CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

HBRKGA: A POPULATION-BASED HYBRID APPROACH TO HYPERPARAMETER OPTIMIZATION FOR NEURAL NETWORKS

Marcello Alberto Soares Serqueira

Dissertation submitted to the Graduate Program of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Advisor: Eduardo Bezerra, D.Sc.
Co-advisor: Pedro González, D.Sc.

Rio de Janeiro,
November 2020

HBRKGA: A POPULATION-BASED HYBRID APPROACH TO HYPERPARAMETER OPTIMIZATION FOR NEURAL NETWORKS

Dissertation submitted to the Graduate Program of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Marcello Alberto Soares Serqueira

Examining jury:

Presidente, Eduardo Bezerra, D.Sc., CEFET/RJ

Coorientador, Pedro González, D.Sc., CEFET/RJ

Prof. Diego Brandão, D.Sc., CEFET/RJ

Prof. Igor Machado, D.Sc., IC/UFF

Rio de Janeiro,

November 2020

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

S486 Serqueira, Marcello Alberto Soares
Hbrkga: a population-based hybrid approach to hyperparameter optimization for neural networks / Marcello Alberto Soares Serqueira — 2020.
51f : il. color. , enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca , 2020.
Bibliografia : f. 47-51
Orientador: Eduardo Bezerra
Coorientador: Pedro Gonzalez

1. Aprendizado de máquina. 2. Algoritmos genéticos. 3. Estimativa de parâmetros. 4. Redes neurais. I. Bezerra, Eduardo (Orient.). II. Gonzalez, Pedro (Coorient.). III. Título.

CDD 006.31

Elaborada pela bibliotecária Tania Mello – CRB/7 nº 5507/04

DEDICATION

This work is dedicated to my family, my girlfriend, my friends and my teachers who teach me so much over the years studying at CEFET/RJ.

ACKNOWLEDGMENTS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Agradece-se ao CNPq pelo financiamento parcial desta pesquisa.

Agradece-se também as contribuições dos Professores Eduardo Bezerra e Pedro González.

RESUMO

Nos últimos anos, grandes quantidades de dados estão sendo geradas e a necessidade de recursos computacionais continua crescendo. Este cenário levou a um ressurgimento do interesse em redes neurais artificiais. Um dos principais desafios no treinamento de modelos eficazes de redes neurais é encontrar uma boa combinação de hiperparâmetros a serem usados. De fato, a escolha de uma abordagem adequada para pesquisar o espaço do hiperparâmetro influencia diretamente a precisão do modelo resultante da rede neural. Abordagens comuns para busca de hiperparâmetros são a Busca em Grade, a Busca Aleatória e Busca por Otimização Bayesiana. Existem também métodos baseados em população, como a CMA-ES. Neste projeto, apresentamos o HBRKGA, uma nova abordagem baseada na população para a optimização de hiperparâmetros. O HBRKGA é uma abordagem híbrida que combina o Algoritmo Genético de Chaves Aleatórias Viciadas com uma técnica de Random-Walk para pesquisar o espaço de hiperparâmetros de forma eficiente. Foram realizados vários experimentos computacionais em oito conjuntos de dados diferentes para avaliar a eficácia da abordagem proposta. Os resultados mostraram que o HBRKGA conseguiu encontrar configurações de hiperparâmetros que superaram (em termos de qualidade preditiva) os métodos de base em seis dos oito conjuntos de dados, mostrando também tempo de execução razoável.

Palavras-chave: Aprendizado de Máquina; Otimização de Hiperparâmetros; Algoritmos Genéticos; Redes Neurais

ABSTRACT

In recent years, large amounts of data have been generated, and computer power has kept growing. This scenario has led to a resurgence in the interest in artificial neural networks. One of the main challenges in training effective neural network models is finding the right combination of hyperparameters to be used. Indeed, the choice of an adequate approach to search the hyperparameter space directly influences the accuracy of the resulting neural network model. Common approaches for hyperparameter optimization are Grid Search, Random Search, and Bayesian Optimization. There are also population-based methods such as CMA-ES. In this paper, we present HBRKGA, a new population-based approach for hyperparameter optimization. HBRKGA is a hybrid approach that combines the Biased Random Key Genetic Algorithm with a Random Walk technique to search the hyperparameter space efficiently. Several computational experiments on eight different datasets were performed to assess the effectiveness of the proposed approach. Results showed that HBRKGA could find hyperparameter configurations that outperformed (in terms of predictive quality) the baseline methods in six out of eight datasets while showing a reasonable execution time.

Keywords: Machine Learning; Hyperparameter Optimization; Genetic Algorithms; Neural Networks

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
BRKGA	Biased Random Key Genetic Algorithm
GD	Gradient Descent
MLP	Multilayer Perceptron
MNIST	Modified National Institute Of Standards And Technology
MSE	Mean Square Error
PCA	Principal Components Analysis
RELU	Rectified Linear Unit
RKGA	Random-key Genetic Algorithms
SGD	Stochastic Gradient Descent
TANH	Hyperbolic Tangent

CONTENTS

Chapter I Introduction

I.1- Contextualization

Artificial Neural Networks (ANN) are an old approach to Machine Learning that have witnessed a renewed interest both from industry and academia in recent years [Goodfellow et al., 2016a; LeCun et al., 2015]. This interest is motivated by cases of success in several application domains, such as audio recognition, image recognition, and language translation. A particular advancement in the field of ANN in the last decade is related to the fact that the research community has been gradually learning to deal with the engineering problem of training neural networks comprised of several hidden layers. This renaissance of neural networks has been called Deep Learning [LeCun et al., 2015].

A prerequisite to training a neural network model is to come up with a particular combination of values of hyperparameters, such as the number of hidden layers, the number of artificial neurons in each, the learning rate, the activation functions to be used, to name a few. Only after a particular set of hyperparameters has been chosen can the training process tune the parameters (i.e., the weights) of the ANN. Many hyperparameters have continuous domains, which accelerate the exponential growth of possible values combination. The huge multidimensional space resulting from combinations of several hyperparameters is an even more significant challenge in the Deep Learning era. So much so that the area of automatic machine learning (AutoML) has emerged to study automation and optimization of machine learning models. AutoML aims to join and to automate the whole process of machine learning (hyperparameter tuning, architectures, optimization algorithms) for the creation of accurate models without the need for deep statistical knowledge and programming [He et al., 2019].

A critical aspect of AutoML is hyperparameter tuning. There are several techniques to perform a search in the hyperparameter space. In general, these techniques work as a procedure that runs an outer loop in the learning process: this procedure suggests a combination of hyperparameter values, which are then used to optimize the set of parameters in the neural network. These techniques usually do not present any assumption

for performing the search; only a fixed range of values is defined by the user to be explored.

Two popular approaches to tuning hyperparameters are Grid Search and Random Search. Bergstra and Bengio [2012] performed Random Search experiments in comparison to the results of the experiments obtained by Larochelle et al. [2007]. They showed that in most datasets, Random Search was able to overcome Grid Search, both in accuracy and in computational performance. Thus far, Random Search has shown to be an efficient alternative to the Grid Search technique.

One downside of both Grid Search and Random Search is that they do not try to improve based on previously tested hyperparameters combinations. Hence, in recent years more intelligent methods have been explored to perform this optimization. Among the most covered is Bayesian Optimization [Snoek et al., 2012]. The method is different from Random and Grid Search because it allows for ANN's optimization without the need to define the search space manually with high precision. Unfortunately, Bayesian Optimization is computationally expensive, since its time complexity is cubic on the number of samples seen before [Snoek et al., 2015].

A recent alternative to hyperparameter tuning is the family of population-based methods [Simon, 2013]. These are evolutionary algorithms that aim to evolve individuals in a hyperparameter configuration population by applying operations such as crossover and mutation [Hutter et al., 2019]. Loshchilov and Hutter [2016] applied the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) with Bayesian Optimization, and some of its variations, and achieved comparable results with Bayesian Optimization, and also with a lower computational cost.

I.2- Motivation

The problem of hyperparameters optimization of machine learning algorithms has become very important in recent years [LeCun et al., 1998; Larochelle et al., 2007; Bergstra and Bengio, 2012]. The demand for accurate automated methods grows as the amount of data increases. Studies with the objective of increasing the efficiency of these algorithms are fundamental to automate machine learning process.

Several factors are addressed in a predictive model generation process: pre-

processing, hyperparameters optimization, experimental evaluation, etc. One of the main needs nowadays is to automate this process in such a way that the scientist or user gives as few manual inputs as possible to generate this model. A hyperparameter optimization algorithm manages most of these current needs.

I.3- Objectives

In this work, we propose a new population-based approach to searching in ANN's hyperparameter space. As a starting point to our approach, we use a well-known population-based optimization algorithm, Biased Random Key Genetic Algorithm (BRKGA) [Martinez et al., 2011]. We devise a hybrid algorithm called HBRKGA by adding a Random-Walk procedure to refine the candidate solution generated by BRKGA. In the experiments, we compared HBRKGA to several other optimization strategies, namely, Grid Search, Random Search, Bayesian Optimization and CMA-ES. We used eight different datasets in our validation experiments. HBRKGA was able to increase the F_1 metric up to 1.2% compared to the others methods in the experiment and has also shown a reasonable execution time.

This work has two contributions. The first is the application of a hybrid population-based strategy with Random-Walk to the hyperparameter optimization problem in MLP ANNs. Also, we created an abstract data type (presented in Section IV.2) to help during the mapping of the values of the domain of hyperparameters (decoded solution) and the keys values of BRKGA (encoded solutions), making it easier to reuse the method for other hyperparameters with another subrange of values.

I.4- Methodology

To support the experimental results of each optimization strategy, a evaluation of performance in different datasets is necessary to provide better support to a conclusion of the results obtained. For this reason, 8 datasets were selected in total (presented in

Section V.1) to give this basis, with some of them being used in important works in the area of hyperparameter optimization in the last years [Larochelle et al., 2007; Bergstra and Bengio, 2012]. The datasets were divided into training, validation and test.

With the datasets selected, the process of hyperparameter optimization starts with the appropriate choice of hyperparameters and the values explored in each of them. In order to have an ANN MLP with three hidden layers, some hyperparameters were chosen that are commonly used in literature: the number of first, second and third layer neurons, learning rate and regularization rate. Each one of them has distinct characteristics that directly influence the training for model generation.

We performed an ablation study to verify the behavior of the addition of a new component (Random-Walk) in the BRKGA algorithm. The analysis was performed by comparing the mean and standard deviation of ten executions of HBRKGA(0) (without the Random-Walk component) and HBRKGA(3) (with the Random-Walk component) in one dataset. This study showed that the Random-Walk component was able to improve the result over the algorithm without perturbance in the solutions.

Each search strategy performs the optimization taking into account a specific dataset and the hyperparameters to be optimized. The model is trained with each solution generated by the strategies and applied to the validation set, where the F_1 metric is calculated to measure the predictive quality of this model. This process was executed 10 times to obtain the average and standard deviation of $F1$. The execution average time in seconds (and standard deviation) for the strategies optimization was also calculated to measure the computational performance.

An important evaluation to be done after the experiments for comparison is the use of statistical tests. With this, we can use statistical significance in the analysis. We use Wilcoxon's non-parametric test to calculate the statistical significance taking into account the set of 10 runs of each method compared to HBRKGA to have a basis for the conclusion in the several datasets studied.

I.5- Chapters Organization

The rest of this work is organized as follow. Chapter II presents the concepts and fundamentals of Artificial Neural Networks, definitions of hyperparameters and the optimization methods. Chapter III expose some releated works with hyperparameter optimization methods. Chapter IV shows the proposed algorithm HBRKGA. Chapter V presents the discussions of the results obtained during the experiments. Finally, Chapter VI presents the conclusions.

Chapter II Background

This chapter approaches the fundamentals and concepts used in this work. Section II.1 describes some strategies used for the hyperparameter optimization problem. Section II.2 presents BRKGA which is used as a starting point for defining the proposed strategy. Section II.3 details the concepts of Multilayer Perceptron (MLP) Neural Network that was the architecture used on the experiments.

II.1- Hyperparameter Optimization Strategies

One of the main objectives of a learning algorithm is to find a function that can reduce the Artificial Neural Network (ANN) error rate. This function will be generated according to a set of parameters [Bergstra and Bengio, 2012]. The parameters are directly linked to the learning of weights in the *training*. However, a Neural Network has other features that will influence the algorithm's learning ability. These features are called hyperparameters. Some examples of hyperparameters are number of neurons per layer, learning rate, number of layers, etc.

Bengio [2012] defines a hyperparameter of a learning algorithm as a variable set prior to the application of this algorithm to the data and it's not being directly selected by learning. The hyperparameters are directly linked to the performance of the algorithm in model generation. Due to this fact, it's optimization is fundamental to obtain good results. Hyperparameters optimization is a problem of optimizing a loss function on Neural Network [Bergstra et al., 2011]. To reduce this loss, each value of hyperparameter should be stipulated in the *validation* and *training* steps.

In most cases, there are no basic theories regarding the value that should be indicated to the hyperparameter. This is due to the fact that different sets of data require different ranges of exploitation in these values of hyperparameters [Bergstra and Bengio, 2012; Bergstra et al., 2011]. The exploration of these bands used to be done in such a way that the scientist himself indicated a value of hyperparameter and on trial and error

set the values manually. This technique is known as Manual Search [LeCun et al., 1998].

The manual search becomes exhausting and demands skilled scientist time to define the value of the hyperparameters. Usually the choices of these values are made by trial and error. In order to automate this process several search algorithms are being studied, such as the Grid Search and the Random Search. These methods will be presented on Section II.1.

In order to optimize the hyperparameters of Neural Networks, currently a vast amount of trial and error are currently required to obtain acceptable values for each of them. Methods that aim to automate the processes are presenting themselves as essential for the generation of a good model. The next subsections presents some of these methods: Grid Search (Section II.1.1), Random Search (Section II.1.2), Bayesian Optimization (Section II.1.3), BRKGA (Section II.2) and CMA-ES (Subsection II.1.4).

II.1.1 Grid Search

Grid Search performs a search by considering a multi-dimensional grid of hyperparameter combinations. The ranges for each hyperparameter in the grid are usually user-defined. Grid Search then computes a Cartesian product corresponding to the possible hyperparameter combinations [Bergstra and Bengio, 2012]. Since there may be some hyperparameters that can assume infinitely many values, the user must also define a step used to jump from one hyperparameter value to another. When done by a specialist in the domain in question, these sampled values can result in satisfactory learning models.

This technique has a trivial implementation and is easy to parallelize: each combination of hyperparameter values can be tested in parallel. However, the amount of hyperparameter combinations grows exponentially with the number of hyperparameters [Bellman, 1961; Bergstra and Bengio, 2012]. As a result, Grid Search may exploit many unimportant areas if the input grid is not carefully designed by a domain expert. This problem causes a waste of computational resources since there is no rule to explore the hyperparameters space. Bergstra and Bengio [2012] show that different datasets will have their particularity when performing hyperparameter optimization, therefore different spaces should be explored. Figure 1 shows the Grid Search process until finding the best

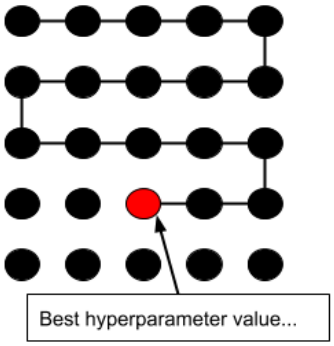


Figure 1 – The Grid layout for hyperparameter optimization problem. Each point represent a solution of a combination of hyperparameters that is executed sequentialy. This strategy finds the best solution (red point) after running a sequential path.

hyperparameter solution in red.

II.1.2 Random Search

Random Search takes as input a bound hyperparameter subspace. The bounds for such subspace are also user-provided. Then it takes random samples from this bounded domain, which are used as the hyperparameter combinations to be tested [Zabinsky, 2009]. It helps to get solutions to large-scale problems instead of using combinatory optimization like Grid Search, to avoid exploring less relevant areas in the hyperparameter subspace.

Compared to Grid Search, Random Search proves to be more efficient in the sense that it does not combine all selected values contained in a user-defined grid to perform hyperparameter optimization. Instead, it randomly explores regions of the hyperparameter subspace that could have a better relevance in the hyperparameter space. This behavior makes it less computationally costly than Grid Search [Bergstra and Bengio, 2012]. Figure 2 shows the Random Search process until finding the best hyperparameter solution in red.

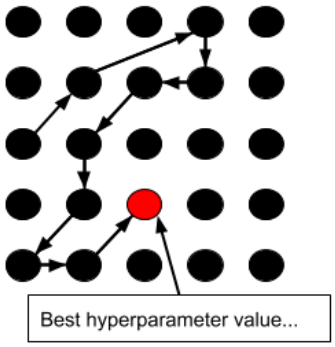


Figure 2 – The Random layout for hyperparameter optimization problem. Each point represents a solution with random hyperparameters values that is randomly generated. This strategy finds the best solution (red point) after some random solutions trials.

II.1.3 Bayesian Optimization

Both Random Search and Grid Search do not have a disciplined basis for optimizing hyperparameters. They perform a search in a trial and error manner. Smarter hyperparameter search algorithms are able to evaluate a decision comparing to prior samples in the hyperparameter space. One of these methods is Bayesian Optimization. This method is able to make assumptions about the hyperparameter space based on prior samples to assist on choosing the next samples [Brochu et al., 2010]. Bayesian optimization is able to construct a probabilistic model for a function $f(x)$ to explore the possible values of this function, using information obtained from previous iterations to generate a search model Snoek et al. [2012].

According to Snoek et al. [2012], there are two components to be defined in Bayesian optimization. The first is the function that will express the assumption about the function to be optimized. Usually the Gaussian Process is used to obtain samples through this function, but there are also other models that can be explored [Dewancker et al.]. The second component is the acquisition function, used to determine the next point to be explored during fits in the Gaussian Process. Some alternatives to this component are Probability of Improvement (most likely point to improve the current best result), Expected Improvement (the point that most improves the current best result of a function to be optimized), and Upper Confidence Interval (maximum or minimum point in the confidence interval during the Gaussian Process). These methods are able to evaluate the best next point taking into consideration the probability of it being the maximum or the minimum.

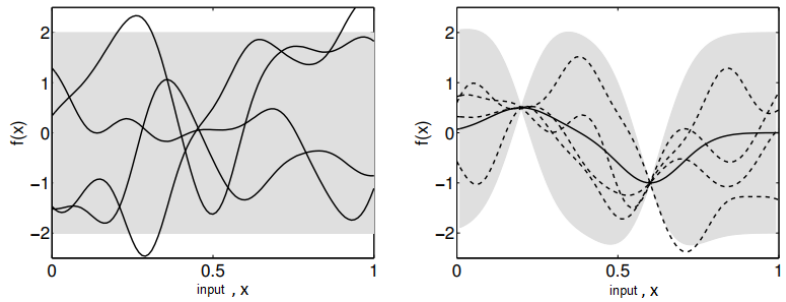


Figure 3 – The prior (left) and posterior (right) function from Gaussian Process. As the points are observed, the Gaussian process is fitted, making it possible for the acquisition functions to evaluate new solutions. Image adapted from [Rasmussen and Williams, 2006].

According to Rasmussen [2004] a Gaussian Process is a generalization of Gaussian Probability Distribution. It generates a non-parametric model of probability to be able to estimate the unknown values of a function. Initially, n multivariate random samples are generated, where each of these is considered a different process.

As the real points of a case study are observed by the Gaussian Process, a posterior function is generated with prior function adjustments. Figure 3 shows this process. The crushed lines on the right are the known points of a given function, making the uncertainty in this area equal to zero. For points that have not yet been observed, the uncertainty increases, as can be seen in the gray areas on the plot. Generally, the prediction function is constructed through the mean of these Processes.

In Multivariate Random Distributions, it is necessary to use a covariance matrix to obtain the distance between points. This concept is the same as the standard deviation in a univariate normal distribution. There are several ways to calculate the distance between these points and to obtain this matrix. In the Gaussian Process, this function is called the kernel [Snoek et al., 2012].

A Gaussian Process requires a specific kernel choice that best fits the data. The evaluation of these functions becomes computationally costly in a high-dimensional, multi-sampled dataset. Some examples of kernels used in the Gaussian Process are Linear, Exponential, Gaussian, Matern, etc. These methods directly affect the distribution of data during the creation of the prior function. Figure 4 shows some samples from kernels used in the Gaussian Process.

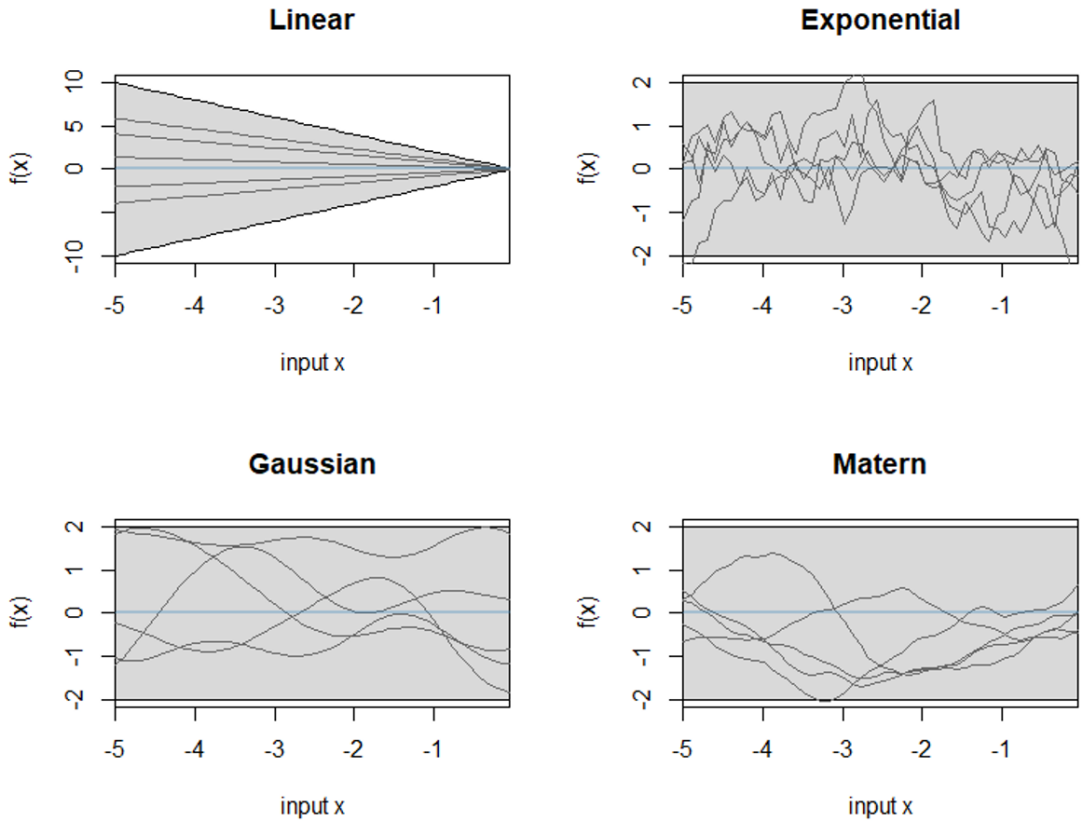


Figure 4 – Kernels examples to Gaussian Process. Image created using the source code `https://bit.ly/2LcSq6x`.

II.1.4 Covariance Matrix Adaptation Evolution Strategy

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a stochastic optimization method belonging to the evolutionary algorithm family [Hansen, 2006]. CMA-ES is a derivative-free strategy used to help in non-linear and non-convex problems [Loshchilov, 2014]. Solutions are generated from normal multivariate sampling distribution, where the covariance matrix and mean are taken into account. CMA-ES proved to be a good alternative to Bayesian Optimization in the hyperparameter optimization problem. It is able to achieve comparable results and with less computational resources [Loshchilov and Hutter, 2016].

During each iteration (generation) of the algorithm, new candidates are generated from the samples from multivariate distribution. According to Hansen [2016] the sample of a CMA-ES generation g with population size λ is done according to Equation (1). $x_k^{(g+1)}$ is

the k-th offspring from generation $g+1$, m^g is the mean of the distribution in the generation g, σ is the step size (perturbance mutation) in generation g and C^g is covariance matrix in generation g. $\mathcal{N}(0, C^g)$ is the normal distribution with zero mean and the standard deviation of the covariance matrix C of the generation g.

$$x_k^{(g+1)} = m^g + \sigma^g \mathcal{N}(0, C^g) \tag{1}$$

The recombination process is done for the adaptation of the covariance matrix C, generating a new value for the mean m^{g+1}. Usually, this change is done through the combination of the best individuals weighted in the current population. m^{g+1} is updated following Equation (2). μ the parent population size (usually the top individuals), w_i is the positive weight coefficients using weighted intermediate recombination, $x_{i:\lambda}^{(g+1)}$ is the i-th best individuals from $g+1$ and c_m is the learning rate. This informations are used to estimate (re-estimate) the covariance matrix through techniques like Rank-μ-Update e Rank-One-Update [Hansen, 2016].

$$m^{g+1} = m^g + c_m \sum_{i=1}^{\mu} w_i (x_{i:\lambda}^{(g+1)} - m^{(g)}) \tag{2}$$

II.2- BRKGA

Genetic algorithms (GA) [Goldberg and Holland, 1988] are methods which simulate the evolution of a population over a certain number of generations. Each individual in a population represents a candidate solution to a given optimization problem. These algorithms apply the concept of survival of the fittest individuals to find good quality solutions to optimization problems.

To escape from local optima and search in the solution space, a population evolves in a number of generations. A solution to the optimization problem is represented by the individuals or chromosomes. The genes encode the solutions in a finite chain of bits or integers. It permits the representation of reproduction between two parents. The fitness criterion in the selection of good individuals is expressed by an objective function.

A new population is generated from the combination of elements belonging to the

current population at each generation of GA. It is performed by three principal operators: reproduction, crossover and mutation. The new population is acquired as follows: i) The next population is sampled from a small percentage of the best individuals in the actual population; ii) crossover applies deterministic or probabilistic operators to randomly selected parents, generating offspring for the next generation; and iii) a random mutation of gene positions is performed to avoid local optima [Whitley, 1994].

Random-key Genetic Algorithms (RKGA) were proposed by Bean [1994]. In this method, vectors of decimal numbers whose values belong to the $[0, 1]$ domain represent the chromosomes. Each vector is given as input to a deterministic algorithm called *decoder*, which associates it with a solution of the optimization problem. The RKGA is an enhancement of the classic Genetic Algorithms, and its main objective is to mitigate GA's operators difficulty in dealing with feasible solutions. Thus, the representation of the problem parameters through random keys allows the development of operators that are problem independent.

Gonçalves and Resende [2011] developed the Biased Random Key Genetic Algorithm (BRKGA) based on RKGA. The main difference between BRKGA and RKGA is the biased way of how parents are chosen in reproduction operator. To BRKGA obtain a new individual, the method combine an individual randomly chosen from an elite set (p_e) of the current population set (p) and another of a non-elite set $(p \setminus p_e)$ of individuals, in which $|p_e| < |p| - |p_e|$. A single individual can be selected more than one time and then can produce more than one offspring.

BRKGA needs $|p_e| < |p| - |p_e|$, the probability of an elite individual being selected for reproduction $(\frac{1}{|p_e|})$ is greater than that of a non-elite individual $(\frac{1}{|p|-|p_e|})$. So, elite individuals have greater probability of passing forward their characteristics to next generations. Moreover, the crossover concept in BRKGA is the Parameterized Uniform Crossing [Spears and De Jong, 1995] with $\mathrm{Pr}_e(i) > 0.5$, where $\mathrm{Pr}_e(i)$ is the probability that the i-th offspring inherits from an elite individual.

The BRKGA has been successfully applied to several optimization problems such as Packaging [Gonçalves, 2007], Routing [Martinez et al., 2011], Traveling Salesman Problem Variants [Snyder and Daskin, 2006; Samanlioglu et al., 2008] and Transmission Network Expansion Planning [Gonzalez and Brandão, 2018]. This work is an opportunity to apply this algorithm to reduce the number of manual inputs in all Neural Networks problems.

II.3- Artificial Neural Networks

The ANN is one of the most popular and studied machine learning algorithms in fields like pattern recognition. It works similarly to the animal nervous system, having many types of layers consisting of processing nodes called neurons. Each layer processes input information and passes it to the next layers until the output.

II.3.1 Perceptron

One of the first ANN models was presented in the 1950s and it's called perceptron. The purpose of the perceptron is to associate relevance weights to a data input with the goal of generating a binary output after a process inside a neuron. Figure 5 shows an example of the perceptron model. The variables x_1, x_2 and x_3 represent the input data, w_1, w_2 and w_3 are the weights and β is the process output.

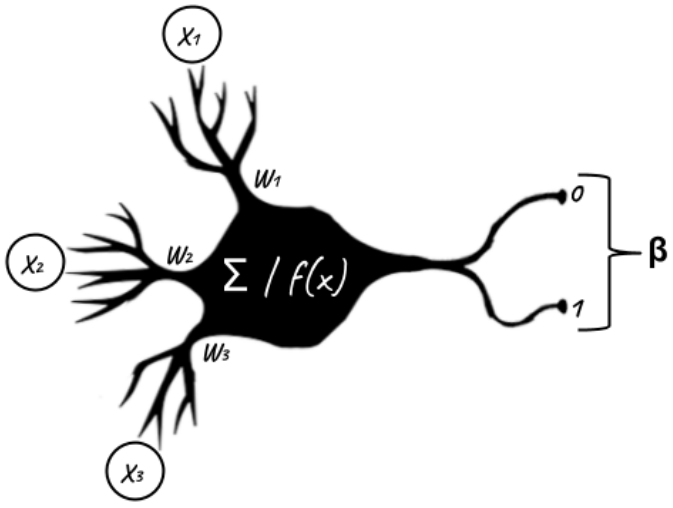


Figure 5 – Perceptron model presented by Rosenblatt [1958].

The β value is calculated by a hyperparameter called activation function that defines the type of processing inside of a neuron. In this binary neuron, the activation function will define the output value taking into account the Equation (3), where $\beta = 1$ case $\alpha \geq 0$ and $\beta = 0$ case $\alpha < 0$. Posteriorly the bias (b) term was added in this equation to

help to get better learning. The bias value helps to shift the activation function output.

$$\alpha = \sum_i x_i p_i \tag{3}$$

Many other activation functions were presented before the idea of perceptron. One of the main is the logistic function or *sigmoid* [Funahashi, 1989]. The *sigmoid* function changes the nature of the neuron used in an ANN. The neuron *sigmoid* don't have a binary β output, it has a value $0 \leq \beta \leq 1$. These output values suggest better learning than the perceptron. The values between 0 and 1 can express how close the classifier is in a binary pattern like A or B. Also, the sigmoid is a non-linear function and allows the Backpropagation algorithm to build classifier models of datasets that aren't linearly separable.

II.3.2 Multilayer Perceptron

Since the perceptron and the activation by a logistic function, many other types of Neural Networks were developed with many other research base. One of them is the architecture MLP that uses many perceptron neurons connected together and divided into 3 types of layers: input, hidden and output [Gardner and Dorling, 1998; Han et al., 2011]. This layers is represented by the Figure 6 where x_1, x_2 and x_3 are the input data and β_1 and β_2 the possibles output for the binary classifier case.

Each layer has a fundamental job in the learning process. The input layer represents the data input that will be passed to the next layer. Commonly the data pass for a preprocessing before being given as input. After this, the data goes to the hidden layer. The hidden layer is composed of a set of neurons that will process the data with the concepts previously presented in the perceptron section and the data passes to the next layer. The passes of information processed by the neuron to the next layers is called Feedforward Network. The hidden layer receives its name because we can't observe the values of each input/output of the neurons [Goodfellow et al., 2016b]. Finally, in the output layer, the final classification is given by taking the processing in the previous layer. As in perceptron, each node has one weight in each connection. With appropriate weights and

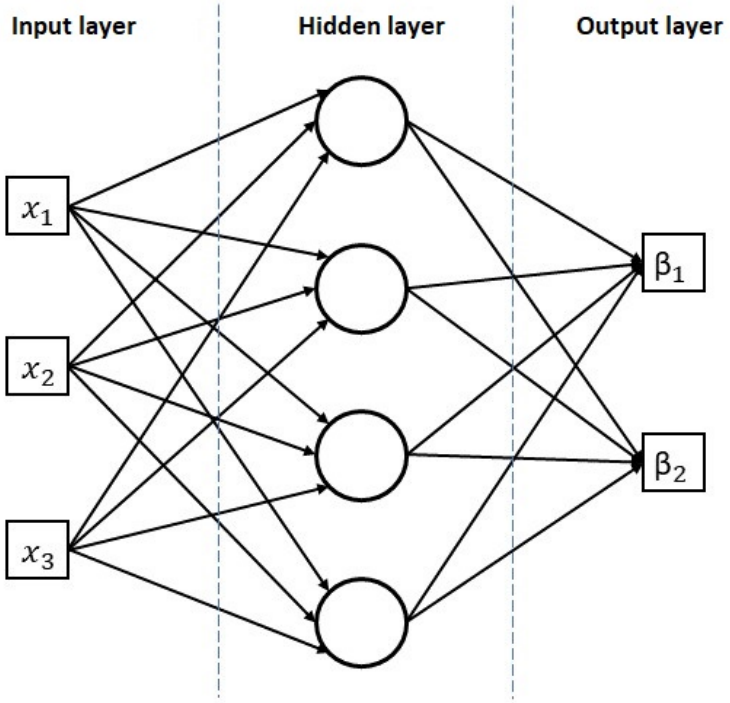


Figure 6 – Example of the basic architeture of MLP.

transfer function, a MLP with just one hidden layer can approximate any function between the input and output [Gardner and Dorling, 1998; Hornik, 1991].

One of the main characteristics of MLP is to generate a model from a given dataset by working on weights adjustments. It's able to learn from examples [Miller, 1993]. This technique can be adjusted to different problems with a model generation in a step called *training*. In the *training* step the input dataset is passed to the algorithm and the bias and weights in each connection are adjusted according to the output of each time the MLP is processed.

When there is a difference between the real input and the predicted output, the error value is computed. This error is used to adjust the weights with the main goal of minimizing the global error rate from the Neural Network [Gardner and Dorling, 1998]. One of the main advantages is the ANN tolerance to noise from the input data. The noise is some kind of error or incomplete data that makes it difficult to classify [Han et al., 2011]. This idea of map the output taking into account the class provided by the input is called supervised learning.

With the weights adjustment problem in *training* step, many solutions were presented to minimize the error rate. One of the most used is the backpropagation [Werbos, 1974; Rumelhart et al., 1988]. The backpropagation algorithm iteratively processes the

data by comparing the input value with the output. The weights are initialized with small random values for each neuron connection and updated in order to reduce an error metric between the real data and predicted. Commonly is used mean square error for this metric. The adjustments are made with the back of the unit processed for the previous layer in an attempt to not let the error propagate. At one point the ANN will no longer be able to significantly reduce the error and will converge and the learning process stops [Han et al., 2011].

The training model is applied to an apart data called *test*. The *test* step is used to evaluate the ANN performance. Other fraction of this set is called *validation* that is responsible to validate the model generated on training step and to adjust the hyperparameters. As an example, the data could be splitted in: *training* 80%, *test* 10% and 10% for the *validation* set. The Figure 7 shows this split applied to a collected data.

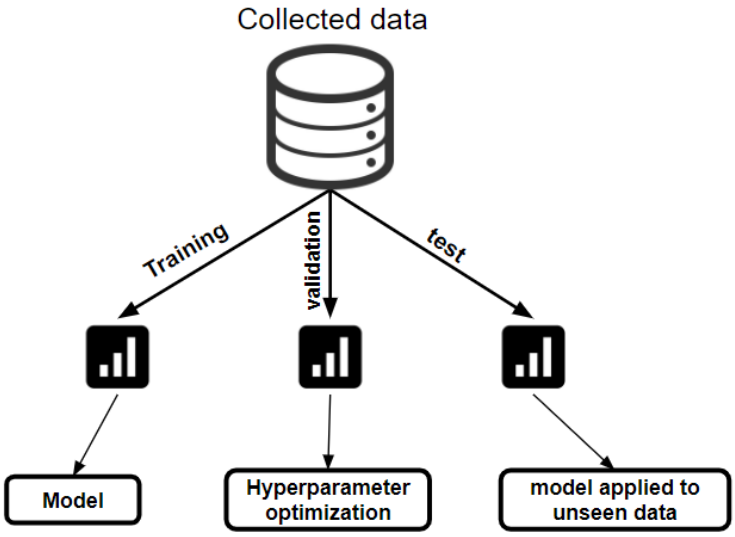


Figure 7 – Dataset splitted into training, validation and test subsets..

II.3.3 Gradient Descent

During the ANN training step, the weights require a constant update after each iteration in the dataset. Each update aims to reduce the error in the values of the weights referring to each input. The most famous way to do that is by using the method Gradient Descent (GD) [Snyman, 2005]. The GD tries to reach the minimum value of a function.

In case of convex functions, the GD converges to global optimum [Li et al., 2018]. But in most cases of ANN, the loss function is non-convex that generates local minimums.

The Figure 8 shows the process to minimize the value of function $z = x^2 + 2y^2$. Each blue arrow is a step with GD that follows the global minimum at the center. In the ANN case, the search is for a value that minimizes the error rate in each iteration.

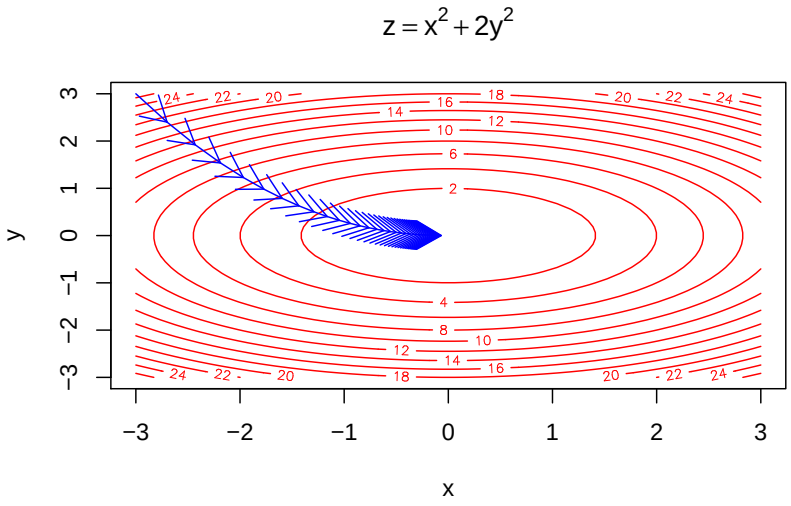


Figure 8 – Plot example of gradient descent on the function $z = x^2 + 2y^2$.

The gradient method has some techniques to conduct a search for a minimum in a function. Two commonly used is the traditional GD and a variation called Stochastic Gradient Descent (SGD) [Snyman, 2005; Bengio, 2012]. The SGD updates the weights at each iteration in a random sample in a batch while the GD updates only when all input set is iterated [Bottou and Cun, 2004]. It causes a GD to have a higher computational cost than the SGD when applied to a large dataset.

Equation (4) and (5) show how a parameter vector θ is updated in GD and SGD, respectively [Bottou and Cun, 2004; Bengio, 2012; Bottou, 2012]. The variable z represents an example of input-output pair (x, y) of iterations t on the samples, ϵ_t is a hyperparameter known as learning rate and $\frac{\delta L}{\delta \theta}(z, \theta)$ is a gradient vector. The SGD takes one random sample z_t of an set with n examples, while the GD process n completely.

$$\theta(k) = \theta(k-1) - \frac{1}{n}\epsilon_k \sum_{i=1}^{n} \frac{\delta L}{\delta \theta}(z_i, \theta(k-1)) \tag{4}$$

$$\theta(t) = \theta(t-1) - \frac{1}{t}\epsilon_t \frac{\delta L}{\delta \theta}(z_t, \theta(t-1)) \quad (5)$$

Beyond GD and SGD, there are many other optimization functions based on the gradient model. One of these is *Adam* [Kingma and Ba, 2014] (according with Kingma and Ba [2014], the name is derived from adaptive moment estimation). The *Adam* is a stochastic optimizer that works with some differences compared to SGD: it adapts the learning rate value instead of keeping it fixed. This update is performed by estimates two gradient moments: the first moment using the average and the second moment using the variance. This method is well compatible when processing a large amount of data, besides being more efficient than SGD from the computational point of view [Kingma and Ba, 2014].

II.3.4 Common Neural Network Hyperparameters

In the context of training neural network models, there are several hyperparameters to be explored. They can be separated into two groups: hyperparameters that will deal with the optimization of the parameters vector of a ANN and hyperparameters that will handle the structure of the model [Bengio, 2012]. Some of the main hyperparameters will be described in more detail below, following the model presented by Bengio [2012], divided into: 1. hyperparameters used on gradient model and 2. architecture hyperparameter.

1. Gradient Model's Hyperparameter

   (a) **Learning rate** - The learning rate ϵ_t (Equation 5) is one of the most important hyperparameters to a good ANN. It defines how fast or slow convergence will occur in the stochastic gradient descent iterations random examples. If this value is too high, the average loss of the network will increase [Bengio, 2012]. High values cause divergence while low values cause faster and imprecise convergence. Bengio et al. [2012] present an efficient way of starting the search in this hyperparameter with a high value and decrease it until there is no more divergence in the *training*. They also indicate a search range of less than 1 and greater than 10^{-6} with 0.01 being a value that usually works well.

(b) **Learning Rate Scheduler** - The idea of this hyperparameter is to control the learning rate over time. Starting a schedule by changing the higher the learning rate quickly improves the result of an expected function. Then make small changes with smaller bands in the learning rate [Senior et al., 2013].

(c) **Mini-batch sizes** - The batch is usually defined as the number of instances in the set of *training*. The gradient descent iteration performs throughout all this set which can be costly. The mini-batch technique is to split the *training* instance into blocks of smaller sizes, resulting in a good impact on the computational performance [Bengio, 2012]. Generally, a very high value in this hyperparameter can cause an overfitting [Bengio, 2012].

(d) **Epoch** - After the ANN iterates on all *training* set, it completes one epoch. The epoch is the number of complete scans in the *training*. Usually, several epochs are needed to make sure that the learning process finds a good combination of model parameters.

2. Neural Network architecture hyperparameters

(a) **Number of hidden layers and neurons** - Larochelle et al. [2009] experiments show that a setup that works well for most cases is to match the number of neurons in each hidden layer. For simpler situations, a ANN with some hidden layers will reach good results. This value should be increased for more complex cases like *Deep Learning*. The *Deep Learning* uses the hierarchical concept features. This model needs many hidden layers for a high level of abstraction.

(b) **Activation function** - As already seen, the activation function of a neuron has an important role in network performance. It expresses how the data is processed internally in a neuron. Besides the function *sigmoid* briefly presented, there are others that can be explored such as Hyperbolic tangent (TanH) [Karlik and Olgac, 2011] and Rectified Linear Unit (ReLU) [Nair and Hinton, 2010]. These functions are presented in Equations (6), (7) and (8), respectively.

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{6}$$

$$f(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}} \tag{7}$$

$$f(\alpha) = max(\alpha, 0) \tag{8}$$

These activation functions have different output values. The sigmoid range is $(0, 1)$, TanH $(-1, 1)$ and ReLU $[0, \infty)$. The TanH curve is similar to *sigmoid*. The difference is tangential being centered at 0 (generating better training) while the sigmoid does not. In addition, both present a linear form at their shapes, making inputs with great distances between them generate outputs with few differences or even the same value. This problem is known as *vanishing gradient*. That's why the use of ReLU has gained much popularity in recent years. In addition to being centered at 0, it does not have a point to occur the *vanishing gradient*, generating better results in general. The curve of each activation function are shown in Figure 9.

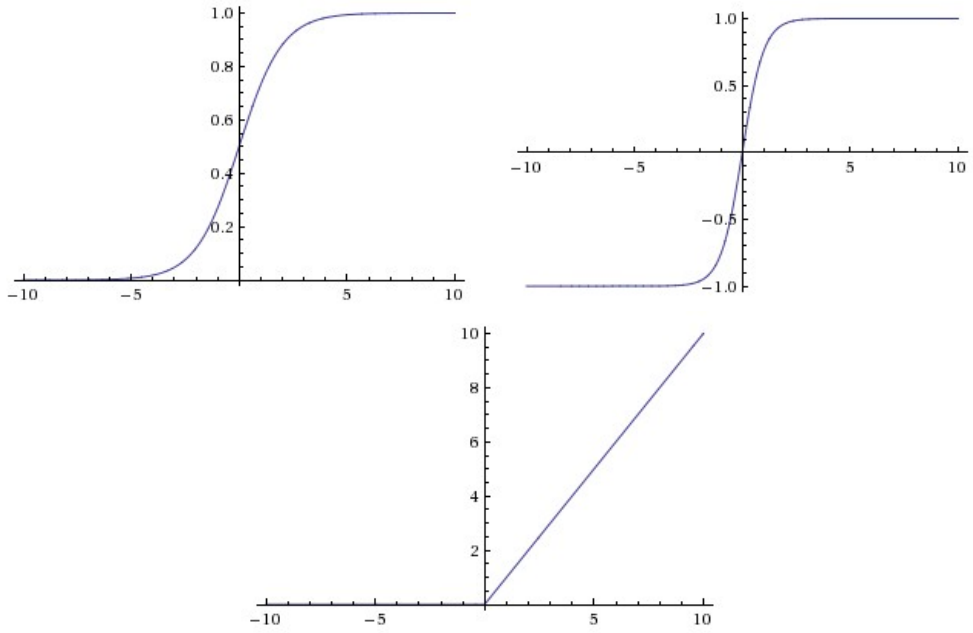


Figure 9 – Activation function plots. The top left represents the sigmoid, followed by the TanH on the side. At the bottom the ReLU.

(c) **Cost function** - The cost function is used to measure the network error in the input-output mapping. It's value indicates how good or bad the result was at the output layer of the network. A main goal of optimization is to decrease this error rate generated by this function. Some functions used are *Cross Entropy* and Mean Square Error (MSE). The Equations (9) and (10) refer to MSE and *Cross Entropy*, respectively [Bishop, 2006]. The value $\hat{y}$ is the ANN output value, y is

the real class value and N is the number of samples in the *training* data.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{9}$$

$$H(y, \hat{y}) = -\sum_{i=1} y(x)\log(\hat{y}(x)) \tag{10}$$

(d) **Regularization** - In some cases, the setting of the network can be overdone upon a particular set of data, impairing performance in a real situation such as in *test* set. This is known as overfitting [Bengio, 2012]. To avoid it, a regularization coefficient λ can be added to a cost function. There are two types of regularization: $L1$ and $L2$. The $L1$ adds the term $\lambda_1 \sum_i |\theta_i|$ to the function, while $L2$ adds the term $\lambda_2 \sum_i \theta_i^2$ [Bengio, 2012]. The values of λ_1 and λ_2 must be defined according to the degree of importance of regularization for the case. The terms of regularization must be chosen with attention, some cases can generate a chaotic behavior [Van Den Doel et al., 2012].

(e) **Initial weights value** - The weights are usually initialized at a small random value. This technique works well on networks with a single hidden layer, but it may occur problems with deep ones. According to Bengio [2012], weights need to be initialized in a way that can break the symmetry between the hidden layers of the ANN. It helps the neurons of the same layer do not receive and propagate the same signal.

(f) **Preprocessing** - The data preprocessing has an important role in network performance. In addition to containing steps that may reduce data noise, such as data cleaning and normalization, this step allows dimensionality reduction with the use of methods such as Principal Components Analysis (PCA) [Bengio, 2012; Machado et al., 2016]. Fewer dimensions in the input data represent faster and therefore more accurate training.

Chapter III Related Work

Several works explore the space of hyperparameters in order to try to optimize some kind of learning algorithm. Manual optimization technique, Grid Search, Random Search and Bayesian optimization are used to get the maximum performance of different algorithms in different areas. Learning methods like Deep Belief Networks (DBN) [Bergstra and Bengio, 2012], Support Vector Machine (SVM) [Larochelle et al., 2007], ANN MLP [LeCun et al., 1998] and Random Forests [Machado et al., 2016] are used together with these optimization techniques to optimize different types of hyperparameters.

Larochelle et al. [2007] explore the hyperparameter space through the Grid Search method. The experiments are made with several datasets like Modified National Institute of Standards and Technology (MNIST) and other images sets. They explore methods like SVM with two kernels variations, a Neural Network with one hidden layer, a DBN and Stacked Autoassociators (SAA). The results show that deep networks like DBN have a better performance compared with shallows architectures.

Snoek et al. [2012] show methods for hyperparameter search by Bayesian Optimization. The Bayesian Optimization takes into account samples of the Gaussian Process to performs an optimization. Due to this, it has characteristics that must be explored like acquisition function and parallelism techniques that are presented by the authors. They briefly show the experimental results applied to the CIFAR-10 dataset and generated a model with one of the smallest errors until then.

Bergstra and Bengio [2012] perform experiments with Random Search in comparison with the results obtained by Larochelle et al. [2007] experiments. They show for most of datasets that the Random Search statistically equal the Grid Search on four datasets and better in one only with a small fraction of computational time, this makes the Random Search superior, both in accuracy and in execution time. But they concluded that Random Search does not overcome the Grid Search when the grid space of hyperparameters is generated by a specialist.

Machado et al. [2016] evaluate an exploratory process in many classifiers methods using a Grid Search to star/galaxy separation problem. A one layer Neural Network, Random Forest, SVM, k-Nearest Neighbor and Naive Bayes are compared in this work.

A workflow for preprocessing and classify the data is used within the Hadoop plataform to execute the hyperparameter optimization. The results show a better result for Neural Network and Random Forest both in purity and computational efficiency metrics.

Loshchilov and Hutter [2016] compared the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) against Bayesian Optimization and some variations in the acquisition function and in the optimizer. The results show that CMA-ES has comparable results taking into account the validation set error in some Bayesian Optimization variations and also with a lower computational cost.

Author	GS	RS	BO	CMA-ES
[Larochelle et al., 2007]	✓	X	X	X
[Bergstra and Bengio, 2012]	✓	✓	X	X
[Snoek et al., 2012]	X	X	✓	X
[Machado et al., 2016]	✓	X	X	X
[Loshchilov and Hutter, 2016]	X	X	✓	✓
This dissertation	✓	✓	✓	✓

Table 1 – Summary of related works by method.

Table 1 summarizes the related works according to each hyperparameter optimization strategy covered in the study. To the best of out knowledge, there is no work in the literature that uses BRKGA meta-heuristics for hyperparameter optimization in a machine learning context, as proposed in this dissertation.

Chapter IV Hybrid Biased Random-Key Genetic Algorithms

This chapter presents the whole basis used for the creation of the HBRKGA algorithm, an adaptation of the BRKGA framework [Toso and Resende, 2015]. It is organized as follows. Firstly, Section IV.1 provides a formal description of the problem of hyperparameter optimization in a machine learning context. Section IV.2 discusses the adaptation of the framework to the problem of hyperparameter optimization of neural networks. Section IV.3 shows the concepts of the Random Walk component. Section IV.4 describes the HBRKGA main procedure.

IV.1- Problem Statement

Let $\Gamma_{\mathcal{A}}$ be the set of possible hyperparameter combinations for a learning algorithm $\mathcal{A}$. Consider the general optimization problem of finding $\gamma^{\star} \in \Gamma_{\mathcal{A}}$, the best set of hyperparameters for $\mathcal{A}$. Here, best is defined using a function $f : \Gamma_{\mathcal{A}} \to \Re$ that provides an estimate of the predictive performance of $\mathcal{A}$ on some validation set $\mathcal{X}$. This optimization problem can be formalized as:

$$\gamma^{\star} = \underset{\gamma \in \Gamma_{\mathcal{A}}}{\operatorname{argmin}} f(\gamma; \mathcal{A}, \mathcal{X}) \tag{11}$$

The variable γ is a vector sampled from hyperparameter space $\Gamma_{\mathcal{A}}$. The goal of the optimization problem is to find a γ that minimizes f. With currently existing methods for hyperparameter search, there is no way to guarantee that the optimal value of γ can be found. However, methods with an approximate approach (such as the ones in evidence in this work) can generate satisfactory models.

IV.2- Encoding and decoding candidate solutions

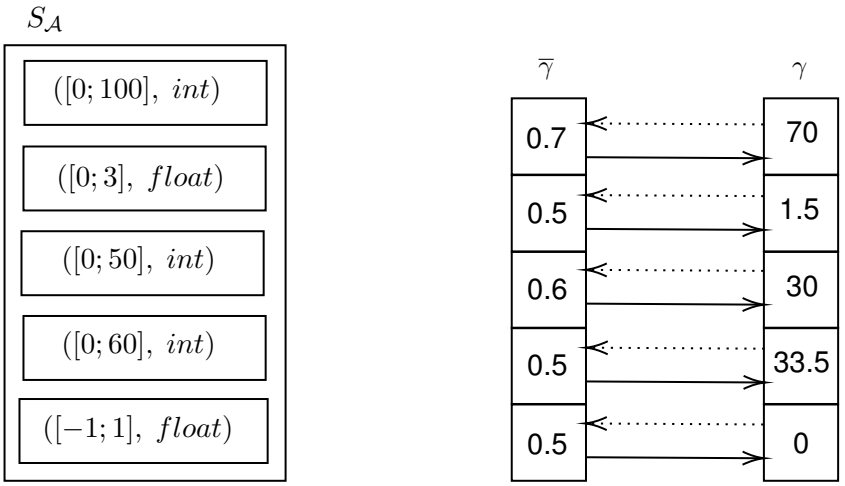
In general, each hyperparameter of a learning algorithm $\mathcal{A}$ has its corresponding range of values and data type. Furthermore, a given strategy for hyperparameter search may be able to search only a particular subrange of values of a hyperparameter. To cope with this, we define an abstract data type to be used in the procedures presented hereafter. Let us denote an instance of this abstract data type by $S_{\mathcal{A}}$. The data part of $S_{\mathcal{A}}$ holds a list. Each entry in this list corresponds to one hyperparameter of $\mathcal{A}$, in which it's considered subrange and data type are stored.

The operations $min(S_{\mathcal{A}}, i)$ and $max(S_{\mathcal{A}}, i)$ are defined for such an abstract data type. These operations return the minimum and maximum values for the i-th hyperparameter, respectively (i.e., its considered subrange). Another operation defined in the context of $S_{\mathcal{A}}$ is $dt(S_{\mathcal{A}}, i)$. This function returns the data type of the i-th hyperparameter. A final operation we define for this abstract data type is $round(S_{\mathcal{A}}, i, v)$. This operation returns the value that is closest to $v \in \Re$ considering two constraints: the returned value (1) has the same data type as the i-th hyperparameter and (2) it is inside the close interval $[min(S_{\mathcal{A}}, i), max(S_{\mathcal{A}}, i)]$.

Given the definition of the abstract data type provided above, we encode a solution in a vector $\bar{\gamma}$ of n random keys, in which n corresponds to the number of hyperparameters in $\mathcal{A}$. This way, a value for the i-th hyperparameter, γ_i ($1 \leq i \leq n$), is mapped to its corresponding key $\bar{\gamma}_i$ using Equation 12.

$$\bar{\gamma}_i = \text{round}\left(S_{\mathcal{A}}, i, \frac{\gamma_i - min(S_{\mathcal{A}}, i)}{max(S_{\mathcal{A}}, i) - min(S_{\mathcal{A}}, i)}\right) \tag{12}$$

Figure 10 illustrates the process of encoding and decoding candidate solutions through the transformation represented by Equation 12. Figure 10a shows the data part of an instance $S_{\mathcal{A}}$ of the abstract data type described in this section. This instance represents information about five hyperparameters. Figure 10b shows the mapping between values of γ (vector of hyperparameters) and $\bar{\gamma}$ (vector of random keys) according to the information in the instance $S_{\mathcal{A}}$ shown in Figure 1a.



(a) $S_{\mathcal{A}}$ data part. (b) Mapping between γ and $\bar{\gamma}$.

Figure 10 – Example of mapping (i.e., encoding or decoding) between a vector of hyperparameter values (γ) and a vector of BRKGA key values ($\bar{\gamma}$).

IV.3- Random-Walk procedure

A crucial component of HBRKGA is its Random-Walk procedure, whose pseudocode is presented in Algorithm 1. Since it is impossible to analyze every solution in the neighborhood of $\bar{\gamma}$, in this work we chose to blindly explore the space in a small neighborhood. Hence, the Random-Walk phase performs a stochastic search in the neighborhood of a given individual $\bar{\gamma}$ of the current population, looking for a better candidate solution. In particular, a sequence of perturbations is generated from the original decoded solution and the best one is returned as the new best solution. Below, we provide details of such the Random-Walk phase.

Algorithm 1 receives $\bar{\gamma}$ as an input parameter, a candidate solution for the optimization problem containing n random keys. In line 2, the algorithm decodes the solution from BRKGA domain (random keys) to the HBRKGA domain (hyperparameter values). The decoded solution is then passed to the $\mathrm{Evaluate}$ procedure (line 3) along with a learning algorithm $\mathcal{A}$ and a dataset $\mathcal{X}$. This procedure corresponds to apply $\mathcal{A}$ (using the values in γ as hyperparameters) to fit a model to $\mathcal{X}$. This procedure returns a solution with its numeric value (i.e. score) that reflects the quality of the fitted model.

The amount of steps to be performed in the random walk is determined by the input

Algorithm 1 – RandomWalk($\bar{\gamma}, \mathrm{nmov}, \mathcal{A}, \mathcal{X}, S_{\mathcal{A}}$)

```
1  begin
2      Map γ̄ to γ using Equation 12;
3      γ_temp.score ← Evaluate(γ, A, X);
4      γ_temp ← γ;
5      for 1 to nmov do
6          γ_temp ← Movement(γ_temp, S_A);
7          γ_temp.score ← Evaluate(γ_temp, A, X);
8          if γ.score < γ_temp.score then
9              γ ← γ_temp;
10             γ.score ← γ_temp.score;
11     Map γ to γ̄ using Equation 12;
12     return γ̄;
```

parameter nmov (line 5). The higher the value of $nmov$, then the higher the amount of pertubations created and a higher processing time. At each step a movement is performed (line 6) in the hyperparameter space. The definition of movement here corresponds to applying a perturbation to one of the components of the input vector. The component to which the perturbation is to be applied is chosen uniformly at random. The neighborhood considered in the exploration for the selected component γ_i is the closed interval $[0, \gamma_i(1 + \epsilon)]$, according to Equation (13). The small positive constant number ϵ is a hyperparameter of HBRKGA. We use the round function here (Section IV.2) to cope with the case in which the value resulting from the perturbation has an incompatible data type (e.g. a floating point value when γ_i only assumes integer values).

$$\gamma_i \leftarrow \text{round}\left(S_{\mathcal{A}}, i, \gamma_i + (1 - 2 \times \text{Bernoulli}(0.5)) \times \text{Unif}(0, \gamma_i(1 + \epsilon)\right) \tag{13}$$

At each Random-Walk step, the resulting perturbed hyperparameters vector is evaluated (line 7) in order to keep track of the best current candidate solution (lines 8-10). At the end (line 11), the algorithm encodes the refined solution back to the BRKGA domain (random keys) before returning it to HBRKGA main procedure.

IV.4- HBRKGA main procedure

Now that the Random-Walk phase has been described, we can proceed to present the main procedure of our proposed population-based approach to hyperparameter search. Algorithm 2 presents the pseudo-code for HBRKGA. The purpose of this algorithm is to find $\gamma^{\star}$, the best possible configuration of hyperparameter values for a given learning algorithm $\mathcal{A}$.

A random initial population p of individuals is generated (line 3). Each individual $p[i]$ ($1 \leq i \leq q_{ind}$) in the population is a vector of random keys. Each component in such a vector is a random value drawn from a standard uniform distribution: $p[i, j] \sim$ Unif$(0, 1)$.

The main loop of the algorithm starts at line 4. This loop is controlled by a stopping criterion. There are several alternative stopping criteria to use. Examples are the maximum number of generations, maximum runtime or until a specific value for the fitness function is reached. The inner loop starting at line 5 performs the Random-Walk phase (Algorithm 1) for each individual. As a result, the i-th individual in p is (potentially) changed to one of its neighbors, if it is the case that the latter evaluates better than the former.

Line 7 partitions the current population p into two subsets p_e and $p_{\bar{e}}$ in such a way that $|p_e| = q_e$ and $|p_{\bar{e}}| = q_{ind} - q_e$. To form p_e, the individuals in the current generation are first sorted according to their scores. Then, the top q_e individuals from the current generation are selected to form the elite set p_e.

Line 9 generates q_m mutant individuals. The mutants replace a fraction of the actual population with new random individuals with n random keys. The mutants are created following a random uniform distribution. These individuals are important since they help the optimization process to escape local minima [Gonçalves and Resende, 2011].

The loop between lines 11-17 starts the reproduction operator. One random parent a from p_e and another one b from $p_{\bar{e}}$ are selected. Then the inner loop in line 14 is executed for each key n. A random variable X is generated to define which parent (a or b) the offspring c will inherit the characteristics of a specific key. The parent is selected according to a Bernoulli distribution with parameter ϕ_a. This selection is biased towards the parent in the elite set (i.e., there is a greater probability of the parent in the elite set to be selected).

Algorithm 2 – $\mathrm{HBRKGA}(q_{\text{ind}}, q_e, q_m, n, \phi_a, \mathrm{nmov}, \mathcal{A}, \mathcal{X}, S_{\mathcal{A}})$

1 **begin**
2 Initialize score of the best solution found: $\gamma^{\star}.score \leftarrow \infty$;
3 Randomly generate a population p with q_{ind} n-dimensional vectors of random keys;
4 **while** stopping criterion not satisfied **do**
5 **for** $i \leftarrow 1$ to q_{ind} **do**
6 $p[i] \leftarrow \text{RandomWalk}(p[i], \mathrm{nmov}, \mathcal{A}, \mathcal{X}, S_{\mathcal{A}})$
7 Partition p into two sets: p_e and $p_{\bar{e}}$;
8 Initialize population of next generation: $p^{+} \leftarrow p_e$;
9 Generate set p_m of mutants, each mutant with n random keys;
10 Add p_m to population of next generation: $p^{+} \leftarrow p^{+} \cup p_m$;
11 **for** $i \leftarrow 1$ **to** $q_{\text{ind}} - (q_e + q_m)$ **do**
12 Select parent a at random from p_e;
13 Select parent b at random from $p_{\bar{e}}$;
14 **for** $j \leftarrow 1$ **to** n **do**
15 Draw random variable $X \sim \text{Bernoulli}(\phi_a)$
16 c[j] $\leftarrow \begin{cases} a[j] & \text{if } X = 1, \\ b[j] & \text{if } X = 0. \end{cases}$
17 Add offspring c to population of next generation: $p^{+} \leftarrow p^{+} \cup \{c\}$;
18 Update population: $p \leftarrow p^{+}$;
19 Find best solution in p: $\gamma^{+} \leftarrow \mathrm{argmin}_{1 \leq i \leq q_{ind}}(p[i].score)$;
20 **if** $\gamma^{+}.score < \gamma^{\star}.score$ **then**
21 $\gamma^{\star} \leftarrow \gamma^{+}$;
22 **return** $\gamma^{\star}$;

After being created, the offspring c is added to the next generation p^{+}.

After the reproduction process, the population p is updated from p^{+} in line 18. Finally, find the best solution y^{+} in the current population to update $\gamma^{\star}$ (only if $\gamma^{+}.score < \gamma^{\star}.score$) between lines 19-21. Here, the $score$ method is used to get the quality metric of the model. The best solution ($\gamma^{\star}$) is finally returned (line 22).

Figure 11 presents a birds-eye view of HBRKGA. Initially, HBRKGA receives a hyperparameter space $S_{\mathcal{A}}$ and sends an individual solution to decode the individual $\bar{\gamma}$ to a value γ mapped to the domain in question. Using a Random Walk, a local search like method is applied nmov times to the solution γ. After that, γ is encoded and the best individual in the current population γ^{+} is returned to the HBRKGA framework, as presented in Algorithm 2. This process is repeated for each generation, until the return of the final best solution $\gamma^{\star}$.

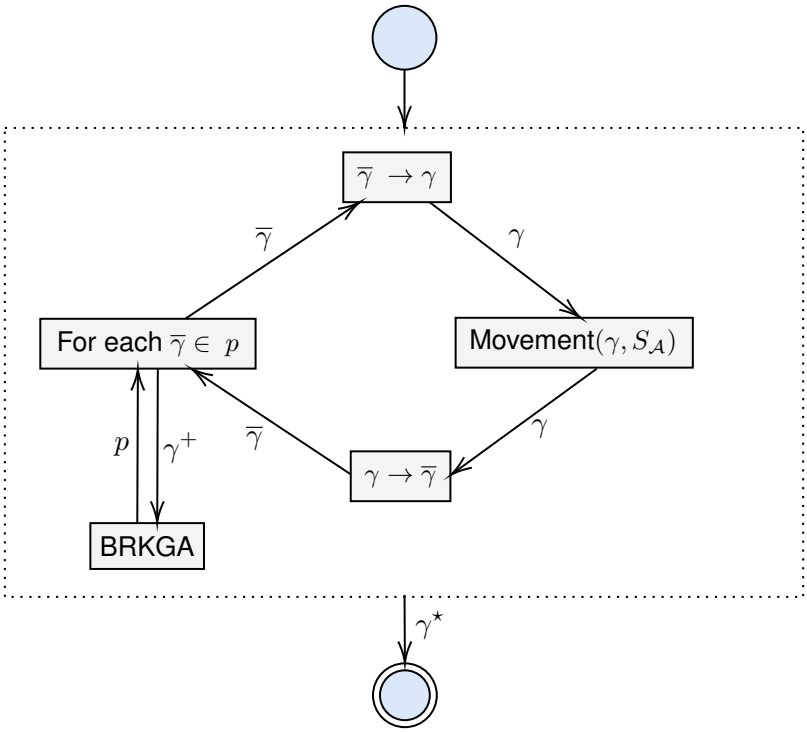


Figure 11 – HBRKGA overview. Individuals of the current generation p are created according to BRKGA rules. Then each individual is possibly refined in the Random-Walk procedure.

Chapter V Experiments

This chapter describes the computational experiments we performed to validate our proposed hyperparameter optimization method. We perform experiments on eight publicly available datasets coming from several different application domains. We start by providing details about the datasets (Section V.1), evaluation metrics (Section V.2), and experimental settings (Section V.3). Further, we describe the main results of the experiments for each dataset (Section V.4) and summarize the results of ablation studies (Section V.5).

V.1- Datasets

To perform our validation experiments, we used eight datasets in total. Six of them are provided by Larochelle et al. [2007]. These are the original version and variants of MNIST, one of the most popular datasets in the image recognition and classification areas. We also reuse the Rectangles dataset from Larochelle et al. [2007]. We also used the Fashion-MNIST dataset [Xiao et al., 2017]. Finally, to provide a better basis for experimental results, we have added COSMOS, an unbalanced dataset [Fadely et al., 2012; Machado et al., 2016].

V.1.1 MNIST

MNIST is a set of handwritten digit image data, having 60,000 examples in the training set and 10,000 examples in the test set [LeCun, 1998]. The images in the dataset have a size of 28x28 pixel, totalizing 784 features. Several studies have already done using this dataset [Bergstra and Bengio, 2012; Larochelle et al., 2009], one of the objectives of these studies being to achieve the smallest possible error in the identification of these

digits.

Larochelle et al. [2007] present a study with many factors of variations on top of MNIST, such as rotated digits and the addition of noise in the background of the images. With these variations, it is possible to observe several factors in the classification. These datasets are also used by Bergstra and Bengio [2012] to perform Random Search experiments. The MNIST variations that we selected are shown below and can be observed in some examples in Figure 12. We follow the split: 12,000 images for training (the last 2,000 examples were used in the validation set) and 50,000 images for the test.

1. MNIST rotated (MNIST-R): the images suffered slight rotation in the digits, trying to reproduce different writing styles.

2. MNIST with a random background (MNIST-RanBack): adding a random background in the digit images. This factor produces noises in the digits.

3. MNIST with image background (MNIST-IB): a background was produced with pieces of 20 images taken from the Internet.

4. MNIST with rotation and background (MNIST-RotBack): the combination of the first two MNIST variations that were presented, resulting in rotate digits with some noise in the background.

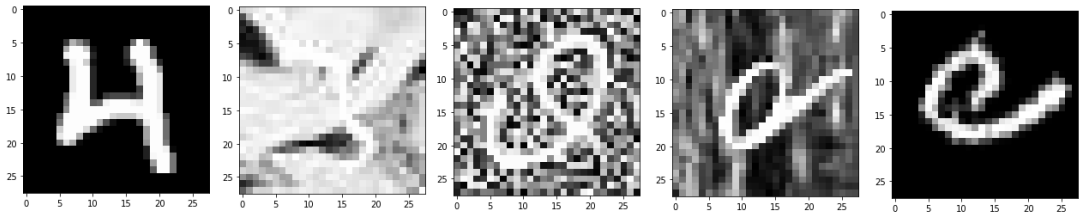


Figure 12 – MNIST and its variations.

V.1.2 Rectangles

In addition to MNIST and its variations, we selected one more case from Larochelle et al. [2007], the rectangles images. The objective of the rectangles dataset is the discrimination between tall and wide rectangles. Like MNIST, it has 28x28 pixel dimensions.

The Figure 13 shows an example of the label tall and wide. The training set has 1000 images and the validation set has 200. The test set has 50000 images.

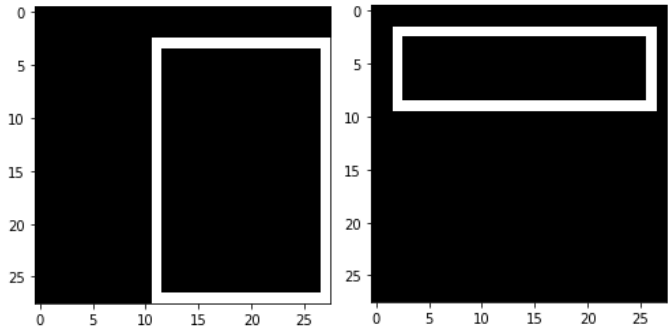


Figure 13 – Tall and wide rectangle example.

V.1.3 Fashion-MNIST

The Fashion-MNIST dataset provided by Xiao et al. [2017] contains 60,000 training examples and 10,000 test examples in 28x28 grayscale images divided by 10 categories of fashion products: t-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag and ankle boots. These classes are balanced over this dataset. It is an alternative to the MNIST benchmark for machine learning algorithms with more complex tasks for the correct classification. For the validation set, we use 10% of the training set. Figure 14 shows some examples of images from this dataset.

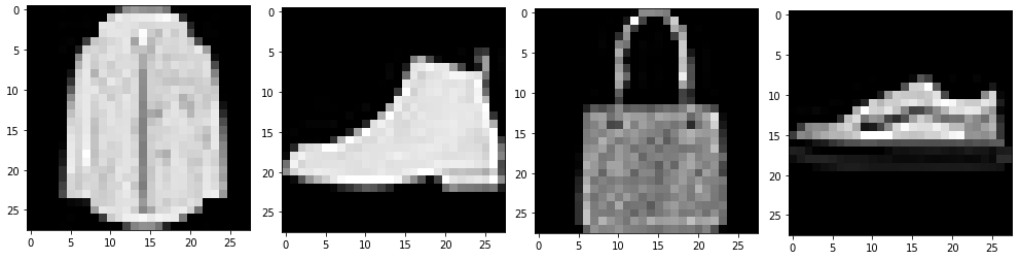


Figure 14 – Fashion-MNIST images example.

V.1.4 The Cosmic Evolution Survey

The Cosmic Evolution Survey [Scoville et al., 2007] is a catalog with information about more than 500000 objects and 90 attributes with its photometric measure. We used the same dataset and preprocessing in Machado et al. [2016], which takes into account feature selection, outliers removal, data cleaning, normalization and test/validation split. t covers a star/galaxy classification problem that is a hard task due to an unbalanced amount between star (386,957 objects after preprocessing that represents 98.55% of the total in the dataset) and galaxies (5,542 objects after preprocessing). We used five photometric features and its related error in the measure as the input for the ANN, totalizing 10 features and a target label for star/galaxy classification.

V.2- Evaluation Metric

In the experiments presented in this paper, we restrict ourselves to classification problems. Hence, we selected the metric F_1-score to measure models' quality in all the evaluated hyperparameter search methods. The F_1 metric (Eq. 14) is computed as the harmonic mean of two other metrics, precision and recall [Han et al., 2011]. Precision is the percentage of examples predicted by the model as belonging to a given class that genuinely belong to that class (Eq. 15). Recall is the percentage of examples of a given class correctly classified as so by the model (Eq. 16).

$$F_1 = \frac{2\pi\rho}{\pi + \rho} \qquad (14) \qquad \pi = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \qquad (15) \qquad \rho = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad (16)$$

In the equations above, TP, FP, and FN are the true positive, false positive, and false negative counts. The F_1 score ranges from 0 to 1. A good model is expected to achieve an F_1 value close to 1, while models with low predictive quality tend to produce an F_1 score near 0. Since all the datasets we use in our experiments present multiple classes, we simply average the F_1-scores for each class and calculate a mean F_1-score as the final evaluation metric.

V.3- Experimental Settings

We ran the experiments on a computer with an Intel(R) Core(TM) i7-6700 CPU 3.40GHz processor, 32GB RAM, equipped with a GeForce GTX 1080 GPU. The ANN algorithm was developed using the *Tensorflow* library [Abadi et al., 2016]. As a basis to develop HBRKGA, we used a BRKGA implementation provided by Toso and Resende [2015]. We implemented the abstract data type described in Section IV.2 to cope with the problem-dependent mapping procedure for converting a vector of random keys into hyperparameter values and vice-versa.

Inspired by previous similar experimental work, namely Bergstra and Bengio [2012] and Larochelle et al. [2007], we selected five hyperparameters in a range of values to be explored by the methods presented. They are based on an ANN with three hidden layers architecture. The reused hyperparameters and their values are presented in three initial lines in Table 2. In particular, we chose the following hyperparameters: number of neurons in first, second, and third hidden layers, learning rate, and regularization rate. For the COSMOS dataset, we defined the range proportionally, since this dataset has a much lower number of input features.

Table 2 – Hyperparameter range values for each dataset and its variations.

Dataset	Neurons Layer 1	Neurons Layer 2	Neurons Layer 3	Learning rate	Beta
MNIST	[1000, 2000]	[2000, 4000]	[2000, 6000]	$[10^{-6}, 10^{-1}]$	[0, 10^{-3}]
Rectangle	[1000, 2000]	[2000, 4000]	[2000, 6000]	$[10^{-6}, 10^{-1}]$	[0, 10^{-3}]
COSMOS	[5, 15]	[5, 30]	[5, 45]	$[10^{-6}, 10^{-1}]$	[0, 10^{-3}]

We use the optimization strategies described in Section II.1 as baseline for comparison to HBRKGA. We implemented Grid Search and Random Search from scratch. We used publicly available implementations for Bayesian Optimization[1] and CMA-ES[2].

We keep track of the number of solutions produced by each optimization strategy in each run of experiments for time comparison between them. The Grid Search optimization generates 240 combinations of different hyperparameters values to run in each dataset. This number results from combining the following values: 2 values for the first layer, three

[1] `https://github.com/fmfn/BayesianOptimization`
[2] `https://github.com/CMA-ES/pycma`

values for the second layer, four values for the third layer, five values for the learning rate, and two values for the regularization rate (denoted Reg in Table 2). Due to that, and to make fair comparisons, we limited the maximum number of searches (i.e., a generation of hyperparameters) in each strategy to 240, already including the initial solution performed by Bayesian Optimization, CMA-ES, and HBRKGA. In this work, we configure HBRKGA parameters according to Table 3. In Bayesian Optimization, we use Upper Confidence Bound as the acquisition function with 20 random initial points and 220 optimization steps. Finally, at CMA-ES we use 10 generations with 24 individuals.

Parameter	Value
Max. number of populations (stopping criteria)	10
Population size (q_{ind})	6
Elite set size (q_e)	2
Mutant set size (q_m)	1
Offspring probability (ϕ_a)	70%
Steps in Random-Walk (nmov)	3
Perturbation ratio (ϵ)	15%

Table 3 – HBRKGA parameters settings.

We use cross-entropy as loss function, with a softmax activation function as output layer. In each hidden layer, the ReLU activation function is used. To save computational resources, we used an early stopping technique. The goal is to stop the network training process when the value for the loss function does not decrease for a number of consecutive epochs. We configured the training process to generate a maximum of 300 epochs. If in 13 consecutive epochs a certain the loss function does no decreased in the validation set, the training is automatically stopped, the best model found is returned. We also use ADAM [Kingma and Ba, 2014] optimizer for training the ANNs.

V.4- Experimental Results

For each dataset, we performed ten runs of experiments for each hyperparameter optimization strategy covered in this work. We then computed statistical summaries for the F_1 metric and the execution time (in seconds). We divide the presentation of the experimental results into two parts. In the first part, we describe results related to the predictive

quality of the classification models produced using each search strategy (Section V.4.1). In the second part, we present computational performance results concerning each strategy (Section V.4.2).

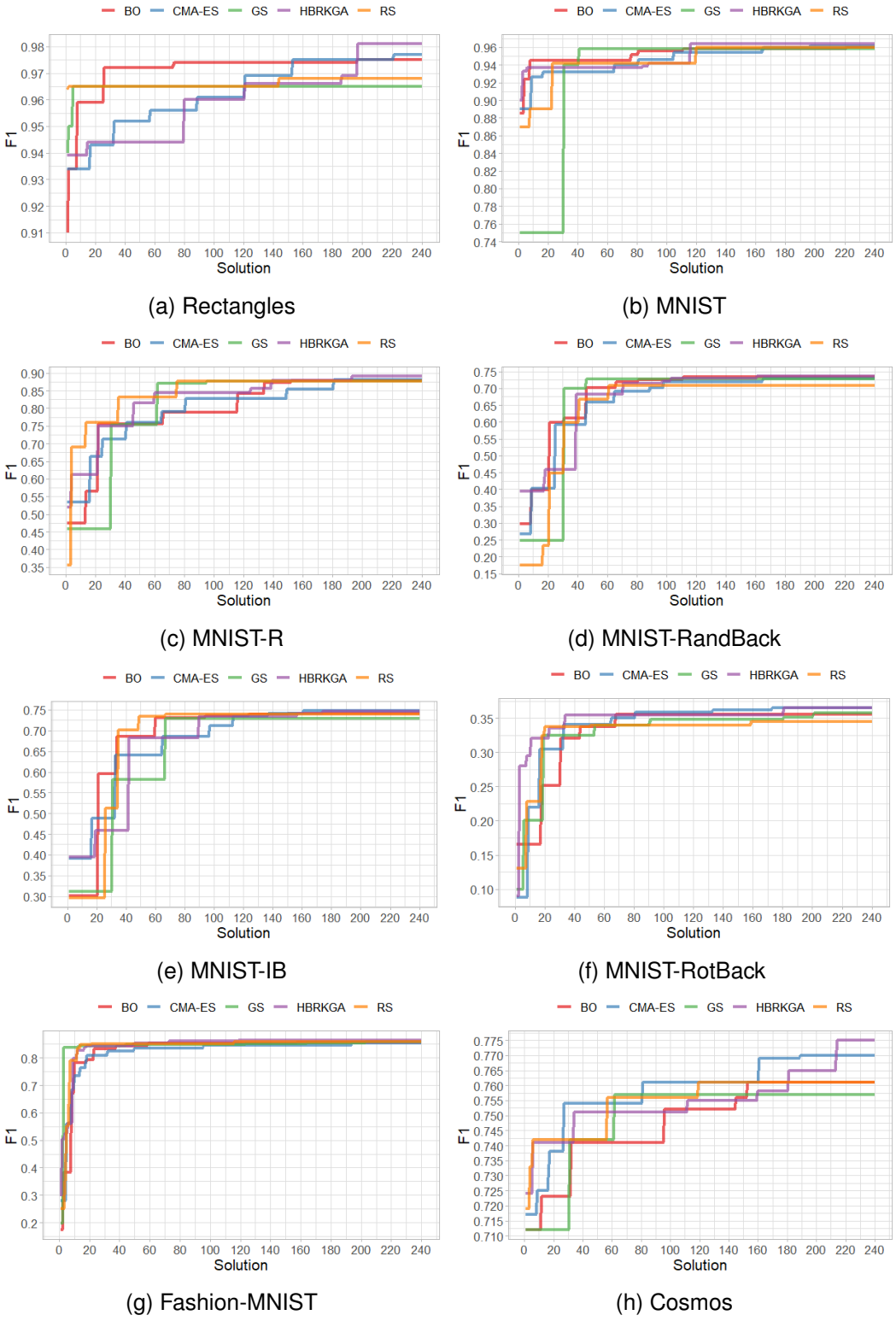
V.4.1 Predictive Quality

Table 4 presents the results obtained by taking the mean and standard deviation of the best F_1 value found in each of the ten trials of experiments in the validation set for each dataset. We observe an increase in the mean of HBRKGA compared to Bayesian Optimization, CMA-ES, Random Search, and Grid Search in 6 of 8 datasets. The CMA-ES method was able to outperform HBRKGA results on MNIST-IB and matched the HRBKGA results on MNIST-RotBack. The most significant difference in the mean of F_1 between the HBRKGA and the second-best method occurred in the COSMOS and MNIST-R datasets. In these cases, HBRKGA increased the mean F_1 value by 0.006 and 0.009, respectively. Only in MNIST-RandBack, the Bayesian Optimization method was able to overcome the CMA-ES.

To summarize, HBRKGA obtained the best average F_1, followed by CMA-ES and Bayesian Optimization. In this global metric, Grid Search and Random Search achieved the worst F_1 averages among the methods tested. It is also possible to observe that the CMA-ES presented the lowest global mean in the standard deviation value.

Table 4 – Average F_1 results for 10 experimental runs for each dataset. The best results are presented in bold face. The last line presents the average results considering all ten runs.

	GS		RS		BO		CMA-ES		HBRKGA	
	avg	std	avg	std	avg	std	avg	std	avg	std
MNIST	0.958	0.0047	0.962	0.0025	0.960	0.0029	0.962	0.0009	**0.965**	0.0014
MNIST-R	0.877	0.0020	0.877	0.0175	0.879	0.0037	0.882	0.0021	**0.891**	0.0017
MNIST-IB	0.729	0.0121	0.741	0.0104	0.742	0.0158	**0.748**	0.0093	0.746	0.0082
MNIST-RotBack	0.358	0.0051	0.345	0.0049	0.359	0.0046	**0.365**	0.0050	**0.365**	0.0039
MNIST-RandBack	0.727	0.0033	0.708	0.0031	0.735	0.0391	0.731	0.0101	**0.73**6	0.0154
Fashion-MNIST	0.859	0.0007	0.860	0.0010	0.865	0.0037	0.865	0.0019	**0.867**	0.0035
Rectangles	0.965	0.0051	0.972	0.0045	0.975	0.0034	0.977	0.0036	**0.981**	0.0031
COSMOS	0.757	0.0161	0.761	0.0126	0.761	0.0049	0.771	0.0107	**0.777**	0.0129
	0.778	0.0061	0.778	0.0070	0.784	0.0097	0.787	0.0054	**0.791**	0.0062



(a) Rectangles

(b) MNIST

(c) MNIST-R

(d) MNIST-RandBack

(e) MNIST-IB

(f) MNIST-RotBack

(g) Fashion-MNIST

(h) Cosmos

Figure 15 – F_1 mean evolution curve of each method for each dataset.

Figure 15 shows the average F_1 value of the solutions found by the algorithms over time. The evolution of both Grid Search and Random Search (green and orange lines) is quick since they reach their corresponding maxima earlier than the other strategies. After this, they remain practically constant over time. However, in most cases the maximum found by these strategies is lower than the ones found in other strategies. A possible reason for this is that those two strategies do not have a way to escape local mimima. On the other hand, Bayesian Optimization, CMA-ES and HBRKGA (red, blue and purple lines) present, in most of the datasets studied, a constant evolution in the value of F_1 along with the generated solutions. This common characteristic of these methods was determinant factors for the overall better results they presented. Also, it is possible to notice that Grid Search and Random Search strategies, despite not being able to produce the best solutions, need approximately 100 solutions to find their best F_1 average result, which seems to be faster (but not more effective) than Bayesian Optimization, CMA-ES and HBRKGA, which find their best at approximately 160 solutions for all datasets. The Fashion-MNIST and COSMOS datasets presented the fastest and slowest convergences, respectively.

Table 5 – p-values resulting from applying the Wilcoxon test ($\alpha = 0.05$) to compare the baseline methods (Grid Search, Random Search, Bayesian Optimization, and CMA-ES) to HBRKGA.

	GS	RS	BO	CMA-ES
MNIST	0.00017	0.00068	0.00072	0.00015
MNIST-R	0.00017	0.01862	0.00016	0.00017
MNIST-IB	0.02323	**0.31500**	**0.48100**	**0.90350**
MNIST-RotBack	0.00147	0.00021	0.00713	**0.93230**
MNIST-RandBack	**0.06352**	0.00018	**0.9370**	**0.47230**
Fashion-MNIST	0.00018	0.00018	**0.08095**	**0.06954**
Rectangles	0.00017	0.00026	0.00735	0.00638
COSMOS	0.01709	0.01395	0.00357	**0.14850**

We used the Wilcoxon non-parametric test [Wilcoxon, 1992] to verify whether the results are statistically significantly different from each other. We set the significance level $\alpha = 0.05$. The set of 10 runs of Grid Search, Random Search, Bayesian Optimization and CMA-ES were compared against the results of HBRKGA runs. The resulting p-values are presented in Table 5. The cases in which there was no observed a statistically significant difference between the distributions are highlighted in boldface. This occurred in some

methods in the datasets MNIST-IB, MNIST-RotBack, MNIST-RandBack, Fashion-MNIST and COSMOS, especially with Bayesian Optimization and CMA-ES. For all the other methods and datasets, in the conditions studied, the results of HBRKGA are statistically significantly different than other methods.

V.4.2 Computational Performance

The computational performance results of each method are presented in Table 6, which shows the mean execution time and their respective standard deviations for the ten experimental runs. The smallest results found are highlighted in each line. Random Search was able to outperform the other methods in 4 out of 8 datasets. Bayesian Optimization showed the highest processing time in all the experiments performed. It is also possible to highlight that HBRKGA surpassed other methods in the MNIST-RandBack and MNIST-RotBack datasets, which were the MNIST variants with the lowest result in the F_1 metric. The values of hyperparameters generated as a solution for each case directly influences the learning time of ANN. Higher learning rate values can make the model converge faster, while smaller values can make the learning time longer. Grid Search gets the best result for Fashion-MNIST and Rectangles datasets.

Table 6 – Average time results (in seconds) for 10 experimental runs. The best results are presented in bold face. The last line (labelled AVG) presents the average results considering all ten runs.

	GS		RS		BO		CMA-ES		HBRKGA	
	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
MNIST	11050	214	**10836**	303	32298	884	13199	386	14474	474
MNIST-R	13544	199	**11213**	288	28950	710	14323	308	15864	380
MNIST-IB	14284	120	**13621**	356	33020	1950	15666	250	19843	312
MNIST-RotBack	19823	286	19862	258	44985	2232	23425	569	**19253**	1807
MNIST-RandBack	25143	580	24540	856	49779	2232	25203	667	**21988**	513
Fashion-MNIST	**56811**	126	59390	241	109471	10396	59560	847	57214	2094
Rectangles	**2281**	50	2357	98	9563	374	5563	185	6407	305
COSMOS	2238	66	**1157**	167	8011	898	3658	401	3726	353
AVG	18146	205	**17872**	230	39509	2459	20074	451	19846	779

To summarize, Random Search obtained the best average time followed by Grid

Search. CMA-ES and HBRKGA presented comparable results, with a small advantage in the processing time of HBRKGA. BO showed the highest average processing time. It is also possible to observe that the Random Search and Grid Search strategies presented the lowest global mean in the standard deviation value.

V.5- Ablation Study

In general, in an ablation study, the goal is to understand the behavior of a system by removing/changing some components of it and observing the impact. In this section, we describe an ablation study we conducted using the rectangle dataset (Section V.1) to evaluate the impact of the Random-Walk component in the behavior of HBRKGA. The seed was fixed in HBRKGA(0) (without applying Random-Walk) and HBRKGA(3) (using Random-Walk with three steps) algorithms, where the solutions were sent to the generation and quality evaluation of the model in ANN without a fixed seed. This process was repeated ten times in each algorithm.

The total number of solutions at the end of runs were kept the same in both algorithms, taking into account the adjustment by Random-Walk that increases the number of solutions in each generation. For HBRKGA(0), ten generations were used (the first reserved for a random initial solution) with a population size 24, generating 240 solutions in total. For the HBRKGA(3), the generations were reduced to 6 with a population size of 10, but adding 3 Random Walk steps in each population individual generated by the algorithm.

Figure 16 presents the results obtained in comparative experiments. The mean of F_1, inside each bar, indicates an advantage for HBRKGA(3), which increased the mean result obtained by HBRKGA(0) in the conditions studied. It is also possible to verify that HBRKGA(3) showed more variance in F_1 than HBRKGA(0), however, it maintained a better result taking into account the upper limit or the lower limit of the interval. These variations can be caused by ANN factors, like weights initialization and the hyperparameter value behavior generated by both algorithms.

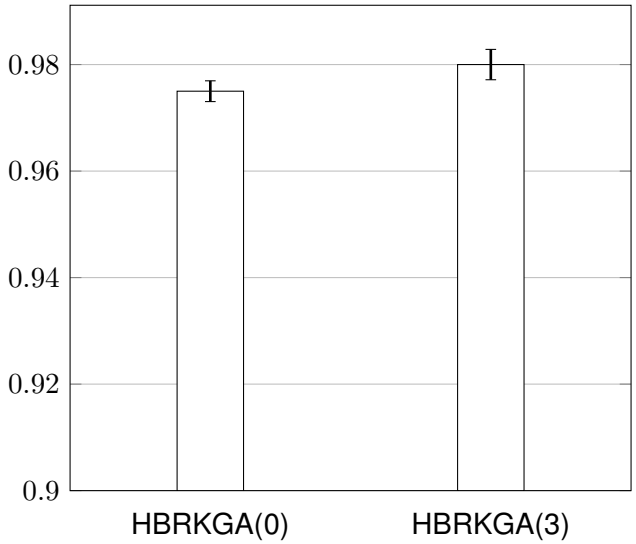


Figure 16 – HBRKGA behavior with (HBRKGA(3)) and without (HBRKGA(0)) the Random-Walk component. HBRKGA(3) reached the mean F_1 metric for 10 runs 0.98 ± 0.00285 while HBRKGA(0) reached 0.975 ± 0.00195.

Chapter VI Conclusion

VI.1- Retrospective Analysis

In this work we presented a new evolution-based approach to calibrate hyperparameters in a machine learning context, more specifically in an MLP ANN. HBRKGA, the proposed hybrid method, combines a genetic algorithm (BRKGA) with a Random-Walk technique, with the goal of finding higher quality hyperparameter configurations. We performed several experiments in the context of artificial neural networks for solving classification problems.

The experiments conducted in this work comparing HBRKGA to other approaches (Grid Search, Random Search, Bayesian Optimization and CMA-ES), HBRKGA produced better average F_1 values (measured in a separate validation dataset) in 6 out of 8 datasets. By using the Wilcoxon test, we found HBRKGA to be statistically significantly better than the other methods in three datasets while being highly competitive in the other datasets.

The second measurement during the experiments was in relation to the execution time of each strategy. The results show that Random Search and Grid Search provide the shortest average execution time compared to HBRKGA, CMA-ES and Bayesian Optimization. The Bayesian Optimization showed by far the longest average processing time.

We also performed an ablation study to assess the impact of the Random-Walk component. In particular, we compared the full-blown variant of HBRKGA with the one in which we removed the Random-Walk component. It was possible to observe a statistically significant difference in the average F_1 values, with the same experimental conditions for both variants. We conclude that the Random-Walk component, albeit simple to implement, is a crucial part of HBRKGA, allowing it to inherit nice properties of Random Search already identified in previous work [Bergstra and Bengio, 2012].

VI.2- Contributions

The experiments applying the proposed HBRKGA strategy to the hyperparameter optimization problem proved to be a good alternative to more conventional methods such as Grid Search, Random Search and Bayesian Optimization. With this, the availability of source code can help to provide an option to these methods with an acceptable processing time to generate the solutions.

All source code and datasets used are publicly available. The source code from the HBRKGA strategy used in the validation experiments is publicly available at `https://github.com/MLRG-CEFET-RJ/HBRKGA`. Also, all datasets used in the experiments of this work (MNIST and variations, rectangles, COSMOS and Fashion-MNIST) can be downloaded at `https://doi.org/10.5281/zenodo.4252922`.

VI.3- Future Work

There are several ways to continue the work we started in this work. There are several machine learning algorithms that also need hyperparameter tuning to perform an efficient task, such as Support Vector Machine (SVM) and Random Forest. Therefore, we plan to investigate the application of HBRKGA to other machine learning methods and tasks.

This work explored the tuning of hyperparameters specifically in fully connected neural networks. We will investigate the use of HBRKGA in tuning hyperparameters of particular ANN architectures, such as convolutional neural networks and recurrent neural networks.

We will also investigate new alternatives to perform the perturbation in HBRKGA instead of Random-Walk; one possibility is to add the points generated and evaluated in HBRKGA to Bayesian Optimization, using these previously known solutions for new acquisitions during the Gaussian Process.

Another point to be studied is the use of the proposed strategy in the general AutoML context. AutoML has the objective of reducing the inputs of a user as a whole in the

machine learning process. This reduction of inputs is not only restricted to hyperparameter optimization but also points such as the automatic selection of the best preprocessing technique in the dataset and choice of the architecture used for model generation [He et al., 2019].

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing, 6:154–160.

Bellman, R. (1961). Adaptive control processes: a guided tour princeton university press. Princeton, New Jersey, USA.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural networks: Tricks of the trade, pages 437–478. Springer.

Bengio, Y. et al. (2012). Deep learning of representations for unsupervised and transfer learning. ICML Unsupervised and Transfer Learning, 27:17–36.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Advances in Neural Information Processing Systems, pages 2546–2554.

Bishop, C. M. (2006). Pattern recognition. Machine Learning, 128:1–58.

Bottou, L. (2012). Stochastic gradient descent tricks. In Neural networks: Tricks of the trade, pages 421–436. Springer.

Bottou, L. and Cun, Y. L. (2004). Large scale online learning. In Advances in neural information processing systems, pages 217–224.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599.

Dewancker, I., McCourt, M., and Clark, S. Bayesian optimization primer. SigOpt. https://sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf.

Fadely, R., Hogg, D. W., and Willman, B. (2012). Star-galaxy classification in multi-band optical imaging. The Astrophysical Journal, 760(1):15.

Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. Neural networks, 2(3):183–192.

Gardner, M. W. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14):2627–2636.

Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. Machine Learning, 3:95–99.

Gonçalves, J. F. (2007). A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research, 183:1212–1229.

Gonçalves, J. F. and Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics, 17:487–525.

Gonzalez, P. H. and Brandão, J. (2018). A biased random key genetic algorithm to solve the transmission expansion planning problem with re-design. In 2018 IEEE Congress on Evolutionary Computation (CEC), pages 1–7. IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016a). Deep learning. MIT press.

Goodfellow, I., Bengio, Y., and Courville, A. (2016b). Deep Learning. MIT Press. http://www.deeplearningbook.org.

Han, J., Pei, J., and Kamber, M. (2011). Data mining: concepts and techniques. Elsevier.

Hansen, N. (2006). The CMA evolution strategy: A comparing review. In Towards a new evolutionary computation, pages 75–102. Springer.

Hansen, N. (2016). The CMA evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772.

He, X., Zhao, K., and Chu, X. (2019). AutoML: A survey of the state-of-the-art. arXiv preprint arXiv:1908.00709.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251–257.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). Automated machine learning: methods, systems, challenges. Springer Nature.

Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. International Journal of Artificial Intelligence and Expert Systems, 1(4):111–122.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. Journal of Machine Learning Research, 10(Jan):1–40.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Proceedings of the 24th international conference on Machine learning, pages 473–480. ACM.

LeCun, Y. (1998). The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems, pages 6389–6399.

Loshchilov, I. (2014). A computationally efficient limited memory cma-es for large scale optimization. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pages 397–404.

Loshchilov, I. and Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. arXiv preprint arXiv:1604.07269.

Machado, E., Serqueira, M., Ogasawara, E., Ogando, R., Maia, M. A., da Costa, L. N., Campisano, R., Guedes, G. P., and Bezerra, E. (2016). Exploring machine learning methods for the star/galaxy separation problem. In Neural Networks (IJCNN), 2016 International Joint Conference on, pages 123–130. IEEE.

Martinez, C., Loiseau, I., Resende, M. G. C., and Rodriguez, S. (2011). Brkga algorithm for the capacitated arc routing problem. Electronic Notes in Theoretical Computer Science, 281:69–83.

Miller, A. (1993). A review of neural network applications in astronomy. Vistas in astronomy, 36:141–161.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10), pages 807–814.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In Advanced lectures on machine learning, pages 63–71. Springer.

Rasmussen, C. E. and Williams, C. (2006). Gaussian Processes for Machine Learning. The MIT Press.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. Cognitive modeling, 5(3):1.

Samanlioglu, F., Ferrell, W. G., and Kurz, M. E. (2008). A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. Computers & Industrial Engineering, 55:439–449.

Scoville, N., Aussel, H., Brusa, M., Capak, P., Carollo, C., Elvis, M., Giavalisco, M., Guzzo, L., Hasinger, G., Impey, C., et al. (2007). The cosmic evolution survey (cosmos): overview. The Astrophysical Journal Supplement Series, 172(1):1.

Senior, A., Heigold, G., Yang, K., et al. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, pages 6724–6728. IEEE.

Simon, D. (2013). Evolutionary Optimization Algorithms. Wiley.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems, pages 2951–2959.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In International conference on machine learning, pages 2171–2180.

Snyder, L. V. and Daskin, M. S. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational Research, 174:38–53.

Snyman, J. (2005). Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms, volume 97. Springer Science & Business Media.

Spears, W. M. and De Jong, K. D. (1995). On the virtues of parameterized uniform crossover. Technical report, DTIC Document.

Toso, R. F. and Resende, M. G. C. (2015). A c++ application programming interface for biased random-key genetic algorithms. Optimization Methods and Software, 30:81–93.

Van Den Doel, K., Ascher, U., and Haber, E. (2012). The lost honour of l2-based regularization. Large Scale Inverse Problems, Radon Ser. Comput. Appl. Math, 13:181–203.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences.

Whitley, D. (1994). A genetic algorithm tutorial. Statistics and computing, 4(2):65–85.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In Breakthroughs in statistics, pages 196–202. Springer.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

Zabinsky, Z. B. (2009). Random search algorithms. Wiley Encyclopedia of Operations Research and Management Science.