SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of

Computer Science, Systems and Communication

PhD program in: Computer Science Cycle XXXIII

Automated Deep Learning through Constrained Bayesian Optimization

Surname: Perego Name: Riccardo

Registration number: 748503

Tutor: Prof. Giuseppe Vizzari

Supervisor: Dr. Antonio Candelieri

Coordinator: Prof. Leonardo Mariani

ACADEMIC YEAR 2019/2020

I would like to express my deepest appreciation to Antonio Candelieri, who has guided me in these years, starting from my master thesis up to now.

I am also grateful to the Professor Francesco Archetti for sharing his research experience with me during these years of collaboration.

I would also like to extend my gratitude to my PhD mates, in particular, Davide Ginelli and Vincenzo Cutrona.

Finally, I would like to thank my family for the steady and careful support that they gave me throughout my life in all the endeavours that I chose to pursue

Abstract

In an increasingly technological and interconnected world, the amount of data is continuously growing, and as a consequence, decision-making algorithms are also continually evolving to adapt to it. One of the major sources of this vast amount of data is the Internet of Things, in which billions of sensors exchange information over the network to perform various types of activities such as industrial and medical monitoring. In recent years, technological development has made it possible to define new high-performance hardware architectures for sensors, called Microcontrollers, which enabled the creation of a new kind of decentralized computing named Edge Computing. This new computing paradigm allowed sensors to run decision-making algorithm at the edge in order to take immediate and local decisions instead of transferring the data on central server processing. To support Edge Computing, the research community started developing new advanced techniques to efficiently manage the limited resources on these devices for applying the most advanced Machine Learning models, especially the Deep Neural Networks.

Automated Machine Learning is a branch of the Machine Learning field aimed at disclosing the power of Machine Learning to non-experts as well as efficiently supporting data scientists in designing their own data analysis pipelines. The adoption of Automated Machine Learning has made it possible to develop increasingly high-performance models almost automatically. However, with the advent of Edge Computing, a specialization of Machine Learning, defined as Tiny Machine Learning (Tiny ML), has been arising, that is, the application of Machine Learning algorithms on devices having limited hardware resources.

This thesis mainly addresses the applicability of Automated Machine Learning to generate accurate models which must be also *deployable* on tiny devices, specifically Microcontroller Units. More specifically, the proposed approach is aimed at maximizing the performances of Deep Neural Networks while satisfying the constraints associated to the limited hardware resources, including batteries, of Microcontrollers. Thanks to a close collaboration with STMicroelectronics, a leading company for design, production and sale of microcontrollers, it was possible to develop a novel Automated Machine Learning framework that deals with the black-box constraints related to the deployability of a Deep Neural Network on these tiny devices, widely adopted in IoT applications.

The application on two real-life use cases provided by STMicroelectronics (i.e., Human Activity Recognition and Image Recognition) proved that the novel proposed approach can efficiently find out configurations for accurate and deployable Deep Neural Networks, increasing their accuracy against baseline models while drastically reducing hardware required to run them on a microcontroller (i.e., a reduction of more than 90%). The approach was also compared against one of the state-of-the-art AutoML solutions in order to evaluate its capability to overcome the issues which currently limit the wide application of AutoML in the tiny ML field.
Finally, this PhD thesis suggests interesting and challenging research directions to further increase the applicability of the proposed approach by integrating recent and innovative research results (e.g., weakly defined search spaces, Meta-Learning, Multi-objective and Multi-Information Source optimization).

Publications

Journal

[j1] **Perego, R.**, Candelieri A., Archetti F., & Pau D. *AutoTinyML for microcontrollers: dealing with black-box deployability.* Expert Systems With Applications (2021). (Under review)

[j2] Candelieri A. **Perego R.**, & Archetti F. *Green Machine Learning via Augmented Gaussian Processes and Multi-Information SourceOptimization*, Soft Computing, Special Issue on Optimization and Machine Learning (2020). (Ahead of printing)

[j3] Candelieri, A., **Perego, R.**, Giordani, I., Ponti, A., & Archetti, F. *Modelling human active search in optimizing black-box functions.* Soft Computing, 24, 17771–17785 (2020). https://doi.org/10.1007/s00500-020-05398-2

[j4] Sergeyev, Y. D., Candelieri, A., Kvasov, D. E., & **Perego, R.** *Safe global optimization of expensive noisy black-box functions in the* δ*-Lipschitz framework.* Soft Computing, 1-21, 17715–17735 (2020). https://doi.org/10.1007/s00500-020-05030-3

[j5] Galuzzi, B. G., **Perego, R.**, Candelieri, A., & Archetti, F. *Bayesian optimization for full waveform inversion.* In New Trends in Emerging Complex Real Life Problems (pp. 257-264) (2018). Springer, Cham. https://doi.org/10.1007/978-3-030-00473-6_28

[j6] Candelieri, A., **Perego, R.**, & Archetti, F. *Bayesian optimization of pump operations in water distribution systems.* Journal of Global Optimization, 71(1), 213-235 (2018). https://doi.org/10.1007/s10898-018-0641-2

Conferences

[c1] **Perego, R.**, Candelieri, A., Archetti, F., & Pau, D. (2020, September). *Tuning Deep Neural Network's Hyperparameters Constrained to Deployability on Tiny Systems.* In International Conference on Artificial Neural Networks (pp. 92-103). Springer, Cham.

[c2] Candelieri A., **Perego R.**, Giordani I., & Archetti F., *Composition of kernel and acquisition functions for High Dimensional Bayesian Optimization.* Learning and Intelligent Optimization conference 14 (LION)(2020).

[c3] Candelieri, A., & **Perego, R.** (2019). *Dimensionality Reduction methods to scale Bayesian Optimization up.* Numerical Computations: Theory and Algorithms (NUMTA), 167.

[c4] Candelieri, A., **Perego, R.**, & Archetti F. *Global Optimization of a Machine Learning based Forecasting Pipeline.* Data Science & Social Research (DSSR) (2019), Milan, Italy.

[c5] Candelieri, A., Galuzzi, B.G., Giordani, I., **Perego, R.**, & Archetti, F. *Optimizing partially defined black-box functions under unknown constraints via Sequential Model Based Optimization: an application to Pump Scheduling Optimization in Water Distribution Networks.* Learning and Intelligent Optimization conference 13 (LION)(2019), Chania, Crete, Greece.

[c6] Tsai, Y. A., Pedrielli, G., Mathesen, L., Zabinsky, Z. B., Huang, H., Candelieri, A., & **Perego, R.** (2018, December). *Stochastic optimization for feasibility determination: an application to water pump operation in water distribution network.* In 2018 Winter Simulation Conference (WSC) (pp. 1945-1956). IEEE.

[c7] Galuzzi, B., **Perego, R.**, Candelieri, A. & Archetti, F. *Bayesian Optimization for Full Waveform Inversion.* International Conference On Optimization and Decision Making (ODS) (2018), Taormina, Italy.

[c8] Candelieri, A., **Perego, R.**, & Archetti, F. (2018, June). *Intelligent pump scheduling optimization in water distribution networks.* In International Conference on Learning and Intelligent Optimization (pp. 352-369). Springer, Cham.

Contents

List of Figures

List of Tables

Abbreviations

ML	**M**achine **L**earning
Tiny ML	**T**iny **M**achine **L**earning
AutoML	**Auto**mated **M**achine **L**earning
AutoDL	**Auto**mated **D**eep **L**earning
HPO	**H**yper**P**arameters **O**ptimization
CASH	**C**ombined **A**lgorithm **S**election and **H**yperparameters optimization
NAS	**N**eural **A**rchitecture **S**earch
GO	**G**lobal **O**ptimization
CGO	**C**onstrained **G**lobal **O**ptimization
BO	**B**ayesian **O**ptimization

Chapter 1

Introduction

In an increasingly digitized and connected society, which produces perhaps as much data every two days as has been generated since the beginning of civilization, Machine Learning techniques and algorithms are playing an increasingly central role. These innovative data-processing algorithms are now being used in various application domains, facilitated by the increased amount of data available.

Given a generic dataset, the model's performance depends on selecting the best algorithm (**Model Selection**) and the optimization of its hyperparameters (**Hyperparameter Optimization**). However, identifying the optimal model is a complicated and costly search process, both in terms of computational resources and time. The field that deals with this kind of optimization process is called **Automated Machine Learning** (**AutoML**).

Today we have the technological ability to infuse some knowledge of the world into every object simply by adding sensors to it, providing it with memory, and connecting it to human social networks. The Internet of Things (IoT), understood as the set of technologies that allow physical objects and devices to generate data and share them with other objects, represents one of the most relevant innovations of the last years. The margins for expansion are wide, and the trend in global spending on this technology is clear evidence of this. Investments in projects incorporating IoT technology have increased by more than 80% from 2015 to 2019 generating a market value of more than $250 billion, and from a recent Fortune Business Insights[1] study it is estimated that the market will reach more than $1.463 billion in 2027. The steady growth process has led to a rapid expansion in the number of smart devices connected to the Cloud. The application opportunities of *smart devices* are rapidly multiplying and differentiating in different sectors, especially in business, industry, and healthcare.

The typical architecture used is Cloud Computing, in which the vast amount of information collected locally by different devices is then transferred to a central server for processing. However,

[1] `https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307`, [Accessed on: 20/10/2020]

the spread of IoT devices leads to an evolution of the internet from the virtual to the physical world.

Smart devices generate a huge amount of data in the periphery of the network and often require the information to be processed in 'near real-time' or even in real time. Consequently, it is easy to see how it is unrealistic to think that each of these devices, and applications in the field, can always rely on a stable channel of connection and constant data communication with the cloud. This progressive decentralization has led companies in the sector to devise new solutions for the IoT network architecture to respond to the new needs and ways of managing data locally.

The main decentralized alternative to Cloud Computing is Edge Computing. Some experts even predict that it will soon spell the end of Cloud computing. Others believe it will complement cloud computing by merely managing users' instantaneous data, while the central cloud will still handle Big Data processing[2]. According to Gartner [3], a leading IT research and advisory company, only 10% of the data generated is currently processed outside of centralized data centers, but by 2025 this percentage will reach 74%. A key role in IoT technology, especially in Edge Computing, is played by microcontrollers (MCUs). Microcontrollers are, to all intents and purposes, self-contained computers of extremely small dimensions, so much so that they are called computers on a single chip.

Equipped with a Central Processing Unit (CPU), Random Access Memory (RAM), and Read-Only Memory (ROM), they are designed to perform specific tasks. They can be integrated into almost any device: from industrial equipment to stock items to wearables, household appliances, and much more. There are now many different types with a wide range of functionalities designed for different use cases.

Despite the pandemic situation, which started in March 2020, the MCU shipments are expected to increase by more than 6% in 2021 to 24.9 billion units, followed by increases of 8% in 2022 and 10% in 2023, when worldwide MCU deliveries are projected to reach a new record-high level of 29.6 billion [4].

If from a hardware point of view, Edge Computing systems are effectively supported by MCUs, instead from a software point of view, they require decision algorithms that are able to analyze data instantaneously and at the same time adapt to the hardware limitations of the microcontroller used. The field of constrained Automated Machine Learning deals with these aspects. Although Machine Learning algorithms' main goal is to have maximum accuracy, there are often other factors that need to be taken into account, for example, training and inference time, model size, memory consumption, and admissibility. These factors become conditions to be included during the optimization process when it is no longer sufficient to find the maximum or minimum value of a performance metric of Machine Learning task. Still, it is necessary to

[2] `https://www.zerounoweb.it/techtarget/searchdatacenter/edge-computing-cose-come-implementarlo/` , [Accessed on: 18/10/2020]

[3] `https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders`, [Accessed on: 20/10/2020]

[4] `https://www.icinsights.com/data/articles/documents/1289.pdf`, [Accessed on: 28/12/2020]

transform the process into a problem subject to constraints, such as the hardware limitations of a microcontroller.

1.1 Thesis organization

The main problem addressed in this thesis is finding out, automatically, an optimal ML model to perform a classification or regression task on a given dataset under known or unknown constraints. In particular the context of this thesis pertains to the application of an AutoML approach to solve **Tiny Machine Learning (TinyML)** tasks, where hardware constraints (e.g., memory and energy consumption) are taken into account during the decision process to select the optimal model.

This work has been carried out thanks to the collaboration with STMicroelectronics, a leading company for the production and sale of Micro-Controllers Units (MCUs), which has made it possible to develop a system that, starting from a given sketched Deep Neural Network, a dataset, and a target MCU, it is able to find out networks that satisfy the limited hardware constraints of the MCU while optimizing the performance metric.

The thesis is organized as follows. Chapter 2 will formally introduce the research field of the thesis regarding the problem of Automated Machine Learning by listing current state-of-the-art solutions both open source and web services.

Then Chapter 3 will give an overview of most of the **Global Optimization** paradigms for searching the model with the highest generalization. Chapter 4 will introduce in detail the **Bayesian Optimization (BO)** framework, one of the most used in Machine Learning community, which was the object of much of the PhD research leading to the publications of several works ([j3],[j5],[j6],[c4],[c5],[c6],[c7],[c8]). Chapter 5 will address the limitations of BO framework in the development of **AutoML**, **AutoDL** and **NAS** applications and the current open research challenges to overcome these limitations. Such challenges were addressed during the PhD, and as a result several papers were submitted and published ([j2],[j4],[c2],[c3]). Finally, Chapter 6 presents the **AutoTinyML** approach proposed as a major contribution of this thesis ([j1,c1]), while Chapters 7 and 8 the results of several experiments on the proposed approach are reported, also on two real-use cases provided by STMicroelectronics. In the conclusion of the thesis, in Chapter 9, several considerations will be made regarding the potential of the proposed system, its limitations and possible future developments. In addition, in Figure 1.1, a graphical representation of the PhD thesis outline is reported, pointing out which are the relevant works submitted and published during the PhD years.

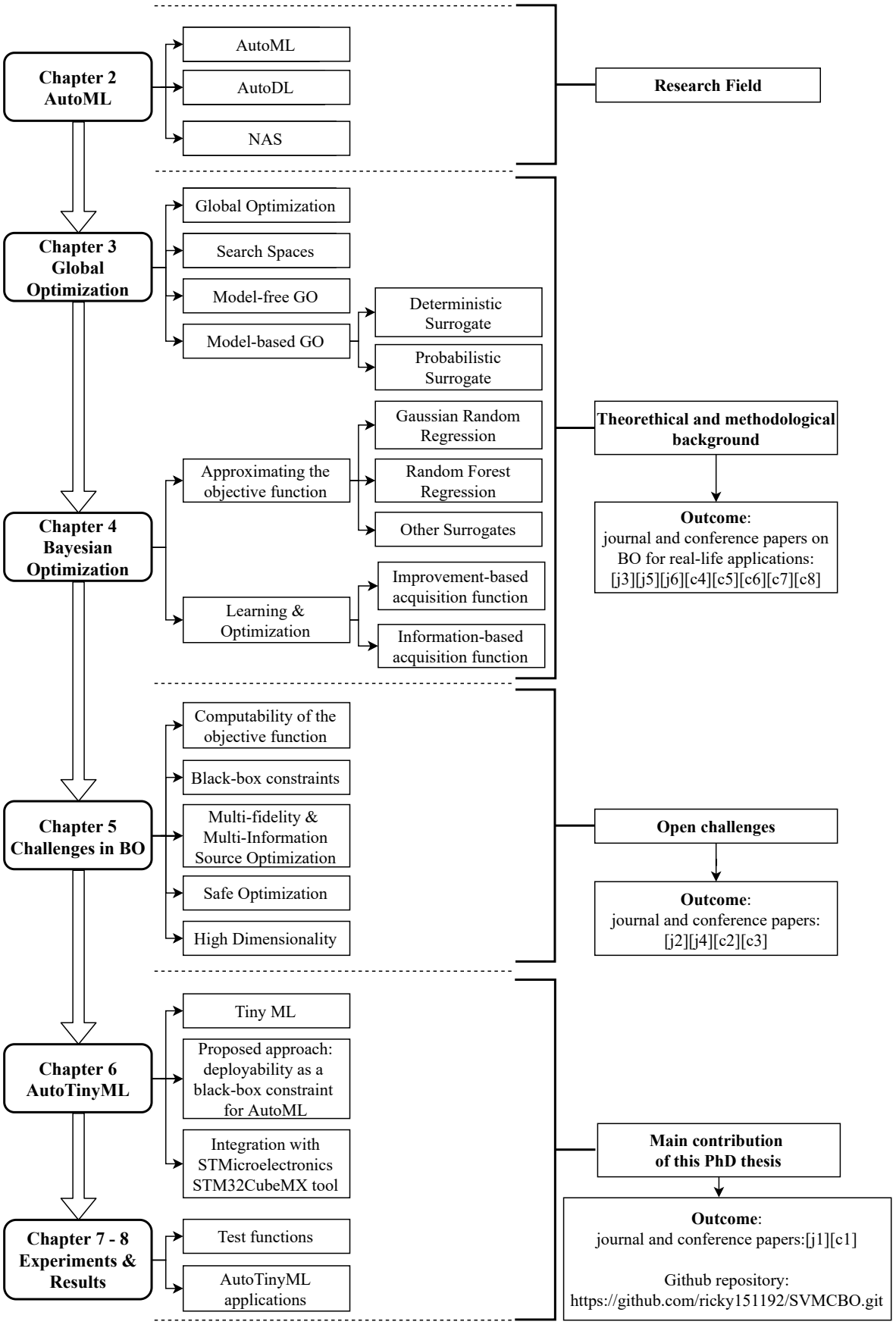


FIGURE 1.1: Thesis outline to highlight the search field, the open research challenges and the contribution of PhD research

1.2 Contributions on publications

Journal

[**j1**] **Perego, R.**, Candelieri A., Archetti F., & Pau D. *AutoTinyML for microcontrollers: dealing with black-box deployability.* Expert Systems With Applications (2021). (Under review)

This publication proposes a novel optimization approach more suitable for TinyML tasks for dealing with black-box and coupled constraints. Different from state-of-the-art approaches that propose multi-objectives optimization, which can be more complex to solve and require analytical representations of constraints, this proposed approach is more data efficient and more suitable for problems with constraints of different nature (e.g., crash constraints – out-of-memory during the training model).
I designed and performed the experiments implementing the AutoTinyML framework, analyzed and performed statistical tests, and contributed on writing several sections of the paper.

[**j2**] Candelieri A. **Perego R.**, & Archetti F. *Green Machine Learning via Augmented Gaussian Processes and Multi-Information SourceOptimization*, Soft Computing, Special Issue on Optimization and Machine Learning (2020). (Ahead of printing)

This publication proposes a novel optimization approach based on Multi-Source/Fidelity approach. The novel aspect is that this approach is able to avoid variance starvation and premature convergence to local optima during the optimization process exploiting the balance among the several "sources" available during the optimization.
I collaborated to designed and performed the experiments, analyzed and interpreted data and contributed to numeric and charts results.

[**j3**] Candelieri, A., **Perego, R.**, Giordani, I., Ponti, A., & Archetti, F. *Modelling human active search in optimizing black-box functions.* Soft Computing, 24, 17771–17785 (2020).
https://doi.org/10.1007/s00500-020-05398-2

Based on the question "how the humans optimize a black-box problem" this paper proposes an analysis of the decision process of humans to perform an optimization of a black-box problem. In particular, this paper shows how the Bayesian Optimization approach can be seen as the closest optimization strategy to human thinking in respect to other strategies as Genetic Algorithm, Particle Swarm Optimization, Direct, and others.
I designed the experiments, performed and interpreted statistical results, contributed to several sections including the numeric and charts results.

[**j4**] Sergeyev, Y. D., Candelieri, A., Kvasov, D. E., & **Perego, R.** *Safe global optimization of expensive noisy black-box functions in the* δ*-Lipschitz framework.* Soft Computing, 1-21, 17715–17735 (2020). https://doi.org/10.1007/s00500-020-05030-3

This publication proposes a novel black-box safe optimization strategy named "δ-Lipschitz framework". In the current state of the art, the "safe" optimization aspect of a noisy function has been neglected. This paper tries to propose an optimization strategy in the presence of noisy objective functions. The main goal of this work was to formalize and test a novel optimization algorithm to "safe" optimize a noisy black-box problem. The notion "safe" means that the objective function $f(x)$ during the optimization process should not violate a "safety" threshold, for instance, a certain a priori given value in a maximization problem.
I designed and performed the experiments on a suite of test functions, analyzed and interpreted the results, contributed to discuss the numeric and charts results.

[**j5**] Galuzzi, B. G., **Perego, R.**, Candelieri, A., & Archetti, F. *Bayesian optimization for full waveform inversion.* In New Trends in Emerging Complex Real Life Problems (pp. 257-264) (2018). Springer, Cham. https://doi.org/10.1007/978-3-030-00473-6_28

This publication proposes the application of Bayesian Optimization strategy, instead of metaheuristics widely adopted in literature, to optimize a Full Waveform Inversion problem, which is a computational method to estimate the physical features of Earth subsurface. The application of Bayesian Optimization proves to better support the FWI problem.
I collaborated with Galuzzi to design and perform the experiments related to applying the Bayesian Optimization approach, analyzing and interpreting the results. I contributed particularly to the Bayesian Optimization section and the numeric results section.

[**j6**] Candelieri, A., **Perego, R.**, & Archetti, F. *Bayesian optimization of pump operations in water distribution systems.* Journal of Global Optimization, 71(1), 213-235 (2018). https://doi.org/10.1007/s10898-018-0641-2

This publication proposes to adopt the Bayesian Optimization strategy to a simulation optimization problem in Water Distribution Network. The paper proves that this model-based optimization approach can be a valid replacement of the metaheuristic algorithms currently adopted in literature for simulation optimization.
I collaborated with Candelieri to design and perform the experiments adopting a hydraulic simulator, analyzing and interpreting the results. I contributed to the results section of the paper with numeric and charts of the results.

Conferences

[**c1**] **Perego, R.**, Candelieri, A., Archetti, F., & Pau, D. (2020, September). *Tuning Deep Neural Network's Hyperparameters Constrained to Deployability on Tiny Systems.* In International Conference on Artificial Neural Networks (pp. 92-103). Springer, Cham.

This work proposes a novel optimization approach more suitable for TinyML tasks for dealing with black-box and coupled constraints. In particular, the work focuses on comparing manual tuning of a Convolutional Deep Neural Network (CNN) in a real-life application in respect to adopting an automatic tuning system that choices CNN based on black-box constraints (e.g., the hardware capacity embedded system).
I designed and performed the experiments, analyzed and interpreted data and results and I was the speaker at the conference.

[**c2**] Candelieri A., **Perego R.**, Giordani I., & Archetti F., *Composition of kernel and acquisition functions for High Dimensional Bayesian Optimization.* Learning and Intelligent Optimization conference 14 (LION)(2020).

This work proposes a model-based optimization approach that leverages the additivity – aka separability – of the objective function into mapping both the kernel and the acquisition function of the Bayesian Optimization in lower-dimensional subspaces. In respect to traditional Bayesian Optimization, this approach makes more efficient both the learning/updating of the probabilistic surrogate model and the optimization of the acquisition function.
I collaborated with Candelieri to design and perform the experiments, and analyze the results.

[**c3**] Candelieri, A., & **Perego, R.** (2019). *Dimensionality Reduction methods to scale Bayesian Optimization up.* Numerical Computations: Theory and Algorithms (NUMTA), 167.

This work proposes to exploit the dimensionality reduction intrinsically offered by Deep autoencoders and then perform BO in the induced latent space. The proposed approach shows to make the whole optimization process more efficient, reducing the effort to learn and update the probabilistic surrogate model.
Under the supervision of Candelieri, I designed and performed the experiments on the implemented Machine Learning pipeline, analyzing and interpreting the results, and I was the speaker at the conference.

[**c4**] Candelieri, A., **Perego, R.**, & Archetti F. *Global Optimization of a Machine Learning based Forecasting Pipeline.* Data Science & Social Research (DSSR) (2019), Milan, Italy.

This work presents a Bayesian Optimization framework for the optimal design of a forecasting pipeline based on time series clustering and Artificial Neural Networks. Random Forest has been adopted as probabilistic surrogate model, due to the nature of decision variables (e.g., conditional and discrete hyperparameters) in the pipeline design.
Under the supervision of Candelieri, I designed and performed the experiments adopting Reinforcement and Model-based optimization techniques for water demand forecasting, analyzed and interpreted the results and I was the speaker at the conference.

[**c5**] Candelieri, A., Galuzzi, B.G., Giordani, I., **Perego, R.**, & Archetti, F. *Optimizing partially defined black-box functions under unknown constraints via Sequential Model Based Optimization: an application to Pump Scheduling Optimization in Water Distribution Networks.* Learning and Intelligent Optimization conference 13 (LION)(2019), Chania, Crete, Greece.

This work proposes a Sequential Model-based Optimization framework for solving optimization problems characterized by a black-box, multi-extremal, expensive, and partially defined objective function under unknown constraints. This is a typical setting for simulation-optimization problems, where the objective function cannot be computed for some configurations of the decision/control variables due to the violation of some (unknown) constraint. The relevant difference with traditional Bayesian Optimization is that the optimization process is performed on the estimated feasibility region, instead of the entire search space, reducing the time and computational resources to learning and updating the surrogate model.
I collaborated to adapt a novel constrained Model-based optimization approach for the optimization of Water Distribution Network, and I designed and performed the experiments, analyzed and interpreted the results.

[**c6**] Tsai, Y. A., Pedrielli, G., Mathesen, L., Zabinsky, Z. B., Huang, H., Candelieri, A., & **Perego, R.** (2018, December). *Stochastic optimization for feasibility determination: an application to water pump operation in water distribution network.* In 2018 Winter Simulation Conference (WSC) (pp. 1945-1956). IEEE.

This work addresses the problem of analyzing the effect of the Variable Speed Pumps (VSP) regulation on the pressure distribution of a Water Distribution Systems (WDN), which is highly correlated to leakages and energy costs in WDN. This paper proposes a new stochastic partitioning algorithm based on Probabilistic Branch and Bound to optimize in a more feasibility-aware manner during the optimization process.
I mainly collaborated with Tsai to implement the approach based on interactions with a hydraulic simulator. I also supported to analyze and interpret the results and I gave my contribution on section mainly related to the definition of the optimization problem.

[**c7**] Galuzzi, B., **Perego, R.**, Candelieri, A. & Archetti, F. *Bayesian Optimization for Full Waveform Inversion.* International Conference On Optimization and Decision Making (ODS) (2018), Taormina, Italy.

This work proposes a comparison between the application of Bayesian Optimization strategy and Simulated Annealing algorithm, widely adopted in literature, to optimize a Full Waveform Inversion problem, a computational method to estimate Earth's physical features subsurface. The Bayesian Optimization application proves to better support the FWI problem producing a more precise starting approximate FWI model as starting point for local search algorithms (e.g., Gradient Descent algorithm).
I supported to implement, design and perform the experiments. I also analyzed and interpreted the results.

[**c8**] Candelieri, A., **Perego, R.**, & Archetti, F. (2018, June). *Intelligent pump scheduling optimization in water distribution networks.* In International Conference on Learning and Intelligent Optimization (pp. 352-369). Springer, Cham.

This work proposes a sequential optimization method based on Approximate Dynamic Programming in order to find a control policy defined as a mapping from states of the system to actions, i.e. pump settings in a Water Distribution Network. The novelty of the proposed approach is that it provides a policy, a strategy to decide how to act from time step to time step according to the physical system's observation instead of optimizing the problem at any time step with Bayesian Optimization and metaheuristics approaches.
I collaborated with Candelieri to design and perform the experiments, analyze and interpret the results, and I contributed to several sections among the discussion of numeric and charts of the results.

Chapter 2

Automated Machine Learning

2.1 AutoML

Automated Machine Learning (AutoML) is the process of automating iterative activities for the development of time-consuming Machine Learning models. It enables data scientists, analysts and developers to create Machine Learning models with high scalability, efficiency and productivity, while ensuring model quality.
The traditional development of Machine Learning models requires a high use of resources, in-depth domain knowledge and a considerable amount of time to produce and compare numerous models. With AutoML, it is possible to speed up the time it takes to get machine learning models ready for production with great ease and efficiency.
In fact, each ML algorithm has capabilities for class/label discrimination (for a classification task) and the ability to estimate values (for a regression task) that can vary significantly based on the data used and how they are configured to solve the target problem.
This Section will introduce the aspect of how to decide which is the best ML algorithm to adopt for a specific target problem, and how to configure the ML algorithm selected in order to provide the optimal solution with the highest performance in terms of resolution of the problem.

2.1.1 Model Selection

The first aspect to consider for implementing AutoML system is the choice of the optimal ML algorithm to solve a given target problem, which is known as *Model Selection*.
A learning algorithm A maps training data points (aka instances) $x_1, \ldots, x_n$ to their corresponding "targets" $y_1, \ldots, y_n$, where y_i is continuous in the case of regression or a "label" (categorical value) in the case of classification. All the pairs (x_i, y_i) are organized into a dataset $D = \{(x_i, y_i)\}_{1:n}$.

The Model Selection problem is formulated as follows:

$$A^* \in \arg\min_{A \in \mathcal{A}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A, D_{train}^{(i)}, D_{valid}^{(i)}) \tag{2.1}$$

Where $\mathcal{L}(A, D_{train}^{(i)}, D_{valid}^{(i)})$ is the loss achieved by A when trained on $D_{train}^{(i)}$ and evaluated on $D_{valid}^{(i)}$. The set A contains all the available ML algorithms, $\mathcal{A} = \{A^{(1)}, \ldots, A^{(h)}\}$.
Moreover, $\mathcal{L}$ can be anyone of the possible loss functions, depending on the specific learning task addressed by A. For instance, in the case of a classification task, $\mathcal{L}$ might be the Binary Cross-Entropy or the Categorical Cross-Entropy (aka Hinge Loss), for binary and multi-class classification, respectively. When a regression task is considered, $\mathcal{L}$ might be, for instance, the (Root) Mean Squared Error (RMSE), the Mean Absolute Error (MAE) or the Mean Percentage Absolute Error (MAPE).
To evaluate in a more realistic way the performance of a selected algorithm $A^{(h)}$ is adopted the k-fold cross validation which splits the training data into k equal-sized validation folds, $D_{valid}^{(1)}, \ldots, D_{valid}^{(k)}$ and associated training sets $D_{train}^{(1)}, \ldots, D_{train}^{(k)}$, where $D = D_{train}^{(i)} \bigcup D_{valid}^{(i)}, \forall i = 1, \ldots, k$.
With this statistical procedure can be checked the ability of algorithm A to predict new data that have not been used during the training process, for each validation fold k, and to give an idea of how the algorithm $A^{(h)}$ will generalize to an independent and unseen dataset.

While Model Selection is aimed at selecting the optimal algorithms among a set of alterna-

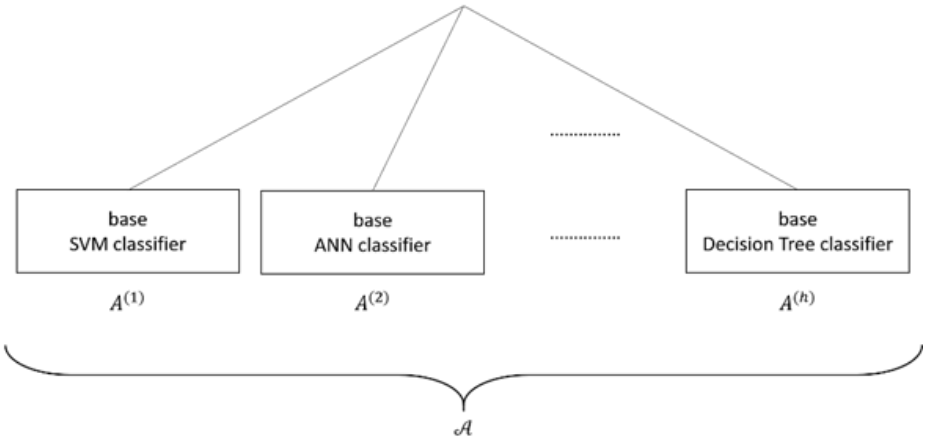


FIGURE 2.1: An example of Model Selection. A set of "base" algorithms are available in the set $\mathcal{A}$, where "base" refers to the adoption of default values for the algorithm's hyperparameters (Archetti and Candelieri, 2019)

tives, Hyperparameter Optimization (HPO), discussed in the following Subsection, is aimed at optimizing the hyperparameters of a given learning algorithm A. Thus, in Model Selection hyperparameters are "fixed" a-priori and the optimal algorithm must be efficiently selected among

a set of possible candidates, while in HPO the algorithm is "fixed" a-priori and the optimal values for its own hyperparameters are to be efficiently identified.
Before addressing HPO, it is important to clarify the difference between *training* and *validating* a ML model, as well as the difference between *parameters* and *hyperparameters* of a ML algorithm.
The algorithm A is characterized by:

- a vector θ of *model parameters*, whose value is learned directly from data, during the *training phase*
- a vector of *hyperparameters* $\gamma \in \Gamma$ that change the way the algorithm "learns" how to set the values for θ. Hyperparameters can be set up manually or optimized on the *validation* phase.

For example, the weights on the connections of an Artificial Neural Network (ANN) are parameters θ to be learned, while number of hidden layers, number of neurons in each hidden layer, activation function and learning rate are hyperparameters γ which rule how the optimal parameters are learned.
The final aim of training is to estimate the value $\hat{\theta}$ minimizing the training loss $\mathcal{L}_{train}$ computed on a given training dataset or fold. The resulting trained ML model can be then validated on a specific validation dataset or fold $D^{(i)}_{valid}$, measuring the corresponding validation loss $\mathcal{L}_{valid}$, which is the argument of 2.1. During the validation phase, the estimate $\hat{\theta}$ does not change. The hyperparameters γ are not estimated during the training phase, they must be set up before training. Therefore, $\hat{\theta}$ as well as $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ depend on the value of γ.
Many ML algorithms, such as ANN or Support Vector Machines (SVM), use an analytical form for $\mathcal{L}_{train}$, so that $\hat{\theta}$ can be estimated by gradient-based methods.
However, in particular for optimizing Deep Neural Network a combination of two levels optimization are proposed in (Franceschi et al., 2018). The authors propose to optimize the ML problem exploiting a bilevel optimization that takes into account both the parameters and the hyperparameters of a Deep Neural Network using a new formalization of a gradient as hyper gradient.
Other ML algorithms do not have parameters (e.g., instance-based algorithms such as K-nearest neighbours) but have hyperparameters like K or the similarity metric used. On the contrary, $\mathcal{L}_{valid}$ is black-box and its optimization, depending on the hyperparameters γ, requires derivative-free Global Optimization (GO) approaches. When k-fold cross validation is concerned, the common way to compute the overall $\mathcal{L}_{valid}$ is to average the values obtained on the k different folds, as in a randomized experiment.
It is important to remark that the argument of 2.1 is the average of the loss on each fold of a k-fold cross validation procedure. This allows to obtain just a single value for each γ, possibly

noisy in the case that the algorithm A is affected by some randomness (e.g., initial weights of an ANN). Another possibility is to consider the distribution of the losses on the k folds and then use the information collected on the resulting randomized experiment for implementing early stopping for those hyperparameters unlikely to yield a good result (Florea and Andonie, 2018). The latter principle is used in recent AutoML libraries such as Hyperband proposed in (Li et al., 2017b), halving sequentially the set of configurations to save "budget" for the most promising ones.

2.1.2 Hyperparameter Optimization

The second aspect to consider to implement an AutoML system is the choice of the optimal configuration of a selected ML algorithm A to solve the target problem, and it knows as *Hyper-Parameter Optimization* (HPO).
Given n hyperparameters $\gamma_1, ..., \gamma_n$ with domains $\Gamma_1, ..., \Gamma_n$, the hyperparameter space Γ is a subset of the product of these domains $\Gamma_1 \times ... \times \Gamma_n$. Γ is a subset because certain settings of one hyperparameter render other hyperparameters inactive. For example, the hyperparameters determining the specifics of the third layer of an ANN are not relevant if the network depth is less than three. Likewise, the hyperparameters of SVM with polynomial kernel are not relevant if we use a different kernel instead.
More formally, we say that a hyperparameter γ_i is conditional on another hyperparameter γ_j, that is γ_i is active if and only if hyperparameter γ_j takes values from a given set $V_i(j) \subset \Gamma_j$; in this case we call γ_j a parent of γ_i. Conditional hyperparameters, generate a tree-structured space or, in some cases, a Directed Acyclic Graph (DAG). Given such a structured space Γ, the (hierarchical) hyperparameter optimization problem can be written as:

$$\gamma^* \in \arg\min_{\gamma \in \Gamma} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_\gamma, D_{train}^{(i)}, D_{valid}^{(i)}) \tag{2.2}$$

The two following Figures 2.2 and 2.3 propose two examples of hyperparameter optimization for an SVM and an ANN, respectively.

The Figure 2.2 shows the conditional tree generated by the available hyperparameters of SVM algorithm, where the γ_3 (the length scale σ) exists due the choice of the γ_2 equals to *RBF*, which represents the kernel adopted inside of algorithm. Instead, in the example of HPO for ANN (Figure 2.3) there are ten hyperparameters whose four are conditioned to the others, two hyper-aparameters related to the choice of the optimizer adopted during the training process and three other hyperparameters for the definition of the shape and depth of the ANN.
Both Model Selection and HPO are Global Optimization problems characterized by a black-box, expensive and multi-extremal objective function. However, they present relevant differences that drive the choice of the optimization methods to solve them. More precisely, Model Selection is

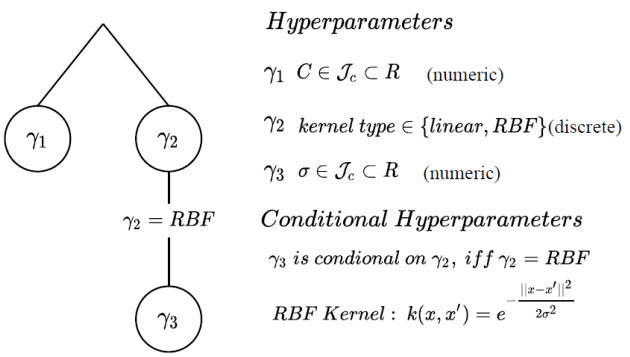
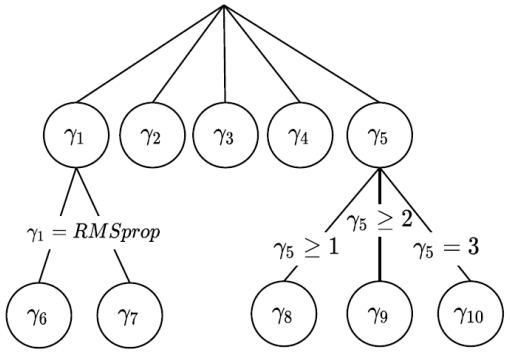


FIGURE 2.2: An example of HPO task for a Support Vector Machine (SVM) classifier with linear or Radial Basis Function (RBF) kernel. $\mathcal{J}_C$ and $\mathcal{J}_\sigma$ are the ranges for values of the regularization hyperparameter C and the *RBF* kernel's hyperparameter σ



Conditional Hyperparameters

γ_6 *is condional on* γ_1, $iff\ \gamma_1 = RSMprop$

γ_7 *is condional on* γ_1, $iff\ \gamma_1 = RSMprop$

γ_8 *is condional on* γ_5, *but it always is* : $\gamma_5 \geq 1$

γ_9 *is condional on* γ_5, $iff\ \gamma_5 \geq 2$

γ_{10} *is condional on* γ_5, $iff\ \gamma_5 = 3$

Hyperparameters

γ_1 *optimizer* $\in \{SGD, RMSprop\}$ (discrete)

γ_2 *learning rate*, $\gamma_2 \in \mathbb{R}^+$ (numeric)

γ_3 *momentum*, $\gamma_3 \in \mathbb{R}^+$ (numeric)

γ_4 *activation function* $\in \{sigmoid, tanh\}$ (discrete)

γ_5 *number of hidden layers*, $\gamma_5 \in \mathcal{J} \subset \mathbb{N}^+$ (discrete)

γ_6 *rho*, $\gamma_6 \in \mathbb{R}^+$ (numeric)

γ_7 *epsilon*, $\gamma_7 \in \mathbb{R}^+$(numeric)

γ_8 *number of neurons in the hidden layer* 1, $\gamma_8 \in \mathcal{J}_1 \subset \mathbb{N}^+$(discrete)

γ_9 *number of neurons in the hidden layer* 2, $\gamma_9 \in \mathcal{J}_2 \subset \mathbb{N}^+$(discrete)

γ_{10} *number of neurons in the hidden layer* 3, $\gamma_{10} \in \mathcal{J}_3 \subset \mathbb{N}^+$(discrete)

FIGURE 2.3: An example of HPO task for an Artificial Neural Network (ANN) classifier with at maximum 3 hidden layers. $\mathcal{J}_1$, $\mathcal{J}_2$ and $\mathcal{J}_3$ are the ranges for number of neurons in the hidden layer 1, 2 and 3, respectively. In addition a choice of two types of optimizers (SGD and RMSprop) are considered to train the ANN.

a problem characterized by a single discrete decision variable (i.e., the best algorithm among a set of predefined alternatives). On the contrary, HPO involves as many decision variables as the hyperparameters of the learning algorithm, and they can be continuous, discrete, mixed as well as conditional. The following subsection will present the most general AutoML problem definition, involving, at the same time, both Model Selection and HPO, which is, from a mathematical point of view, an extension of the HPO formulation.

2.1.3 Combined Algorithm Selection and Hyperparameter optimization (CASH)

An AutoML system has to take into account selecting, at the same time, the optimal hyperparameters configuration on an optimal ML algorithm for solving the target problem and can be so-called Combined Algorithm Selection and Hyperparameter optimization (CASH) (Thornton et al., 2013).
More formally, given a set of algorithms $\mathcal{A} = A^{(1)},...,A^{(n)}$ with associated hyperparameter spaces, $\Gamma^{(1)},...,\Gamma^{(n)}$, the CASH problem can be formalized as follows:

$$A^*_{\gamma^*} \in \underset{A^{(j)}\in\mathcal{A},\gamma\in\Gamma^{(j)}}{\arg\min} \frac{1}{k}\sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\gamma}, D^{(i)}_{train}, D^{(i)}_{valid}) \tag{2.3}$$

It has to be remarked that this problem can be reformulated as a single combined hierarchical HPO problem with hyperparameter space $\Gamma = \Gamma^{(1)} \cup ... \cup \Gamma^{(n)} \cup \{\gamma_r\}$, where γ_r is a new root-level hyperparameter that selects between algorithms $A^{(1)},...,A^{(n)}$. The root-level hyperparameter of each subspace $\Gamma^{(i)}$ are made conditional on γ_r, as can be seen in Figure 2.4.

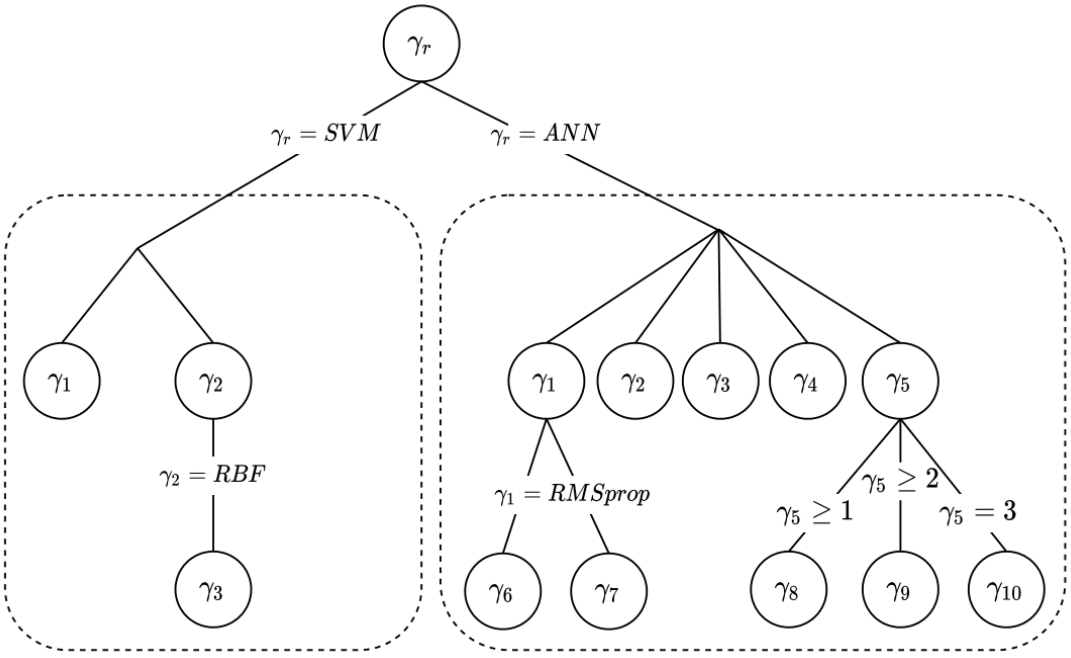


FIGURE 2.4: An example of CASH considering an SVM and an ANN classifier. Description of the hyperparameters of each algorithm follows from previous Figure 2.2 and Figure 2.3, while γ_r is the further "root" CASH hyperparameter introduced to model the choice between the Machine Learning algorithm

Another critical feature is that the value of the loss function, for each hyperparameter configuration, is the outcome of the randomized process of k fold-cross validation. An often-overlooked point is to measure the uncertainty of the prediction error estimators which is important because the accuracy of model selection is limited by the variance of model estimates. Cross validation

provides an unbiased estimate of the prediction error on the training set, but the estimation of the variance is still crucial. An important contribution to this problem is given in (Bengio and Grandvalet, 2004).
In a recent paper (Jiang and Wang, 2017) a uniform normalized variance is proposed which not only measures model accuracy but relates it to the number of folds.
Finally, AutoML could be in principle tackled through any derivative-free Global Optimization method. The most relevant approaches will be discussed in the following Chapter, discussing pros and cons with respect to the characteristic of the specific setting (i.e., Model Selection, HPO or CASH). Here it is important to anticipate that Sequential Model-based Bayesian Optimization (SMBO) – or simply Bayesian Optimization (BO) – is currently the most widely adopted method in developing open-source and commercial tools for AutoML. SMBO is a versatile stochastic optimization framework that can work with both categorical and continuous hyperparameters, and that can exploit hierarchical structures stemming from conditional parameters (Kotthoff et al., 2017). The main reason of its success is its sample efficiency in selecting promising algorithms and/or hyperparameters configurations by effectively dealing with the exploration-exploitation dilemma, which will be introduced in Section 4.2.

2.1.4 Recents advances in AutoML

Currently in the community of Machine Learning (ML) and Data Scientist exist several open source frameworks and webservices that are able to efficiently apply AutoML techniques to a single ML algorithm or an entire ML pipeline. This section lists several frameworks currently available to be used even by people who are not experts in the domain of Machine Learning and optimization.

Open source.
Auto-WEKA[1] is one of the first AutoML system based on CASH paradigm. The first version of Auto-WEKA was proposed in (Thornton et al., 2013) and it is implemented in Java. Indeed, this system uses the WEKA Machine Learning library written in Java, which was the first to be chosen as library, because it used several implementation of available classification algorithms in literature (e.g. Multilayer Perceptron, BayesNet, J48 and others). This system adopts Bayesian Optimization paradigm based on TPE (Bergstra et al., 2011) and SMAC (Hutter et al., 2011) libraries. The first library is based on the Tree-structured Parzen Estimator which is a probabilistic density estimator, while the second library uses Random Forest as a surrogate model that shows better performance in terms of optimization in respect of Gaussian Processes, which are common choice in the Bayesian Optimization paradigm. This library with the version 2.0 is natively installed as an optional package in the WEKA library written in Java. The main goal

[1] `https://github.com/automl/autoweka`

of this system is to provide to the final users a very autonomous optimizer using GUI or CLI interface, which works without any suggestion of the final users. With this new version was added the possibility to optimize also Regression algorithms besides the classification ones.

hyperopt-sklearn[2] is an AutoML open source library which is written in Python proposed in (Komer et al., 2014), it is the first attempt to apply AutoML approach in combination with python library Sklearn (Pedregosa et al., 2011) that is widely used by Machine Learning community. The optimization paradigm of this library is hyperopt[3] (Bergstra et al., 2013), which enables to implement an optimization process that could exploit scalability in respect of the Auto-WEKA (Thornton et al., 2013) implemented, in its first version, only in Java without the possibility to scale the computation of the optimization process.

Auto-sklearn[4] is a library that could be considered as the first approach to a complete AutoML system written in Python language. However, it is different from Auto-WEKA 2.0 and hyperopt-sklearn because Auto-sklearn implements the Bayesian Optimization paradigm and also Meta-Learning techniques. This library, firstly proposed in (Feurer et al., 2015), tries to follow the same path of Auto-WEKA associating its implementation with a famous Machine Learning library written in Python named Sklearn (Pedregosa et al., 2011). The adoption of sklearn library is already experimented by hyperopt-sklearn in a smaller and less complete library. This system for the BO paradigm exploits SMAC3[5], currently supported version on Github of the originally SMAC algorithm reimplemented totally in Python language to promote the scalability of the computation. Auto-sklearn showed its incredible performance by winning six out of ten phases of the first ChaLearn AutoML challenge. A comprehensive analysis on over 100 diverse datasets showed that it substantially outperforms the previous state of the art in AutoML. Recently a new version of Auto-sklearn, is proposed in (Feurer et al., 2020) named Auto-sklearn 2.0.

Auto-Keras[6][7] is an open source Python library for AutoML. It is developed by DATA Lab at Texas A&M University and community contributors. Auto-Keras provides functions to automatically search for architecture and hyperparameters of Deep Neural Networks with Keras[8] Deep Learning framework. This library adopts internally Keras-Tuner[9] a Python library that implements several optimization algorithms (Bayesian Optimization, Hyperband, RandomSearch)

[2] https://github.com/hyperopt/hyperopt-sklearn
[3] https://github.com/hyperopt/hyperopt
[4] https://github.com/automl/auto-sklearn
[5] https://github.com/automl/SMAC3
[6] https://github.com/jhfjhfj1/autokeras
[7] https://autokeras.com/
[8] https://keras.io/
[9] https://github.com/keras-team/keras-tuner

for optimizing specifically DNN architectures for classification and regression tasks.

TPOT[10] is a Tree-based Pipeline Optimization Tool (TPOT), proposed in (Olson and Moore, 2019), and it is a Python open source implementation of AutoML system that exploits Genetic Programming for optimizing Machine Learning pipelines of classification tasks. TPOT is a wrapper for the Python machine learning package, such as Scikit-learn, and adopts Python package named DEAP (Fortin et al., 2012) to implements Genetic Programming algorithm. The operators of selection, crossover and mutation are provided by NSGA-II (Deb et al., 2002) during the optimization process of a pipeline. To show the efficacy of this system the authors evaluated it on 150 supervised classification benchmarks[11], with different kinds and number of attributes for each dataset. In respect of the others AutoML systems reported in this section TPOT is the only one that adopts a different optimization paradigm based on Metaheuristic approach. However it has the main drawback of evaluating a huge number of Machine Learning pipelines configurations to converge to an hypothetical optimal pipeline configuration.

H2O[12] is an open source written in Java language, in-memory, distributed, fast, and scalable Machine Learning and predictive analytics platform that allows to build Machine Learning models on Big Data and provides easy deployability of those models in an enterprise environment. H2O's REST API allows access to all the capabilities of H2O from an external program or script via JSON over HTTP. The Rest API is used by H2O's web interface (Flow UI), R binding (H2O-R), and Python binding (H2O-Python). H2O's AutoML can be used for automating the Machine Learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. H2O uses Stacked Ensembles which is a feature based on using only the best model of each family algorithm available previously trained.

Dragonfly[13] is a scalable Bayesian Optimization Python open source library proposed in (Kandasamy et al., 2020). However, in respect of the other open source libraries, it is a general-purpose optimizer that can be also used for AutoML task (e.g. classification task exploiting DNN). Indeed this library is composed by several approaches proposed in literature to manage problems with a large number of dimensions (as in DNN models) and parallelization systems in order to reduce the time required from the library for completing the optimization process on a classification task. Internally, it adopts a recent AutoML system for DNN proposed in (Kandasamy et al., 2018) so-called NASBOT.

[10] `http://epistasislab.github.io/tpot/`
[11] `https://github.com/EpistasisLab/penn-ml-benchmarks`
[12] `https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html#experimental-features`
[13] `https://github.com/dragonfly/dragonfly`

Webservice.

Google AutoML[14] is a service of Google Cloud Platform (GCP) for automating the whole process carried out by a data scientist. In more detail, AutoML suite of GCP can be separated in three macro categories which are defined by the format of the dataset that we want to adopt to solve a specific task. In particular there are three types of Google services for AutoML, which are AutoML Vision[15]/Video[16] Intelligence based on images datasets, AutoML Natural Language[17]/Translation[18] based on text datasets and the most general purpose one which is based on structure dataset (e.g. relational tables) named AutoML Tables[19].
The last Google Service is the most relevant because is applicable to several use cases. The main goal of this system is to propose the best Machine Learning model to solve classification or regression tasks on a given structured dataset with the aim of using only the resources available on GCP, which are "virtually" unlimited in terms Central Processing Units (CPU), Random Access Memory (RAM) and dedicated hardware such as Graphical Processing Unit (GPU).
This system is currently taking into account a very limited set of ML algorithms, which are:

- Linear
- Feedfoward Deep Neural Network
- AdaNet
- Ensembles of various model architectures

As a featuring engineering task Google AutoML Tables adopts several kinds of preprocessing techniques depending on the types of features contained in the given dataset. For example, the normalization of data is adopted for numeric continuous features, while several kinds of new representations are adopted for categorical features, such as One-Hot encoding or a more complex embedding representations.

Amazon SageMaker Autopilot[20] is an AutoML service offered by Amazon for its Machine Learning platform Amazon SageMaker. The final user can use this service through SageMaker Python SDK or other scripting languages (e.g., R, GO and others), which gives to the user the API access to service of hyperparameters tuner on Amazon Cloud Platform (aka AWS).
Amazon SageMaker Autopilot supports only the optimization of Machine Learning pipelines based on structured datasets that are hosted mainly on cloud storage system in AWS platform. The ML tasks supported by this service are Binary and Multiclass classification and regression tasks. However the documentation of Amazon SageMaker Autopilot does not report the

[14] https://cloud.google.com/automl?hl=en
[15] https://cloud.google.com/vision/overview/docs?hl=it#automl-vision
[16] https://cloud.google.com/video-intelligence/automl/docs?hl=it
[17] https://cloud.google.com/natural-language/automl/docs?hl=it
[18] https://cloud.google.com/translate/automl/docs?hl=it
[19] https://cloud.google.com/automl-tables/docs?hl=it
[20] https://aws.amazon.com/it/sagemaker/autopilot/

models and the preprocessing techniques, which are adopted to optimize the Machine Learning pipelines.
In addition, a relevant newest feature related to Amazon Sagemaker is the Amazon SageMaker Neo[21] which enables a reduction of hardware requirements to compute the inference of Deep Neural Networks models with different Deep Learning frameworks (such as TensorFlow, Pytorch and others).

Azure Automated Machine Learning[22] is the AutoML service provided by Microsoft on its machine learning cloud platform so-called Azure Machine Learning. The final user can interact with Microsoft cloud platform and its AutoML service through Python SDK or Command Line Interface (CLI) or Graphical Unit Interface (GUI) with a common browser. With this web-service it is possible to automatically optimize and perform classification, regression and time series forecasting tasks. Currently, the only supported dataset for Azure Automated Machine Learning has to be in tabular form. As the others AutoML cloud solutions mentioned before, Microsoft enables to automatize also the preprocessing of the attributes of the input dataset with widely used techniques as normalization, One-Hot encoding and others to fix problems such as missing values that could be present into the dataset.
This system is currently taking into account a wider set of ML algorithms in respect of Google, which are the following: Averaged Perceptron Classifier, Bernoulli Naïve Bayes, Decision Tree, Extra Trees, Gradient Boosting, K-Nearest Neighbours Classifier, Light GBM Classifier, Linear Support Vector Machine, Logistic Regression, Multinomial Naïve Bayes, Random Forest, SGD Classifier, Support Vector Machine, TensorFlow Deep Neural Network Classifier, XGBoost Classifier.
Another relevant feature of this service is that it is possible to apply this AutoML system to Azure Machine Learning Pipeline. The latter functionality of the Azure Cloud platform allows the final user to create a ML pipeline to solve a specific task (classification, regression, and time series forecasting), optimizing all components within the pipeline with multiple ML models in the same optimization process.

Sigopt[23] is sold as a "Black Box Optimization as a Service". The main core of this web service optimizer was initially delevoped for a project named MOE by Cornell University. The final user has to interact through REST API in several languages like Java, R and Python with the most updated version of each language. The main idea behind this optimization service is that the user is totally blind to the computational architecture infrastructure which is adopted for optimization. To use this service the user has to subscribe an Academia or Enterprise account.

[21] `https://aws.amazon.com/it/sagemaker/neo/`
[22] `https://azure.microsoft.com/it-it/services/machine-learning/automatedml/`
[23] `https://sigopt.com/`

Recently this system demonstrated the ability to increase performances in particular of NLP model based on DNN as showed in SigOpt Blog[24]

2.2 AutoML for Deep Neural Networks

When AutoML is used to optimize a Deep Neural Network (DNN) some specificities about Deep Learning must be appropriately considered, in particular with regard to the types and composition of layers constituting the network's architecture. Indeed, in literature it is possible to distinguish two main *tracks* related to Automated Deep Learning (AutoDL): the problem is addressed either as HPO or – from a more advanced perspective – by considering both network's hyperparameters and architecture in the optimization process. This Section provides an overview about the state of the art of both the two perspectives.

2.2.1 Automated Deep Learning as HPO Problem

Firstly, it should be noted that DNN could have ten times more hyperparameters than other ML algorithms, which are characterized by a small number of hyperparameters. Despite the high number of DNN's hyperparameters that increases the complexity of the problem that needs to be optimize, the AutoML framework is still applicable for searching the optimal configuration of DNN, specifically by considering it as an HPO problem.
According with this vision, an initial approach was proposed in (Snoek et al., 2012) to optimize a fully connected feed-forward network on the CIFAR-10 dataset, comparing Bayesian Optimization (BO) against other evolutionary global optimization methods.
As the main result, BO was able to identify more accurate models than other optimization strategies, and in less time. Moreover, BO was also able to surpass a human expert at selecting hyperparameters on the competitive CIFAR-10 dataset, beating the state of the art by over 3%.
Another application of AutoML for optimizing a DNN was proposed in (Mendoza et al., 2016). They proposed Auto-Net 1.0, a system to optimize a fully connected feed-forward Deep Neural Network, for both classification and regression tasks. The system is implemented within Auto-sklearn (Feurer et al., 2015) by adding to this library a new classification and regression algorithm. The authors motivate the choice of exploiting the structure of Auto-sklearn, because it allows them to leverage on existing parts of Machine Learning pipeline, such as data preprocessing and feature preprocessing.
The authors of Auto-Net 1.0, following what is suggested in (Bergstra et al., 2013, 2011; Domhan et al., 2015), distinguish between: *(i)* layer-independent network hyperparameters (e.g., batch size, number of hidden layers, optimizer and learning rate) which affect both architecture

[24] `https://sigopt.com/blog/efficient-bert-overview/` [Accessed on: 30/08/2020]

and learning, and *(ii)* per-layer hyperparameters (e.g., activation function, number of units), which are set for each layer. For all types of supervised learning task (binary, multiclassification and regression) they optimize the same configuration space composed by 63 hyperparameters, which are reported with the table in Figure 2.5. Authors kept the number of hyperparameters by limiting the number of layers in one to six. With respect to the optimizer to use for learning network's weight, authors have considered several famous optimizers in literature (e.g. vanilla stochastic gradient descent (SGD), stochastic gradient descent with momentum (Momentum), Adam (Kingma and Ba, 2014), Adadelta (Zeiler, 2012), Nesterov momentum (Nesterov, 1983) and Adagrad (Duchi et al., 2011)).

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓
	leakiness	$[0.01, 0.99]$	$\frac{1}{3}$	-	float	✓
	tanh scale in	$[0.5, 1.0]$	$2/3$	-	float	✓
	tanh scale out	$[1.1, 3.0]$	1.7159	✓	float	✓

FIGURE 2.5: Hyperparameter configuration space of Auto-Net 1.0 proposed in (Mendoza et al., 2016)

With this initial and limited work (Mendoza et al., 2016), due to the use of only feed-forward Network, however, it is clear that the adoption of AutoML system to DNN, only as HPO task, could lead to generating optimal and performing models.

Successively, in (Mendoza et al., 2019; Zimmer et al., 2020), a new version of Auto-Net, namely Auto-Net 2.0 (aka Auto-pytorch) has been proposed. The main upgrades are related to the a wider set of possible architectures for classification as well as regression tasks and the possibility to implement HPO through BO – as in the previous version – or through BOHB (Falkner et al., 2018), a combination of BO and Hyperband (Li et al., 2017b) to substantially improve the optimization process efficiency.

Having several types of DNN architectures in Auto-Net 2.0, it has been defined a wider and hierarchical search space in respect of the search space defined for Auto-Net 1.0 reported above in Figure 2.5.

There are four types of DNN. First, Multi-Layer Perceptrons where each layer of the network is

parameterized in respect of for example activation function and number of units. Second, Residual Neural Networks is a DNN composed by several blocks with same or different fixed DNN architecture inside of each block and a famous residual network is ResNets. The authors of Auto-Net 2.0 starting from the concept of ResNet network compose the network by M groups, each of which composed by N residuals blocks in sequence.
Third and fourth, Shaped Multi-Layer Perceptrons and Shaped Residual Networks which allow the authors to define a-priori several different shapes for both kinds of DNN as a funnel, long funnel, diamond, hexagon, brick, or triangle of quite common use in DL community.
Moreover, as other parameters under optimization, Auto-Net 2.0 in respect of the previous version, currently offers five different schedulers for modifying the optimizer's learning rate during the optimization process, which are: Exponential, Step, Cyclic (Smith, 2017), Cosine Annealing with Warm Restarts (Loshchilov and Hutter, 2016), and On Plateau (in case the performance metric not improving for several iterations).
Furthermore, Auto-Net 2.0 supports several preprocessing techniques (e.g., Nyström (Christopher et al., 2001), Kernel principal component analysis (Schölkopf et al., 1997), random kitchen sinks (Rahimi and Recht, 2009) and truncated singular value decomposition (Halko et al., 2009)) which are retrieved by exploiting Auto-sklearn library. For all types of supervised learning tasks (binary, multiclassification and regression), with Auto-Net 2.0, the authors increased almost of two times the number of the hyperparameters that compose the configuration search space arriving to a total 112 hyperparameters, which are reported in Figure 2.6.
As already mentioned above, the strength of Auto-Net 2.0 compared to the previous version is also the optimizer used during the optimization process. In fact, the authors adopting BOHB can drastically reduce the time to run the entire optimization process, because by leveraging Hyperband the optimizer invests most of the runtime in promising neural network configurations and stops training neural network configurations with poor performance in advance. Furthermore, after focusing only on the most promising regions of the research space in terms of DNN's performance metrics, it adopts TPE (Bergstra et al., 2011) to further refine and improve the most promising network configurations. In addition, the authors of Auto-Net 2.0, point out that BOHB is easily parallelizable, achieving almost linear speedup with an increasing number of workers (Falkner et al., 2018).
None of the systems previously mentioned applies HPO task on DNN taking into account during the optimization one or more computability constraints. Indeed, it happens that some hyperparameters configurations could require more computational resources than it was accounted for initial hypothesis. For this reason, the work that will be presented in Chapter 6 could be considered a novel approach to Automated Deep Learning through constrained optimization in an HPO problem.

	Name	Range	Default	Log scale	Type	Conditional
General hyperparameters	Batch size	[32, 500]	32	✓	int	–
	Use mixup	{True, False}	True	–	bool	–
	Mixup alpha	[0.0, 1.0]	1.0	–	float	✓
	Network	{MLP, ResNet, ShapedMLP, ShapedResNet}	MLP	–	cat	–
	Optimizer	{Adam, SGD}	Adam	–	cat	–
	Preprocessor	{nystroem, kernel pca, fast ica, kitchen sinks, truncated svd}	Nystroem	–	cat	–
	Imputation	{most frequent, median, mean}	Most frequent	–	cat	–
	Use loss weight strategy	{True, False}	True	–	cat	–
	Learning rate scheduler	{Step, Exponential, OnPlateau, Cyclic, CosineAnnealing}	Step	–	cat	–
Preprocessor						
Nystroem	Coef	[−1.0, 1.0]	0.0	–	float	✓
	Degree	[2, 5]	3	–	int	✓
	Gamma	[0.00003, 8.0]	0.1	✓	float	✓
	Kernel	{poly, rbf, sigmoid, cosine}	rbf	–	cat	✓
	Num components	[50, 10000]	100	✓	int	✓
Kitchen sinks	Gamma	[0.00003, 8.0]	1.0	✓	float	✓
	Num components	[50, 10000]	100	✓	int	✓
Truncated SVD	Target dimension	[10, 256]	128	–	int	✓
Kernel PCA	Coef	[−1.0, 1.0]	0.0	–	float	✓
	Degree	[2, 5]	3	–	int	✓
	Gamma	[0.00003, 8.0]	0.1	✓	float	✓
	Kernel	{poly, rbf, sigmoid, cosine}	rbf	–	cat	✓
	Num components	[50, 10000]	100	✓	int	✓
Fast ICA	Algorithm	{parallel, deflation}	Parallel	–	cat	✓
	Fun	{logcosh, exp, cube}	Logcosh	–	cat	✓
	Whiten	{True, False}	True	–	cat	✓
	Num components	[10, 2000]	1005	–	int	✓
Networks						
MLP	Activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	–	cat	✓
	Num layers	[1, 15]	9	–	int	✓
	Num units (for layer i)	[10, 1024]	100	✓	int	✓
	Dropout (for layer i)	[0.0, 0.5]	0.25	–	int	✓
ResNet	Activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	–	cat	✓
	Residual block groups	[1, 9]	4	–	int	✓
	Blocks per group	[1, 4]	2	–	int	✓
	Num units (for group i)	[128, 1024]	200	✓	int	✓
	Use dropout	{True, False}	True	–	bool	✓
	Dropout (for group i)	[0.0, 0.9]	0.5	–	int	✓
	Use shake drop	{True, False}	True	–	bool	✓
	Use shake shake	{True, False}	True	–	bool	✓
	Shake drop β_{max}	[0.0, 1.0]	0.5	–	float	✓
ShapedMLP	Activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	–	cat	✓
	Num layers	[3, 15]	9	–	int	✓
	Max units per layer	[10, 1024]	200	✓	int	✓
	Network shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	–	cat	✓
	Max dropout per layer	[0.0, 0.6	0.2	–	float	✓
	Dropout shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	–	cat	✓
Shaped ResNet	Activation function	{Sigmoid, Tanh, ReLu}	Sigmoid	–	cat	✓
	Num layers	[3, 9]	4	–	int	✓
	Blocks per layer	[1, 4]	2	–	int	✓
	Use dropout	{True, False}	True	–	bool	✓
	Max units per layer	[10, 1024]	200	✓	int	✓
	Network shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	–	cat	✓
	Max dropout per layer	[0.0, 0.6	0.2	–	float	✓
	Dropout shape	{Funnel, LongFunnel, Diamond, Hexagon, Brick, Triangle, Stairs}	Funnel	–	cat	✓
	Use shake drop	{True, False}	True	–	bool	✓
	Use shake shake	{True, False}	True	–	bool	✓
	Shake drop β_{max}	[0.0, 1.0]	0.5	–	float	✓
Optimizers						
Adam	Learning rate	[0.0001, 0.1]	0.003	✓	float	✓
	Weight decay	[0.0001, 0.1]	0.05	–	float	✓
SGD	Learning rate	[0.0001, 0.1]	0.003	✓	float	✓
	Weight decay	[0.0001, 0.1]	0.05	–	float	✓
	Momentum	[0.1, 0.9]	0.3	✓	float	✓
Schedulers						
Step	γ	[0.001, 0.9]	0.4505	–	float	✓
	Step size	[1, 10]	6	–	int	✓
Exponential	γ	[0.8, 0.9999]	0.89995	–	float	✓
OnPlateau	γ	[0.05, 0.5]	0.275	–	float	✓
	Patience	[3, 10]	6	–	int	✓
Cyclic	Cycle length	[3, 10]	6	–	int	✓
	Max factor	[1.0, 2.0]	1.5	–	float	✓
	Min factor	[0.001, 1.0]	0.5	–	float	✓
Cosine annealing	T_0	[1, 20]	10	–	int	✓
	T_{mult}	[1.0, 2.0]	1.5	–	float	✓

FIGURE 2.6: Hyperparameter configuration space of Auto-Net 2.0 proposed in (Mendoza et al., 2019)

2.2.2 Neural Architecture Search

Despite the successfully approaches reported in the previous Subsection, addressing AutoDL as an HPO problem has many limitations. Indeed, HPO on a shallow neural network, such as an MLP, can modify its architecture if number of layers and number of units for each layer are among the hyperparameters to be optimized. However, when a DNN is considered, its architecture is more complicated, with different types of layers (e.g., convolutional, dropout, dense, etc.) and/or sub-networks which can be connected among them. Consequently, modelling complicated deep architectures through hyperparameters becomes impractical as showed from Figure 2.7.

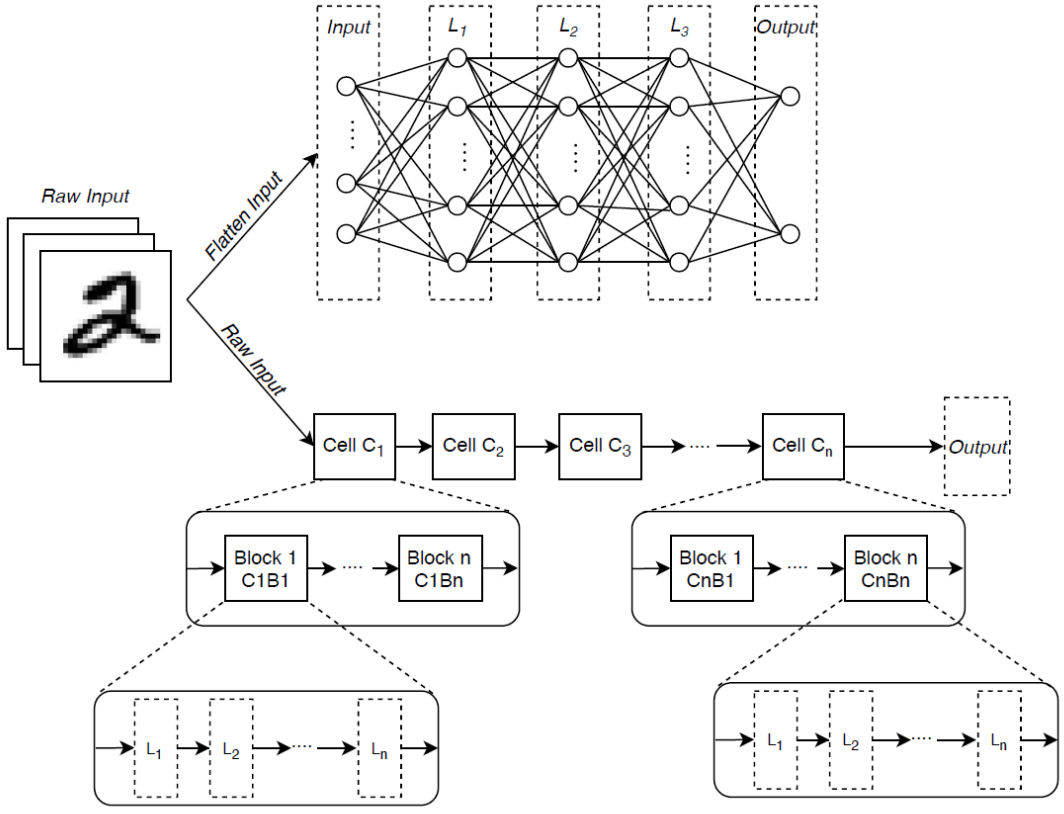


FIGURE 2.7: On the top of the figure a traditional Multi-Layer Perceptron is represented, instead in the bottom is represented a more complex Deep Neural Network based on fixed cells and blocks with different types of layers

The need for efficiently searching for an optimal deep architecture led to develop appropriate Neural Architecture Search (NAS) approaches (Elsken et al., 2019), such as Evolutional Algorithms (EAs), Reinforcement Learning (RL), gradient-based methods (DARTs), Random Search (RS), and Bayesian Optimization (BO).

The first approach is based on Evolutionary Algorithms (EAs) (Angeline et al., 1994; Floreano et al., 2008; Stanley and Miikkulainen, 2002; Stanley et al., 2009), aimed at "evolving" neural

architectures by following the paradigm of the natural evolution, that is generating new individuals (i.e., architectures) by combining the fittest ones (i.e., the most accurate) while giving some chance for casual mutation (i.e., random changes the generated architectures).
More precisely, in NAS the mutation operations can include to remove or add connections between layers or remove/add entire layers or altering some hyperparameters related to a single layer (such as activation function and number of units) or the learning process (such as batch size, number of training epochs, and learning rate).
In literature have been proposed several works that exploit EAs to solve NAS problem, the differences between these several works (Elsken et al., 2018; Liu et al., 2018; Miikkulainen et al., 2019; Real et al., 2017, 2019; Suganuma et al., 2017; Xie and Yuille, 2017) are essential in how they sample the most promising parents (selection operation), update population, and generate offsprings (mutation operation).
Indeed for selection operation, several works (Liu et al., 2018; Real et al., 2017, 2019) adopt tournament selection (Goldberg and Deb, 1991), a famous selection approach in EAs community, while a new selection approach in (Elsken et al., 2018) selects the parents from a multi-objective Pareto front using an inverse density. Instead, in respect to offsprings generation, a common practice in AEs community is to randomly initialize child networks, which allows escaping from local optima. However, recent work by (Elsken et al., 2018) proposed to transfer the knowledge of the weights of the parent network to its children by using network morphisms (Chen et al., 2016; Wei et al., 2017).
Another branch of optimization technique to solve NAS is RL, which can be implemented in several different ways. For instance in (Baker et al., 2017; Zhong et al., 2018; Zoph and Le, 2017; Zoph et al., 2018) the process of generation of a neural architecture is provided by the agent's action, where the action space is identical to the search space of the NAS problem. The reward of the RL agents typically is represented by the cumulative reward that in this task represents the performances of the DNN generated during the RL sequential process. In particular, the sequential aspect in the optimization process in (Zoph and Le, 2017) is given by the adoption of a Recurrent Neural Network (RNN) to define the policy that the RL agent has to adopt in order for proposing new DNN architectures. Instead, in (Baker et al., 2017) Q-Learning algorithm is used to train a RL policy, whose choices are sequentially the layer's type and its hyperparameters.
Another approach to solve the NAS problem with the RL approach is in (Cai et al., 2018), where the authors adopt a Long-Short Term Memory Neural Network (LSTM) to implement an RL policy, where the actions are represented as mutations of the initial network. Despite the RL approach to solve NAS, seems to be very effective in terms of optimal networks generated, it exists another branch of methods name the Differentiable ARchiTecture Search (DARTs) (Liu et al., 2019a). The main idea of this approach is to relax the search space to be continuous instead to have a discrete set of candidate architectures. Due to this relaxation of the search space it is possible to apply gradient-based optimization approach to discover new performance neural

architectures, leveraging on the fact that gradient-based optimization is data efficiency, which is essentially opposed to inefficient black-box search as EAs and Random Search.

To be a fair review of all optimization strategies for NAS problem, in (Li and Talwalkar, 2020) the authors analyzed the efficacy of using Random Search (RS) to find out an optimal or at least competitive DNN architecture. The authors proved that RS strategy with early-stopping is a competitive NAS baseline on both benchmarks tested, achieving a state-of-the-art NAS result on PTB and a highly competitive result on CIFAR-10. Finally, they explore the existing reproducibility issues of published NAS results.

Other previous works (Liu et al., 2018; Real et al., 2019) conduct comparison studies between RS with RL and EAs, showing that the latter strategies perform better in terms of final test accuracy on famous datasets as CIFAR-10 and ImageNet; however, with just small margin of error with RS.

Since 2011 (Bergstra et al., 2011), the application of Bayesian Optimization approach shows several successes with Convolutional Neural Network (in vision recognition) and automatically tuned neural network on competition datasets against human experts (Mendoza et al., 2016, 2019).

In respect to the others initial application of BO strategy to DNN, previously introduced in Section 2.2.1, the following approaches are not limited to optimize the hyperparameters, but also consider to optimize the architecture of the DNN taking into account a distance between architectures, seen as graphs, and not anymore in terms of distance between hyperparameter configurations. Indeed, DNN can be seen as a graph as reported in Figure 2.8.

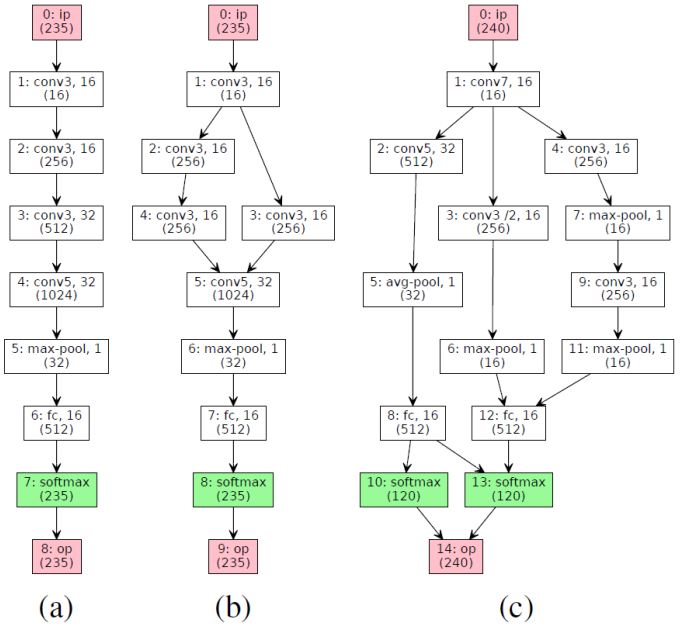


FIGURE 2.8: Three different architectures of Convolutional Neural Network represented as graphs for the same classification task (Kandasamy et al., 2018)

An initial work that tries to manage the architecture search space of NAS problem is in (Kandasamy et al., 2018). The authors proposed an optimizer named Neural Architecture Search via Bayesian optimization and Optimal Transport (NASBOT), which is able to dive in a search space composed of a set of graphs that represents the possible architectures to solve classification and regression tasks. In order to navigate during the optimization process, they developed a distance metric, named OTMANN, which can be computed efficiently via an optimal transport program, for measuring the distance between graphs (DNN), which enables the balancing of exploration and exploitation trade-off during the optimization process.
NASBOT finds better architectures for MLPs and CNNs more efficiently than other baselines on several datasets as: blog feedback (Buza, 2014), indoor location (Torres-Sospedra et al., 2014), slice localisation (Graf et al., 2011), naval propulsion (Coraddu et al., 2016), protein tertiary structure (Rana, 2013), news popularity (Fernandes et al., 2015), CIFAR-10 (Krizhevsky and Hinton, 2009). Despite promising results on the benchmark datasets in respect to other competitive optimizers (e.g., TreeBO (Jenatton et al., 2017), EAs, RS), a relevant drawback is that NASBOT requires a fine-tuning of the hyperparameters that guides the construction of the distance metric OTMANN, which is used during the optimization process to compare the distance between networks.
A more recent approach that proposes to dive in the search space of DNN architectures exploiting a graph representation of a network is proposed in (Jin et al., 2019). The authors of this work proposed the optimization framework named Auto-keras, because it is totally implemented above on the famous DL library named Keras . The optimizer of this work is based on graph-level network morphism, modifying the neural architectures based on layer-level network. In figure 2.9 is reported an application of neural network morphism, that using two operations makes it possible to reduce the differences between two different DNN architectures.

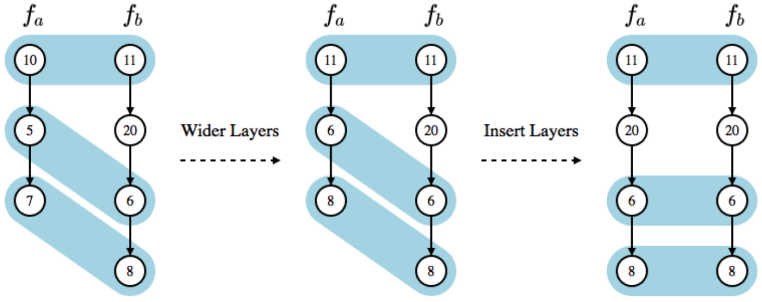


FIGURE 2.9: An example of network morphism between two neural networks f_a and f_b with the operations applied to morph one network (f_a) to the other (f_b) (Jin et al., 2019)

Auto-keras library can be applied to both classification and regression tasks with multiple optimization levels. Indeed, it can be applied not only as HPO task on the hyperparameters of a fixed architecture (e.g., number of units, activation function, and learning rate), but also as a complete NAS optimizer optimizing jointly the architecture components (e.g., type of layer) and the hyperparameters for each layer (e.g., number of units for each layer, activation function and

other hyperparameters specific for the type of layer).
Exhaustive and wide experiments on real-world benchmark datasets have been done to demonstrate the superior performance of the Auto-keras framework over the state-of-the-art methods. This approach is compared with the state-of-the-art NAS methods (Elsken et al., 2017; Kandasamy et al., 2018) on benchmark datasets of MNIST, CIFAR-10, and FASHIONMNIST, achieving the lowest error rates on all of the datasets.
Another key important aspect for this optimizer is the ability to execute in parallel on GPU or CPU, with an adaptive search strategy for different GPU memory limits. Another recent work that tries to explore and optimize into the search space of NAS problem is (White et al., 2020). The authors proposed an AutoML system named BANANAS: Bayesian Optimization with Neural Architectures for NAS. In order to adopt BO as optimizer they adopt a novel architecture representation, so-called path encoding, and a neural network-based predictive uncertainty model, so-called meta neural network defined on this path encoding representation of DNNs (reported in Figure 2.10).

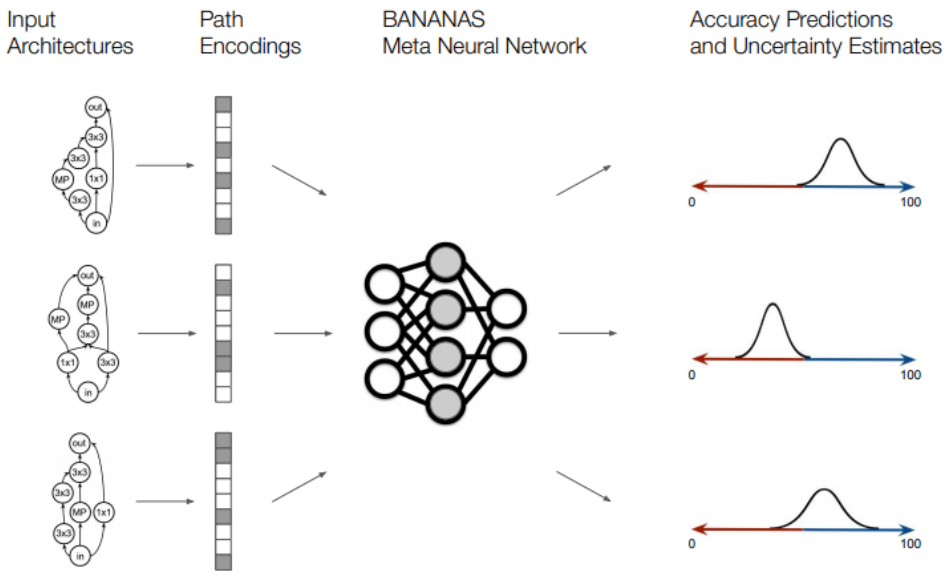


FIGURE 2.10: An illustration of the *Meta Neural Network* into BANANAS algorithm (White et al., 2020)

Differently from previous work such as NASBOT (Kandasamy et al., 2018), the authors declared that the model is powerful enough to provide high accuracy predictions of neural architecture accuracies, and for this reason there is no need to construct a distance function between architectures. In addition, they highlight the fact that the meta neural network adopted by BANANAS scales far better in respect to traditional model used in BO framework which can require significant computation (e.g., matrix inversion to construct Gaussian Processes model). From the exhaustive and wide experiments conducted by the authors in (White et al., 2020) BANANAS on the NASBench dataset outperforms the state-of-art NAS optimizer as NASBOT (Kandasamy et al., 2018), DARTS (Liu et al., 2019a), EAs (Real et al., 2019).

Chapter 3

Searching for the model with the highest generalization

Summarizing from previous Chapter, it is clear that AutoML, AutoDL and NAS, are all tools aimed at disclosing the value of ML and DL also to non-experts, making more efficient the complex process to search for an optimal ML/DL model given a dataset. Thus, we can summarize that they are aimed to maximize the generalization of the learned model.

Although optimization has always had a crucial role in learning, the black-box and expensive nature of generalization metrics, typically obtained through cross validation, have brought into vogue the adoption of derivative-free Global Optimization strategies. Learning weights (i.e., parameters) of an artificial neural network, for instance, is anyway a global optimization problem – with potentially multiple local optima – but the training loss is known in analytical form and, therefore, gradient-based algorithms are applied to update weights depending on the gradient of the training loss. Instead, searching for the optimal number of layers and neurons, type of activation function, value of learning rate, etc (i.e., hyperparameters) requires to globally optimize the validation loss, which is black-box.

This Chapter is aimed at reviewing the most relevant background about derivative-free global optimization strategies – both model-free and model-based – taking also into consideration the specificities related to the different types of search spaces occurring in AutoML, AutoDL and NAS tasks.

3.1 Generalization: a black-box, expensive and multi-extremal function to be optimized

When a data scientist, as well as a ML/DL non-expert, wants to identify an optimal ML/DL model to be fitted on a given dataset, he is interested in searching for the most generalizing model, meaning that it must provide accurate prediction also on data not used for training the model.

Cross-validation techniques, such as hold-out, k-fold cross and leave-one-out, are all well-known procedures to estimate the generalization capabilities of a ML/DL algorithm – and a specific configuration of its own hyperparameters – when trained on the given dataset. For sure, the data scientist (or the non-expert) would like to try different algorithms and/or hyperparameters settings and, if he does not know AutoML, he will search by a "trial-&-error" procedure, according to his experience. Moreover, cross-validation might require a huge amount of resources, especially time for training and validating all the models, while our data-scientist (or non-expert) would like to obtain an accurate model as soon as possible. Always supposing that he does not know AutoML, the best idea he can have is to fix a given number of algorithms and hyperparameters settings to test ("budget") and then exploit the embarrassing parallelism of a simple Grid Search. Assuming he has as many computational resources as the configurations of the grid, he can run all of them in parallel, solving the issue about time. However, it has been largely proved that Grid Search is not sample efficient, meaning that other global optimization strategies – analysed in this Chapter – can identify a better solution, by using the same number of configurations into the grid. This means that, at the end, the model selected by our data-scientist (or non-expert) could be far worse than the one obtained by a clever data scientist using AutoML, properly.

Just to summarize, in AutoML we are interested to optimize a generalization metric, based on a cross-validation procedure, with respect to alternative algorithms and their own hyperparameters (as well as preprocessing and postprocessing methods).

This means that our metric is:

- Black box
- Expensive to evaluate
- Multi-extremal

that is exactly the derivative-free global optimization problem:

$$x^* = \arg\min_{x \in X} f(x) \tag{3.1}$$

In the cases seen in previous Chapter 2, $x = \gamma$ for the Hyperparameter Optimization (Section 2.1.2), while, in the most general case of CASH (Section 2.1.3), x is a vector whose first component is the decision variable associated to the ML algorithm to choose in the set A and the remaining components are the associated hyperparameters γ.
In addition, in AutoML community not so long ago, the landscape of loss function was considered as multi-extremal function as the example reported in Figure 3.1. However, it has been discovered in recent studies (Pimenta et al., 2020; Pushak and Hoos, 2018) that the landscape of the loss function of an AutoML problem turns out to be predominantly uni-modal and often convex as it was otherwise previously assumed by the scientific community.

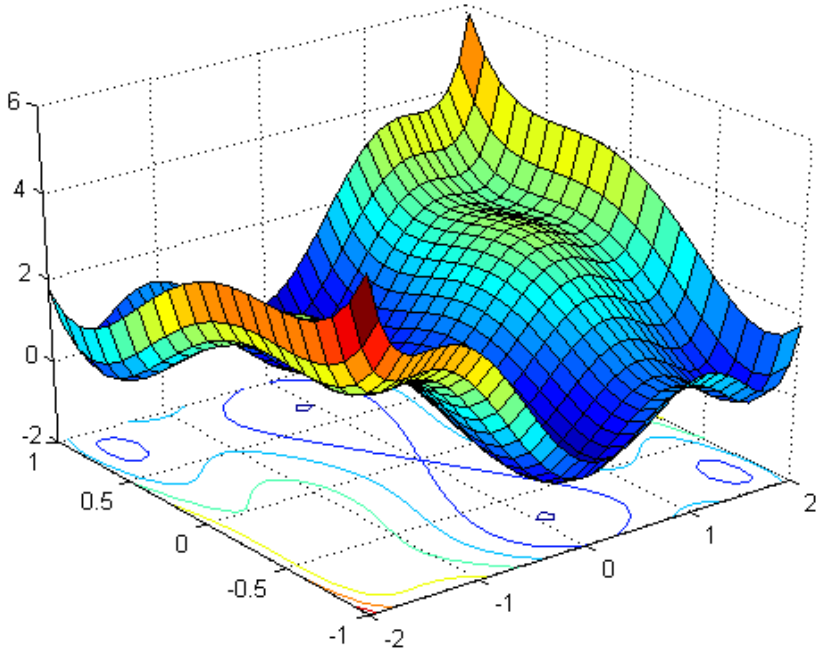


FIGURE 3.1: A example of multi-extremal two-dimensional function

3.2 Types of search spaces

In the Global Optimization (GO) community many test functions have been proposed to evaluate novel optimization algorithms proposed by the scientific community. These functions are diversified in terms of smoothness, number of local and global optima, and domain in which they are optimized. This Section is specifically devoted to domain, that consists of a number of variables (i.e., dimensions) potentially of different nature. The most common test functions are defined on a domain usually characterized by continuous and integer variables. From a GO point of view, domain represents the search space where the optimal solution has to be searched

for, formally:

$$X = \Gamma_1 \times \Gamma_2 \times ... \times \Gamma_n \tag{3.2}$$

where each $\Gamma_i, i = 1, .., n$, can be bounded in integer or continuous interval of values.
In the most general case, Γ_i can also be a set of discrete values, sortable (i.e., ordinal) or not (i.e., categorical). When X is composed by the product of discrete variables only, the problem becomes a *combinatorial* problem. More complex problems are characterized by a search space spanned by continuous, integer and categorical variables.
A typical example is AutoML, aimed at selecting both the Machine Learning algorithm (i.e., represented by a categorical variable) and the optimal configuration of its own hyperparameters (i.e., represented by continuous and discrete variables). Furthermore, the search space for AutoML is even more complicated because some hyperparameters are (*active*) depending on the value of other hyperparameters. This *conditioning* among hyperparameters - that are dimensions of the search space - leads to a *hierarchical search space*.
More specifically, in AutoML, selecting the ML algorithm among a set of possible alternatives activates all the variables of the hierarchical search space associated to the hyperparameters of the selected algorithm and deactivates all the other variables related to hyperparameters of the other ML algorithms.
Although this new formalization of the search space better supports AutoML tasks, have flaws due to the difficult application for specific problems related to NAS, where a high number of conditional variables is present. In fact, in literature, it has been demonstrated with several works (Jin et al., 2019; Kandasamy et al., 2018) that a *graph* can better represent the search space associated to a NAS task. Thus, NAS required to introduce a new formalization of the search space for the GO problem, defined by all the graphs associated to the possible neural architectures to consider. This leads to the need for defining suitable measures to evaluate similarity between two graphs/architectures. An example of similarity distance between graphs/architectures is OTMANN distance proposed in (Kandasamy et al., 2018), where the distance can be computed efficiently via an optimal transport program. Moreover, another distance to compare different graphs/architectures is based on network morphism. Indeed in (Jin et al., 2019), inspired by Deep Graph Kernels (Yanardag and Vishwanathan, 2015), the authors proposed an edit-distance based on how many operations (e.g. inserting a layer, skipping a layer) are needed to morph one graph/architecture to another one.
Another relevant aspect to take into account when defining the search space is the size of the range/set of values for each variable/dimension. The larger the size of the range/set of each variable the larger the search space and, consequently, it becomes more difficult to retrieve the global optimum within a limited number of trials.
This situation can be controlled by the careful definition of these intervals for AutoML tasks. However, since AutoML tasks should be used by people who do not necessarily have ML skills, it is necessary to define a search space that adapts according to the evaluations assumed during

the optimization process.
For this reason, *weakly search spaces* are defined that allow for expansion and contraction depending on the needs of the task on which they are applied.
In particular in (Nguyen et al., 2017) they proposed an optimization framework based on weakly search space, which increases its size depending on the needs of the optimizer.
Another optimization system that supports the weakly search space is *Aisaratuners*[1]. Aisatuners adopts Latin Hypercube Sampling to generate initial samples through which AiSara Hyperparameter Tuning API optimizer uses state-of-the-art pattern recognition algorithms to reduce the search space boundaries during optimization process. By reducing the search space, the system speeds up the whole optimization process while improving the optimum value achieved.
Another good strategy to obtain a more descriptive and useful search space is inferring an initial good solution by transferring knowledge from other, already solved, black-box functions. In (Perrone et al., 2019) the authors proposed an approach to automatically design the search space of a black-box problem in order to speed up the optimizer. Despite the considerable boost of the optimizer, achieved by reducing the size of the search space, this approach cannot guarantee to define a good search space without any knowledge of the associated black-box problem.

3.3 Searching through model-free global optimization strategies

This section is devoted to the so-called *model-free* global optimization strategies. Basically, they assume that all the information is contained into the function evaluations. On the contrary, the *model-based* strategies (analysed in Section 3.4) try to approximate the objective function depending on the evaluations, with the idea that this approximation – even if biased by some assumptions about function "smoothness" – can provide more information to make the optimization process more efficient.

Following Subsections will analyse the most well-known model-free global optimization strategies, that are:

- Pure and Adaptive Random Search
- Evolutionary Approaches (which are proved to be a special case of Random Search)
- Deterministic (Lipschitz) Global Optimization (which uses a bit of additional "structural" information about the objective function, regarding its maximum slope. This additional information can be given a-priori or estimated along the optimization process).

[1] https://pypi.org/project/aisaratuners/

3.3.1 (Pure/Adaptive) Random Search

Random Search (RS) methods have been considered since the earliest studies in GO. A masterful analysis of the statistical issues associated to RS is contained in (Zhigljavsky, 1985). RS is both a Global Optimization method itself and the basis of many other methods, including Bayesian Optimization, due to its exploration properties. Its use has been advocated for Hyperparameter Optimization (Bergstra and Bengio, 2012), and Reinforcement Learning (Mania et al., 2018).
According to the general RS framework, a sequence of random points $x_1, x_2, \ldots, x_j$ is generated according to a *probability density function* (*pdf*) which might, in general, depend on previous points x_j and the associated values $f(x_1), f(x_2), ..., f(x_j)$. Different computational strategies belong to this general framework. In the basic RS, the points $x_1, x_2, \ldots, x_j$ are sampled from a uniform distribution over the search space and the sequence of points is stopped after a prefixed number of function evaluations or when some statistical test is satisfied. Some RS techniques have been proposed with the aim to cover the search space and they are usually adopted to generate an initial set of function evaluations, aka initial *design*, for others Global Optimization strategies, such as Bayesian Optimization, as well as other meta-heuristics.
Latin Hypercube Sampling (Huntington and Lyrintzis, 1998) is a points filling space technique that allows to cover, as much as possible, the whole search space. Due to its not-completely random nature, this method is also known as Near-Random Search.
Another specialization of RS is when sampling from uniform distribution is performed within a specific portion of the search space, at each iteration j.
This portion is defined as a *level set* A_j:

$$A_j = \{x \in X : f(x) \leq f(x_j^+)\} \tag{3.3}$$

where $f(x_j^+)$ is the "best seen" among the sequence samples $x_1, x_2, \ldots, x_j$ and x_j^+ is usually also known as *incumbent* (solution). The resulting algorithm, called Pure Adaptive search (Patel et al., 1989; Zabinsky and Smith, 1992; Zhigljavsky, 1985) has a good theoretical convergence but uniform sampling over a generic A_j is significantly more complicated than over an box-bounded space.
Another approach using level sets is the Probabilistic Branch & Bound (PBnB), which significantly increases performances compared to pure RS (which it is based on) (Huang and Zabinsky, 2013; Zabinsky and Huang, 2020; Zabinsky et al., 2011). PBnB is a partitioning-based RS approach for simulation-optimization problems and designed for optimizing both noisy and noise-free functions over mixed continuous-integer domains. The main goal of the algorithm is to approximate a user-defined target level set: PBnB iteratively *maintains*, *prunes* and *branches* sub-regions to approximate the unknown target level set with a probabilistic guarantee of accuracy. At the end of the optimization process PBnB identifies sub-regions of the search space with a relatively high concentration of high quality solutions, and returns an estimation of the

global optima. Figure 3.2 shows the results of three test functions applying PBnB, and it can be seen that the approach converges to evaluate points of the functions in the most promising (green) regions.

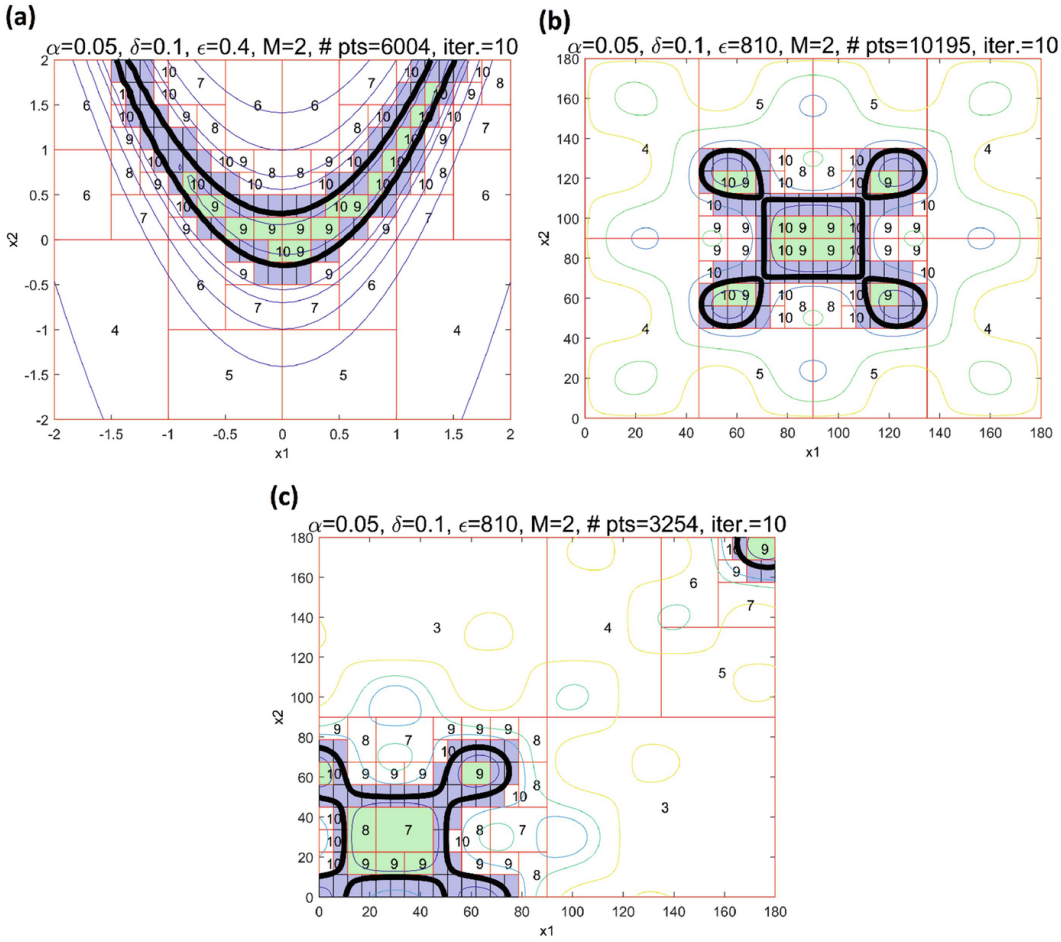


FIGURE 3.2: Approximating the level set bounded by the 0.1 quantile on the tenth iteration of PBnB for two-dimensional (a) Rosenbrock's function, (b) centered sinusoidal function, and (c) shifted sinusoidal function by the maintained green (light gray) subregions (Zabinsky and Huang, 2020)

Methods based on RS can be further improved through a local search mechanism, according to the following procedure:

(i) Draw q points from a uniform distribution in X, $\{x_j \in X\}_{j=1,\ldots,q}$ and observe $\{f(x_j)\}_{j=1,\ldots,q}$

(ii) Select the "most promising points" and start from them a local optimization routine obtaining at least value $f(x_j^+)$

(iii) Test whether $f(x_j^+)$ is the global minimum of f in X.

The above cycle is repeated until the stop criteria is met.

In this very simple implementation of the algorithm, the point(s) yielding the best sampled value

is used in *(ii)* as starting point(s), and the test *(iii)* is satisfied if no improvement over $f(x_j^+)$ (the current best seen) is observed in one (or more) further executions of phase *(i)*.

This simple test *(iii)* relies upon the fact that, as the sample size increases, the probability of not improving over $f(x_j^+)$ decreases unless $f(x_j^+)$ is the global minimum. In this method no more than one local optimization is performed in the region of attraction of a minimum: this good feature, unfortunately, is obtained at the cost of wasting most of the information contained in all the other samples.

More effective algorithms require a balanced compromise between the conflicting goals of making good use of the available information (exploitation) and reducing the risk of converging to a local minimum (exploration). An effective way to reach a better trade-off (exploitation versus exploration) is based on cluster analysis, which allows to predict whether a starting point, for the local search, is likely to lead to a new local optimum or to one that has already been discovered. In this way the local searches are started at promising candidate points (Zabinsky and Huang, 2020).

Moreover, the effectiveness of the general three-steps procedure can be increased by using *multistart*. A recent theoretically important result about *multistart* RS is Multistart with Early Termination of Descents (METOD) (Žilinskas et al., 2019) where it's argued that multistart is the best solution for high-dimensional GO problems. The key result states that, for a quadratic function, given two arbitrary points, the trajectory of the gradient method force the two points closer to each others. The authors generalize this result by restricting the computation of derivatives only in a very few directions, different at each iteration. This step might be likened to the Stochastic Gradient Descent (SGD) step. In order to endow METOD with intelligent stopping rules the authors resort to the method suggested by (Zieliński, 1981). METOD is definitely a major advance in the topic of multistart methods but still has limitations: *(i)* the search domain must have a simple structure, *(ii)* the computation of the objective function and its derivatives must not be expensive, and *(iii)* the function must have not too many local optimizers. These limitations make this approach not fully suitable to AutoML and NAS. In particular, (Zieliński, 1981) suggested a stopping rule based on a statistical estimate of the structure of multi-extremal problems.

Given the important role of RS in all stochastic algorithms both theoretically, in the convergence proofs, and in actual computations, in combination with sequential algorithms, it is important to analyze the effectiveness of uniform sampling. This is particularly important in view of the "pathologies" i.e. unexpected results connected with high dimensions. The following arguments are drawn from the work in (Pepelyshev et al., 2018), whose results have been proven for a general distribution:

$$P_j = \alpha_j P_U + (1 - \alpha_j) Q_j \tag{3.4}$$

where $0 \leq \alpha_j \leq 1$, P_U is the uniform distribution and Q_j is a specific probability measure on X

which usually depends on evaluations of the $f(x)$ at the previous points. Sampling from the distribution (3.4) corresponds to taking a uniformly distributed random point in X with probability α_j and sampling from Q_j with probability $1-\alpha_j$.

Let $\varepsilon, \delta > 0$ to be fixed and define $W(\delta) = \{x \in X : f(x) - f^* \leq \delta\}$ and $B(x^*, \varepsilon) = \{z \in X : |z - x^*| \leq \varepsilon\}$. Assume that B as $B = B(x^*, \varepsilon)$ if we are interested in the convergence to x^* and $B = W(\delta)$ if the convergence is to f^*. As x_j are independent we have $Pr\{x_1 \notin B, x_2 \notin B, \ldots, x_n \notin B\} = (1 - P(B))^n$ and then the probability that at least one is in $B = 1 - (1 - P(B))^n$ tends to 1 if $n \to \infty$.

To derive a finite sample bound let:

$$n_\gamma = min\{n : 1 - (1 - P(B))^n \geq 1 - \gamma\} \tag{3.5}$$

we obtain

$$n_\gamma = \lceil \ln\gamma / \ln(1 - P(B)) \rceil \cong (-\ln\gamma)/P(B) \tag{3.6}$$

since $P(B)$ is small and $\ln(1 - P(B)) \cong -P(B)$ for small $P(B)$. The numerator is not too large, but the denominator is very small and n_γ is astronomically large. Let $X = [0,1]^d$ assume the uniform distribution P_U and the set $B = B(x_*, \varepsilon)$ then $Vol(B) \leq V_d \varepsilon^d$, where V_d, the volume of the unit ball, is $V_d = \pi^{\frac{d}{2}} / \Gamma(\frac{d}{2} + 1)$.

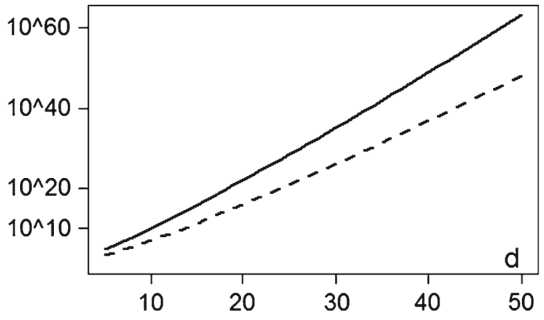


FIGURE 3.3: The number n_γ of points which are required for Pure RS (PRS) to reach the set $B = B(x_*, \varepsilon)$ with probability at least $1 - \gamma = 0.95$ for $\varepsilon = 0.1$ (solid) and $\varepsilon = 0.2$ (dashed) as the dimension d varies in $[5, 50]$ (Pepelyshev et al., 2018)

If $B = W(\delta)$ and we know the Lipschitz constant of f the relation translated into a probabilistic error bound on the error $f(x) - f^*$.

3.3.2 Evolutionary & Nature-inspired Meta-heuristics

Meta-heuristics represents a broad class of nature-inspired optimization methods. In particular, in terms of evolutionary biology, natural selection is an example of how a specie survives and evolves in respect to others due to the ability of adapting to the environment where it lives. This process of a biological nature inspired the so-called Evolutionary Computation (EC) algorithms.

One of the most famous EC algorithm are the Genetic Algorithms (GAs), originally proposed in (Holland, 1975), which try to pursue two objectives: the abstraction of the adaptive processes in natural systems, and the design of artificial systems which would take advantage of the adaptive processes abstracted from natural systems. Traditionally GAs are introduced in (Holland, 1975) to solve combinatorial optimization problems.

The main idea behind GAs is to represent a population of candidate solutions (also called individuals) as strings of bits (usually called chromosomes), and evolve them by means of stochastic variation operators, which are usually called genetic operators. The new solutions are successively evaluated against a fitness function (aka objective function), which generally represents the objective function to be optimized, and solutions with a higher fitness have a higher chance of being selected for the next iteration. The process is repeated until a certain number of generations (aka iterations) is reached, or another termination criterion is met.

This approach is defined as more robust than single-state optimization methods (e.g., Gradient Descent, Hill Climbing) due to not working directly on the parameters of the candidate solutions, (e.g., fixed-length bit-string representation). In addition, this approach during the optimization process evaluates in parallel a population of candidate solutions, and also does not require necessary additional information of the search space (e.g., derivates) although it can help the convergence to the global optimum. A final advantage, no less important than the others previously mentioned, is that GAs adopt probabilistic operators to change the populations of candidate solutions during the optimization process, this characteristic leads to different results that are obtained using deterministic operators (e.g., step) used by Gradient Descent or Hill-Climbing.

In fact, in the generational version of GAs, the population is simply replaced by the new offspring for the next generation. This means that the selection and crossover operators are applied enough times to create a number of offspring candidate solutions (aka individuals) equal to the current population size, in order to keep the population stable. Of course, by just replacing the old population with the new one, the individuals with the best fitness in the old population will not survive to the next generation. Hence, if the best offspring of individuals has a fitness value which is lower than that of the best individual in the old population, the GAs will actually produce a worse candidate solution.

In order to solve this problem, generational GAs are usually coupled with an elitist strategy: if the best individual in the new population is not better than the best individual in the old population, then use the latter to overwrite one of the individuals in the new population. In this way, the best individual is always preserved, and its fitness is a monotone non-decreasing function of the number of GAs iterations.

Indeed, based on the fact that GAs require the encoding of decision variables of the problem under optimization at genotype level (with chromosomes), a wide portion of meta-heuristic community started to use Evolutionary Algorithms (EAs) that allow using directly the decision variables of the problem at phenotype level, which avoid the effort of encoding the decision

variables in a very similar set up of population-based approach. With the adoption of EAs, it is possible to keep the benefits of the GAs, in terms of exploration of the search space of the problem, without losing decision variables information.

In addition, another important approach in meta-heuristics algorithms is Covariance Matrix Adaptation – Evolution Strategies (CMA-ES) (Hansen et al., 2003). One of the most important advantages of CMA-ES is to be particularly useful if the fitness function (aka objective function) is ill-conditioned, so when associated to a small change in a candidate solution there is a large change in the value of the fitness function.

Another branch in the meta-heuristics is nature-inspired algorithms based on animals and insects' behaviours to solve a specific task. In the community of meta-heuristics, the behaviours of several species of animals have been taken into account to propose novel, more efficient and effective optimization algorithms.

Firefly Algorithm (FA) proposed originally in (Yang, 2008, 2009), simulating the flashing patterns and behaviour of the fireflies, seems to be the most efficient and effective in respect for hundreds of others. Indeed, it showed its ability to deal with efficacy on the intensification-diversification trade-off (aka exploitation-exploration trade-off) a common issue in global optimization problems. The term diversification means to generate different solutions in order to explore the search space on the global point of view, while intensification means to focus on more promising local region by exploiting the current local information provided by the good solutions found in this region.

This algorithm is based on three fundamental rules of fireflies' life: *(i)* They are unisex so that one firefly will be attracted to other fireflies regardless of their sex, *(ii)* The attractiveness is proportional to the brightness, and they both decrease as their distance increases. Thus for any two flashing fireflies, the less bright one will move towards the brighter one (in case there is not a brighter one between a couple of fireflies, they move randomly) *(iii)* the landscape of objective function determines the brightness of a firefly.

In a very synthetic way, the definition of the movement of a firefly to another more attractive (brighter) one is determined by the following formula:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha_t \varepsilon_i^t \tag{3.7}$$

Where the first term represents the old position of firefly (x_i^t), the second term is due to the definition of attraction at a specific r distance (typically Euclidean distance) between fireflies, represents the intensification component into the algorithm using a γ decay factor to reduce the impact of attraction. The last term allows to perform the diversification component into the algorithm for escaping from local optimums. Indeed α_t is a randomization parameter and ε_i^t is a numeric random vector sampled by a Gaussian distribution or by uniform distribution at a specific time t.

Moreover, in nature, animals-inspired algorithms also exist in another particular branch of metaheuristics, so-called Swarm Intelligence. This particular optimization paradigm is guided by the collective intelligence of a group of simple individuals with limited capacities. Indeed, several times in nature, intelligent behaviour emerges from the collective effort of multiple similar agents to converge to the optimal solution of a complex problem. There are several algorithms based on this swarm intelligence paradigm, such as Particle Swarm Optimization, Ant Colony Optimization, and Artificial Bee Colony.

In particular, Particle Swarm Optimization (PSO) is one of the most famous Swarm Intelligence algorithm, firstly proposed in (Kennedy and Eberhart, 1995). The algorithm is inspired by the collective movement of animals (e.g. flocks of birds, schools of fish). This approach is based on two main elements: a population (the swarm) of N individuals (the particles). The swarm intelligence algorithms are based on information exchange about promising solutions (individuals – particles) during the optimization process.

PSO has multiple settings in terms of hyperparameters, such as social factor which modulates the attraction towards the global best (local exploitation), Cognitive factor which modulates the attraction towards the personal best (global exploration), and Inertia weight which balances the aforementioned settings: low inertia helps the local search; high inertia helps global search.

There are several works (Abraham and Liu, 2009; Nobile et al., 2015, 2018; Shi and Eberhart, 2001; Tian and Li, 2009) that try to automatize, through fuzzy logic, the choices of PSO's hyperparameters during the optimization process, and additionally in (Senthilnath et al., 2011) they demonstrated that PSO can be considered as a special case of FA, when its hyperparameter related to discount factor of the attraction between fireflies equals $\gamma = 0$.

Another famous Swarm Intelligence algorithm is the Ant Colony Optimization (ACO) (Dorigo et al., 2000), where the optimization process is guided by the collective intelligence of an ant colony that try to find out food in the world (into the search space of the objective function).

This algorithm was initially created to solve combinatorial problems (e.g., Traveling Salesman Problem). ACO algorithm is mainly based on the stigmergy, which is a mechanism of indirect coordination between agents (e.g., ants) mediated by the environment. More in detail, stigmergy is a form of self-organization that produces complex, seemingly intelligent structures without any planning, control, or even direct communication between the agents. Indeed, a quiet intuitive example of stigmergy in nature is the ants' pheromone trails, where the ants, during the search of food, leave pheromone in their paths in order to suggest to other ants the chosen path. More ants chose the path with the stronger pheromone trail to reach the food as it entails a higher probability of find it. However, the pheromone gradually decreases its intensity (evaporates) over time. So essentially, through the time only survives the path with a stronger signal of pheromone.

It should be noted that as for other metaheuristics, the ACO algorithm has to set several hyperparameters to increase the possibility of converging to the global optimum of the problem under optimization.

Furthermore, another important Swarm Intelligence algorithm is Artificial Bee Colony (ABC), firstly proposed in (Karaboga and Basturk, 2007). This algorithm emulates the foraging behaviour of honey bees, where "food" and "nectar" are metaphors for candidate solutions and fitness values (objective function values). This algorithm is founded on three fundamental roles: employed bees are assigned to source food and gather information about the region, onlooker bees are distributed to the food sources, proportionally to nectar amount, and finally the scouts bees that perform random search in the search space, looking for new food sources. After several iterations the algorithm, based on the previous roles, through the bee colony identifies the optimal food sources (aka optimal solutions).

Instead, an example of metaheuristic based on searching optimal state of a physical system is Simulated Annealing (SA) firstly presented in (Van Laarhoven and Aarts, 1987). In chemical bonds, the optimal state (optimal solution) is the one that minimizes the potential energy of the electrons shared between two atoms, so to create a stable bond. SA is a single-state optimization method which tries to avoid local optima by adopting the following principle: occasionally, accept to replace the current candidate solution with a worse one in terms of objective function value. The "occasionally" part is quantitatively expressed by an acceptance probability, which in SA progressively decreases over time. Intuitively, worse solutions are more likely to be accepted in the earliest steps of the optimization process, and less likely in the last ones. In this way, SA prefers exploration in the first part of the optimization, while after a certain number of evaluations of the objective function the algorithm exploits the neighbourhood of the current solution. The name "annealing", in fact, derives from the process of metal annealing, where a piece of metal is initially brought to a high temperature, thereby loosening the bonds of its atoms, and then cooled down. If the cooling down happens too abruptly, however, the atoms do not settle on a stable lattice, resulting in a fragile piece of metal. For this reason, the metal is cooled down slowly.

In particular, the acceptance probability of modifying x in a worse solution x' is defined as:

$$P(x,x',T) = e^{-\frac{|f(x)-f(x')|}{T}} \tag{3.8}$$

where f is the objective function, and T is the temperature parameter that is gradually decreased during the execution of the SA algorithm. There are several ways for defining a temperature decrease scheduling, one of the most common one being the linear scheduling: formally, given a cooling parameter $\alpha \in (0,1)$, the temperature is modified as $T \leftarrow \alpha \cdot T$.

In conclusion metaheuristics have some drawbacks, including usually a high number of hyperparameters that need to be tuned for obtaining optimal performances and the absence of rigorous proofs of the global convergence of these approaches. Indeed, there is another branch of the global optimization community that makes the proofs of global convergence a key factor of the optimization strategies, which are the so-called deterministic approaches and that will be introduced in the next section.

3.3.3 Deterministic (Lipschitz) Global Optimization

Different from the previous Sections (3.3.1, 3.3.2) there is another branch of the Global Optimization community that tries to converge to the optimal solution of problem 3.1 based on a deterministic paradigm which has strong and solid proof of convergence rate based on mathematics fields, called Deterministic Global Optimization.
This branch of GO is strongly defined based on the assumption that some information on the structure of objective functions is known. In particular, to adopt this kind of optimization approach is required that the objective function f respects the Lipschitz condition as follows:

$$|f(x') - f(x'')| \leq L||x' - x''||, \quad x', x'' \in X \tag{3.9}$$

where $||\cdot||$ denotes a distance metric on X (e.g., Euclidean distance) and L is the (unknown) Lipschitz constant such that $0 < L < \infty$. The information of L Lipschitz constant can represent the mathematical expression to define the bounded rate of variation of the objective function f, and in Figure 3.4 (a) can be seen the geometric representation of this constant on f.
The assumption to know a priori the information related to L constant for real-life applications is justified at least in technical systems, where the rate of change is typically limited by the physical laws of the underlying problem (Strongin, 1978, 1992).
More in detail in the Figure 3.4 (b) can be seen, in one dimension, that L constant allows to establish a lower bound $\varphi_i(x)$ (aka *minorant*) of the objective function f into the interval $\left[x_{i-1}, x_i\right]$ as follows:

$$\varphi_i(x) = \max\{z_{i-1} - L(x - x_{i-1}), z_i + L(x - x_i)\} \tag{3.10}$$

where z_{i-1} and z_i are the evaluations of the points x_{i-1} and x_i on f. The minimum value of $\varphi_i(x)$ can be considered as a reasonable suggestion for the next evaluation point in the sequential optimization process. In a sense, $\varphi_i(x)$ gives a worst estimate of the uncertainty of the objective function.

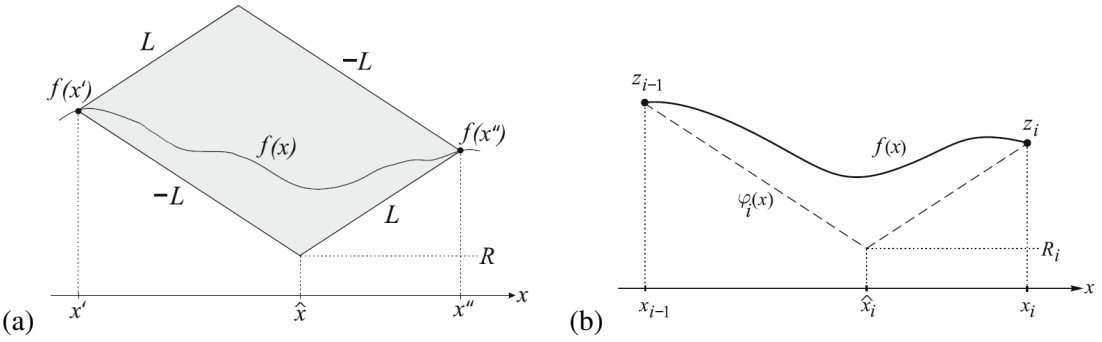


FIGURE 3.4: A graphic interpretation of Lipschitz condition (a) and a geometric representation of Lipschitz condition (b) on a one dimensional f (Sergeyev and Kvasov, 2017)

These approaches can be extended to the multi-dimensional case, bearing in mind that finding the minimum of $\varphi_i(x)$ is a multi-extremal problem and in the worst case the number of evaluations is anyway exponential.

The restriction of the class of the objective function to the differentiable functions with the first derivative satisfying a Lipschitz condition allows to develop efficient geometric LGO methods: $\theta_i(x)$ is the new lower bound, built on smooth piecewise quadratic auxiliary functions $\psi_i(x)$, which speeds the search process up.

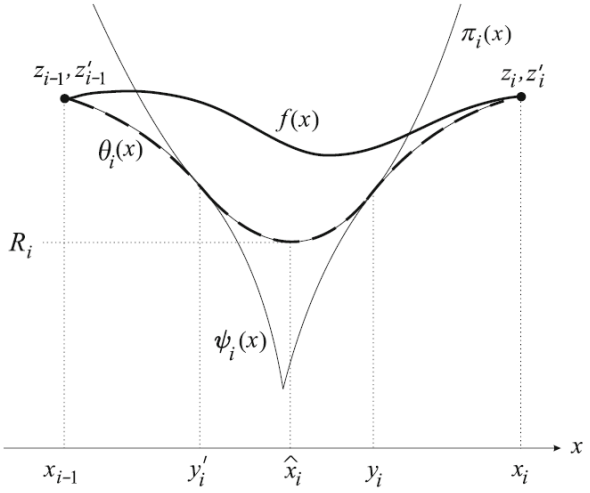


FIGURE 3.5: Smooth piecewise quadratic auxiliary function $\theta_i(x)$ (dashed line) for the objective function $f(x)$ (thick line) with the Lipschitz first derivative over $[x_{i-1}, x_i]$ (Sergeyev and Kvasov, 2017)

An important improvement in LGO was proposed in (Jones et al., 1993) under the name of DIRECT (Dividing RECTangles). This algorithm can estimate the Lipschitz constant used at each iteration of the optimization process. However, the main issue related to this approach is how to obtain the value of L. Many approaches have been proposed, and a complete overview of them is given in (Sergeyev and Kvasov, 2017). The most interesting LGO, based on the DIRECT algorithm, enables an adaptive estimation of L, yielding either a global estimate of the Lipschitz constant or a local estimate valid for some regions. With multiple values of L (global and local), it can be possible to speed up the entire optimization process better dealing with exploration-exploitation trade-off in different regions of the problem's search space.

Recently several works propose to merge the Lipschitz optimization with other global optimization strategies such as model-based optimization. Such as in (Ahmed et al., 2020; Jalali et al., 2013) exploits the L Lipschitz constant or at least an estimation to speed up the entire sequential optimization process avoiding regions of the search space where is not promising to get closer to the global optimum.

3.4 Searching by learning & optimizing: model-based global optimization strategies

To solve a black-box problem, there are not only model-free optimization strategies; another optimization branch that aims to solve black-box problems is model-based optimization strategies. The main difference between the previous optimization strategies and the strategies introduced in this section is that model-based strategies can replace the black-box objective function with an approximate and less expensive one during the optimization process. In this way, the number of evaluations of the objective function is drastically reduced compared to model-free optimization strategies.
These model-based strategies are essentially divided into two different branches that adopt different surrogate models: *deterministic* models and *probabilistic* models.
In this section, deterministic models will be introduced first, and then the pros and cons of adopting this particular type of surrogate will be analyzed. Then, taking into account the limitations in using deterministic models, the most advanced probabilistic models will be introduced, which allow to define better optimization strategies, considering the probability of the uncertainty of the approximation of the surrogate models during the optimization process.

3.4.1 Deterministic Surrogates

The deterministic surrogates models are known in Derivate-Free Optimization as a response surface models introduced by George E. P. Box and K. B. Wilson in 1951. The main goal of these models is to find out the relationships between the explanatory variables, aka the dimensions of the problem under optimization, and the one or more response variables which can be represented by the value or values of the objective functions.

Linear Regression.
Linear Regression is one of the most simple and famous deterministic model. It is based on the fact that the objective function can be modelled as a linear combination of the input variables as:

$$f(x) = w_0 + \sum_{j=1}^{d} x_j w_j \tag{3.11}$$

where x is a vector of size d; d is the number of input variables; w is the vector of length $d+1$. To obtain optimal w vector the following formula has to minimized:

$$\min_{w} ||Xw - y||_2^2 \tag{3.12}$$

which can be seen as an unconstrained minimization problem of the sum of squared error between the actual data y and the predicted value Xw, where X is a matrix of size n by $d+1$

which represents the sample points in the input space and the w is the weight vector to allow the approximation of the black-box objective function.
Although linear regression is not the best choice to optimize functions that are supposed to be highly non-linear, it remains a widely adopted method due to its simplicity and its non-linear extensions such as *piecewise linear regression* and *local linear regression*. Other non-linear models are analyzed in the following subsections.

Support Vector Regression.
Support Vector Regression (SVR) surrogates (Smola and Schölkopf, 2004) are more sophisticated than linear regression, indeed they are represented as the weighted sum of basis functions added to a constant term.
A traditional formalization of SVR surrogate as given by:

$$f(x) = \mu + \sum_{i=1}^{n} w^i \psi(X, X^i) \tag{3.13}$$

considering as a simple basis function $\psi(\cdot) = X$, the surrogate can be formalized as:

$$f(x) = \mu + w^T X \tag{3.14}$$

to obtain the unknown parameters μ and w in the model, the following equation has to be minimized:

$$\min \frac{1}{2}|w|^2 + C\frac{1}{n}\sum_{i=1}^{n}\left(\xi^{+(i)} + \xi^{-(i)}\right) \tag{3.15}$$

such that:

$$wx^i + \mu - y^i \leq \varepsilon + \xi^{-(i)} \tag{3.16}$$

$$y^i - wx^i - \mu \leq \varepsilon + \xi^{+(i)} \tag{3.17}$$

$$\xi^{+(i)}, \xi^{-(i)} \geq 0 \tag{3.18}$$

fixing the constraints at 3.16 and 3.17 allows the sample points to lie within $\pm\varepsilon$ deviation from the value at sampled points without affecting the surrogate model. This band of allowed deviation is referred to as ε insensitive tube. Slack variables $\xi^{+(i)}$ and $\xi^{-(i)}$ ensure feasibility of the problem by allowing outliers that do not fall within ε insensitive tube. Trade-off between model complexity and fit is achieved by penalizing outliers by a predefined constant $C \geq 0$. Combined contribution of the model complexity and the penalty for outliers (3.15) is minimized.
The version as mentioned earlier of the SVR model, is defined under the assumption to have a linear basis function. However, it is possible to consider different basis functions that might require determining additional hyperparameters associated with that specific basis function (Smola and Schölkopf, 2004).

Radial Basis Functions Regression.
Radial Basis Functions Regression (RBF) is another type of deterministic model, based on the sums of several radial basis functions, this model is typically used to point-wise approximation of a black-box function and can be formalized as follows:

$$f(x) = \sum_{i=1}^{n} \lambda_i \phi(||x - x_i||_2) + p(x) \tag{3.19}$$

where $\lambda_1, ..., \lambda_n \in \mathbb{R}$ are the weights to be determined; $||\cdot||$ is the Euclidean norm; $\phi(\cdot)$ is the basis function. In literature several basis functions exist such as:

- *Linear*: $\phi(r) = r$
- *Cubic*: $\phi(r) = r^3$
- *Gaussian*: $\phi(r) = e^{-(\gamma r)^2}$
- *Multi-quadratic*: $\phi(r) = \sqrt{r^2 + \gamma^2}$
- *Thin plate spline*: $\phi(r) = r^2 \ln(r)$

where $r = ||x - x'||$.
None of the different types of basic function has proven to be better than the others, indeed the choice of the basic function may depend significantly on the type of black-box function that has to be approximated.

Kriging Regression
Kriging Regression is based on a model that shapes the response surface model as a sum of two components, and an additional error term.
The general form of the kriging regression model is:

$$f(x) = g(x) + Z(x) + \varepsilon(x) \tag{3.20}$$

where the first component $g(x)$ is the polynomial function which represents the global trend of the surface to regret, the second component $Z(x)$ is a gaussian random process (with mean zero and stationary non-negative covariance) which represents the local deviations of the estimated surface from $g(x)$. While the last component $\varepsilon(x)$ is the random error with zero mean and covariance which is the error assumed in the quadratic polynomial approximation.
The strength of this regressor is the use of a gaussian random process. Through this process the kriging regressor is able to be more precise in the objective function approximation, exploiting the correlation between the x points used to construct the approximated surface and all the points present in the domain of the function to approximate.

A detailed description of gaussian random process will be introduced in the next section.

Decision Tree Regression.
Decision Tree (DT) (Mitchell, 1997) is a Machine Learning algorithm which is applicable on both classification and regression task. Based on the Divide-et-Conquer paradigm, the original task problem can be decomposed in subsets of sequential problems/decisions that have to be taken in order to perform the task, which can be classification or regression. Of course in regression tasks this ML algorithm is able to produce an approximation of an objective function exactly as the others deterministic surrogate models.
For regression task are used different node split criteria such as the *Chi-Square* and *Variance Reduction* measures to ensure the reduction of the error in regression estimations.
In case of Chi-Square the main goal is to have the mean of the dependent features (p), which must be different from their child node. Instead in case of Variance Reduction criteria the main goal is to reduce the variance of the features in respect to the target value (independent variable) to predict. The Variance Reduction of a node N is defined as the total reduction of the variance of the target variable x due to the split at this node, and can be formalized as follows:

$$I_V(N) = \frac{1}{|S|^2}\sum_{i\in S}\sum_{j\in S}\frac{1}{2}(x_i - x_j)^2 - \Big(\frac{1}{|S_t|^2}\sum_{i\in S_t}\sum_{j\in S_t}\frac{1}{2}(x_i - x_j)^2 + \frac{1}{|S_f|^2}\sum_{i\in S_f}\sum_{j\in S_f}\frac{1}{2}(x_i - x_j)^2\Big) \quad (3.21)$$

Where S, S_t and S_f are the set of presplit sample indices, set of sample indices for which the split test is true, and the set of sample indices for which the split is false, respectively. Each element on the above formula represent variance estimation. A remarkable advantage is related to handle both categorical and numerical data without a pre-processing task. Indeed, DNN, in order to be trained on a dataset with categorical data, requires to change the representation of these particular variables into binary encoding.
In addition, a strength of the DT algorithm is that it has inbuilt feature selection to ignore the irrelevant features in respect to the relevant ones.
However, there are also some limitations and issues of this algorithm. Firstly, the tree algorithm can be a very non-robust algorithm, because a small change in the training dataset (aka points of objective function already evaluated) can drastically modify the final predictions based on different splits of features into nodes of the tree.
Also finding an optimal decision tree is a NP-Hard problem, for this reason is typically adopted a greedy approach to find the optimal choice for each level of tree, which cannot guarantee to find the optimal tree structure at the end of its construction.
For data including categorical variables with different numbers of levels, information gain in decision trees is biased in favor of attributes with more levels.
However, this ML algorithm in both version of Classification and Regression algorithm suffers of overfitting issue. One possible solution is the pruning of branches.

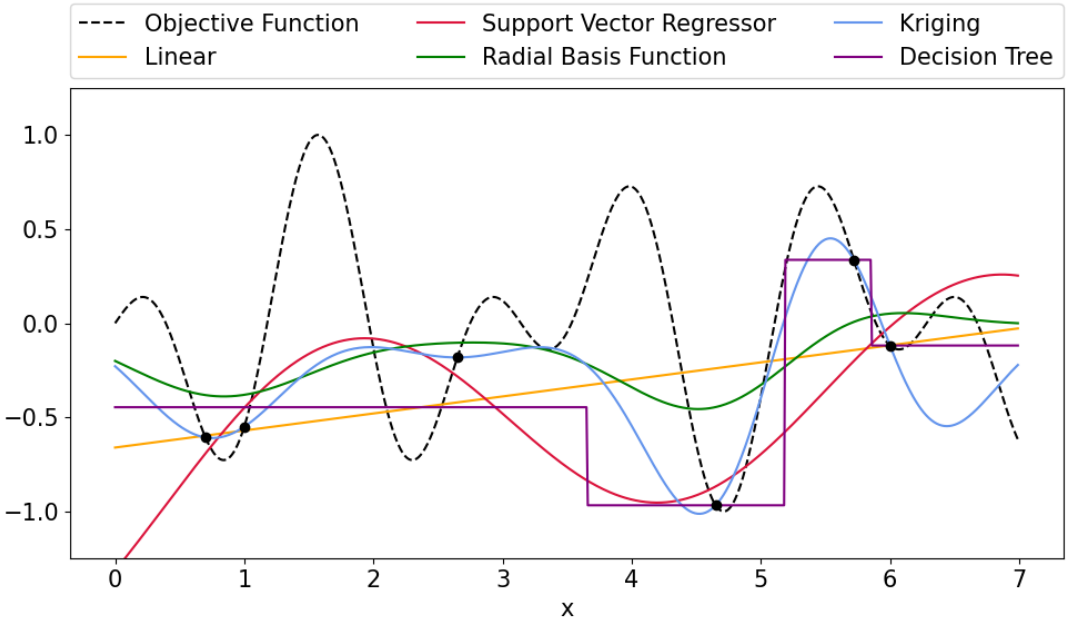


FIGURE 3.6: Several deterministic regressors to approximate an objective function

3.4.2 Embracing the prediction uncertainty: probabilistic surrogate models

A significant limitation of the previous regression methods is that they try to approximate an objective function by the data currently available without considering the uncertainty of the approximation of the function. Indeed, another branch of surrogate models exists, which is called probabilistic surrogate models. Such branch, instead of considering only the response surface to approximate the objective function, it takes into account the uncertainty of the approximated response surfaces.

The key idea is that the distance between the available already evaluated points $f(x)$ and other points not yet evaluated impacts the approximation process of the objective function. In fact, to approximate much better could be a good idea to converge to evaluate points x that essentially are quite far from the points that are already evaluated. In this way, it is possible to achieve a better approximation reducing the error in the approximation itself.

The most famous probabilistic surrogate models adopted in the Global Optimization community are Gaussian Process and Random Forest, which are introduced in detail in the next chapter.

The first one is essentially a particular Kriging regression, which takes into account the estimation of uncertainty. Instead, the Random Forest model is based on the concept of Ensemble models in Machine Learning, which means to combine many Decision Tree regressors to obtain a better approximation of the objective function combining in an average of the Decision Trees predictions.

3.4.3 Overall considerations on strategies and dialects

With this Chapter, we have tried to effectively analyze the pros and cons of adopting different optimization strategies with the ultimate goal of identifying an ideal optimization strategy to support the optimization of the generalization task, typical in Machine Learning, and which is typically black-box and expensive to evaluate in both terms of time and computational resources. It is quite clear that there are different global optimization strategies applicable to ML algorithms that have their advantages and disadvantages. For example, Grid Search, Random Search, and, albeit partially, Evolutionary Algorithms have the advantage of being easy to parallelize and implement. However, their biggest drawback is that they require a huge number of evaluations to converge to the optimal global solution or a very close solution.
This issue leads to these optimization strategies not being suitable to support Automated Machine Learning, where the evaluation of the objective function, i.e., the training and validation of the model, can be very expensive both in terms of computational resource requirements and time required for the evaluation of the objective function.
However, strategies such as Deterministic Optimization, with Lipschitz, and strategies based on Deterministic and Probabilistic Surrogate Models pose other critical issues as the introduction of learning the target function. In fact, the latter strategies are difficult to implement and have little parallelization capacity. However, the main advantage of these strategies is that they exploit knowledge as much as possible through evaluations of the objective function already evaluated, and by optimizing sequentially, they modify the choice of future evaluations based on the knowledge collected up to that moment.
Despite the model-based strategies seems to be the optimal solution for solving AutoML problems, recently several works adopt Gradient-Based optimization strategy as in (Franceschi et al., 2018; Lorraine et al., 2020).
The following Figure 3.7 shows a spatial representation of how the strategies presented in this chapter are arranged with respect to their knowledge of the objective function. Clearly, when we speak of knowledge of the objective function, we know the analytical formalization of the problem with which the optimization strategies' points are evaluated. In the case of the problem of generalization in ML, the problem itself tends to be black-box, as mentioned above.

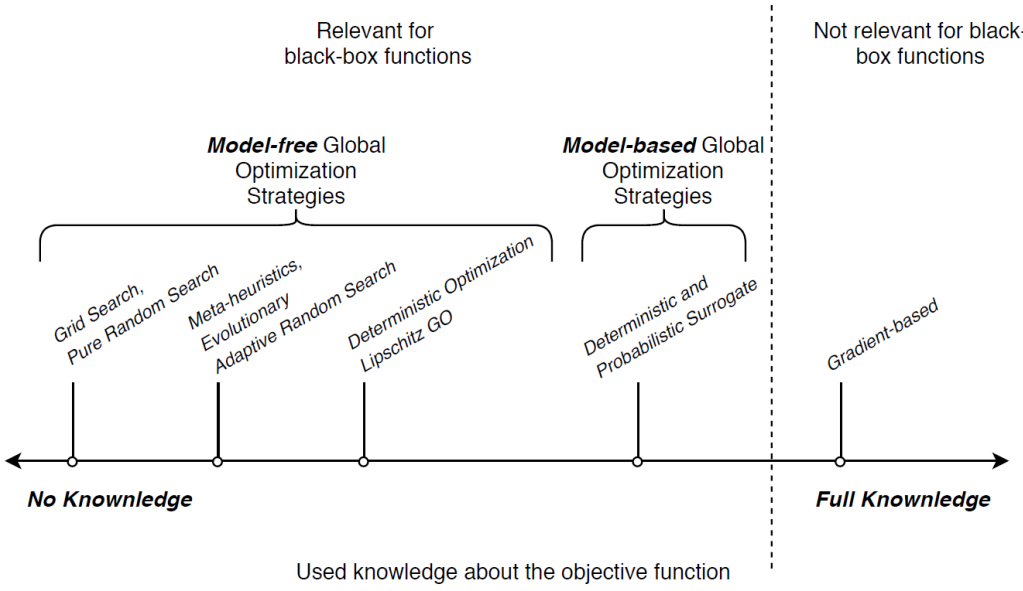


FIGURE 3.7: A spatial representation of the different categories among available knowledge of the objective function

Chapter 4

Bayesian Optimization: a principled Sequential Learning & Optimization framework for AutoML

Bayesian Optimization (BO) is a model-based global optimization strategy based on two components: a *surrogate model* approximating the objective function and an *acquisition function* (aka *utility function* or *infill criterion*). The main idea of BO is to *(*sequentially) update the surrogate model depending on the observations collected so far (aka current *knowledge*) and then select, through the acquisition function, a "promising" solution to evaluate (aka *query*). Thus, every new observation (aka function evaluation), contributes in both *learning* something more about the objective function and *searching for its optimum*. This leads to the final aim of the acquisition function: balancing between *exploitation* - selecting a solution which could improve with respect to the incumbent depending on the current approximation of the objective function - and *exploration* - selecting a solution which could improve the quality of the approximation.
Figure 4.1 summarizes the BO workflow, showing each step of the sequential process starting from the generation of an initial sample of solutions up to the termination, depending on some specified criterion, and the final best solution (i.e., final incumbent) observed along the sequential optimization process. Due to its sequential nature, BO is also known as Sequential Model-Based Optimization (SMBO). As it will be detailed in this Chapter, BO is a *sample-efficient* global optimization strategy, contrary to most of the techniques described in the previous Chapter 3. This means that, given the same number of queries, BO can usually identify a better solution.
As already remarked in this thesis, AutoML is characterized by a black-box and expensive-to-evaluate objective function, usually the generalization of a given ML algorithm, under a specific configuration of its hyperparameters, on a given dataset. The AutoML objective function is *black-box* because it can only be observed pointwise, and it is "expensive" in terms of time

and/or computational resources. Due to its sample-efficiency, BO is currently the most widely adopted method for solving AutoML problems.

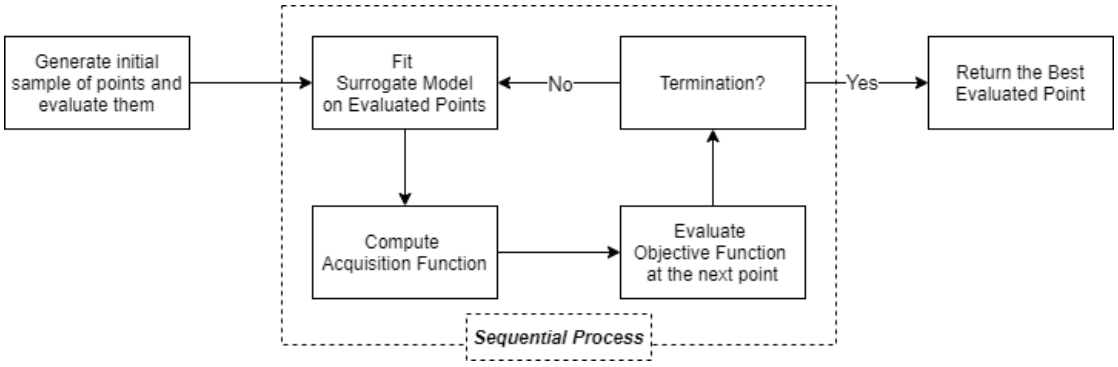


FIGURE 4.1: A representation of the Sequential Model-Based Optimization process workflow

In many cases X, the domain of points sampled during optimization, is assumed to be a hyper-rectangle (box-bounded). In the following examples in this Chapter, for display and graphic simplicity, the sequential optimisation process is illustrated taking into account only a continuous variable (i.e., a ML algorithm's hyperparameter). Anyway, this optimisation strategy can deal with problems defined on search space spanned by continuous, integer, categorical and conditional variables.

The rest of this Chapter is organized into two sections focusing on the two BO's components: the possible strategies to model the generalization of a ML algorithm (Section 4.1) and the possible acquisition functions to balance between exploration and exploitation (Section 4.2).

4.1 Approximating the generalization metrics depending on current knowledge

Approximating $f(x)$ leads to the need to deal with two different kinds of *uncertainty*:

1. *Aleatoric* (noise) - it arises due to the stochastic variability inherent in the data generation process. This means that we do not observe directly $f(x_i)$, instead we observe $y_i = f(x_i) + \varepsilon$, where ε is the observation noise assumed to be $\varepsilon \sim N(0, \lambda^2)$ as reported in Figure 4.2
2. *Epistemic* - it is due to our lack of knowledge about the structure of the objective function. This means that we, given a set of available observations, have a potentially infinite number of approximations of $f(x)$, as reported in the example in Figure 4.3

With respect to the *aleatoric* uncertainty, most of the results in the BO literature explicitly take measurement noise into account (Frazier, 2018). Dealing with the *structural* uncertainty can be

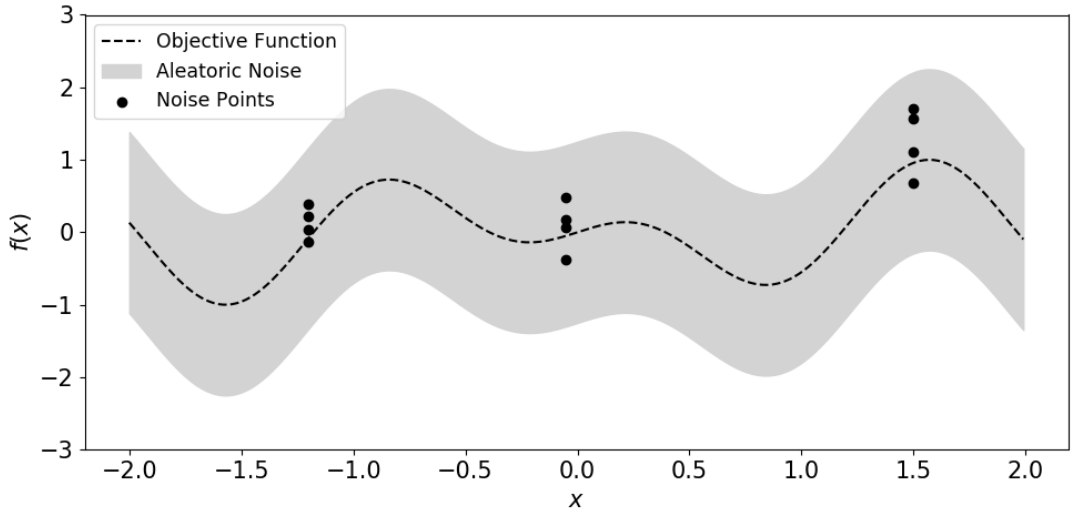


FIGURE 4.2: An example objective function with aleatoric noise: for the same point x_i is possible to observe different observations y_i

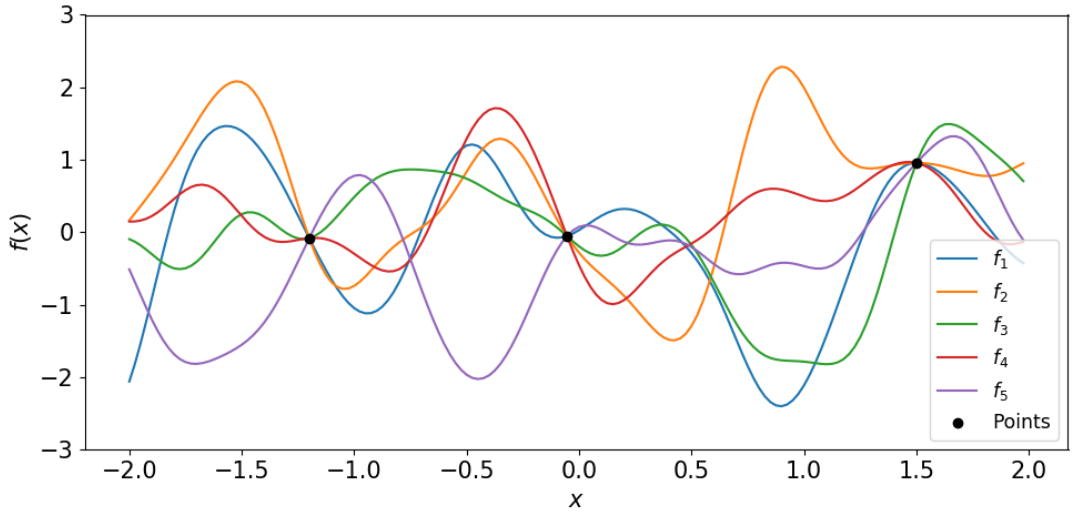


FIGURE 4.3: Different functions compatible with the current set of observations (noise-free setting)

more complicated. Indeed, it is usually divided in two different types, depending on the source: *parametric* uncertainty comes from estimation of the parameters of the model approximating $f(x)$ and *structural* uncertainty comes from the modelling assumptions (typically the type of modelling strategy). Contrary to aleatoric, epistemic uncertainty - especially structural uncertainty - can be reduced increasing data. In Figure 4.4 is reported the overview, in a tree structure, of the types of uncertainty proposed in (Liu et al., 2019b).

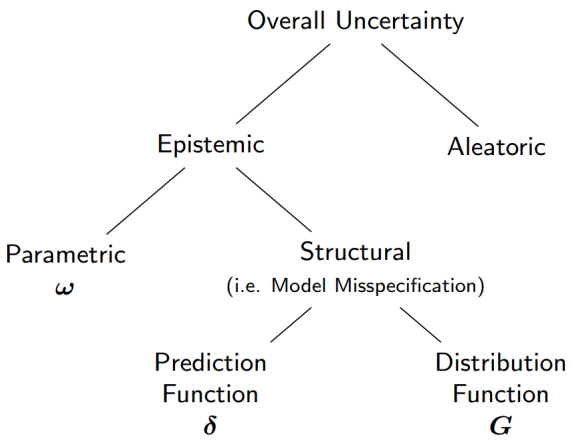


FIGURE 4.4: Different types of uncertainty in ML models (Liu et al., 2019b)

4.1.1 Gaussian Process regression

Gaussian Process regression (GPR, or simply GP) is the most widely adopted probabilistic surrogate model in BO. Moreover, GP modelling is widely used in ML for both classification and regression. Basically, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams, 2006). In more general terms, a GP is a finite collection of realizations (i.e., m observations) modelled as having a multivariate normal (MVN) distribution (Gramacy, 2020). This means that the realizations are completely described by an m-dimensional mean vector μ and a $m \times m$ covariance matrix Σ. When the goal is to model a function $f(x)$ - like in BO - we are interested in having a *mean function* $\mu(x)$ and a *covariance function* $\Sigma(x,x')$: ultimately, we obtain the vector μ and the matrix Σ after evaluating the two functions at m specific locations $x_1,...,x_m$. The mean and covariance function can be written as:

$$\begin{aligned} \mu(x) &= \mathbb{E}\big[f(x)\big] \\ \Sigma(x,x') &= \mathbb{E}\big[\big(f(x)-\mu(x)\big)\big(f(x')-\mu(x')\big)\big] \end{aligned} \tag{4.1}$$

with x and x' two locations in X. The GP is denoted as:

$$f(x) \sim \mathcal{GP}\big(\mu(x);\Sigma(x,x')\big) \text{ or, equivalently, } f(x) \sim \mathcal{N}(\mu(x);\Sigma(x,x')) \tag{4.2}$$

In 4.1 and 4.2 the covariance matrix considers two locations in X, in the more general case related to a bunch of m locations it is an $m \times m$ matrix. Usually, for notational simplicity $\mu(x) = 0$, even if this need not to be done (Rasmussen and Williams, 2006). The covariance function allows to impose structural assumptions about $f(x)$ because its specification implies a distribution over functions. The common approach is to use a *kernel function*, $k(x,x')$, as covariance function, that is $\Sigma(x,x') = k(x,x')$. This allows to write the covariance - which is a function of *outputs* (as defined in 4.1) - as a function of *inputs*. Many kernels are available, every one

assuming different structural properties about $f(x)$. Kernel functions will be described in detail later on.

More interesting, when n function evaluations (aka observations or realizations) of $f(x)$ are available, then the GP's mean and covariance functions can be conditioned on them. Let $X_{1:n} = \{x_1,...,x_n\}$ and $y = \{y_1,...,y_n\}$ denote, respectively, a set of n locations in the search space (i.e., $x_i \in X, i = 1,..,n$) and the associated function values, in case noisy (i.e., $y_i = f(x_i) + \varepsilon, i = 1,...,n$ and $\varepsilon = \mathcal{N}(0,\lambda^2)$). Finally, let define $D_n = (X_n, Y_n)$, then the GP's mean and covariance functions can be conditioned as follows:

$$\begin{aligned} \mu_n(x) &= \mathbb{E}\big[f(x)|D_{1:n},x\big] = k(x,X_{1:n})\big[K(X_{1:n},X_{1:n}) + \lambda^2 I\big]^{-1} y \\ \sigma_n^2(x) &= k(x,x) - k(x,X_{1:n})\big[K(X_{1:n},X_{1:n}) + \lambda^2 I\big]^{-1} k(X_{1:n},x) \end{aligned} \tag{4.3}$$

where the matrix $K(X_{1:n},X_{1:n})$ has entries $K_{ij} = k(x_i,x_j)$, with $i,j = 1,\ldots,n$, and $k(x,X_{1:n})$ is a n-dimensional vector with components $k_i = k(x,x_i)$.
Equations 4.1 and 4.3 are respectively known as *GP's prior* (Tong, 1990) and *GP's posterior*. Moreover, equation 4.3 is for obtaining "pointwise" predictions through the GP's posterior. Indeed, we have a variance function, $\sigma^2(x)$, instead of a covariance function. These equations are also known as *kriging equations*, especially in the geospatial context (Gramacy, 2020). They can be used to obtain a prediction about $f(x)$ - that is $\mu(x)$ - and its uncertainty - that is $\sigma(x) = \sqrt{\sigma^2(x)}$ - at one location x at a time. However, if we repeat this operation many times, for single locations, we would ignore that they have to be sampled from a MVN distribution. Instead, if we are interested in considering a bunch of m locations jointly, $X_m = \{x_1,...,x_m\}$, then we have to replace $\sigma^2(x)$ with the following covariance matrix:

$$\Sigma(X_m) = K(X_m,X_m) - K(X_m,X_{1:n})[K(X_{1:n},X_{1:n} - \lambda^2 I)]^{-1} K(X_{1:n},X_m) \tag{4.4}$$

Having a full covariance structure offers a more complete picture of $f(x)$, according to the GP's posterior, but has a significant computational load because $\Sigma(X_m)$ could be enormous even for seemingly moderate m. Sampling from posterior can be, ideally, considered as generating functions from the prior and rejecting the ones that disagree with the observations. Naturally, this strategy would not be computationally very efficient (Wilson et al., 2020).
From equation 4.3 it is easy to notice that the GP's mean is a linear combination of n functions, each one centred on an evaluated location. This allows to write $\mu(x)$ as:

$$\mu(x) = \sum_{i=1}^{n} \alpha_i k(x,x') \tag{4.5}$$

where the vector $\alpha = [K(X_{1:n},X_{1:n}) + \lambda^2 I]^{-1} y$ and α_i is the i-th component of the vector α, given by the product between the i-th row of the matrix $[K(X_{1:n},X_{1:n}) + \lambda^2 I]^{-1}$ and the vector y. This means that, to make a prediction at a given x, we only need to consider the $(n+1)$-dimensional

distribution defined by the n available function evaluations and the location x.
This formulation of the GP modelling is known as *function-space view* and it is easy to understand that the most computationally expensive operation is the matrix inversion in 4.3 and 4.4, with computational complexity $\boldsymbol{O}(n^3)$. Another possible formulation is the so-called *weight-space view*, which approximates $f(x)$ through an explicit set of basis functions. According to the *kernel trick* (Scholkopf and Smola, 2018), the kernel function $k(\cdot,\cdot)$ can be considered as the inner product in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ equipped with a feature map function $\varphi :\to \mathcal{H}$ such that $k(x,x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$. If $\mathcal{H}$ is separable, then the kernel function can be approximated through an explicit feature map function $\phi :\to \mathbb{R}^q$, such that:

$$k(x,x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} \approx \phi(x)^\top \phi(x') \tag{4.6}$$

For *stationary* kernel functions, Bochner's theorem provides a suitable q-dimensional feature map based on a set of random Fourier features (Rahimi and Recht, 2007). More precisely, $\phi_i(x) = \sqrt{\frac{2}{q}} \cos(\boldsymbol{\theta}_i x + \tau_i)$, with $\boldsymbol{\theta}_i$ sampled assuming the (scaled) kernel's spectral density as probability distribution and $\tau_i \sim \mathbf{U}(0, 2\pi)$. Then $f(x)$ is approximated as follows:

$$f(x) = \sum_{i=1}^{q} w_i \phi_i(x) \tag{4.7}$$

where weights w_i are sampled from a posterior distribution conditioned to the set of n available observations, that is $w_i \sim \mathcal{N}(\mu_w, \sigma_w^2)$ with:

$$\begin{aligned} \mu_w &= \left(\Phi^\top \Phi + \lambda^2 I\right)^{-1} \Phi^\top y \\ \sigma_w &= \left(\Phi^\top \Phi + \lambda^2 I\right)^{-1} \lambda^2 \end{aligned} \tag{4.8}$$

and where Φ is an $n \times q$ matrix whose i-th row is given by $\phi(x_i) = [\phi_1(x_i), ..., \phi_q(x_i)]$ with $x_i \in X_{1:n}$. Typically, $q \ll n$, so the computational complexity for fitting the GP – according to (4.7) – is reduced to $\boldsymbol{O}(q^3)$ due to the inversion of the matrix $\left(\Phi^\top \Phi + \lambda^2 I\right)$ in (4.8). Thus, the weight space view leads naturally to a GP approximation whose quality is controlled through q: the greater the value of q the better the approximation (but the higher the computational cost). Recently, the combination of the two "views" has been proposed to perform efficient function sampling from a GP (Hahn et al., 2019).

Kernels

Kernel is a general name for any function mapping a pair of inputs x and x' into a scalar value, formally $k(x,x') : X \times X \to \mathbb{R}$, with $X \subset \mathbb{R}^d$. In order to ensure that a kernel is well-suited to be a covariance function it must satisfy the following conditions:

- symmetry: $k(x,x') = k(x',x)$

- the Gram matrix K, with entries $K_{ij} = k(x_i, x_j)$, must be positive semidefinite (PSD).

A *stationary* kernel is a function of $x - x'$, meaning that it is invariant to translations in the input space. Furthermore, if it specifically depends only on $||x - x'||$, then it is called *isotropic* and it is invariant to all rigid motions. Finally, a *dot product* kernel depends only on $x \cdot x'$ and is invariant to a rotation of the coordinates about the origin, but not translations.

Several kernels have been proposed to be used as covariance functions in GP modelling, and some of the most widely adopted are presented in this chapter. Everyone of the presented kernel functions has at least one hyperparameter to set up depending on data. Therefore, while the choice of the kernel imposes some basic structural properties about $f(x)$, setting the values of the kernel's hyperparameter(s) allows for better fitting the observations $D_{1:n}$. According to the different sources of uncertainty, previously summarized in Figure 4.4, this means that kernel type and its hyperparameters are both associated to *epistemic* uncertainty: *structural* and *parametric*, respectively. Common approaches for tuning the kernel's hyperparameters are Maximum Log-likelihood Estimation (MLE) or Maximum A Posteriori estimation (MAP). Recently, an alternative approach has been proposed in (Berkenkamp et al., 2019), where the values of hyperparameters are iteratively decreased instead of using MLE and MAP. The basic idea is that this scheduling can avoid "oversimplification" of $f(x)$ induced by MLE and MAP.

Common kernel functions

Squared Exponential (SE) kernel:

$$k_{SE}(x, x') = exp\left\{ \frac{-||x - x'||^2}{2\ell^2} \right\} \tag{4.9}$$

with ℓ known as characteristic length-scale: the role of this hyperparameter is to rescale any location x by $\frac{1}{\ell}$ before computing the kernel value. A large value of the length-scale will map x to a narrower range of values, while a small length-scale does the opposite. Consequently, a large length-scale implies long-range correlations, whereas a short length-scale makes function values strongly correlated only if their respective inputs are very close to each other. This kernel is infinitely differentiable, meaning that the associated GP is very "smooth". The value of this kernel approaches to one for $||x - x'|| \to 0$, while it decreases as $||x - x'||$ increases.

Matérn kernel(s):

$$k_{Mat}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{||x - x'||\sqrt{2\nu}}{\ell} \right)^2 K_\nu \left(\frac{||x - x'||\sqrt{2\nu}}{\ell} \right) \tag{4.10}$$

with two hyperparameters ν and ℓ, and where K_ν is a modified Bessel function. Analogously to the SE kernel, ℓ is the characteristic length-scale. For $\nu \to \infty$ the Matérn kernel coincides with

the SE kernel. The Matérn kernel becomes especially simple when ν is an half-integer, that is $\nu = p + \frac{1}{2}$, with p a non-negative integer. In this particular case, this kernel is a product of two terms, one exponential and one polynomial of order p. Formally,

$$k_{\nu=p+\frac{1}{2}}(x,x') = \mathbf{exp}\left\{ ||x-x'||\sqrt{\frac{2\nu}{\ell}} \right\} \frac{\Gamma(p+1)}{\Gamma(2p+1)} \sum_{i=0}^{p} \frac{(p+1)!}{i!(p-1)!} \left(||x-x'||\sqrt{\frac{8\nu}{\ell}} \right)^{p-1} \tag{4.11}$$

The most common versions of this kernel, widely adopted in ML, are characterized by $\nu = 3/2$ and $\nu = 5/2$.

$$\begin{aligned} k_{\nu=3/2}(x,x') &= \left(1 + \frac{||x-x'||\sqrt{3}}{\ell}\right) \mathbf{exp}\left\{ \frac{-||x-x|'|\sqrt{3}}{\ell} \right\} \\ k_{\nu=5/2}(x,x') &= \left(1 + \frac{||x-x'||\sqrt{5}}{\ell} + \frac{||x-x'||^2}{3\ell^2}\right) \mathbf{exp}\left\{ \frac{-||x-x'||\sqrt{5}}{\ell} \right\} \end{aligned} \tag{4.12}$$

Matèrn kernel has recently received considerable attention, because it allows to control, through ν, the smoothness of the resulting GP.

Rational Quadratic kernel:

$$k_{RQ}(x,x') = \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{4.13}$$

where α and ℓ are two hyperparameters (again, ℓ is the characteristic length-scale). This kernel can be considered as an infinite sum (scale mixture) of SE kernels, with different characteristic length-scales. Indeed, one of the most important properties of kernel functions is that a sum of kernels is a kernel.

Exponential kernel:

$$k_{exp}(x,x') = \mathbf{exp}\left\{ \frac{-||x-x'||}{2\ell^2} \right\} \tag{4.14}$$

where ℓ is the characteristic length-scale. The important property of this kernel is that the resulting GP is not infinitely differentiable. An equivalent formulation is known as *Laplacian kernel*, that is $k_{exp}(x,x') = \mathbf{exp}\left\{ \frac{-||x-x'||}{\ell} \right\}$, it is just less sensitive for changes in the length-scale.

The following Figure 4.5 summarizes how the value of the above kernels decreases with x moving away from $x' = 0$ (on the left side) and which are possible resulting samples, with the associated smoothness (on the right side). The maximum value of all the kernels described above is unity, as it can be seen also in the left hand side of the figure. However, it is possible to introduce a further hyperparameter, namely the signal variance σ_f^2, as a multiplier of the above kernel formulas.

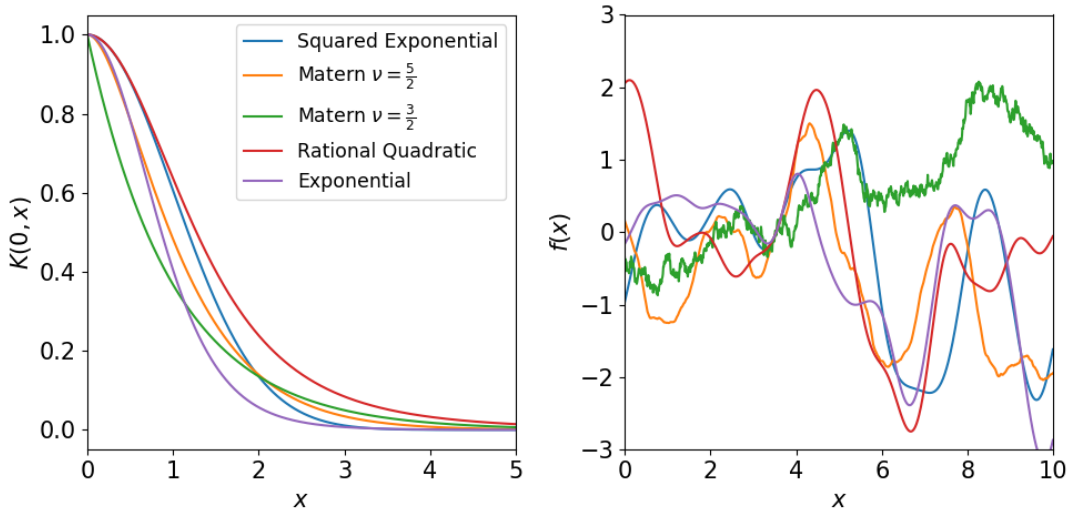


FIGURE 4.5: Value of different kernels with x moving away from $x' = 0$ (left) and samples from GP prior, one for each kernel considered (right). The value of the characteristic length-scale ℓ is 1 for all the five kernels; α of the RQ kernel is set to 2.5

The aforementioned kernels are just the most widely adopted in GP regression. More details and a more comprehensive set of kernel functions are reported in (Rasmussen and Williams, 2006), including non-stationary kernels and dot product kernels. Kernel for BO is a very actively researched area, relevant papers are (Duvenaud et al., 2013; Shilton et al., 2018). (Schulz et al., 2016) note that a kernel mismatch about the smoothness of the objective function leads to a substantial drop in accuracy of objective function approximation and sample efficiency, which increases with the dimensions and cannot be mitigated by GP's hyperparameter tuning (neither by the choice of acquisition function, that will be disussed in the next subsection).

A space-temporal kernel has been proposed in (Nyikosa et al., 2018) to allow the GP to capture all the instances of the function over time and track a temporally evolving minimum. Some issues on kernel have been considered in recent publications, such as: *kernel composition*, safe optimization in relation to *cognition* (Schulz et al., 2018) as well as *kernel learning*, adaptation and *sparsity*; dealing with functions that are smooth in a subset of their search space and vary rapidly elsewhere is analysed in (Peifer et al., 2020) from the viewpoint of computational complexity in the framework of RKHS (Reproducing Kernel Hilbert Spaces).

A relatively new part of GP regression, specifically useful for BO, is the integration of derivative information of $f(x)$ in the modelling and optimization process. Differentiation is a linear operator so that the derivative of a GP is also a GP: if the kernel is sufficiently smooth, one can derive the joint distribution over a function and its derivatives and then perform their Bayesian updating given the observations of function, derivatives and hessians (Wu et al., 2017a). The idea of using derivative information for optimization problems has been taken up in (Hennig, 2013; Hennig and Kiefel, 2013) and in (Wills and Schön, 2017) which has developed it along two different directions. The first, originally suggested in the papers by Henning, brings to the

reinterpretation in probabilistic terms of standard quasi-Newton like methods considered as particular instances of GP or Bayesian regression. The second is to use the joint GP as a global model of the objective function and its derivatives. A first remark is that adding a derivative observation reduces the prediction uncertainty (i.e., the GP predictive standard deviation, $\sigma(x)$).

Now that kernels have been introduced, we can come back to sampling from GP prior and posterior, and present a graphical example. Given the same kernel, that is a SE kernel, Figure 4.6 shows five different samples (aka *GP's sample paths*) from GP's prior (with zero-mean prior) while Figure 4.7 shows five samples drawn from the GP's posterior conditioned to six function observations (the noise-free setting has been considered, just for simplicity).

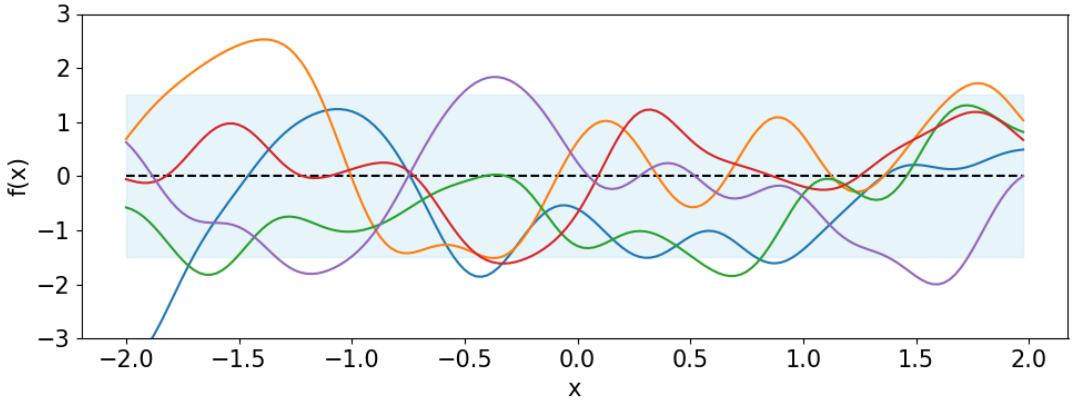


FIGURE 4.6: Five different samples drawn from the prior of a GP with a SE kernel as covariance function

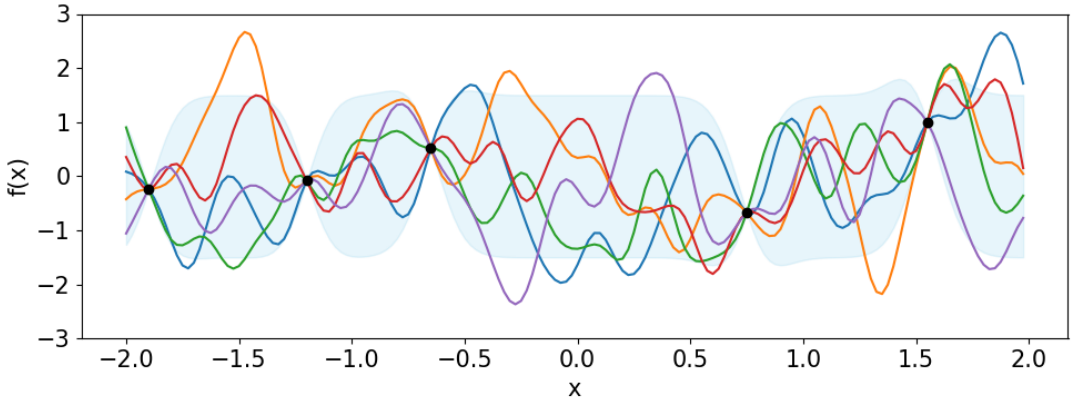


FIGURE 4.7: Five different samples drawn from the posterior of a GP with SE kernel, conditioned on six observations (the noisy-free setting is considered, just for simplicity).

Numerical Instability

Numerical instability issues arise due to the inversion of the matrix $[K(X_{1:n}, X_{1:n}) + \lambda^2 I]$, required to update $\mu(x)$ and $\sigma^2(x)$. The condition number of this matrix, that is the ratio between the highest and lowest eigenvalue, gives a bound on the accuracy of the matrix inversion. Large condition numbers are indicators for numerical instability. Extreme eigenvalues appear when

function values are strongly correlated, especially in the noise-free setting (i.e., $\lambda = 0$). Strong positive correlation between function values will result in near-identical corresponding rows and columns in the kernel matrix $K(X_{1:n}, X_{1:n})$, making it close to singular.
Some heuristics have been proposed to control the condition number of the matrix $[K(X_{1:n}, X_{1:n}) + \lambda^2 I]$. The most common consists of adding an explicit penalty on the signal-to-noise ratio, when fitting the GP, or adding a jitter term. The first method derives from the observation that the condition number grows linearly with the number n of data points and quadratically with the signal-to-noise ratio $\frac{\sigma_f}{\lambda}$ (where σ_f^2 is the signal variance previously introduced, that is multiplier defining the max value of the kernel function). More precisely, $\frac{e_{max}}{e_{min}} = n\frac{\sigma_f^2}{\lambda^2} + 1$, where e_{max} and e_{min} are the largest and smallest eigen value, respectively. It is easy to understand that a high signal-to-noise ratio makes quickly the matrix $[K(X_{1:n}, X_{1:n}) + \lambda^2 I]$ ill-conditioned. Therefore, although most of the research works consider the noise-free setting, this should be avoided in practical applications for numerical stability reasons. When fitting the GP by maximizing the log-marginal likelihood, a penalty term is added to the log-marginal likelihood that discourages extreme signal-to-noise ratios.
The second heuristic method, consisting of adding a jitter term, is also based on the previous consideration: it basically adds some artificial "noise" to the function evaluations.
Summarizing, the condition number can be controlled by modifying the amount of noise in the observations, which leads to a degradation of the model's accuracy. However, the condition number is only a problem when there is so little noise in the data that we are predicting at an accuracy close to the numerical precision of our hardware, a level of accuracy usually not necessary for many practical problems.
An interesting solution, called K-optimality and aiming at choosing the next point x_{n+1} to reduce the condition number, has been recently suggested in (Yan et al., 2018).
Finally, it is important to remark that using derivatives induces a faster onset of the numerical instability issues.

Complex search spaces

GP modelling is well-suited for optimizing over search spaces spanned by continuous variables. As already mentioned in this thesis, this is rarely the case of AutoML and NAS. Moreover, other applications domains - such as simulation-optimization of complex systems - usually involve also discrete (integer as well as categorical) decision variables.
To still use the advantages offered by GP modelling, some modifications have been recently proposed. In (Garrido-Merchán and Hernández-Lobato, 2020) a taxonomy of the possible approaches is reported. The so-called *naive* method consists in considering integer variables as real/numeric and then rounding their values to the closest integer. Every categorical variable is instead replaced with a set of extra binary variables, one for each modality of the original one, where only one variable takes value 1 and all the others take value 0 (aka *one-hot encoding*).

The "active" variable is the one with the highest value of the objective function associated. The so-called *basic* approach uses the same workarounds of "naive" method, but they are applied internally to the acquisition function instead of to the solution provided by solving it. Finally, the so-called *transformation* approach leads to the definition of an alternative covariance function, that is $\tilde{k}(x,x') = k(T(x)), T(x'))$, where $T(.)$ is a transformation applying the same workarounds of the "basic" approach but applied internally to the covariance function - and consequently internally to the surrogate model - allowing for a better modeling of the objective function.

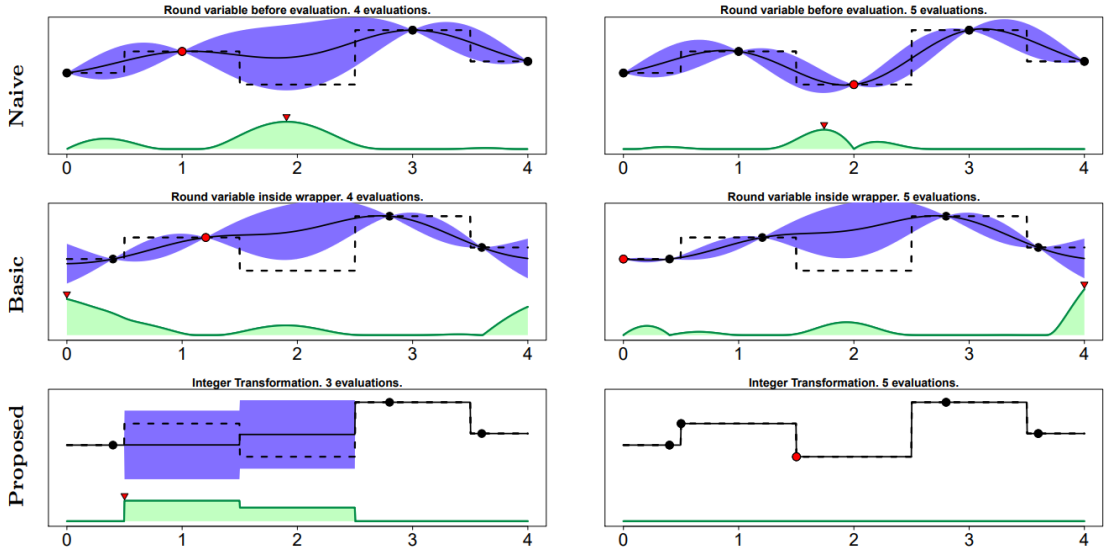


FIGURE 4.8: An example of all three possible solution for dealing Integer variables with GP into BO framework. The last row of the figure reports the proposed approach in (Garrido-Merchán and Hernández-Lobato, 2020) previously indicated as *transformation* approach

Another really interesting and recent approach has been proposed in (Ru et al., 2020). More in detail, the so-called CoCaBO algorithm proposes a novel GP kernel to capture complex interactions between continuous and categorical variables, enabling information sharing across the different types without the need to adopt any transformation. The algorithm works according to the following steps:

- Multi Armed Bandit (MAB) (Slivkins et al., 2019) is used to define the values of the categorical/integer variables
- optimising the acquisition function over the continuous variables (while keeping fixed the categorical/integer variables set at the previous step)

To capture correlations between categorical and numeric variables, the proposed kernel composes two different kernels: one working on categorical/integer variables and one for continuous variables. Let denote with $k_h(h,h')$ the kernel working on the categorical variables only, then it is defined as $k_h(h,h') = \frac{1}{d_h}\sum_1^{d_h}\mathbb{I}(h_{[i]} - h'_{[i]})$, where d_h is the number of categorical variables, h

and h' are two vectors of categorical variables, the subscript $[i]$ represents the i-th component of these vectors, and $\mathbb{I}(h_{[i]} - h'_{[i]})$ is a n indicator function, such that $(h_{[i]} - h'_{[i]}) = 1$ if $(h_{[i]} = h'_{[i]})$, 0 otherwise. With respect to the continuous variables, let denote with $k_z(z,z')$ the associated kernel, which can be any one of the kernels previously introduced.
Finally, the CoCaBO's kernel is given by combining these two kernels, by considering the fact that a sum of kernels as well as a multiplication of kernels are still a kernel. Indeed, both the two possible combinations are considered and weighted depending on an additional kernel's hyperparameter, $\alpha \in [0,1]$, formally:

$$k(x,x') = (1-\alpha)\big[k_h(h,h') + k_z(z,z')\big] + \alpha\big[k_h(h,h')k_z(z,z')\big] \tag{4.15}$$

where $x = (h,z)$ and $x' = (h',z')$.

4.1.2 Random Forest regression

A significant drawback of the probabilistic model GP is that it requires the assumption that the variables considered during optimization are only continuous, which is not practicable in HPO and Model Selection, and more in general for AutoML problems.
However, there are some naive "workaround" to solve this issue. In the case of integer-valued variables, such as the number of layers of a DNN, the basic idea is rounding to the closest integer, while, in the case of categorical variables, as many as possible extra variables categories are added and chosen according to the largest one as showed in (Snoek et al., 2012).
These operations can be applied in two different ways. In the so-called "naïve" approach the rounding to the closest integer is applied before updating the GP, so x_{n+1} is a mixed-integer vector, while in the "basic" approach it is applied just before evaluating the objective function, so $x_{n+1} \in \mathbb{R}^d$. The naïve approach might lead to a relevant mismatch between the (continuous) optimizer of the objective function and the point evaluated. The basic approach overcomes this limitation because GP ignores the approximation, but it is expected to provide a suboptimal solution. As previously mentioned, a more advanced approach is proposed in (Garrido-Merchán and Hernández-Lobato, 2020), where a kernel is defined for essentially embedding the same transformation of the basic approach.
A radical solution to the GP issues for managing integer or categorical variables consists in adopting alternative probabilistic surrogate models, like Tree-based surrogate models.
Random Forest (RF) (Ho, 1995) is an ensemble learning method, based on Decision Trees, for both classification and regression problems. An ensemble learning method is a system that provides a classification or a regression estimation based on the predictions of multiple models which can be based on one or several kind of ML algorithms.
The ensemble learning method is divided in three main different types:

- *Bagging* which aims to create a collection of ML models with the same importance among them to prediction for classification or regression task;
- *Boosting* which aims to create a collection of ML models where each classifier/regressor influences the final vote with a certain weight, which is calculated on the accuracy error that each model will make during the learning phase;
- *Stacking* which aims to create a collection of ensemble models to provide better predictions.

According to the original implementation (Ho, 1995), RF aims to generate a multitude of Decision Trees at training time and provides as output the mode of the classes (classification) the mean or median prediction (regression) of the individual trees.
Figure 4.9 shows a schematic representation of how the final output of an RF is generated.
Decision Trees are ideal candidates for ensemble methods since they usually have low bias and

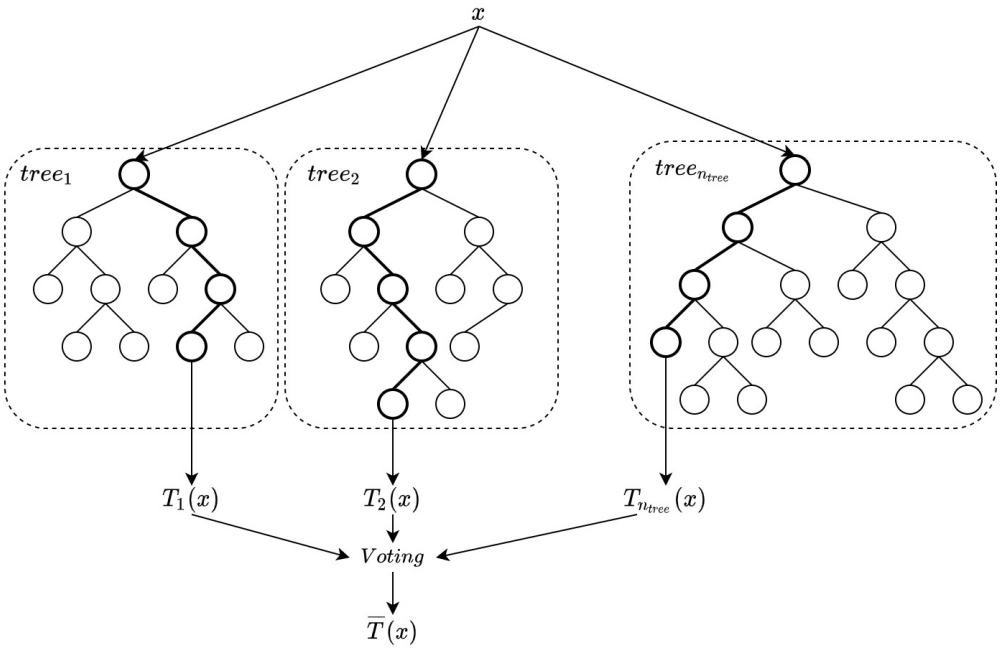


FIGURE 4.9: A schematic representation of the regression/classification performed by a RF, where a voting mechanism is adopted for regression (e.g., mean or median) or classification (e.g., simple or weighted majority) tasks

high variance, making them very likely to benefit from the averaging process. RF mostly differs from other typical ensemble methods (e.g. voting systems) in the way they introduce random perturbations into the training phase. The basic idea is to combines bagging, to sample examples from the original dataset, and random selection of features, in order to train every tree of the forest. Injecting randomness simultaneously with both strategies yields one the most effective off-the-shelf methods in Machine Learning, working surprisingly well for almost any kind of problems, allowing for generating a collection of decision trees with controlled variance.

Although designed and presented as a Machine Learning algorithm, RF is also an effective and efficient alternative to GP for implementing BO. To better understand how RF, for regression, can replace GP, one has to consider that: *(i)* the dataset, in the case of Machine Learning, is replaced by the design $D_{1:n}$ of the BO process, *(ii)* the features correspond to the dimensions of the global optimization problem.

Moreover, since RF consists of an ensemble of different regressors, it is possible to compute – as for GP – both $\mu(x)$ and $\sigma(x)$, simply as mean and variance among the individual outputs provided by the regressor. Due to the different nature of RF and GP, the associated probabilistic surrogate models will also result significantly different. The following Figure 4.10 offers an example. While GP is well-suited to model smooth functions in search space spanned by continuous variables, RF can deal with discrete and conditional variables by its nature. Indeed, discontinuity in the RF-based surrogate is given by the underlying decision tree models.

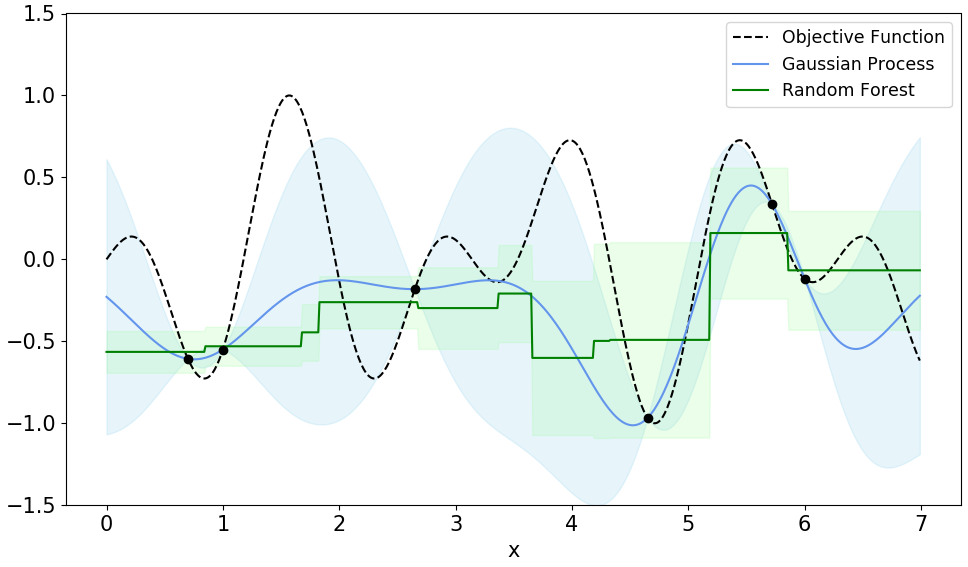


FIGURE 4.10: A graphical comparison between probabilistic surrogate models offered by a GP (blue) and a RF (green), both fitted on the same set of observations

The definition of $\mu(x)$ of an RF model can be defined as follow:

$$\mu(x) = \frac{1}{S}\sum_{i=1}^{S} T_i(x) \tag{4.16}$$

where S is the size of the forest (i.e. the number of Decision Trees in the forest) and $T_i(x)$ is the output provided by the i-th Decision Tree for the input x.

Another important property of RF is that the variance can be controlled, as explained in the following, where the variance of the RF-based estimator, for the regression case, can be easily computed as follows:

$$Var\left(\frac{1}{S}\sum_{i=1}^{S}T_i(x)\right) = Cov\left(\frac{1}{S}\sum_{i=1}^{S}T_i(x), \frac{1}{S}\sum_{j=1}^{S}T_j(x)\right) =$$
$$\frac{1}{S^2}\sum_{i=1}^{S}\left(\frac{1}{S}\sum_{i\sim j}^{S}Cov\big(T_i(x),T_j(x)\big) + Var(T_i(x))\right) \leq \frac{1}{S^2}\sum_{i=1}^{S}\left((S-1)\rho\sigma^2+\sigma^2\right) = \tag{4.17}$$
$$\frac{S(S-1)\rho\sigma^2+S\sigma^2}{S^2} = \rho\sigma^2+\sigma^2\frac{1-\rho}{S}$$

where σ^2 is the maximum variance computed over all the $T_i(x)$ and $\rho\sigma^2 = max_{i,j}Cov\big(T_i(x),T_j(x)\big)$. The variance of the RF estimator is proportional to σ^2 and ρ (i.e., if the number m of selected features decreases also σ^2 and ρ decrease) and with the size S the forest increasing.

Finally, a basic description of the RF learning algorithm is provided. Thanks to the bagging and random feature selection, the optimization of acquisition function and the updating of the RF model result more computationally efficient than GP, in particular when the number of decision variables is larger than 10-20. In fact, RF does not require to invert any kernel matrix and training of every decision tree can be performed in parallel. These features give the RF a computational time request equal to $\boldsymbol{O}(n\log(n))$ which is not dramatically increased when the n samples increases as with GP model that requires $\boldsymbol{O}(n^3)$.

Algorithm 1 Random Forest Algorithm

input : S - Size of the forest;
N - the number of samples in the dataset (i.e. evaluations of the objective functions);
M - the number of features (i.e. decision variables) representing every sample;
m - the number of features (i.e. decision variables) to be randomly selected from the M available for each leaf node of the tree to be split;
n_{min} - the minimum node size in the tree;
output: Random Forest regressor

for $i = 1$ *to* S **do**
  sample, with replacement, a subset of N instances from the dataset
  for *every terminal node (leave) of the i-th Decision Tree having size lower than* n_{min} **do**
    select m features (i.e. decision variables) at random from the M available
    pick the best feature (i.e. decision variable) and split among the possible m
    split the node in two children nodes (new leaves of the decision tree)
  end
end

4.1.3 Other Surrogates

Despite the extensive use of the two probabilistic surrogate models just described in the previous sections (GP and RF), new surrogate models have been proposed to try to overcome the limitations imposed by the model canons adopted within Bayesian Optimization. This section tries to give an overview of the most famous more advanced surrogate models to support the Bayesian Optimization paradigm.

Tree Parzen Estimator.
Whereas the GP based approach models $p(y|x)$ directly, a Tree Parzen Estimator (TPE) models both $p(x|y)$ and $p(y)$ (Bergstra et al., 2011). More in detail, $p(x|y)$ is replaced, under a specific condition, by several density distributions on the whole history of evaluated points $\{(x_1,\dots,x_n)\}$ of the problem to optimize. In particular, this specific condition depends on the choice of a parameter τ allowing to choose $\hat{y}$ to be some specific quantile of the observed y values, so that $p(y<\hat{y})=\tau$. The key aspect with this approach is that there is no need to approximate $p(y)$. Defining $\hat{y}$ is possible to divide the whole history of evaluated points $\{(x_1,\dots,x_n)\}$ into two different probability density distributions as follows:

$$p(x|y)=\begin{cases} l(x), & \text{if } y<\hat{y} \\ g(x), & \text{if } y\geq\hat{y} \end{cases} \tag{4.18}$$

where $l(x)$ is the density formed by using the evaluated points $\{(x_1,\dots,x_n)\}$ such that corresponding objective function $\{(y,\dots,y_n)\}$ was less than $\hat{y}$ and $g(x)$ is the density formed by using the remaining evaluations. TPE is a powerful probabilistic surrogate model because it can use several Kernel Density Estimators (KDE) to create density distributions for $p(x|y)$. One of the most used estimators is based on Gaussian KDE; another possibility is based on the t-Student distribution. Finally, as proved in (Bergstra et al., 2011), the maximization of the ratio on these two density distributions $\frac{l(x)}{g(x)}$, named "TPE score", corresponds to maximize EI. The following Figure 4.11 shows, on the left, the two density distributions, $l(x)$ and $g(x)$, based on Gaussian KDE, and, on the right, the proposed point (pointed with the red vertical line) that is considered the most promising point to evaluate on the objective function. Another important property of TPE is its capability to scale linearly with respect to the number of points evaluated so far, as well as the dimensionality of the problem. TPE uses a tree of Parzen estimators in the case of conditional variables and demonstrated good performance on such hierarchical structure of search space. The density distribution $l(x)$ and $g(x)$ are hierarchical processes when the optimization problem is characterized by both discrete-valued and continuous-valued variables.

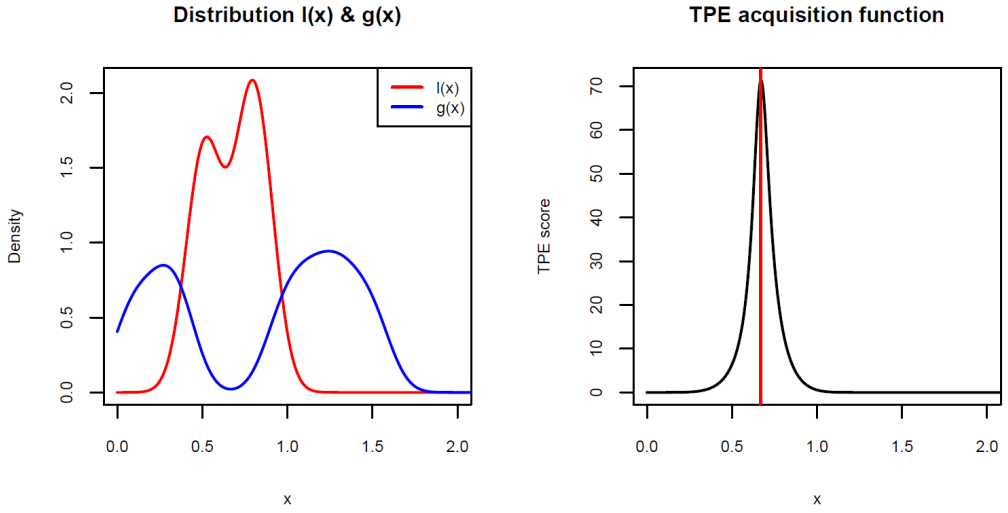


FIGURE 4.11: Density functions $l(x)$ and $g(x)$ (left) and resulting TPE score suggesting the next point to evaluate (right)

Neural Networks: feedforward, Deep and Bayesian Network.

The role of Neural Networks in BO is two-folds: as already discussed they are Machine Learning algorithms typically optimized in the AutoML setting; on the other hand, they can offer a well-suited alternative to GP. Indeed, as already previously reported in this Chapter, GP model is sample efficient, but its computational complexity is $\boldsymbol{O}(n^3)$ for n points evaluated and it does not scale well in high dimensions. In (Snoek et al., 2015) the authors propose an approach that aims to replace the GP surrogate with a model that scales better in terms of computational complexity and in high-dimensional problems, but retains the flexibility and well-calibrated uncertainty that are the main advantages of using GP. More specifically, Deep Learning models are investigated. The authors proposed adding a Bayesian linear regressor to the last hidden layer of a Deep Neural Network (DNN), marginalizing only the output weights of the net while using a point estimate for the remaining parameters. This results in adaptive basis regression, a well-established statistical technique which scales linearly in the number of observations, and cubically in the basis function dimensionality. Consequentially there is an explicit trade-off between evaluation time and model capacity. The resulting algorithm DNGO (Deep Networks for Global Optimization available online [1]) has been extensively tested for optimizing the hyperparameters of Deep Convolutional Neural Networks. Empirical results show that DNGO provides the same modelling properties of a GP, but with a significantly lower computational cost which is linear in the number of function evaluations. The following figure shows ANN and DNN for creating alternative surrogate models within BO framework and allows for a comparison of predictive mean (solid blue line) and uncertainty (shaded blue region). The first line figures show the surrogate models generated by DNN with three hidden layers, the first one using only Tanh as activation function and the second one using only Sigmoid as the activation function.

[1] `https://github.com/Anmol6/DNGO-BO`

The second line figures show the surrogate models generated by ANN with one hidden layer with the same activation functions. Moreover, in (Springenberg et al., 2016) Bayesian Neural

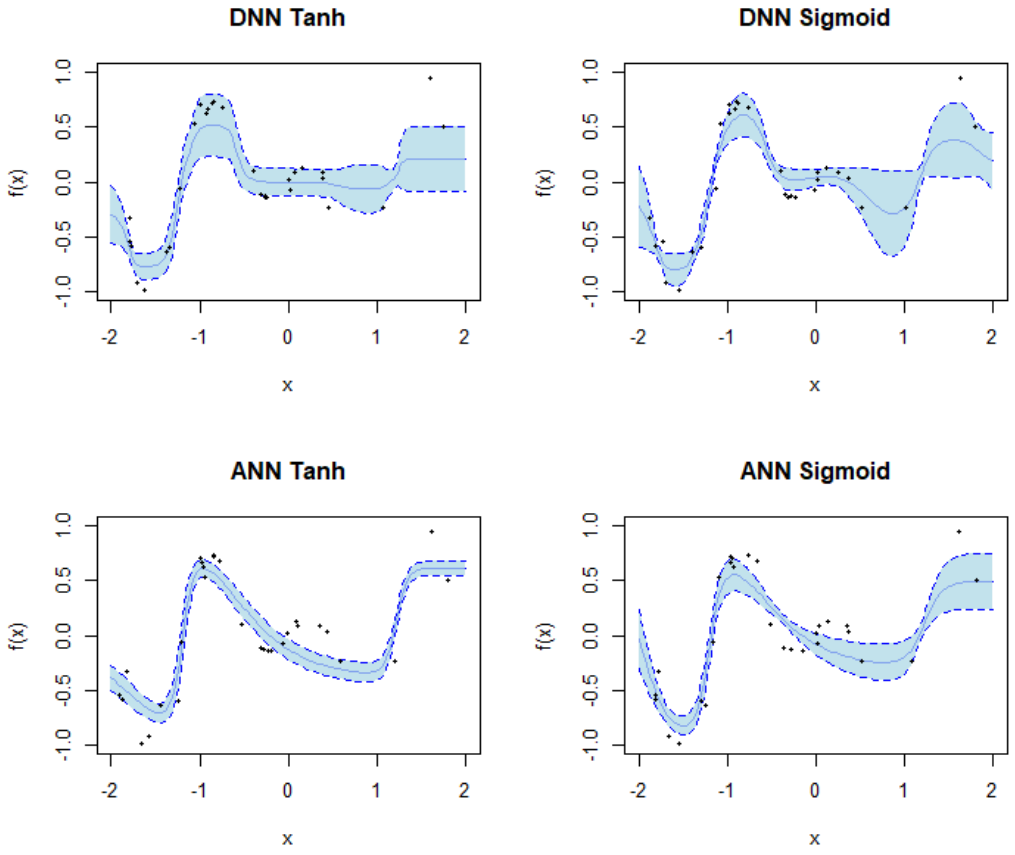


FIGURE 4.12: Probabilistic surrogate models based on Deep and Artificial Neural Networks, with different activation functions (i.e., sigmoid and tanh) for the neurons (Archetti and Candelieri, 2019)

Network has been suggested as one of the principled alternatives to GP. The algorithm is called BOHAMIANN (Bayesian Optimization with HAMIltonian Artificial Neural Network) and has been tested with a three layers neural network with 50 tanh units. The method has been presented also for multi-task optimization (i.e., finding the set of optimizers for k black-box functions, each with the same domain X). In order to deal with non-stationarity, a possible approach is to use Deep Gaussian Processes. In (Hebbal et al., 2019) functional composition of stationary GPs is proposed, providing a multiple layer structure.

4.2 Learning and optimizing: dealing with the exploitation-exploration dilemma

The main goal of the acquisition function is to drive the sequential search process towards more promising locations, depending on the predictions, and the associated uncertainty, provided by the (probabilistic) surrogate model conditioned on the current knowledge about $f(x)$.
At each iteration, the next location to evaluate, n_{n+1}, is chosen by balancing between two different - often antagonistic - goals: *exploiting* current knowledge with the aim to find a better solution than the current optimal one (aka *incumbent*) (i.e., optimizing) and *exploring* the search space to improve the quality of the approximation of $f(x)$ (i.e., learning).
Choosing the next location x_{n+1} requires to solve an (internal) optimization problem, whose computational cost is anyway negligible with respect to evaluating (i.e., *query*) $f(x)$ at a given location. The basically inexpensive optimization of the acquisition function is implied by the adoption of the probabilistic surrogate model. However, depending on the modelling strategy, the method used for optimizing the acquisition function can be different (e.g., a gradient-based method for GP and an evolutionary algorithm for RF). The really important aspect is that the overall BO framework still holds whichever is the modelling strategy adopted and the specific acquisition function used.
This section summarizes the most common acquisition functions proposed to effectively deal with the *exploitation-exploration* dilemma and the most recent advances in the literature.

4.2.1 Acquisition function: choosing the next ML model to train and validate

More than a decade ago, searching for an optimal ML model for a given dataset was an expensive and time consuming trial and error approach. This totally inefficient methodology was only possible for a few people who had the knowledge of the performance impact of choosing a particular model and its hyperparameters configuration. Obviously this inefficient trial-and-error approach was not ideal because, as already remarked in this thesis, finding an accurate ML model for a given task can be expensive in terms of both computational resources and time.
It is interesting to remark that recently many studies have compared the strategies adopted by humans optimizing black-box functions against other global optimization algorithms, such as Genetic Algorithm, Particle Swarm Optimization, DIRECT and BO, among others. In (Candelieri et al., 2020c; Wilson et al., 2018) it was empirically demonstrated that BO behaves like humans' strategies, with the optimal solutions identified by BO significantly closer than other optimization strategies to the optimal solution identified by humans. Moreover, humans and BO also resulted more sample-efficient, identifying solutions with better values in a limited number of trials with respect to Genetic Algorithms, Particle Swarm Optimization and other deterministic algorithms such as DIRECT.

Considering BO as the best strategy to emulate the human's decision making process in a black-box setting has been leading the scientific community to propose more sophisticated exploration-exploitation trade-off mechanisms and, consequently, novel acquisition functions.
For a better presentation, acquisition functions have been organized in two different "families" depending on what they search for: the "optimum" (Section 4.2.2) and the "optimizer" (Section 4.2.3), respectively. From a ML point of view, the acquisition functions belonging to the first family search for the optimal value of the generalization performance metric, while those belonging to the second family search for the ML algorithm and/or hyperparameters configuration having the best value of the generalization performance metric. Although they solve the same problem - and the difference seems to be just a matter of formalism - the two different formulations lead to completely different approaches, with their *pros* and *cons*.

4.2.2 *Improvement-based* acquisition functions: searching for the optimum

Probability of Improvement

Probability of Improvement (PI) was the first acquisition function proposed in the literature (Kushner, 1963):

$$PI(x) = P\big(f(x) \leq f(x^+)\big) = \Phi\left(\frac{f(x^+) - \mu(x)}{\sigma(x)}\right) \tag{4.19}$$

where $f(x^+)$ is the best value of the objective function observed so far (aka "best seen"), x^+ is the current optimal solution (aka incumbent), $\mu(x)$ and $\sigma(x)$ are mean and standard deviation of the probabilistic surrogate model, such as a GP, and $\Phi(\cdot)$ is the normal cumulative distribution function. Finally, the next point to evaluate is chosen according to: $x_{n+1} = \arg\max_{x \in X} PI(x)$.
One of the drawbacks of PI is that it is biased towards exploitation. To mitigate this effect one can introduce the parameter ξ which modulates the balance between exploration and exploitation. The resulting equation is:

$$PI(x) = P\big(f(x) \leq f(x^+) + \xi\big) = \Phi\left(\frac{f(x^+) - \mu(x) - \xi}{\sigma(x)}\right) \tag{4.20}$$

More precisely, $\xi = 0$ is towards exploitation while $\xi > 0$ is more towards exploration. The following figure shows how the selected location, x_{n+1}, changes depending on ξ. Ideally ξ should be adjusted dynamically to decrease monotonically with the function evaluations.

A weakness of PI is that it assigns a value to a new point independently on the potential magnitude of the improvement. This is the reason why the next acquisition function was proposed.

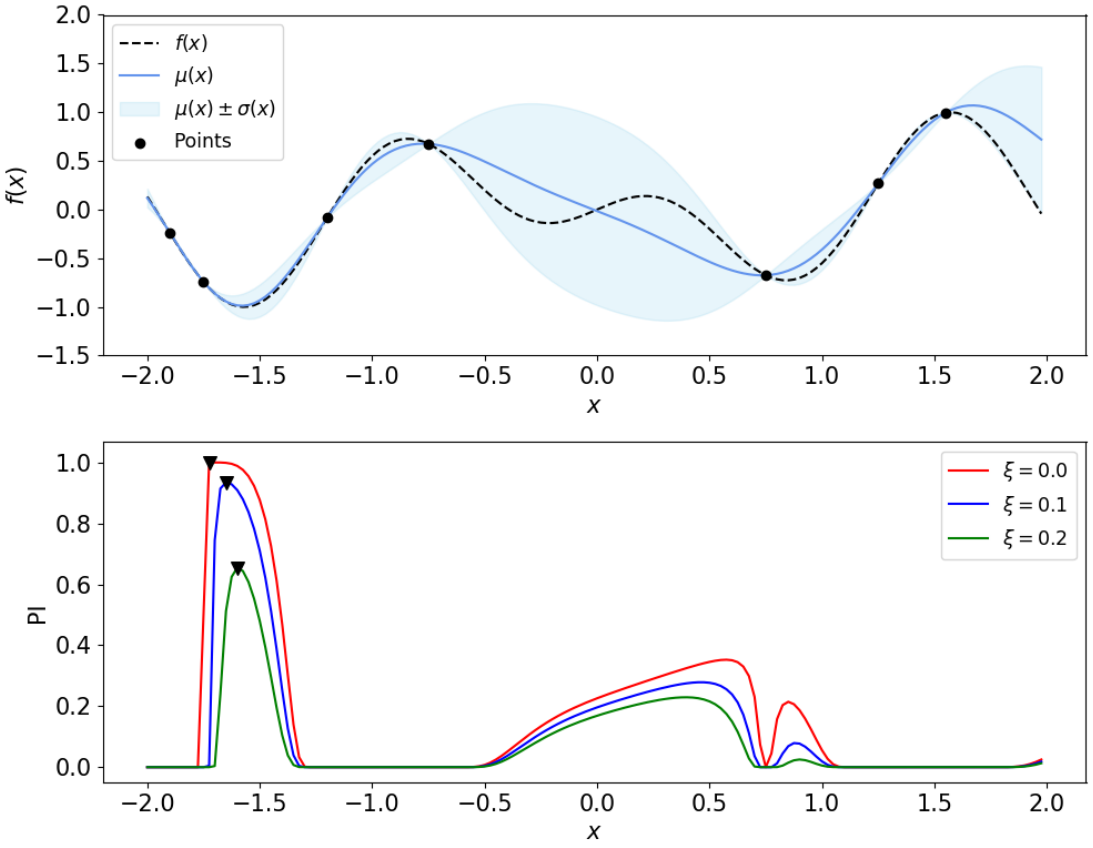


FIGURE 4.13: GP trained depending on 7 observations (top), PI with respect to different values of ξ and max values corresponding to the next point to evaluate (bottom)

Expected Improvement

Expected Improvement (EI) was initially proposed in (Močkus, 1975) and then made popular in (Jones et al., 1998) which measures the expectation of the improvement on $f(x)$ with respect to the predictive distribution of the probabilistic surrogate model.

$$EI(x) = \begin{cases} \big(f(x^+) - \mu(x)\big)\Phi(Z) + \sigma(x)\phi(Z), & \text{if } \sigma(x) > 0 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{4.21}$$

where $\Phi(Z)$ and $\phi(Z)$ are the probability distribution and the cumulative distribution of the standardized normal, respectively, where:

$$Z = \begin{cases} \frac{f(x^+) - \mu(x)}{\sigma(x)}, & \text{if } \sigma(x) > 0 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{4.22}$$

EI consists of two terms: the first is increased by decreasing the predictive mean; the second by increasing the predictive uncertainty. Finally, the next location to evaluate is chosen according to: $x_{n+1} = \arg\max_{x \in X} EI(x)$.

Although EI can automatically manage exploration and exploitation, it can nevertheless be formalized to allow active management of the exploration-exploitation trade-off by introducing the parameter ξ. When exploring, points associated to high uncertainty of the probabilistic surrogate model are more likely to be chosen, while when exploiting, points associated to low value of the mean of the probabilistic surrogate model are selected.

$$EI(x) = \begin{cases} (f(x^+) - \mu(x) - \xi)\Phi(Z) + \sigma(x)\phi(Z), & \text{if } \sigma(x) > 0 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{4.23}$$

and

$$Z = \begin{cases} \frac{f(x^+) - \mu(x) - \xi}{\sigma(x)}, & \text{if } \sigma(x) > 0 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{4.24}$$

The following figure shows how the selected location x_{n+1} changes depending on the values of ξ. Ideally ξ should be adjusted dynamically to decrease monotonically with the function evaluations.

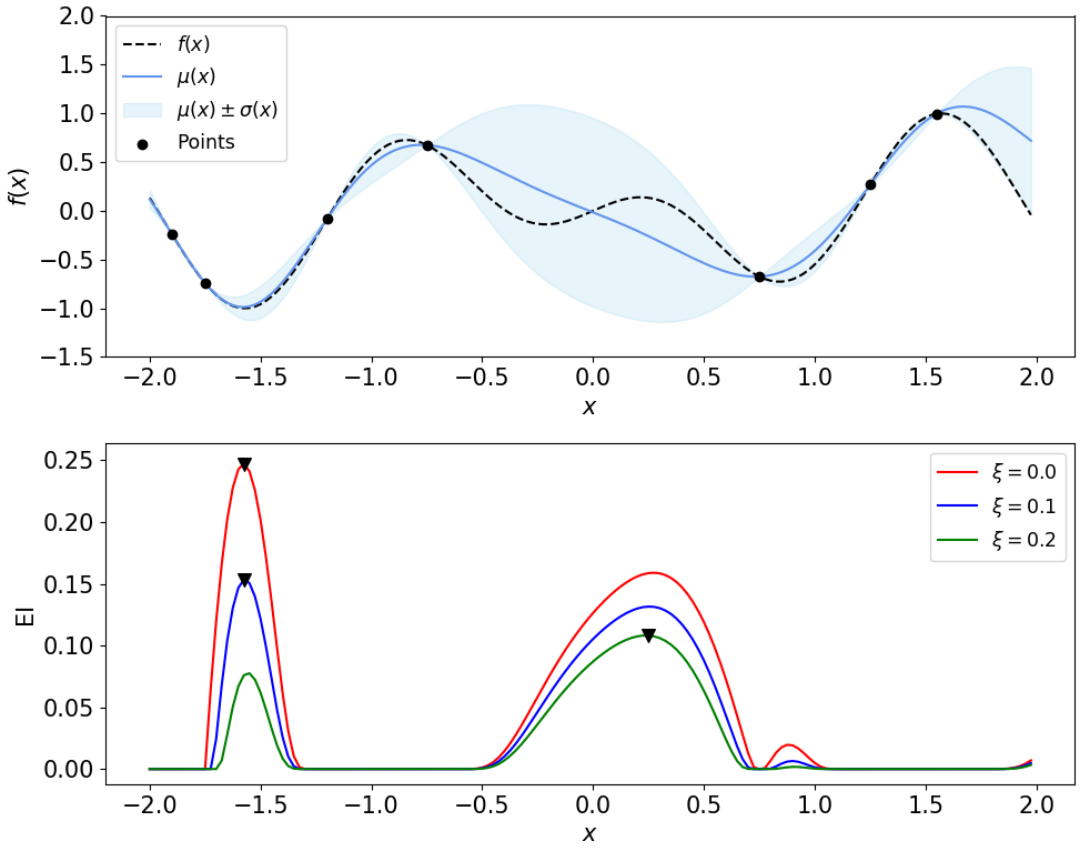


FIGURE 4.14: GP trained depending on 7 observations (top), EI with respect to different values of ξ and max values corresponding to the next point to evaluate (bottom)

Compared to PI, it is possible to notice that EI is anyway less biased towards exploitation than PI.

Upper/Lower Confidence Bound

Confidence Bound – where Upper and Lower are used, respectively for maximization and minimization problems – is an acquisition function that manage exploration-exploitation by being *optimistic in the face of uncertainty* (Auer, 2002).
In the case of minimization, LCB is given by:

$$LCB(x) = \mu(x) - \xi\sigma(x) \tag{4.25}$$

where $\xi \geq 0$ is the parameter to manage the trade-off between exploration and exploitation ($\xi = 0$ is for pure exploitation; on the contrary, higher values of ξ emphasizes exploration by inflating the model uncertainty as can be seen from following Figure). For this acquisition function

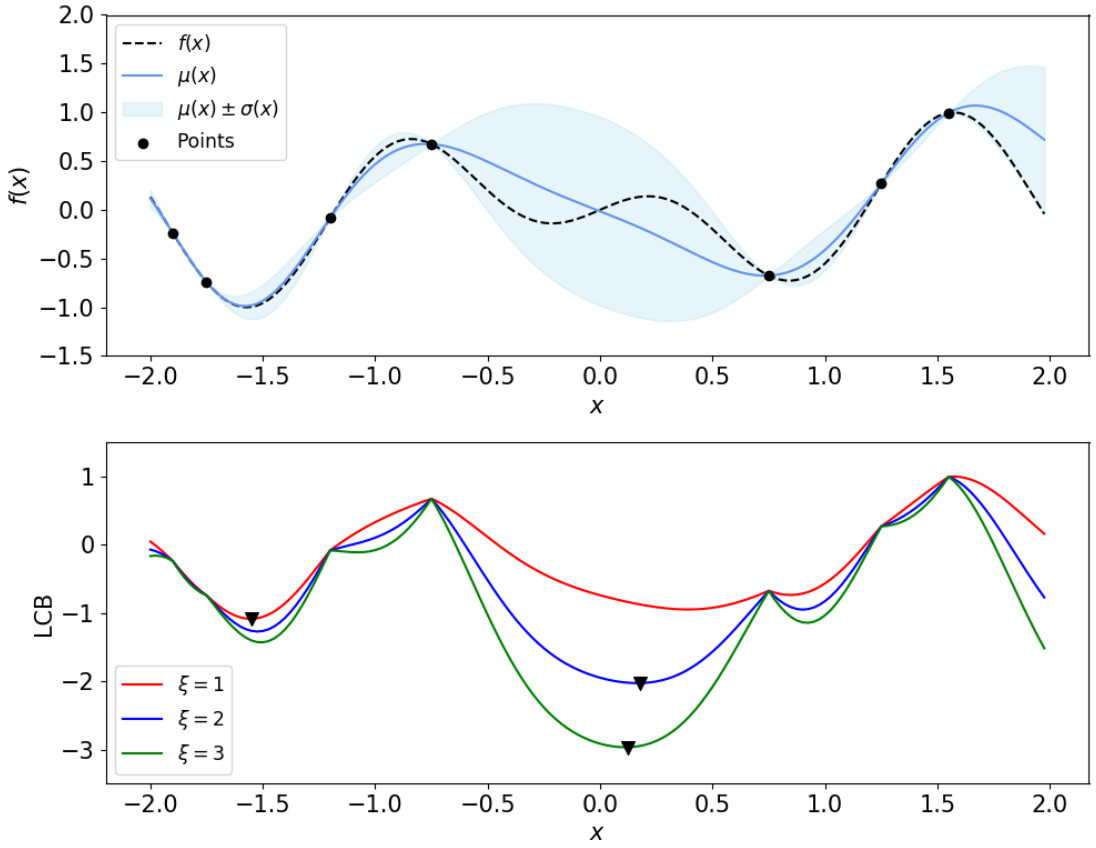


FIGURE 4.15: GP trained depending on 7 observations (top), LCB with respect to different values of ξ and min values corresponding to the next point to evaluate (bottom). Contrary to the other acquisition functions, LCB is minimized instead of maximized

there are strong theoretical results, originated in the context of multi-armed bandit, on achieving no-regret. More precisely, in (Srinivas et al., 2012) a scheduling of ξ - that is $\xi = \sqrt{\beta_n}$ in the quoted paper - is proposed to guarantee no-regret of the optimization process, with ξ increasing with the number of observations to encourage exploration and escape from local optima. Compared to PI and EI is clear the leaning of LCB towards exploration and how this effect is

augmented by an increase of ξ.

Finally, the next point to evaluate is chosen according to $x_{n+1} = \arg\min_{x \in X} LCB(x)$, in the case of a minimization problem, or $x_{n+1} = \arg\max_{x \in X} UCB(x)$ in the case of a maximization problem.

Knowledge Gradient

Knowledge Gradient (KG) is an acquisition function based on revising the typical assumption made in the BO process that the best evaluated point is returned as final solution. KG revises this assumption by allowing to return any point as a solution, even if it has not been previously evaluated. Another important difference with other acquisition functions, which are usually "risk-seeker", is that KG assumes risk-neutrality (Berger, 2013), meaning that any random x is evaluated depending on the expected value $f(x)$. The solution to choose after n function evaluations, stored in $D_{1:n}$, would be the one with the smallest $\mu_n(x)$ value.

Let denote this solution with $\bar{x}_n$; the value $f(\bar{x}_n)$ is random under the posterior, and has the following expected value conditioned to $D_{1:n}$:

$$\mu_n : \mu_n(\bar{x}_n) = \min_{x'} \mu_n(x') \tag{4.26}$$

If further function evaluations were available, we could sample x one more time and obtain the additional observation (x_{n+1}, y_{n+1}) and, consequently, the updated posterior mean $\mu_{n+1}(\cdot)$. The expected value of the solution after this further function evaluation would be $\mu_{n+1} = \min_{x'} \mu_{n+1}(x')$. Thus, the improvement in conditional expected solution value is given by: $\mu_n - \mu_{n+1}$. As this quantity is unknown, before evaluating $f(x_{n+1})$, we can only compute its expected value, conditioned to the previous observations at $x_1, ..., x_n$. This quantity is named Knowledge Gradient and can be computed at any x:

$$KG_n(x) = \mathbb{E}\big[\mu_n - \mu_{n+1} | x_{n+1} = x\big] \tag{4.27}$$

The simplest way to compute KG is through simulation: a possible value y_{n+1} is simulated at a certain x_{n+1}, then the optimum of the new posterior mean μ_{n+1} is computed, by considering the sampled value y_{n+1} as the actual value obtained through function evaluation at x_{n+1}. The simulation can be performed either by Thompson Sampling or directly by using $\mu_n(x_{n+1})$ and $\sigma_n(x_{n+1})$. Finally, this value is subtracted to μ_n to obtain the corresponding improvement in solution quality. The entire procedure is repeated many times and the differences $\mu_n - \mu_{n+1}$ are averaged on the simulated values y_{n+1}. This allows to compute the expected value required for the computation of the estimate of $KG_n(x)$, that converges to the actual value as the number of samples increases.

Therefore, contrary to other acquisition functions, KG computes the posterior not only at the sampled points but on the entire domain, estimating how a new function evaluation will change

the posterior. KG assigns a positive value on measurements that cause the minimum of the posterior mean to improve. According to (Frazier et al., 2009) this provides a small performance benefit in the case of BO with noise-free evaluations, but a substantial improvement in problems with noisy, derivative, multi-fidelity observations and other "exotic" problem features. In these cases, the value of sampling comes not through an improvement in the best solution at the sampled point, but through an improvement in the minimum of the posterior mean across possible solutions. For instance, a derivative observation can provide information that the function is decreasing, along a specific direction, in the neighbourhood of the sampled point. Consequently, the minimum of the posterior mean could be significantly smaller than the previous minimum, even if the function value at the sampled point is worse than the best previously sampled point. In this cases KG can reportedly outperform EI (Poloczek et al., 2017; Wu and Frazier, 2016; Wu et al., 2017b). In particular, the following Figure 4.16 shows that both KG and EI, computed on a GP model with derivatives, make different sampling decisions during the optimization process (Wu and Frazier, 2017).

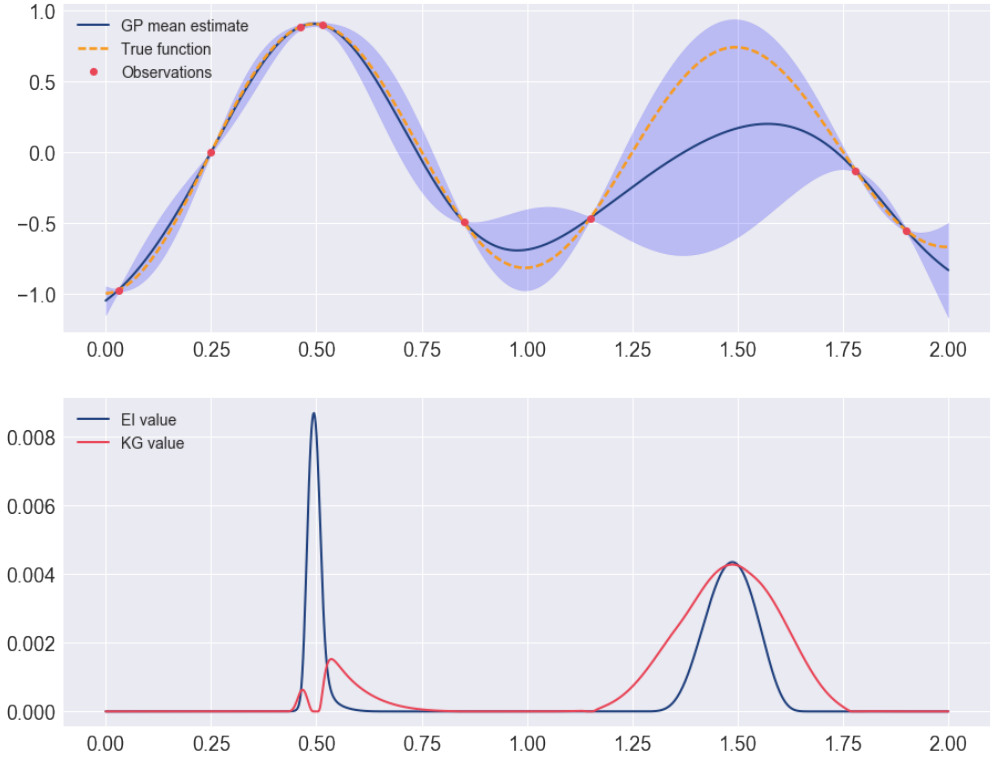


FIGURE 4.16: Comparison between KG and EI on a maximization problem. The EI acquisition function prefers to sample around region around 0.5 and the KG policy prefers to sample around the region around 1.5. Thus, KG gives a chance to exploration while EI is biased towards exploitation [Source Accessed on: 19/09/2020]

Portfolio allocation

Another aspect to consider when choosing an acquisition function for BO framework is that some times the same acquisition function is not necessarily the best choice for the whole optimization process. An interesting mixed strategy has been proposed where the acquisition function is chosen, adaptively at each iteration, from a "portfolio" (Brochu et al., 2010).

The selection mechanism is formalized as an on online multi-armed bandit problem: different strategies are investigated with the result that a hierarchical hedging approach, is the winning one. The basic algorithm, so-called "GP-Hedge", summarized in Algorithm 2.

Empirical evidence shows a significant improvement over EI and LCB, which is anyway offset by the much higher computational cost. These acquisition functions can be extended to leverage the knowledge of $f(x)$ into a more efficient search for x (Nguyen and Osborne, 2020).

Algorithm 2 GP-Hedge

input : $D_{1:n} = \{(x_1, y_1), ..., (x_n, y_n)\}$ - is the initial set of evaluated point;
N - is the overall number of available function evaluations (aka budget);
M - is the number of acquisition functions in the portfolio
$g_n = 0$ - is the initial "gains" with $g_n \in \mathbb{R}^M$;

for $i = 1$ *to* N **do**

Compute the next point to evaluate, according to each acquisition function:
$x_{i,k} = \arg\max_x \alpha_k(x|D_{1:n})$, $k = 1, ..., M$

Select one point x_i among the alternative $x_{i,k}$, with probability:
$p_{i,k} = \frac{e^{g_{i[k]}}}{\sum_{l=1}^{M} e^{g_{i[l]}}}$ (i.e., soft-max policy)
where $g_{i[k]}$ is the gain of the k-th acquisition function at iteration i

Evaluate objective function, eventually with noise, $y_i = f(x_i) + \varepsilon$

Update the function evaluations dataset $D_{1:i} = D_{1:n} \bigcup \{(x_i, y_i)\}$

Update GP and update gains $g_{i[k]} = g_{i-1[k]} + \mu_i(x_{i,k})$

end

Other very recent acquisition functions

Recently, some papers have been proposing to generalize the acquisition mechanism by considering the exploration-exploitation dilemma as a bi-objective optimization problem: maximizing the predictive mean (exploitation) while maximizing uncertainty (exploration). For instance, in (Žilinskas and Calvin, 2019) global optimization has been considered with respect to the *theory of rational decision making under uncertainty*. The important result is that PI and EI are special cases of this bi-optimization framework, because they lay on the Pareto frontier of all the predictive mean and standard deviation pairs computed for – theoretically – every possible decision. The mean-variance framework has been also considered, more recently, in (Iwazaki et al., 2020), for multi-task, multi-objective and constrained optimization.

Moreover, taking a decision by randomly sampling from the Pareto frontier, as proposed in (De Ath et al., 2019) and (De Ath et al., 2020), empirically proved to outperform other acquisition functions. In addition in (Berk et al., 2020), the authors have recently obtained better performance by randomly sampling ξ, of UCB/LCB, from a given distribution instead of using the scheduling proposed by Srinivas. They proved that this allows to identify more suitable trade-offs between exploration and exploitation and to outperform "traditional" GP-CB on a range of synthetic and real-world problems.

The main motivation is that the Pareto frontier offers a set of Pareto-efficient decisions which can be significantly larger than those offered by other "traditional" acquisition functions.

Another interesting approach has been recently proposed in (Volpp et al., 2019) where the acquisition function is not defined a-priori, but learned through a neural network trained on several solved tasks, where inputs of the neural network is the posterior prediction of a GP.

4.2.3 *Information-based* acquisition functions: searching for the optimizer

Thompson Sampling.

Thompson Sampling (TS) is more a sequential optimization process *per-sé* rather than an acquisition function. It is included in this family because it is at the basis of most of the following *entropy-based* acquisition functions.

TS is based on the following steps: *(i)* updating a posterior depending on a set of observations, *(ii)* drawing a sample from the posterior as an approximation to the function to be optimized, *(iii)* minimizing this sample function to identify the next candidate point, *(iv)* evaluating the objective function at that point and *(v)* iterate. Clearly, TS is in itself a sequential decision making process, like BO, where the acquisition function is replaced by a sample from the current probabilistic surrogate model (e.g., GP, RF and others).

In the following Figure 4.17 is depicted the difference in the selected next location x_{n+1}, depending on TS and LCB. The sample function from the GP is quite different from LCB and, at least in this case, TS is less explorative than LCB. Indeed, TS is in principle exploitative due to

the sample minimization step.

The following Figure 4.18 shows another case where TS is, instead, more explorative than

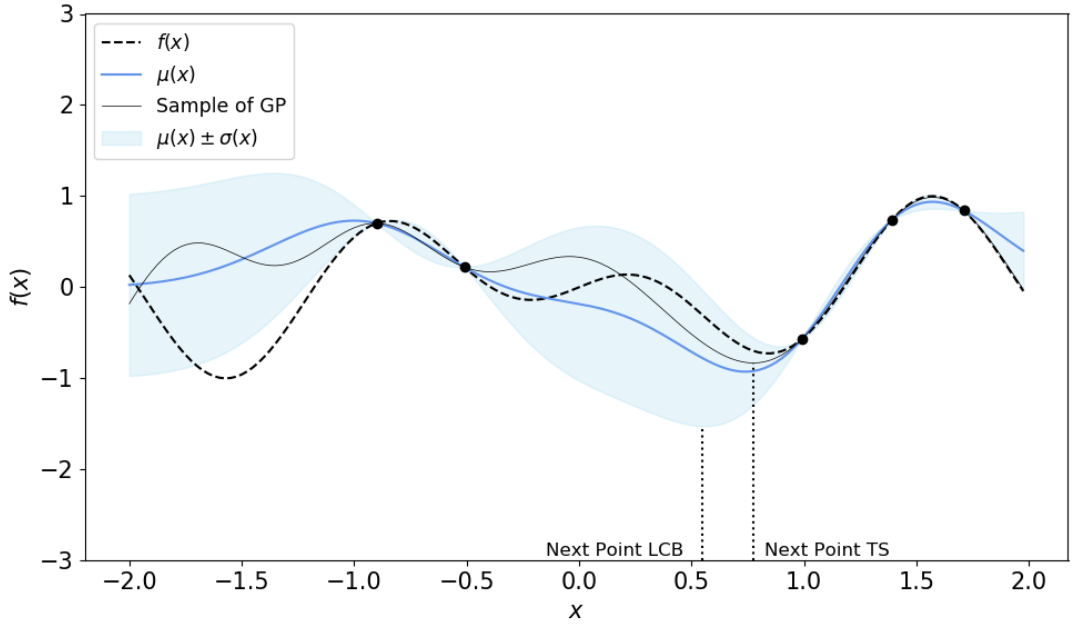


FIGURE 4.17: The next point to evaluate according to TS: the sample from GP posterior implies a more exploitative choice than LCB

LCB.

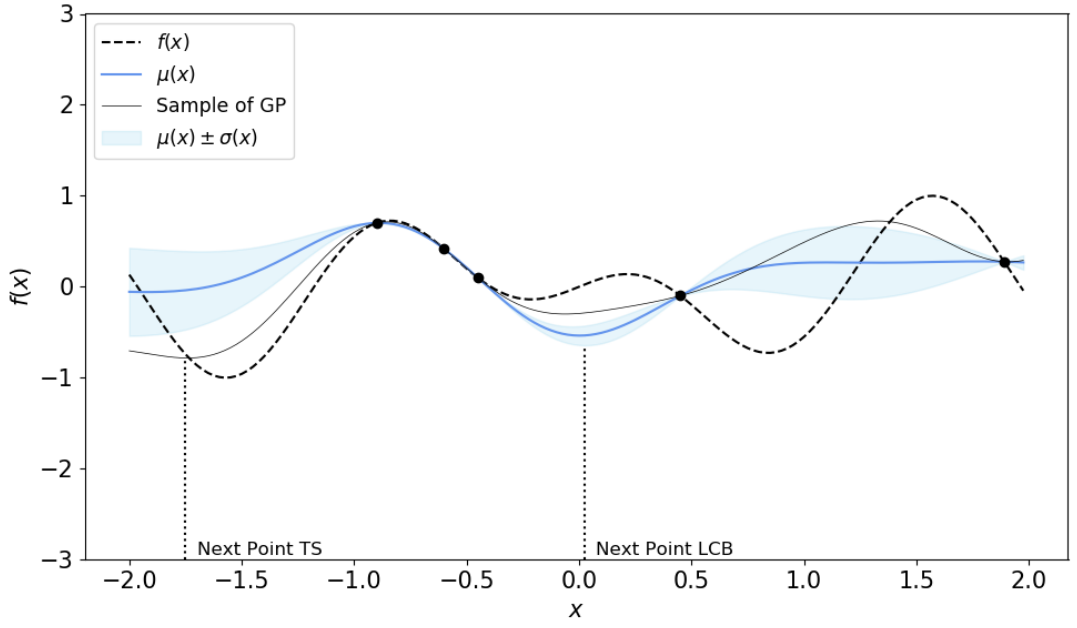


FIGURE 4.18: The next point to evaluate according to TS: the sample from GP posterior implies a more explorative choice than LCB

In (Russo et al., 2018) a theoretical analysis is provided, highlighting an analogy between TS and GP-CB. Moreover, another work (Basu and Ghosh, 2017) provides a convergence result of TS acquisition function when a random selection is taken with probability *epsilon* in order to avoid the risk of getting stuck in local optima during the optimization process.

Entropy-based acquisition functions

The *improvement-based* acquisition functions, presented in previous Section 4.2.2, are based on probabilistic measures of improvement in the f domain. Exploitation and exploration are represented, respectively, by the predictive mean and the "uncertainty bonus" represented by the standard deviation. On the contrary, *information-based* acquisition functions aim at selecting the next query to learn more about the optimizer (i.e., location of the optimum). This informational approach was originally proposed in (Villemonteix et al., 2009), and further developed into Entropy Search (ES) (Hennig and Schuler, 2012), and Predictive Entropy Search (Hernández-Lobato et al., 2014) (PES). In both ES and PES, the basic idea is to maximize the information about the global optimizer, according to information-theoretic perspective. The negative differential entropy is used to characterize the uncertainty and the query maximizing the reduction of uncertainty about the optimizer location is selected. More in detail, ES and PES are defined as follows:

$$ES(x) = H\Big(p(x^*|D_{1:n})\Big) - \mathbb{E}\Big[H\big(p(x^*|D_{1:n} \cup \{x,y\})\big)\Big] \tag{4.28}$$

$$PES(x) = H\Big(p(y|D_{1:n},x)\Big) - \mathbb{E}\Big[H\big(p(y|D_{1:n},x,x^*)\big)\Big] \tag{4.29}$$

where $H[p(\alpha)] = -\int p(\alpha)\log p(\alpha) d\alpha$ is the differential entropy.

While ES uses the expectation over $p(x^*|D_{1:n})$, PES uses the equivalent, symmetric formulation, where the expectation is over $p(y|D_{1:n},x)$. Both $p(x^*|D_{1:n})$ and its entropy are analytically intractable and must be approximated through expensive simulation. To add further complexity to the computation of this distribution, even if the global optimizer is unique, it is not stable, meaning that a small perturbation can result in an approximation far away from the global optimizer. The solution proposed in (Hernández-Lobato et al., 2014) is inspired by TS, and uses several samples from posterior, at every iteration, to optimize the acquisition function probabilistically. The following Figure 4.19 depicts how PES improves over ES, when compared to the ground truth.

The software for PES and MES can be downloaded at their respectively repositories[2][3].

[2]https://bitbucket.org/jmh233/codepesnips2014

[3]https://github.com/zi-w/Max-value-Entropy-Search

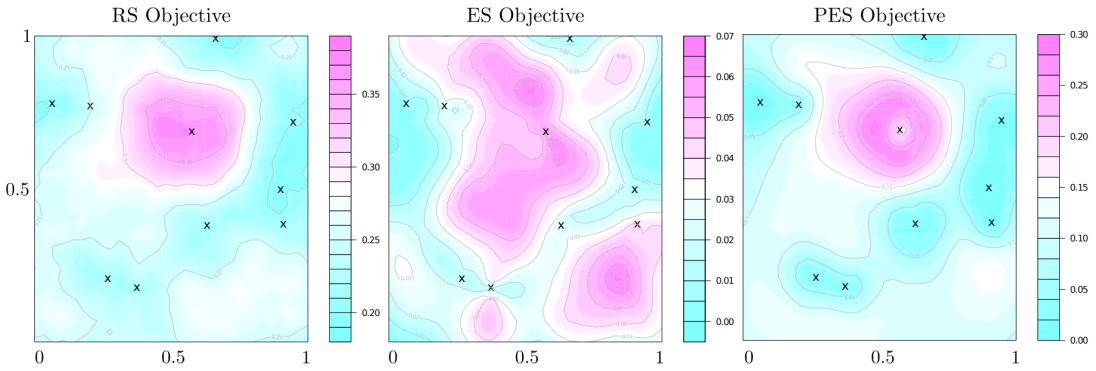


FIGURE 4.19: Ground truth (left), approximation by the ES (middle) and PES (right). (Hernández-Lobato et al., 2014)

In (Wang and Jegelka, 2017) a more computationally efficient and robust acquisition function, namely Max-value Entropy Search (MES) is proposed. MES is based on sampling from the conditional distribution of the global optimum, y^* (instead of the global optimizer, x^*), given the current set of observations. This would make MES conceptually closer to the *improvement − based* family, however it is introduced in this section for its strong link with ES and PES. Formally, MES is defined as follows:

$$MES(x) = H\Big(P(y|D_{1:n},x)\Big) - \mathbb{E}\Big[H\big(p(y|D_{1:n},x,y^*)\big)\Big] \tag{4.30}$$

It is important to remark that, while ES and PES rely on the expensive d-dimensional distribution $p(x^*|D_{1:n})$, MES relies the one-dimensional $p(y^*|D_{1:n})$, which is computationally much cheaper. Moreover, two sampling strategies are considered to make the entropy estimation efficient: sampling from a Gumbel distribution and Monte Carlo approach based on random features, respectively. Finally, an extension of MES to the high dimensional settings via additive Gaussian processes is also proposed.

Chapter 5

Addressing limitations of Bayesian Optimization in the development of AutoML and NAS applications

Despite the wide variety of GO methods, BO has become the *defacto standard* of AutoML tools, thanks to its effectiveness and sample efficiency. Indeed, it can identify an optimal solution within a limited number of trials (i.e., function evaluations). Despite the immediate advantages offered by BO to implement AutoML and NAS tools, there are still open issues limiting their adoption in the implementation of real-life ML and DL based applications.

As a first example, the generalization performance metric - that is the objective function for AutoML and NAS tasks - could be *not-computable* for some specific ML/DL algorithm and/or configuration of its own hyperparameters. This means that the objective function cannot be evaluated within some sub-regions of the search space and, even more complicated, this portions are not known a-priori: *computatbility* is black-box, as well as the objective function itself. In AutoML and NAS, *not-computatibility* of the objective function can occur quite commonly: for instance, training and validating a certain DNN may exceed some prefixed limit in the available computational time, or computational crashes may occur (e.g., out-of-memory or overflow errors). In both cases, the premature termination of the training-and-validation procedure leads to a missing computation of the objective function for that given DNN.

Another example is related to constraints limiting the *deployability* of the trained and validated ML/DL model. In this case the objective function is usually computable, but some other constraints must be satisfied, associated to the system where the (optimal) trained ML/DL model is going to be run. Typical *deployability constraints* are related to a threshold on the *prediction time* (i.e., time required by the model to perform its inference process and generate the output for a given input) or limits on the hardware resources available on the system, such as RAM, ROM, and battery.

Constrained BO is analysed in this Chapter from a theoretical perspective, and it is linked to the main contribution of this PhD thesis (presented in Chapter 6): a constrained-BO approach for AutoML of models to be deployed on tiny devices - specifically Micro-Controller Units (MCUs). Moreover, this Chapter also analyzes other relevant challenges recently arisen in the literature and still open or partially covered by recent works. More in detail, *Multi-Fidelity* and *Multi-Information Source Optimization* (MISO) are considered, with a focus on AutoML on large-scale datasets. Then, *safe optimization* and optimization in high-dimensional search spaces are also considered.

5.1 Not-computability of the objective function

A typical issue arising in training and validating a ML/DL model is the possible not-computability of its generalization performance metric. While it is only disappointing for a data scientist to manually search for the optimal configuration of his/her model, instead it could be detrimental in an AutoML and NAS task.

This issue is also known as *trainability*, in the ML community, that is the ability of ML algorithm or a ML pipeline to be *trainable* on a given dataset with specific hyperparameters configurations. In a recent paper (Nguyen et al., 2020), the topic of trainability of ML pipelines has been addressed. A pipeline composition and optimization requires a tremendous amount of time that prevents from exploring complex pipelines to find better predictive models. The authors of (Nguyen et al., 2020) have conducted experiments adopting many current state of the art AutoML systems (e.g. AutoWeka, TPOT and Auto-sklearn), showing that many of the generated pipelines are invalid, and it is unnecessary to execute them to find out whether they are good pipelines or not. To solve this issue, they propose a novel approach called AVATAR to evaluate the validity (trainability) of ML pipelines using a surrogate model. They show that AVATAR combined with the current AutoML systems allows to accelerate the automatic composition and optimization of ML pipelines by quickly excluding invalid pipelines and increasing the performance of the predictive models.

In more general terms, a *not-computable* objective function is also known, in the GO community, as a *partially defined* objective function (Sergeyev et al., 2007).

Furthermore, *not-computable* aka *partially defined* objective functions are also quite common in simulation-optimization problems. An example is Pump Scheduling Optimization (PSO) in Water Distribution Networks (WDNs), aimed to identify a schedule for the pumps, by minimizing the energy-related costs while satisfying the water demand. An hydraulic software model of the WDN is needed to simulate the effects of any given schedule - both in terms of energy-related costs and hydraulic feasibility - while the optimization mechanism must efficiently search for an optimal schedule among all the possible ones.

Recently, different strategies have been proposed for PSO (Candelieri et al., 2018; Tsai et al.,

2018). In (Candelieri et al., 2018) a naive BO approach was initially proposed, where a penalty value was assigned to the objective function in the case that it resulted not-computable (i.e., the schedule is not hydraulically feasible because water demand is not satisfied or pressure in the WDN is not within a working range). This was just a workaround and its main drawback is that the surrogate model can be significantly influenced by the choice of the penalty value, leading to a completely misleading approximation of the objective function. In (Tsai et al., 2018) an Adaptive Random Search (ARS) approach has been proposed to optimize the objective function while approximating the sub-region of the search space associated to computable (aka hydraulically feasible) schedules. However, the major drawback of this approach is the huge number of function evaluations it requires.

A more advanced and recent work is proposed in (Candelieri et al., 2019), where an SVM classifier is used to estimate the sub-region of the search space where the objective function is computable/defined. The approach is organized in two consecutive BO stages, characterized by two different objective functions: in the first stage, the next query is selected with the aim to improve the estimation of the feasible region; in the second stage, a "traditional" BO is performed, but only within the estimated *computable* sub-region. Therefore, the overall number of function evaluations to perform is divided over the two stages (suggested organization is 10% for initialization, 60% for stage 1 - namely, *feasibility determination* - and 30% for stage 2 - namely, Constrained BO). In any case, any query along the entire approach contributes to improve the estimation of the computable sub-region, while only the computable ones can contribute in approximating the objective function.

5.2 Black-box constraints

The not-computability of the objective function, described in the previous Section, can be considered as a specific case of a more general topic, known as *Constrained Global Optimization* (CGO) in the GO community. In CGO, an additional set of constraints is used, further restricting the typical box-bounded search space. The main difference, with the previous setting, is that the objective function can be in any case computed, even for an infeasible location. Otherwise, there is a one-to-one correspondence between computability and feasibility.

Although there are CGO works considering *white-box* constraints, they are not relevant for this thesis. AutoML and NAS are deeply characterized by their intrinsic black-box nature, in terms of both objective function and constraints. With respect to black-box constraints, another important difference is between constraints *coupled* and *decoupled* with the objective function. As the term suggests, a decoupled constraint can be evaluated independently on the objective function (and, if it is the case, on other constraints, also). This could lead to more efficient approaches well-suited for problems having an expensive objective function but cheap constraints (Ariafar et al., 2019; Gelbart et al., 2014; Hernández-Lobato et al., 2016).

Unfortunately, in ML/DL the most common types of constraints are *coupled* with the objective function, such as a threshold on the training and/or inference time, *deployability*, that is a guarantee that the resources required by the trained model are less than the hardware of the system on which the model is to be run.
With respect to constraints on the training/inference time, an example is the ChaLearn AutoML challenge (Liu et al., 2020) for which a new generation of AutoSklearn (Feurer et al., 2020), namely AutoSklearn 2.0, has been developed. The AutoSklearn 2.0 can generate an optimal ML model or ML pipeline on a given dataset in a fixed time and resources, adopting advanced meta-learning and efficient robust resource management.
It is important to remark that, in the AutoML community, a promising optimization strategy for resource-efficient AutoML/NAS tasks consists in adopting multi-objective optimization (Dong et al., 2018; Elsken et al., 2018; Zhou et al., 2018), where, instead of considering, at least two objective functions are optimized, jointly. Multi-objective optimization proved to be well-suited for all the applications where we are interested in searching for an optimal trade-off between two or more - often antagonistic - objectives. However, there is a large number of applications where we are interested in optimizing one objective while guaranteeing that the others are in a given range (without any need to optimize them). In these cases, it might make sense to deal with a constrained single-objective problem instead of a multi-objective one. In appendix B a toy example on solving the same problem as a constrained and multi-objective is provided. Obviously, the natural generalization is a constrained multi-objective setting, where some measures have to be optimized jointly while other must satisfy some constraints.
With respect to constrained single-objective optimization - under black box constraints - the *deployability* of ML models on tiny devices has been recently addressed in (Perego et al., 2020), significantly extending the Constrained-BO approach initially proposed in (Candelieri, 2019). The final goal is to efficiently search for an optimal DNN model which is also deployable and runnable on tiny/embedded devices. The approach, named AutoTinyML, is the major contribution of this PhD thesis, and it will be detailed in the next Chapter 6.

5.3 Multi-Source/Fidelity Optimization

This Section summarizes recent studies on BO approaches optimizing an expensive black-box objective function, $f(x)$, by querying multiple *information sources* which provide less expensive approximations of $f(x)$. The final goal is to optimize the original function while keeping low the overall cumulated query cost. This setting is known as Multi-Information Source Optimization (MISO). More formally, the reference problem for MISO is:

$$x^* = \underset{x \in X \subseteq \mathbb{R}^d}{\arg\min} f(x) \tag{5.1}$$

with $f(x)$ black-box, multi-extremal and expensive, and X the search space.
What makes MISO a different problem from GO is the availability of *sources* approximating $f(x)$, with a different cost for querying each of them. All the sources are also black-box and potentially multi-extremal. Let $\{f_1(x), ..., f_s(x), ..., f_S(x)\}$ denote the S different sources available, where $s = 1$ identifies the most expensive source: in the case that also $f(x)$ can be queried obtaining that $f_1(x) = f(x)$. In the following it was made this assumption without any loss of generality.
Let c_s denote the cost for querying the source $f_s(x)$, with $c_s > 0 \ \forall s = 1, ..., S$. One can always sort sources so that $c_s > c_{s+1}$. Since fidelity changes over the search space, this cost-based ranking does not imply a fidelity-based ranking or hierarchy of the sources.
Querying the source s at a certain location $x \in X$ leads to the observation $y_s = f_s(x)$, or $y_s = f_s(x) + \varepsilon_s$ in the case of a noisy setting, with $\varepsilon_s \sim \mathcal{N}(0, \lambda_s^2)$.

When the different sources come with an explicit information about their level of approximation, usually named *fidelity*, MISO specializes into *multi-fidelity* optimization, first introduced in (Kennedy and O'Hagan, 2000). Knowledge about fidelities can be exploited to sort hierarchically the sources in order to implement efficient and effective multi-fidelity optimization methods (Chaudhuri et al., 2019; Kandasamy et al., 2019; Marques et al., 2018; Peherstorfer et al., 2017; Sen et al., 2018). However, as already reported in (March and Willcox, 2012), a number of drawbacks can arise with respect to sources hierarchically organized: once one has queried a fidelity source at a location x, no further knowledge on can be obtained querying any other source of lower fidelity, at any location. Moreover, hierarchical organization requires the assumption that information sources are unbiased, admitting only *aleatoric* error that must be independent across sources. The mentioned drawbacks were addressed first in (Lam et al., 2015) who proposed an approach to generate a single model integrating the different information sources with fidelities changing over the search space. Thus, sources are not necessarily unbiased and independent and allow for *epistemic* error. More recently, (Poloczek et al., 2017) introduced a general notion of *model discrepancy* to quantify the difference between each source and the function to optimize, depending on the location.

MISO and multi-fidelity optimization have gained growing interest in AutoML: the seminal paper adopting MISO in ML is (Swersky et al., 2013), which proposes a method to use small datasets to quickly optimize the hyperparameters of a ML algorithm on a large dataset. Results proved that it is possible to transfer the knowledge gained from previous optimizations to new tasks in order to speed up k fold-cross validation. Successively, in (Klein et al., 2017),it is proposed an approach for hyperparameters optimization on large datasets that selects hyperparameters values and a dataset size, iteratively, in order to identify optimal hyperparameters values for the entire large dataset.

From the GO perspective, (Lam et al., 2015) was the first paper addressing location-dependent fidelities of the sources, removing the assumption about hierarchical relations across them. The approach uses a separate GP for each information source and then *fuses* their predictions – and associated uncertainties – through the method proposed by Winkler (Winkler, 1981), which came to represent the standard practice for the fusion of normally distributed data. The detailed process of estimating the correlation between the errors of two models, at the basis of the fusing procedure, is given in (Thomison and Allaire, 2017). The approach proposed in (Poloczek et al., 2017) uses a GP to capture the model discrepancy of each information source with respect to $f(x)$, while a single statistical model is used to perform BO jointly on the search space and the information sources. A kernel able to deal with both *location* and *source* is used to exploit correlations across different information sources. This allows to reduce the uncertainty on all information sources whenenver a new function evaluation is performed, even if it comes from a less accurate source.

Analogously to mentioned approaches, also (Ghoreishi and Allaire, 2019) use a GP for each information source and fuse them into a single statistical model. More precisely all the GPs are *fused*: the main contribution is the adoption of a two-step look ahead acquisition function for the selection of the next *source-location* pair to query. The main drawback in *fusing* GPs is that the computation of correlations requires to use a further set of N_f points, randomly selected, which determines both the computational complexity and the smoothness of the resulting fused GP. For instance, in (Ghoreishi and Allaire, 2019) N_f locations are used, even if the authors do not provide any information on how they have chosen this value.

More recently, in (Candelieri et al., 2020b) a MISO approach using GP *sparsification* instead of fusion has been proposed. The main contributions of the paper are:

- a new mechanism for generating a single model on the information sources, based on GP *sparsification* instead of *fusion*. A low complexity criterion (i.e., with compexity $O(1)$) is introduced to decide whether a function evaluation performed on a cheap source can be selected to "augment" the set of function evaluations on $f_1(x)$. The GP fitted on the augmented dataset is called *Augmented GP* (AGP): this entails a lower computational complexity than fusing GP.
- a new acquisition function to select the next *source-location* pair, by mixing together: the GP Confidence Bound, the cost of the source and the (location dependent) model discrepancy between the source specific GP and the Augmented GP.
- avoiding variance starvation, premature convergence to local optima as well as ill-conditioning in the GP training, by replacing, if needed, the acquisition function with a variance maximization step on the most expensive source.

- Computational experiments to confirm the actual performance of the MISO-AGP method on benchmark functions and the hyperparameter optimization of a SVM classifier on a large dataset (i.e., MAGIC GAMMA TELESCOPE dataset, available at the UCI Repository).

In Figure 5.1 is reported a one-dimensional MISO minimization problem, from which it is possible to observe the approximations of the objective function $f(x)$ using a GP model for each available information source (on the left) and the proposed AGP (on the right). It is evident that the approximation provided by the AGP model reduces the uncertainty significantly near the global minimum of $f_1(x)$.

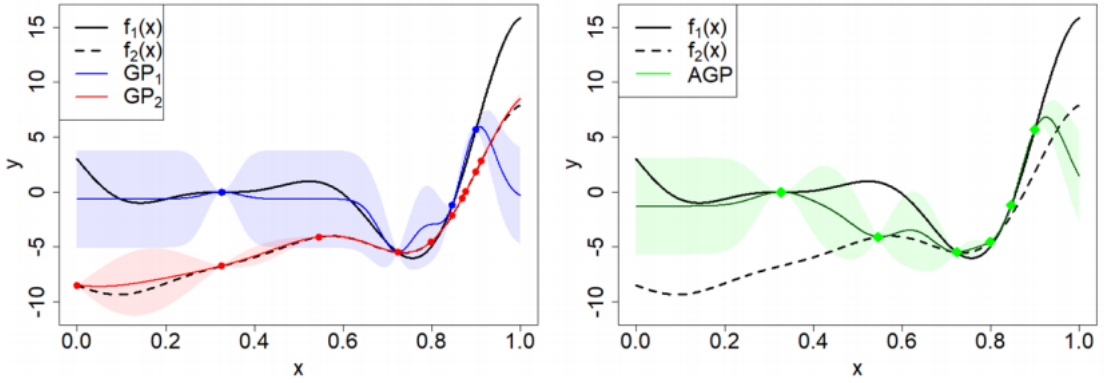


FIGURE 5.1: An example of AGP on a one-dimensional MISO minimization problem with two information sources (Candelieri et al., 2020b)

The following Figure 5.2 (from (Candelieri et al., 2020b)) shows the improvement offered by the proposed MISO-AGP approach, using two sources (i.e., the large dataset and a small portion of it, specifically 5% of the original one), when compared to BO performed on $f(x)$ only. Improvement is obtained in terms of both query cost (x-axis) and misclassification error on 10 fold-cross validation (y-axis). A wider set of results, including test functions and more than two sources, has been reported in the paper ahead of printing on Soft Computing[1].

[1] Candelieri A. Perego R., & Archetti F. *Green Machine Learning via Augmented Gaussian Processes and Multi-Information Source Optimization*, Soft Computing, Special Issue on Optimization and Machine Learning (2020) (ahead of printing)

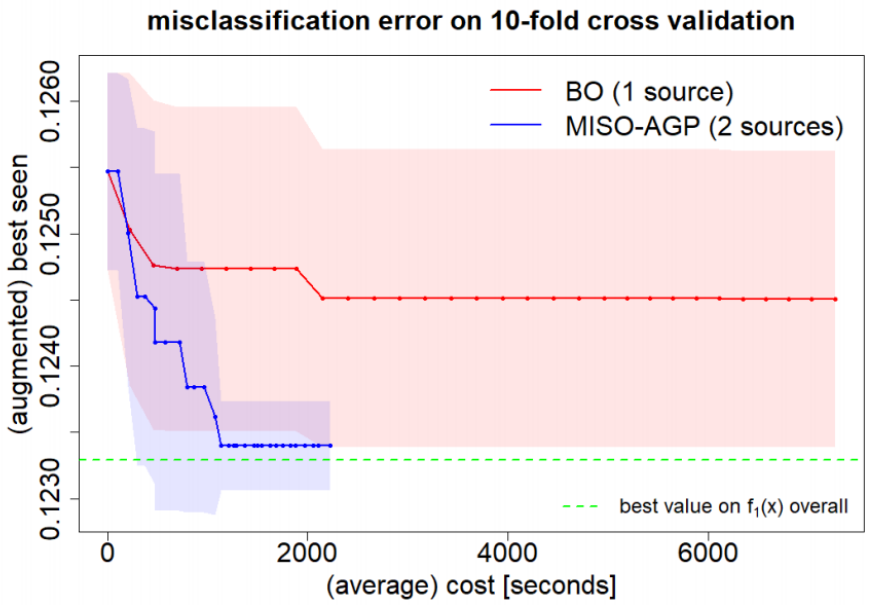


FIGURE 5.2: HPO of C-SVC on the MAGIC dataset. Comparison between traditional BO based HPO and MISO-AGP on two information sources. Results refer to 10 independent runs (Candelieri et al., 2020b)

5.4 Other challenging topics

Safe Optimization.

A topic closely related to Constrained Global Optimization is Safe Optimization, which can be included as part of the Bayesian optimization paradigm by defining itself as Safe Bayesian Optimization. In this kind of optimization approach a constraint is defined on the value of the objective function (i.e. not violating a given threshold) instead of on variables extending into the search space.

The important difference with respect to CGO is that Safe Optimziation does not allow – at least with a given probability – any function evaluations outside the feasible region. Moreover, in Safe Optimization the objective function is not necessarily partially defined: indeed, a function evaluation is "*unsafe*" if the corresponding value of the objective function violates the safety threshold.

An initial approach is presented in (Sui et al., 2015), with the SafeOpt algorithm. This novel algorithm tries to optimize taking into account the – supposedly known – Lipschitz constant of the objective function. An extension of the previous algorithm is presented in (Berkenkamp et al., 2016), where the SafeOpt algorithm is applied on high dimensional problems and without knowing a priori the information on Lipschitz constant.

The Safe Optimization can be formulated as maximization problem:

$$\begin{aligned} x^* &= \arg\max_{x \in X} f(x) \\ &\text{s.t.} f(x) \geq h \end{aligned} \tag{5.2}$$

where X is the domain of the f objective function, $f(x) \geq h$ is the safe constraint and h is a safety threshold.

The goal is to optimize the objective function by considering only "safe" points during the optimization process, i.e. points that satisfy the constraint $f(x) \geq h$. This is a more general formalization of the original SafeOpt (Sui et al., 2015), which used only a simple threshold on the value of the objective function, leading to a single constraint $g(x) = f(x) \geq h$.

A new algorithm, namely StageOpt, has been proposed in (Sui et al., 2018), as an extension of the original algorithm proposed in (Berkenkamp et al., 2016; Sui et al., 2015) to be more efficient and applicable to a broader class of problems.

The goal is to optimize an unknown utility function (aka objective function) $f : X \to \mathbb{R}$ from noisy evaluations at the sample points $x_1, x_2, \ldots\ldots, x_n \in X$. Any point in the sequence, when sampled, must be "safe", which means that for each one of m unknown safety functions $g_i(x) : X \to \mathbb{R}$ the sampled points lie above some safety threshold $h_i \in \mathbb{R}$. The final extended optimization problem can be defined as in equation 5.2 subject to the safety constraints $g_i(x) \geq h_i$ for $i = 1, \ldots, m$, where the m is the total number of safe constraints considered.

As in constrained optimization, GPs are used to model both the objective function and the constraints. An important assumption in StageOpt is that each safety function g_i is L_i-Lipschitz continuous, with respect to some metric on X. This assumption is usually satisfied by the most commonly-used kernels ((Srinivas et al., 2012), (Sui et al., 2015)). In order to work, both algorithms (SafeOpt and StageOpt) require at least one initial "seed" set of safe points that must be given, denoted as $S_0 \subset X$.

It should be underlined, however, that expanding the initial S_0 does not guarantee to identify the global optimum x^* specifically when the region that contains this optimum is disconnected from the initial region S_0.

For this reason the formalization of SafeOpt and StageOpt algorithm is based on the one-step reachability operator that can provide an expansion rule of the initial set S_0, and can be formalized as:

$$R_\varepsilon(S) = S \ \cup \ \bigcap_i \{x \in D \ | \ \exists x' \in S, \ g_i(x') - \varepsilon - L_i d(x', x) \geq h_i\} \tag{5.3}$$

where d is a distance measure (e.g. Euclidean distance) in $\mathbb{R}^d$ and ε is the absolute error in considering x as safe point.

This operator allows to identify the set of all the points estimated as safe depending on the previous evaluations of f on S. Then, given the maximum budget of function evaluations, N, the subset of X reachable after N iterations can be defined from the initial safe seed set S_0 as the

following: $R_\varepsilon^N(S_0) = R_\varepsilon(R_\varepsilon...(R_\varepsilon(S))...)$, N times.
Thus, the optimization goal becomes:

$$x^* = \arg\max_{x \in R_\varepsilon^T(S_0)} f(x) \tag{5.4}$$

In particular, StageOpt algorithm separates into two stages the Safe Optimization process: an exploration phase in which the safe region is iteratively expanded, followed by an optimization phase where the well-known Bayesian Optimization paradigm is adopted only into identified safe region. The approach of splitting into two different phases the optimization process is a common practice in constrained BO as reported in (Candelieri, 2019).
For both algorithms (SafeOpt and StageOpt) is adopted the well-known Upper Confidence Bound acquisition function for mapping into the GP the uncertainty at the generic iteration n, reported as follows:

$$Q_n^i(x) = \left[\mu_n^i(x) \pm \beta_n \sigma_n^i(x)\right] \tag{5.5}$$

where β_n defines the level of confidence. In particular, in the formula 5.4 the superscripts index corresponds to the m safety functions (for StageOpt), while the subscripts index is related to the iterations of the optimization process as usual. Then, to guarantee both safety and progress in safe region expansion, StageOpt uses the following confidence intervals $C_{n+1}^i(x) = C_n^i(x) \cap Q_n^i(x)$, with $C_0^i(x) = [h_i, \infty]$ so that C_{n+1}^i are sequentially contained in C_n^i for all $n = 0, \ldots, N$. Upper and lower bounds of C_n^i can be computed and denoted as u_n^i and ℓ_n^i respectively.
The first stage of StageOpt is safe region expansion, given by an increasing sequence of the safe subsets $S_n \subseteq X$ by the following formula based on the confidence intervals of the GP posterior:

$$S_{n+1} = \bigcap_i \bigcup_{x_i \in S_n} \left\{x' \in X \;\middle|\; \ell_{n+1}^i(x) - L_i d(x, x') \geq h_i\right\} \tag{5.6}$$

At each iteration, StageOpt computes a set of expanders points G_n (which contains the most promising points that are likely to expand the current safe region) and picks the expander with the highest predictive uncertainty. The definition of the expander set G_n is defined as follows:

$$\begin{aligned} G_n &= \left\{x \in S_n : e_n(x) > 0\right\} \\ e_n(x) &= \left| \bigcap_i \left\{x' \in X \setminus S_n \mid u_n^i(x) - L_i d(x, x') \geq h_i\right\} \right| \end{aligned} \tag{5.7}$$

Finally, at each iteration StageOpt chooses the most promosing point selects x_{n+1} defined by $x_{n+1} = \arg\max_{x \in G_n}(u_n^i - \ell_n^i)$. The second stage of StageOpt applies the common Bayesian Optimization approach within the identified safe region at the end of the first stage. In Figure 5.3 is illustrated a graphical comparison of the behavior of SafeOpt (Berkenkamp et al., 2016; Sui et al., 2015) and StageOpt starting from the same safe seed point.

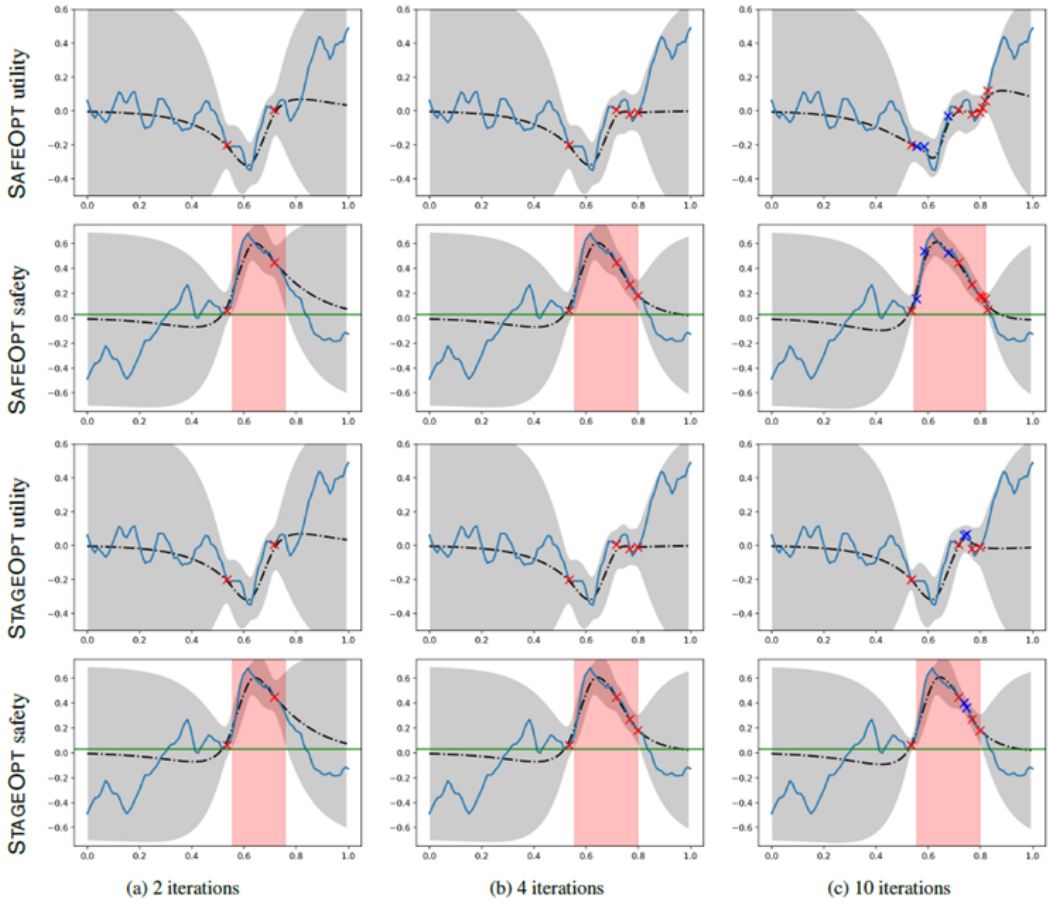


FIGURE 5.3: Evolution of GPs in SafeOpt and StageOpt for a fixed safe seed; dashed lines correspond to the mean and shaded areas to ± 2 standard deviations. The first and third rows depict the utility function, and the second and fourth rows depict a single safety function. The utility and safety functions were randomly sampled from a zero-mean GP with a Matern kernel and are represented with solid blue lines. The safety threshold is shown as the green line, and safe regions are shown in red. The red markers correspond to safe expansions and blue markers to maximizations and optimizations. We see that StageOpt identifies actions with higher utility than SafeOpt (Sui et al., 2018)

It should be highlighted that at current state of the art, nobody has considered a Safe Optimization problem with at the same time both constraints $c_i(x)$ and safety constraints $g_i(x)$. The main reason is that non-computable function evaluations are not allowed into Safe Optimization methods in contrast to constrained optimization methods, where this kind of evaluation is allowed. Another relatively recent work related to SafeOpt and StageOpt algorithms is proposed in (De Blasi and Gepperth, 2020) with the Self-Adaping Safe Bayesian Optimization (SASBO) approach. In this paper, the authors try to automatically tune the hyperparameters related to the GP surrogate model exploited during the optimization process. The hyperparameters currently considered into the tuning process are the variances and the length scale of the GP kernel currently fixed to be the Squared Exponential (Rasmussen and Williams, 2006). With this approach, the authors proved that while the exploration iterations are very conservative at the beginning of the optimization process, the updates of GP hyperparameters minimize the required iterations to find

the safe optimum.

Despite the various works created by (Sui et al., 2015) there are still limitations on the effective safe optimization guarantee. For example in (Sui et al., 2015) the authors prove to evaluate only points of the objective function that with high probability satisfy the safety constraint ($f(x) \geq h$), however they have not been able to provide guarantees of a complete safe optimization process in presence of noisy function evaluations ($f(x) + \delta$, where δ is noise component).

A recent work (Sergeyev et al., 2020) tries to fill this lack of guarantees enforcing the usage of only deterministic constrained approach with a novel approach so-called δ-Lipschitz framework.

They propose an optimization strategy based on two separate phases as already suggested in (Sui et al., 2018). The first phase, exploiting the L Lipschitz constant information, tries to find out the maximum safe region in the domain X where the function evaluations satisfy the safety constraints ($f(x) \geq h$). In the second phase, the proposed strategy, exploiting the identified safe region that could be composed by several sub-regions, optimizes using the Lipschitz optimization strategy that adopts the algorithm of Piyavskij proposed in (Piyavskii, 1972).

In particular, using a Lipschitz optimization approach, the information of the L Lipschitz constant must be known a priori. However, assuming that the Lipschitz constant is known, deterministic optimization approaches do not guarantee - exactly as the Safe Optimization paradigm - that only safe function evaluations are evaluated during the optimization process. For this reason certain assumptions have to be made in order to use the δ-Lipschitz framework.

First, to state the problem formally, it has to be supposed that a function $f(x)$ satisfies over a search domain $X = [a,b]$ the *Lipschitz condition*:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|,\ 0 < L < \infty,\ x_1, x_2 \in X \tag{5.8}$$

with an a priori known L Lipschitz constant. Then, given a safety threshold $h > 0$, it is required, in the presence of noise $\xi(x)$, to find an approximation of the point x^* and an estimate of the corresponding value $f(x^*)$ such that

$$\begin{aligned} x^* &= \arg\max_{x \in \Omega \subseteq X} g(x) \\ g(x) &= f(x) + \xi(x) \end{aligned} \tag{5.9}$$

where Ω is the safe region that can consist of several disjoint sub-regions Ω_j, $1 \leq j \leq m$, and the noise $\xi(x)$ is bounded by a maximum known value δ of noise. Based on this assumptions

the new domain of the noisy objective function $g(x)$ can be formalized as:

$$\begin{gathered} |\xi(x)| \leq \delta, \quad \delta > 0 \\ \Omega = \{x : x \in X, g(x) \geq h\}, \quad \Omega = \bigcup_{j=1}^{m} \Omega_j, \quad \Omega_i \cap \Omega_j = \emptyset, \quad i \neq j \end{gathered} \tag{5.10}$$

Thus, at each point x the value $g(x)$ can belong to the set

$$G(x) = \{y : y = f(x) + \xi(x), \quad \xi(x) \in [-\delta, \delta]\} \tag{5.11}$$

Based on new formalizations of the objective function $g(x)$ and its domain Ω, with the Lipschitz constant L can be construct the Lipschitz minorant function (Sergeyev and Kvasov, 2017), which is a deterministic approach to create a lower bound function of the $g(x)$.
However, since noise ($\xi(x)$) is applied to the objective function $f(x)$, the new formalization of the noisy objective function $g(x)$ (5.9) cannot satisfy the Lipschitz condition in 5.8, for this reason it is defined that $g(x)$ has to satisfy δ-*Lipschitz condition* as follows:

$$|s(x_1) - s(x_2)| \leq L|x_1 - x_2| + \delta, \quad 0 < L < \infty, \quad 0 < \delta < \infty, \quad x_1, x_2 \in X \tag{5.12}$$

Based on the previous new condition it is possible to formalized the mechanism for safe expansion. Supposing that $g(x)$ has been evaluated at several safe points $x_i \in \Omega$, $1 \leq i \leq k$, it can be formulated as a new δ-Lipschitz minorant function that satisfies Lipschitz condition (5.8) as follows:

$$\varphi(x, x_i) = g(x_i) - L|x_i - x| - 2\delta, \quad x \in X \tag{5.13}$$

where δ is defined in 5.10. The introduction of this new minorant function $\varphi(x, x_i)$ allows the δ-Lipschitz framework to safely expand the initial safe region taking into account noise, only considering points x_i that satisfy $\varphi(x, x_i) \leq f(x) - \delta$, where $i = \{1, ..., n\}$ sampled points.
Figure 5.4 shows the first phase of the algorithm δ-Lipschitz framework. In particular, the beginning and the end of expansion phase is showed on a one dimensional test function, where the green and red rectangles represent the safe and unsafe regions respectively identified by the algorithm. The figure on top shows how from the initial safe point x_1 it is possible to expand the safe region (green rectangle) by evaluating the points x_4 and x_5 which represent the boundaries of the safe region identified by the evaluation of $g(x_1)$ decreased by 2δ. Iteratively evaluating these two new points $g(x_4)$ and $g(x_5)$ generates a wider safe region (green rectangle), identifying the two new safe points x_6 and x_7 at the bounderies of this safe interval. This expansion process is repeated until the three stop criteria are reached. These criteria are: ν the maximum number of repetitions of the same point x_i on the boundaries of the safe region (because for the same point x_i can be obtained a different value of $g(x_i)$ due to noise $\xi(x)$), ε the minimum expansion value ($|x_1 - x_2|| \leq \varepsilon$), and σ tolerance with respect to the maximum range of values that can be

assumed by the evaluations of the same point x_i.

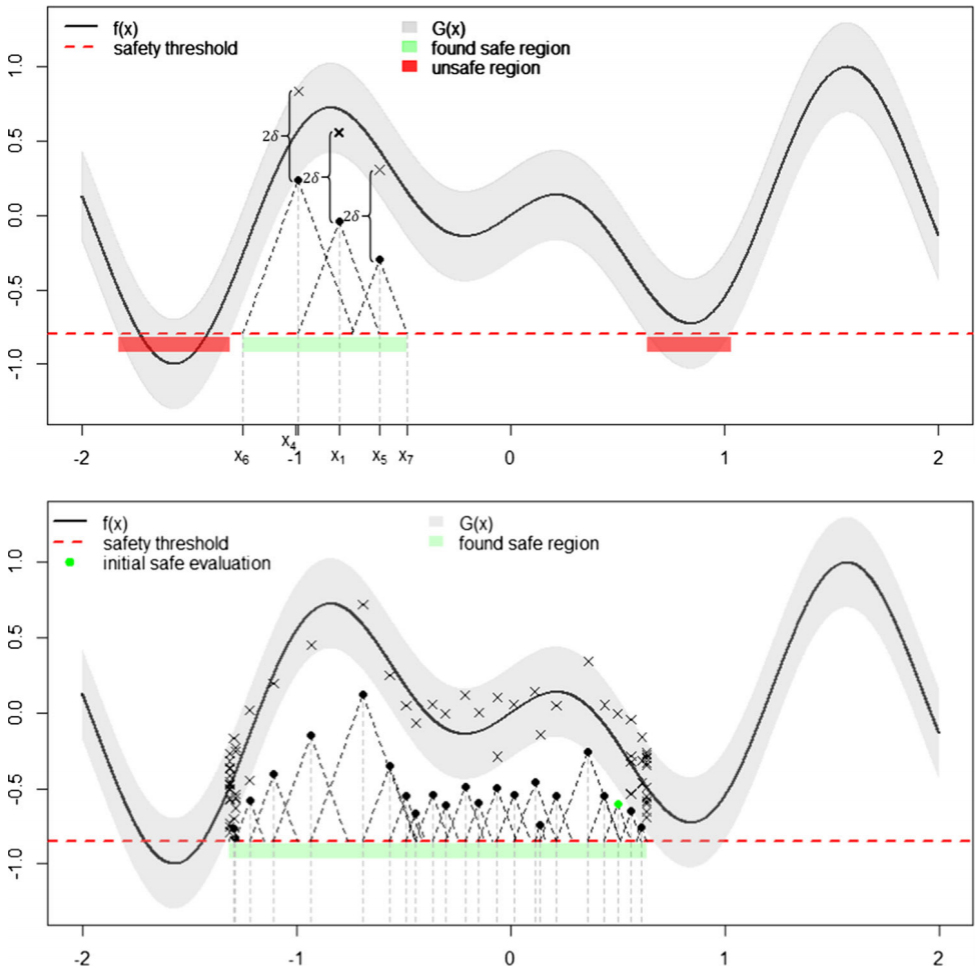


FIGURE 5.4: An example of safe expansion mechanism based on the first phase of δ-Lipschitz framework (Sergeyev et al., 2020)

After the identification of the maximum safe region $\hat{\Omega}$, where $\hat{\Omega} \subseteq X$, the second phase of δ-Lipschitz framework can be applied optimizing the following objective function:

$$x^* = \underset{x \in \hat{\Omega} \subseteq \Omega \subseteq X}{\arg\max}\ g(x), \quad g(x) = f(x) + \xi(x) \tag{5.14}$$

In this second phase currently only Lipschitz optimization approach (Piyavskii, 1972) is adopted, however, considering as the search space only the safe region $\hat{\Omega}$ any other more efficient optimization approach can be applied, such as Bayesian Optimization.

Although this system makes it possible to identify a search space that meets certain safety constraint, despite the fact that the objective function is subject to noise, this approach is not as sample efficient as other optimization strategies, such as Bayesian Optimization.

A future improvement in term of sample efficiency could be to adopt BO to optimize the noisy objective function in the second phase of the δ-Lipschitz framework. In addition, using the same

points already evaluated for the first phase of the algorithm can reduce the overall number of the function evaluations.

High Dimensionality.

Even if BO is the *defacto* standard, at least in the ML community, for the optimization of $\mathcal{L}$ loss functions like those arising in NAS or AutoML the computational complexity of the traditional BO approach, with respect to the dimensionality of the search space, is still a limiting factor. There are two basic ideas: one is that the objective function can be dealt with more efficiently by exploiting an underlying additive structure (Kandasamy et al., 2015), the second refers to embedding the original problem in a lower dimensional space in which the optimization is performed and then relay the results in the original space (Candelieri and Perego, 2019; Candelieri et al., 2020a; Griffiths and Hernández-Lobato, 2017; Li et al., 2017a; Qian et al., 2016; Wang et al., 2013).

The main contribution of high dimensional BO on objective functions with *additive dimensions* is proposed in (Kandasamy et al., 2015). This work assumes that the objective function $f(x)$ can be decomposed into individual functions, where each one is referred to a subset of decision variables, and shows that the additive form is also applied to $\mu(x)$ and to $K(x,x')$. The key contribution of the paper is a decomposed acquisition function Additive Gaussian Process Upper Confidence Bound (Add-GP-UCB), which makes it possible for BO with additive GP to achieve a demonstrably lower regret than standard UCB.

The formula of Add-GP-UCB acquisition function is as follows:

$$\tilde{\varphi}_t(x) = \mu_{t-1}(x) + \beta_t^{1/2} \sum_{j=1}^{M} \sigma_{t-1}^{(j)}(x^{(j)}) \tag{5.15}$$

Where $\tilde{\varphi}_t$ can be formalized as a sum of functions on orthogonal domains:

$$\tilde{\varphi}_t(x) = \sum_j \tilde{\varphi}_t^{(j)}(x^{(j)}) \tag{5.16}$$

Where

$$\tilde{\varphi}_t^{(j)}(x^{(j)}) = \mu_{t-1}^{(j)}(x^{(j)}) + \beta_t^{(1/2)} \sigma_{t-1}^{(j)}(x^{(j)}) \tag{5.17}$$

By this formulation the additive acquisition function $\tilde{\varphi}_t$ can be maximise by maximising each $\tilde{\varphi}_t^{(j)}$ on specific domain of each decision variable domain $X^{(j)}$.

The basic intuition is that as each individual function is drawn from a GP, it is possible to perform GP regression on each individual function to get individual estimates and estimate the final objective. Besides, the uncertainty of each individual approximation can be propagated to the original objective function and it drives a bound on the cumulative regret.

The method is based on the assumption, well grounded in statistics, that using an additive model has several advantages iff is not additive. Experimental results, for hyperparameter optimization

and face detection, confirm that in large-scale problems there is indeed an "hidden" additivity which can be computationally exploited.

In addition, in (Gardner et al., 2017) the authors propose a method based on MCMC sampling to discover the additive model underlying while exploring it in BO.

In a more recent work (Mutny and Krause, 2018) a generalized additive model is assumed. To make the optimization efficient and feasible a Fourier feature approximation is used. The error in this approximation decreases exponentially with the number of features and allows a precise approximation of posterior $\mu(x)$ and $\sigma(x)$. Indeed, this study provides a bound on cumulative regret for both TS and LCB/UCB. As an acquisition function is used Thompson Sampling, whose formulation allows to use first order optimizers.

A different decomposition is given in (Wilson et al., 2020). The authors remark that traditional approaches to sampling using μ and σ is statistically well behaved, but scale poorly owing to a need to invert increasingly large matrices. On the other hand, approximation strategies using Fourier features as in (Mutny and Krause, 2018) avoid costly matrix operations, but are prone to errors in predictive posteriors. Two representations are given of GP: the usual function space view and the weigh-space view, where $f(x)$ is given by a set of basic functions. In the function view the authors use a kind of sparsification through a set of inducing variables enabling a lower complexity in the inversion of the kernel matrix.

In the weight space approximation, f can be looked through an explicit set of basic functions. The kernel k can be viewed as the inner product in a Reproducing Kernel Hilbert Space (RKHS) and defined as:

$$k(x,x') = \langle \varphi(x), \varphi(x') \rangle_H \approx \phi(x)^T \phi(x) \tag{5.18}$$

Where $\varphi : X \to \mathbb{R}^\ell$ is a finite-dimensional feature map (Rasmussen and Williams, 2006).

A suitable feature can be constructed by a set of Random Fourier Features (RFF) as for instance $\varphi_i(x) = \sqrt{2/\ell}\cos(\theta_i^T + \tau_i)$ and defining Bayesian linear model as $f(\cdot) = \sum_{(i=1)}^{\ell} \omega_i \phi_i(x)$, where $\omega_i \sim N(0,1)$, we obtain an ℓ-dimensional GP approximation, where f is now a random function but its randomness is controlled by the distribution of weights ω.

Approximating the posterior $f|y$ as the weighted sum of basis functions, as in the formula above, yields advantages for sampling. Unlike the standard method we can now sample weight values rather than function values and each weight draw defines an actual function evaluable at arbitrary locations $x \in X$.

Another strategy to deal with high dimensional problems with BO is the representation of the dimensions of the problem under optimization remapped in another space exploiting the *embeddings dimensions*. There are several works (Li et al., 2017a; Qian et al., 2016; Wang et al., 2013) which suggest that not all the dimensions/variables of the problem to be optimised are equally relevant in terms of convergence towards the optimum. In fact in (Li et al., 2017a) they base the approach proposed as *Dropout BO* where only a handful of variables/dimensions are considered at each step of the sequential optimization process. This selection, as well as having the aim of

carrying out exploration to all effects, allows at the same time to make exploitation as it fixes the variables/dimensions of the problem with the best values seen so far. Instead of a brute dropout, random embedding (Qian et al., 2016; Wang et al., 2013) was suggested.
In (Candelieri and Perego, 2019) the authors propose to exploit the dimensionality reduction intrinsically offered by Deep Autoencoders and then perform BO in the induced latent space. A first attempt to combine autoencoders and BO was presented in (Griffiths and Hernández-Lobato, 2017) to encode continuous representations of discrete variables and, eventually, reduce dimensionality. Instead in (Candelieri and Perego, 2019) has been analyzed the interplay between BO and the two mappings computed by the autoencoder - from the original to the latent space (encoding) and viceversa (deconding). In order to avoid being stuck in local optima, it was concluded that adaptive sampling in the original space is required when exploration is nullified by (inaccurate) decoding.

TABLE 5.1: Table reports preliminary results obtained by the proposed approach (BOODLE) in (Candelieri and Perego, 2019)

	20 Dimensions			***30 Dimensions***		
	Rosenbrock Function (Dixon and Szegö, 1978)					
Optimizer	***Iterations***	***Optimal Value***	***Time (Sec.)***	***Iterations***	***Optimal Value***	***Time (Sec.)***
BOODLE	294 (±144)	**92.25 (±49.63)**	**5672.50 (±1503.58)**	410 (±71)	**238.93 (±112.54)**	**4178.67 (±815.79)**
BO	490 (±10)	6109.08 (±4801.85)	22367.55 (±495.76)	475 (±23)	13629.45 (±11201.16)	48180.39 (±5779.05)
	Schwefel 1.2 Function (Al-Roomi, 2015)					
Optimizer	***Iterations***	***Optimal Value***	***Time (Sec.)***	***Iterations***	***Optimal Value***	***Time (Sec.)***
BOODLE	260 (±132)	3.14 (±0.57)	**4671.74 (±718.21)**	244 (±127)	5.96 (±1.64)	**3428.87 (±732.43)**
BO	162 (±87)	**2.24 (±0.36)**	14379.29 (±4262.99)	292 (±111)	**3.14 (±1.84)**	36114.97 (±8481.30)

Preliminary results of the proposed approach in (Candelieri and Perego, 2019) for high dimensional Bayesian Optimization are reported in Table 5.1 into two famous optimization benchmark functions (Al-Roomi, 2015; Dixon and Szegö, 1978). Despite the promising results obtained on Rosenbrock test function with 20 and 30 dimensions respectively, for the Schwefel 1.2 the results were negative in terms of the optimal value reached at the end of the optimization process. The main reason for the poor performance of the proposed approach on this test function is due to the fact that Schwefel 1.2 consists of interactive dimensions, meaning that one dimension of the function depends on a combination of all the others.
Although observing Table 5.1 there is a clear gap between the time required to optimize using the proposed system (BOODLE) and the traditional BO.

Chapter 6

AutoTinyML

This Chapter is entirely dedicated to the introduction of a particular branch of Machine Learning defined as Tiny Machine Learning. For this particular Machine Learning several strategies are studied to obtain an hardware-energy efficient Machine Learning models that are executable on computers with limited hardware capabilities.
In this Chapter the main contribution of the research carried out during the PhD will be introduced, which also comprised a collaboration with STMicroelectronics[1] for the development of a constrained optimization system applicable to Deep Neural Networks for deployability on microcontrollers.

6.1 Tiny Machine Learning

To support pervasive computing, the most crucial requirement is the possibility to deploy Machine Learning (ML) models on small devices surrounding us. Typical examples are applications for object recognition, speech/audio recognition and motion activity recognition, with ML models to be deployed onto smart-home devices such as light switches, door handles, windows as well as personal/mobile devices such as smartphones, smartwatches and others.
Microcontroller sales have increased significantly in recent years, mainly driven by the Internet-of-Things wave. According to a report by IC Insight[2], there are around 250 billion microcontrollers in the world today, with 28.1 billion of devices sold in 2018 and an annual shipment volume which is estimated to grow to 29.6 billion by 2023: a massive untapped market.
One of the most relevant advantages of microcontrollers is that they can be programmed without requiring custom electronics for each task. This has given the chance to exploit the power of Machine Learning (ML) and Deep Learning (DL) for creating specific applications (e.g.,

[1] https://www.st.com/content/st_com/en.html
[2] https://www.icinsights.com/data/articles/documents/1289.pdf [Accessed on: 21/09/2020]

speech/audio recognition, image recognition, etc.) to run on microcontrollers. While the model training is usually performed on typically large computational platforms, also using specialized hardware (e.g., GPUs and TPUs), especially for DL, instead the *inference* is taken to microcontrollers.

Automated Machine Learning (AutoML) and the Neural Architecture Search (NAS) (Hutter et al., 2019) have been consolidating as the standard methods for model selection and hyperparameters optimization (HPO) of a ML algorithm and definition of novel DNN architectures, respectively, both with the aim to train accurate models.

However, AutoML and NAS can make a large use of computational resources for training - and validating - these models. Indeed, in (Strubell et al., 2019) the authors start to investigate how ML models, especially DNNs combined with NAS and/or HPO, can be hungry for energy power as well as for the data used for training. Moreover, the models obtained through AutoML and NAS are usually very accurate but require computational and/or memory resources which do not fit with *tiny* devices with limited hardware resources, typically embedded systems and microcontrollers.

According to the literature, several NAS approaches are available, such as Evolutionary Algorithms (AE) (Angeline et al., 1994; Bianco et al., 2020; Miikkulainen et al., 2019; Suganuma et al., 2017; Xie and Yuille, 2017), Random Search (Liu et al., 2018), Reinforcement Learning (Baker et al., 2017; Zhong et al., 2018; Zoph and Le, 2017) and Sequential Model-based Optimization (SMBO) - aka Bayesian Optimization (BO) (Bergstra et al., 2013; Domhan et al., 2015; Jin et al., 2019; Mendoza et al., 2016, 2019), but they do not consider any constraint about the deployment onto a device with limited hardware capacity. Indeed, a promising direction is to develop NAS methods for multi-objective problems (Dong et al., 2018; Elsken et al., 2018; Zhou et al., 2018), in which measures of resource efficiency are used as objectives along with the predictive performance on unseen data.

With respect to the industrial sector, many top Artificial Intelligence (AI) companies have been recently proposing AutoML and NAS systems to produce high-performance ML/DL models exploiting the cloud-based capacities, which are virtually unlimited for their nature. Some examples, already reported in Chapter 1, are Google AutoML Tables[3], Amazon SageMaker[4] and Microsoft AutoML [5].

Both the aforementioned research papers and the industrial solutions do not consider any hardware constraint arising in case that the optimized trained model should be deployed on tiny devices. Typically the main hardware constraints can include the storage capacity of the devices (as Read-Only-Memory ROM), the memory used for the computation (as Random Access Memory RAM), the latency of the inference based on the model and the energy consumption required by the device that could depend from a limited energy supply.

[3] https://cloud.google.com/automl-tables?hl=uk

[4] https://aws.amazon.com/sagemaker/

[5] https://www.microsoft.com/en-us/research/project/automl/

Thus, while many companies are currently leveraging on Cloud, data centers and specialized hardware to train accurate ML models, the need to deploy and run these models on tiny devices is emerging as the most relevant trend. Tiny Machine Learning (TinyML) is a sub-field of ML, aiming at bringing the power of AI to devices with limited computational resources.
Most of the widely adopted ML algorithms are not designed to provide learned models that are *deployable* on tiny devices such as Micro Controller Units (MCUs). For this reason, the TinyML community has been growing in the last years (Fedorov et al., 2019). An example of IT company supporting and investing on TinyML is STMicroeletronics, which developed an extension package of its software *STM32CubeMX* to allow data scientists to design accurate Deep Neural Networks while checking for the resources required to execute the trained model and validate its deployability on a specific STM32 MCU (Fedorov et al., 2019; Varenne et al., 2019).
Therefore, resource efficiency is an even more critical issue when the trained models have to run on tiny devices. The main idea is to consider the hardware resources of the small devices as constraints of the *AutoTinyML* problem, naturally leading to a new approach in the constrained optimization setting which could be more suitable than a multi-objective method.

6.2 Auto Tiny Deep Learning

6.2.1 Proposed Approach

The approach proposed in this research aims to extend AutoML to include deployability constraints into the optimization of a ML algorithm. Figure 6.1 compares: (top) a situation where AutoML is used to obtain an accurate model whose deployability is validated only at the end of the process, against (bottom) the proposed AutoTinyML schema, which requires to validate the deployability of every model generated over the optimization process.

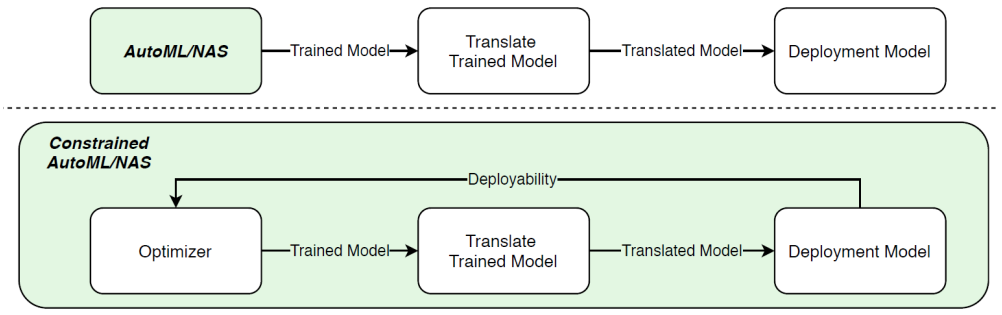


FIGURE 6.1: Above the dashed line: AutoML/NAS applied to train an optimized model and a final check for its deployability on tiny device. Below the dashed line: the inclusion of deployability constraints into a Constrained AutoML/NAS framework (i.e. AutoTinyML)

Into AutoTinyML task the problem to optimize can be formalized starting from equation 2.2 as follows:

$$\begin{aligned} \gamma^* &= \arg\min_{\gamma \in \Gamma} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(\mathcal{N}_\gamma, D_{train}^{(i)}, D_{val}^{(i)}) \\ s.t. & \\ & c_j(\gamma) \leq 0 \quad \forall j = 1, .., n_c \end{aligned} \tag{6.1}$$

Where:

- $\mathcal{N}_\gamma$ is the DNN architecture whose hyperparameters – represented by γ – are to be optimized;
- $\Gamma = \{\Gamma_1, ..., \Gamma_n\}$ is the overall search space for the hyperparameters, where Γ_i is the domain for the single hyperparameter γ_i;
- D_{train} and D_{val} are respectively the training and validation dataset;
- $\mathcal{L}$ is the *loss function* measuring the performance of $\mathcal{N}_\gamma$, which is trained on D_{train} and validated on D_{val};
- k is the number of folds used in a k-fold cross validation procedure, aimed at providing a more robust estimation of the *generalization loss*, that is the loss on new unseen data;
- γ^* represents the optimal hyperparameters configuration providing the minimum value of $\mathcal{L}$ function;
- c_j is the j-th constraint that $\mathcal{N}_\gamma$ must satisfy to be deployable on the selected tiny device;
- n_c is the number of constraints.

Thus, in AutoTinyML the next hyperparameters configuration to evaluate is selected by considering both accuracy and deployability, according to 6.1.
In a nutshell, the AutoTinyML approach is based on SVM-CBO algorithm, which consists of two different phases. In the first phase, the goal is to estimate which is the portion of the search space associated to deployable hyperparameers configurations. This phase is named *feasibility determination* because the portion of the search space to estimate corresponds to the feasible region of the problem (6.1). In this study *deployability* coincides with *feasibility*.
In the second phase, a sequential model-based optimization approach is adopted to optimize the loss function, but only in the restricted estimated feasible region. As in SVM-CBO, an SVM classifier (Saunders et al., 1998) is used in the first phase to approximate the feasible region, but contrary to the implementation proposed in (Candelieri, 2019), here the RF model is used – instead of a GP – to approximate the objective function in the second phase. To differentiate, the new version of SVM-CBO proposed in this work was named SVM-CBO$_{RF}$.
However, before reporting the summarised Algorithm 3 some notation must be introduced.
Let $H_{1:m}$ denotes the set of hyperparameters configurations evaluated so far, that is $H_{1:m} =$

$\left\{\left(\gamma^{(i)}, \hat{\mathcal{L}}^{(i)}\right)\right\}_{i=1,\ldots,m}$, with $\gamma^{(i)} \in \Gamma$ and $\hat{\mathcal{L}}^{(i)}$ the associated validation losses (e.g., $\hat{\mathcal{L}}^{(i)}$ is computed on a hold-out validation set or via k-fold cross validation). Furthermore, let $\Delta_{1:m}$ denotes the set storing the actual deployability of the models in $H_{1:m}$, that is $\Delta_{1:m} = \left\{\left(\gamma^{(i)}, \delta^{(i)}\right)\right\}_{i=1,\ldots,m}$, with $\delta^{(i)} \in \{+1, -1\}$ a label for deployable/undeployable, respectively.

As previously mentioned, SVM-CBO deals with partially defined objective functions and in its original implementation it stores in the first set only the hyperparameters configurations resulting *computable* (i.e., training and validation are completed without any crash). Thus the result should be $H_{1:p}$ and $\Delta_{1:m}$, with $p \leq m$.

In this work, *deployability* and *computability* (aka *trainability*) are two completely different issues. Indeed, given a hyperparameter configuration, the validation loss of the associated model could be computable, but the resulting trained model could be undeployable on the tiny device. On the other hand, the uncomputability of the loss function occurs in case that the model cannot be trained using that hyperparameters configuration: in this case, there is no model to deploy. Thus, without any loss of generality, the two sets have been considered, H and Δ, having the same number of elements, m, and assign $\hat{\mathcal{L}}^{(i)} = \emptyset$ in case that the validation loss cannot be computed for the i-th hyperparameter configuration. So it is always possible to keep track of both computability and deployability of every model.

The last important point is related to acquisition functions, which are different in the two phases of SVM-CBO$_{RF}$. In phase 1, the next hyperparameters configuration to evaluate is given by the minimization of the following equation:

$$\gamma^* = \arg\min_{\gamma \in \Gamma} \left| \sum_{j=1}^{\nu} \alpha_j \delta^{(j)} k(\gamma^{(j)}, \gamma) \right| + \sum_{l=1}^{m} e^{-\frac{||\gamma^{(l)} - \gamma||^2}{2\ell^2}} \tag{6.2}$$

where the first addendum is the distance from the separation hypersurface defined by the SVM classifier in the search space (i.e., separation hyperplane in the induced Feature Space), while the second addendum is a measure of *coverage* of the search space according to the spatial distribution of the m hyperparameters configurations evaluated so far. Moreover, ν and α_j are, respectively, the number of support vectors and the Lagrangian coefficients in the SVM classifier, $k(\cdot,\cdot)$ is the kernel function used by the SVM classifier – a Radial Basis Function kernel has been adopted to catch and model non-linearities. Then, ℓ is a parameter to model the coverage effect of each evaluated hyperparameters configuration: the larger ℓ the wider is the coverage associated to a hyperparameters configuration.

Accordingly to (6.2), in phase 1 a configuration of hyperparameters is selected that is closer to the SVM separation hypersurface and far from the previously evaluated configurations.

The output of the phase 1 is an estimate of the feasible region associated to hyperparameters configurations which should generate deployable models.

Let $\tilde{\Omega} \in \Gamma$ denote the estimate of the feasible region, such that:

$$\tilde{\Omega} = \left\{ \gamma \in \Gamma : \sum_{j=1}^{\nu} \alpha_j \delta^{(j)} k(\gamma^{(j)}, \gamma) < 0 \right\} \tag{6.3}$$

In phase 2, the next hyperparameters configuration to evaluate is proposed considering as follows:

$$\gamma^* = \arg\min_{\gamma \in \tilde{\Omega}} \mu(\gamma) - \sqrt{\beta}\sigma(\gamma) \tag{6.4}$$

The above formula represents the well-known *Lower Confidence Bound* (LCB) (Srinivas et al., 2010). Where $\mu(\gamma)$ and $\sigma(\gamma)$ are, respectively, the mean prediction and the associated uncertainty provided by the RF probabilistic surrogate model, and β is a parameter to manage the exploration-exploitation trade-off, for which suitable scheduling with convergence proof is provided in (Srinivas et al., 2012). The reason for its adoption in the SVM-CBO$_{RF}$ approach is driven by the fact that this acquisition function outperforms the most performing and widely used acquisition function such Expected Improvement (EI) (Srinivas et al., 2012).
Besides, it is important to remark that while (6.2) is minimized on the overall search space Γ, (6.4) is minimized only on the estimate $\tilde{\Omega}$ of the feasible region.

Algorithm 3 SVM-CBO$_{RF}$

input : m_0 - number of initial configurations;
M_1 - configurations to evaluate in phase 1;
M_2 - configurations to evaluate in phase 2.

output: $(\gamma^+, \hat{\mathcal{L}}^+) \in H_{1:M} : \hat{\mathcal{L}}^+ = \min_{i=1,\dots,M} \{\hat{\mathcal{L}}^{(i)}\}$ with $M = m_0 + M_1 + M_2$.

initialization:
generate $\gamma^{(1)}, \dots, \gamma^{(m_0)}$
compute $\hat{\mathcal{L}}^{(1)}, \dots, \hat{\mathcal{L}}^{(m_0)}$
evaluate $\delta^{(1)}, \dots, \delta^{(m_0)}$

—— *Phase 1* ——

while $m < M_1$ **do**
  train an SVM classifier on $\Delta_{1:m}$
  select $\gamma^{(m+1)}$ according to (6.2)
  train $\mathcal{N}_{\gamma^{(m+1)}}$ and compute $\hat{\mathcal{L}}^{(m+1)}$
  translate and deploy $\mathcal{N}_{\gamma^{(m+1)}}$ to obtain $\delta^{(m+1)}$
  update $H_{1:m+1} \leftarrow H_{1:m} \cup (\gamma^{(m+1)}, \hat{\mathcal{L}}^{(m+1)})$
  update $\Delta_{1:m+1} \leftarrow \Delta_{1:m} \cup (\gamma^{(m+1)}, \delta^{(m+1)})$
  $m \leftarrow m + 1$
end

—— *Phase 2* ——

while $m < M_2$ **do**
  train a RF regressor on $H_{1:m}$
  select $\gamma^{(m+1)}$ according to (6.4)
  train $\mathcal{N}_{\gamma^{(m+1)}}$ and compute $\hat{\mathcal{L}}^{(m+1)}$
  translate and deploy $\mathcal{N}_{\gamma^{(m+1)}}$ to obtain $\delta^{(m+1)}$
  update $H_{1:m+1} \leftarrow H_{1:m} \cup (\gamma^{(m+1)}, \hat{\mathcal{L}}^{(m+1)})$
  update $\Delta_{1:m+1} \leftarrow \Delta_{1:m} \cup (\gamma^{(m+1)}, \delta^{(m+1)})$
  if $\delta^{(m+1)} = -1$ **then**
    re-train the SVM classifier on $\Delta_{1:m+1}$
  end
  $m \leftarrow m + 1$
end

6.2.2 Integration with STM32CubeMX

STMicroelectronics developed a software named *STM32CubeMX*[6] which is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code for the Arm® Cortex®-M core (adopted in this study) or a partial Linux® Device Tree for Arm® Cortex®-A core, through a step-by-step process.

Moreover, this software allows the final user to add different extension packages to enable several functionalities that might be available for specific MCUs with particular hardware. In addition with this software it is possible to interact with a virtualized MCU to speed up the whole development process of new applications without having a real MCU. Indeed, inside this software there are almost all MCUs produced by STMicroelectronics, which can be applied from health-care to industry monitoring applications.

Also, to support TinyML community, within STM32CubeMX is developed an extension package named *STM32Cube.AI*[7], that provides useful porting of DNN architectures on MCUs.

This package is able to automatically convert pre-trained DNN into optimized C-language code, in terms of computation and memory usage, to run efficiently on any STM32 MCUs. One of the most important features of this extension package is that it provides the optimization of a pre-trained DNN using several DNN libraries currently available in DL community, such as Keras[8], Tensorflow[9], TensorflowLite[10] and ONNX[11].

With this extension it is also possible, through a Graphic Unit Interface (GUI) or Command Line Interface (CLI), to analyze and validate the DNN architectures and observe the proper amount of computational resources and memory required for storing and running the original (reference) and the C-optimized networks on MCUs.

The original (reference) model can be compressed with three compression factors ($\times 1$, $\times 4$, $\times 8$) to reduce the size of weight/bias parameters of the model.

With the validation of the network the STM32Cube.AI provides several information in terms of performance of the DNN models. For each validation are provided the following performance measures: Accuracy, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

Also, in case of classification task, to compare the original (reference) DNN model with the C-optimized one, the tool uses a so-called *X-CROSS Accuracy*, which is able to measure how far are the predictions of C-optimized model in respect to the original (reference) one. To compute this particular Accuracy measure between models, the tool creates a Confusion Matrix, used in classification task, but instead of using the ground truth values and the predicted values, it uses

[6] https://www.st.com/en/development-tools/stm32cubemx.html
[7] https://www.st.com/en/embedded-software/x-cube-ai.html
[8] https://keras.io/
[9] https://www.tensorflow.org/versions/r2.0/api_docs/python/tf
[10] https://www.tensorflow.org/lite
[11] https://onnx.ai/supported-tools.html

as the ground truth the predictions of the original (reference) model against the predictions of the C-optimized model. With this measure it is possible to quantify the loss of accuracy reached using the C-optimized model in respect to the original (reference) model.
Based on the collaboration with STMicroelectronics, a good C-optimized DNN model must have a X-CROSS Accuracy greater or equal to 95%, which is a good trade-off between the loss in accuracy and the compression factor provided by STM32Cube.AI.

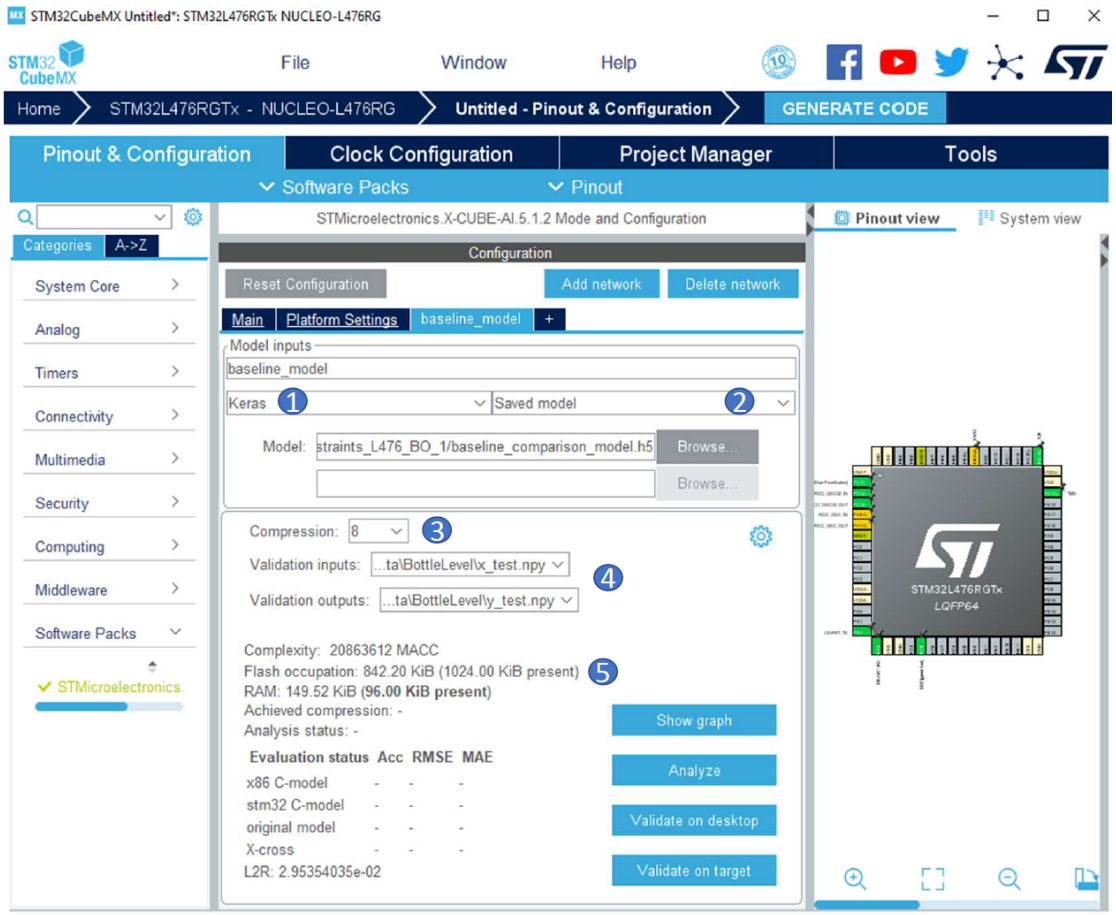


FIGURE 6.2: An example of STM32CubeMX GUI window with one of the DNN baseline computed to check hardware constraints

Figure 6.2 shows an example of the GUI window of STM32CubeMX adopting the STMicroelectronics X-CUBE-AI extension tool (STM32Cube.AI). The blue point with number ***1***, in the Figure, identifies the Deep Learning framework adopted by the Deep Neural Network that has to be optimized in C-language, while the point with ***2*** indicates whether the model to be loaded is a complete neural network file (architecture and network weights) or just the network topology (architecture).
The number ***3*** identifies the compression factor that must be applied to the network (in specific to the weights of the network) during the translation of the loaded model into its C-optimized

version that can be deployed on the MCU. The number ***4*** indicates the validation input and output (aka ground truth of the test set) with which the C-optimized network is tested. With the number ***5*** are reported the complexity of the model through the Multiply ACCumulated operations (MACC), and the hardware resources used by the C-optimized version model on a virtual MCU. Always in Figure 6.2 is reported the example of a baseline model of one of the real use cases that will be introduced into next Chapter.

In order to implement the AutoTinyML framework, it was necessary to automate this network translation, compression and validation process using the CLI version of the STMicroelectronics X-CUBE-AI tool (STM32Cube.AI).

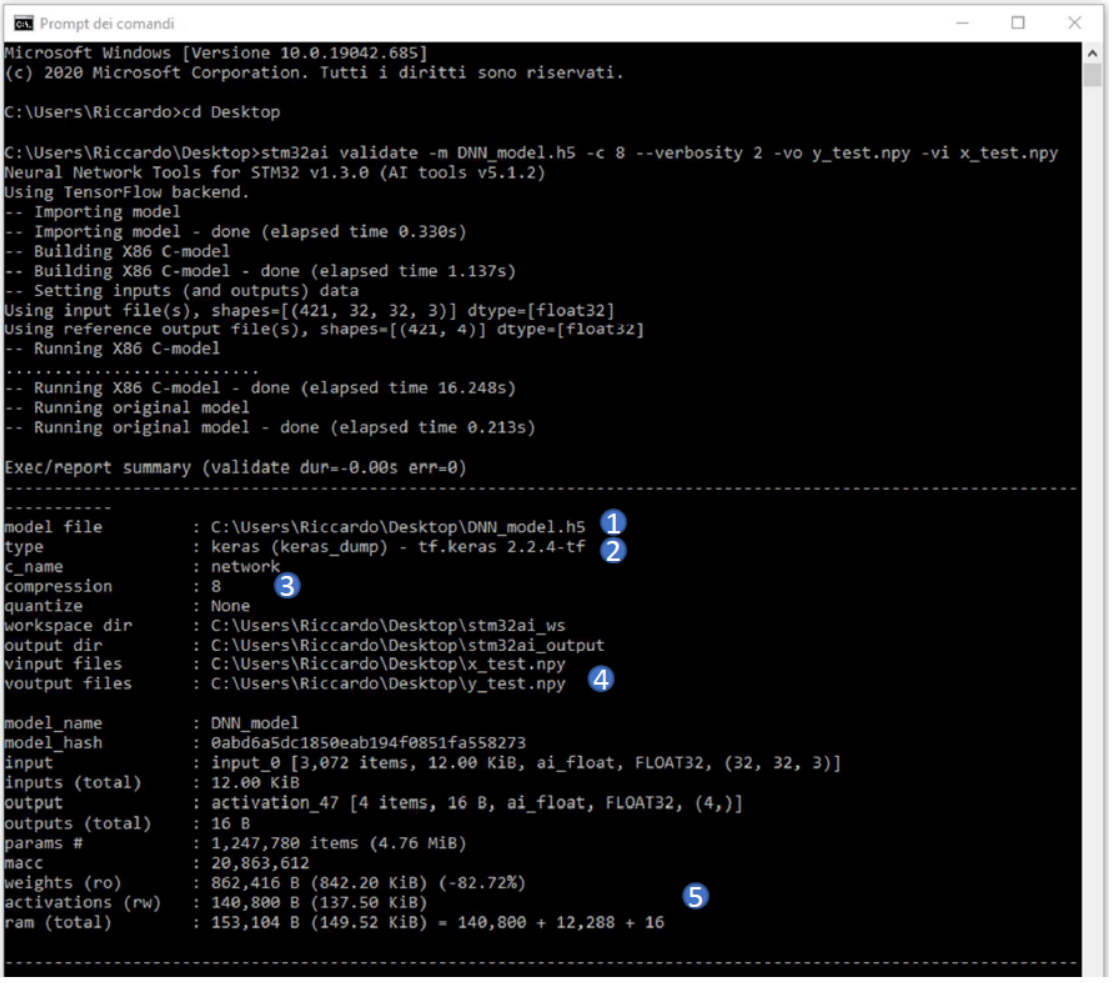


FIGURE 6.3: An example of STM32Cube.AI CLI window with one of the DNN baseline computed to check the performance metrics (e.g., Accuracy and RMSE) and hardware constraints (e.g., RAM and ROM) (Part 1 of 2)

In Figure 6.3 is reported the result of the validation of the same neural network seen before. In fact, the numbers reported (from ***1*** to ***5***) represent the same information previously reported by the GUI window. Moreover, in the Figure 6.4 is reported the rest of the command started in Figure 6.3, from where it is possible to observe the performances of the original (reference) model and of the C-optimized model. In particular the points with number ***6*** and ***7*** report the accuracy,

RMSE, MAE and the Confusion Matrix for C-optimized model and original (reference) model respectively. Instead the number ***8*** represents the performance metrics calculated on X-CROSS accuracy, previously introduced, to quantify the discrepancy between the original (reference) model and the C-optimized one, while with number ***9*** a summary of the all metrics available is reported. When the command from the CLI is completed, a report file is generated that contains exactly what is shown in the output of the CLI window from the last two Figures 6.3 and 6.4.

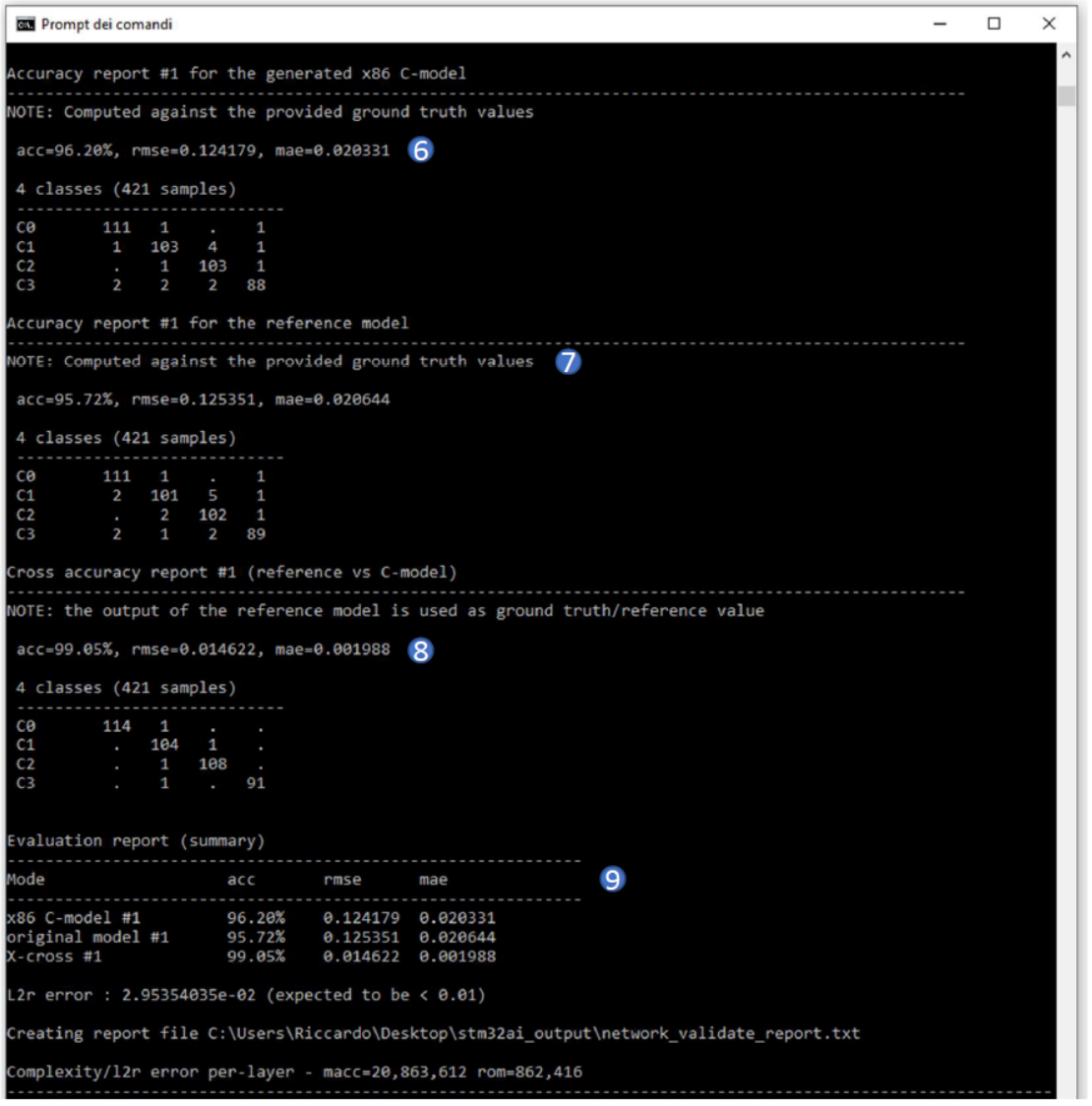


FIGURE 6.4: An example of STM32Cube.AI CLI window with one of the DNN baseline computed to check the performance metrics (e.g., Accuracy and RMSE) and hardware constraints (e.g., RAM and ROM) (Part 2 of 2)

In particular the entire AutoTinyML framework workflow is described in Figure 6.5. The green rectangle contains the three main modules that implement the search process of the optimal DNN configuration on the given real-use cases that will be presented in the next Chapter. The *Keras* module has as the main goal to train DNN of a given hyperparameter configuration exploiting the DL Keras library, the *STM32Cube.AI* module is used to translate the trained model,

from the previous module, to check the performance metric and deployability constraints for an MCU, while the *SVM-CBO*$_{RF}$ module[12] is the optimizer adopted to find out the optimal DNN configuration. Due to the limited computational capacity, it was considered to evaluate the performance and deployability constraints with the validation technique *hold-one-out*, i.e. testing the network using data that are not present within the dataset used for training. Moreover, at the end of the optimization process, the deployable DNNs found are then rebuilt and trained with the dataset composed by training and validation set, and tested with the test dataset never before used during the process of finding the optimal configuration.

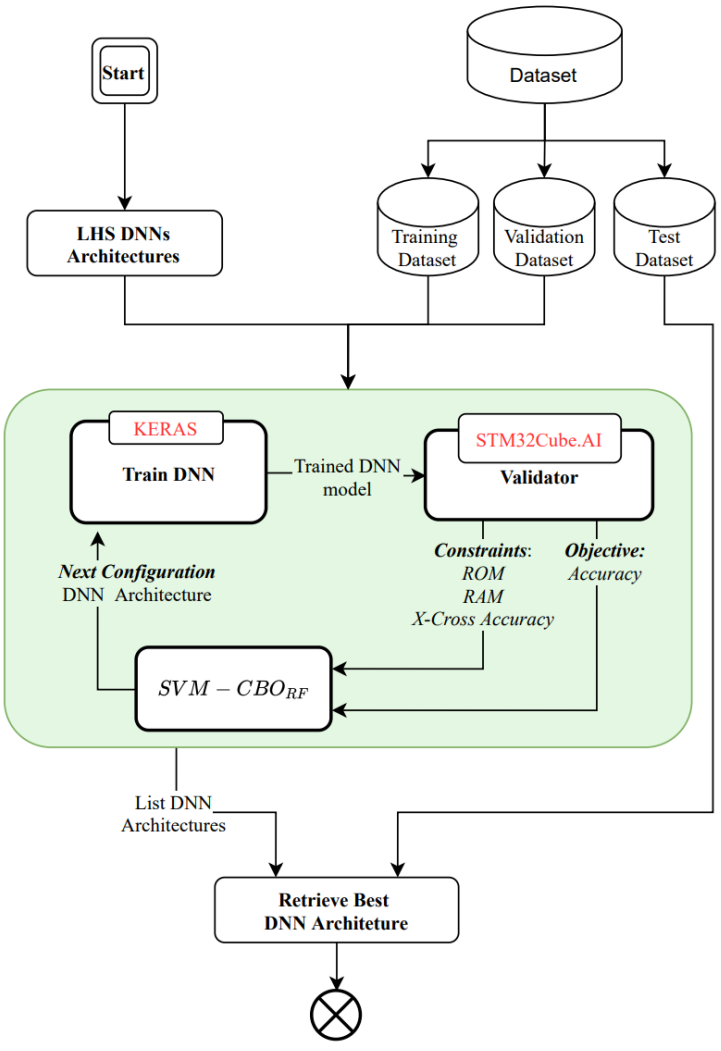


FIGURE 6.5: Figure reports the AutoTinyML workflow in detail. The green rectangle identifies the optimizer proposed in this thesis, which is composed by the constrained optimizer (SVM-CBO$_{RF}$), the Deep Learning framework adopted (Keras), and the translator system for obtained C-optimized Deep Neural Networks provided by STMicroelectronics

In addition, in Figure 6.6 two reports show the compilation and deploy of the C-project, representing the deployable and optimized DNN, and the validation of the network directly on the STM32 device using a test dataset of one of the real use cases provided.

[12] https://github.com/ricky151192/SVMCBO

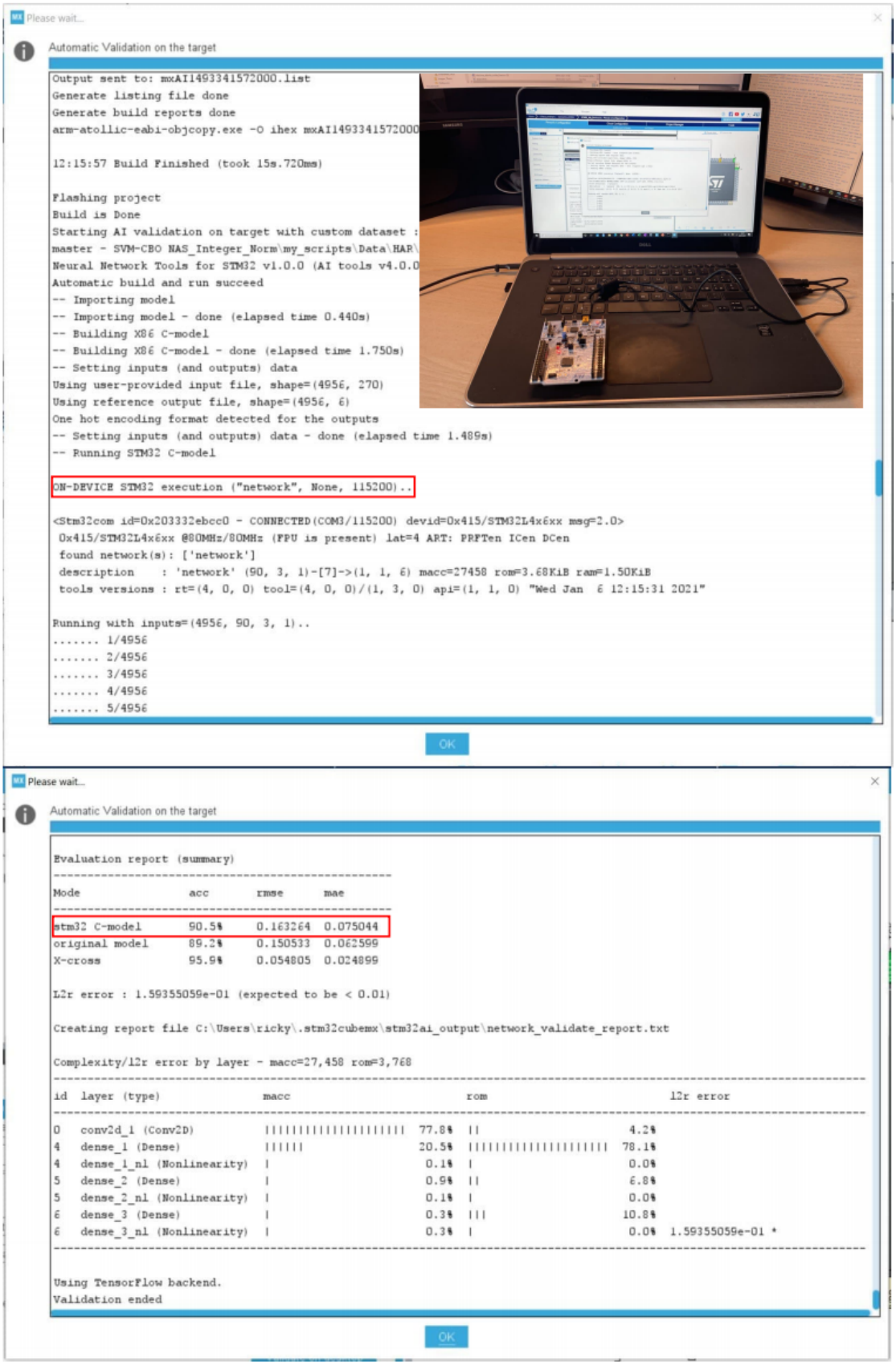


FIGURE 6.6: Figure shows two reports, one is the compilation and deploy process on top figure, while the other is the validation process on a given device

Chapter 7

Experiments

In order to test the optimization approach previously introduced in this chapter an exhaustive validation is reported here, starting with common test problems adopted by the Constrained Global Optimization community, and passing then to the validation of the proposed approach of AutoTinyML on two real-life use cases.
All the experiments have been performed on a machine equipped with: Intel i9 (4.0Ghz) processor with 8 cores, 64Gb of RAM and NVIDIA GeForce RTX 2080 SUPER 8Gb. Moreover, for the AutoTinyML experimentation has been adopted Keras 2.2.4 with Tensorflow 2.1.0 as Deep Learning framework, which are the current supported versions by the software STM32CubeMX with STM32Cube.AI extension provided by STMicroelectronics.

7.1 Test problems

The first step to validate the approach proposed in this thesis was to optimize test functions widely adopted by the community in order to test constrained optimization approaches.

7.1.1 Simple 2D test functions

The functions considered were: Rosenbrock constrained to a line and a cubic (Simionescu and Beale, 2002), Mishra's Bird constrained (Mishra, 2006), a test function proposed in (Gramacy et al., 2016) (from now on defined as Gramacy), Michelowicz (Al-Roomi, 2015) and Alpine N.2 (Jamil and Yang, 2013) with the proposed not linear constraints.
In addition, the approach has been tested on the Branin (Picheny et al., 2013) (rescaled) function bound in two disconnected ellipses, in order to show SVM-CBO's ability to optimise two or

more eligible disconnected regions. The appendix A reports the complete set of the test functions and their constraints adopted to validate the constrained optimization approach.

7.1.2 Test function generator: Emmental-type GKLS

Emmental-type GKLS (Sergeyev et al., 2017) is a software for generating multi-dimensional functions that can be continuously differentiated multi-extremal functions with non-linear constraints. The overall minimizer of these constrained test functions is always at the limit of the feasible region. The dimensionality and complexity of the test functions can be easily controlled by setting a few generator parameters. The experimental setting defined for the experiments with Emmental-type GKLS was inspired by the one used in (Sergeyev et al., 2018): test functions of growing dimensionality, from 2D to 5D, have been considered, for 2 different classes of complexity each. In this exhaustive validation are considered a total of 80 different test functions, which are generated based on the 10 initial different functions with several dimensionality and class complexity. The parameters to control the generation process of the test functions are setted as suggested in (Sergeyev et al., 2018), setting the number of local minima at 30, the number of constraints at 20, 2 and 10, respectively for the possible three types considered by Emmental-type GKLS, with 5 active constraints at the global optimum. To show the complexity of the generated constrained test function in Figure 7.1 is reported an example proposed in (Sergeyev et al., 2017), where it is possible to notice the very complex unfeasible and feasible region drawn by the generator. More recently, another flexible extension of GKLS (Sergeyev et al., 2017) for constrained problems has been proposed in (Gergel et al., 2019), namely GCGen (Global Constrained optimization problem Generator). Although it can be considered more flexible than Emmental-type GKLS, it has been discovered only after the experimental campaign performed by using Emmental-type GKLS. Moreover, in the last period of the PhD studies the effort was focused on addressing the real-life application, therefore, unfortunately, GCGen could not be considered.

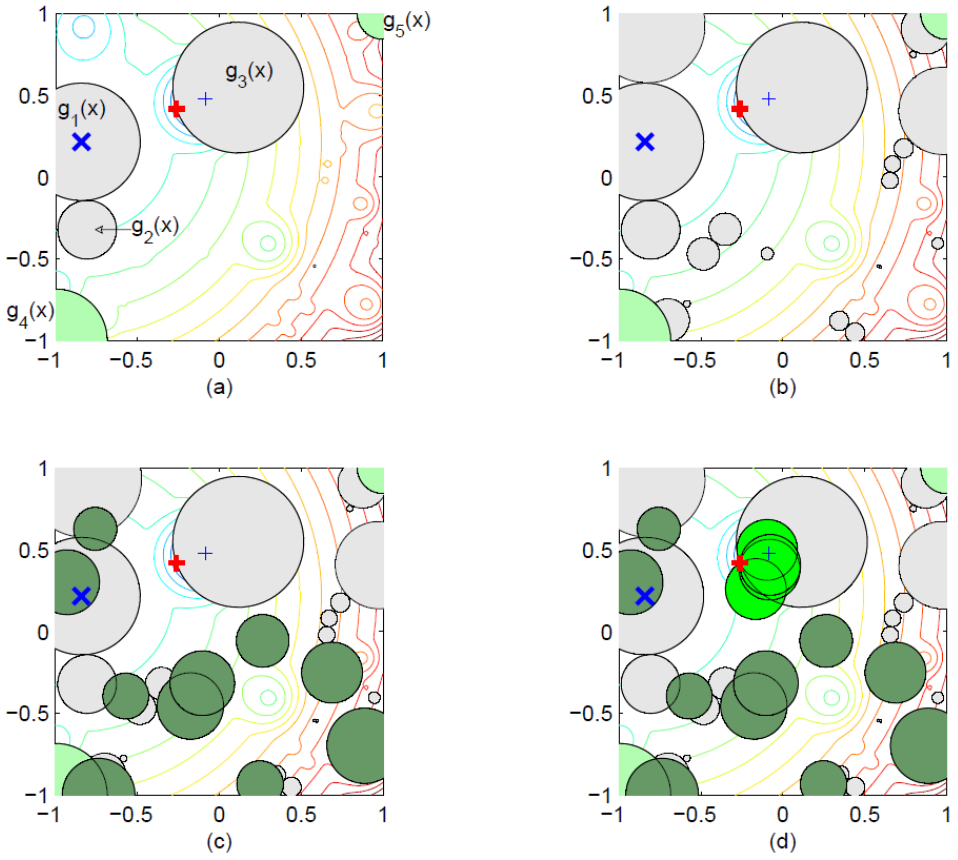


FIGURE 7.1: The feasible regions and the level curves of the constrained Emmental-type test functions with increasing complexity with the following numbers of constraints (a) $p1 = 3$, $p2 = 2$, $p3 = 0$, $Nactive = 1$; (b) $p1 = 20$, $p2 = 2$, $p3 = 0$, $Nactive = 1$; (c) $p1 = 20$, $p2 = 2$, $p3 = 10$, $Nactive = 1$; (d) $p1 = 20$, $p2 = 2$, $p3 = 10$, $Nactive = 5$. The global minimizer of the constrained problem is indicated by red bold $+$, and the global minimizer of the corresponding box-constrained D-type GKLS problem is indicated by blue thin $+$ (Sergeyev et al., 2017)

7.2 AutoTinyML

In this section are introduced the AutoTiny ML Tasks considered to validate the performances of the optimization approach proposed as a major contribution of this PhD thesis. The main goal of the proposed approach is not to create the optimal CNN model from scratch but to start from a baseline/draft network and apply the proposed constrained optimization framework in order to find the best DNN in terms of accuracy considering the hardware limitations of MCUs as constraints. In this way, the proposed optimization framework can adapt the CNN baseline model, currently applied to a given task, by finding a variant model that increases its accuracy while taking into account the stringent hardware constraints of the target MCU.

Initially, the tiny hardware devices adopted in the experiments will be presented, and then the classification benchmark tasks will be illustrated, analyzing their DNN baseline models and

defining the search space for the optimization for each task. Finally, the current deployability of each baseline model will be investigated to show that in most combinations between the compression level, provided by STM32Cube.AI, and the selected MCU it is not actually possible to run a DNN on these tiny devices.

7.2.1 Tiny Devices

The proposed SVM-CBO$_{RF}$ for AutoTinyML has been validated on two benchmark classification tasks and two different tiny devices provided by STMicroelectronics.
The name and characteristics of the two MCUs used in the experiments are:

- **STM32L476RGT6**[1] (from now on as *Big Board*) with ARM Cortex M4 80MHz, 1 Mbytes of ROM and 128 KBytes of RAM
- **STM32F303K8T6**[2] (from now on as *Tiny Board*) with ARM Cortex M4 72MHz, 64 KBytes of ROM and 16 KBytes of RAM.

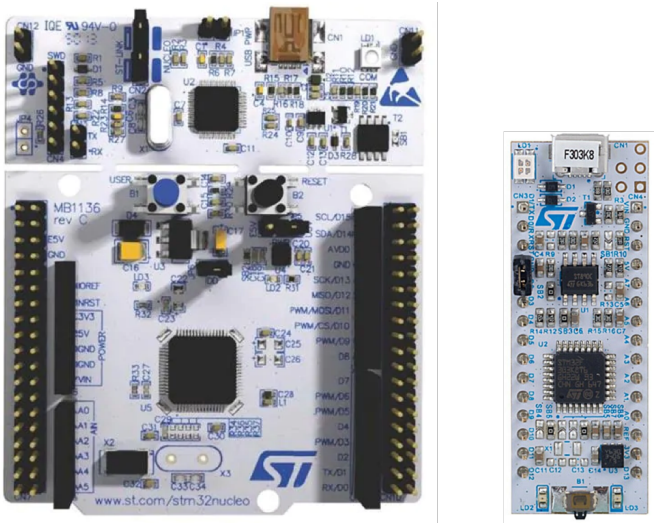


FIGURE 7.2: The STMicroelectronics MCUs adopted in this study. On the left the *Big Board* and on the right the *Tiny board*

[1] https://www.st.com/en/microcontrollers-microprocessors/stm32l476rg.html
[2] https://www.st.com/en/microcontrollers-microprocessors/stm32f303k8.html

7.2.2 Benchmark Classification Tasks

Human Activity Recognition

The benchmark classification task considered is the classification of human activities from sensors data. The data is generated by a 3D accelerometer used to sample accelerations on the *x*-, *y*-, and *z*-axis, at a given frequency. All the recordings are stored in a time series dataset, described in (Casale et al., 2012), freely accessible on the UCI Repository and named *User Identification From Walking Activity Data Set*[3]. A baseline Convolutional Neural Network (CNN) for Human Activity Recognition (HAR) is available on GitHub[4]: in Table 7.1 the architecture of the CNN chosen from this repository is reported.

TABLE 7.1: Table reports the baseline CNN model architecture for Human Activity Recognition task

Layer	***Type Layer***	***Parameter***	***Value***
1	Input Layer	Units	(90,3)
2	Convolutional 2D	Filters	128
		Kernel Size	2
		Activation	ReLU
3	Max Pooling	Window	2
4	Dropout	Rate	0.2
5	Dense	Units	128
		Activation	ReLU
6	Dense	Units	128
		Activation	ReLU
7	Output Layer	Units	6
		Activation	Softmax

This baseline CNN is obtained by using the *Adam* as learning algorithm, using default values for its hyperparameters (learning rate $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$) and number of batch and epochs for training both set to 10.

The file (format *'.h5'*) size of this baseline CNN is 8.49 MBytes. However, when the C-optimized network is created through *STM32Cube.AI* (CLI), its size decreases to just 2.82 MBytes. Despite this reduction in size, the C-optimized CNN still does not satisfy the hardware requirements in terms of ROM and RAM.

The overall dataset (i.e., 19512 instances) has been divided into a training set (i.e., 15610 instances) and a validation set (i.e., 3902). The accuracy on a further test set (i.e., 4596 instances) has been computed and considered as the baseline performance value (i.e., 92.09%) to compare

[3] `http://archive.ics.uci.edu/ml/datasets/User+Identification+From+Walking+Activity`

[4] `https://github.com/Shahnawax/HAR-CNN-Keras`

with performances obtained through SVM-CBO$_{RF}$.

Image Recognition

The second classification benchmark task considered is the classification of level of chloride sterile liquid in bottles through image recognition. The dataset is made by a collaboration between STMicroelectronics and Sesovera.ai[5] which published the dataset at (Pau et al., 2020). The main goal of this dataset is to create a benchmark for classification task for image recognition runnable on MCU with limited hardware resources. The dataset is composed with images pixel resolution of 3456 $\times$ 3456 reduced to 32 $\times$32 to facilitate image saving and processing the classification inference by neural network on a limited MCU.

The dataset contains an overall of 4217 images, equally splitted in 4 different classes based on the level of liquid contained into the bottle in the image: *empty bottles*, *50%*, *80%* and *100%* levels (e.g. Figure 7.3).

FIGURE 7.3: Example of images data capture for each Image Recognition task (Pau et al., 2020)

The baseline CNN model was provided by STMicroelectronics and it is reported in Table 7.2. This baseline CNN is obtained by using the *Adam* learning algorithm, using default values for its hyperparameters (learning rate $\alpha = 0.01$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$) and trained with 200 epochs with an early stopping strategy of 20 epochs and the batch size equals to 32. The file (format *'.h5'*) size of this baseline CNN is 14.30 MBytes. However, when the C-optimized network is created through *STM32Cube.AI* (CLI), its size decreases to just 4.87 MBytes. Despite this significant reduction in size, the C-optimized CNN still does not satisfy the hardware requirements in terms of ROM and RAM.

The overall dataset (i.e., 4217 instances) has been splitted into training set (i.e., 3036 instances) and a validation set (i.e., 760 instances). Instead the accuracy on a further test set (i.e., 421 instances) has been computed and considered as the baseline performance value (i.e., 95.72%)

[5]http://www.sesovera.ai/

to compare with performances obtained through SVM-CBO$_{RF}$.

TABLE 7.2: Table reports the baseline CNN model architecture for Image Recognition task

Layer	***Type Layer***	***Parameter***	***Value***
1	Input Layer	Units	(32,3)
2	Convolutional 2D	Filters	32
		Kernel Size	3
		Activation	ReLU
3	Convolutional 2D	Filters	32
		Kernel Size	3
		Activation	ReLU
4	Max Pooling	Window	2
5	Dropout	Rate	0.25
6	Convolutional 2D	Filters	64
		Kernel Size	3
		Activation	ReLU
7	Convolutional 2D	Filters	64
		Kernel Size	3
		Activation	ReLU
8	Max Pooling	Window	2
9	Dropout	Rate	0.25
10	Dense	Units	512
		Activation	ReLU
11	Dropout	Rate	0.25
12	Output Layer	Units	4
		Activation	Softmax

Experiments Setup

Human Activity Recognition: *Search Space, Compression & Deployability*

First, we introduce the CNN's hyperparameters to optimize. Depending on the baseline CNN architecture reported in Table 7.1, we make some relevant considerations to define the search space.

The CNN's size is a relevant factor for solving the problem of constrained HPO, in 6.1, and it is strictly connected with the size of the CNN's layers. We decided to limit to 256 the maximum number of units for each layer, which should be enough to obtain accurate but not too large models. However, considering the differences in the hardware resources between the two selected MCUs, we set different values for the minimum number of units in each layer: 16 for the *Big Board* and 8 for the *Tiny Board*. The main reason is that we prefer to obtain smaller models for

the *Tiny Board* while for the *Big Board* we can reduce the range of this hyperparameters, thanks to its larger hardware resources.
We decided to maintain the number of layers of the baseline CNN architecture, thus the number of layers is not a hyperparameter to be optimized. We have also decided a further simplification for the search space: layers of the same type have the same activation function and number of units. This allowed to further reduce the search space while leading to accurate CNN models, anyway. A possible extension of the approach would be to discard this assumption in the case that the optimization could not improve along trials. Consequently the search space could be widened, 'dynamically', to increase the set of possible neural architectures. Moreover, the last dense layer is not a hyperparameter to optimize, because it is defined according to the classification task. Indeed, the number of units of this layer is the number of classes, with the *Softmax* activation function providing a probability distribution of the classes for each input to the CNN. Finally, the 8 hyperparameters to optimize are summarized in Table 7.3 with their associated domains.

TABLE 7.3: Table reports the search space of HPO problem for Human Activity Recognition task

Hyperparameter		***Values***	
		Big Board	***Tiny Board***
Convolutional 2D	***Filters***	[16, 32, 64, 128, 256]	[8, 16, 32, 64, 128, 256]
	Activation	[ReLU, Sigmoid, Selu, Tanh]	
	Kernel Size	[1, 2]	
MaxPooling	***Windows Pooling***	[1, 2]	
Dropout		[0.1, 0.2, 0.3]	
Dense	***Units***	[16, 32, 64, 128, 256]	[8, 16, 32, 64, 128, 256]
	Activation	[ReLU, Sigmoid, Selu, Tanh]	
Optimizer		[Adam, SGD, RMSProp, Adadelta, Adamax]	

Different experiments have been performed according to the different compression factors provided by STM32Cube.AI when converting the CNN model into the C-optimized one to deploy on the MCUs. Specifically, the three compression factors available are: $\times 1$, $\times 4$, and $\times 8$.
Considering the deployability constraints related to hardware resources of the *Big Board* (i.e., 1 MBytes of ROM and 128 KBytes of RAM) a compression factor $\times 4$ makes the C-optimized version of the baseline CNN deployable on this MCU. Thus, we have decided to limit experiments for the *Big Board* to just compression factors $\times 1$ and $\times 4$. On the contrary, the deployablity constraints related to the hardware resources of the *Tiny Board* (i.e., 64 KBytes of ROM and 16 KBytes of RAM) do not allow, for any compression factor, to generate a C-optimized version of the baseline CNN model deployable on this MCU.
In this case we limited the experiments for *Tiny Board* to just compression factors $\times 4$ and $\times 8$.

TABLE 7.4: The deployability of the baseline model reported in table 7.1 on different adopted compression factor

Compression Factor		**×1**	**×4**	**×8**
Accuracy		92.09%	92.07%	91.32%
RAM (KBytes)		24.58	24.58	24.58
ROM (KBytes)		2955.80	794.14	375.45
X-CROSS Accuracy		100.00%	99.98%	98.04%
Deployable	***Big Board***	NO	YES	YES
	Tiny Board	NO	NO	NO

Table 7.4 summarizes the hardware resources required by the C-optimized versions of the baseline CNN model depending on the different compression factors.

Image Recognition: *Search Space, Compression & Deployability*
As seen for the previous task (HAR), also for the Image Recognition task, considerations must be made for the best possible definition of the search space of the problem to be optimized.
First of all, it must be underlined that unlike the HAR task, for this new task the baseline CNN model (provided by the STMicroelectronics company) is not deployable for any compression factor adopted. In fact, from Table 7.5 it can be seen that for both MCUs tested, the baseline model does not allow the hardware constraints to be satisfied, especially due to the large usage of RAM.

TABLE 7.5: The deployability of the baseline model reported in table 7.2 on different adopted compression factor

Compression Factor		**×1**	**×4**	**×8**
Accuracy		95.72%	95.72%	96.20%
RAM (KBytes)		140.80	140.80	140.80
ROM (KBytes)		4991.12	1453.20	862.42
X-CROSS Accuracy		100.00%	100.00%	99.05%
Deployable	***Big Board***	NO	NO	NO
	Tiny Board	NO	NO	NO

In fact the compression factor (e.g. $\times 4$ and $\times 8$) only affects the use of the ROM hardware resource constraint, since the compression impacts on the numerical representation of the values representing the weights of the dense layers present in the trained CNN. Being a more complex task in terms of use of hardware resources like RAM and ROM it was decided to diversify the values of the units according to the type of the layer (e.g. Convolutional and Dense). In fact, for

the convolutional layers a number of units was set up to a maximum of 64 per layer and with a smaller minimum of 6 units compared to the previous HAR task. In addition, the possible kernel size parameter of convolutional layer and pooling windows are increased to a maximum of 3, while the maximum value of the dropout layer is also increased to 0.4 in respect to the previous HAR task. As far as the dense layer is concerned, a minimum of 6 and a maximum of 512 units per layer have been set, since the compression factor has a great impact on the size allocable on the ROM in terms of KBytes for this type of layer.

TABLE 7.6: Table reports the search space of HPO problem for Image Recognition task

Hyperparameter		***Values***
Convolutional 2D	***Filters***	[6, 8, 10, 12, 16, 32, 64]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
	Kernel Size	[1, 2, 3]
MaxPooling	***Windows Pooling***	[1, 2, 3]
Dropout		[0.1, 0.2, 0.3, 0.4]
Dense	***Units***	[6, 8, 10, 12, 16, 32, 64, 128, 256, 512]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
Optimizer		[Adam, SGD, RMSprop, Adagrad, Adadelta]

Due to the very limited hardware resources for the Tiny Board (16 KBytes RAM - 64 Bytes ROM) and RAM's large use by the baseline CNN model architecture, it was necessary to insert an additional hyperparameter within the search space of the HPO problem. In the modified search space in the Table 7.7, this new hyperparameter allows to eliminate at most one convolutional layer, since this particular layer significantly increases the use of the RAM resource to be able to run the CNN model on the Tiny Board.

TABLE 7.7: Table reports the modified search space of HPO problem for Image Recognition task including an hyperparameter which modifies the architecture of CNN

Hyperparameter		***Values***
Convolutional 2D	***Filters***	[6, 8, 10, 12, 16, 32, 64]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
	Kernel Size	[1, 2, 3]
MaxPooling	***Windows Pooling***	[1, 2, 3]
Dropout		[0.1, 0.2, 0.3, 0.4]
Dense	***Units***	[6, 8, 10, 12, 16, 32, 64, 128, 256, 512]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
Optimizer		[Adam, SGD, RMSprop, Adagrad, Adadelta]
Removal Layer		[1, 2, 3, 4]

SVM-CBO$_{RF}$ settings.
We have performed 15 experiments for each pair <*MCU - compression factor*>: 5 different sets of initial hyperparameters configurations times 3 different initializations of network's weights for each set.
We have fixed a maximum of 100 hyperparameters configurations to evaluate, for each experiment, organized as follows:

- 10 initial hyperparameters configurations generated via Latin Hypercube Sampling to initialize SVM-CBO$_{RF}$;
- 60 hyperparameters configurations for the phase 1 of SVM-CBO$_{RF}$;
- 30 hyperparameters configurations for the phase 2 of SVM-CBO$_{RF}$.

The maximum number of function evaluations has been defined according to STMicroelectronics' requirements in the case of a real-life application. Depending on the time approximately required to run a single optimization process - over the computational setting adopted - the most suitable number resulted in 100 different configurations to be tested for each optimization run.
Due to the complexity of the Image Recognition task, the fixed budget for initial hyperparameters configurations procedure has been set to 20 initial configurations, because it was difficult to find out deployable configurations in the random initialization of the optimization process.
The loss function used to train the CNN models is the *Categorical-Cross Entropy*, which is correlated to classification error, and consequently to accuracy. The objective function optimized by SVM-CBO$_{RF}$ is the classification error computed on the validation set (i.e., hold-out validation), while the deployability constraints are:

- **RAM** $\leq$ 128 KBytes for Big Board and $\leq$ 16 KBytes for Tiny Board
- **ROM** $\leq$ 1024 KBytes for Big Board and $\leq$ 60 KBytes for Tiny Board
- **X-CROSS Accuracy** $\geq$ 95% for both MCUs

Chapter 8

Results

This Chapter will present the results obtained by the proposed approach on a suite of artificial test problems, provided by the Constrained Global Optimization community, and two real-life applications related to the classification task through Deep Neural Network models. Initially the results will be presented on the suite of test functions and statistical tests will be carried out to prove the actual significance of the results obtained by the proposed optimization approach. Then, the performance of the application of the proposed approach will be evaluated to two benchmark tasks for classification, and considerations will be made on the performance obtained from the optimal models found out by the optimization processes.

8.1 Test Problems

In order to evaluate the performances of the optimizer SVM-CBO$_{RF}$ adopted in the proposed AutoTinyML approach in the previous Chapter, the performances of SVM-CBO in both variants (GP, RF) are compared against the most famous optimizer in Sequential Model-based Optimization which is SMAC (Hutter et al., 2011). In order to compare the performance of the optimizers, the Mann-Whitney test was adopted from which the p-values for each pair of optimizers were obtained and analyzed.
In particular, for the tests adopted between SMAC with each variant of SVM-CBO, the unpaired Mann-Whitney test was considered since the initial points evaluated are chosen differently between the optimizers. Instead for the test between the two variants of SVM-CBO, the paired Mann-Whitney test was used as the optimizers' initial points are based on the same sampling technique.
The Table 8.1 reports the mean and deviation of best seen values of 30 independent runs on the 2D test functions for each adopted optimizer. It is possible to observe in terms of best seen value that in more than half of the functions SVM-CBO and its variant SVM-CBO$_{RF}$ outperform the

TABLE 8.1: Results of the six test functions on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value$<$0.001, "**" for p-value$<$0.01, "*" p-value$<$0.05, and "-" whether p-value$\geq$0.05

Test Function	***Mean ($\pm$ Std)***			***Significance Level***		
	SMAC	***SVM-CBO***	***SVM-CBO$_{RF}$***	***SMAC vs SVM-CBO***	***SMAC vs SVM-CBO$_{RF}$***	***SVM-CBO vs SVM-CBO$_{RF}$***
Branin	-1.018 ($\pm$0.046)	**-1.047** ($\pm$0.001)	-1.021 ($\pm$0.026)	***	-	***
Rosenbrock	1.529 ($\pm$0.855)	**1.177** ($\pm$0.232)	1.99 ($\pm$0.534)	***	***	***
Michalewicz	**-1.463** ($\pm$0.323)	-1.286 ($\pm$0.32)	-1.38 ($\pm$0.290)	*	-	-
Mishra Bird	-81.248 ($\pm$16.93)	**-106.761** ($\pm$0.004)	-103.943 ($\pm$4.585)	***	***	***
Gramacy	0.734 ($\pm$0.077)	**0.693** ($\pm$0.010)	**0.693** ($\pm$0.010)	*	*	-
Alpine N.2	-6.949 ($\pm$0.971)	**-7.557** ($\pm$0.588)	-7.335 ($\pm$0.827)	***	*	**

best seen reached by SMAC.
In terms of statistical significance based on p-value, it is possible to observe how effectively the SVM-CBO and SVM-CBO$_{RF}$ approaches differ from SMAC. However, it should be noted that the level of significance is slightly lower between SMAC and the SVM-CBO$_{RF}$ variant as both approximate the objective function with the surrogate Random Forest model, which can lead to a worse approximation of smooth functions. This fact is precisely shown for functions such as Michalewicz, for which one can see that, on average, the approaches based on the RF surrogate model achieve better values than the original SVM-CBO approach.

To further emphasise this aspect, the Mishra Bird function, in which the global optimum lies on the boundary of the admissible region, was discretized. Discretizing the domain of this function in a grid of 625 points (25$\times$25) reported in Figure 8.1, it was possible to observe that with 30 independent runs on average the variant SVM-CBO$_{RF}$, by better approximating the objective function which is no longer smooth, reaches better values than its traditional version (SVM-CBO). The fact of having a mixed search space is common in AutoML tasks, especially for finding the optimal values of categorical and integer hyperparameters.

Moreover, from Table 8.1, for almost every experiment in which SVM-CBO reaches values better than SMAC, it is possible to see the same behavior for the optimal values reached by the variant SVM-CBO$_{RF}$.
After evaluating the performances on two dimensional test problems, Emmental-GKLS function

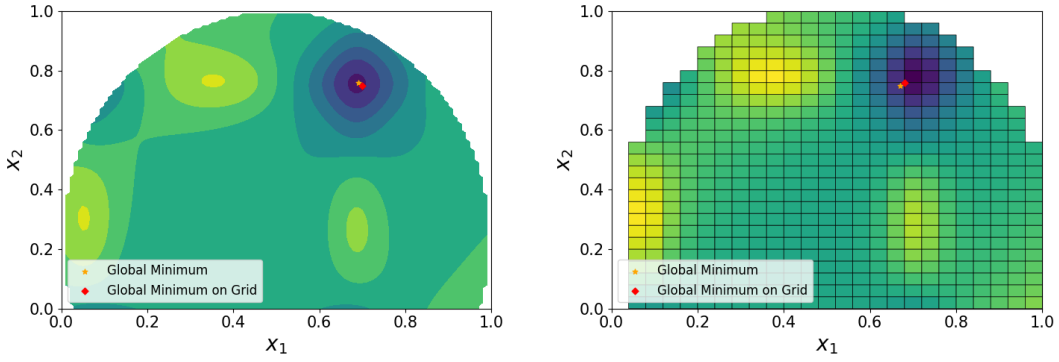


FIGURE 8.1: Comparison between the smooth (left) and discretized (right) Mishra Bird function based on grid of 25 $\times$25 points

generator is considered to generate functions starting from two dimensions up to five dimensions with different levels of difficulty previously reported in (Sergeyev et al., 2018). As before, the Mann-Whitney statistical test was carried out on 30 independent runs for ten different test functions generated through the Emmental-GKLS generator (Sergeyev et al., 2017). The results of all experiments and the p-values are reported in Appendix A due to the number and size of the tables results.

Initially, applying the three optimizers on ten different GKLS test functions with two dimensions, the performance of SVM-CBO$_{RF}$ and SMAC gives similar results, while the results of the original SVM-CBO are not as good as the other optimizers.

However, by increasing the number of dimensions over two, it was possible to observe that SVM-CBO outperforms the other optimizers with a high level of significance in terms of p-values of the statistical tests. The main reason for this obvious gap between the optimizers is still related to using the RF surrogate model, instead of GP, for the approximation of smooth functions.

Despite the relevant differences with the original SVM-CBO, the other optimizers seem on the average to obtain similar results in terms of global optimum found. The only difference that was possible to observe between SVM-CBO$_{RF}$ and SMAC is that when the test function is based on the hard class complexity (Sergeyev et al., 2018), the SVM-CBO$_{RF}$ obtains slight better results in respect to SMAC. Moreover, always taking into account only SVM-CBO$_{RF}$ and SMAC, the percentage of problems where SVM-CBO$_{RF}$ outperforms SMAC is more than 50%.

8.2 AutoTinyML

In this Section the results of the experiments are summarized by comparing the optimal CNN models, obtained through the AutoTinyML framework, and the baseline CNN models of the real use cases provided by STMicroelectronics. The variant SVM-CBO$_{RF}$ was chosen because preliminary experiments showed that the original SVM-CBO algorithm did not lead to good results in terms of accuracy. The main reason is that the Tiny Machine Learning tasks are defined on a mixed search space where tree-based surrogates (e.g. RF) perform better in approximating the non-smooth objective function.
For this real-life application, BOHB (Falkner et al., 2018) has been chosen as benchmark approach - instead of SMAC - since it was already succesufully and recently adopted for HPO of DNN models, such as in (Mendoza et al., 2019; Zimmer et al., 2020). Since 15 different experiments were performed with SVM-CBO$_{RF}$ and BOHB for each pair $<$ *MCU - compression factor* $>$, the results in terms of mean and standard deviation of the following metrics are the following:

- **Accuracy** – computed on the test set by using the C-optimized CNN models;
- **RAM** – whose value must satisfy the associated deployability constraint related to the MCU's hardware;
- **ROM** – whose value must satisfy the associated deployability constraint related to the MCU's hardware;
- **X-CROSS Accuracy** – whose value must satisfy the associated deployability constraint reported in Section 7.2.2 to quantify how much different are the predictions of compressed model to respect to the original one;
- **MACC** – that is the Multiply ACCumulated operations. This measure is not considered neither as objective or constraint in SVM-CBO$_{RF}$. It is reported just as a further index about the logical complexity of the C-optimized CNN models.

Human Activity Recognition
All the results are summarized in Table 8.2 and Table 8.3 for the *Big Board* and the *Tiny Board*, respectively.
With respect to the *Big Board*, SVM-CBO$_{RF}$ was able to always identify CNN models which are deployable on the MCU, for both the compression factors considered, and with a significant reduction in terms of requested resources compared to the baseline model. Furthermore, the CNN models identified by SVM-CBO$_{RF}$ offer, on average, a slightly better accuracy. Another relevant consideration is that the higher compression factor ($\times 4$) allows to further reduce ROM, but shows higher values for RAM and MACC compared to the lower compression factor ($\times 1$).

The constraint related to X-CROSS accuracy is largely satisfied for both the compression factors considered.

Moreover, as it can be seen in Table 8.2, the comparison between the proposed optimization system (SVM-CBO$_{RF}$) and BOHB (Falkner et al., 2018) applied on HAR task. Using the same budget of evaluations (100 evaluated CNN configurations) during the optimization process, SVM-CBO$_{RF}$ is able to provide on average better CNN configurations models with which it was possible to reach higher values of accuracy metrics for each compression factor respectively. Moreover, using the same evaluation budget as SVM-CBO$_{RF}$, BOHB turns out to have lower average accuracy values than the non-deployable baseline CNN models for each compression factor adopted. Although BOHB finds CNN configurations with reduced average accuracy values, it manages to propose configurations that use less hardware resources than SVM-CBO$_{RF}$ and the CNN baseline model.

TABLE 8.2: Optimization results using *Big Board*

	Compression $\times 1$			***Compression*** $\times 4$		
	Baseline	***SVM-CBO$_{RF}$***	***BOHB***	***Baseline***	***SVM-CBO$_{RF}$***	***BOHB***
Accuracy	92.09%	**92.44%** (± 0.64)	91.48% (± 1.20)	92.07%	**92.93%** (± 0.55)	91.95% (± 0.77)
RAM	24.57	9.75 (± 5.11)	6.79 (± 3.41)	24.57	15.16 (± 7.92)	9.28 (± 5.82)
ROM	2,955.80	688.30 (± 144.42)	607.45 (± 282.46)	794.13	546.31 (± 283.34)	336.27 (± 255.41)
X-CROSS	99.98%	100.00% (± 0.00)	100.00% (± 0.00)	99.98%	99.70% (± 0.53)	99.90% (± 0.03)
MACC	874,970	269,327 (± 83,895)	228,884 (± 73,044)	874,970	784,767 (± 318,078)	409,050 (± 257,357)

Summarizing, with respect to the *Big Board* SVM-CBO$_{RF}$ was able to obtain a smaller deployable model than the baseline one, without any reduction in classification accuracy.

Since this could mean that a further reduction of the model could be possible, we run SVM-CBO$_{RF}$ also on the *Tiny Board*, which is characterized by even more limited hardware resources. Also for *Tiny Board* reported in Table 8.3, SVM-CBO$_{RF}$ was able to always identify CNN models which are deployable on the MCU, for both the compression factors considered, and with a significant reduction in terms of requested resources compared to the baseline model. However, in this case, and with respect to the lower compression factor ($\times 4$), we have experienced a reduction in accuracy compared to the baseline model. With respect to the higher compression factor ($\times 8$), considerations are similar to those reported for the *Big Board*. Moreover, we have obtained a significant reduction in both RAM and ROM, similar for both the two compression

factors considered. Also MACC showed a drastic reduction which is more significant with respect to the lower compression factor ($\times 4$). Again, the constraint about X-CROSS accuracy is largely satisfied.
In addition, Table 8.3 shows a direct comparison between the results obtained from the proposed system (SVM-CBO$_{RF}$) and BOHB (Falkner et al., 2018).

TABLE 8.3: Optimization results using *Tiny Board*

	Compression $\times 4$			***Compression*** $\times 8$		
	Baseline	***SVM-CBO$_{RF}$***	***BOHB***	***Baseline***	***SVM-CBO$_{RF}$***	***BOHB***
Accuracy	92.07%	**88.27%** (± 2.47)	87.26% (± 1.63)	91.32%	**91.41%** (± 0.85)	89.65% (± 1.11)
RAM	24.57	5.17 (± 3.04)	5.94 (± 3.14)	24.57	5.42 (± 3.22)	6.74 (± 3.33)
ROM	794.13	44.93 (± 12.34)	38.28 (± 12.02)	375.45	49.51 (± 11.04)	42.97 (± 16.42)
X-CROSS	99.98%	99.86% (± 0.03)	99.83% (± 0.11)	98.04%	98.42% (± 0.59)	98.23% (± 0.77)
MACC	874,970	87,110 (± 34,747)	75,509 (± 27,950)	874,970	159,714 (± 61,613)	134,306 (± 57,017)

Moreover, to prove the difference in accuracy between the models proposed by SVM-CBO$_{RF}$ and BOHB, the unpaired Mann-Whitney test was performed as previously done for the test functions. Table 8.4 shows that the level of significance increases when the highest compression factor is applied for the Big Board and the Tiny Board, respectively. To be fair, it should be noted that BOHB manages to produce deployable CNNs even in the more complex case of the Tiny Board. However, looking at Table 8.3 the only accuracy metric of the proposed networks, with the same evaluation (budget) of the objective function, SVM-CBO$_{RF}$ produces higher results than both the baseline model and the average of the models proposed by BOHB. In particular, it is evident how SVM-CBO$_{RF}$ both compression levels ($\times 4$ and $\times 8$) use less RAM memory than the solution found by BOHB, while using more ROM memory which is presumably a less stringent hardware constraint than RAM. From this point of view SVM-CBO$_{RF}$ seems to be working better if the hardware constraints are more stringent due to its ability at finding out the optimal configuration of the CNN at the boundary of the deployable region.

Some considerations must be made regarding the results of the use-case just presented. The comparison of the two approaches (SVM-CBO$_{RF}$ and BOHB) was made with the main goal of finding out the model with the best result in terms of performance metrics in a fixed budget of 100 evaluations satisfying the requirements of the given MCU target.

TABLE 8.4: Results of unpaired Mann-Whitney test on the accuracy values obtained with the experiments. The significance level is calculated on p-value, where "***" for p-value$<$0.001, "**" for p-value$<$0.01, "*" p-value$<$0.05, and "-" whether p-value$\geq$0.05

Experiment	***SVM-CBO$_{RF}$***	***BOHB***	***p-value***	***Significance Level***
Big Board Compression $\times 1$	92.441% ($\pm$0.637)	91.478% ($\pm$1.204)	0.02018	*
Big Board Compression $\times 4$	92.929% ($\pm$0.546)	91.948% ($\pm$0.767)	0.00097	***
Tiny Board Compression $\times 4$	88.267% ($\pm$2.469)	87.260% ($\pm$1.627)	0.23695	-
Tiny Board Compression $\times 8$	91.413% ($\pm$0.853)	89.645% ($\pm$1.115)	0.00016	***

However, BOHB has requested more evaluations despite the fixed evaluations budget.The reason for the higher number of evaluations is due to the BOHB implementation itself. In fact, the system, based on the Hyperband algorithm (Li et al., 2017b), has the η hyperparameter which, in order to explore better the search space among the possible configurations, increases the number of evaluations of the prefixed budget, arriving in the use case at evaluating up to a maximum of 300 possible configurations with different training strategies (e.g., number of epochs during the training process). BOHB, adopting a diversified training strategy, allows the exclusion of less promising configurations by focusing only on those that lead to optimizing the performance metric as much as possible.

For completeness are also reported the results related to the progress over the SVM-CBO$_{RF}$ and BOHB iterations. Figure 8.2 refers to the *Big Board*, with compression factor $\times 1$ and $\times 4$, respectively. These charts show how the best observed value of the classification error on the validation set changes over the iterations (i.e., hyperparameters configurations suggested by the previously mentioned approaches). The same kind of information is reported in Figure 8.3 for the *Tiny Board*, with compression factor $\times 4$ and $\times 8$, respectively. As shown in Figures 8.2 and 8.3 the proposed approach outperforms BOHB.
Besides, in all the four figures, it is possible to note that, even if the phase 1 of SVM-CBO$_{RF}$ is not aimed at optimizing, but just to estimate the feasible region, there is an improvement in terms of error reduction. Similar behavior has been already observed in (Candelieri, 2019), in which a large set of test functions of different dimensionality was considered. This usually occurs when the optimum is close to the boundary of the actual feasible region and, consequently, sampling for obtaining a better estimate of this region can lead to observing better values of the objective function. It is quite easy to imagine that this situation can easily occur in real-life problems, where constraints limit further improvement otherwise achievable.

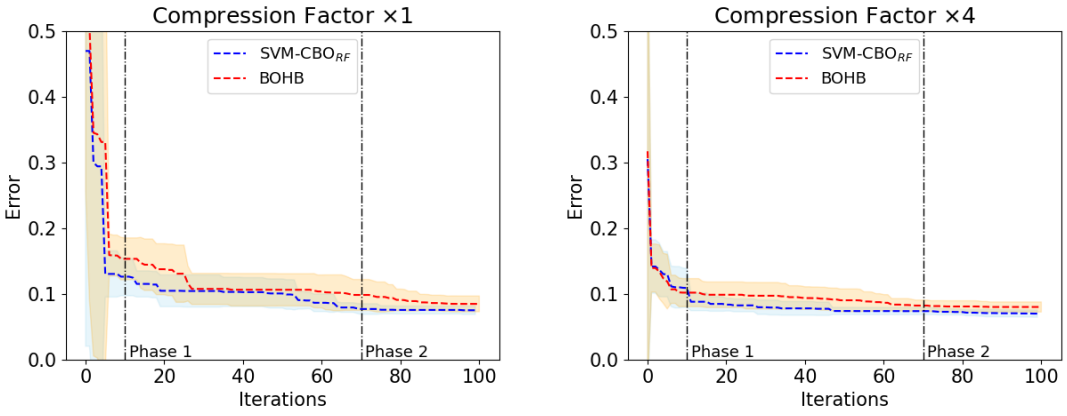


FIGURE 8.2: *Big Board* with compression factors ×1 and ×4: error on the validation set over the SVM-CBO$_{RF}$ and BOHB optimization processes

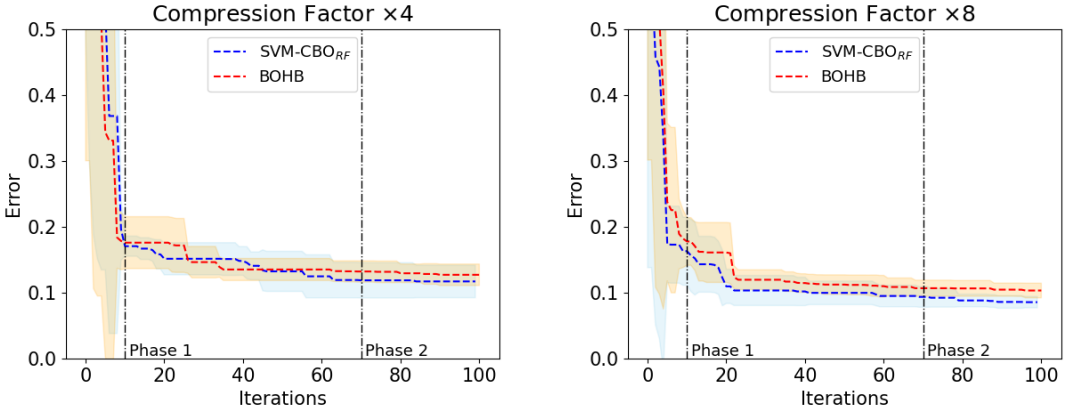


FIGURE 8.3: *Big Board* with compression factors ×4 and ×8: error on the validation set over the SVM-CBO$_{RF}$ and BOHB optimization processes

Image Recognition

This second HPO task was strongly driven by STMicroelectronics and the request to develop a real-life system for image recognition. According to the results obtained in the previous task, the choice of STMicroelectronics was to invest time and computational effort only on performing AutoTinyML through SVM-CBO$_{RF}$. Thus, just for this application, only results provided by the approach proposed in this thesis are reported, without any comparison against other tools. The experiments were mainly driven by the need to identify, as soon as possible, a DNN model more accurate than the baseline model on a new dataset recently collected and shared by STMicroelectronics (Pau et al., 2020).

For the second HPO task (Image Recognition), further studies were required to apply the current proposed system to the baseline CNN. In fact, as previously reported in Section 7.2.2, this task is more complex to optimize than the previous one (Human Activity Recognition). Looking at

Table 7.5 it is possible to note that for each MCU considered (Big Board/Tiny Board) and compression factor, it is not possible to execute a CNN model that respects the hardware constraints of the target MCUs.

Despite the difficulty of the task, by applying SVM-CBO$_{RF}$ to this task on the Big Board MCU with maximum compression factor ($\times$8), it was possible to generate optimal and deployable CNN models which are better on the average (on 15 different indipendent runs) than current CNN baseline. The optimal models increase up to 2% the accuracy while reducing significantly more than 80% the ROM usage, MACC operations, and reducing more than 50% on RAM usage, which is the main issue related to being deployable for the current CNN baseline on this task.

TABLE 8.5: Results Image Recognition task with compression $\times$8 on *Big Board*

	Compression $\times$8		
	Baseline	***SVM-CBO$_{RF}$***	***Avg. Difference***
Accuracy	96.20%	**98.20%** ($\pm$ 0.57)	**+2.00%**
RAM	140.80	65.29 ($\pm$ 10.62)	**-53.63%**
ROM	862.42	119.60 ($\pm$ 72.50)	**-86.13%**
X-CROSS	99.05%	99.57% ($\pm$ 0.63)	**+0.52%**
MACC	20,863,612	3,240,113 ($\pm$ 1,263,412)	**-84.47%**

Instead, some considerations must be made regarding the experiments carried out with the Tiny Board as the target MCU. As previously reported in Table 7.5, with this target MCU was not possible to retrieve any deployable CNN models only adopting the traditional HPO task modifying the values of hyperparameters of the baseline CNN.

For this reason, an additional hyperparameter, named *removal layer*, related to the structure of the CNN is considered to simplify the original baseline model as reported in the search space in Table 7.7. With the introduction of the new structural hyperparameter into search space, it was possible to observe from Table 8.6 that, although the proposed system was able to find deployable CNN configurations, the average accuracy reached is reduced by more than 9% compared to the baseline model. Despite the reduction in accuracy measure, the hardware constraints are largely satisfied, reducing the average RAM and ROM usage of 95% and more than 97% the MACC operations required for the MCU to make the inference through the CNN model.

The poor performance in terms of accuracy could be motivated by the fact that adding this new structural hyperparameter of CNN can significantly increase the size of whole search space, making the HPO task more challenging to find optimal models with almost the same iterations

budget of the optimization process. However, to increase the performance in terms of accuracy taking the best hyperparameters configurations of the several runs, new networks were trained, increasing the number of training epochs and early stopping in order to generate networks that, with the same hardware resources used, had on average values of accuracy superior to the trained networks exactly like the starting baseline model.
In addition, the last two columns of Table 8.6 show that besides a refinement of the hyperparameters that defines the architecture of the neural network, it can be relevant to consider also a manipulation a posteriori or a priori of the optimization process of the variables that define the learning process of the neural network.

TABLE 8.6: Results Image Recognition task with compression $\times 8$ on *Tiny Board* with hyperparameters for changing the CNN architecture and modified training strategy

	Baseline Training Strategy			***Advanced Training Strategy***	
	Baseline	***SVM_CBO$_{RF}$***	***Avg. Difference***	***SVM_CBO$_{RF}$***	***Avg. Difference***
Accuracy	96.20%	86.72% (± 11.68)	**-9.48%**	**92.00**% (± 6.19)	**-4.20%**
RAM	140.80	8.92 (± 1.24)	-93.66%	8.92 (± 1.24)	-93.66%
ROM	862.42	20.46 (± 15.92)	-97.63%	20.46 (± 15.92)	-97.63%
X-CROSS	99.05%	98.07% (± 1.35)	-0.98%	98.07% (± 1.35)	-0.98%
MACC	20,863,612	538,691 (± 163,633)	-97.42%	538,691 (± 163,633)	-97.42%

The two real-life experiments were defined in two different moments, exactly in the order they are presented in the thesis. Contrary to the first experiment (HAR task), results on the second one (Image Recognition task) were less successful and this led to consider the improved training strategy. Although it could lead to better results also for the first experiment, STMicroelectronics was not interested in further investigating that case, but this improved training strategy has been suggested as a more effective and efficient procedure for designing new real-life applications.

Chapter 9

Conclusion and Future Works

In recent years, Machine Learning techniques are used for several applications for real-life problems, typically adopting very powerful machines for computing. However, in the last period, the scientific development has allowed to create more powerful small devices (e.g., microcontrollers) and software increasingly portable and accessible to all, to better support the advent of Edge Computing. Through the development of this decentralized computing paradigm, it will be possible - now in the not so distant future - to analyze and make decisions in real-time (or near real-time) thanks to sophisticated machine learning models on sensors connected to each other through a network. Major companies such as Google with the implementation of Tensorflow Lite, NVIDIA with the development of the Jetson Nano series of boards, and STMicroelectronics with the STM32Cube.AI framework are trying to exploit the use of high-performance Deep Neural Networks directly on edge devices. Despite their limited hardware capacity, these devices are very relevant in several contexts where immediate decisions need to be made, such as autonomous driving, monitoring for medical and industrial purposes.

Although there are developments in terms of hardware, instead from a software point of view, despite increasingly portable libraries (e.g. Tensorflow Lite), it is required the adoption of more sophisticated techniques to define Machine Learning models that respect hardware limitations, training and inference time constraints for specific applications.

In Edge Computing, the Automated Machine Learning field can help to propose new advanced frameworks for dealing with this decentralized computing paradigm. On this track, the main contribution of this thesis regards the proposal of novel constrained Automated Machine Learning framework, named *AutoTinyML*, for dealing the Hyperparameters Optimization on Deep Neural Network with known and unknown constraints. It starts from recent advances in the field of *Bayesian Optimization under unknown constraints* and introduces innovations to overcome the specific issues limiting the wide application of Automated Machine Learning for tiny devices. Based on collaboration with STMicroelectronics, it was possible to develop the AutoTinyML framework adopting its translating system (STM32Cube.AI) for deploying Deep

Neural Networks on STM32 microcontrollers. In Chapters 7 and 8 the proposed constrained optimization framework is adopted for two real-life use cases on two different MCU provided by the company. The promising results obtained, and presented in this thesis, encourage the adoption of the proposed AutoTinyML, while they also allowed to identify further research and innovation activities. Indeed, the proposed AutoTinyML framework can be considered a suitable solution to unlock the untapped market of TinyML and Automated TinyML. However, it still has a limitation concerning broader applicability, and several future directions will be considered to improve the framework.

A first step to improve the framework could be to analyze whether the constraints are closely related to evaluating the objective function. In fact, the software developed by STMicroelectronics allows, with a larger approximation error, to obtain the values of the constraints (such as RAM and ROM) using pre-trained networks initialized with random weights. With this functionality can be evaluated several configurations with a cheaper cost in terms of time and computational resources, because the training process of the deep neural network is skipped on a given hyperparameters configuration. This functionality, increasing the number of hyperparameters configurations evaluated without training the network, would improve the performance of the classifier used within the SVM-CBO$_{RF}$ optimization process, discriminating better between the deployable and undeployable configurations.

Increasing the number of evaluated hyperparameters configurations can help us better understand which are the more relevant constraints to improve the performance of the deep neural network. This aspect might enable the discrimination between three different classes of hyperparameter configurations: *deployable* configuration which satisfies the constraints, *undeployable* configuration that does not satisfy the constraints, and the *invalid* configuration that does not allow the correct building of a trainable deep neural network. Moreover, additional improvements can be added to the AutoTinyML framework applying some changing inside of its optimizer SVM-CBO$_{RF}$. To better combine the current surrogate model (i.e. Random Forest) with the classifier for deployability discrimination, the acquisition function of the first phase SVM-CBO$_{RF}$ will have to be redefined, adopting RF instead of SVM as a classifier. Through this modification it will be possible to better manage problems with mixed variables without any preprocessing techniques currently necessary for categorical and integer variables. Besides with the adoption of the RF also as a classification algorithm, it will be possible to manage conditional variables, which, if inactive, would be represented as missing values during the learning process of the classification algorithm. In fact, one of the strengths of the RF is its robustness and effectiveness in terms of classification performance in the presence of missing values.

In addition, it may happen that for a given initial baseline network, dataset and target MCU for a specific TinyML task, it is not possible to find models that are really deployable on the given MCU. The impossibility of obtaining a deployable model may be because: (*i*) the search space has been incorrectly defined, (*ii*) the deployability region, defined by the MCU constraints, is much smaller than the defined search space, or even non-existent.

Moreover, to improve the AutoML framework, the adoption of the weak search spaces and constraints will be investigated. Weakly defined search spaces would allow a better and more flexible search for the optimal model by extending or excluding certain regions of the search space. In this way it will be possible, for example, to extend the domain of some decision variables in the case where the proposed solutions are on the border of the set of values that can be assumed by these variables, allowing the improvement of the trade-off between exploration and exploitation during the optimization process.

Instead, adopting weakly defined constraints during the optimization process would allow the AutoTinyML framework to propose to the final-user at least one deployable model on a specific MCU. This would enable STMicroelectronics to suggest what type of MCU is best to adopt for the given dataset and baseline neural network provided by the customer to solve a specific TinyML task.

To further improve the AutoTinyML framework, Meta-Learning techniques could be adopted to better define the search space and constraints of the problem before starting the optimization process based on previous experiments conducted by other STMicroelectronics customers with the same system. The Meta-Learning techniques could be based on using previous optimal models found according to the hardware constraints required by a similar type of MCU or according to the features of the dataset of the given Tiny ML task which are similar to others previously adopted.

Appendix A

Benchmark Test Functions

A.1 Definition of the test functions

Branin test function (Picheny et al., 2013):

$$f(x_1,x_2) = \frac{\left[\left(\frac{x_2-5.1x_1^2}{4\pi^2}+\frac{5x_1}{\pi}-6\right)^2+\left(10-\frac{10}{8\pi}\right)\cos(x_1)-44.81\right]}{51.95} \tag{A.1}$$

with

$$x_1^{c1}=\frac{1}{3} \quad x_2^{c1}=\frac{1}{4} \quad x_1^{c2}=\frac{5}{6} \quad x_2^{c2}=\frac{7}{8}$$

$$a_1=0.45 \quad b_1=0.27 \quad a_2=0.25 \quad b_2=0.1 \quad \alpha=\frac{\pi}{4} \quad \alpha_2=\frac{3\pi}{4}$$

$$s.t.$$

$$c_1(x)<1 \quad \text{OR} \quad c_2(x)<1$$

$$c_1(x)=\frac{\left[\left(x_1-x_1^{c1}\right)\cos(\alpha)+\left(x_2-x_2^{c1}\right)\sin(\alpha)\right]^2}{a_1^2}+\frac{\left[\left(x_2-x_2^{c1}\right)\cos(\alpha)+\left(x_1-x_1^{c1}\right)\sin(\alpha)\right]^2}{b_1^2}$$

$$c_2(x)=\frac{\left[\left(x_1-x_1^{c2}\right)\cos(\alpha_2)+\left(x_2-x_2^{c2}\right)\sin(\alpha_2)\right]^2}{a_2^2}+\frac{\left[\left(x_2-x_2^{c2}\right)\cos(\alpha_2)+\left(x_1-x_1^{c2}\right)\sin(\alpha_2)\right]^2}{b_2^2} \tag{A.2}$$

where $x_1,x_2 \in [0;1]$, $x^* = (0.543, 0.151)$ and $f(x^*) = -1.0474$

Rosenbrock test function constrained to a cubic and a line (Simionescu and Beale, 2002):

$$\begin{gathered} f(x_1,x_2) = (1-x_1)^2 + 100(x_1+x_2^2)^2 \\ s.t. \\ (x_1-1)^3 - x_2 + 1 \leq 0 \\ \text{AND} \\ x_1 + x_2 - 2 < 0 \end{gathered} \tag{A.3}$$

where $x_1 \in [-1.5; 1.5]$, $x_2 \in [-0.5; 2.5]$, $x^* = (1.0, 1.0)$ and $f(x^*) = 0.0$

Michalewicz test function (Al-Roomi, 2015):

$$\begin{gathered} f(x) = -\sum_{i=1}^{2} \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right) s.t. \\ x_1^3 + x_2^3 > 15 \\ s.t. \\ x_1^3 + x_2 < 30 \end{gathered} \tag{A.4}$$

where $x_1, x_2 \in [0; \pi]$, $x^* = (2.20, 1.57)$ and $f(x^*) = -1.8013$

Mishra's Bird test function (Mishra, 2006):

$$\begin{gathered} f(x_1,x_2) = \sin(x_2) e^{(1-\cos(x_1))^2} + \cos(x_1) e^{(1-\sin(x_2))^2} + (x_1 - x_2)^2 \\ s.t. \\ (x_1+5)^2 + (x_2+5)^2 < 25 \end{gathered} \tag{A.5}$$

where $x_1 \in [-10; 0]$, $x_2 \in [-6.5; 0.0]$, $x^* = (-3.1302468, -1.5821422)$ and $f(x^*) = -106.7645367$

Gramacy test function (Gramacy et al., 2016):

$$\begin{gathered} f(x_1,x_2) = x_1 + x_2 \\ s.t. \\ \frac{1}{2}\sin\left(2\pi(x_1^2 - 2x_2)\right) + x_1 + 2x_2 - \frac{3}{2} \geq 0 \\ \text{AND} \\ -x_1^2 - x_2^2 + \frac{3}{2} \geq 0 \end{gathered} \tag{A.6}$$

where x_1,$x_2 \in [0.0; 1.0]$, $x^* = (0.1954, 0.4044)$ and $f(x^*) = 0.5998$

Alpine N.2 test function (Jamil and Yang, 2013):

$$\begin{aligned} f(x) &= -\prod_{i=1}^{2} \sin(x_i) \cdot \sqrt{x_i} \\ &s.t. \\ (x_1-5)^2 + (x_2-5)^2 - \frac{1}{4} &< 17 \end{aligned} \tag{A.7}$$

where x_1,$x_2 \in [0.0; 10.0]$, $x^* = (7.917, 7.917)$ and $f(x^*) = -2.808^2$

A.2 Results of statistical test

TABLE A.1: Results of the six 2D test function on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value<0.001, "**" for p-value<0.01, "*" p-value<0.05, and "-" whether p-value≥0.05

Test Function	*SMAC*		*SVM-CBO*		*SVM-CBO$_{RF}$*		*SMAC vs SVM-CBO*		*SMAC vs SVM-CBO$_{RF}$*		*SVM-CBO vs SVM-CBO$_{RF}$*	
	Mean	*±Std*	*Mean*	*±Std*	*Mean*	*±Std*	*p-value*	*SignLevel*	*p-value*	*SignLevel*	*p-value*	*SignLevel*
Branin	-1.018	0.046	**-1.047**	0.001	-1.021	0.026	0	***	0.647		0	***
Rosenbrock	1.529	0.855	**1.177**	0.232	1.994	0.534	0.001	***	0.001	***	0	***
Michalewicz	**-1.463**	0.323	-1.286	0.32	-1.38	0.29	0.041	*	0.187		0.243	
Mishra Bird	-81.248	16.93	**-106.761**	0.004	-103.943	4.585	0	***	0	***	0	***
Gramacy	0.734	0.077	**0.693**	0.01	**0.693**	0.01	0.022	*	0.022	*	0.761	
Alpine N.2	-6.949	0.971	**-7.557**	0.588	-7.335	0.827	0	***	0.027	*	0.007	**

TABLE A.2: Results of the ***2D Emmental GKLS test function*** on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value<0.001, "**" for p-value<0.01, "*" p-value<0.05, and "-" whether p-value≥0.05

Class	*Seed*	*SMAC*		*SVM-CBO*		*SVM-CBO$_{RF}$*		*SMAC vs SVM-CBO*		*SMAC vs SVM-CBO$_{RF}$*		*SVM-CBO vs SVM-CBO$_{RF}$*	
		Mean	*±Std*	*Mean*	*±Std*	*Mean*	*±Std*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*
Simple	1	-0.3228	0.2304	-0.2637	0.2664	**-0.4414**	0.2406	0.3953	-	0.0169	*	0.0054	**
	2	0.0737	0.1243	0.1033	0.1245	**0.0671**	0.1269	0.2939	-	0.9240	-	0.1614	-
	3	0.1378	0.1350	0.0831	0.0779	**0.0312**	0.1946	0.0571	-	0.0285	*	0.3187	-
	4	0.1206	0.1679	0.1072	0.0232	**0.0933**	0.0575	0.0000	***	0.0000	***	0.0961	-
	5	**-0.0960**	0.2095	-0.0748	0.1378	-0.0941	0.2199	0.1353	-	0.6100	-	0.1599	-
	6	-0.0493	0.0837	-0.0599	0.1272	**-0.1403**	0.2089	0.8360	-	0.0333	*	0.0164	*
	7	0.0156	0.0313	0.0057	0.0750	**-0.0069**	0.1059	0.6256	-	0.9764	-	0.8717	-
	8	**-0.1798**	0.2103	-0.1550	0.1861	-0.1492	0.1964	0.3255	-	0.1714	-	0.7628	-
	9	**0.0221**	0.0841	0.0297	0.0243	0.0246	0.0153	0.0948	-	0.2143	-	0.2357	-
	10	**-0.0414**	0.1502	-0.0134	0.1331	-0.0159	0.1270	0.1137	-	0.3366	-	0.2794	-
Hard	1	-0.3297	0.2164	-0.2387	0.2907	**-0.4003**	0.2233	0.4761	-	0.0676	-	0.0175	*
	2	**0.0853**	0.0368	0.1041	0.0210	0.0984	0.0186	0.0013	**	0.0103	*	0.2036	-
	3	0.1375	0.0850	0.0940	0.0293	**0.0806**	0.0157	0.0321	*	2,00E-04	***	0.0129	*
	4	0.1780	0.0139	0.1764	0.0133	**0.1392**	0.1552	0.6152	-	0.7973	-	0.7132	-
	5	-0.1152	0.1149	**-0.3305**	0.0621	-0.1791	0.1135	0.0000	***	0.0551	-	0.0000	***
	6	0.0449	0.0443	**0.0423**	0.0376	0.0511	0.0455	0.9528	-	0.5444	-	0.5390	-
	7	**-0.0413**	0.1675	-0.0207	0.0613	-0.0071	0.0613	0.5153	-	0.4597	-	0.1824	-
	8	0.1030	0.0928	0.0550	0.0338	**0.0520**	0.0187	0.0044	**	0.0467	*	0.9330	-
	9	**0.0229**	0.0408	0.0242	0.0244	0.0329	0.0407	0.5444	-	0.3994	-	0.8382	-
	10	**0.0221**	0.0219	0.0249	0.0193	0.0284	0.0225	0.5543	-	0.4671	-	0.6971	-

TABLE A.3: Results of the ***3D Emmental GKLS test function*** on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value<0.001, "**" for p-value<0.01, "*" p-value<0.05, and "-" whether p-value≥0.05

Class	*Seed*	*SMAC*		*SVM-CBO*		*SVM-CBO$_{RF}$*		*SMAC vs SVM-CBO*		*SMAC vs SVM-CBO$_{RF}$*		*SVM-CBO vs SVM-CBO$_{RF}$*	
		Mean	±*Std*	*Mean*	±*Std*	*Mean*	±*Std*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*
Simple	1	**0.0619**	0.0458	**0.0403**	0.0140	0.1246	0.0859	0.0000	***	9,00E-04	***	0.0000	***
	2	0.0319	0.1529	**-0.1821**	0.1359	**-0.0247**	0.1323	0.0000	***	0.1297	-	7,00E-04	***
	3	0.1435	0.1074	**0.1165**	0.0826	**0.1239**	0.1810	0.1494	-	0.8534	-	0.7132	-
	4	**0.0826**	0.0711	**0.0202**	0.0506	0.0845	0.0920	0.0000	***	0.6309	-	0.0070	**
	5	**0.0905**	0.0653	**0.0453**	0.0458	0.1224	0.0744	1,00E-04	***	0.0555	-	0.0000	***
	6	**0.0911**	0.0605	**0.0905**	0.0676	0.1401	0.1024	0.7973	-	0.0878	-	0.0762	-
	7	0.1798	0.0904	**0.1080**	0.0196	**0.1415**	0.0561	0.0000	***	0.0263	*	0.0034	**
	8	**0.0915**	0.0534	**0.0621**	0.0146	0.1159	0.0421	0.0067	**	0.0026	**	0.0000	***
	9	0.1204	0.0871	**0.0314**	0.0250	**0.0396**	0.0630	0.0000	***	1,00E-04	***	0.3707	-
	10	**0.1569**	0.0366	**0.1414**	0.0262	0.2097	0.0583	0.2062	-	1,00E-04	***	0.0000	***
Hard	1	0.0861	0.1294	**0.0299**	0.0353	**0.0655**	0.1609	1,00E-04	***	0.5692	-	0.0491	*
	2	0.0387	0.1248	**-0.2043**	0.1486	**-0.0113**	0.1133	0.0000	***	0.0519	-	1,00E-04	***
	3	**0.1376**	0.0805	**0.1331**	0.0179	0.1580	0.0599	0.7506	-	0.2580	-	0.0345	*
	4	0.0824	0.0752	0.0330	0.0411	**0.0154**	0.0829	2,00E-04	***	0.0028	**	0.9655	-
	5	**0.0866**	0.0651	**0.0325**	0.0285	0.1246	0.1140	0.0000	***	0.0274	*	0.0000	***
	6	**0.1320**	0.0965	**0.0817**	0.0499	0.1450	0.0914	0.0224	*	0.5520	-	0.0057	**
	7	0.1894	0.1135	**0.0970**	0.1023	**0.1577**	0.1430	9,00E-04	***	0.6650	-	5,00E-04	***
	8	**0.0824**	0.0418	**0.0608**	0.0204	0.1127	0.0464	0.0067	**	0.0029	**	0.0000	***
	9	0.0806	0.0867	**0.0162**	0.0180	**0.0452**	0.0651	0.0000	***	0.0933	-	0.0164	*
	10	**0.1530**	0.0555	**0.1109**	0.1520	0.1726	0.1744	0.0232	*	0.0073	**	0.0000	***

TABLE A.4: Results of the ***4D Emmental GKLS test function*** on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value<0.001, "**" for p-value<0.01, "*" p-value<0.05, and "-" whether p-value≥0.05

Class	*Seed*	*SMAC*		*SVM-CBO*		*SVM-CBO$_{RF}$*		*SMAC vs SVM-CBO*		*SMAC vs SVM-CBO$_{RF}$*		*SVM-CBO vs SVM-CBO$_{RF}$*	
		Mean	±*Std*	*Mean*	±*Std*	*Mean*	±*Std*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*
Simple	1	0.3608	0.2221	**0.1396**	0.0474	**0.3364**	0.1956	0.0000	***	0.8766	-	0.0000	***
	2	**0.2746**	0.0748	**0.2064**	0.0309	0.3206	0.1807	0.0000	***	0.5106	-	0.0000	***
	3	**0.3638**	0.1050	**0.2648**	0.0828	0.3704	0.0963	0.0000	***	0.8883	-	3,00E-04	***
	4	0.3905	0.1780	**0.1400**	0.0122	**0.3500**	0.2135	0.0000	***	0.2282	-	0.0000	***
	5	0.3855	0.1174	**0.3137**	0.0519	**0.3842**	0.1191	6,00E-04	***	0.9352	-	0.0010	**
	6	0.3577	0.1712	**0.1858**	0.0187	**0.3009**	0.1966	0.0000	***	0.0436	*	0.0000	***
	7	0.1706	0.1669	**0.0525**	0.1443	**0.1236**	0.3013	0.0000	***	0.5011	-	0.1096	-
	8	0.2888	0.2474	**0.0923**	0.0264	**0.2261**	0.1140	0.0000	***	0.7191	-	0.0000	***
	9	**0.3193**	0.0870	**0.2182**	0.0137	0.3824	0.1657	0.0000	***	0.2359	-	0.0000	***
	10	0.5613	0.2279	**0.1968**	0.0261	**0.3714**	0.1446	0.0000	***	7,00E-04	***	0.0000	***
Hard	1	0.3336	0.1512	**0.1472**	0.0107	**0.2446**	0.0837	0.0000	***	0.0127	*	0.0000	***
	2	**0.2639**	0.0679	**0.2178**	0.0344	0.3969	0.1660	0.0076	**	4,00E-04	***	0.0000	***
	3	0.3467	0.1547	**0.2215**	0.0591	**0.3464**	0.1788	2,00E-04	***	0.947	-	1,00E-04	***
	4	0.3922	0.1684	**0.1395**	0.0134	**0.3623**	0.2086	0.0000	***	0.2398	-	0.0000	***
	5	**0.4138**	0.1163	**0.3352**	0.0425	0.4272	0.1012	3,00E-04	***	0.4232	-	0.0000	***
	6	0.3601	0.1698	**0.1836**	0.0171	**0.2775**	0.1153	0.0000	***	0.0877	-	0.0000	***
	7	**0.139**	0.1971	**0.0262**	0.1811	0.1558	0.2195	0.0000	***	0.5298	-	0.0017	**
	8	0.2775	0.2451	**0.0905**	0.0358	**0.2045**	0.1510	0.0000	***	0.843	-	1,00E-04	***
	9	**0.3072**	0.0902	**0.2244**	0.0194	0.4296	0.2192	0.0000	***	0.0415	*	0.0000	***
	10	0.5019	0.2015	**0.1992**	0.0270	**0.3568**	0.1616	0.0000	***	0.0037	**	0.0000	***

TABLE A.5: Results of the ***5D Emmental GKLS test function*** on the best seen for all optimizers. The significance level is calculated on p-value, where "***" for p-value<0.001, "**" for p-value<0.01, "*" p-value<0.05, and "-" whether p-value≥0.05

Class	*Seed*	*SMAC*		*SVM-CBO*		$SVM\text{-}CBO_{RF}$		*SMAC vs SVM-CBO*		*SMAC vs* $SVM\text{-}CBO_{RF}$		*SVM-CBO vs* $SVM\text{-}CBO_{RF}$	
		Mean	±*Std*	*Mean*	±*Std*	*Mean*	±*Std*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*	*P-Value*	*SignLevel*
Simple	1	**0.4390**	0.1469	**0.1738**	0.0203	0.4948	0.2773	0.0000	***	0.8015	-	0.0000	***
	2	**0.3962**	0.1792	**0.0768**	0.0223	0.4339	0.1967	0.0000	***	0.3790	-	0.0000	***
	3	**0.4275**	0.1548	**0.1428**	0.0200	0.4601	0.1848	0.0000	***	0.4161	-	0.0000	***
	4	**0.5659**	0.3075	**0.2239**	0.0200	0.6863	0.4218	0.0000	***	0.4290	-	0.0000	***
	5	**0.3708**	0.1986	**0.0757**	0.0466	0.4457	0.2400	0.0000	***	0.1170	-	0.0000	***
	6	0.4752	0.1972	**0.2317**	0.0250	**0.4646**	0.1630	0.0000	***	0.9411	-	0.0000	***
	7	0.4301	0.1979	**0.0980**	0.0151	**0.4288**	0.1671	0.0000	***	0.9000	-	0.0000	***
	8	0.6032	0.2740	**0.2047**	0.0184	**0.3940**	0.1787	0.0000	***	0.0040	**	0.0000	***
	9	**0.4619**	0.1750	**0.2382**	0.0270	0.4622	0.1616	0.0000	***	0.9470	-	0.0000	***
	10	0.6290	0.3914	**0.2128**	0.0293	**0.4558**	0.1704	0.0000	***	0.1188	-	0.0000	***
Hard	1	**0.4552**	0.1457	**0.1698**	0.0161	0.4796	0.1915	0.0000	***	0.8015	-	0.0000	***
	2	**0.4552**	0.1457	**0.0827**	0.0242	0.4760	0.1879	0.0000	***	0.5997	-	0.0000	***
	3	**0.4043**	0.1974	**0.1464**	0.0312	0.4515	0.2005	0.0000	***	0.2805	-	0.0000	***
	4	**0.5526**	0.2012	**0.2969**	0.0179	0.6558	0.3832	0.0000	***	0.4853	-	0.0000	***
	5	0.3708	0.1986	**0.0878**	0.0286	**0.3659**	0.1786	0.0000	***	0.7006	-	0.0000	***
	6	0.4102	0.3048	**0.2228**	0.1295	**0.3498**	0.2366	0.0000	***	0.3364	-	4,00E-04	***
	7	**0.4160**	0.1909	**0.0944**	0.0169	0.4261	0.1786	0.0000	***	0.5346	-	0.0000	***
	8	0.6097	0.2703	**0.2112**	0.0345	**0.4289**	0.2035	0.0000	***	0.0044	**	0.0000	***
	9	0.4554	0.1604	**0.2491**	0.0308	**0.4483**	0.2336	0.0000	***	0.9528	-	1,00E-04	***
	10	0.6461	0.3968	**0.2122**	0.025	**0.4287**	0.1738	0.0000	***	0.0184	*	0.0000	***

Appendix B

Multi-objective vs Constrained Single-objective optimization

This Appendix reports simple examples of why, in particular situations, it is better to adopt a constrained single-objective optimization instead of a multi-objective optimization strategy. In many practical cases, the optimum may lie on the boundary of the feasible region. In these situations, adopting a multi-objective optimization strategy can not be the best choice because finding out a trade-off among the considered objectives can improve some of them and negatively impact others. Instead, it could be a better option to optimize a single objective function which results to be the most relevant objective among the several considered, setting the other objectives as constraints. The two following toy examples are presented to compare multi-objective and constrained single-objective optimization. The multi-objective optimization approach considered is NSGA-II (Deb et al., 2002) while for the constrained single-objective optimization is considered SVM-CBO (Candelieri, 2019). Figure B.1 shows the two objectives (from now named *metric*) that are depended by one decision variable x.

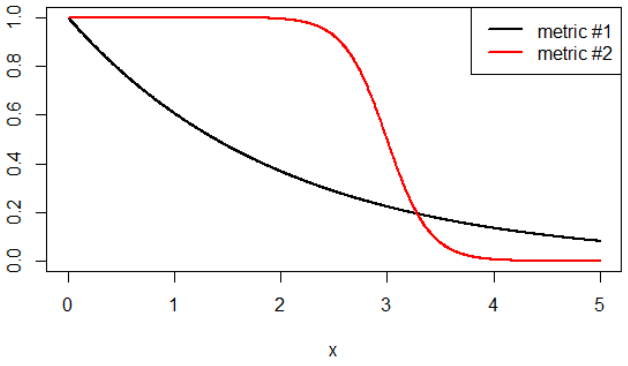


FIGURE B.1: Two metrics (metric #1 and metric #2) as functions of the decision variable x

The metric #1 is characterized by an exponential decay while the metric #2 is characterized by a logistic switch beyond a specific threshold. Suppose we want to minimize the metric #1 and maximize the metric #2, where improving the metric #1 has a negative impact on the metric #2. Considering that the formulations of the two metrics are totally black-box, we set up the optimization experiments considering to evaluate only 20 points of the decision variable x. In particular, for the evolutional algorithm NSGA-II, we set a population size of 4 individuals (points) with 5 generations for an overall of 20 evaluated points.
In the following Figure B.2, on top, it is possible to observe the "true" Paretian frontier (in light blue) and the one approximated by the NSGA-II solutions[1] (in blue). However, considering the case where there is a desired minimum value for metric #2- specifically we do not want it to fall below the value 0.8 - it is possible to see that NSGA-II does not identify the global constrained optimum with the found solutions on the Paretian frontier.

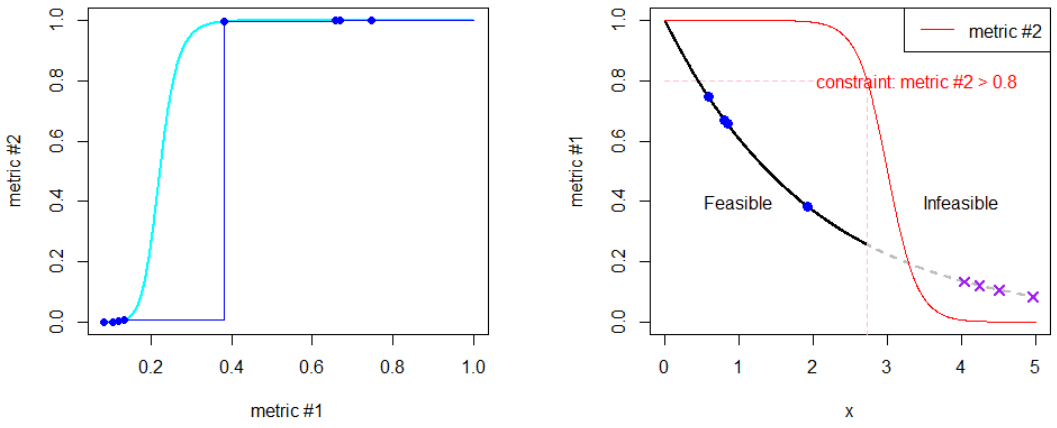


FIGURE B.2: The Paretian frontier of NSGA-II (left) and the best solutions proposed by NSGA-II (right) on the first toy example after 20 function evaluations

Instead, Figure B.3 reports the same previously considered experiment with the adoption of a constrained optimization approach (SVM-CBO). This optimizer minimizes only the metric #1 considering the minimum desirable value of metric #2 as a constraint.
Figure B.3 shows that after the first phase of the constrained optimization process, the proposed solutions lie on the boundary of the feasible region, providing better solutions in respect to the multi-objective optimization approach. At the end of the second phase, the constrained optimization process converges to propose almost the same solution approximating better the metric #1.

In terms of numerical results reported in Table B.1, even if maximizing the metric #2, NGSA-II has a significant loss in terms of metric #1.

[1] Only the last best proposed solution are showed into the plot

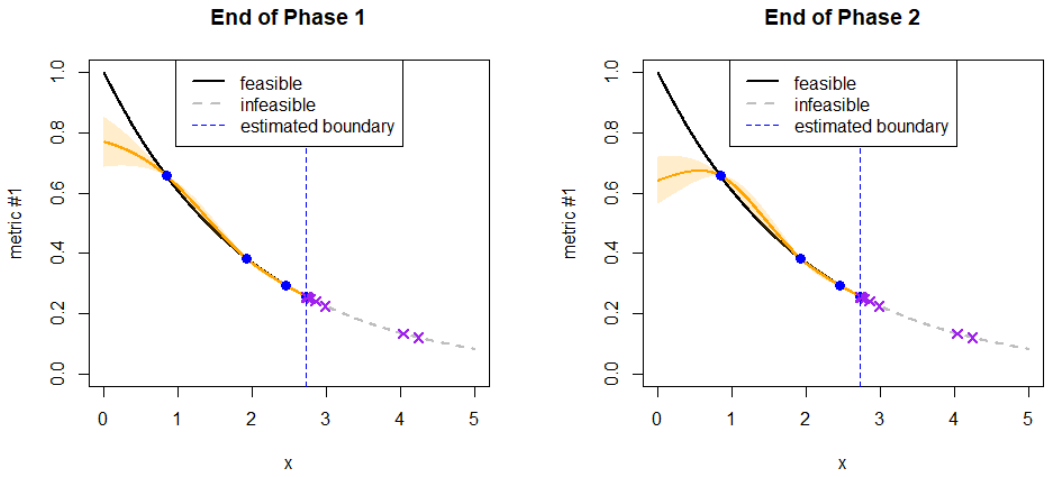


FIGURE B.3: The solutions provided by SVM-CBO approach after 20 functions evaluations

	Optimal value (only feasible solutions)	
Optimizer	**Metric #1**	**Metric #2**
Multi-objective	0.3820	0.9953
Constrained Single-objective	0.2563	0.8000

TABLE B.1: Comparison of the best feasible solutions found on the first toy example by the two different optimizers

However, the difference between these two optimizers is more remarked if we consider a less smooth function for the metric #1 on Figure B.4. Indeed, from Figure B.4 can be observed that NSGA-II proposes its best solutions quite far from the optimal solution that minimizes the metric #1.

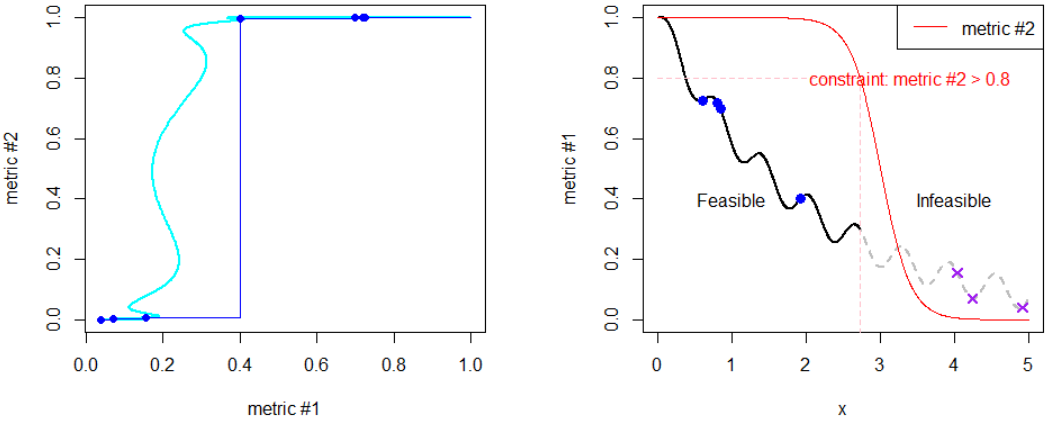


FIGURE B.4: The Paretian frontier of NSGA-II (left) and the best solutions proposed by NSGA-II (right) on the second toy example after 20 function evaluations

Instead, from Figure B.5 can be seen that the constrained optimizer, exploiting better the feasibility of the proposed solutions, can explore in a better way the search space of the problem quickly converging to the constrained global optimal solution of metric #1 that lies on the boundary of the feasible region.

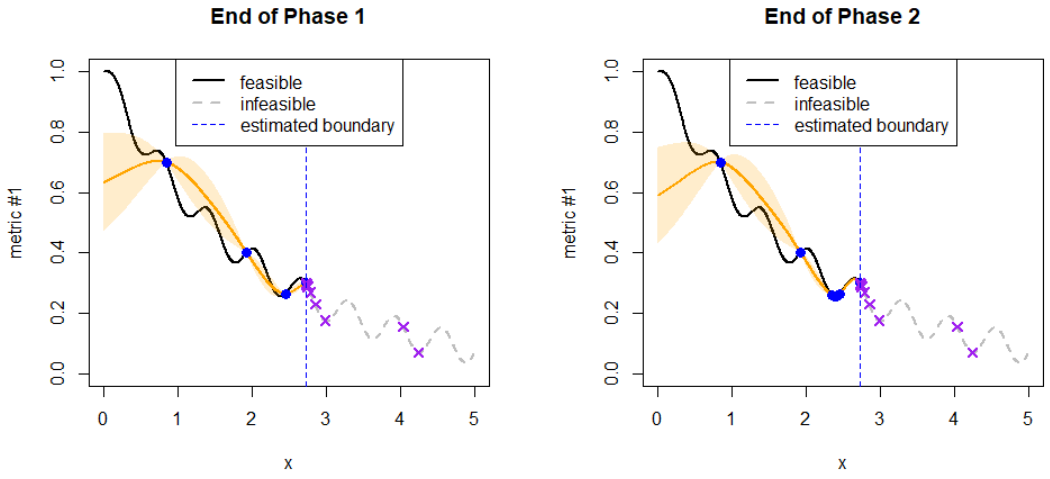


FIGURE B.5: The solutions provided by SVM-CBO approach after 20 function evaluations

As reported for the previous example, in Table B.2 can be seen the significant numeric difference between the optimal values reached by the two different optimizers. In fact, it can be observed that the adoption of constrained optimization strategy leads to a slightly lower value of metric #2 (considered as constraint) and a significant difference in metric #1 compared to the multi-objective optimization approach.

	Optimal value on feasible solutions	
Optimizer	**Metric #1**	**Metric #2**
Multi-objective	0.4013	0.9953
Constrained Single-objective	0.2555	0.9553

TABLE B.2: Comparison of the best feasible solutions found on the second toy example by the two different optimizers

Bibliography

Ajith Abraham and Hongbo Liu. Turbulent particle swarm optimization using fuzzy parameter tuning. In *Foundations of Computational Intelligence Volume 3*, pages 291–312. Springer, 2009.

Mohamed Osama Ahmed, Sharan Vaswani, and Mark Schmidt. Combining bayesian optimization and lipschitz optimization. *Machine Learning*, 109:79–102, 2020.

Ali R. Al-Roomi. Unconstrained Single-Objective Benchmark Functions Repository, 2015. URL `https://www.al-roomi.org/benchmarks/unconstrained`.

Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

Francesco Archetti and Antonio Candelieri. *Bayesian Optimization and Data Science*. Springer, 2019.

Setareh Ariafar, Jaume Coll-Font, Dana H Brooks, and Jennifer G Dy. ADMMBO: Bayesian Optimization with Unknown Constraints using ADMM. *Journal of Machine Learning Research*, 20(123):1–26, 2019.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017.

Kinjal Basu and Souvik Ghosh. Analysis of thompson sampling for gaussian process optimization in the bandit setting. *arXiv*, pages arXiv–1705, 2017.

Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123. PMLR, 2013.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

Julian Berk, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Randomised gaussian process upper confidence bound for bayesian optimisation. *arXiv preprint arXiv:2006.04296*, 2020.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 491–496. IEEE, 2016.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. No-regret bayesian optimization with unknown hyperparameters. *Journal of Machine Learning Research*, 20(50):1–24, 2019.

Simone Bianco, Marco Buzzelli, Gianluigi Ciocca, and Raimondo Schettini. Neural architecture search for image saliency fusion. *Information Fusion*, 57:89–101, 2020.

Eric Brochu, Matthew Hoffman, and Nando de Freitas. Portfolio allocation for bayesian optimization. *arXiv preprint arXiv:1009.5419*, 2010.

Krisztian Buza. Feedback prediction for blogs. In *Data analysis, machine learning and knowledge discovery*, pages 145–152. Springer, 2014.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2787–2794, 2018.

Antonio Candelieri. Sequential model based optimization of partially defined functions under unknown constraints. *Journal of Global Optimization*, pages 1–23, 2019.

Antonio Candelieri and Riccardo Perego. Dimensionality reduction methods to scale bayesian optimization up. *Numerical Computations: Theory and Algorithms NUMTA 2019*, page 167, 2019.

Antonio Candelieri, Riccardo Perego, and Francesco Archetti. Bayesian optimization of pump operations in water distribution systems. *Journal of Global Optimization*, 71(1):213–235, 2018.

Antonio Candelieri, Bruno Galuzzi, Ilaria Giordani, Riccardo Perego, and Francesco Archetti. Optimizing partially defined black-box functions under unknown constraints via sequential model based optimization: an application to pump scheduling optimization in water distribution networks. In *International Conference on Learning and Intelligent Optimization*, pages 77–93. Springer, 2019.

Antonio Candelieri, Ilaria Giordani, Riccardo Perego, and Francesco Archetti. Composition of kernel and acquisition functions for high dimensional bayesian optimization. In Ilias S. Kotsireas and Panos M. Pardalos, editors, *Learning and Intelligent Optimization*, pages 316–323, Cham, 2020a. Springer International Publishing. ISBN 978-3-030-53552-0.

Antonio Candelieri, Riccardo Perego, and Francesco Archetti. Green machine learning via augmented gaussian processes and multi-information source optimization, 2020b.

Antonio Candelieri, Riccardo Perego, Ilaria Giordani, Andrea Ponti, and Francesco Archetti. Modelling human active search in optimizing black-box functions. *Soft Computing*, 24: 17771–17785, 2020c. URL https://doi.org/10.1007/s00500-020-05398-2.

Pierluigi Casale, Oriol Pujol, and Petia Radeva. Personalization and user verification in wearable systems using biometric walking patterns. *Personal and Ubiquitous Computing*, 16(5):563–580, 2012.

Anirban Chaudhuri, Alexandre N Marques, Remi Lam, and Karen E Willcox. Reusing information for multifidelity active learning in reliability-based design optimization. In *AIAA Scitech 2019 Forum*, page 1222, 2019.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016.

KI Christopher, Matthias Seeger, et al. Using the nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems*, 13:682–688, 2001.

Andrea Coraddu, Luca Oneto, Aessandro Ghio, Stefano Savio, Davide Anguita, and Massimo Figari. Machine learning approaches for improving condition-based maintenance of naval propulsion plants. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 230(1):136–153, 2016.

George De Ath, Richard M Everson, Alma AM Rahat, and Jonathan E Fieldsend. Greed is good: Exploration and exploitation trade-offs in bayesian optimisation. *arXiv preprint arXiv:1911.12809*, 2019.

George De Ath, Richard M Everson, Jonathan E Fieldsend, and Alma AM Rahat. ε-shotgun: ε-greedy Batch Bayesian Optimisation. *arXiv preprint arXiv:2002.01873*, 2020.

Stefano De Blasi and Alexander Gepperth. SASBO: Self-Adapting Safe Bayesian Optimization. 2020.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Laurence Charles Ward Dixon and Giorgio P Szegö. *Towards global optimisation*, volume 2. North-Holland Amsterdam, 1978.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. DPP-Net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 517–531, 2018.

Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR, 2013.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *International Conference on Learning Representations*, 2018.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21, 2019.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 1436–1445, July 2018.

Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. In *Advances in Neural Information Processing Systems*, pages 4977–4989, 2019.

Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In *Portuguese Conference on Artificial Intelligence*, pages 535–546. Springer, 2015.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074*, 2020.

Adrian Cătălin Florea and Răzvan Andonie. A dynamic early stopping criterion for random search in svm hyperparameter optimization. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 168–180. Springer, 2018.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62, 2008.

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577, 2018.

Peter Frazier, Warren Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for bayesian optimization. In *Artificial Intelligence and Statistics*, pages 1311–1319, 2017.

Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes. *Neurocomputing*, 380:20–35, 2020.

Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 250–259, 2014.

Victor Gergel, Konstantin Barkalov, Ilya Lebedev, Maria Rachinskaya, and Alexander Sysoyev. A flexible generator of constrained global optimization test problems. *AIP Conference Proceedings*, 2070(1):020009, 2019.

Seyede Fatemeh Ghoreishi and Douglas Allaire. Multi-information source constrained bayesian optimization. *Structural and Multidisciplinary Optimization*, 59 (3):977–991, 2019.

David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. volume 1 of *Foundations of Genetic Algorithms*, pages 69 – 93. Elsevier, 1991.

Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Sebastian Pölsterl, and Alexander Cavallaro. 2d image registration in ct images using radial image descriptors. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 607–614. Springer, 2011.

Robert B Gramacy. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. CRC Press, 2020.

Robert B Gramacy, Genetha A Gray, Sébastien Le Digabel, Herbert KH Lee, Pritam Ranjan, Garth Wells, and Stefan M Wild. Modeling an Augmented Lagrangian for Blackbox Constrained Optimization. *Technometrics*, 58(1):1–11, 2016.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design. *arXiv preprint arXiv:1709.05501*, 2017.

P Richard Hahn, Jingyu He, and Hedibert F Lopes. Efficient sampling for gaussian linear regression with arbitrary priors. *Journal of Computational and Graphical Statistics*, 28(1):142–154, 2019.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. 2009.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1):1–18, 2003.

Ali Hebbal, Loic Brevault, Mathieu Balesdent, El-Ghazali Talbi, and Nouredine Melab. Bayesian optimization using deep gaussian processes. *arXiv preprint arXiv:1905.03350*, 2019.

Philipp Hennig. Fast probabilistic optimization from noisy gradients. In *International conference on machine learning*, pages 62–70, 2013.

Philipp Hennig and Martin Kiefel. Quasi-newton method: A new direction. *Journal of Machine Learning Research*, 14(Mar):843–865, 2013.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research*, 13(1):1809–1837, 2012.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

José Miguel Hernández-Lobato, Michael A Gelbart, Ryan P Adams, Matthew W Hoffman, and Zoubin Ghahramani. A general framework for constrained bayesian optimization using information-based search. *The Journal of Machine Learning Research*, 17(1):5549–5601, 2016.

TK Ho. Random decision forest, 3rd international conference on document analysis and recognition, 1995.

JH Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. pages 99–103, 1975.

Hao Huang and Zelda B Zabinsky. Adaptive probabilistic branch and bound with confidence intervals for level set approximation. In *2013 Winter Simulations Conference (WSC)*, pages 980–991. IEEE, 2013.

DE Huntington and CS Lyrintzis. Improvements to and limitations of latin hypercube sampling. *Probabilistic engineering mechanics*, 13(4):245–253, 1998.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

Shogo Iwazaki, Yu Inatsu, and Ichiro Takeuchi. Mean-variance analysis in bayesian optimization under uncertainty. *arXiv preprint arXiv:2009.08166*, 2020.

Ali Jalali, Javad Azimi, Xiaoli Fern, and Ruofei Zhang. A lipschitz exploration-exploitation scheme for bayesian optimization. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 210–224. Springer Berlin Heidelberg, 2013.

Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

Rodolphe Jenatton, Cedric Archambeau, Javier González, and Matthias Seeger. Bayesian optimization with tree-structured dependencies. In *International Conference on Machine Learning*, pages 1655–1664, 2017.

Gaoxia Jiang and Wenjian Wang. Error estimation based on variance analysis of k-fold cross-validation. *Pattern Recognition*, 69:94–106, 2017.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181, 1993.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International conference on machine learning*, pages 295–304, 2015.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in neural information processing systems*, pages 2016–2025, 2018.

Kirthevasan Kandasamy, Gautam Dasarathy, Junier Oliva, Jeff Schneider, and Barnabas Poczos. Multi-fidelity gaussian process bandit optimisation. *Journal of Artificial Intelligence Research*, 66:151–196, 2019.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39 (3):459–471, 2007.

James Kennedy and Russell Eberhart. Particle Swarm Optimization. In *International Conference on Neural Networks (ICNN)*, volume 4, pages 1942–1948, 1995.

Marc C. Kennedy and Anthony O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, 2017.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9, page 50. Citeseer, 2014.

Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. In *Joint Automatic Control Conference*, number 1, pages 69–79, 1963.

Rémi Lam, Douglas L Allaire, and Karen E Willcox. Multifidelity optimization using statistical surrogate modeling for non-hierarchical information sources. In *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0143, 2015.

Cheng Li, Sunil Gupta, Santu Rana, Vu Nguyen, Svetha Venkatesh, and Alistair Shilton. High dimensional bayesian optimization using dropout. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2096–2102, 2017a.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017b.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a.

Jeremiah Liu, John Paisley, Marianthi-Anna Kioumourtzoglou, and Brent Coull. Accurate uncertainty estimation and decomposition in ensemble learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 8952–8963, 2019b.

Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Fabio Ferreira, Isabelle Guyon, Sirui Hong, Frank Hutter, Rongrong Ji, Julio Jacques Junior, Lindauer Marius, Luo Zhipeng, Madadi Meysam, Nierhoff Thomas, Niu Kangning, Pan Chunguang, Stoll Danny, Treguer

Sebastien, Wang Jin, Wang Peng, Wu Chenglin, Xiong Youcheng, Zela Arbër, and Zhang Yang. Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019. 2020.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 1540–1552. PMLR, 2020.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Andrew March and Karen Willcox. Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives. *AIAA journal*, 50(5):1079–1089, 2012.

Alexandre Marques, Remi Lam, and Karen Willcox. Contour location via entropy reduction leveraging multiple information sources. In *Advances in neural information processing systems*, pages 5217–5227, 2018.

Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Maximilian Dippel, Marius Lindauer, and Frank Hutter. Towards automatically-tuned deep neural networks. In *Automated Machine Learning*, pages 135–149. Springer, Cham, 2019.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. pages 293–312, 2019.

Sudhanshu K Mishra. Some new test functions for global optimization and performance of repulsive particle swarm method. *Available at SSRN 926132*, 2006.

Tom M Mitchell. Machine learning, chapter decision tree learning. *McGraw-Hill*, 2:10121–0101, 1997.

Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

Mojmir Mutny and Andreas Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In *Advances in Neural Information Processing Systems*, pages 9005–9016, 2018.

Yuri Nesterov. A method for solving a convex programming problem with convergence rate O(1/K2). *Soviet Mathematics Doklady*, 27:372–367, 1983.

Tien-Dung Nguyen, Tomasz Maszczyk, Katarzyna Musial, Marc-André Zöller, and Bogdan Gabrys. Avatar - machine learning pipeline evaluation using surrogate model. In Michael R. Berthold, Ad Feelders, and Georg Krempl, editors, *Advances in Intelligent Data Analysis XVIII*, pages 352–365. Springer International Publishing, 2020. ISBN 978-3-030-44584-3.

Vu Nguyen and Michael A Osborne. Knowing the what but not the where in bayesian optimization. In *International Conference on Machine Learning*, pages 7317–7326. PMLR, 2020.

Vu Nguyen, Sunil Gupta, Santu Rane, Cheng Li, and Svetha Venkatesh. Bayesian optimization in weakly specified search space. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 347–356. IEEE, 2017.

Marco S Nobile, Gabriella Pasi, Paolo Cazzaniga, Daniela Besozzi, Riccardo Colombo, and Giancarlo Mauri. Proactive particles in swarm optimization: a self-tuning algorithm based on fuzzy logic. In *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2015.

Marco S Nobile, Paolo Cazzaniga, Daniela Besozzi, Riccardo Colombo, Giancarlo Mauri, and Gabriella Pasi. Fuzzy self-tuning pso: A settings-free algorithm for global optimization. *Swarm and evolutionary computation*, 39:70–85, 2018.

Favour M Nyikosa, Michael A Osborne, and Stephen J Roberts. Bayesian optimization for dynamic problems. *arXiv preprint arXiv:1803.03432*, 2018.

Randal S. Olson and Jason H. Moore. *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, pages 151–160. Springer International Publishing, 2019.

Nitin R Patel, Robert L Smith, and Zelda B Zabinsky. Pure adaptive search in monte carlo optimization. *Mathematical programming*, 43(1-3):317–328, 1989.

Danilo Pau, Bipin P Kumar, Prashant Namekar, Gauri Dhande, and Luca Simonetta. Dataset of sodium chloride sterile liquid in bottles for intravenous administration and fill level monitoring. *Data in Brief*, page 106472, 2020.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12: 2825–2830, 2011.

Benjamin Peherstorfer, Boris Kramer, and Karen Willcox. Combining multiple surrogate models to accelerate failure probability estimation with expensive high-fidelity models. *Journal of Computational Physics*, 341:61 – 75, 2017.

Maria Peifer, Luiz FO Chamon, Santiago Paternain, and Alejandro Ribeiro. Sparse multiresolution representations with adaptive kernels. *IEEE Transactions on Signal Processing*, 68: 2031–2044, 2020.

Andrey Pepelyshev, Anatoly Zhigljavsky, and Antanas Žilinskas. Performance of global random search algorithms for large dimensions. *Journal of Global Optimization*, 71(1):57–71, 2018.

Riccardo Perego, Antonio Candelieri, Francesco Archetti, and Danilo Pau. Tuning deep neural network's hyperparameters constrained to deployability on tiny systems. In *International Conference on Artificial Neural Networks*, pages 92–103. Springer, 2020.

Valerio Perrone, Huibin Shen, Matthias W Seeger, Cédric Archambeau, and Rodolphe Jenatton. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pages 12771–12781, 2019.

Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, 2013.

Cristiano G Pimenta, Alex GC de Sá, Gabriela Ochoa, and Gisele L Pappa. Fitness landscape analysis of automated machine learning search spaces. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 114–130. Springer, 2020.

SA Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12(4):57–67, 1972.

Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. *Advances in neural information processing systems*, 30:4288–4298, 2017.

Yasha Pushak and Holger Hoos. Algorithm configuration landscapes. In *International Conference on Parallel Problem Solving from Nature*, pages 271–283. Springer, 2018.

Hong Qian, Yi-Qi Hu, and Yang Yu. Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings. In *IJCAI*, pages 1946–1952, 2016.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20:1177–1184, 2007.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 1313–1320, 2009.

PS Rana. Physicochemical properties of protein tertiary structure data set. *UCI Machine Learning Repository*, 2013.

Carl Edward Rasmussen and Christopher KI Williams. Gaussian process for machine learning. adaptive computation and machine learning, 2006.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, 2019.

Binxin Ru, Ahsan Alvi, Vu Nguyen, Michael A Osborne, and Stephen Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning*, pages 8276–8285. PMLR, 2020.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11:1, 2018.

Craig Saunders, Mark O Stitson, Jason Weston, Leon Bottou, and A Smola. Support vector machine-reference manual. 1998.

Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series, 2018.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In Wulfram Gerstner, Alain Germond, Martin Hasler, and Jean-Daniel Nicoud, editors, *Artificial Neural Networks — ICANN'97*, pages 583–588, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69620-9.

Eric Schulz, Maarten Speekenbrink, José Miguel Hernández-Lobato, Zoubin Ghahramani, and Samuel J Gershman. Quantifying mismatch in bayesian optimization. In *Nips workshop on bayesian optimization: Black-box optimization and beyond*, 2016.

Eric Schulz, Maarten Speekenbrink, and Andreas Krause. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

Rajat Sen, Kirthevasan Kandasamy, and Sanjay Shakkottai. Multi-fidelity black-box optimization with hierarchical partitions. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4538–4547. PMLR, 2018.

J Senthilnath, SN Omkar, and V Mani. Clustering using firefly algorithm: performance study. *Swarm and Evolutionary Computation*, 1(3):164–171, 2011.

Yaroslav D Sergeyev and Dmitri E Kvasov. *Deterministic global optimization: An introduction to the diagonal approach*. Springer, 2017.

Yaroslav D Sergeyev, Dmitri E Kvasov, and Falah MH Khalaf. A one-dimensional local tuning algorithm for solving go problems with partially defined constraints. *Optimization Letters*, 1 (1):85–99, 2007.

Yaroslav D Sergeyev, Dmitri E Kvasov, and Marat S Mukhametzhanov. Emmental-type gkls-based multiextremal smooth test problems with non-linear constraints. In *International Conference on Learning and Intelligent Optimization*, pages 383–388. Springer, 2017.

Yaroslav D Sergeyev, Dmitri E Kvasov, and Marat S Mukhametzhanov. On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. *Scientific reports*, 8(1):1–9, 2018.

Yaroslav D Sergeyev, Antonio Candelieri, Dmitri E Kvasov, and Riccardo Perego. Safe global optimization of expensive noisy black-box functions in the δ-lipschitz framework. *Soft Computing*, pages 1–21, 2020.

Yuhui Shi and Russell C Eberhart. Fuzzy adaptive particle swarm optimization. In *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 101–106. IEEE, 2001.

Alistair Shilton, Sunil Gupta, Santu Rana, Pratibha Vellanki, Cheng Li, Laurence Park, Svetha Venkatesh, Alessandra Sutti, David Rubin, Thomas Dorin, et al. Covariance function pre-training with m-kernels for accelerated bayesian optimisation. *arXiv preprint arXiv:1802.05370*, 2018.

Petru-Aurelian Simionescu and David G Beale. New concepts in graphic visualization of objective functions. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 36223, pages 891–897, 2002.

Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

AJ Smola and B Schölkopf. A tutorial on support vector regression, statist. *Comput*, 14:199–222, 2004.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25:2951–2959, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, volume 29, pages 4134–4142, 2016.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, number CONF. Omnipress, 2010.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, 2012.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Roman G Strongin. Numerical methods in multiextremal problems. *Moscow, USSR: Nauka*, 1978.

Roman G Strongin. Algorithms for multi-extremal mathematical programming problems employing the set of joint space-filling curves. *Journal of Global Optimization*, 2(4):357–378, 1992.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 497–504, 2017.

Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. Safe exploration for optimization with gaussian processes. In *International Conference on Machine Learning*, pages 997–1005. PMLR, 2015.

Yanan Sui, Vincent Zhuang, Joel W Burdick, and Yisong Yue. Stagewise safe bayesian optimization with gaussian processes. *Proceedings of Machine Learning Research*, 80:4781–4789, 2018.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in neural information processing systems*, 26:2004–2012, 2013.

William D Thomison and Douglas L Allaire. A model reification approach to fusing information from multifidelity information sources. In *19th AIAA non-deterministic approaches conference*, page 1949, 2017.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

Dong-ping Tian and Nai-qian Li. Fuzzy particle swarm optimization algorithm. In *2009 International Joint Conference on Artificial Intelligence*, pages 263–267. IEEE, 2009.

YL Tong. Fundamental properties and sampling distributions of the multivariate normal distribution. In *The Multivariate Normal Distribution*, pages 23–61. Springer, 1990.

Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P Avariento, Tomás J Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In *2014 international conference on indoor positioning and indoor navigation (IPIN)*, pages 261–270. IEEE, 2014.

Yi-An Tsai, Giulia Pedrielli, Logan Mathesen, Zelda B Zabinsky, Hao Huang, Antonio Candelieri, and Riccardo Perego. Stochastic optimization for feasibility determination: an application to water pump operation in water distribution network. In *2018 Winter Simulation Conference (WSC)*, pages 1945–1956. IEEE, 2018.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

Remi Varenne, Jean Michel Delorme, Emanuele Plebani, Danilo Pau, and Valeria Tomaselli. Intelligent recognition of tcp intrusions for embedded micro-controllers. In *International Conference on Image Analysis and Processing*, pages 361–373. Springer, 2019.

Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44 (4):509, 2009.

Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *International Conference on Learning Representations*, 2019.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635, 2017.

Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784, 2013.

Tao Wei, Changhu Wang, and Chang Wen Chen. Modularized morphing of neural networks. *arXiv preprint arXiv:1701.03281*, 2017.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2020.

Adrian G Wills and Thomas B Schön. On the construction of probabilistic newton-type algorithms. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 6499–6504. IEEE, 2017.

James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 31, pages 9884–9895, 2018.

James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from Gaussian process posteriors. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 10292–10302. PMLR, 2020.

Robert L Winkler. Combining probability distributions from dependent information sources. *Management Science*, 27(4):479–488, 1981.

Anqi Wu, Mikio C Aoi, and Jonathan W Pillow. Exploiting gradients and hessians in bayesian optimization and bayesian quadrature. *arXiv preprint arXiv:1704.00060*, 2017a.

Jian Wu and Peter I Frazier. The parallel knowledge gradient method for batch bayesian optimization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3134–3142, 2016.

Jian Wu and Peter I Frazier. Discretization-free knowledge gradient methods for bayesian optimization. *arXiv*, pages arXiv–1707, 2017.

Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5267–5278, 2017b.

Lingxi Xie and Alan Yuille. Genetic CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1388–1397, 2017.

Liang Yan, Xiaojun Duan, Bowen Liu, and Jin Xu. Bayesian optimization based on k-optimality. *Entropy*, 20(8):594, 2018.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

Xin-She Yang. Nature-inspired metaheuristic algorithms. 2008.

Xin-She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009.

Zelda B Zabinsky and Hao Huang. A partition-based optimization approach for level set approximation: Probabilistic branch and bound. In *Women in Industrial and Systems Engineering*, pages 113–155. Springer, 2020.

Zelda B Zabinsky and Robert L Smith. Pure adaptive search in global optimization. *Mathematical Programming*, 53(1-3):323–338, 1992.

Zelda B Zabinsky, Wei Wang, Yanto Prasetio, Archis Ghate, and Joyce W Yen. Adaptive probabilistic branch and bound for level set approximation. In *Proceedings of the Winter Simulation Conference*, pages 4151–4162, 2011.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Anatoly A Zhigljavsky. Mathematical theory of global random search. *LGU, Leningrad*, 1985.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

Ryszard Zieliński. A statistical estimate of the structure of multi-extremal problems. *Mathematical Programming*, 21(1):348–356, 1981.

Antanas Žilinskas and James Calvin. Bi-objective decision making in global optimization based on statistical models. *Journal of Global Optimization*, 74(4):599–609, 2019.

Antanas Žilinskas, Jonathan Gillard, Megan Scammell, and Anatoly Zhigljavsky. Multistart with early termination of descents. *Journal of Global Optimization*, pages 1–16, 2019.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *arXiv preprint arXiv:2006.13799*, 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.