\max}/2$.

If $p \geq 2$, the function is increasing since the derivative becomes always positive, and therefore its minimum is its value in 0, which is, for all $p \geq 2$, at least $\frac{1}{2}$. We then have $\Delta_n \geq (g_n^{\max})^p/2^{p+1}$.

In both cases, using equation 9.6 leads to equation 9.5 for some K. Interestingly, the dependence in d disappears for $p \geq 2$. □

Remark 6. As the proof relies on Pythagora's theorem, the underlying norm for the search space is necessarily L_2. In a bounded domain of finite dimension, however, the norm equivalence support the extension of the result to other norms.

Theorem 11 (Hoeffding rates for the polynomial S-T sphere model)**.** *Assume fitness f is of the polynomial S-T sphere model. The number of evaluations requested by algorithm 8 for reaching precision ϵ with probability at least $1-\delta$ is $O(\frac{p\log(1/\epsilon)+\log(\log(1/\epsilon)/\delta)}{\epsilon^{2p}})$.*

Proof. Equation 9.5 ensures that $\Delta_n = \Omega(\|x_n^+ - x_n^-\|^p)$. Therefore, applying the concentration inequality 7.3, the number of evaluations in the n^{th} iteration is at most

$$O\left(-\log\left(\frac{6\delta}{\pi^2(n+1)^2}\|x_n^+ - x_n^-\|^p\right)/\|x_n^- - x_n^+\|^{2p}\right). \tag{9.8}$$

Now, let us consider the number $N(\epsilon)$ of iterations before a precision ϵ is reached, that is, $N(\epsilon) = \min\{n|\ \|x_n^+ - x_n^-\| \leq \epsilon\}$. From equation 9.8, the cost (the number of evaluations) in the last call to the Bernstein race is

$$Bound_{last}(\epsilon) = O\left(-\log\left(\frac{6\delta}{\pi^2(N(\epsilon)+1)^2}\epsilon^p\right)/\epsilon^{2p}\right). \tag{9.9}$$

From equation 9.2,

$$\epsilon \leq \frac{3}{4}^{\lfloor (N(\epsilon)-1)/d \rfloor}$$

which yields $N(\epsilon) = O(\log(1/\epsilon))$. Then, $Bound_{last}(\epsilon) = O(\log(\log(1/\epsilon)/(\delta\epsilon^p))/\epsilon^{2p})$. For a fixed dimension d, there exists $k' > 1$ such that the cost of the $(N(\epsilon)-i)^{th}$ iteration is at most

$$O(\lceil Bound_{last}/(k')^i \rceil) \tag{9.10}$$

because the algorithm ensures that after d iterations, $\|x_n^+ - x_n^-\|$ decreases by at least 3/4.

The sum of the costs for $N(\epsilon)$ iterations is therefore the sum of $O(Bound_{last}(\epsilon)/(k')^i)$ for $i \in [[0, N(\epsilon)-1]]$, that is $O(Bound_{last}(\epsilon)/(1-1/k')) = O(Bound_{last}(\epsilon))$ (plus $O(N(\epsilon))$ for the rounding associated to the $\lceil ... \rceil$ in equation 9.10).

The overall cost is therefore $O(Bound_{last}(\epsilon) + \log(1/\epsilon))$. This yields the expected result. □

Theorem 12 (Bernstein rates for null variance at optimum)**.** *Assume fitness f is of polynomial sphere model, with $c = 0$. Then, the number of evaluations requested for reaching precision ϵ with probability at least $1 - \delta$ is*

$$O(\frac{p \log(1/\epsilon) + \log(\log(1/\epsilon)/\delta)}{\epsilon^p}).$$

Proof. The proof follows the lines of the proof of theorem 11, except for one point. For the polynomial scaled sphere model as well as for the polynomial S-T sphere model, equation 9.5 holds, which implies

$$\Delta_n = \Omega(\|x_n^+ - x_n^-\|^p). \tag{9.11}$$

However, for the scaled sphere model, we can also claim

$$\sup_{i \in [[1,3]]} f(x_n^{(i)}) = O(\|x_n^+ - x_n^-\|^p) \tag{9.12}$$

because x^* and the $x_n^{(i)}$ belong to $[x_n^-, x_n^+]$ and $f(x_n^{(i)}) = \gamma \|x_n^{(i)} - x^*\|^p$. Equations 9.11 and 9.12 lead to equation 7.4.

Furthermore, equation 7.4 implies that equation 9.9 can be replaced by

$$Bound_{last}(\epsilon) = O\left(-\log\left(\frac{6\delta}{\pi^2 (N(\epsilon)+1)^2}\epsilon^p\right)/\epsilon^p\right). \tag{9.13}$$

The summation as in the proof of theorem 11 now leads to an overall cost of $O(\frac{p \log(1/\epsilon) + \log(\log(1/\epsilon)/\delta)}{\epsilon^p})$. □

9.3.3 Summary and remarks

The bounds address optimization of noisy fitness functions, where the fitness in x is randomized, with values in $[0, 1]$, and expected values $\mathbf{f}(x, x^*)$ where x^* is the optimum. On fitnesses of the form $x \mapsto \gamma \|x - x^*\|^p + c$, the same algorithm (using Bernstein's inequality) ensures that with probability $1 - \delta$, the optimum x^* is in a set of diameter ϵ after $\tilde{O}(1/\epsilon^{2p})$ fitness evaluations if the variance is not null at the optimum, and after $\tilde{O}(1/\epsilon^p)$ evaluations if the variance is null.

This result does not need that the fitness values at a given point be Bernoulli (which was assumed for the lower-bound proof of last chapter): it works for any distribution with values between 0 and 1 (whose variance is therefore upper bounded by the expectation).

The tightness of the bound is established (as matching the lower bound) for $p = 1$ and $c = 0$. Intuitive arguments and experimental results suggest that the

bound might be tight for $p = 1$ and $p = 2$ with $c > 0$, although not tight for other settings. This work opens several perspectives for further research. Firstly, it is expected that the dependency in d can be improved. Secondly, the results can be generalized as the convergence rate only depends on $\mathbf{f}(x, x^*)$ for x close to x^*: all f such that $f(x) = \Theta(||x - x^*||^p)$ lead to the same asymptotic rate as the polynomial scaled sphere; and all f such that $f(x) - c = \Theta(||x - x^*||^p)$ for some c lead to the same asymptotic rate as the polynomial scaled and translated sphere function.

Chapter 10

Experiments

This chapter is devoted to the experimental validation of the results presented in chapters 8 and 9, related to expensive noisy optimization problems; it also presents and experimentally studies some bandit algorithms devoted to noisy optimization.

The chapter is organized as follows. Section 10.1 empirical performance of two algorithms on transformed sphere models (chapters 8 and 9), namely UH-CMA and QLR. Section 10.2 is devoted to a particular type of noisy optimization problem: parameter tuning. Section 10.3 presents a multi-armed bandit algorithm similar to $BAAL$ (chapter 4), applied to expensive optimization.

In a nutshell, the chapter provides empirical evidence that bandit methods are suited to tackle noisy optimization problems.

10.1 Experiments on UH-CMA/QLR

In this section, the empirical performance of two algorithms, an evolution strategy called UH-CMA and introduced by Hansen et al. (2009), and an algorithm based on quadratic logistic regression, therefore called QLR(Coulom, 2010), are studied in light of the theoretical findings of chapters 8 and 9. These results first appeared in Coulom et al. (2011).

Convergence rates are given for minimization; the fitness at point x is the Bernoulli random variable $\mathcal{B}(f(x))$ with parameter $\min(1, \max(0, f(x)))$, x_n is the approximation of the optimum after n fitness evaluations, x^* is the optimum, $c > 0$, and g is some increasing mapping. Results for QLR and for UH-CMA are empirical, based on current versions of the algorithms.

10.1.1 Experimental results for UH-CMA—optimization without surrogate models

UH-CMA has been developed with intensive testing on the BBOB challenge (Auger et al., 2010), which includes mild models of noise[1]. The optimization domain is $\mathbb{R}^2$. Let $\mathcal{B}(q)$ denote a Bernoulli distribution of parameter q, $\mathcal{N}(\mu, \sigma^2)$ denote a Gaussian distribution centered on μ with variance σ^2, and $\mathcal{U}(I)$ denote a uniform distribution on interval I. UH-CMA was tested on four different noise settings (noting again $Y(x)$ the random variable associated to the computation of $f(x)$):

1. $Y(x) \sim \mathcal{N}(\|x\|^2, 0.1)$;
2. $Y(x) \sim \|x\|^2 + \mathcal{U}([0, 1])$;
3. $Y(x) \sim \mathcal{B}(\|x\|^2)$;
4. $Y(x) \sim \mathcal{B}(\|x\|^2 + 0.5)$.

The initial values required by UH-CMA to start the search were sampled from $\mathcal{U}([0, 1]^2)$. The convergence (and divergence) of UH-CMA—illustrated on figure 10.1—is known to be log-linear.

For the first setting (Gaussian noise), the algorithm converges efficiently: the precision decreases exponentially as the number of iterations increases. For $Y(x) \sim ||x||^2 + \mathcal{U}([0, 1])$, the precision stops improving after a few hundred iterations. For $\mathcal{B}(\|x\|^2)$ and $\mathcal{B}(\|x\|^2+0.5)$ we can see that the algorithm diverges. The available implementations of UH-CMA cope quite well with small noise situations, but as soon as the variance does not go to zero sufficiently fast they do not succeed.

After personal communication with Hansen (2010), it appears that adding some specific rules for averaging multiple fitness evaluations depending on the step-size, specifically for each fitness function lead to better rates. However, the rates are still dominated by QLR.

10.1.2 Experiments with QLR—optimization with surrogate models

QLR is based on a Bayesian quadratic logistic regression. It samples regions of the search space with maximum variance of the posterior probability, i.e. regions with high variance conditionally to past observations. This is a key difference w.r.t. algorithms without surrogate models, which tend to sample points close to the optimum. QLR is fully described by Fackle Fornius (2008), Chaloner (1989), Khuri et al. (2006) (design of experiments for quadratic logistic model), and by Schein and Ungar (2007) (active learning for logistic regression). See Coulom (2010) for a description of the algorithm used here specifically tailored to binary noisy fitnesses.

[1]The source code for these experiments is publicly available, see Hansen (2008).

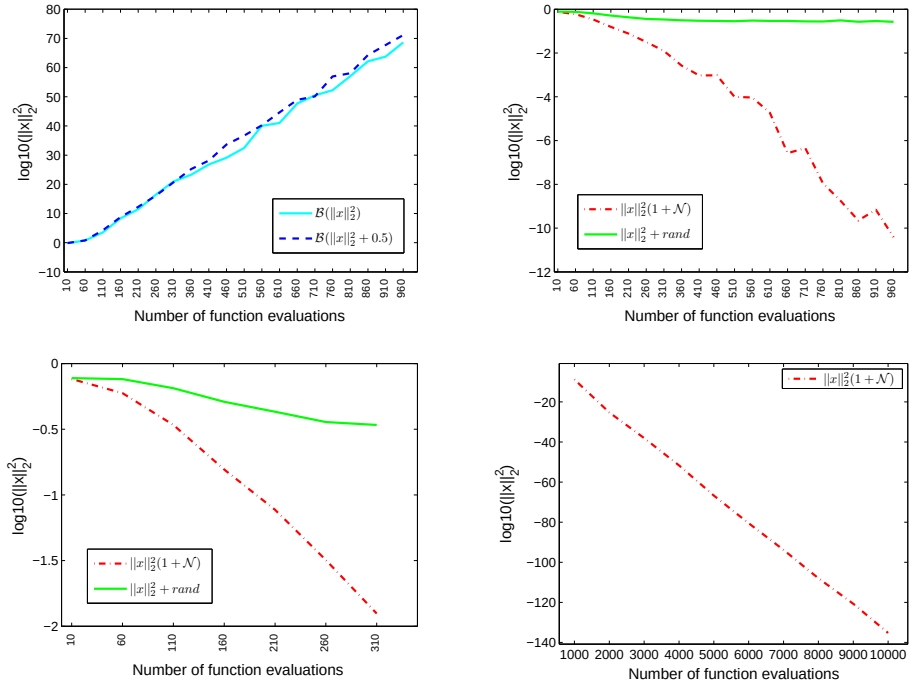


Figure 10.1: Optimization with UH-CMA. The number of fitness evaluations is plotted against the learning precision in log-scale (base 10). Top-left: Bernoulli models, exhibiting a divergence. Bottom-left: Gaussian and uniform models, with up to 300 fitness calls: log-linear behaviors occur. Top-right: Gaussian and uniform models, with up to 1000 fitness calls: UH-CMA stops improving for the uniform noise model. Bottom-right: Gaussian model with up to 10000 fitness calls: the loglinear improvement is confirmed.

QLR was tested on fitnesses whose noisy measurements follow a law $\mathcal{B}(||x||^p + c)$, for p in $\{1, 2\}$ and c in $\{0, 1/2\}$. The domain is $\mathbb{R}^2$. Figure 10.2 shows the experimental results:

Top left (p=1, c=0) QLR converges on $\mathcal{B}(||x - x^*||)$, but with a suboptimal exponent $\frac{1}{2}$ (the slope of the curve is $-\frac{1}{2}$ in log-scale), i.e. $f(x_n) - f(x^*) = \Theta(1/\sqrt{n})$, which translates in $n = \Theta(1/\epsilon^2)$. The upper bound for R-EDA is $n = \tilde{\Theta}(1/\epsilon)$ in this case;

Top right (p=1, c=1/2) QLR converges with supposedly optimal rate $n = \Theta(1/\epsilon^2)$ which match the optimal rate of $R - EDA$;

Bottom left (p=2, c=0) QLR reaches $f(x_n) - f(x^*) = f(x_n) = \Theta(1/n)$ which translates into $n = \Theta(1/\epsilon^2)$ since $f(x) = \|x\|^2$. This is similar to the upper bound of $R - EDA$.

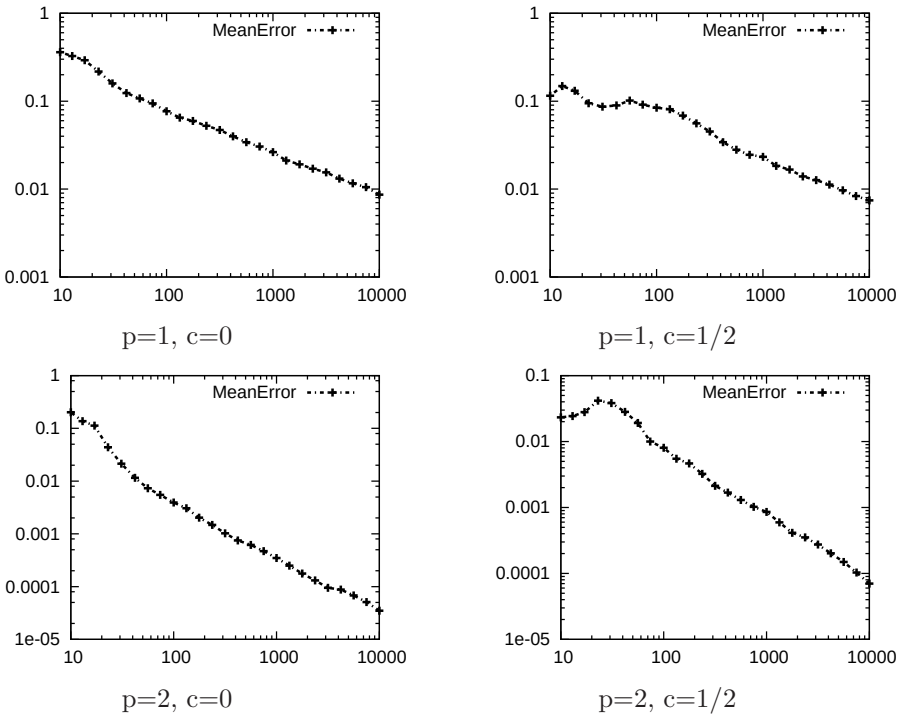


Figure 10.2: Convergence rates of QLR. The average precision $f(x_n) - f(x^*)$ is reported versus the number of fitness evaluation in log scale. The noisy fitness measurements are of the form $Y(x) \sim \mathcal{B}(||x||^p + c)$ (top: $p = 1$, bottom: $p = 2$, left: $c = 0$, right: $c = 1/2$).

Bottom right (p=2, c=1/2) QLR still reaches $f(x_n) - f(x^*) \simeq \Theta(1/n)$, thus $n = \Theta(1/\epsilon^2)$ whereas the bound on R-EDA only guarantees $n = \Theta(1/\epsilon^4)$.

10.1.3 Discussion

The convergence rates for R-EDA and QLR are summarized in table 10.1.3. For the rightmost column, it is important to point out that we tested QLR *without* knowledge of the parameter p or c, so that the comparison with other algorithms is fair. In particular, R-EDA, which provably realizes the upper bounds above, does not require such a knowledge. The quality of an optimization algorithm clearly depends on the required prior knowledge: the less, the better.

As a partial conclusion, $R - EDA$'s behavior is shown to be efficient with respect to all considered types of noise. However, in the (easy) case $\mathcal{N}(||x - x^*||^2, 0.1)$, UH-CMA is the only algorithm reaching a log-linear convergence.

In the particular case of $\mathcal{B}(c + ||x - x^*||^2)$, UH-CMA does not converge, suggesting that algorithms tailored to small noise models uneasily extend to large noise assumptions. $R - EDA$'s convergence rates are outperformed by

f	$n(\epsilon)$ for R-EDA	Known lower-bound	$n(\epsilon)$ for QLR
$\gamma\|\|x-x^*\|\|$	$\tilde{O}(1/\epsilon)$	$\Omega(1/\epsilon)$	$\simeq 1/\epsilon^2$
$\gamma\|\|x-x^*\|\|+c$	$\tilde{O}(1/\epsilon^2)$	$\Omega(1/\epsilon)$	$\simeq 1/\epsilon^2$
$\gamma\|\|x-x^*\|\|^2$	$\tilde{O}(1/\epsilon^2)$	$\Omega(1/\epsilon)$	$\simeq 1/\epsilon^2$
$\gamma\|\|x-x^*\|\|^2+c$	$\tilde{O}(1/\epsilon^4)$	$\Omega(1/\epsilon)$	$\simeq 1/\epsilon^2$
$\gamma\|\|x-x^*\|\|^p$	$\tilde{O}(1/e^p)$	$\Omega(1/\epsilon)$	–
$\gamma\|\|x-x^*\|\|^p+c$	$\tilde{O}(1/\epsilon^{2p})$	$\Omega(1/\epsilon)$	–
$g(\|\|x-x^*\|\|)$	$o(1)$	–	–

Table 10.1: Rates for $R-EDA$ and QLR. Typical cases $p=1$ and $p=2$ are emphasized.

QLR in this case. An explication is that QLR is tailored to Bernoulli-like noise models, while $R-EDA$ is not limited to those models.

Table 10.1.3 shows that the lower bounds given in chapter 8 for $p>1$ or $c>0$ are not tight. Further work, taking inspirations from the intuitions of section 8.3 will be devoted to finding tighter bound.

10.2 Simple Regret and Optimization

This section focuses on a particular case of noisy optimization, namely parameter tuning.

Bandits and races can be used even more directly for noisy optimization, and specifically parameter tuning. We introduce a parameter-tuning bandit algorithm, UCB_p (introduced below), which is compared to the baseline $Uniform$ selection (section 2.3.3). First, a practical problem is addressed: tuning the parameters of the UCT-based Go-Playing program MoGo. Second, artificial problems are derived from this real-world problem, and used to investigate simple regret procedures further. These results have been included and published in Coulm et al. (2010).

10.2.1 The tuning of MoGo

Monte-Carlo tree search (Chaslot et al., 2006; Coulom, 2006; Kocsis and Szepesvari, 2006) (introduced in chapter 4) is efficient both for games and for planning problems, and it performs particularly well when the size of the state space is huge, adversely affecting supervised learning. MoGo (Lee et al., 2009) is one of the main MCTS programs[2], dedicated to a major testbed for artificial intelligence techniques: the game of Go.

MoGo can be viewed as a complex system, involving a large number of parametrized modules Chaslot et al. (2009), each requiring specific tuning. In this section, a specific parameter tuning will be considered (detailed in Lee

[2]The authors of MoGo make its CVS available upon request.

Algorithm	Horizon	Selected arm	Generalization perf.
Baseline			51.37 % ± 0.3%
UCB(4)+MPA	88 000	(0.00,0.09)	52.65% ± 0.3%
UCB(4)+MPA	150 000	(0.03,0.09)	52.42% ± 0.3%
Uniform+EBA	88 000	(0.09,0.03)	52.00% ± 0.3%

Table 10.2: Comparison between the different algorithms in the non-blitz settings, where MPA stands for "most played arm", while EBA stands for "empirical best arm".

et al., 2009), concerned with biasing the patterns used in MoGo. The tuning problem focuses on optimizing two discretized parameters. The nature of the handcrafting and the meaning of the parameters are not very relevant to this work: they are described in Coulm et al. (2010).

Let us consider the tuning those two parameters of MoGo for 10 seconds per move in 19x19. Using the $Uniform$ arm pulling strategy, each parameter (α_1, α_2) in $\{-0.09, -0.06, -0.03, 0, 0.03, 0.06, 0.09\}^2$ was tested nearly 1800 times—which has a huge computational cost.

The same experiment was then repeated, but with UCB_p as arm-pulling strategy instead of uniformly sampling all the values of the parameters. UCB_p is a very simple modification of UCB1 introducing a constant p adjusting the strength of the exploration term. With the notations of section 2.3.3,

$$UCB_p(t) = \arg \max_{a \in [[1,K]]} \left[\hat{X}_{a,T_a(t)} + \sqrt{p \frac{2 \ln(t)}{T_a(t)}} \right].$$

In this experiment, p is set to 4.

The two approaches are compared to the baseline algorithm (without tuning) from the point of view of their generalization performance in table 10.2. The baseline is the algorithm without any tuning. The performance of Baseline, UCB_p and $Uniform$ is the percentage of victories of the black player against a white player that always uses the untuned algorithm (Baseline). The black player has a slight advantage in Go, which explains why the Baseline has a generalization performance strictly bigger than 50% when playing against itself. Although the table shows that UCB_p and $Uniform$ outperform the baseline (statistically significantly for UCB_p, and less so for $Uniform$), some uncertainty remains as each result comes from a single run of the tuning algorithm[3]. Extended experiments are therefore performed.

The results are somewhat relevant in the sense that the algorithms provided an arm which significantly outperformed the baseline (although the statistical significance of the dominance of *Uniform* over Baseline is questionable). However, some uncertainty remains since these results are equal within 2 standard deviations and each correspond to a single run of the algorithm. This is why carefully designed artificial experiments will be performed in the next section.

[3]However, the performance of tuned MoGos are tested by performing large numbers of games against the baseline.

10.2.2 The tuning of MoGo—extended experiments

The purpose of this experimental section is to empirically support multiple claims regarding multi-armed bandit problems optimizing simple regret instead of cumulative regret.

Let us bear in mind that the goal is to get the minimal expected simple regret for a given number of pulls. As mentioned in section 2.3.3, Bubeck et al. (2008) have shown that uniformly pulling arms is asymptotically optimal in this case. Nevertheless, they also suggest that in practical applications, using strategies designed for cumulative regret on simple regret problems often yields much better results, although those strategies are suboptimal asymptotically.

Remember that after having exhausted the allowed number of pulls to choose an arm, various recommendation rules are used to decide which arm to keep:

Empirical best arm (EBA): the arm kept is the one with the best empirical mean;

Most played arm (MPA): the arm is the one which has been played most often;

Empirical distribution of plays (EDP): the arm is selected by random draw among the K arms, where each arm has a probability of being chosen proportional to the number of times it has been played.

it must be noted however that only EBA can be used for the $Uniform$ selection strategy: MPA makes no sense, since the first $N\%K$ arms have been played exactly once more than the remaining arms. Furthermore, EDP degenerates to randomly choosing an arm.

For UCB-based selection strategies, or other confidence bound strategies, EBA is the most aggressive choice, since an arm that has been played less, but that has a better average, will be preferred. On the other hand, MPA will select arms with high confidence. Both recommendation rules are sound, however, and converge to the same result for confidence-bound based selection strategies.

The following experiments investigate these arm selection procedures on three artificial experiments constructed from the MoGo real-world application, designed as follows:

- the $Uniform$ strategy was applied during a very long time on all parameters;
- the results (estimated generalization performance for each parameter) were recorded;
- various strategies could then be simulated "offline"; for each simulation, the random draws were permuted, so that there is no systematic bias.

Each point of each curve is the result of an average over 1600 experiments. The 95% error bars were almost unnoticeable, and therefore they are not depicted on the graphs. Consequently, even small gaps between curves are statistically significant.

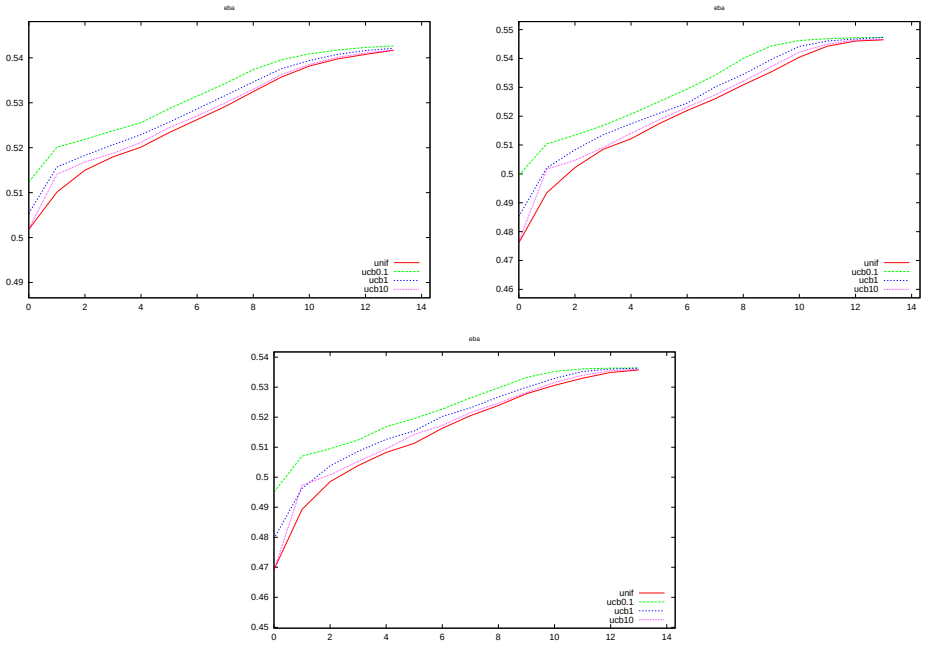


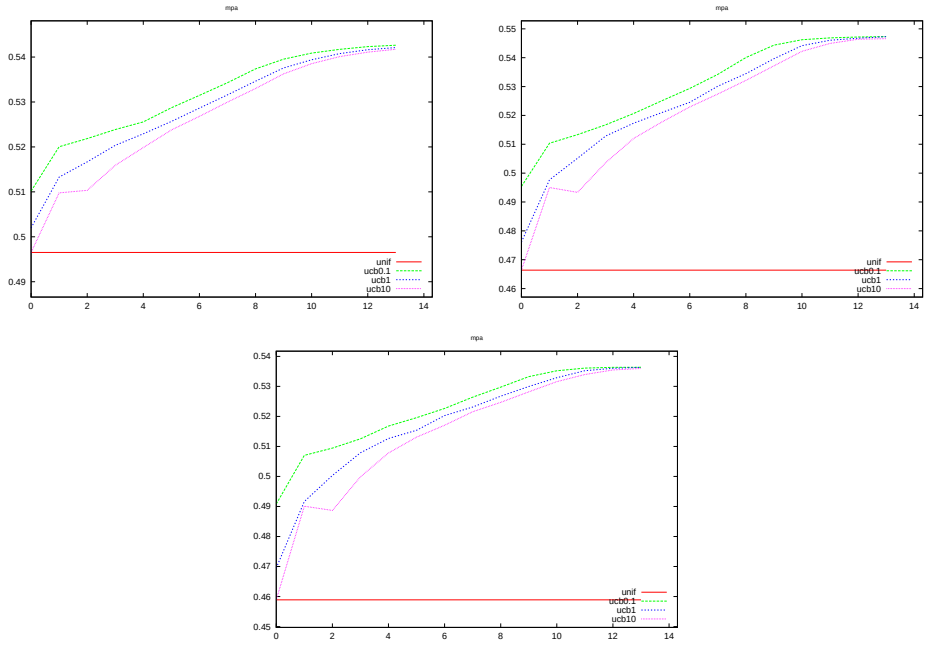
Figure 10.3: Results for empirical best arm, with regret strategies compared on the blitz case as black (on the left), blitz case as white (on the right) and standard case (on the bottom) for various horizons. The Y-axis is the true mean of the arm chosen by a simple regret strategy being allowed H pulls on arms on the whole. On the X-axis lies $\log(H/100)$. The order of the curves is as follows: in all cases, $UCB_{0.1} \geq UCB_1 \geq UCB_{10} \geq Uniform$, and the difference vanishes as the horizon goes to ∞.

Each problem consists in a few hundred arms whose rewards are Bernoulli distributions with parameter p close to $1/2$. Experiments are performed by confronting strategies $Uniform$, $UCB_{0.1}$, UCB_1 and UCB_{10} for each recommendation rule (EBA, MPA and EDP). For each of the 3 experiments, each selection strategy and each decision rule, experiments were performed for various horizons (number of allowed pulls), going from 100 to 819200. The results are presented on figures 10.2.2, 10.2.2 and 10.2.2 and show that

- UCB performs better than $Uniform$;

- the improvement is moderate. A strength of $Uniform$ is that the performance is naturally evaluated on the fly for all arms; therefore both the statistical validation and the visualization are straightforward.

Figure 10.4: Results for most played arm, with regret strategies compared on the blitz case as black (on the left), blitz case as white (on the right) and standard case (on the bottom) for various horizons. The Y-axis is the true mean of the arm chosen by a simple regret strategy being allowed H pulls on arms on the whole. On the X-axis lies $\log(H/100)$. The order of the curves is as follows: in all cases, $UCB_{0.1} \geq UCB_1 \geq UCB_{10} \geq Uniform$, and the difference vanishes as the horizon goes to ∞.

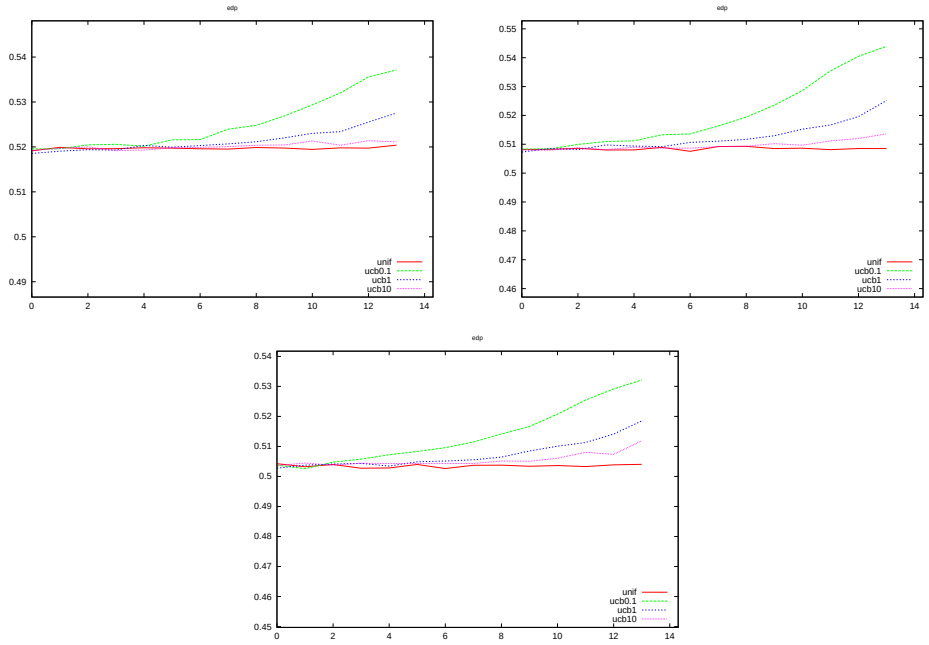


10.2.3 Discussion

This section has considered simple regret algorithms viewed as noisy optimization algorithms, without assuming any structure of the search space. The comparison of $Uniform$ (known as optimal for sufficiently large horizon, i.e. sufficiently large time budget) and UCB_p shows that:

- **Uniform vs Sophisticated** If the number of arms K is very small relatively to the number of pulls the $Uniform$ strategy is the recommended one; at its best, UCB_p can reduce the number of calls by a number $K\log(K)$. When K gets larger w.r.t. the horizon, UCB becomes better. Note that the larger p is, the more UCT degenerates into pure exploration: this is why UCB_{10} has results close to $Uniform$ while $UCB_{0.1}$, emphasizing exploitation, performs better. A drawback is that UCT does not include any statistical validation, and cannot be easily parallelized;

Figure 10.5: Results for empirical distribution of plays, with regret strategies compared on the blitz case as black (on the left), blitz case as white (on the right) and standard case (on the bottom) for various horizons. The Y-axis is the true mean of the arm chosen by a simple regret strategy being allowed H pulls on arms on the whole. On the X-axis lies $\log(H/100)$. The order of the curves is as follows: in all cases, $UCB_{0.1} \geq UCB_1 \geq UCB_{10} \geq Uniform$, and the difference vanishes as the horizon goes to ∞.



- **Results on the application to MCTS.** For the specific MoGo application, the results were significant but moderate; however, it can be pointed out that many handcrafted modifications around Monte-Carlo tree search provide such small improvements of a few percents each. Moreover, improvements performed automatically by bandits can be applied incrementally, leading to huge cumulative improvements;

- **Comparing recommendation techniques: most played arm is better.** Empirical best arm and most played arm in UCB are usually the same (this is not the case for various other bandit algorithms), and are much better than the "empirical distribution of play" recommendation strategy, which is not tailored for this application[4]. This result confirms earlier work (Wang and Gelly, 2007) suggesting that MPA is the best recommendation rule.

[4] As already mentioned, most played arm and the empirical distribution of plays obviously do not make sense for $Uniform$.

Further work will investigate the application of the proposed approach to larger search spaces. Another direction is to consider UCB-Tuned (Auer et al., 2002), taking into account the variance of the results attached to a given arm.

10.3 Optimal Expensive Noisy Optimization in Finite Horizon

This section presents yet another multi-armed bandit algorithm applied to optimization, inspired from *BAAL* (active learning, chapter 4), and called OUCT (Optimization with UCT). Following the same procedure as for active learning, an optimal expensive noisy optimization strategy is derived, and OUCT embodies the Monte-Carlo tree search used to approximate the optimal optimization strategy.

A specificity of OUCT is that it deals with noisy functions, while *BAAL* is currently limited to the noise-free case. This difference is at the root of a few small differences in the theoretical optimality formulation as well as in the algorithm itself.

10.3.1 Formalization

We will first formalize non-linear noisy optimization as a planning problem, along the same lines as in chapter 4. The main difference is the use of a partially observable Markov decision process for optimization:the noisy setting makes a POMDP formulation more appropriate.

The idea of seeing optimization as a planning problem has already been suggested by Igel (2004), with the conclusion that the problem is not tractable. It has been reconsidered by Auger and Teytaud (2009), with positive results in a simplified noise-free setting.

The main finding of this section is to show that even in the noisy setting, UCT and general methods can be used to make the problem tractable, at least for small horizon, i.e. when the number of time steps in the planning problem is not too large. Furthermore, the approach can be applied in general to any problem that can be formulated as a partially observable planning task.

As already mentioned, this approach is suited to expensive fitness functions: it needs a lot of computational power for choosing each iteration. Besides, noisy optimization makes the problem much more challenging because the process is even less observable than in the classical optimization framework. Note that optimization, like active learning, can indeed be formalized as a partially observable problem in any case, as the fitness function is always unknown; still, the unobservable part is easier to deal with in the deterministic case.

Let us denote $\mathcal{F}$ the class of fitness functions considered. Let $P_{\mathcal{F}}$ be a probability measure on $\mathcal{F}$, and given $f \in \mathcal{F}$, let $p_f(.|x)$ be the density associated to noisy measurements of f. With same notations as section 4.1, the noisy

optimization process clearly matches the active learning process (equation 4.2):

$$\begin{cases} p^{(0)}(s|f) & = \delta_{\emptyset}(s) \\ p(s'|s,x,f) & = \delta_{s+(x,y)}(s')p_f(y|x) \\ \pi(x|s,f) & = \pi(x|s) = \mathbf{S}(x,s). \end{cases} \tag{10.1}$$

The sampling process however involves noise in fitness evaluations, modeled by p_f. Equation 10.1 defines a POMDP whose states are made of training sets s *and* fitnesses f. The transition density $p(s'|s,x,f) = p((s',f)|(s,f),x)$ defined above embodies the noisy measurements.

As in section 4.1, a complete state is a pair (s,f) containing a training set and an objective function. The set of observations, Ω, is the set of all possible noisy training sets. If an agent is in state (s,f), he/she observes only training set s: conditional observation density ω is deterministic, and simply defined as $\omega : (s,f) \mapsto \delta_s$.

The intuitive arguments presented in section 4.1 and the work of Astrom (1965) suggest that an optimal optimization strategy is obtained by solving the above POMDP. The proof of this claim is left for further work.

10.3.2 Algorithm

OUCT proceeds like *BAAL*: the possible strategies are sought via UCT, progressive widening is used to deal with the fact that the domain is continuous, and optimization heuristics guiding the search can be integrated in OUCT just as the maximal uncertainty heuristic for active learning was integrated in *BAAL*. The most important difference between OUCT and *BAAL* is that since OUCT deals with noise, there is no concept equivalent to a "version space" that would contain all the optima consistent with observations.

As such, *conditional sampling* techniques are required to pick surrogate fitnesses so that they fit the noisy observations. Indeed, when selecting a point after having visited points $x_1, \ldots, x_i$, with noisy fitness values $y_1, \ldots, y_i$ in OUCT, a posterior probability can be computed: assuming the noise model is known, each fitness function f of $\mathcal{F}$ can be mapped to a probability describing how likely f produced outcomes (x_i, y_i), $p_{\mathcal{F}}(f\ |\{(x_j,y_j)|j \in [[1,i]]\})$. Let us assume as in previous chapters that f can be parametrized by its optimum x^*, and write this probability $p(x^*\ |\{(x_j,y_j)|j \in [[1,i]]\})$.

As mentioned in chapter 4, there are at least three classical methods to sample from constrained spaces. Rejection methods are the simplest but most expensive solutions. Billiard methods, more difficult to use, are much faster; Markov chain Monte-Carlo sampling is another quite general method with sometimes better performance than rejection methods. Unfortunately, billiard algorithms only handle a few particular densities yet (such as the uniform density), and to our best knowledge no extension exist to apply this method to the general case. Therefore, OUCT must rely either on rejection algorithms or on MCMC algorithms. It is assumed that $p((x_1,y_1),\ldots,(x_i,y_i)|x^*)$ can be computed and that the prior on x^*, p_0, can be sampled at will.

Rejection methods

Rejection methods for specific densities are slightly different from the uniform rejection method on a constrained domain presented in section 4.4. They consist in:

- generating $r_1, r_2, r_3, \ldots$ independently and uniformly in $[0,1]$ and x_1^*, x_2^*, x_3^*, ... independently from p_0;
- choosing x_j^* with j minimal such that $r_j < P((x_1,y_1),\ldots,(x_i,y_i)|x_j^*)$.

It is known that rejection methods are consistent, i.e. they effectively generate x^* drawn from a law of density $p(.|(x_1,y_1),\ldots,(x_i,y_i))$. Of course, the procedure can be very slow if the likelihood of $(x_1,y_1),\ldots,(x_i,y_i)$ is close to 0. This likelihood may decrease exponentially in the number of observations in our case, making rejection methods hardly tractable.

Markov chain Monte-Carlo (MCMC).

A very simple and classical form of Markov chain Monte-Carlo (MCMC) method is presented here, referring the interested reader to Gilks (1995) for a comprehensive presentation. It is assumed that a distribution of probability K_{x_0} on the domain of x^* is given (called a transition kernel), for each possible realization x_0 of x^*, and that $K_{x_1}(x_0) = K_{x_0}(x_1)$ (the transition kernel is symmetric).

Let L be some large integer.
Let m be some point in the domain of x^*.
for $i \in [[1,L]]$ **do**
  Let m' be drawn according to distribution K_m.
  Let r be randomly independently uniformly distributed in $[0,1]$.
  Let s be $\frac{p((x_1,y_1),\ldots,(x_i,y_i)|m)p_0(m)}{p((x_1,y_1)\ldots,(x_i,y_i)|m')p_0(m')}$.
  if $r \leq s$ **then**
    $m = m'$
  end if
end for

Under some very general assumptions, it is shown that asymptotically in L, the output m is distributed according to $p(.|(x_1,y_1),\ldots,(x_k,y_k))$. The main issue with MCMC strategies are the free parameters to set, and in particular the choice of transition kernel K.

Improved rejection

Given the limitations of both above methods, respectively the high cost of the rejection method and the parameter tuning of the MCMC, an improved rejection method, called rescaling, was used in OUCT.

- generate $r_1, r_2, \ldots$ independently and uniformly in $[0,1]$;

- choose x_j^* with j minimal such that

$$r_i < \frac{P((x_1, y_1), \ldots, (x_i, y_i)|x_j^*)}{\max_x P((x_1, y_1), \ldots, (x_i, y_i)|x_j^*)} \tag{10.2}$$

This implies the computation of $\max_x P((x_1, y_1), \ldots, (x_i, y_i)|x_j^*)$ before starting the rejection method. This is done using $1+1$-evolution strategy (ES) and one-fifth rule (Rechenberg, 1973), with 37 random restarts. In the fixed budget of 12 hours, the rescaling method could perform 4 times more simulations, with similar results: the method is empirically sound.

10.3.3 Experiments

In this section, various simple experiments are conducted as a proof-of-principle of the soundness of the OUCT approach.

OUCT is compared to random search, and the evolution of OUCT's performance with respect to the computational power involved for each point selection (represented by the number of simulation) is depicted. This is done in order to check the tractability and consistency of the algorithm at least for small values of the horizon (i.e. the number of time steps).

The experimental setting is as follows:

- the optimization domain is $[0, 1]^d$, where d ranges in $[[1, 4]]$; the optimum is uniformly randomly drawn in $[0, 1]^d$;
- the time horizon is 13 (i.e. 13 points are visited). The error is the average distance to the optimum for the 13^{th} point;
- the random seed is discretized (replaced by a sample of 1000 values sampled by rejection);
- the bandit is applied on a fixed pool of $30d$ points; this random (uniform, independent) pool is the same at all nodes of the UCT tree;
- the fitness function is the Bernoulli sphere function, i.e. the fitness of x is 1 with probability $\min(1, \|x - x^*\|^2)$.

The results of OUCT are comparatively assessed to those of two baselines. The first one is provided by running OUCT with a single simulation (returning the maximum likelihood given fitnesses of 12 random points); the second one is the constant algorithm returning the central point of the search space. Note that in a noisy setting, and with only 13 points, these baselines are not trivial to overcome.

The experimental results (figure 10.6) show that the distance to the optimum (on the Y-axis) decreases with the computational power (in X-axis, log-scale). The random search/maximum likelihood approach is outperformed when OUCT is used on top of it, and the constant baseline is exceeded too. An MCMC method with Gaussian transition kernel was also tested in addition to the rejection method with rescaling. The less conclusive results are likely to be due to a bad tuning of the MCMC algorithm.

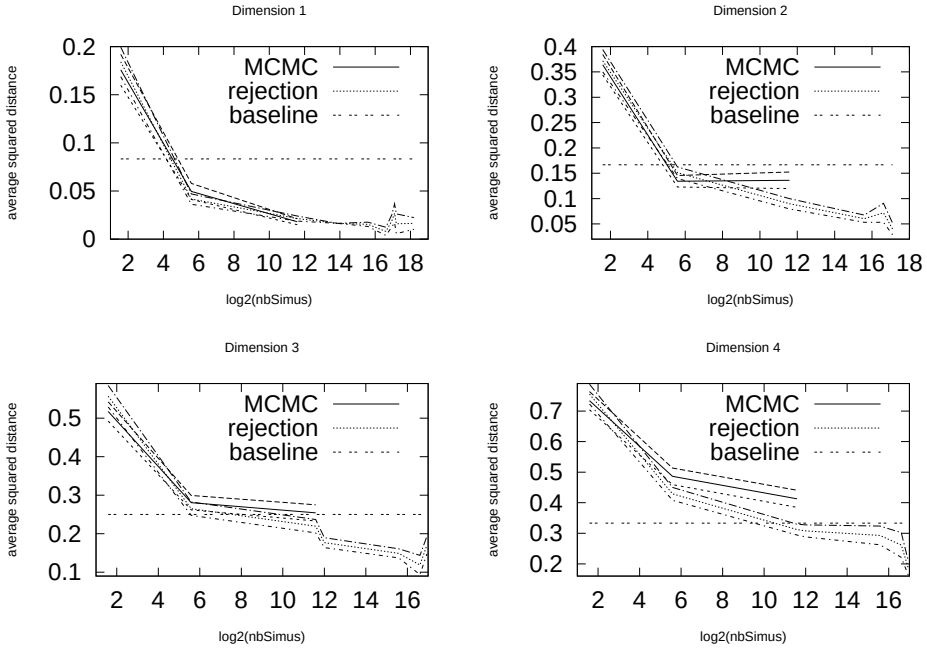


Figure 10.6: Performance of OUCT measured by the average square distance between x_i and $\arg\min f^*$ (Y axis) as a function of the number of simulations (X axis). The baseline is the naive algorithm which just outputs the middle of the domain as the optimum. The y-intercept corresponds to the performance of a random search with maximum likelihood (details in the text).

10.3.4 Partial conclusion

Let us summarize the findings of this section:

- expensive noisy optimization can be formalized as a POMDP problem and bandit-based tree search can be used to yield an approximate optimal strategy;
- the main difficulty, induced by the presence of noise, is that sampling from conditional densities becomes expensive since billiard strategies cannot be used. A specific heuristic, rescaling, has been proposed, yielding some performance improvement even for small horizon;
- OUCT can be hybridized with noisy optimization heuristics just as $BAAL$ can be hybridized with active learning heuristics. Ongoing work, related to the use of the BREDA heuristic (Gelly et al., 2007) within OUCT shows promising results.

Part IV

Conclusion

Chapter 11

Conclusion

The core focus of this manuscript is on machine learning an optimization for expensive functions. The general frameworks of Markov decision processes and multi-armed bandit algorithms were chosen to tackle these problems, yielding multiple theoretical and algorithmic results.

This chapter concludes by discussing perspectives regarding the three main lines of work addressed in this manuscript, namely noisy expensive optimization algorithms (section 11.1, Rolet and Teytaud, 2010c,b; Coulom et al., 2011), batch active learning potential (section 11.2, Rolet and Teytaud, 2010a) and finally Markov decision processes and Monte-Carlo tree search for expensive problems (section 11.3, Rolet et al. 2009a,b).

11.1 Noisy Optimization: Bounds, Algorithms

Noisy optimization was analyzed in this manuscript from the angle of comparison-based algorithms known as EDAs, in a classical setting of the related literature: monotonic transformations of the sphere model. A strength of the analysis is that the noise model is fairly general.

The general lower bound of chapter 8 is one of the first of its kind. Although it can be applied in many settings, it is more relevant to functions that are $O(\|x - x^*\|)$ in the neighborhood of the optimum: the bound is tight up to logarithmic factors in this case. A further work is to improve the bound from $O(\log(D)/\epsilon)$ to $O(D/\epsilon)$[1].

In the more general case of functions $O(\|x - x^*\|^p + c)$, the bound of course remains valid. It is however believed that the bound is not tight, as explained in chapter 8. This is why a natural extension to this research would be to improve the bound in these cases. Analyzing the bound when the model is not exactly a monotonic transformation of the sphere could also be worthwhile: the proof remains valid, but the distance used in the proof may not yield the same interpretation.

[1] Ongoing work that exploits *kissing numbers* to derive this very result.

The algorithm designed to prove the upper bound is another example of the success of multi-armed bandit frameworks in optimization, applied in this case through the *race* formalism. $R-EDA$ matches the lower bound for the simple sphere model, and exhibits a rate that is likely to be optimal either in the translated sphere model (when the fitness is $O(\lambda\|x-x^*\|+c)$). The convergence rate adapts to the model: there is no need to specify the value of c to obtain the best possible rates. However, empirical evidence points to a non-optimal behavior in polynomial cases of degree 2 when c is not null: algorithms such as QLR have empirical convergence rates that overcome the theoretical convergence rate of $R-EDA$. The next step is then to try and prove that the rates of QLR are optimal, maybe by adapting $R-EDA$ so that it can deal with polynomial fitnesses more accurately, while remaining good on $p=1$ cases, so that it would keep its nice "adaptive" feature.

11.2 Batch Active Learning Potential

The work on batch active learning revolved around the question "is it worth parallelizing a sequential active learning technique?". Sometimes, for the desired application, it is possible to obtain a batch of function points rather than obtaining answers one-by-one. Some examples for that come (again) from the approximation of heavy computational codes. The codes are often black-boxes, that are very hard to parallelize. It is however not a (scientific) challenge to run them on multiple processors at once. Nevertheless, it might cost lots of money, maintenance time and/or programming effort. By providing upper and lower bounds on how a parallel active learning algorithm may improve on its sequential counterpart, chapter 5 emphasizes the possible benefits of parallelization for low batch sizes, i.e. batch sizes in the order of the problem's complexity, measured by its dimension or VC-dimension. It also shows that large-scale parallelization is probably not worthwhile: it will only yield a logarithmic improvement.

A crucial point is that active learning algorithms are difficult to parallelize, on the contrary of passive learning. Many active learning heuristics can suggest multiple instances for labeling at a given point in the learning process. However, it is likely that a parallelization that simply labels all those points will not be very efficient. Indeed, an active learning suggestion *after* obtaining the label of an instance is often quite different from *before* obtaining the label: the added information changes the shape of the version space. An example that illustrates this idea well is the following: consider two points that are very close to each other. In many scenarios, if one of them is good for the learning, the other one is probably good too. However, once the first one has been labeled, the second one becomes uninteresting, since their label are likely to be similar (of course, this is not true for *all* active learning situations).

Consequently, an interesting work would be to focus on a generic method to turn a sequential active learning algorithm into a parallel algorithm. Interestingly, the "simulation" result of chapter 5 is such a generic method. Further, a surprising fact is that the algorithm of chapter 4, $BAAL$, can be used in a

way that mimics this "simulation" method: after performing the N authorized tree walks, a tree of possible learning scenarios is stored in memory. This tree may be used to select not only the best instance at depth one, but also the best instances at depth 2 for each possible label of the first example, and the best instances at depth 3 or more: this is almost exactly what the upper-bounding algorithm of section 5.4 does.

Finally, it would be useful to know what happens between the linear speedup for small batch sizes (less than the VC-dimension), and the logarithmic speedup for very large batch sizes (asymptotically). It seems hard to maintain the linear speedup on the simple classifiers of section 5.4 once the batch size reaches the VC-dimension: a formal proof showing that the speedup steadily becomes logarithmic would be a great addition to this work.

11.3 Markov Decision Processes and Monte-Carlo Tree Search for Expensive Problems

A major contribution was the formalization of optimality for active learning and expensive optimization in terms of MDPs (or POMDPs) for finite horizon problems. From this formulation, bandit-based algorithms *BAAL* and OUCT were derived to approximate optimal strategies. They outperform classical algorithms on proof-of-principle experiments using simple target functions.

There are many extensions to this work. On the theoretical side, the MDP formulation is based on a definition of optimality that uses a prior on the set of hypotheses: the strategy is optimal *in expectation* with respect to this prior. It would be interesting to see if the MDP formalization can be adapted to a *worst-case* definition of optimality: the strategy would aim at having the best possible generalization error with the worst possible hypothesis from the chosen hypothesis class. This would correspond to an *adversarial* setting, which does not require a probabilistic prior. Nonetheless, adaptations of *BAAL* or OUCT to this setting raise multiple issues: for instance, instead of randomly picking surrogate hypotheses, worst-case hypotheses have to be chosen, which is not straightforward. Furthermore, a shortcoming of the adversarial approach is that the optimality guarantee on the worst-case performance will probably result in a poorer performance on average.

On the practical side, a natural next step is to address more sophisticated function spaces, by extending *BAAL* to handle non-linear hypothesis spaces such as kernel spaces, thanks to the famed kernel trick. It must be noted that billiard-based algorithms have been investigated for kernel spaces (Herbrich et al., 2001; Ruján, 1997; Ruján and Marchand, 2000), featuring good theoretical and computational results. Ongoing work on this direction shows promising results.

Similarly, there might be ways to use the approach on other common machine learning approximators, such as neural networks or Gaussian processes. A trickier (and therefore more challenging) part would be to find a way to apply *BAAL* to regression rather than classification. Indeed, having a continuous set

of possible answers to each query produces a "game tree" in which each path is visited only once: UCT makes no sense in this case. A possibility for this would be to use another layer of progressive widening; it is however not as easy to enforce for the value space as it was for the input space.

Another question regards the possible misspecification of the model: what happens if the prior used on the hypothesis space is not accurate? It seems reasonable to believe that the loss in performance may depend smoothly on a distance between the "right" prior and the prior that was actually used, such as the Kullback-Leibler divergence—the question remains open, however.

Applying the presented approaches to real-world applications is of course a significant further work. One might think that an extension of *BAAL* to noisy cases is necessary for use on practical problems. Addressing noisy settings would indeed be a significant step forward, and is not out of reach. However, although noise often occur in numerous applications, many other problems are noise-free. This is the case for numerical engineering applications that first motivated this work: computational codes used to simulate engineering tasks are often deterministic. The fact that an algorithm cannot handle noise does not disqualify it for real-world applications. It is easy to derive classification tasks (or optimization tasks with a Bernoulli noise) from such numerical engineering codes; using *BAAL* on such tasks can then be done directly.

A larger term perspective is to reconsider how exploration is handled within UCT.

UCT has been used in many different algorithms, notably in games such as Go, Havannah or Hex. Incidentally, a difference between *BAAL*/OUCT and applications of UCT to games is that the exploration term in games tends to disappear; within games, exploration may sometimes be replaced by prior knowledge. On the contrary, the exploration term of UCB is quite necessary in active learning and expensive optimization. In games such as Go, a possible reason for the disappearance of the exploration term is that many expert heuristics to assess a move can be embedded in the algorithm: opening books, pattern recognition, etc. These heuristics overtake the need for exploration, since they generally give a not-so-bad estimation of the quality of moves (although this estimation in itself is far from enough to play well). In *BAAL* and OUCT, such expert knowledge is scarcer.

In game applications, some of the improvements that have been made to the original method yielded with impressive results. Transposing those improvements to *BAAL* and OUCT could fasten convergence to the optimal strategy. An example of a very successful improvement is rapid action-value estimation (RAVE, Gelly and Silver, 2007), a heuristic that adds a term to the UCB evaluation of a node consisting in an weighted average of the rewards obtained for all paths that perform the move leading to this node at any point in the simulation (rather than only the paths going through the exact same node). Similarly to the exploration term, its weight decreases with the number of times the node has been explored.

The use of heuristics such as RAVE provides online priors, summarizing the search results. As many perspectives of the presented work are concerned

with continuous spaces, a major goal would be to tailor RAVE-like heuristics suited to spaces of continuous nodes. A learning layer, e.g. involving "similarity kernels", might then be developed to identify and exploit the structure of the search space within the UCT exploration.

Bibliography

Naoki Abe and Hiroshi Mamitsuka. Query learning strategies using boosting and bagging. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 1–9, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8.

Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022821128753.

D. V. Arnold and H.-G. Beyer. Efficiency and mutation strength adaptation of the (mu/muI,lambda)-ES in a noisy environment. In M. Schoenauer et al., editor, *Parallel Problem Solving from Nature*, volume 1917 of *LNCS*, pages 39–48. springer, 2000.

Dirk V. Arnold and Hans-Georg Beyer. Evolution strategies with cumulative step length adaptation on the noisy parabolic ridge. *Natural Computing: an international journal*, 7(4):555–587, 2008. ISSN 1567-7818. doi: http://dx.doi.org/10.1007/s11047-006-9025-5.

Dirk V. Arnold and D. C. Scott Van Wart. Cumulative step length adaptation for evolution strategies using negative recombination weights. In *EvoWorkshops*, pages 545–554, 2008.

K.J. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10: 174–205, 1965.

Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvari. Use of variance estimation in the multi-armed bandit problem. NIPS Workshop on On-line Trading of Exploration and ExploitationWorkshop, 2006. URL http://http://hal.inria.fr/inria-00203496/en/.

J.Y. Audibert, R. Munos, and Cs. Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 2008.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002. URL http://dx.doi.org/10.1023/A:1013689704352.

Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *NIPS*, pages 89–96, 2008.

Anne Auger. Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible markov chains. *Theoretical Computer Science*, 334:35–69, 2005.

Anne Auger and Olivier Teytaud. Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, page accepted, 2009. doi: http://dx.doi.org/10.1007/s00453-008-9244-5. URL http://dx.doi.org/10.1007/s00453-008-9244-5.

Anne Auger, Steffen Finck, Nikolaus Hansen, and Raymond Ros. BBOB 2009: Comparison Tables of All Algorithms on All Noisy Functions. Technical Report RT-0384, INRIA, 04 2010. URL http://hal.inria.fr/inria-00471253/en/.

Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 65–72, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: http://doi.acm.org/10.1145/1143844.1143853.

Maria-Florina Balcan, Andrei Broder, and Tong Zhang. Margin based active learning. In *Proc. of the 20 th Conference on Learning Theory*, 2007.

Maria-Florina Balcan, Steve Hanneke, and Jennifer Wortman. The true sample complexity of active learning. In *COLT*, pages 45–56, 2008.

R. Bellman. *Dynamic Programming*. Princeton Univ. Press, 1957.

S.N. Bernstein. On a modification of chebyshev's inequality and of the error formula of laplace. *Original publication: Ann. Sci. Inst. Sav. Ukraine, Sect. Math. 1*, 3(1):38–49, 1924.

S.N. Bernstein. *The Theory of Probabilities*. Gastehizdat Publishing House, Moscow, 1946.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, September 1996. ISBN 1-886529-10-8.

Dimitri P. Bertsekas and Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, September 1999. ISBN 1886529000. URL http://http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1886290

H.-G. Beyer. *The Theory of Evolutions Strategies*. Springer, Heidelberg, 2001.

Hans-Georg Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. In *Computer Methods in Applied Mechanics and Engineering*, pages 239–267, 1998.

Alexis Bienvenue and Olivier Francois. Global convergence for evolution strategies in spherical problems: some simple proofs and difficulties. *Theor. Comput. Sci.*, 306(1-3):269–289, 2003. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(03)00284-6.

A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17:1–13, 1999. ISSN 1615-147X. URL http://dx.doi.org/10.1007/BF01197708. 10.1007/BF01197708.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787. URL http://http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0521833787.

Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration for multi-armed bandit problems. *CoRR*, abs/0802.2655, 2008.

R. Castro and R. Nowak. Minimax bounds for active learning. In *to appear in Proceedings of the 20th Annual Conference on Learning Theory (COLT)*, 2007.

R. Castro, R. Willett, and R. Nowak. Faster rates in regression via active learning. *Proceedings of Neural Information Processing Systems (NIPS)*, pages 05–3, 2005.

Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. Learning probabilistic linear-threshold classifiers via selective sampling. In *In Proc. 16th COLT*, pages 373–386. Springer, 2003.

Kathryn Chaloner. Bayesian design for estimating the turning point of a quadratic regression. *Communications in Statistics—Theory and Methods*, 18(4):1385–1400, 1989.

Edward Chang and Simon Tong. Support vector machine active learning for image retrieval. In *Proceedings of the 9 th ACM international conference on Multimedia*, pages 107–118, 2001.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006. URL http://www.kyb.tuebingen.mpg.de/ssl-book.

G. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.

Guillaume Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Monte-Carlo Strategies for Computer Go. In Pierre-Yves Schobbens, Wim Vanhoof, and Gabriel Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006. URL http://www.cs.unimaas.nl/g.chaslot/papers/mcscg.pdf.

Guillaume Chaslot, Jean-Baptiste Hoock, Fabien Teytaud, and Olivier Teytaud. On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers. In *ESANN*, Bruges Belgium, 2009. URL http://hal.inria.fr/inria-00380125/en/.

H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–509, 1952.

D.A. Cohn, Z. Ghahramani, and M.I. Jordan. Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221, 1994. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022673506211.

F. Comets, S. Popov, G. M. Schütz, and M. Vachkovskaia. Billiards in a General Domain with Random Reflections. *Archive for Rational Mechanics and Analysis*, 191:497–537, March 2009. doi: 10.1007/s00205-008-0120-x.

A. Conn, K. Scheinberg, and L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives, 1997. URL http://citeseer.ist.psu.edu/conn97recent.html.

M. Coulm, P. Rolet, O. Teytaud, and P. Vayssiere. Parameter tuning by simple regret algorithms and multiple simultaneous hypothesis testing. In *Proceedings of ICINCO'2010*, 2010.

R. Coulom. Details on the QLR algorithm for stochastic optimization, March 2010. http://remi.coulom.free.fr/QLR/.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.

Rémi Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.

Rémi Coulom, Philippe Rolet, Nataliya Sokolovska, and Olivier Teytaud. Handling expensive optimization with large noise. In *Proceedings of FOGA'2011 (to appear)*, 2011.

Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 337–344. MIT Press, Cambridge, MA, 2005.

Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 235–242. MIT Press, Cambridge, MA, 2006.

Sanjoy Dasgupta, Adam Tauman Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. In *In COLT*, pages 249–263, 2005.

Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 353–360. MIT Press, Cambridge, MA, 2008.

Brian Denton. Review of "stochastic optimization: Algorithms and applications" by stanislav uryasev and panos m. pardalos, kluwer academic publishers 2001. *Interfaces*, 33(1):100–102, 2003. ISSN 0092-2102.

L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic Theory of Pattern Recognition*. Springer, 1997.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

Ellinor Fackle Fornius. *Optimal Design of Experiments for the Quadratic Logistic Model*. PhD thesis, Department of Statistics, Stockholm University, 2008.

Ronald A. Fisher. *The design of experiments*. Oliver and Boyd, 1951.

J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Mach. Learn.*, 28(2-3): 133–168, 1997. ISSN 0885-6125.

Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273531.

Sylvain Gelly, Sylvie Ruette, and Olivier Teytaud. Comparison-based algorithms are robust and randomized algorithms are anytime. *Evolutionary Computation*, 15(4):411–434, 2007.

Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Query by committee made real. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 443–450. MIT Press, Cambridge, MA, 2006.

W. R. Gilks. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, December 1995. ISBN 0412055511.

J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41.2:148–177, 1979. URL http://www.jstor.org/stable/2985029.

Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. In *COLT '91: Proceedings of the fourth annual workshop on Computational learning theory*, pages 303–314, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1-55860-213-5.

Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 593–600, Cambridge, MA, 2008. MIT Press.

Ulrich Hammel and Thomas Bäck. Evolution strategies on noisy functions: How to improve convergence properties. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving From Nature*, volume 866 of *LNCS*, pages 159–168, Jerusalem, 1994. springer.

David J Hand. Evaluating diagnostic tests: The area under the roc curve and the balance of errors. *Statistics in Medicine*, 29:1502–1510, 2010. URL http://http://dx.doi.org/10.1002/sim.3859.

Steve Hanneke. A bound on the label complexity of agnostic active learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 353–360, New York, NY, USA, 2007a. ACM. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273541.

Steve Hanneke. Teaching dimension and the complexity of active learning. In *Proceedings of the 20th Annual Conference on Learning Theory (COLT)*, 2007b.

N. Hansen. Source code for uh-cma, June 2008. Version 3, http://www.lri.fr/ hansen/cmaesintro.html.

N. Hansen. Personal communication, 2010.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 11(1), 2003.

Nikolaus Hansen, Andre Niederberger, Lino Guzzella, and Petros Koumoutsakos. A Method for Handling Uncertainty in Evolutionary Optimization with an Application to Feedback Control of Combustion. *IEEE Transactions on Evolutionary Computation*, 2009. URL http://hal.inria.fr/inria-00276216/en/.

David Haussler, Michael Kearns, and Robert E. Schapire. Bounds on the sample complexity of bayesian learning using information theory and the vc dimension. *Mach. Learn.*, 14(1):83–113, 1994. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022698821832.

Tibor Hegedüs. Generalized teaching dimensions and the query complexity of learning. In *COLT '95: Proceedings of the eighth annual conference on Computational learning theory*, pages 108–117, New York, NY, USA, 1995. ACM. ISBN 0-89791-723-5. doi: http://doi.acm.org/10.1145/225298.225311.

Verena Heidrich-Meisner and Christian Igel. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553426.

R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, 2001.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Probability Inequalities for Sums of Bounded Random Variables*, 58 (301):13–30, March 1963.

Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 417–424, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: http://doi.acm.org/10.1145/1143844.1143897.

Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Semisupervised svm batch mode active learning with applications to image retrieval. *ACM Trans. Inf. Syst.*, 27(3):1–29, 2009. ISSN 1046-8188. doi: http://doi.acm.org/10.1145/1508850.1508854.

C. Igel. Recent results on no-free-lunch for optimization. In H. Beyer, T. Jansen, C. Reeves, and M. D. Vose, editors, *Theory of Evolutionary Algorithms*, number 04081 in Dagstuhl Seminar Proceedings, Abstract Collection. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, 2004.

V. S. Iyengar, C. Apte, and T. Zhang. Active learning using adaptive resampling. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–98, 2000.

Mohamed Jebalia and Anne Auger. On multiplicative noise models for stochastic search. In *Parallel Problem Solving From Nature*, Dortmund, Germany, 2008. URL http://hal.inria.fr/inria-00287725/en/.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, 1998. ISSN 0925-5001. doi: http://dx.doi.org/10.1023/A:1008306431147.

Matti Kaariainen. Active learning in the non-realizable case. In *Algorithmic Learning Theory, 17th International Conference, ALT 2006, Barcelona, Spain, October 2006, Proceedings*, volume 4264 of *Lecture Notes in Artificial Intelligence*, pages 63–77. Springer, October 2006.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101 (1-2):99–134, 1998.

P. Kall. *Stochastic Linear Programming*. Springer, Berlin, 1976.

André I. Khuri, Bhramar Mukherjee, Bikas K. Sinha, and Malay Ghosh. Design issues for generalized linear models: A review. *Statistical Science*, 21(3):376–399, 2006.

L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. *ECML'06*, 2006. URL http://zaphod.aml.sztaki.hu/papers/ecml06.pdf.

S. R. Kulkarni, S. K. Mitter, and J. N. Tsitsiklis. Active learning using arbitrary binary valued queries. *Mach. Learn.*, 11(1):23–35, 1993. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022627018023.

T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6:4–22, 1985.

C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, page accepted, 2009.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. Using markov decision process for learning dialogue strategies. In *Proc. ICASSP*, pages 201–204, 1998.

M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, 54:125–152, 2004.

David J. C. Mackay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

K. Marti. *Stochastic Optimization Methods*. Springer, 2005.

T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. Empirical bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: http://doi.acm.org/10.1145/1390156.1390241.

Netflix. The Netflix Prize, 2006. URL http://en.wikipedia.org/wiki/Netflix_Prize.

Hammond Niederreiter. Low-discrepancy and low-dispersion sequences. *J. Number Theory*, 1988.

I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

Philippe Rolet. Java source code of the baal algorithm, 2010. URL http://www.lri.fr/řolet/misc/baal.tgz.

Philippe Rolet and Olivier Teytaud. Complexity bounds for batch active learning in classification. In *ECML PKDD '10: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, Berlin, Heidelberg, 2010a. Springer-Verlag.

Philippe Rolet and Olivier Teytaud. Adaptive noisy optimization. In *EvoSTAR 2010 Proceedings*, volume EvoSTAR, 2010b. To appear.

Philippe Rolet and Olivier Teytaud. Bandit-based Estimation of Distribution Algorithms for Noisy Optimization: Rigorous Runtime Analysis. In *Lion4*, Venice, Italy, 2010c. URL http://hal.inria.fr/inria-00437140/en/.

Philippe Rolet, Michèle Sebag, and Olivier Teytaud. Boosting active learning to optimality: A tractable monte-carlo, billiard-based algorithm. In *ECML PKDD '09: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 302–317, Berlin, Heidelberg, 2009a. Springer-Verlag. ISBN 978-3-642-04173-0. doi: http://dx.doi.org/10.1007/978-3-642-04174-7_20.

Philippe Rolet, Michèle Sebag, and Olivier Teytaud. Optimal robust expensive optimization is tractable. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1951–1956, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-325-9. doi: http://doi.acm.org/10.1145/1569901.1570255.

Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proc. 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, San Francisco, CA, 2001. URL http://citeseer.ist.psu.edu/roy01toward.html.

Pál Ruján and Mario Marchand. Computing the bayes kernel classifier. In A. J. Smola, P. L. Bartlett, B. Schoelkopf, and D. E. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 329–347, Cambridge, MA, USA, 2000. MIT Press.

Paul Ruján. Playing billiards in version space. *Neural Computation*, 9(1):99–122, January 1997. URL http://arxiv.org/pdf/cond-mat/9508130.

Norbert Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13 (1):145–147, 1972.

Andrew I. Schein and Lyle H. Ungar. Active learning for logistic regression: an evaluation. *Mach. Learn.*, 68(3):235–265, 2007. ISSN 0885-6125. doi: http://dx.doi.org/10.1007/s10994-007-5019-5.

G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. *Proceedings of the Seventeenth International Conference on Machine Learning*, 282:285–286, 2000.

H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New-York, 1981. 1995 – 2^{nd} edition.

J. K. Sengupta. *Stochastic Programming. Methods and Applications*. North-Holland, Amsterdam, 1972.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: http://doi.acm.org/10.1145/130385.130417.

Masashi Sugiyama and Neil Rubens. A batch ensemble approach to active learning with model selection. *Neural Netw.*, 21(9):1278–1286, 2008. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2008.06.004.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: an Introduction*. MIT Press, Cambridge, MA, 1998.

Gerald Tesauro. Practical issues in temporal difference learning. In *Machine Learning*, pages 257–277, 1992.

Olivier Teytaud and Anne Auger. On the adaptation of the noise level for stochastic optimization. In *IEEE Congress on Evolutionary Computation*, Singapour, 2007. URL http://hal.inria.fr/inria-00173224/en/.

Olivier Teytaud, Sylvain Gelly, and Jérémie Mary. Active learning in regression, with application to stochastic dynamic programming. In *ICINCO and CAP*, 2007. URL http://www.grappa.univ-lille3.fr/ mary/paper/ldsfordp.pdf.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, pages 42–66, 2001.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1968.1972.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995. URL http://books.google.com/books?id=sna9BaxVbj8C&printsec=frontcover.

Emmanuel Vazquez, Julien Villemonteix, Maryan Sidorkiewicz, and Eric Walter. Global optimization based on noisy evaluations: an empirical study of two statistical approaches. *Journal of Global Optimization*, page 17 pages, 2008. doi: 10.1007/s10898-008-9313-y. URL http://hal-supelec.archives-ouvertes.fr/hal-00354656/en/.

M. Vidyasagar. *A Theory of Learning and Generalization*. Springer-Verlag, New York, New York, 1997.

Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, page 26 pages, 09 2008. doi: 10.1007/s10898-008-9354-2. URL http://hal-supelec.archives-ouvertes.fr/hal-00354262/en/.

Yizao Wang and Sylvain Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.

Yizao Wang, Jean-Yves Audibert, and Remi Munos. Algorithms for infinitely many-armed bandits. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1729–1736. Curran Associates, Inc., 2009.

Manfred K. Warmuth, Jun Liao, Gunnar Rätsch, Michael Mathieson, Santosh Putta, and Christian Lemmen. Support vector machines for active learning in the drug discovery process. *Journal of Chemical Information Sciences*, 43: 667–673, 2003.

C. Williams and C. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*, volume 8, pages 514–520, 1996. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.8841.

G. Xiao, F. Southey, R. C. Holte, and D. Wilkinson. Software testing by active learning for commercial games. In *In AAAI-05: Twentieth National Conference in Artificial Intelligence*, pages 609–616, 2005.

Zongzhao Zhou. Hierarchical surrogate-assisted evolutionary optimization framework. In *In Evolutionary Computation, 2004. CEC2004. Congress on*, pages 1586–1593, 2004.