TÉCNICO LISBOA

Using Artificial Neural Networks to Size Analog Integrated Circuits

João Pedro da Silva Rosa

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Nuno Calado Correia Lourenço
Prof. Ricardo Miguel Ferreira Martins

Examination Commitee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. Nuno Calado Correia Lourenço
Members of the Committee: Prof. Fábio Moreira de Passos

June 2018

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to acknowledge my supervisors, Prof. Nuno Lourenço, for all his support throughout the development of my Master Thesis and for his admirable availability, without which I probably would not have accomplished any satisfying results, and Prof. Ricardo Martins, whose insight in every discussion was fundamental to keep the work focused on its essential aspects.

I would also like to thank Prof. Nuno Horta and Prof. Helena Aidos, for their ideas and valuable contribution to the progress of this work.

I am grateful to all my colleagues and friends that motivated me to keep going, even when the future didn’t seem so bright, at times. Their ongoing support was crucial to reach another milestone in my life.

Finally, a very special word of gratitude goes to my parents, who forged the person I am today and pushed me to reach my goals, and my sister, who highly contributed to my education and helped me become a more humble and conscious person. To them I owe all of my success.

Resumo

O trabalho desenvolvido no âmbito desta dissertação enquadra-se na área científica de automação de projectos electrónicos e aborda o dimensionamento automático de circuitos integrados analógicos. Em particular, desenvolveu-se uma abordagem inovadora para automatizar o dimensionamento de circuitos usando *deep learning* e redes neuronais artificiais para aprender os padrões de dimensionamento de soluções previamente optimizadas. Em oposição a estratégias de dimensionamento clássicas baseadas em optimização, onde técnicas computacionais inteligentes são usadas para iterar sobre o mapeamento entre dimensões e *performances* de circuitos provenientes de equações de dimensionamento ou simulações de circuitos, as redes neuronais artificiais mostram-se capazes resolver o dimensionamento de circuitos analógicos integrados através de um mapeamento directo entre especificações e dimensões de aparelhos. Duas arquitecturas de redes neuronais artificiais são propostas: um modelo de Regressão e um modelo de Classificação e Regressão. O objectivo do modelo de Regressão é aprender os padrões de dimensionamento de circuitos estudados, usando as *performances* desses circuitos como *features* de entrada e as suas dimensões como *targets* à saída. Este modelo consegue dimensionar circuitos dadas as especificações para uma única topologia. O modelo de Classificação e Regressão tem as mesmas capacidades que o modelo anterior, mas consegue adicionalmente seleccionar a topologia mais apropriada e as respectivas dimensões para uma dada especificação. A metodologia proposta foi implementada e testada em duas topologias de circuitos analógicos distintas. Os resultados obtidos mostram que as redes neuronais artificiais treinadas foram capazes de estender as *performances* de circuitos para lá do conjunto de dados de treino/ validação, demonstrando que, mais do que um mapeamento a partir do conjunto de dados de treino, o modelo é na verdade capaz de aprender padrões de dimensionamento reutilizáveis.

Palavras-chave

Projecto de Circuitos Integrados Analógicos

Automação de Projecto Electrónico

Deep Learning

Redes Neuronais Artificiais

Abstract

The work presented in this dissertation belongs to the scientific area of electronic design automation and addresses the automatic sizing of analog integrated circuits. Particularly, this work explores an innovative approach to automatic circuit sizing using deep learning and artificial neural networks to learn patterns from previously optimized design solutions. In opposition to classical optimization-based sizing strategies, where computational intelligent techniques are used to iterate over the map from devices' sizes to circuits' performances provided by design equations or circuit simulations, artificial neural networks are shown to be capable of solving analog integrated circuit sizing as a direct map from specifications to the devices' sizes. Two separate artificial neural network architectures are proposed: a Regression-only model and a Classification and Regression model. The goal of the Regression-only model is to learn design patterns from the studied circuits, using circuit's performances as input features and devices' sizes as target outputs. This model can size a circuit given its specifications for a single topology. The Classification and Regression model has the same capabilities of the previous model, but it can also select the most appropriate circuit topology and its respective sizing given the target specification. The proposed methodology was implemented and tested on two analog circuit topologies. The achieved results show that the trained artificial neural networks were able to extend the circuit performance boundaries outside the train/ validation set, showing that, more than a mapping from the training data, the model is actually capable of learning reusable design patterns.

Keywords

Analog Integrated Circuits Design

Electronic Design Automation

Deep Learning

Artificial Neural Networks

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CAD	Computer-Aided Design
DT	Decision Tree
EA	Evolutionary Algorithm
EDA	Electronic Design Automation
FoM	Figure-of-Merit
IC	Integrated Circuit
MAE	Mean Absolute Error
ML	Machine Learning
MS	Mixed-Signal
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SoC	System-on-Chip
SSCE	Sparse Softmax Cross Entropy
SVM	Support Vector Machines

Chapter 1. Introduction

This Chapter provides historical background on analog Integrated Circuit (IC) design and showcases some of the challenges faced by designers when trying to come up with new automatic design tools. Furthermore, the concept of Machine Learning (ML) is introduced, and, the possibility to use the studied techniques in this branch of Artificial Intelligence (AI) to automate analog IC design is discussed.

1.1. Motivation

In recent years, electronics industry has seen a tremendous increase in the demand of more complex and highly integrated systems that must be placed in a single chip for power and packaging efficiency. In the era of portable devices, integration and power consumption matter more than ever and developers are faced with the challenge of increasing the capabilities of the systems while ensuring they can be effectively integrated in energy efficient, small and light end products. The complexity of these systems is highly associated with the trade-off between their analog and digital sections. Developing these Mixed-Signal (MS) Systems-on-Chip (SoC) constitutes a great challenge both to the designers of chips and to the developers of the Computer-Aided Design (CAD) systems that are used during the design process.

While in most MS SoCs the area occupied by digital blocks is bigger than the area occupied by analog blocks, the effort to implement the latter is considerably larger, as illustrated by Figure 1.

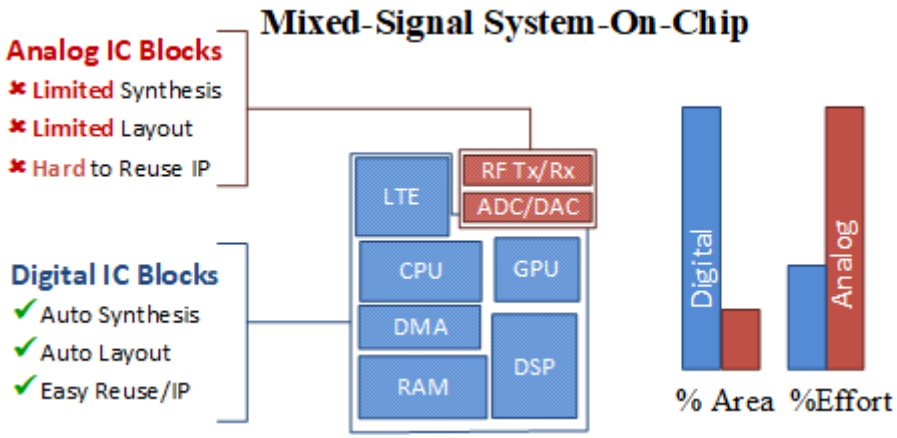

Figure 1 - Contrast between Analog and Digital blocks' area of implementation in an IC and the corresponding effort to implement them from the perspective of a designer [1].

This imbalance in design effort, as well as economic pressures, has motivated the development of new methods and tools for automating the analog design process. Despite the considerable evolution verified in the past few years, automating the circuit sizing process is still a relevant research topic. Automatic analog IC sizing tools have still a long way to go to reach the automation levels observed in their digital design counterpart [2].

Deriving from the fact that CAD tools for electronic design automation (EDA) targeting analog ICs have not yet reached a desirable state of maturity, human intervention during all phases of the design process is very much a given. This is a necessity because the exploration of the solution space is done manually. Even considering the use of circuit simulators, layout editing environment and verification tools, the design flow will still be very time-consuming and error-prone [3]. This is aggravated when the number of devices increases, not because of the quantity per se, but because of the number of interactions between them. Other problems may arise specifically when designing an analog circuit. Due to the nature of the signals being handled, these circuits are extremely sensible to parasitic disturbances, crosstalk, thermal noise, etc., and the variety of schematics and diversity of devices' sizes and shapes is enormous [1].

Therefore, this discrepancy between analog and digital automation is due to the more knowledge intensive, more heuristic and less systematic type of approach required for effective analog IC design.

All things considered, designing an analog block takes a considerably longer time than designing a digital one, and it is crucial that the designer is experienced and knowledgeable [1]. These adversities can be summarized by the following three reasons:

- Lack of systematic design flows supported by effective EDA tools;
- Integration of analog circuits using technologies optimized for digital circuits;
- Difficulty in reusing analog blocks, since they are more sensitive to surrounding circuitry, and environmental and process variations than their digital counterpart [1].

Keeping up with the demands brought by the technological revolution is of paramount importance. The algorithms and techniques developed in the last few decades offer a plethora of solutions to solve new problems that arise with the vicious cycle of innovation in the technological markets, but designers need to be constantly adapting and improving their own automation tools. And to adapt is to search for new options that might not seems obvious at first glance.

1.2. Using Machine Learning for IC Analog Sizing

ML is the process in which a computer improves its capabilities through the analysis of past experiences. It is now commonly accepted by developers of AI systems that, for a vast number of applications, training a system by showing it examples of desired input-output behaviour can be easier to program than to anticipate all possible responses for all inputs [4]. Nevertheless, the process of automatically improving behaviours through experience can be costly and time consuming, especially when large amounts of data must be interpreted and processed. There is a wide variety of techniques that have evolved through the years that can be used to overcome these problems and choosing the correct one can sometimes be a challenging task itself. While the initial goal when processing data might be to find a recognizable pattern, ML studies go further and attempt to build algorithms that can learn from and make predictions on data. The more data is available, the better the algorithm will perform. With the advent of *big data*, where users have now sufficient computational power to collect

more data than ever before, ML took another step forward, proving sceptics that there is a bright future ahead for this field of study. One that can even change the way we see the world.

The most recent effects of ML can be felt within AI, in fields such as computer vision, speech recognition, language processing, medical diagnosis, economics and search engines. A good example of an everyday task where one can come across ML would be searching for a book or a song name in Google. Based on that search, Google will attempt to build a profile and place specific ads that target your interests. Similarly, Facebook collects massive amounts of data from its users every day, building models from the posts they make, the news they share and the pages they like. While Facebook tells us that these models are used to improve our experience using this social network, the truth is some of the data can be used unduly and even influence elections [5]. Netflix meticulously evaluates the movies and TV-shows you watch so it can build a list of recommendations based on the shows you rated the highest. Uber not only uses ML to improve their customer experience (by training their drivers to offer customers a more comfortable experience), but also to build their own autonomous car using a model based on Neural Networks. Less malicious uses can be felt across computer science and across a range of industries concerned with data-intensive issues, such as drug testing in medicine, online fraud detection, and the control of logistics chains [4].

ML is slowly taking over our lives. We just don't know it yet at a conscious level. Most of the devices and services we use employ some form of a learning algorithm. Companies relentlessly mine data from their users to sell more and better products. Politicians use data from voters to manipulate their perceptions. Demand for data is at an all-time high.

Many are the available techniques to build these models. Some of the most popular nowadays are Artificial Neural Networks (ANNs). This technique has been cyclically picked-up and abandoned over the years, but a new trend emerged recently, called *deep learning*, where much more complex networks are employed, yielding very interesting results in image processing [6], for example. Bayesian Networks are used extensively for speech recognition [7]. Support Vector Machines can be used to classify images or to recognize handwriting [8]. Decision Trees have been used for drug analysis [9]. And Evolutionary Algorithms (EAs) have been used in a number of applications, namely in analog IC design [10].

The fact that ML methods have been developed to analyse high throughput experimental data in novel ways makes them an attractive approach when tackling more demanding and complex problems. Nevertheless, as powerful as these techniques may seem at first sight, there are some caveats when trying to use them. The problem lies in the amount of the data we are able to collect from the situations we are trying to analyse and extract knowledge from. ML algorithms are very dependent on the number of available examples. If the problem we have at hand is hard to characterize and offers a low amount of explorable data, it will be difficult to build a model from which we can extract a set of rules that will be used to obtain the best possible generalization. Generalization is one of the key goals of every ML technique. It means that models should be trained in order to correctly classify unseen data, while avoiding being too specific for the examples that were used to train the algorithm (a problem commonly known as *overfitting*).

ML is a vast area of research and offers many solutions. Regarding automatic analog IC sizing, the questions posed are: which parts of the sizing process can benefit the most from ML methodologies? And after identifying them, what are the most appropriate techniques we can choose from the ones available?

Some approaches have been made to use ML algorithms in analog IC sizing. Particularly, ANNs have been employed in this field of study (these applications are further discussed in Chapter 2). Analog IC design is still very dependent on the intervention of the designer on all stages of the analog design flow. Compared to their digital IC design counterpart, automatic tools are still scarce. Therefore, applying knowledge coming from AI seems like a very auspicious way to improve analog IC design, since there are so many techniques to pick from. Automatic design of analog IC can be divided into two categories [11]:

1. Automatic design whose task is to find devices' sizes, such as widths and lengths of resistors and capacitors, for a given topology from a set of specifications;
2. Automatic design whose task is to find a circuit topology and determine the element values from the set of specifications.

The approaches suggested by these categories can be used as a starting point to envision models for analog IC sizing using ML.

1.3. Goals

The primary goal of this work is to accelerate the process of analog IC sizing through ML methodologies. This work is a development branch that runs in parallel with a genetic algorithm, the non-dominated sorted genetic algorithm II (NSGA-II), which is one of the core blocks in the optimization kernel of the Analog Integrated circuit Design Automation environment (AIDA), developed in the Integrated Circuits Group at Instituto Superior Técnico. The main objectives for this work are detailed below:

- Improve the automation of analog ICs by reutilizing data collected from previous circuit projects;
- Provide an overview of ML methodologies. These will then be assessed to estimate their applicability to analog IC automation;
- Create and implement functional and automated models that can learn patterns from circuit projects, and can generalize that knowledge to new projects;
- Apply the developed models in analog IC projects in order to prove that the models can indeed learn reusable knowledge.

1.4. Achievements

During the development of this work, the following achievements were obtained:

- Creation of ANN models that can size a circuit given its specifications for a single topology;

- Creation of ANN models that can both select the most appropriate circuit topology and its respective sizing given the target specification;
- Acceptance of a scientific paper on the 15th International Conference on Synthesis, Modelling, Analysis and Simulation Methods and Applications to Circuit Design, SMACD 2018.

1.5. Document Structure

The document is organized as follows:

- Chapter 2 presents state-of-the-art on ML models and dissects the advantages and disadvantages of several techniques. Existent approaches for analog IC automation using ML techniques are also discussed. Finally, an assessment is made to evaluate the applicability of the discussed methodologies for analog IC automation;
- Chapter 3 extends ANN discussion started in Chapter 2, detailing the structure of this model and analyzing the algorithms behind it;
- Chapter 4 presents two different architectures that attempt to accelerate the process of analog IC sizing: a Regression-only Model and a Classification and Regression Model. Model structure, used datasets and other model parameters are discussed;
- Chapter 5 describes results obtained with the proposed architectures and showcases the capabilities of the implementation;
- Chapter 6 presents conclusions drawn from this work and outlines future recommendations concerning the use of automatic learning as a tool to automate analog IC design.

Chapter 2. State-of-the-Art on Machine Learning Techniques

This chapter presents the state of the art on ML techniques. First, we explore ML by categorizing existing methods into 5 tribes of knowledge. Advantages and disadvantages of each tribe will be discussed to have a clear picture of their strengths and weaknesses in certain scenarios. Then, we will explore existing work where ML techniques were successfully applied to analog IC sizing. Finally, the strengths of each Tribe are weighed in to choose which technique will be more fitting to implement in this work, while also considering existing implementations of ML in analog IC sizing.

2.1. Machine Learning Overview

The possibility of assigning tasks to machines to avoid repetitive actions, as a theoretical formulation, has been postulated by many mathematicians through the ages. The study of mathematical logic often demands exhausting calculations that cannot be solved in a fast and efficient way, so it would only make sense to build models that would automatically execute these steps. Many questions rose from this proposition, not only in the field of mathematics but also in philosophy. The essential one being – can an artificially intelligent machine think and act like the human brain?

The answer to this question may be found in Machine Learning. While this field of study borrows its foundations from the concepts of early AI research, its approach on problem solving is more focused on statistical knowledge. This dissonance was what caused Machine Learning to branch out from AI and reorganize as a separate field of research, since AI researchers were more concerned with automatically constructing expert systems [12] that were modelled using heavily symbolic language. Machine Learning researchers abandoned this in favour of more practical methodologies that could produce more tangible results, using models from statistics and probability theory.

It was Thomas Bayes who initially proposed theorems of probability theory on his Essay on Probability Theory (1763). These concepts, such as conditional probability, would later result in what is now called the Bayes' Theorem (1812) and would be of immense of importance in the formulation of some early Machine Learning techniques, such as Naive Bayes or Markov Chains. The early movement in Machine Learning was also characterized by an emphasis on symbolic representations of learned knowledge, such as production rules, decision trees and logical formulae [12]. The research continued and other discoveries were made, with the invention of first Neural Network machine (1951) being one of the most important. However, it wasn't until Frank Rosenblatt invented the perceptron (1957), a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector, that ANN began to receive more attention from other researchers. In 1986, the backpropagation process was proposed and it furthered ANN's development. A timeline comprised of Machine Learning history highlights is illustrated in Figure 2.

Since then, many other discoveries have been done in this field, but it wasn't until the turn of the century that Machine Learning became a commercial success. It was only in the 1990s that ANN and

Support Vector Machines (SVM) became widely popular, as the available computational power started to increase. However, true groundbreaking developments would only be possible a few years later.

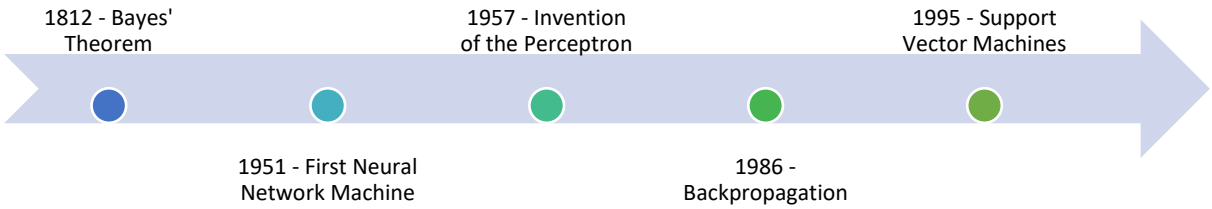

Figure 2 - Machine Learning history highlights timeline.

Despite its practical and commercial successes, Machine Learning remains a young field with many underexplored research opportunities. Some of these opportunities can be seen by contrasting current Machine Learning approaches to the types of learning we observe in naturally occurring systems such as humans and other animals, organizations, economies, and biological evolution. For example, whereas most machine learning algorithms are targeted to learn one specific function or data model from one single data source, humans clearly learn many different skills and types of knowledge, from years of diverse training experience in a simple-to-more-difficult sequence (e.g., learning to crawl, then walk, then run) [4].

Machine Learning tasks can be broadly classified into two separate categories [13]:

- **Supervised Learning**, where a model is trained to generalize rules based on example inputs and corresponding desired outputs. Once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into *classification* tasks (Figure 3.a) illustrates an example) and *regression* tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities [13];
- **Unsupervised Learning**, where the algorithm is given the task to find rules by itself based on input data, without reference to any target label. These models include tasks such as *clustering* (Figure 3.b) illustrates an example) and *dimensionality reduction.* Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data [13];

In addition, a third category falls in between the above-mentioned categories called semi-supervised learning [13], where the model is fed with an incomplete dataset in which some target labels are missing.

To further clarify, classification tasks are models that predict labels of unseen data based on previously labeled examples (e.g. a spam classifier that that uses examples from e-mails previously classified as spam), while regression tasks are models that predict continuous labels (e.g. a multi-variate analysis on a person's biometric data). Clustering algorithms are models that correctly detect and identify

distinct groups in the data (e.g. labeling classes of animals based on physical characteristics) and dimensionality reduction algorithm are models that detect and identify lower-dimensional structure in higher-dimensional data (e.g. principal component analysis).

a) b)

Figure 3 - a) An example of a supervised learning algorithm, where a Support Vector Machines technique was applied to classify the data into two different classes; b) an example of an unsupervised learning algorithm, where a k-means technique was used to classify data into 4 different classes [13].

In 2015, one of the world's most renowned Machine Learning researchers and a professor at the University of Washington, Pedro Domingos, proposed another type of classification [14], where Machine Learning techniques can be segmented into five different Tribes, as illustrated in Figure 5. Domingos argues that the three main sources of human intelligence (evolution, experience, and culture) will be replaced by computers, since the majority of knowledge is going to be computerized [14]. Having that in mind, it will be of paramount importance to understand the proper ways in which computers extract knowledge from the physical world around them. The best way to start is to categorize each methodology into separate camps, or 'tribes', as Domingos chose to call them. Each Tribe has its own set of core methods and philosophy, as well as an algorithm that can pursue categorically different kinds of Machine Learning. Those Tribes are the symbolists, bayesians, connectionists, evolutionaries and analogizers [14] (a summary of these Tribes' characteristics can be found in Figure 4).

For **symbolists**, all knowledge can be reduced to manipulating symbols [14]. Instead of starting with an initial premise and looking for a conclusion, inverse deduction starts with some premises and conclusions, and essentially works backwards to fill in the gaps. This is done so by deducing missing rules that fit the pre-established conclusions (much like solving a puzzle). Their favored algorithms are rules and decision trees.

Bayesians are concerned above all with uncertainty [14]. This type of learning evaluates how likely a hypothesis will turn out to be true, while considering *a priori* knowledge. Different hypotheses are compared by assessing which outcomes are more likely to happen. This is called probabilistic inference. This tribe's favored algorithms are Naïve Bayes or Markov Chains.

For **connectionists**, learning is what the brain does, and so the goal is to reverse engineer it [14]. Models belonging to this tribe attempt to build complex networks comprised of nodes that resemble neurons from the human brain, and adjust the strength of each connection by comparing obtained outputs with desired ones. Their favored algorithm is backpropagation, a method commonly applied to Neural Networks.

Evolutionaries believe that the mother of all learning is natural selection [14]. In essence, an EA is a meta-heuristic optimization algorithm used in AI that uses mechanisms based on biological evolution. Beyond their ability to search large spaces for multiple solutions, these algorithms are able to maintain a diverse population of solutions and exploit similarities of solutions by mechanisms of recombination, mutation, reproduction and selection [15]. A fitness function is responsible for analysing the quality of the proposed solutions, outputting values that will then be compared to a predefined cost or objective function, or a set of optimal trade-off values in the case of two or more conflicting objectives. This tribe's favored algorithm is Genetic Programs.

For **analogizers**, the key to learning is recognizing similarities between situations and thereby inferring other similarities [14]. Learning comes down to building analogies between available data. If you have an Amazon account, new products will be recommended to you based on your purchase history, which is one of the most powerful technologies the company implemented in their business (one third of Amazon's revenue is based on recommendations [16]). Their favored algorithm is Support Vector Machines.

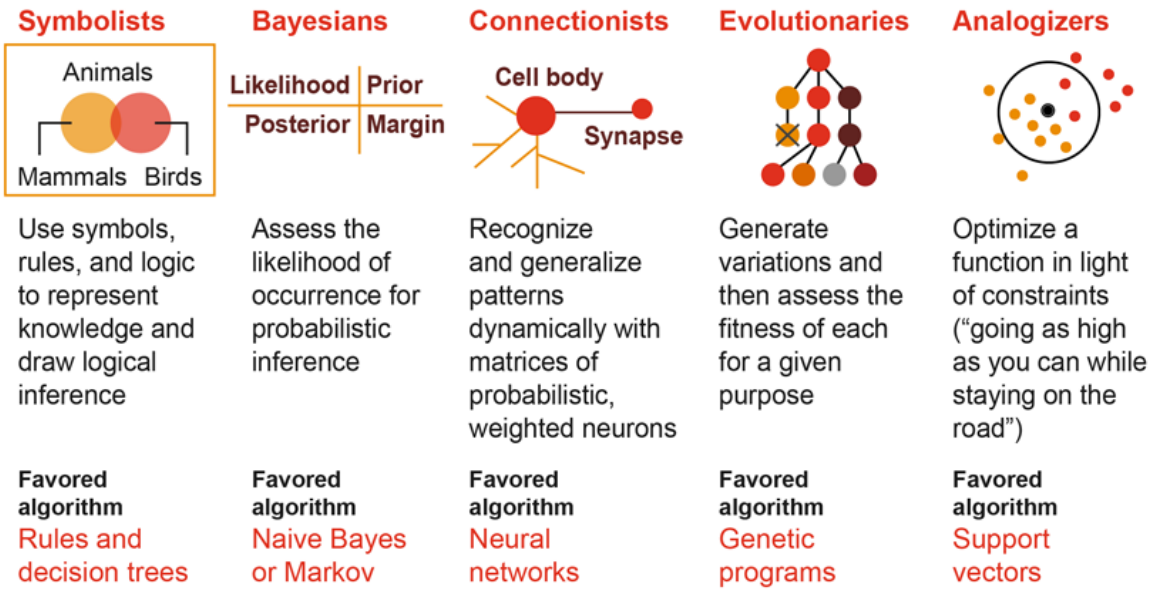

Figure 4 - The Five Tribes of Machine Learning, according to Pedro Domingos [14] [17].

All five tribes have something unique to offer. In theory, a master algorithm (hypothesized by Domingos in his book "The Master Algorithm") would be a combination of all five angles. The key is to understand what type of data we have at hand and figure out which methodology will best suit our needs. Different problems require different approaches, and what we will try to understand in the next sections is what are the strengths and weaknesses of each tribe and which technique will be the most

appropriate in the context of electronic design automation. Interestingly enough, the algorithm we are trying to find is one that will be responsible for the acceleration of an already implemented Genetic Algorithm, which is part of AIDA, an analog integrated circuit design automation environment. After analyzing each tribe, the structure of the data we have available will also need to be assessed, and only then will we be able to choose the best technique.

The next five sections will give a more detailed description about the capabilities of each tribe of Machine Learning and provide an overview of each tribe's favored algorithm.

2.1.1. Symbolists

The act of decision making is often plagued with a variety of decisions to make and consequences to judge, with many different outcomes to analyse that may be dependent on a series of previous decisions which, initially, may have not seemed related. Consequences are not only dependent on the person making the decisions that caused them, but also on possible external factors or events (e.g. a person buying stock should consider the eventuality of a stock market crash). Decision makers must consider that actions and reactions are knotted in such a way that their relations should be carefully mapped in order to make the most advantageous decision. A good way to do is to use a Decision Tree (DT) model, which helps structuring such decision-making problems.

According to Kamiński et al [18], a DT, an algorithm that belongs to the Symbolists tribe [14], is a tool used in decision analysis that is constructed using a directed graph $G = (V, E), E \subset V^2$. The tree is comprised of a set of nodes V which are split into three disjoint sets $V = \mathrm{D} \cup \mathrm{C} \cup \mathrm{T}$. Each set represents decision, chance, and terminal nodes, respectively. For each edge $e \in E$, the first element (parent node) can be denoted by $e_1 \in V$, while $e_2 \in V$ is used to denote its second element (child node).

As mentioned before, a sequential decision problem is comprised of three types of nodes. These nodes represent different stages of the decision tree flow:

- In a **decision node**, the decision maker selects an action. An action is represented by one of the edges having the node in question as the parent;
- In a **chance node**, one of the edges stemming from it (denoted as *reaction*) is selected randomly;
- **Terminal nodes** mark the end of a sequence of actions/ reactions in the decision problem.

A DT model is used to visualize sequential decision problems by mapping the degree of uncertainty associated with each outcome. These are especially useful when the decision makers need to make a long sequence of decisions or when they need to weigh which outcome will yield more favourable results in the long run. Two parameters should be determined: the probability of an event or outcome and the payoffs of the consequences of a decision. The main goal of a DT is to determine the path which yields the greatest payoff or the smallest loss.

In a DT, decision nodes are typically represented as squares, chance nodes as circles, and terminal nodes as triangles. In Figure 5 we can observe a sample decision tree where $\mathrm{D} = \{\mathrm{d}\}, \mathrm{C} = \{\mathrm{c1}, \mathrm{c2}\}, \mathrm{T} =$

$\{t1, t2, t3, t4\}$, and $E = \{(d, c1), (d, c2), (c1, t1), (c1, t2), (c2, t3), (c2, t4)\}$. This illustration demonstrates how an investment decision model can be represented with a DT. The structure of this specific tree includes one decision node, two chance nodes and four terminal nodes. In the decision node (red square), the decision maker has the possibility of selecting exactly one of the branches emanating from the node. Each of the two chance nodes (yellow circles) has two random outcomes, whose probabilities are shown below the emanating branches. The terminal nodes (green triangles) represent the outcome of the sequence of action/ reactions, starting from the root node and ending in the particular terminal node. An edge is the combination of the parent and the child node.

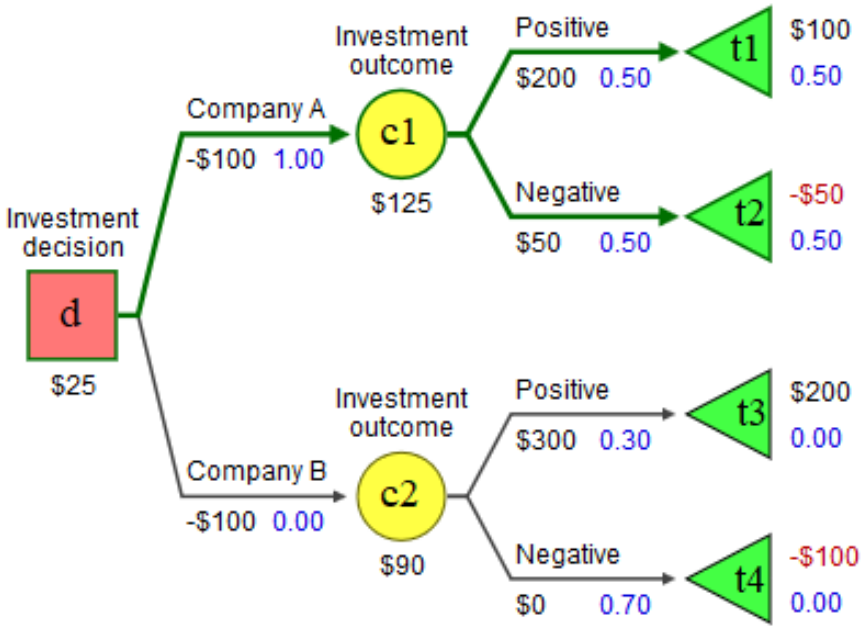

Figure 5 - An example of a decision tree, where tree paths and nodes illustrate an investment decision model, built with the SilverDecisions© application [18].

As already mentioned, two parameters should be determined when building DTs: one denoting probabilities, $p: \{e \in E: e1 \in C\} \rightarrow [0,1]$ and the other denoting payoffs, $y: E \rightarrow R$,. With this formalism we make the following assumptions: payoffs are defined for all edges and may follow both actions and reactions; probabilities are defined only for edges stemming from chance nodes. We allow zero probabilities in the general definition of a DT, which simplifies the technicalities in subsequent sections. In Figure 5, we have $p(c1, t1) = 50\%, p(c1, t2) = 50\%$ and $y(d, c1) = -100, y(d, c2) = -100$, $y(c1, t1) = 200, y(c1, t2) = 50, y(c2, t3) = 300, y(c2, t4) = 0$.

The DT shown in Figure 5 was built with the SilverDecisions© application [18]. The optimal path which yields the most favourable results is highlighted in green. Under each decision and chance node, there is the expected payoff, which is calculated by adding all the expected payoffs from the node's subtree. Terminal nodes have two values to their right: the aggregated payoff of the whole tree path, at the top, and the probability of the tree path ending in a particular terminal node. After the algorithm is computed to discover which is the most advantageous solution, only probabilities associated to

terminal nodes from the optimal path remain. Lastly, the values on each edge represent the payoff of selecting the given edge and the probability of the edge being chosen from its parent node.

When growing trees, some algorithms may be used to determine which is the best decision when choosing a certain path. These methods apply a greedy strategy, in a sense that the decision is always taken with the intent of achieving the highest payoff or lowest loss. The decision criteria for selecting the best choice is often based on the measures of the degree of impurity of child nodes. Node impurity measures include:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t) \quad (1)$$

$$Gini(t) = -\sum_{i=0}^{c-1} [p(i|t)]^2 \quad (2)$$

$$Classification\ error(t) = 1 - \max_i [p(i|t)] \quad (3)$$

Random Forests are an alternative supervised learning algorithm that also belong to the Symbolists tribe and are built from Decision Trees. The *forest* the algorithm builds an ensemble of Decision Trees, meaning that it relies on aggregating the results of an ensemble of simpler estimators. One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. They're also able to correct Decision Tree's habit of overfitting to their training set.

2.1.2. Bayesians

According to Kevin P. Murphy [19], a classifier is a function f that maps input feature vectors $x \in \mathcal{X} = \mathbb{R}^D$ to output class labels $y \in \{1, \dots, C\}$. $\mathcal{X}$ is the feature space vector, which is a vector of D real numbers, while C is the number of classes.

A Bayesian classifier, belonging to the tribe of the Bayesians [14], is based on probabilistic theory and aims at implementing methods that return a conditional probability of type $p(y|x)$. There are two main ways to do this. The first is to directly learn the function that computes the class posterior $p(y|x)$. This is called a discriminative model, since it discriminates between different classes given the input. The alternative is to learn the class-conditional density $p(x|y)$ for each value of y and the class priors $p(y)$, and then computing the posterior by applying the Bayes rule:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_{y'=1}^{C} p(x|y)p(y')} \quad (4)$$

This is called a generative model, since it specifies a way to generate the feature vectors x for each possible class y. Specifying this generative model for each label is the main piece of the training of a Bayesian classifier. The general version of such a training step is a very difficult task, but we can make it simpler using some simplifying assumptions about the form of this model.

One simpler alternative to generative and discriminative learning is to dispense with probabilities altogether, and to learn a function, called a discriminant function, that directly maps inputs to outputs:

$$f(x) = \hat{y}(x) = arg\max_{y} p(y|x) \tag{5}$$

This is perhaps the most popular approach to classification. Another alternative is called the Naive Bayes assumption, in which all the features are conditionally independent given the class label:

$$p(x|y = c) = \prod_{i=1}^{D} p(x_i|y = c) \tag{6}$$

Naive Bayes models are a group of extremely fast and simple classification algorithms that are often suitable for very high-dimensional datasets. Because they are so fast and have so few tuneable parameters, they end up being very useful as a quick-and-dirty baseline for a classification problem [13]. In a practical context, the Naïve Bayes assumption may not hold true, since features are usually dependent. For example, a supervised learning Naive Bayes model would assume that the features that characterize a fruit, such as colour or shape, all have an independent contribution to the probability of calculating whether a fruit is a strawberry, a banana or an orange. However, the resulting model is fast and easy to train and would produce very satisfying and efficient results. Essentially, the dependence distribution; i.e., how the local dependence of a node distributes in each class, evenly or unevenly, and how the local dependencies of all nodes work together, consistently (supporting a certain classification) or inconsistently (cancelling each other out), plays a crucial role [20]. Therefore, no matter how strong the dependences among attributes are, Naive Bayes can still be optimal if the dependences distribute evenly in classes, or if the dependences cancel each other out.

In the case of continuous data, it is usually assumed that the values that belong to each class that is intended to be classified are distributed according to a Gaussian distribution. Hence,

$$p(x|y = c, \theta_c) = \prod_{i=1}^{D} \mathcal{N}(x_i|\mu_{ic}, \sigma_{ic}) \tag{7}$$

where we just have to estimate a number of Gaussian parameters (μ_{ic}, σ_{ic}) equal to $C \times D$.

2.1.3. Connectionists

Neural Network learning methods, which belong to the tribe of Connectionists [14], provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions. For certain types of problems, such as learning to interpret complex real-world sensor data, ANNs are among the most effective learning methods currently known. The study of ANNs has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons. In rough analogy, ANNs are built out of a densely interconnected set of simple units, where each unit

takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output, which may become input to other units [21].

The most common structure of a neural network one can come across is shown in Figure 6. It consists of three layers of processing units which are also referred as nodes or neurons – the **Input Layer**, the **Hidden Layer** and the **Output Layer**. These types of ANN are called multi-layer perceptrons.

After feeding neurons with input values in the Input Layer, the values are processed within the individual neurons of the layer. In general, each neuron receives the same input - one real value from every neuron at the previous layer - and produces one real value, which is passed to every neuron at the next layer. The arrows indicate connections nodes of different layers. First the output values of the input nodes are passed on to the hidden nodes. These values obtained as inputs by the hidden nodes are again processed within them and passed on to either the Output Layer or to the next hidden layer (ANNs can have more than one hidden layer).

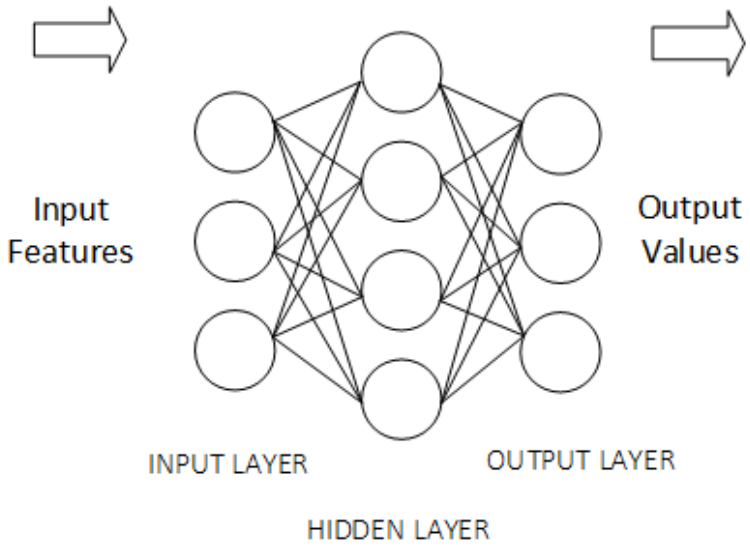

Figure 6 – Basic structure of an Artificial Neural Network.

Each connection has an associated parameter indicating the strength of this connection, the so-called *weight*. By changing the weights in a specific manner, the network can learn to map patterns presented at the input layer to target values on the output layer. The description of this procedure, by means of which this weight adaptation is performed, is called *learning* or *training algorithm*. By comparing the input data with the output, the measure of the error will determine which connections within the network are strengthened and which ones are weakened.

In Chapter 3, ANNs are studied in greater detail.

2.1.4. Evolutionaries

The NSGA-II kernel is an evolutionary optimization structure which belongs to the tribe of Evolutionaries [14] and whose methodologies are inspired by natural evolution processes. This algorithm is designed to solve the general multi-objective multi-constraint optimization problem defined as:

$$\begin{aligned} find\ x\ that\ minimize\ f_m(x) \qquad & m = 1,2,\dots M \\ subject\ to\ g_j(x) \geq 0 \qquad & j = 1,2,\dots J \\ x_i^L \leq x_i \leq x_i^U \qquad & i = 1,2,\dots N \end{aligned} \tag{8}$$

where, x is a vector of *N* design variables, $g_j(x)$ one of the *J* constraints and $f_m(x)$ one of the *M* objective functions. The number of design variables defines the space order, while the variable ranges will define the size of the search space. The definition in (8) is a general introduction that makes way for new optimization methods to be explained, and is used to create an abstraction layer between the optimization method and the circuit being optimized [1].

The genetic algorithm starts by generating an initial population of individuals (randomly or using other sampling methods), which are denominated by parents, each one representing a different solution. What sets NSGA-II apart from other multi-objective algorithms is the use of Pareto dominance to sort the multi-objective solutions. According to Lourenço et al. [1], given the multi-objective nature of the sizing method, the optimizer's output is not one solution but a set of solutions all compliant with the design specifications, i.e., a set of Pareto non-dominated solutions or Pareto front. Pareto dominance states that one point in the solution space, A, is not dominated by another point, B, if $\exists m: f_m(A) < f_m(B)$. Figure 7 depicts a Pareto front, illustrating the Pareto dominance concept, where solutions A and B are non-dominated and both dominate solution point C.

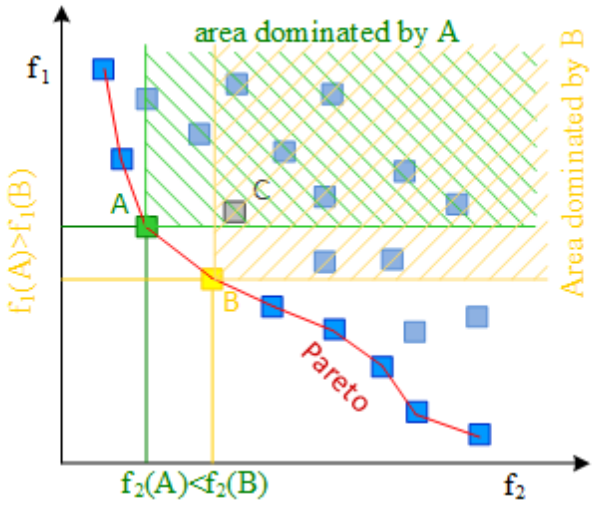

Figure 7 - Pareto Front illustrative example [1].

To ensure a good exploration of the search space, the initial population should be diverse in terms of genetic material.

New individuals are obtained from the current population by the application of the genetic operators: mutation and crossover. The crossover operation uses two population elements (parents) to generate the new elements (children or offspring), combining randomly selected sets of information from each of the parents into the offspring. The mutation is a random change in one individual's genetic information.

The mutation operator introduces new information which helps escaping from local minima and increases the diversity in the population, whereas, the crossover recombines pieces of information already present in the population. The new individuals' fitness is then computed and ranked together with the parents.

Fitness can be described as the evaluation of how good the candidate solution is, which is done by assigning a rank to each individual. The fittest individuals are selected as the new parents and the less fit discarded. The process is repeated until convergence or an ending criterion is reached. The ranking is made by using the Pareto dominance method. The rank of the individuals is set by finding the non-dominated fronts iteratively. The rank 1 individuals are the ones that are not dominated by any other. These individuals are then removed from the population and the process is repeated for the next ranks until there are no more individuals in the population. The individuals with lower rank dominate the ones with higher rank. The cycle of an EA is summarized in Figure 8.

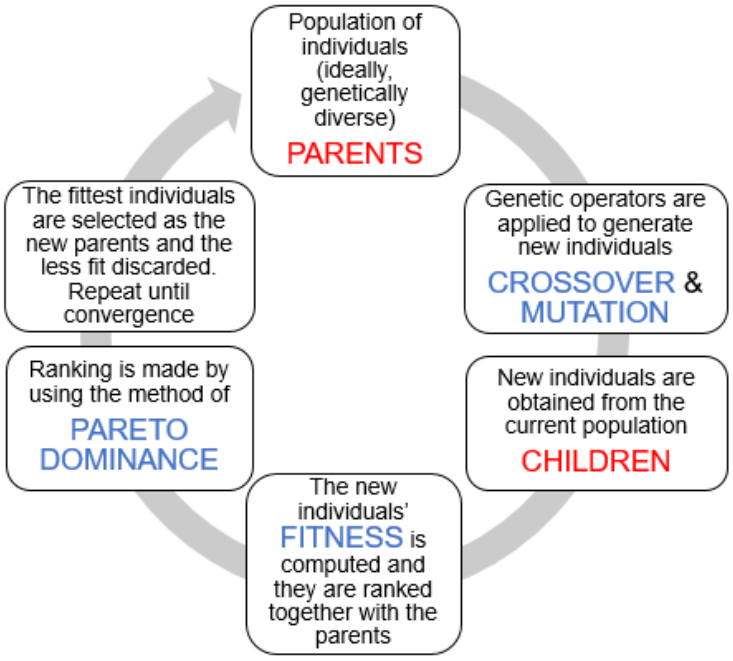

Figure 8 - Evolutionary algorithm loop.

2.1.5. Analogizers

Support Vector Machines (SVM) are models of supervised learning, which fall in the tribe of the Analogizers [14]. When it was first introduced by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963, this algorithm was only used for linear classification. However, in 1992, Bernard E. Boser et al. suggested a new approach which could be able to perform non-linear classification through a so-called kernel trick [22]. A linear classification problem uses the concept of large-margin separation while a non-linear one uses kernel functions.

Assuming we want to classify a set of objects in two separate classes, let x denote a vector with M components x_j, for $j = 1, \dots, M$ i.e. a point in a M-dimensional vector space. The notation x_i will

denote the i[th] vector in a dataset $\{(x_i, y_i)\}_{i=1}^{n}$, where y_i is the label associated with x_i. The objects x_i can be called patterns or inputs [23].

Taking the inner product, $\langle w, x\rangle = \sum_{j=1}^{M} w_j\, x_j$, between two vectors into account, a linear classifier is based on a linear discriminant function formulated as follows:

$$f(x) = \langle w, x\rangle + b \tag{9}$$

The discriminant function $f(x)$ assigns a score for the input x and is used to decide which class it will be classified into. The vector w is known as the weight vector, and the scalar b is called the bias. In two dimensions, the points satisfying the equation $\langle w, x\rangle = 0$ correspond to a line through the origin, while in three dimensions it corresponds to a hyperplane. The bias b translates the hyperplane with respect to the origin.

The hyperplane in Figure 9 divides the space into two half spaces according to the sign of $f(x)$, which indicates the side of the hyperplane a point will be located on by a simple rule of thumb: if $f(x) > 0$, then the point will fall into the positive class, otherwise into the negative. It can be described by $w \cdot x + b = 0$, where:

- w is normal to the hyperplane;
- $\frac{b}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin.

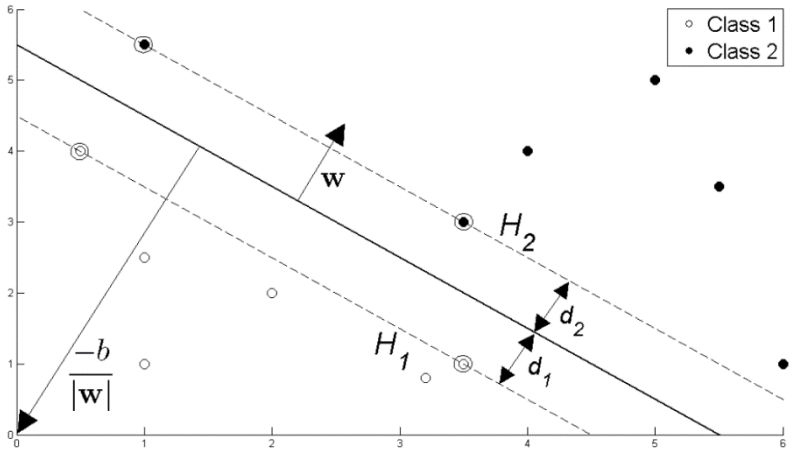

Figure 9 - Hyperplane through two linearly separable classes [24].

Support Vectors are the examples closest to the separating hyperplane and the aim of SVM is to orientate this hyperplane in such a way as to be as far as possible from the closest members of both classes [24].

The boundary between regions classified as positive and negative is called the decision boundary of the classifier. The decision boundary defined by a hyperplane is said to be linear because it is linear in the input. A classifier with a linear decision boundary is called a linear classifier. One example of an SVM linear classifier is the Large Margin classifier, which can be applied when the data is linearly

separable. This classifier can be formulated as an optimization problem, since it aims to maximize the distance between each classified point to the hyperplane of separation between classes:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$subject\ to\ y_i(\langle w, x\rangle + b) \geq 1, \text{for } i = 1, \ldots, n \tag{10}$$

In terms of non-linear classification, the classification algorithm is similar to the one used in linear classification, except that every inner product $\langle w, x\rangle$ is replaced by a kernel function, with the most popular kernels being the Polynomial, the Gaussian and the Radial [23].

2.2. Assessing the Different Tribes of Knowledge

Each Machine Learning Tribe of knowledge is suited to different tasks. Some problems may be solved using only one technique from a specific Tribe or by a combination of different techniques belonging to different Tribes. The size and structure of the data are important details to analyse early. Assessing which type of learning problem (Supervised or Unsupervised) we're dealing with is also an easy way to exclude some options. Computation times are also important to take into account.

Symbolists are probably the simplest to understand, interpret and visualize set of techniques one can come across. Rules and Decision Tree offer clear insight about data and are easy to extrapolate conclusions from. They can handle both numerical and categorical data, and multi-output problems. Nonlinearities in the data do not affect the performance of trees. Nevertheless, complex datasets where several rules need to be inferred may not be the most fitting for these techniques. Over-complex trees may end up not generalizing the data well, thus resulting in overfitting. Some classes may also dominate others if the input dataset is not balanced, so it is recommended to have a similar number of examples for each class. In terms of achieving satisfying results, a global maximum may not be found by simply running one decision tree. A more favourable set of results can be obtained by training multiple trees where features and samples are randomly sampled with replacement. Alternatively, Random Forests may be used to overcome some of the problems inherent to Decision Trees. Namely, the problem of overfitting. Because there are enough trees in the forest, the classifier won't overfit the model. Both training and prediction are very fast, because of the simplicity of the underlying Decision Trees. In addition, both tasks can be straightforwardly parallelized, because the individual trees are entirely independent entities. Nevertheless, Random Forests still have their limitations. A large number of trees can make the algorithm too slow and ineffective for real-time predictions and, despite these algorithms being fast to train, they are quite slow in making predictions once they are trained. Results are also not easily interpretable, that is, drawing conclusions about the *meaning* of the classification model is not an intuitive process.

The Naive Bayes, from the **Bayesians** Tribe, algorithm affords fast, highly scalable model building and scoring. Naive Bayes can be used for both binary and multiclass classification problems. These models are relatively easy to understand and build. They are easily trained and don't need big

datasets to produce effective results. They are also insensitive to irrelevant features. These advantages mean that a Naive Bayesian classifier is often a good choice as an initial baseline classification. If it performs suitably, it means you'll have a very fast and interpretable classifier for your problem without much effort. If it does not perform well, other models should be explored. Since this algorithm always assumes that features are independent, which is not true for most real-life situations, it is expected that this model will not perform well in most cases.

Connectionists offer a wide range of applications. Deep Learning has become quite popular in the last few years in Image Processing, Speech Recognition and other areas where a high volume of data is available. Neural Networks, the preferred model of this tribe, can yield some impressive results since they can generalize rules between input features and output variables, given we have enough examples in our dataset. This technique is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones. The target function output may be discrete-valued, real-valued, or a vector of several real or discrete-valued attributes. It's an appropriate technique for datasets which contain training examples with errors and, while ANN learning times are relatively long, evaluating the learned network is typically very fast. However powerful these techniques may be, achieving the most favourable model architecture may be a challenging and iterative process. There are several details the designer needs to be mindful of when crafting a network, namely the number of layers, activation functions, the loss function and optimizers used or the normalization of the data. The goal is to achieve a low training error of the network while avoiding overfitting.

EAs, from the **Evolutionaries** Tribe, are a set of modern heuristics used successfully in many applications with great complexity. The most immediate advantage of evolutionary computation is that it is conceptually simple. Almost any problem that can be formulated as a function optimization task, can be modelled by an EA [25]. These algorithms are capable of self-optimization and their processes are highly parallel. This means that the evaluation of each obtained solution can be handled in parallel. Only the mechanism of selection requires some serial processing. EAs can also be used to adapt solutions to changing environments. Usually, it is not necessary to obtain new populations at random and restart the model when new solutions want to be obtained. This is due to the high flexibility of the algorithm, making the available populations a solid foundation to make further improvements. Disadvantages of this algorithm include the lack of guarantee on finding global maxima and high computation times. Usually a decent sized population and generations are needed before good results are obtained.

The tribe of the **Analogizers** is better known for the SVM technique. It is one of the most efficient Machine Learning algorithms and is mostly used for pattern recognition [26]. It has a wide range of applications, such as speech recognition, face detection and image recognition. This is a very powerful supervised learning algorithm for data separation which builds a model that foresees the category of a new example, based on a given set of featured examples. It works on the principle of fitting a boundary to a region of points that meet the same criteria of classification i.e. belong to the same class. Once a boundary is fitted on the training sample points, for any new points that need to be

classified, the designer must only check whether they are inside the boundary or not. The advantage of SVM is that once a boundary is established, most of the training data is redundant. All it needs is a core set of points which can help identify and set the boundary. These data points are called support vectors because they "support" the boundary. Data types for this algorithm include linear and non-linear patterns. Linear patterns are easily distinguishable and can be separated in low dimensions, but the same cannot be said about non-linear patterns. The latter need to be manipulated for the data to become separable e.g. by means of kernel functions. Another advantage of SVMs is that they generalize new samples very well. When an optimal set of hyperplanes that separate the data is achieved, SVMs can produce unique solutions, which is a fundamental difference between this technique and ANNs. The latter yields multiple solutions based on local minima, which might not be the accurate over different test data. In terms of disadvantages, SVMs might not be the most desirable choice when tackling problems with high-dimensional data. These are heavily reliant on the choice of the kernel and its parameters, and even then, the obtained results might not be transparent enough to extrapolate any meaningful conclusions.

In Table 1 we can observe a summary of the pros and cons of each tribe.

Table 1 - Comparison of advantages and disadvantages between each tribe of Machine Learning

Tribe	Advantages	Disadvantages
Symbolists	• Easy to understand; • Can handle both numerical and categorical data, and multi-output problems.	• Not a practical approach if there are several decisions to be made; • Over-complex trees may end up not generalizing the data well, thus resulting in overfitting.
Bayesians	• Requires less data to be effective; • Fast in the prediction of a given feature's class; • Insensitive to irrelevant features.	• Not useful when approaching real life situations, since features are usually inter-dependent.
Connectionists	• Appropriate for complex and noisy data sets; • Good generalization of rules between input features and output values.	• Might be necessary to iterate the algorithm several times to yield favourable results; • Overfitting is likely to occur.
Evolutionaries	• Appropriate for problems with a wide range of parameters.	• Expensive computation times; • May not find global maxima.
Analogizers	• Can solve both linear (classification) and nonlinear (regression) problems; • Can produce unique solutions.	• Inappropriate for complex and noisy data sets; • Undesirable choice when tackling problem with high-dimensional data.

2.3. Related Work on Machine Learning applied to Analog IC Sizing

Some approaches have already been made to implement Machine Learning for analog IC sizing. Particularly, ANN models have been used to address this problem.

In this [27] work from 2004, ANN models for estimating the performance parameters of CMOS operational amplifier topologies were presented. In addition, effective methods for the generation of training data and consequent use for training of the model were proposed to enhance the accuracy of the ANN models. The efficiency and accuracy of the performance results was then tested in a genetic algorithm-based circuit synthesis framework. This genetic synthesis tool optimizes a fitness function based on a set of performance constraints specified by the user. Finally, the performance parameters of the synthesized circuits were validated by SPICE simulations and later compared with those

predicted by the ANN models. The set of test bench circuits presented in this work can be used to extract performance parameters from other op-amp topologies other than ones specifically studied here. Circuits with different functionalities than an op-amp would need new sets of SPICE test bench circuits to create appropriate ANN models.

The ANN models trained in this work used data generated from SPICE, where time-dependant and frequency-dependant data points were created for numerous circuit topologies instantiating the target op-amp. The training set was comprised of 3095 points, while the validation set was comprised of 1020 points. The neural network toolbox from Matlab was used to simulate the networks. The structure of the network is simply comprised of an input layer, an output layer and a single hidden layer. Different numbers of hidden layer nodes (from 8 to 14) were iteratively tested to obtain the best possible generalization and accuracy on both training and validation examples. The hyperbolic tangent sigmoid function was used as the activation function for all hidden layer nodes and a linear function was used as activation function for all output layer nodes. It should be noted that each network only has one single output node, since a different network was designed to model each individual op-amp performance parameter.

Collecting the data took approximately 1h47m and generating all seven performances estimates using the ANN models took around 51.9 μs, which, when compared with using SPICE directly, resulted in a speedup factor of about 40,000 times.

In this [28] work from 2015, a method for circuit synthesis that determines the parameter values by using a set of ANNs was presented. Two different ANN architectures were considered: the multilayer perceptron and the radial basis function network. Each of the two networks is optimized to output one design parameter. Hyper-parameters (such as the number of nodes from the hidden layers) from both models are tuned by a genetic algorithm. The presented methodology was tested on the design of a radio-frequency, low noise amplifier, with ten design parameters to set.

The goal of this work was to find design parameters in sequence, each one constraining the determination of the next one. The process starts with an ANN being trained to correctly determine a first design parameter, by taking a set of desired performances as input and only a single target output representing the chose design parameter. From the two architectures specified above, multilayer perceptron and radial basis function network, one is chosen to characterize this ANN. This selection is performed by a genetic algorithm, which is also responsible of determining the network's hyper-parameters and which design parameter should be chosen as output. The genetic algorithm stops after an ANN implementation achieves satisfiable results during the training in terms of generalization and accuracy. The same process is repeated for a second ANN, that will also take as input the output of the first ANN. This way, the second ANN will specify the second design parameter as a function of the first design parameter, as well as of the performance criteria. The process is repeated until all the design parameters have been found.

The number of points in the dataset was 235, and a varying number of them (selected by the genetic algorithm) was used for training the ANNs, while the remainder of the points were used for the test set.

All the tests were performed with the Matlab neural network toolbox. The considered ANN architectures consist of an input layer, an output layer and a single hidden layer. The number of nodes from the hidden layer is selected by the genetic algorithm and it can range from 2 to 62 nodes for the multilayer perceptron architecture, and from 1 to 128 nodes for the radial basis function architecture. Logarithmic sigmoid and Tangent sigmoid functions were used as activation functions for the hidden layer nodes for the multilayer perceptron architecture.

The method proposed in this work was able to find the ten design parameters on all twenty performed simulations. Computation times were approximately 5.375h.

In this [11] work from 2017, a prediction method of element values of OP Amp for required specifications using deep learning was proposed. The proposed method is a regression analysis model implemented via a feed forward ANN that can learn the relation between the performances and element values of a target circuit. Good results were achieved when attempting to infer element values which meet the required performances of the circuit selected for this procedure.

The circuit performances used for learning were 13 items: current consumption (I_{dis}), power consumption (P_{dis}), DC gain, phase margin (PM), gain bandwidth product (GBP), slew rate (SR), total harmonic distortion (THD), common mode rejection ratio (CMRR), power supply rejection ratio (PSRR), output voltage range (OVR), common mode input range (CMIR), output resistance (OR), and input referred noise (IRN). The used dataset was comprised of 13,500 different points, in which 13,490 are used as a training set and 10 are used as a test set. TensorFlow was used as the machine learning resources library. The structure of the feed forward neural network was comprised of an input layer, two hidden layers with 100 and 200 nodes, respectively, and an output layer. Both hidden layers used Rectified Linear Unit as an activation function.

Collecting the data to feed the neural network took approximately 18 hours and only 19 minutes were spent on training the network. Results from simulations indicated that the proposed deep learning method succeeded in predicting 7 element values which satisfies the required 13 performances with an accuracy of 93% in average.

2.4. Choice of the Model Approach

After analysing the pros and cons of each of the aforementioned Tribes of knowledge of Machine Learning, conclusions can be drawn in order to choose the most appropriate technique for solving the problem that motivates this work. By inspecting some successful Machine Learning applications in analog IC sizing, it was also possible to grasp more clearly which approaches might be the most desirable.

Analog systems are usually characterized by a set of performances parameters that are used to quantify the properties of the circuit, i.e. design parameters. The relationship between circuit performances and design parameters can be interpreted as either a direct problem or an inverse problem. The former asserts circuit performances as a function of design parameters, while the latter deals with finding the most appropriate design parameters for a given set of performance

specifications [29]. Mapping the relationship between these two features is a heavily non-linear process, given the high interdependence among design variables. Moreover, a set of performances might be valid to more than one set of design variables, i.e. different circuit topologies might be satisfied by the same set of performances.

The problem we are trying to solve in this work is indeed an inverse problem. Having this in mind, the ideal candidate solution should be able to deal with non-linearities and try to map the complex relationship between performances figures and design parameters. This type of problem also tells us something about the nature of the model we want to build: performances will be used as input data and design parameters will be used as output data. Specifically, for this work we want to obtain devices' sizes, such as the lengths and widths of transistors.

These ground rules will allow us to make a more informed judgement regarding what technique to choose for the model we want to build. ANNs seem the most fitting choice for that model, not only because they have already been used to solve similar problems (as seen in the previous Section), but also because of their nature. ANNs are very appropriate for non-linear problems and for mapping relationships between input and output data. Nevertheless, other options should be entertained for the sake of completeness.

The Evolutionary tribes was not chosen because the model proposed in this work will be built on top NSGA-II, an EA mentioned in Section 2.1.4. This EA is responsible for generating solutions that will be used as input data to feed the selected automatic learning model.

The Bayesians tribe was also promptly discarded. If the training set is small, a high bias/ low variance classifier such as Naive Bayes has an advantage over low bias/ high variance classifiers (e.g. ANNs), since the latter will overfit. But low bias /high variance classifiers start to win out as the training set grows, since high bias classifiers aren't powerful enough to provide accurate models. The dataset we will present in Chapter 4 is expected to have a variable size between 5k and 20k (excluding a possible artificial augmentation of the data), depending on the architecture, and as such, Naïve Bayes might not a suitable solution.

SVMs, from the Analogizers tribe, provides a high accuracy in classification, has well documented methods to avoid overfitting, and with an appropriate kernel they can work well even if the training data isn't linearly separable in the base feature space. These are especially popular in text classification problems where very high-dimensional spaces are the norm, which is not the case of the analog IC sizing problem we are tackling. SVMs are notoriously memory-intensive and the obtained results are somewhat hard to interpret, as well as hard to tune.

Decision Trees, from the Symbolists tribe, are easy to interpret algorithms. They easily handle feature interactions and they're non-parametric, so the designer doesn't have to worry about outliers or whether the data is linearly separable. One disadvantage is that they don't support online learning, so a new tree has to be built each time new examples come on. Another disadvantage is that they easily overfit, but a workaround to this problem is to use ensemble methods like Random Forests. These methods are fast and scalable, and require a low amount of tuning. Random Forests could've been

chosen, as there are already some papers where a multi-variate random forest was successfully implemented (but not in the context of EDA) [30]. Nevertheless, ANNs were the chosen technique.

The set of techniques from the Connectionists tribe has become quite popular in recent years. ANNs are widely used in pattern recognition because of their ability to generalise and to respond to unexpected inputs/ patterns. It is a low bias /high variance classifier that is appropriate for high volume datasets. These are also very flexible models that allow the implementation of different tasks in the same networks. In the context of analog IC sizing, ANNs might be the most appropriate choice because they implement a black-box methodology that relies on analogies and statistical knowledge, instead of precise models [28]. Moreover, one of the goals of this work is to implement a hybrid model that is able to perform both regression and classification tasks at the same time, and ANNs are capable of providing that.

2.5. Conclusions

In this chapter, State-of-the-Art for Machine Learning was presented. A categorization of techniques was described as well as the most used methods that belong to each Machine Learning Tribe. These techniques were discussed in great detail in order to evaluate which was the most appropriate technique to solve the problem at hand. Several aspects were considered, such as computation times, accuracy of solutions and whether or not an algorithm was appropriate for the type of data used in this work.

ANNs were ultimately chosen as the preferred ML technique because of their strong ability to generalize rules and patterns from data. Additionally, they are appropriate for a high throughput of data. While their computation times may vary with the amount of data being processed and become quite long, this will not affect posterior predictions, since the most time-consuming task is the network training. Another reason for this choice was the unique ability ANNs have of mixing both regression and classification tasks in one single network, which is one of the goals this work aims to achieve. The fact that some researchers have already implemented ANN models to solve analog IC sizing problems was also a contributing factor to choose ANNs.

Chapter 3. **Brief Overview of Artificial Neural Networks**

This Chapter continues exploring the topic of ANNs started in the previous Chapter. Details regarding the structure of this model are further discussed here, as well as the most recent applications of ANNs in Deep Learning.

3.1. Structure

As already mentioned before, ANNs are structured into layers: a single input layer, a single output layer, and a variable number of hidden layers. These are comprised of a set of neurons. In each layer, every neuron outputs a single real number, passing it to every neuron in the next layer. At the next layer, each neuron will form its own weighted combination of the values received from the preceding layer's neurons. Then, the neuron adds its own bias, b, and applies an activation function, σ.

Assuming that the collection of values produced by neurons from a single layer are stored in a vector, a, then the next layer will have vector of output in the form:

$$\sigma(Wa + b) \tag{11}$$

In this expression, W is a matrix that contains the weights from the neurons of a given layer, and b is a vector containing the biases of each neuron. The number of columns in W has the same number of neurons from the previous layer, while its number of rows is the same as the number of neurons in the current layer. The same can be said about the number of components in vector b, which are the same as the number of neurons in the current layer.

That being said, the expression in (11) can be completed by taking into account all neurons i and layers j:

$$\sigma\left(\sum_j w_{ij}\, a_j + b_i\right) \tag{12}$$

where the sum runs over all entries in a [31].

3.2. Activation Functions

An activation function (or transfer function) is needed to decide whether a neuron fires or not. Since the output value may be any number ranging from $-\infty$ to $+\infty$ and the neuron doesn't know its bounds, we need a function that processes these values and confines them within a range. These functions loosely resemble an *on* and *off* switch i.e. they provide an output between a closed and well-defined boundary. A step function $f(x)$, which only has two possible outputs, 0 and 1, would work like a binary switch and thus solve the boundary problem. Neurons would fire when an output's value was higher than 0, or otherwise if their output was lower than the threshold. This option would be acceptable if, for instance, we wanted to create a binary classifier, but most real-world problems are comprised of

several classes. For such cases, we need an activation functions that provides intermediate values. Functions with this capability can be linear or non-linear in nature.

The most immediate solution to solve the lack of boundary problem would be to use a linear function:

$$\sigma(x) = ax \tag{13}$$

This function, illustrated in Figure 10, is a simple straight line where the activation is proportional to the input. Therefore, a range of activations is provided. Nevertheless, this activation function will be rendered useless when we take into account the backpropagation algorithm used to train ANNs. This algorithm (which will be furthered analysed in the next Section) is based on an optimization method called gradient descent. The method proceeds iteratively, computing sequences of vectors that aim to minimize a cost function [31]. Since this method is based on the computations of the derivatives of a function, and the derivative of the linear activation function is constant, the gradient would be constant and never find a local minimum.

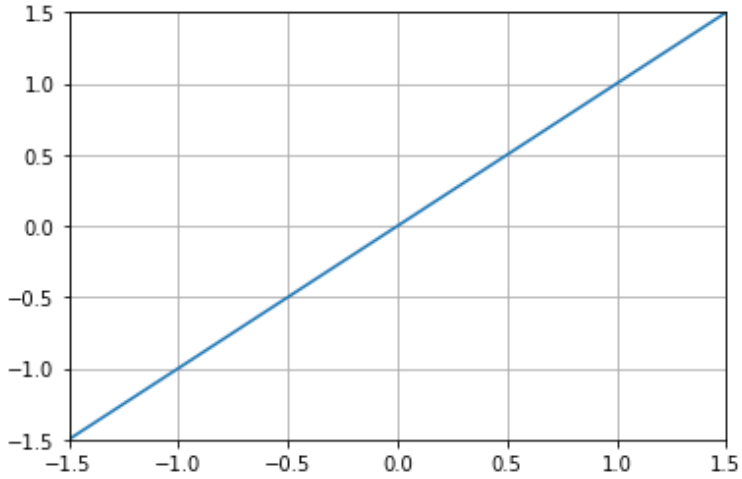

Figure 10 - Linear function.

To overcome this problem, designers can use non-linear functions. Traditionally, the sigmoid (or logistic) function shown in (14), has been the most used activation function in ANN architectures, but the advent of more complex networks made it fall into disuse.

$$\sigma(x) = \frac{1}{1 + e^{x}} \tag{14}$$

This is a non-linear function that has a range from 0 to 1. It is especially appropriate to obtain outputs that represent probabilities e.g. a binary classification problem. While this function overcomes some of the limitations of the linear function, it still has its drawbacks. In Figure 11 we can observe that, towards either end of the function, the y values tend to respond significantly less to changes in x. This means that the gradient at that region, where the curve is nearly horizontal, is going to be small. This will result in a problem commonly known as *vanishing gradients*. The network will therefore start learning at a much slower rate, since there are no great variations in the gradient.

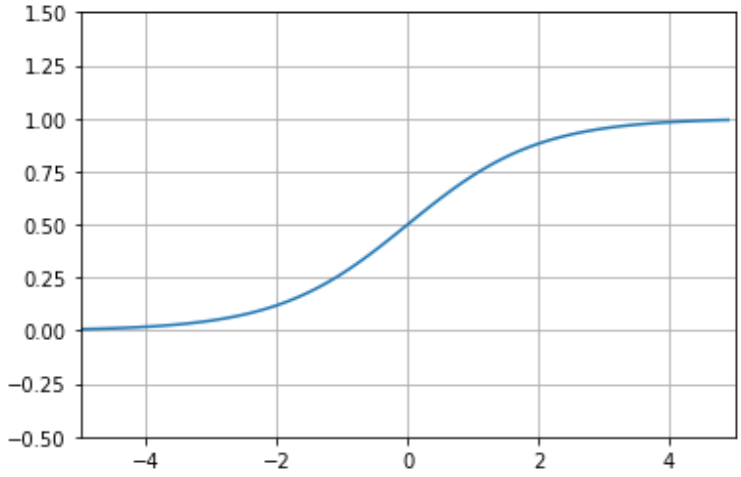

Figure 11 - Sigmoid Function.

The hyperbolic tangent function (Figure 12) is another option:

$$\sigma(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{15}$$

It is nonlinear in nature and ranges from -1 to 1. The gradient for this function is stronger than for sigmoid (i.e. derivatives are steeper).

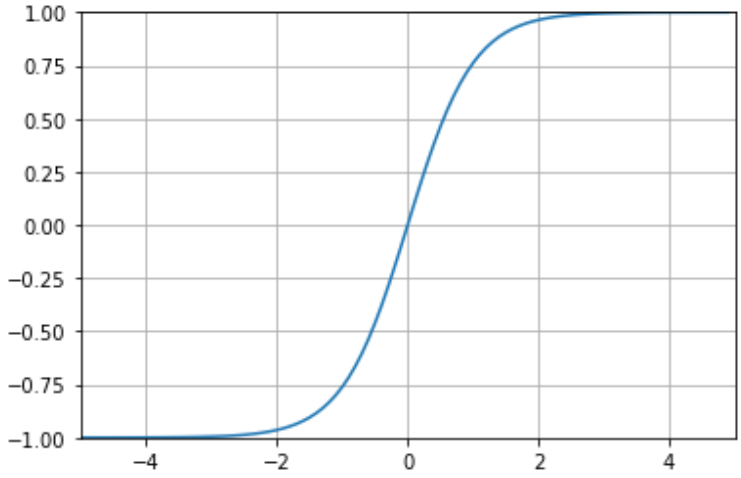

Figure 12 - Hyperbolic tangent function.

The final activation function worth mentioning is the Rectified Linear Unit (ReLU):

$$\sigma(x) = \max(0, x) \tag{16}$$

This non-linear function (Figure 13) has become quite popular following the recent *boom* in Deep Learning. It gives an output x if x is positive and 0 otherwise. It was in 2011 that Glorot et al. [32] demonstrated that this activation function enables better training of deep networks than the classical sigmoid approach. Particularly, it was proved that the gradient is better propagated while minimizing

the cost function of the network. Problems regarding *vanishing gradients*, which can for example be observed in both ends of the sigmoid function, are also less common in ReLU.

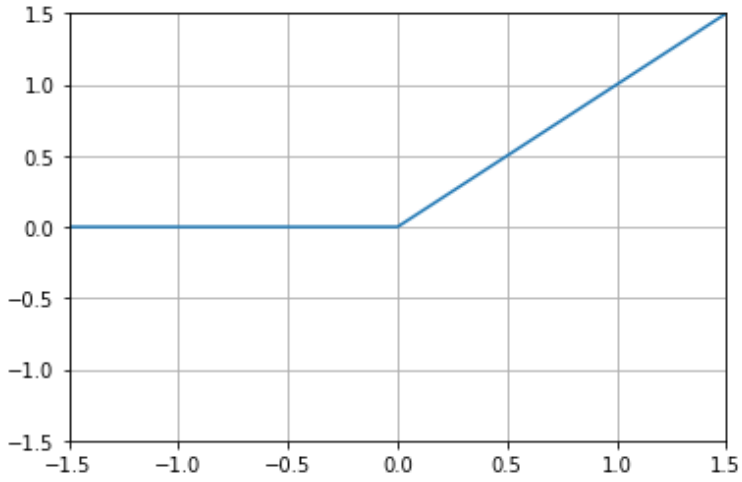

Figure 13 - ReLU function.

3.3. Back-Propagation Algorithm

The most common approach when modelling an ANN is to use the back-propagation algorithm, which is a method to calculate the gradient of the transfer or activation function with respect to the weights of each node. When an input x is fed to an ANN, it flows forward through the network producing an output $\hat{y}$. This is called forward propagation. During the training of the network, the forward propagation can continue until a scalar cost, $J(\theta)$, is obtained. This is the Jacobian of the loss function chosen for the model. The information from the cost will then flow back through a mechanism called backward propagation (or simply *backprop*) and be used to calculate the gradient. The back-propagation algorithm is able to calculate the gradient in a simple and effective way, which would otherwise be computationally expensive were it numerically evaluated [33].

Training a network corresponds to choosing the parameters (weights and biases from (12)) that minimize its cost function. Minimization of the cost function involves a classical optimization method referred to as *steepest descent* or *gradient descent*.

Nowadays, several minimization methods for cost functions are used in the training of ANNs, which are commonly referred to as *optimizers*. Popular optimizers among researchers are Adam [34] and the Stochastic Gradient Descent optimizer (SGD). These are more powerful versions of gradient descent and more appropriate for Machine Learning problems where large training sets are processed. Since these training sets are more computationally expensive, the cost function will often be decomposed as a sum over training examples of a given loss function [33]. For example, the negative conditional log-likelihood of the training data in a Machine Learning problem can be written as:

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}_{data}} L(x, y, \theta) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) \tag{17}$$

Where L is the loss $L(x, y, \theta) = -\log p(y|x; \theta)$.

For additive cost functions, gradient descent requires an additional computation:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta) \tag{18}$$

This operation is computationally expensive, since its complexity is $O(m)$. This will become a problem when high amounts of data are trained, which will show in the computation times of the gradient.

The gradient can be interpreted as an expectation, which can be approximately estimated using small sets of examples $\mathbb{B} = \{x^{(1)}, \ldots, x^{(m')}\}$, drawn uniformly from the training set, called *minibatches*. These can be sampled on each step of the algorithm. The size m' of the minibatch is usually chosen to be small, ranging from 1 to a few hundred.

Using examples from the minibatch $\mathbb{B}$, the estimate of the gradient is formulated as:

$$g = \frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta) \tag{19}$$

The SGD algorithm then follows the estimated gradient downwards, looking for a minimum:

$$\theta \leftarrow \theta - \epsilon g \tag{20}$$

where ϵ is the learning rate.

Several researchers have pointed out in the past that gradient descent is a slow and ineffective algorithm [33]. Applying this algorithm to non-convex optimization problems was often considered a bad practice, since the gradient would never be able to find an optimal minimum. But today, this algorithm is seen as a state-of-the-art optimizer for most Machine Learning problems. In Machine Learning, researchers favour solutions that come at a low cost and take little to no time to achieve, which gradient descent can provide. Even if the solution is sub-optimal, it usually is good enough to be useful.

3.4. Deep Learning

Deep Learning is a subset of Machine Learning characterized by complex and robust models that can process high throughput data, surpassing in performance some of the most traditional techniques used in Machine Learning. Models like Deep Neural Networks, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks have become a staple in some modern applications, such as computer vision, speech recognition, bioinformatics and drug design [35].

CNNs are a specialized kind of ANN for processing data that has a known, grid-like topology (e.g. time series data and image data) [33]. The word *convolutional* is used to denominate these networks

because the operations involved, such as linear transformations, are written in the form of a convolution. In the 1-dimensional case, the convolution of the vector $x \in \mathbb{R}^p$ with filter $g_{1-p}, g_{2-p}, \cdots, g_{p-2}, g_{p-1}$ has k^{th} component given by:

$$y_k = \sum_{n=1}^{p} x_n g_{k-n} \tag{21}$$

Layers from CNNs are usually comprised of three stages [33]. In the first stage, a set of linear activations are obtained from a series of convolutions that are performed in parallel by the layer. In the second stage, called the detector stage, each linear activation is subject to a non-linear activation function, such as ReLU. Finally, a pooling function is used to modify the output of the layer. A pooling function is responsible for aggregating statistics from nearby outputs and replacing the output of the network with those.

RNNs are a family of ANNs specialized in processing sequential data [33]. Unlike other typical ANNs, RNNs are able to use their internal memory to process sequences of variable length. This allows these types of networks to exhibit temporal dynamic behaviour. RNNs can also scale to much longer sequences than would be practical for networks without sequence-based specialization.

3.5. Conclusions

Throughout its lifetime, ANNs have been cyclically abandoned and picked up. Since its creation, this method has been controversial among mathematicians [36]. The McCulloch-Pitts neuron, invented in 1943 and Frank Rosenblatt's Perceptron were very primitive in their capabilities, despite providing a solid framework for an algorithm that aimed to imitate how the human brain works. While the Perceptron allowed the network to iteratively adjust its connection's weights, a stunning breakthrough in relation to McCulloch-Pitts Neuron, it was only capable of binary classification.

The fact that this model couldn't even solve a XOR function was an indication that ANNs would never strive in the eyes of many researchers in the field. It wasn't until 1986, when the Backprop algorithm was proposed by David Rumelhart, that ANNs saw a resurgence in Machine Learning. ANNs were now capable of iteratively adjust their connection's weights in a more efficient way and remarkably generalize rules from data.

Still, the most notable *boom* in ANN usage was only seen years later with the advent of more powerful data mining tools. The turn of the century saw a colossal increase in computational power that allowed researchers to extract higher volumes of data and build more robust models. Thus, a new designation for these set of tools was created: Deep Learning. This term is mostly used when addressing ANN application in such fields as image recognition, speech recognition or natural language processing.

Chapter 4. Proposed Artificial Neural Network Architectures

In this work, two ANN models are proposed. The first one, a Regression-only Model, serves as a proof of concept (i.e. the applicability of ANNs to analog IC sizing is tested). In this architecture, we explore how an ANN that is trained using circuit sizing solutions from previous optimizations can learn the design patterns of the circuit (a single circuit topology is considered in each training phase). The second one, a Classification and Regression model, is also presented. This architecture not only selects the most appropriate circuit topology but also its respective sizing given the target specification (more than one topology is considered in the training phase).

4.1. Design Flow

The proposed automatic design flow of analog IC sizing using ANNs is as follows:

1. Determine the prediction target;
2. Collect data for learning;
3. Create learning model;
4. Train learning model;
5. Confirm the accuracy of the prediction;
6. Sample the obtained results;
7. Test the sampled results in AIDA.

The first step in the creation of an ANN model is to determine the prediction target. In this case, we want the network to learn design patterns from the studied circuits, using circuit's performances (DC Gain, current consumption (IDD), gain bandwidth (GBW) and phase margin (PM)) as input features and devices' sizes (such as such as widths and lengths of resistors and capacitors) as target outputs. The next step involves gathering data so that the model can learn patterns from the input-output mapping. Data should be split into three different sets: the training set, the validation set and the test set. After determining the prediction target and assembling the data, hyper-parameters of the model should be selected. These are the number of layers, number of nodes per layer, the activation functions, and the loss function, to name a few. After selecting the most appropriate hyper-parameters, the model is ready to be trained. At this stage, the model will attempt to iteratively find its optimal network weights. After training the model and achieving a sufficiently low error on the training phase and on the validation set, the model is ready to be used for predicting the output of real-world input data. This is where we will evaluate the accuracy of the predicted results and obtain devices' sizes from a set of desired circuit performances. The next step involves sampling the results from the model. This step is essential to circumvent possibly biased solutions predicted by the ANN. In the final step, we test the feasibility of the obtained solutions in AIDA.

AIDA, whose architecture is illustrated in Figure 14, is an analog integrated circuit design automation environment, which implements a design flow from a circuit-level specification to physical layout description. AIDA results from the integration of two in-house tools, namely, AIDA-C and AIDA-L. AIDA-

C is based on multi-objective multi-constraint optimization kernels and uses electrical simulators as evaluation engine to perform circuit-level optimization. AIDA-L implements an innovative fully automated layout generation methodology. By integrating both, AIDA-C and AIDA-L, a layout-aware synthesis methodology for automatic optimization-based sizing of analog ICs is achieved. Despite implementing a fully automated synthesis in all aspects of the tool, AIDA is not a black box optimizer, as it allows the designer to monitor and intervene throughout all the synthesis process, e.g., the designer can provide high level guidelines for the floorplan; interrupt the synthesis process whenever an acceptable solution is already present; interrupt to redefine the specifications; select the desired solution from the Pareto optimal front generated by AIDA-C to be layout by AIDA-L; etc. [37]

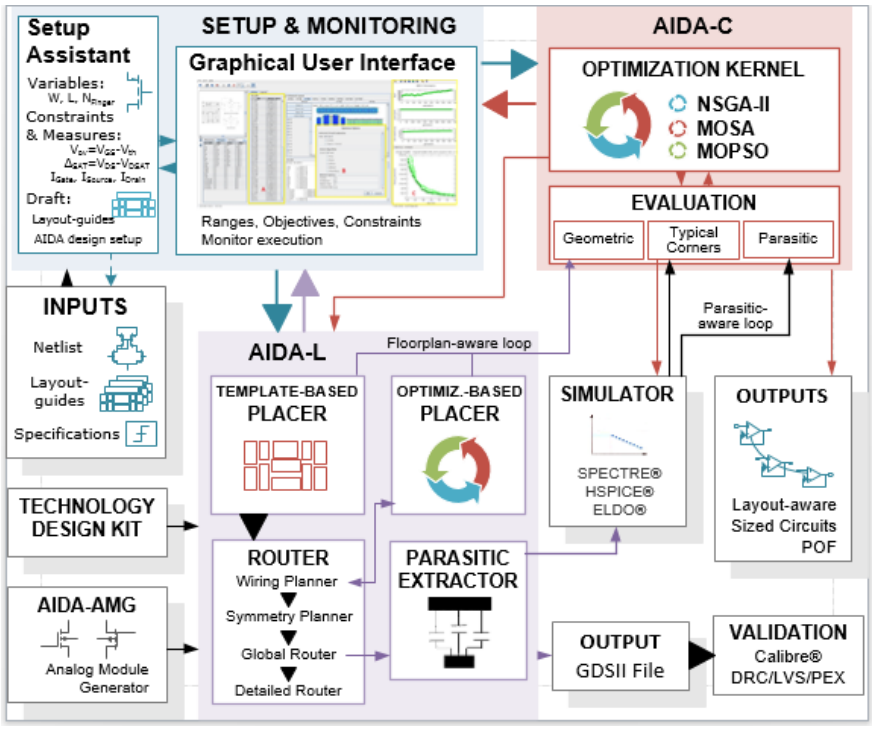

Figure 14 - AIDA Architecture.

4.2. Problem and Dataset Definition

Let $V_{<i>}\epsilon\mathbb{R}^{N}$ be the vector of design variables that define the sizing of the circuit, where the index i inside the chevron identifies solution point i in the dataset, and $S_{<i>}\epsilon\mathbb{R}^{D}$ be the vector of the corresponding circuit performance figures. The ANNs presented in this work were trained to predict the most likely sizing given the target specifications, as shown in (22).

$$V_{<i>} \sim \operatorname{argmax}\left(\mathrm{P}(\mathrm{V}_{<i>}|\mathrm{S}_{<i>})\right) \tag{22}$$

Hence to train the ANN, the training data is comprised of a set T, of M data pairs $\{V,S\}_{<i>}$. Since we want the model to learn how to design circuits properly, these pairs must correspond to useful designs, e.g., optimal or quasi-optimal design solutions for a given sizing problem.

While the previous definition in (22) allows to train a model suitable for analog IC sizing, circuits' target specifications are, more often than not, defined as inequalities instead of equalities. Therefore, an ANN trained to map $S \rightarrow V$ may have difficulties extrapolating to some specification values that are actually worse than the ones provided in training.

The point is, if the sizing $V_{<i>}$ corresponds to a circuit whose performance is $S_{<i>}$, then it is also a valid design for any specifications whose performance targets, $S'_{<i>}$, are worse than $S_{<i>}$. Having this in mind, an augmented dataset, T' can be obtained from T as the union of T and K copies of T, as indicated in (23), where the for each sample i, the $S_{<i>}$ is replaced by $S'_{<i>}$ according to (24).

$$T' = \{T \cup T^{C1} \cup T^{C2} \cup T^{CK}\} \tag{23}$$

$$S'_{<i>} = S_{<i>} + \left(\frac{\gamma}{M}\sum_{j=1}^{M} S_{<j>}\right)\Delta\Gamma \tag{24}$$

Where, $\gamma \epsilon]0,1[$ is a factor used to scale the average performances, Δ is a diagonal matrix of random numbers between $[0,1]$, and, $\Gamma \in \{-1,1\}$D is the target diagonal matrix that define the scope of usefulness of the circuit. Its diagonal components take the value -1 for performance figures in which a smaller target value for the specification is also fulfilled by the true performance of the design, e.g., DC Gain, and, the value 1 is for the diagonal components corresponding to performance figures that meet specification targets that are larger than the true performance of the circuit, e.g., power consumption.

When using supervised learning models, the data is usually split into three separate sets:

- **Training Set**: set of examples used for learning. In the case of an ANN, the training set is used to find the optimal network weights with back-propagation;
- **Validation Set**: set of examples used to tune model hyper-parameters. Performance metrics such as Mean Squared Error (MSE) or Mean Absolute Error (MAE) can be applied to the classification of this set of data to evaluate the performance of the model;
- **Test Set**: set of examples used to evaluate the performance of a fully trained classifier on completely unseen real-world data, after the model has been chosen and fine-tuned.

This segmentation is one the first measures a designer can take to avoid overfitting. If the model would be tested on the same data it was trained with, the obtained classification would yield no error because the test data would have already been presented to the model. Therefore, no meaningful conclusions regarding the effectiveness of the model could be made. To avoid this, part of the available data is held out and used as validation data to test the model on unseen examples so that the designer can evaluate performance metrics and fine-tune model hyper-parameters. After the model has been finalized, a test set is fed to the fully trained classifier to assess its performance. Validation and test sets are separate because the error rate estimate of the final model on validation data is usually biased (smaller than the true error rate) since the validation set is used to select the final model [38]. The percentage assigned to each set is variable, but it is common to use 60%, 20%

and 20% of total data for the training, validation and test sets, respectively, especially if the available data is scarce.

On the topic of scarcity of data, there are some re-sampling methods that are more appropriate for models where there are low amounts of data. Simply partitioning the data into 3 distinct sets might affect the performance of the model in the training phase. These alternative procedures are known as cross-validation methods. Although they come at a higher computational cost, cross-validation methods are ideal for small datasets. A test set should still be held out for final evaluation, but the need for a validation set vanishes. A particular method of cross-validation, called *k-fold cross-validation*, consists of splitting the training set into *k* smaller sets, as it can be observed in Figure 15. For each of the k-folds, the following procedure is applied:

- The model is trained using $k - 1$ of the folds of the training data;
- The remainder of the data is used to validate the model and compute performance metrics;
- The true error is estimated as the average error rate on validation data of all k-folds.

The advantage of k-fold cross-validation is that all the data selected for the training set is used for both training and validation.

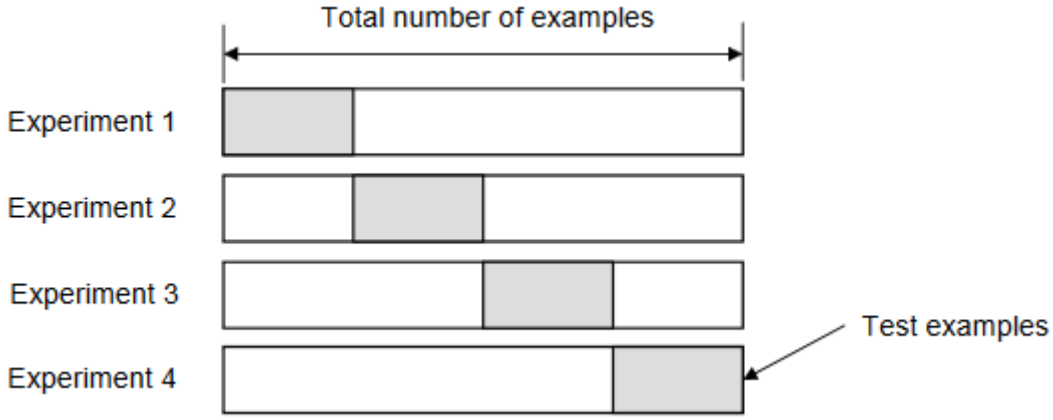

Figure 15 - K-Fold Cross-Validation [38].

4.3. Regression-Only Model

The ANNs models discussed in this Section consider fully connected layers without weight sharing. Given the number of features that are used in the model and the size of the datasets that will be considered in this application, the model is not very deep, containing only a few of hidden layers. A base structure for this model can be observed in Figure 16. The best structure depends of the dataset, but a systematic method is proposed later in this Section to specify such models. To train and evaluate the model, the datasets are split in training (80%-90%) and validation sets (20%-10%). An additional small test set, whose specifications are drawn independently, is used to verify the real-world application of the model.

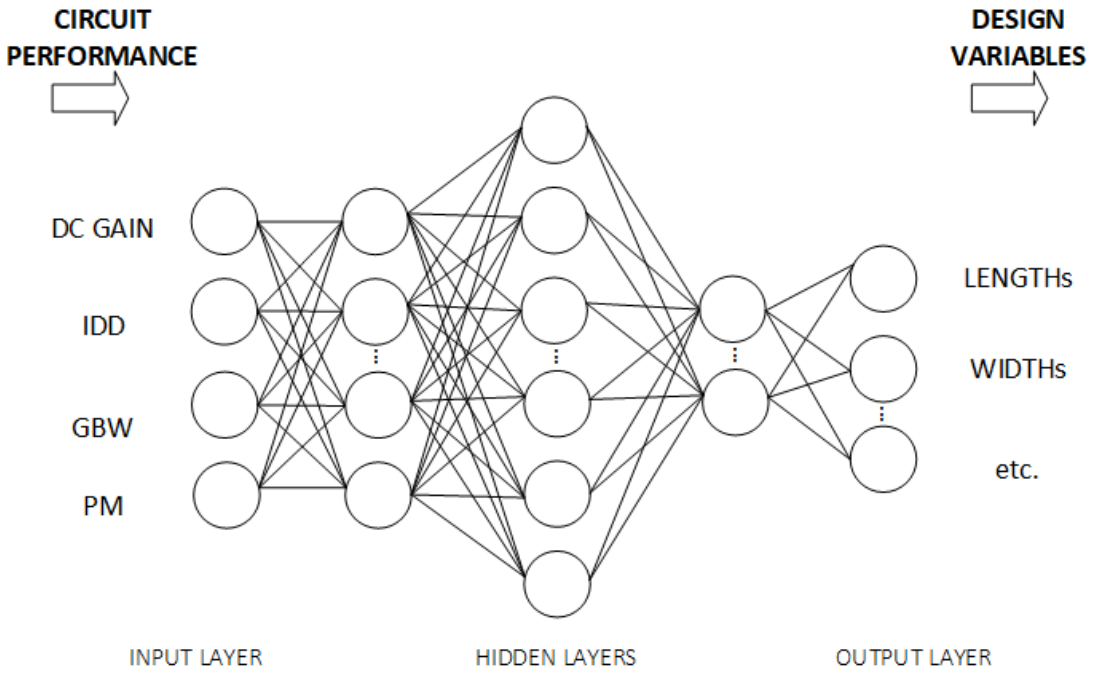

Figure 16 - Base Structure of the Regression-only Model.

Initial architecture design was simply comprised of an Input Layer and an Output Layer. Hidden layers were later added to evaluate model complexity. The number of input nodes (15) is the same throughout test cycles, as well as the number output nodes (12 or 15, depending on the circuit topology). Only the number of hidden layers and its nodes were persistently changed to figure out which architecture yielded lower training and validation error.

4.3.1. Polynomial Features and Data Normalization

The input features measure different attributes from the circuits, so it's recommended to standardize the input matrix. A scaler is applied after we expand the feature space by generating polynomial features, which is quite useful since we only consider 4 circuit performances as input features. Adding extra columns of features helps giving more flexibility to the model.

More specifically, the ANN is trained with an input X that is the feature mapping Φ of S normalized. Each input data sample $X_{<i>}$ is given by:

$$X_{<i>} = \frac{\Phi(S_{<i>}) - \mu_{\Phi}}{\sigma_{\Phi}} \tag{25}$$

where $\Phi(S_{<i>})$ is a second order polynomial mapping of the original specifications, e.g., for $S_{<i>} = [a, b, c]$,, $\Phi(S_{<i>}) = [a, b, c, a^2, a*b, a*c, b^2, b*c, c^2]$; μΦ is the mean of the components of Φ, and, σ_{ϕ} is the standard deviation of the components of Φ. The output of the network, Y, is defined from V by:

$$Y_{<i>} = \frac{V_{<i>} - \min(V)}{\max(V) - \min(V)} \tag{26}$$

Polynomial features degree was initially chosen to be of second order and proved to be effective. A higher order degree was tested, but no considerable improvements were observed.

4.3.2. Model Structure and Hyper-Parameter Tuning

The guidelines that were used to select the hyper-parameters for the ANN Regression-only Model architecture shown in Chapter 5 were as follow:

- The number of input nodes is 15 (obtained from the second order polynomial feature extension of 4 performance figures). After applying *gridsearch* (i.e. iterative method that exhaustively considers all parameter combinations that the designer wishes to explore), the number of selected hidden layers was 3. As a rule of thumb, the number of nodes should increase in the first hidden layer to create a rich encoding (120 nodes), and then, decreases toward the output layer to decode the predicted circuit sizing (240 and 60 nodes in the second and third hidden layers, respectively). The number of output nodes depends on the number of devices' sizes from the topology being studied (e.g. 12 for VCOTA and 15 for Two Stage Miller);
- In initial testing, Sigmoid was used as the activation function of all nodes from all layers, except the output layer, but ReLU ended up being the preferred choice. As mentioned in Chapter 3, Section 3.2, the gradient using ReLU is better propagated while minimizing the cost function of the network. Output layer nodes don't use activation functions because we wish to find the true values predicted by the network instead of approximating them to either end of an activation function;
- Initially, the SGD optimizer was used, but later dropped in favour of Adam, which has better documented results. Parameters chosen for Adam, such as learning rate, are the ones recommended by [34] (these parameters are addressed in the next Section of this Chapter);
- The models were first designed to have good performance (low error) in the training data, even if overfitting was observed. This method allowed to determine the minimum complexity that can model the training data. Overfitting was then addressed using L2 weight regularization, as shown in Figure 18;
- Initial random weights of the network layers were initialized by means of a normal distribution;
- Model performance was measured through a MAE loss function, while the training of the model was performed using a MSE loss function;
- A high number of epochs (5000) was chosen for the training of early networks to ensure that the training reached the lowest possible error. One epoch occurs when all training examples execute a forward pass and a backward pass through the network. It is often a good practice to train the network for a high number of epochs, save the network weights from the training, and perform future testing using those weights with a lower number of epochs (e.g. 500);
- A variable number was chosen for batch size, between 256 and 512, depending on the number of epochs. Batch size is the number of training examples in one forward pass/ backward pass;
- Finally, *gridsearch* is done over some hyper-parameters (number of layer, number of nodes per layer, non-ideality and regularization factor) to fine tune the model.

The hyper-parameters chosen for this architecture are summarized in Table 2.

Table 2 – Hyper-parameters for the Regression-only Model

Hyper-parameter	Name/ Value	
Input Layer	1 Layer (15 nodes)	
Hidden Layers	3 Layers (120, 240, 60 nodes, respectively)	
Output Layer	1 Layer (12 nodes for VCOTA topology or 15 nodes for Two Stage Miller topology)	
Activation Function	ReLU	
Optimizer	Adam ($learning\ rate = 0.001$)	
Kernel Regularizer	L2	$lambda = 1.3E-05$ (VCOTA)
		$lambda = 2.0E-05$ (Two Stage Miller)
Kernel Initializer	Normal Distribution	
Loss Function	MSE	
Accuracy Metrics	MAE	
Number of Epochs used for Training	500 to 5000	
Batch Size used for Training	256 to 512	

4.3.3. Training

The loss function, L_1, of the model that is optimized during training is the MSE of the predicted outputs Y' with respect to the true Y plus the L2 norm of the model's weights, W, times the regularization factor λ, according to (27).

$$L_1 = \frac{1}{M}\sum_{j=1}^{M}((Y'_{<j>} - Y_{<j>})^T(Y'_{<j>} - Y_{<j>})) + \lambda\|W\|^2 \quad (27)$$

The training of the models is done using the Adam optimizer [34], a variant of stochastic steepest descent with both adaptive learning rate and momentum that provides good performance. Moreover, it is quite robust with respect to its hyper-parameters, requiring little or no tuning. It is well suited for problems that are large in terms of data and appropriate for problems with noisy or sparse gradients. Adam's configuration parameters are as follows:

- **alpha**: also referred to as the learning rate or step size, i.e. the proportion in which weights are updated. Larger values result in faster initial learning before the rate is updated. Smaller values result in slower learning during training;
- **beta1**: the exponential decay rate for the first moment estimates;
- **beta2**: the exponential decay rate for the second-moment estimates. This value should be set close to 1.0 on problems with a sparse gradient;
- **epsilon**: parameter that should be as close to zero as possible to prevent any division by zero in the implementation;
- **decay**: learning rate decay over each update.

Authors of the paper proposing Adam as a stochastic optimizer, suggest using $alpha = 0.001,\ beta1 = 0.9,\ beta2 = 0.999, epsilon = 10^{-8}$ and $decay = 0.0$ as parameters.

When training a supervised learning model, the main goal is to achieve an optimal generalization of unseen data. When a model achieves low error on training data but performs much worse on test data, we say that the model has *overfit*. This means that the model has caught on to very specific features of the training data, but hasn't really picked up on general patterns. The best way to evaluate this behaviour is through error analysis. This can be done by plotting test and training data's prediction error over model complexity (Figure 17). Test error is also commonly referred to as generalization error because it reflects the error of the model when generalized to previously unseen data. When we have simple models and abundant data, we expect the generalization error to resemble the training error. When we work with more complex models and fewer examples, we expect the training error to go down but the generalization gap to grow. Some factors that may affect generalization error are: the number of hyperparameters, the values taken by them and the number of training examples.

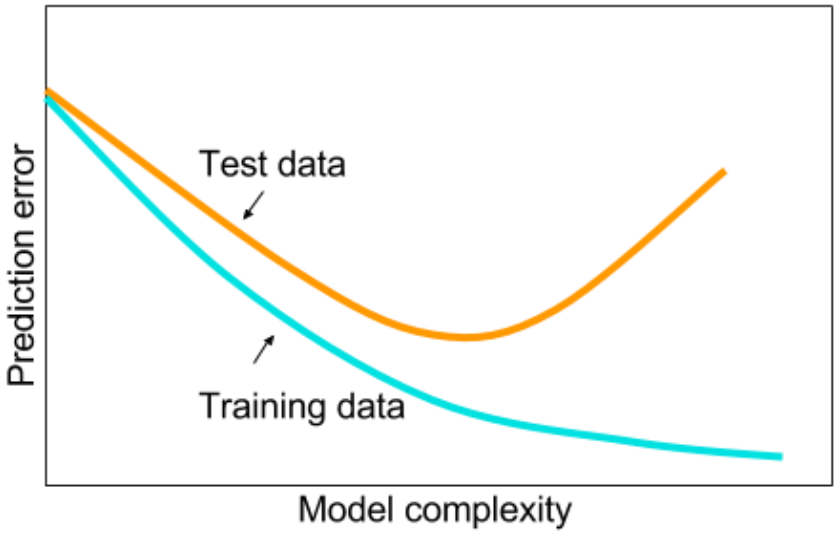

Figure 17 - Example of an overfitting model.

Model complexity is governed by many factors and its definition is not straightforward. For example, a model with more parameters might be considered more complex. A model whose parameters can take a wider range of values might be more complex. Often with neural networks, it is common to think of a model that takes more training steps as more complex, and one subject to *early stopping* as less complex. Complexity is no exact science, though, but it can be loosely defined by models that can readily explain *arbitrary* facts, whereas models that only have a limited expressive power but still manage to explain the data well are probably closer to the truth.

L2 regularization was used in the models proposed in this Chapter, as seen in (27). This is a technique that helps overcoming overfitting and is a regularization term that is added to the loss function in order to prevent coefficients to fit perfectly and under-generalize. There are two regularization techniques: L1 and L2. The difference between these two is just that L2 is the sum of the square of the weights, while L1 is just the sum of the weights. L2 was chosen because of its computational efficiency.

The use L2 regularization proved to be effective in the case study. Initial testing cases, where only input and output layers were considered, didn't achieve overfitting in the validation set because the

complexity of the model was not high enough to produce that effect. When hidden layers were later added with a variable number of nodes, model complexity started to continually increase until overfitting became apparent, a behaviour that can be observe in Figure 18 a). By adding the L2 regularization term, overfitting was completely negated, which can be observed in Figure 18 b).

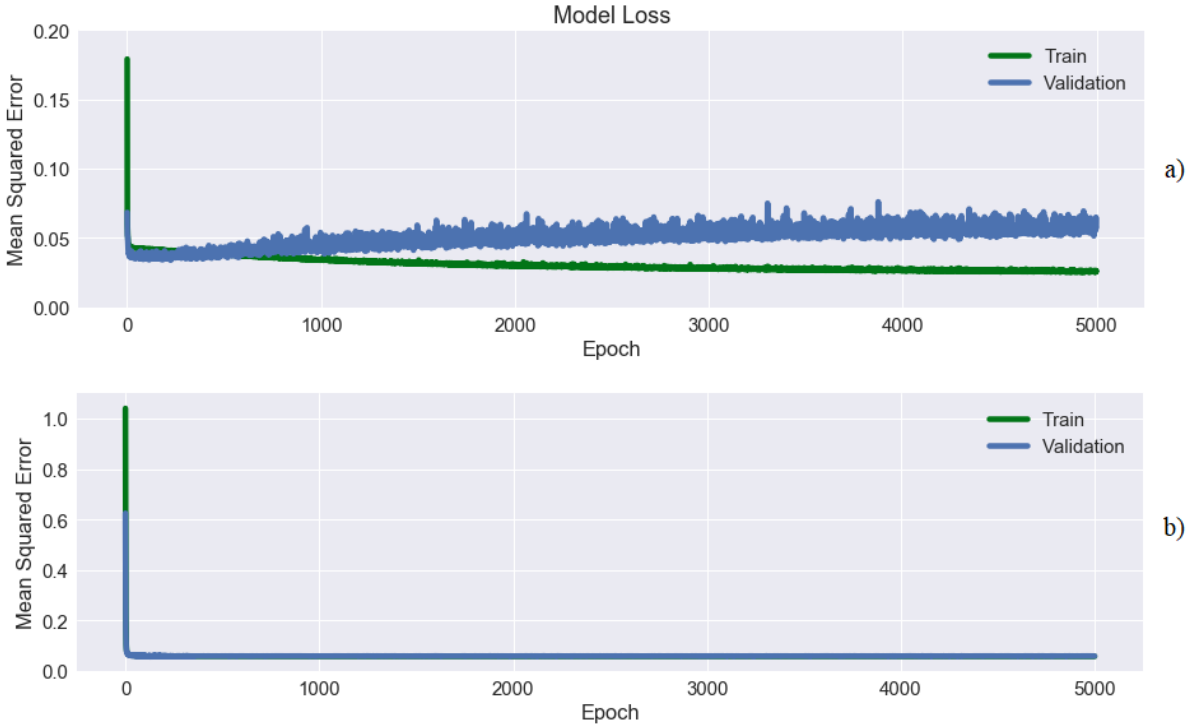

Figure 18 - Evolution of prediction error on train and validation sets during training: (a) ANN that overfits the training data, showing high error on the validation set. (b) Same ANN trained with L2 norm weigh regularization, showing better performance on the validation set.

4.3.4. Transfer Learning

Transfer learning is a common practice in the deep learning community, where an ANN trained with a large dataset is repurposed for other, similar, applications where data is scarce or, training is more expensive [33].

In the context of analog EDA, transfer learning creates opportunities in technology and/or topology migration, where an ANN trained for a functional block might be able to accelerate and reduce the data required to extend their applicability to other technology nodes, corners and variability awareness, and/or other circuit topologies.

4.3.5. Sampling from the ANN

Sampling from the ANN is done using (24) P times, with Γ replaced by –Γ, i.e., we ask the model to predict a set of P sizing solutions given circuit performances that are better than the desired specifications. In case not all performance figures used to train the model are specified, then the corresponding component in the diagonal random matrix Δ from (24) should be a random value in the range of [-1, 1]. For instance, if the target specifications are gain bandwidth product (GBW) over 30 MHz and current consumption (IDD) under 300 µA, and, the model was trained with DC Gain, GBW

and IDD. A set of circuit performances given to the ANN could be, e.g., {(50dB, 35MHz, 290µA), (75dB, 30MHz, 285µA), (60dB, 37MHz, 250µA), …, (90dB, 39MHz, 210 µA)}.

The reasoning behind this sampling is that, even if the ANN has properly learned the designs patterns present in the performances of the sizing solutions in the training data, when the performance trade-off implied by the target specifications being requested are not from the same distribution than the training data, the prediction of the ANN can be strongly and badly biased. While using the augmented dataset described in Section 4.3.1 alleviates this bias, it is still better to sample the ANN this way. Another reason for this sampling is that a given set of predictions might not be accurate enough for some devices' sizes. Specific values of items from the sizing output might be very far from the ones desired, as we will see in Chapter 6, and sampling is a good way to circumvent this problem.

The selection of solutions from the P predictions of the ANN is done by simulating the predicted circuit sizing, and, either using a single value metric, such as some Figure-of-Merit (FoM) for the circuit, select the most suitable solution, or, using some sort of Pareto dominance to present a set of solutions exploring the trade-off between specifications.

4.4. Classification and Regression Model

The ANN architecture considered in this Section is similar to the one used for the Regression-only Model, but now there is an increased number of output nodes, as seen in Figure 19. The input features are now not only restricted to one class of circuits, but to three. The features still correspond to the same four performance measures used in the Regression-only Model. The output layer is now not only comprised of a series of nodes that represent the circuit's sizes, but also an additional node for each class of circuits present in the dataset. The loss function used in the training of the networks will also be different, now taking into account both errors from the regression and the classification tasks. The weights assigned to each error measures are malleable, but weights of 70% and 30%, respectively, were used as a starting point.

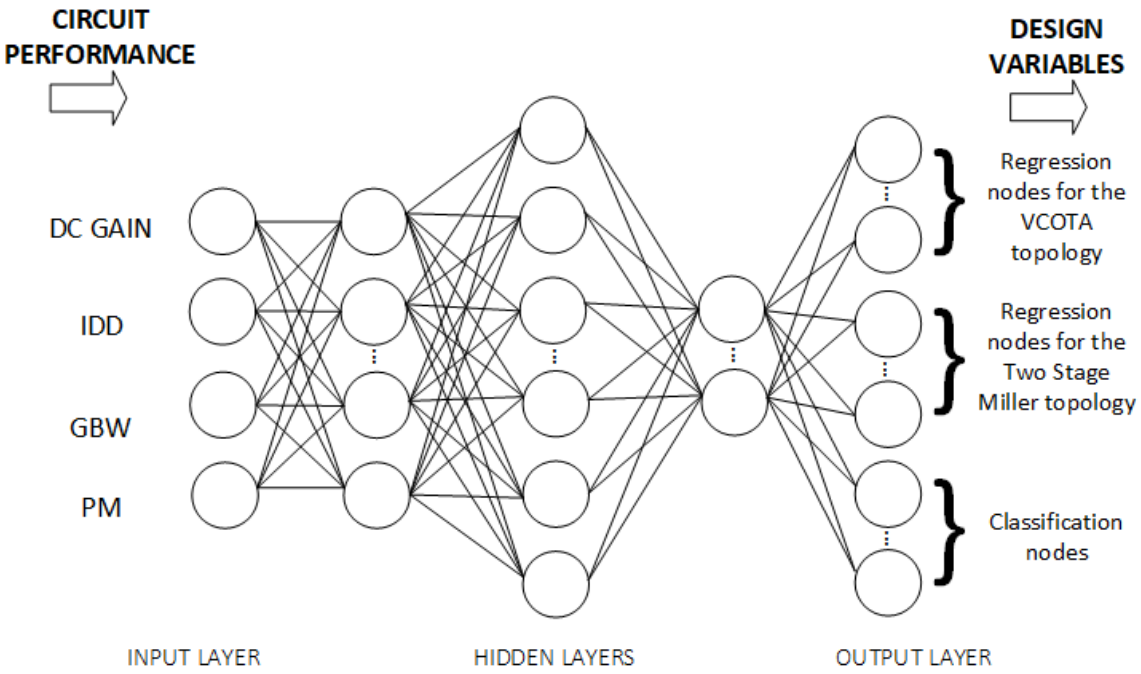

Figure 19 - Base Structure of the Classification and Regression Model.

For this case study, three classes of circuits were considered. The output nodes responsible for classification assign a probability to the predicted class.

4.4.1. Polynomial Features and Data Normalization

Data preparation for this model involved the same steps of normalization and data augmentation through polynomial features as the ones from the Regression-only Model.

4.4.2. Model Structure and Hyper-Parameter Tuning

The guidelines that were used to select the hyper-parameters for the ANN Classification and Regression Model architecture shown in Chapter 5 were as follow:

- The number of input nodes and the number of hidden layers are the same as the Regression-only model architecture. The number of nodes in the output layer increases in relation to the previous model, which are now 30. This reflects the fact that the network is now processing different circuit performances and target circuit measures: 12 nodes for the VCOTA topology and 15 nodes for the Two Stage Miller Amplifier topology, and 3 additional nodes that encode the circuit class;
- The activation function used in all nodes (except in the output layer' nodes) is ReLU;
- Adam was the chosen optimizer, with $learning\ rate = 0.001$;
- Overfitting was addressed using L2 weight regularization, as seen in Figure 20, after the model showed to have a good performance;
- Initial random weights of the network layers were initialized by means of a normal distribution;
- Model performance was measured through a custom loss function (see expression (30)) that takes into account the error measurements from the classification nodes and from the regression nodes. Different percentages are assigned to each type of error, 30% and 70% respectively.

Individual metrics were also used to prove the effectiveness of each task in the network. Regression error is calculated though a MSE function, while classification error is calculated through a Sparse Softmax Cross Entropy (SSCE) function;

- 5000 was the number of epochs chosen for initial testing. After having trained the first model, subsequent ANNs were trained with fewer epochs (500), using network weights from the ANN trained for 5000 epochs;
- A variable number was chosen for batch size, between 256 and 512, depending on the number of epochs;
- Finally, *gridsearch* is once again done over the hyper-parameters (number of layer, number of nodes per layer, non-ideality and regularization factor) to fine tune the model.

The hyper-parameters chosen for this architecture are summarized in Table 3.

Table 3 - Model Hyper-parameters

Hyper-parameter	Value
Input Layer	1 Layer (15 nodes)
Hidden Layers	3 Layers (120, 240, 60 nodes, respectively)
Output Layer	1 Layer (36 nodes)
Activation Function	ReLU
Optimizer	Adam ($learning\ rate = 0.001$)
Kernel Regularizer	L2 ($lambda = 2.0E-05$)
Kernel Initializer	Normal Distribution
Loss Function	Custom (see expression (30))
Accuracy Metrics	MAE, SSCE
Number of Epochs used for training	500 to 5000
Batch Size used for training	256 to 512

4.4.3. Training

The loss function, L_2, of the model that is optimized during training, is a weighted sum of two distinct losses – one from the regression task and the other from the classification task. Since this model's input features are not restricted to only one class of circuit performances, the regression loss will itself be a sum of the training errors from each circuit included in the dataset. Each individual regression loss is determined using MSE, like the previous model, while the classification error is measured through a SSCE function. This function measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class) [39].

The loss function, L_{class}, that is optimized for the classification task is obtained by computing the negative logarithm of the probability of the true class, i.e. the class with highest probability as predicted by the ANN:

$$L_{class} = -\log p(TrueClass) \tag{28}$$

The loss function, L_{reg}, that is optimized for the regression task is the MSE of predicted outputs Y' with respect to the true Y plus the L2 norm of the model's weights, W, times the regularization factor λ:

$$L_{reg} = \frac{1}{M}\sum_{j=1}^{M}((Y'_{<j>} - Y_{<j>})^T(Y'_{<j>} - Y_{<j>})) + \lambda\|W\|^2 \quad (29)$$

The total loss function, L_2, is the weighted sum between the two previous loss functions. Since there are two classes of circuits (excluding the third one, which is ignored in this function), there will be a distinct loss function value from each regression applied to each class. The MSE from each class is then multiplied by the true class predicted by the network in each step. This means that the MSE for the other class that was not predicted, will be neglected and become zero, i.e. if for a given step, VCOTA is the predicted topology, $TrueClass_1$ will be greater than zero, while $TrueClass_2$ will be equal to zero. The formulation of L_2 is as follows:

$$L_2 = 0.30 \times L_{class} + (0.70 \times (L_{reg1} \times TrueClass_1 + L_{reg2} \times TrueClass_2)) \quad (30)$$

The training of the models is again done using the Adam optimizer [34]. Other error metrics such as MAE and SSCE are also considered when validating the results. The results below are obtained for a model with one input and one output layer, and three hidden layers with 120, 240, 60 nodes each, for the demonstration of L2 regularization effectiveness.

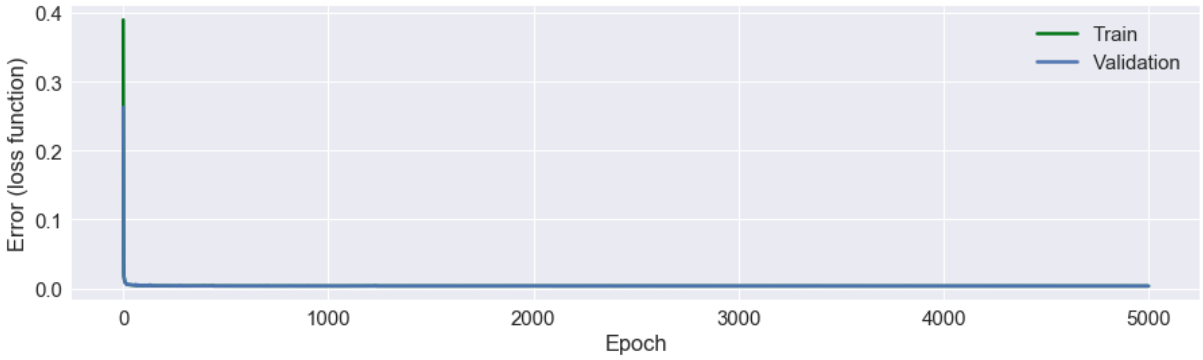

Figure 20 - Evolution of prediction error on train and validation sets during training, using L2 norm weigh regularization.

Similar to the previous architecture, model loss didn't show overfitting after L2 regularization was included, as shown in Figure 20.

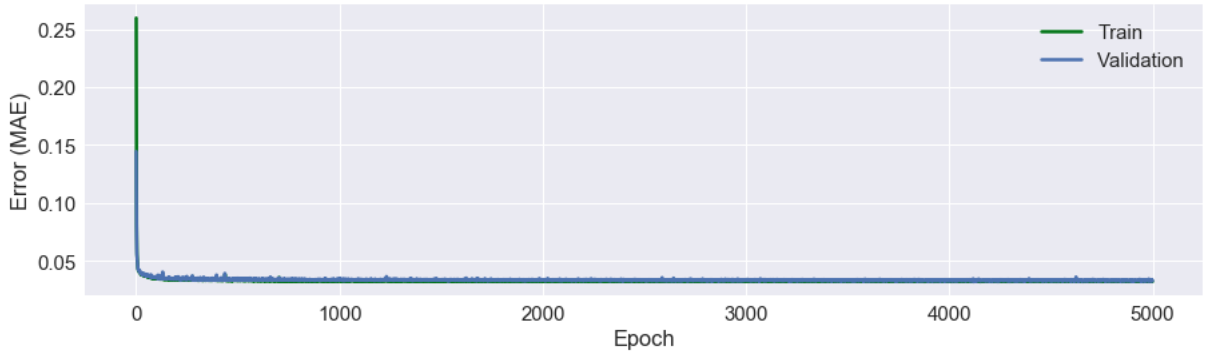

Figure 21 - Model Regression Error.

In this architecture, regression is performed using the same functions as the previous architecture: MSE for the training of the network and MAE for error measurement. Thus, the error obtained is similar to the one obtained in the Regression-only model, as shown in Figure 21.

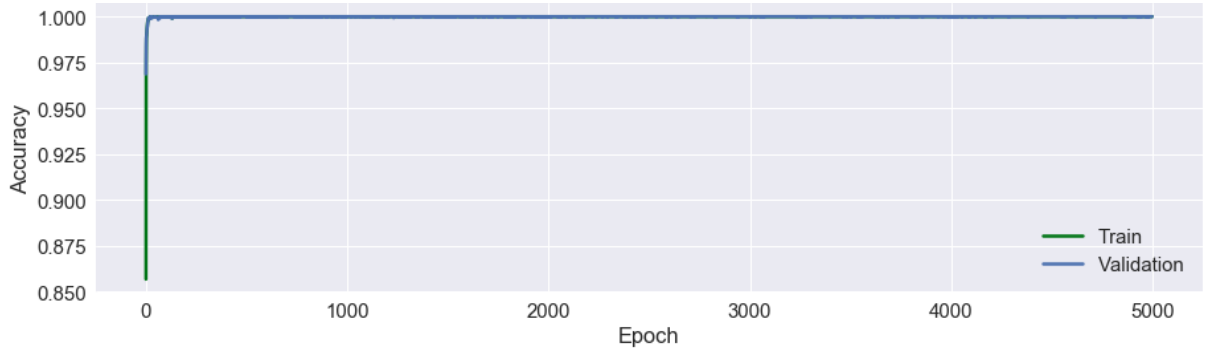

Figure 22 - Model Classification Error.

As shown in Figure 22, classes from the all the design points are correctly predicted on the validation set.

4.5. Conclusions

In this Chapter, the design flow for this work was presented, as well as two ANN architectures: a Regression-only model and a Classification and Regression model.

For both models, their structure and hyper-parameter selection were explained, as well as the process of polynomial features and data normalization applied to the input training data. The selection of the correct hyper-parameters, as being one of the most important tasks during the preparation of the model, was discussed. Furthermore, the preparation of the data for the training phase of the model was also analyzed. In particular, the concept of polynomial features was introduced. This is a good way to give the model more flexibility by overcoming the problem of having a low amount of input features.

Details regarding the training phase of the models, the concept of transfer learning and sampling results from the ANNs were also addressed. The use of L2 regularization was proposed as a way to avoid overfitting of the models. Transfer learning was introduced as being a popular practice in the

deep learning community, where ANNs trained with large datasets are repurposed for problems with fewer available data points. Finally, the process of sampling, where outputs of the ANNs are *filtered* to avoid possible biased solutions, was discussed.

Chapter 5. **Results**

In this Chapter, some results for both architectures are presented. All ANNs were implemented in Python 3.5 with Keras [40], using TensorFlow [41] as backend. The code was run, on an Intel® Core™ i7 Quad CPU 2.6 GHz with 8 GB of RAM.

5.1. Regression-Only Model

For proof of concept, the amplifier using voltage combiners for gain enhancement (VCOTA) from [42] was used, and, for a second example, a Two Stage Miller amplifier was considered. The circuit's schematic is shown in Figure 23, showing the devices with annotated design variables.

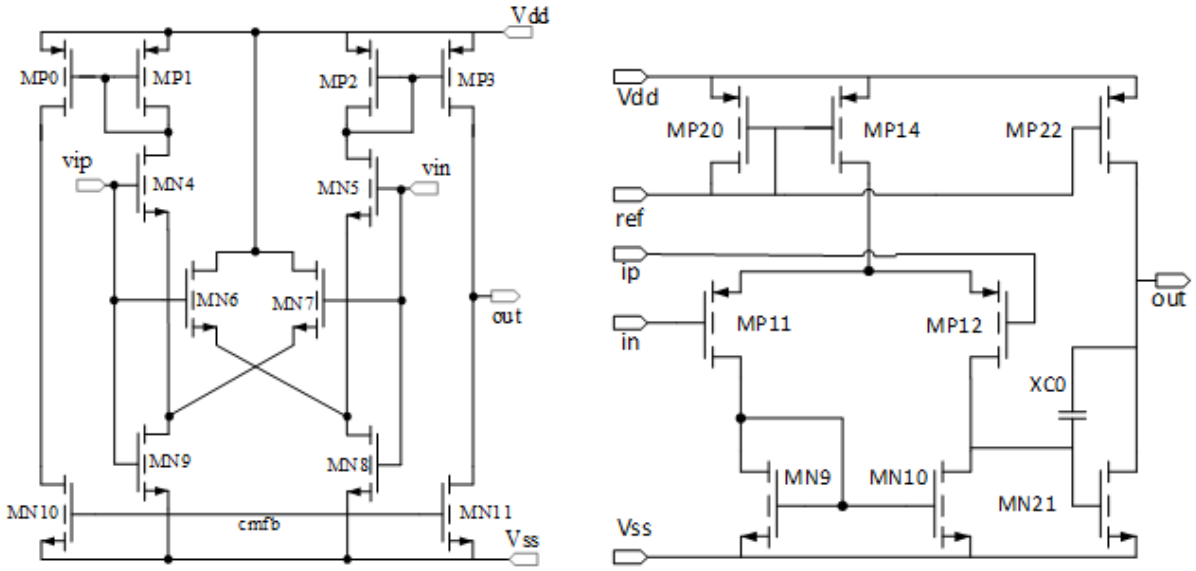

Figure 23 - Circuit schematic showing the devices and corresponding design variables: a) Single stage amplifier with gain enhancement using voltage combiners; b) Two Stage Miller amplifier. The variables lc and nfc are the length and number of fingers of the MOM capacitor XC0; lb, wb and nrb are the length, width and number of rows of MP20; lb, wb, nfbp and nrb are the length, width, number of fingers and number of rows of M14; lb, wb, nfb2 and nrb of MP22; lp, wp, nfp and nrp of MP11 and MP12; lal, wal, nfal and nral of MN9 and MN10; l2, w2, nf2 and nr2 of MN21.

The goal of this architecture is to learn design patterns from the studied circuits. By mapping from the device's sizes to the circuit's performances, using regression, the networks should be able to predict new devices' sizes by using as input features the desired circuit performances.

5.1.1. Dataset

Datasets were obtained from a series of previously done studies on this circuit for the UMC 130 nm technology design process, and contain only optimized circuit sizing solutions.

For proof of concept, the first study was performed on the VCOTA topology. Dataset for this example, i.e. Dataset-1, before any augmentation has 16,600 different design points. The second study was performed on the Two Stage Miller topology. For this example, the dataset before any augmentation, i.e. Dataset-2, has 5162 different design points. The circuit performances that were considered to train

the ANN in both cases were DC Gain, IDD, GBW and PM, and the ranges of values found in the dataset are shown in Table 4.

Table 4 - Performance Ranges in the two Datasets

		DC Gain	GBW	IDD	PM
VCOTA	***Max***	56.8 dB	78 MHz	395 uA	80º
	Min	44.7 dB	34 MHz	221 uA	60º
Two Stage Miller	***Max***	97.2 dB	102.8 MHz	0.8 uA	89.9º
	Min	59.8 dB	1.5 MHz	0.3 uA	55º

5.1.2. ANN Structure and Training

Three ANNs were trained for the VCOTA topology, i.e., ANN-1 to ANN-3. The structure considered has 15 input variables (obtained from the second order polynomial feature extension of the 4 performance figures from Table 4), 3 hidden layers with 120, 240, 60 nodes each, and, the output layer has 12 nodes. The trained parameters are the ones described in Chapter 4.

ANN-1 was trained on the original dataset, for 5000 epochs with batches of 512 samples. Its training took less than 15 minutes. ANN-2, was trained on the dataset augmented 40 times (almost 700K samples) for the same 5000 epochs. Its training took approximately 8 hours. ANN-3 was also trained on the same augmented dataset, for only 500 epochs, but was initialized with weights from ANN-1. Its training took less than an hour. Their performance after training on the training and validation sets is summarized in Table 5.

Table 5 - Performance of Trained ANNs for the VCOTA topology

		MSE Train	MSE Val.	MAE Train	MAE Val.
VCOTA	***ANN-1***	0.0159	0.0157	0.0775	0.0776
	ANN-2	0.0124	0.0123	0.0755	0.0750
	ANN-3	0.0124	0.0124	0.0754	0.0753

Table 6 indicates the average MAE between all the predicted and true devices' sizes from the test set.

Table 6 - Average MAE between the predicted and true devices' sizes for the VCOTA topology.

Devices' sizes items	MAE		
	ANN-1	ANN-2	ANN-3
_w8	2.75E-09	2.70E-09	2.65E-09
_w6	1.48E-05	1.62E-06	1.62E-06
_w4	5.24E-06	6.05E-06	6.04E-06
_w10	4.33E-06	4.59E-06	4.51E-06
_w1	8.91E-07	2.22E-06	2.17E-06
_w0	1.06E-05	1.84E-06	1.83E-06
_l8	2.98E-08	3.41E-08	3.35E-08
_l6	1.39E-07	1.56E-08	1.56E-08
_l4	1.21E-07	1.37E-08	1.36E-08
_l10	6.61E-08	6.99E-08	7.05E-08
_l1	5.32E-08	9.73E-08	9.98E-08
_l0	2.11E-08	5.08E-08	5.07E-08

In terms of performance, ANN-2 and ANN-3 showed the best results. From Table 5, we can observe that MSE and MAE error for the training and validation sets were lower for these networks, when compared to ANN-1, but not by a long margin. In terms of individual MAE for each devices' sizes, results were very similar across all ANNs, with a slight advantage for ANN-1.

Three other ANNs were trained for the Two Stage Miller topology, i.e. ANN-4 to ANN-6. The structure considered has 15 input variables (obtained from the second order polynomial feature extension of the 4 performance figures from Table 4), 3 hidden layers with 120, 240, 60 nodes each, and, the output layer has 15 nodes.

ANN-4, was trained on the original dataset, for 5000 epochs with batches of 512 samples, taking approximately 12 minutes to conclude the training. ANN-5 was trained on the dataset augmented 20 times (more than 100K samples) for 500 epochs. Its training took approximately 20 minutes. The training set was comprised of 96% of total data, while the validation set of 4%. ANN-6, was trained on the dataset augmented 20 times, for 500 epochs with batches of 512 samples, initialized with weights from ANN-1. The training set and the validation set were split with a 50% ratio (since the dataset was augmented, there is no problem in choosing the same percentage for both sets). Its training took approximately 20 minutes. Their performance after training on the training and validation sets is summarized in Table 7.

Table 7 - Performance of Trained ANNs for the Two Stage Miller topology.

		MSE Train	MSE Val.	MAE Train	MAE Val.
Two Stage Miller	***ANN-4***	0.0561	0.0414	0.1357	0.0937
	ANN-5	0.0072	0.0073	0.0590	0.0595
	ANN-6	0.0072	0.0073	0.0590	0.0597

Table 8 indicates the average MAE between all the predicted and true devices' sizes from the test set.

Table 8 - Average MAE between the predicted and true devices' sizes for the Two Stage Miller topology.

Devices' sizes items	MAE		
	ANN-4	ANN-5	ANN-6
_wb	1.27E-05	2.75E-06	2.86E-06
_wp	7.76E-05	8.59E-06	8.85E-06
_wal	2.78E-05	4.01E-06	4.20E-06
_w2g	7.35E-05	8.76E-06	8.86E-06
_lb	4.65E-06	7.77E-07	8.15E-07
_lp	4.54E-06	6.06E-07	6.11E-07
_lal	8.09E-06	9.41E-07	9.47E-07
_l2g	5.43E-06	6.57E-07	6.67E-07
_mbp	4.27E-00	5.64E-01	5.89E-01
_mb2	8.91E-00	6.49E-01	6.55E-01
_mal	9.34E+01	1.69E+01	1.77E+01
_mp	1.35E+02	2.84E+01	2.86E+01
_m2g	1.30E+02	2.84E+01	2.84E+01
_lc	7.49E-05	1.18E-05	1.20E-05
_nfc	1.44E+02	2.30E+01	2.31E+01

In terms of performance, ANN-5 and ANN-6 showed the best results. From Table 7, we can observe that MSE and MAE for the training and validation sets were significantly lower for these networks, when compared to ANN-4. From Table 8, we can see that ANN-4 performed much worse than ANN-5 and ANN-6, yielding higher error than almost all items.

5.1.3. Sampling the ANNs for New Designs

Sampling the ANNs was done as described in Section 4.3.5, with $P = 40$ and $\gamma = 0.15$, i.e., 100 random samples with a deviation of up to 15% from the specifications.

For the VCOTA topology, selection of the best solution was done by FoM for target 1, GBW for target 2, and IDD[a] and FoM[b] for target 3. It is easily seen by the performance of the obtained circuits that the ANNs learned the design patterns and can even extrapolate for specifications outside those of the training data. Moreover, circuits with FoMs larger than 1000 were obtained in all samplings of the ANNs. FoM is used in this example, as it was an optimization target in the process used to obtain the dataset.

Observing Table 9, we conclude ANN-1 can easily explore new specifications, but generates greater variability and worse designs when sampled. ANN-2 and ANN-3 are more stable, always generating good designs when sampled inside the training data. ANN-2 shows more limitations when trying to explore new specifications. ANN-3, because it used transfer learning from ANN-1, is more flexible to new specifications, but still lags when compared to ANN-1.

Table 9 - Performance of Sampled Designs for the VCOTA topology

	#HF[a]	DC Gain	GBW	IDD	PM	FoM [b]
Target 1		50 dB	60 MHz	300 uA	65º	
ANN-1	0.33	50 dB	63 MHz	318 uA	64º	1180
ANN-2	1	51 dB	61 MHz	320 uA	65º	1153
ANN-3	1	51 dB	63 MHz	325 uA	65º	1165
Target 2		40 dB	150 MHz	700 uA	55º	
ANN-1	0.24	44 dB	148 MHz	822 uA	54º	1082
ANN-2	0.21	49 dB	60 MHz	325 uA	73º	1106
ANN-3	1	43 dB	100 MHz	509 uA	61º	1182
Target 3		50 dB	30 MHz	150 uA	65º	
ANN-1 [c]	0.73	50 dB	3 MHz	141 uA	74º	116
ANN-1		50 dB	30 MHz	205 uA	74º	889
ANN-1 [d]		49 dB	67 MHz	329 uA	60º	1215
ANN-2 [c]	1	54 dB	38 MHz	240 uA	71º	950
ANN-2 [d]		54 dB	46 MHz	268 uA	64º	1033
ANN-3 [c]	0.97	55 dB	30 MHz	217 uA	69º	842
ANN-3 [d]		54 dB	54 MHz	309 uA	56º	1050

[a] Ratio of the number of solutions with FoM higher than 850 (the min value in the training data was 900) to the total number of samples; [b] MHz.pF/mA; [c] Best IDD; [d] Best FoM.

Observing Table 10, we conclude that ANN-5 and ANN-6 are capable of generating stable designs, despite being inaccurate for some items. ANN-2 shows some limitations for one target, but can generate designs similar to the other networks for the remaining targets.

Table 10 - Performance of Sampled Designs for the Two Stage Miller topology

	DC Gain	GBW	IDD	PM
Target 1	70 dB	10 MHz	30 uA	65º
ANN-4	77 dB	9 MHz	39 uA	65º
ANN-5	78 dB	9 MHz	35 uA	64º
ANN-6	77 dB	8 MHz	36 uA	64º
Target 2 [a]	100 dB	10 MHz	30 uA	65º
ANN-4	95 dB	12 MHz	48 uA	89º
ANN-5	87 dB	6MHz	41 uA	68º
ANN-6	85 dB	13 MHz	39 uA	42º
Target 3 [a]	70 dB	2 MHz	10 uA	65º
ANN-4	80 dB	3 MHz	29 uA	70º
ANN-5	75 dB	1 MHz	19 uA	75º
ANN-6	72 dB	1 MHz	18 uA	75º

[a] Targets outside the performances present in the dataset.

5.2. Classification and Regression Model

For the second architecture, the two previously studied circuits were again considered. The goal of this architecture is to, not only learn design patterns from these circuits, but also to identify which circuit class should be used for the set of input specifications. To do this, regression is applied to learn devices' sizes and classification is used to learn circuit classes.

5.2.1. Dataset

For this example, the used dataset, i.e. Dataset-3, has 15,000 different design points. Each third of the dataset belongs to three different chunks of data: two classes of circuits and one additional group of data. The first class refers to circuit specifications that belong to the VCOTA topology (encoded as 001); the second class refers to circuit specifications that belong to the Two Stage Miller topology (encoded as 010); the additional group of data is comprised of augmented data built up from the other two circuits (encoded as 100), but designed to not meet any of the specifications required by those circuits (i.e. maximum and minimum performance specifications are outside the required ranges). This last chunk was added to the problem to check if the network would be misled by input specifications that are out of the ranges required for the two studied circuits. The circuit performances that were considered to train the ANN were again DC Gain, IDD, GBW and PM, and, the ranges of values found in the dataset are shown in Table 11.

Table 11 - Performance Ranges in the Dataset

		DC Gain	GBW	IDD	PM
VCOTA	***Max***	56.8 dB	78 MHz	395 uA	80º
	Min	44.7 dB	34 MHz	221 uA	60º
Two Stage Miller	***Max***	97.2 dB	102.8 MHz	0.8 uA	89.9º
	Min	59.8 dB	1.5 MHz	0.3 uA	55º
Augmented Data	***Max***	117.1 dB	33.2 MHz	0.3 uA	89.9º
	Min	69.7 dB	1.5 MHz	0.1 uA	55º

5.2.2. ANN Structure and Training

Three ANNs were trained for this circuit, i.e., ANN-7 to ANN-9. The structure considered has 15 input variables (obtained from the second order polynomial feature extension of the 4 performance figures from Table 11), 3 hidden layers with 120, 240, 60 nodes each, and, the output layer has 30 nodes, which represent the different devices' sizes of the VCOTA and Two Stage Miller topologies (12 and 15 nodes, respectively) and the class to which they belong to (3 nodes to encode each of the three classes). For this dataset, the augmented data doesn't have any devices' sizes specified. Only performance figures were specified for this chunk of the dataset (as input features) so that a different class of circuits could be simulated.

ANN-7 was trained on the original dataset, for 5000 epochs with batches of 512 samples. Its training took less than 46 minutes. ANN-8 was trained on the augmented dataset, where 75K samples were generated for each circuit class, but only for 500 epochs. The network was initialized with weights from ANN-7. Its training took less than 50 minutes. ANN-9 was trained on the augmented dataset, where 100K samples were generated for each circuit class, for 5000 epochs. Its training took approximately 12 hours. Three performance metrics were used: a custom loss function (expressed in (30)), MAE for the regression nodes, and SSCE for the classification nodes. Their performance after training on the training and validation sets is summarized in Table 12.

Table 12 - Performance of Trained ANNs for the Classification and Regression model

	Loss Train	Loss Val.	Regression Train (MAE)	Regression Val. (MAE)	Classification Train (SSCE)	Classification Val. (SSCE)
ANN-7	0.0033	0.0034	0.0324	0.0329	1.0	1.0
ANN-8	0.0033	0.0033	0.0323	0.0324	0.9999	0.9999
ANN-9	0.0033	0.0033	0.0322	0.0321	0.9999	0.9998

Table 13 indicates the average MAE between all the predicted and true devices' sizes, and class prediction accuracy.

Table 13 - Average MAE between the predicted and true devices' sizes, and class prediction accuracy.

Devices' sizes items		ANN-7		ANN-8		ANN-9	
		MAE	Class Acc.	MAE	Class Acc.	MAE	Class Acc.
VCOTA	_w8	7.51E-09	100%	6.64E-09	100%	6.08E-09	100%
	_w6	5.78E-06		5.82E-06		5.69E-06	
	_w4	2.38E-06		2.19E-06		2.21E-06	
	_w10	1.68E-06		2.22E-06		1.50E-06	
	_w1	1.27E-06		9.09E-07		8.71E-07	
	_w0	6.73E-06		6.92E-06		7.30E-06	
	_l8	1.71E-08		1.75E-08		1.54E-08	
	_l6	5.59E-08		5.64E-08		5.48E-08	
	_l4	5.13E-08		4.93E-08		4.94E-08	
	_l10	2.89E-08		2.76E-08		2.63E-08	
	_l1	4.26E-08		4.14E-08		3.89E-08	
	_l0	2.67E-08		2.45E-08		2.84E-08	
Two Stage Miller	_wb	1.18E-06	100%	1.03E-06	100%	1.05E-06	100%
	_wp	4.00E-06		3.64E-06		4.21E-06	
	_wal	1.57E-06		1.72E-06		1.65E-06	
	_w2g	3.34E-06		3.18E-06		3.82E-06	
	_lb	3.37E-07		3.05E-07		3.20E-07	
	_lp	2.31E-07		2.58E-07		2.68E-07	
	_lal	4.08E-07		3.82E-07		3.89E-07	
	_l2g	2.79E-07		2.47E-07		2.82E-07	
	_mbp	2.76E-01		2.64E-01		2.77E-01	
	_mb2	3.73E-01		3.39E-01		3.65E-01	
	_mal	6.75E+00		6.98E+00		6.72E+00	
	_mp	1.28E+01		1.00E+01		1.18E+01	
	_m2g	1.14E+01		1.10E+01		1.14E+01	
	_lc	4.65E-06		4.41E-06		4.78E-06	
	_nfc	9.44E+00		8.73E+00		8.87E+00	

In terms of performance, all three ANNs showed favourable results. From Table 12, we can observe that Loss, Regression and Classification errors were similar for all three networks. The MAE for each individual devices' sizes was also very similar across all ANNs.

5.2.3. Sampling the ANNs for New Designs

From Table 14, we can conclude that ANN-9 can generate stable solutions for either topology, despite the considerable variability verified in certain specifications.

Table 14 - Performance of Sampled Designs

	DC Gain	GBW	IDD	PM	Topology
Target 1	50 dB	50 MHz	300 uA	60º	
ANN-9	54 dB	58 MHz	322 uA	57º	VCOTA
	54 dB	58 MHz	323 uA	58º	
	54 dB	58 MHz	322 uA	58º	
Target 2	70 dB	10 MHz	70 uA	60º	
ANN-9	83 dB	10 MHz	64 uA	59º	Two Stage Miller
	83 dB	10 MHz	63 uA	59º	
	83 dB	10 MHz	62 uA	59º	

5.3. Conclusions

In this Chapter, results for the two architectures were presented. In the Regression-only model, three ANNs were tested for each circuit topology, VCOTA and Two Stage Miller. In the Regression and Classification model, three ANNs were tested for a dataset comprised of design points from both topologies.

The first architecture consisted of a simple Regression model, with 15 input nodes, three hidden layers with 120, 240 and 30 nodes, each, and a variable number of output nodes: 12 when the regression task was applied to the VCOTA topology, and 15 when applied to the Two Stage Miller topology.

The second architecture consisted of a more complex model, which was capable of being trained on multiple topologies for the same analog function and select the most appropriate solution and its sizing. Its structure included 15 input nodes, three hidden layers with 120, 240 and 30 nodes, each, and 30 output nodes: 12 for the regression task applied to the VCOTA, 15 to the regression task applied to the Two Stage Miller topology, and 3 nodes to encode the predicted circuit class.

For both architectures, low error on the training and validation sets was achieved. In terms of prediction accuracy rate, the results were more favorable for the VCOTA topology in both architectures, where a low prediction error was achieved. The Two Stage Miller topology performed more poorly, but still was able to reach a low prediction error in some of the evaluated items. In terms of sampling the obtained results, both models were able to generate good designs.

Chapter 6. Conclusions and Future Work

This chapter presents the conclusions of all the work performed for this dissertation, and the future directions for the continuous development of ANNs applied to analog IC sizing.

6.1. Conclusions

This work presented deep learning methodologies that were used to develop ANNs that successfully predicted analog IC sizing for two amplifiers, given their intended target performances. This is a disruptive work, as no such approach has been taken in the field of analog and RF IC sizing, showing that a properly trained ANN can learn design patterns and generate circuit sizing that are correct for specification trade-offs, including those not provided in the training data.

ANNs proved to be very flexible models, capable of performing a satisfying mapping of devices' sizes from circuits performances. This technique showed to be quite effective in overcoming the high-non-linearity of this type of task.

The proposed methodologies were two: the first model showed to be capable of solving analog IC sizing as a direct map from specifications to the devices' sizes for a single topology; a second more complex model showed to be capable of being trained on multiple topologies for the same analog function and select the most appropriate solution and its sizing. Two different amplifiers were used in the training of the aforementioned models: an amplifier using voltage combiners for gain enhancement (VCOTA), and a Two Stage Miller amplifier.

Finally, the proposed goals for this work were achieved and a functional ANN model was created.

6.2. Future Work

This works only scratches the surface of the impact ANNs and deep learning may have in analog CAD and EDA. There are still several opportunities where deep learning and ANNs might improve analog EDA. A great possibility, and, at the same time one of the most challenging issues, is how to collect enough data to train such models. Given, of course, the importance data has, both in terms of quantity and quality, in the training of the models. Data collections as aggregation is a great opportunity to the EDA community to define intra and inter-organization protocols and formats to create rich and meaningful datasets that can potentially enable true automatic analog design reuse. This is, reuses of the design patterns instead of specific solutions.

There are also further improvements that could be applied to the models proposed in this work. One suggestion is to improve the structure of the networks. One hypothesis would be to increase the complexity of the model by building deeper ANNs. The use of L2 regularization proved to be effective in neglecting overfitting when the complexity of the networks increased, so the risk of overfitting the model by adding more hidden layers is virtually inexistent. Nevertheless, higher complexity does not necessarily equate better performance. A second hypothesis would be to experiment with other hyper-

parameters. The loss function, particularly of the Classification and Regression model, could be improved. Moreover, changing some parameters of the optimizer, such as the learning rate, could yield better results in the training phase of the model.

References

[1] N. Lourenço, R. Martins and N. Horta, Automatic Analog IC Sizing and Optimization Constrained with PVT Corners and Layout Effects, 2017.

[2] M. Barros, J. Guilherme and N. Horta, "Analog Circuits Optimization based on Evolutionary Computation Techniques," *PRODEP Program,* p. 6, 2010.

[3] N. Lourenço, R. Martins and N. Horta, "Layout-Aware Sizing of Analog ICs using Floorplan & Routing Estimates for Parasitic Extraction," *Design, Automation & Test in Europe Conference & Exhibit (DATE),* pp. 1156-1161, 2015.

[4] M. I. Jordan and T. M. Mitchell, “Machine Learning: Trends, Perspectives and Prospects,” *Science,* vol. 349, pp. 255-260, 17 July 2015.

[5] C. Cadwalladr and E. Graham-Harrison, "Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach," The Guardian, 17 March 2018. [Online]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election.

[6] M. Bojarski, D. D. Testa, D. Dworakowski and B. Firner, "End to End Learning for Self-Driving Cars," 2016.

[7] G. Zweig and S. Russell, "Speech Recognition with Dynamic Bayesian Networks," UC Berkeley, 1998.

[8] C. Bahlmann, B. Haasdonk and H. Burkhardt, "Online handwriting recognition with support vector machines - a kernel approach," in *Frontiers in Handwriting Recognition*, Ontario, Canada, 6-8 Aug. 2002.

[9] K. Dago, R. Luthringer, R. Lengelle, G. Rinaudo and J. Matcher, "Statistical decision tree: A tool for studying pharmaco-EEG effects of CNS-active drug," in *Neuropsychobiology*, 1994, pp. 91-96.

[10] F. Rocha, N. Lourenço, R. Póvoa, R. Martins and N. Horta, "A New Metaheuristc Combining Gradient Models with NSGA-II to Enhance Analog IC Synthesis," in *IEEE Congress on Evolutionary Computation*, Cancún, México, 2013.

[11] N. Takai and M. Fukuda, "Prediction of Element Values of OPAmp for Required Specifications

Utilizing Deep Learning," in *International Symposium on Electronics and Smart Devices*, 2017.

[12] P. Langley, "The Changing Science of Machine Learning," *Machine Learning,* vol. 82, pp. 275-279, 2011.

[13] J. VanderPlas, "Machine Learning," in *Python Data Science Handbook - Essential Tools for Working with Data*, O'Reilly Media, 2016, p. 541.

[14] P. Domingos, The Master Algorithm, Basic Books, 2015.

[15] J. A. Vrugt and B. A. Robinson, "Improved Evolutionary Optimization From Genetically Adaptive Multimethod Search," *PNAS,* vol. 104, pp. 708-711, 16 January 2006.

[16] I. Mackenzie, C. Meyer and S. Noble, "McKinsey&Company," October 2013. [Online]. Available: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

[17] "US Blogs," April 2017. [Online]. Available: http://usblogs.pwc.com/.

[18] B. Kamiński, M. Jakubczyk and P. Szufel, "A framework for sensitivity analysis of decision trees," *Central European Journal of Operations Research,* vol. 26, March 2018.

[19] K. P. Murphy, "Naive Bayes Classifiers," 2006.

[20] H. Zhang, "The Optimality of Naive Bayes," *American Association for Artificial Intelligence,* 2004.

[21] M. Minsky and S. Papert, Perceptrons, Cambridge, MA: MIT Press, 1969.

[22] B. E. Boser, I. M. Guyon and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classiers," 1992.

[23] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf and G. Rätsch, "Support Vector Machines and Kernels for Computational Biology," Department of Computer Science, Colorado State University, USA, 2008.

[24] T. Fletcher, "Support Vector Machines Explained," 2009.

[25] D. B. Fogel, "The Advantages of Evolutionary Computation," *Natural Selection, Inc,* 1997.

[26] S. Karamizadeh, S. M. Abdullah, M. Halimi, J. Shayan and M. j. Rajabi, "Advantage and

Drawback of Support Vector Machine Functionality," *International Conference on Computer, Communication, and Control Technology,* 2014.

[27] G. Wolfe and R. Vemuri, "Extraction and Use of Neural Network Models in Automated Synthesis of Operational Amplifiers," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS,* vol. 2, pp. 198-212, 2003.

[28] E. Dumesnil, F. Nabki and M. Boukadoum, "RF-LNA Circuit Synthesis Using an Array of Artificial Neural Networks with Constrained Inputs," in *IEEE*, 2015.

[29] M. V. Korovkin, V. L. Chechurin and M. Hayakawa, Inverse Problems in Electric Circuits and Electromagnetics, Springer, 2007.

[30] M. Segal and Y. Xiao, "Multivariate random forests," vol. 1, pp. 80-87, 2011.

[31] C. F. Higham and D. J. Highamy, "Deep Learning: An Introduction for Applied Mathematicians," 2018.

[32] X. Glorot, A. Bordes and Y. Bengio, "Deep Sparse Rectifier Neural Networks," 2011.

[33] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[34] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[35] F. Ghasemi, A. Mehridehnavi, A. Fassihi and H. Perez-Sanchez, " Deep Neural Network in Biological Activity Prediction using Deep Belief Network," in *Applied Soft Computing.*, 2017, p. 62.

[36] P. Domingos, "A Few Useful Things to Know about Machine Learning," *Communications of the ACM,* vol. 55, pp. 78-87, October 2012.

[37] N. Lourenço, R. Martins, A. Canelas, R. Póvoa and N. Horta, "AIDA: Layout-Aware Analog Circuit-Level Sizing," Lisboa, 2016.

[38] R. Gutierrez, Introduction to Pattern Analysis, OsunaTexas A&M University, Texas A&M University, 2016.

[39] T. Flow, "Tensor Flow - API Documentation," 2018. [Online]. Available: https://www.tensorflow.org/api_docs/.

[40] F. e. a. Chollet, "Keras," 2015. [Online]. Available: Github.

[41] e. a. M. Abadi, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online].

[42] R. Povoa, N. Lourenco, R. Martins, A. Canelas, N. Horta and J. Goes, "Single-Stage Amplifier biased by Voltage-Combiners with Gain and Energy-Efficiency Enhancement," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017.

[43] D. Pissarenko, "Neural Networks for Financial Time Series Prediction: Overview Over Recent Research," 2001-2002, pp. 35-121.

[44] S. L. Ho, M. Xie and T. N. Goh, "A comparative study of neural networks and Box-Jenkins ARIMA modeling in time series prediction," *Computers & Industrial Engineering ,* pp. 371-375, 2002.

[45] M. Khashei and M. Bijari, "A novel hybridization of artificial neural networks and ARIMA models for time series forecasting," *Applied Soft Computing,* pp. 2664-2675, 2011.

[46] P.-F. Pai and C.-S. Lin, "A hybrid ARIMA and support vector machines model in stock price forecasting," *Omega,* pp. 497-505, 2005.

[47] J. v. d. Herten, I. Couckuyt, D. Deschrijver and T. Dhaene, "Adaptive classification under computational budget constraints using sequential data gathering," *Advances in Engineering Software,* pp. 137-146, 2016.

[48] P. Singh, J. v. d. Herten, D. Deschrijver, I. Couckuyt and T. Dhaene, "A sequential sampling strategy for adaptive classification of computationally expensive data," Springer-Verlag Berlin Heidelberg, Belgium, 2016.

[49] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *Transactions on Evolutionary Computation,* pp. 182-197, 2002.

[50] G. Alpaydın, S. Balkır and G. Dündar, "An Evolutionary Approach to Automatic Synthesis of High-Performance Analog Integrated Circuits," *Transactions on Evolutionary Computation,* vol. 7, 2003.

[51] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *n Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*, 2010.

[52] N. Lourenço and N. Horta, "GENOM-POF: Multi-Objective Evolutionary Synthesis of Analog ICs with Corners Validation," in *GECCO'12*, Philadelphia, Pennsylvania, USA, 2012.

[53] R. Rahman, "Multivariate Random Forest," 2017.

[54] "Oracle," 2018. [Online]. Available: https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_nb.htm#BABIIDDE.

[55] C. E. Rasmussen, "Gaussian Processes in Machine Learning," Max Planck Institute for Biological Cybernetics, Tübingen, Germany, 2006.