Semi-Supervised Artificial Neural Networks towards Push-Button Analog IC Placement

António Paiva Lapas de Gusmão

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 Committee

Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. Ricardo Miguel Ferreira Martins
Member of the Committee: Prof. Helena Isabel Aidos Lopes Tomás

November 2019
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. Ricardo Martins and Prof. Nuno Lourenço for all their support throughout the development of the Master Thesis and for the valuable knowledge they provided along the way. I would also like to thank all of the teachers that guided me throughout my academic life, shaping my mind to what it is today. Finally, a word of gratitude to all who share their knowledge with everyone, be it through the internet or through the development of textbooks and articles. All knowledge is precious to the development of a fair, flexible and peaceful society, as is the one where we all wish to live.
Resumo

Neste trabalho é feito um estudo das possibilidades do uso de redes neurais na tarefa do posicionamento automático de um circuito integrado analógico, foi desenvolvido um modelo capaz de gerar posicionamentos válidos, na prática, instantaneamente. A metodologia usada explora uma abordagem semi-supervised ao problema onde nenhuma orientação é dada à rede durante parte do treino, em vez de treinar o modelo para imitar os posicionamentos fornecidos no dataset, a rede aprende a cumprir restrições topológicas. Como prova de conceito, neste trabalho consideram-se as restrições mais usualmente consideradas durante o desenvolvimento da solução (simetria e current flow) que são avaliadas directamente pela função de custo desenvolvida. Adicionalmente, um vector de input que contém a descrição das restrições a que cada dispositivo está sujeito é desenvolvido. Esta nova representação, juntamente com uma metodologia de padding, permite criar um modelo único para múltiplas topologias de circuito. O resultado é um modelo capaz de interpretar restrições topológicas resultando num aumento significante da generalização do conhecimento adquirido. O foco na avaliação da qualidade topológica das previsões feitas leva à produção de diversas soluções distintas de forma imediata, mesmo para um único input, fornecendo ao designer, instantaneamente, várias recomendações válidas de placement.

Palavras-chave: Redes Neuronais, Design de Circuitos Integrados Analógicos, Automação de Design Electrónico, Optimização de Posicionamento
Abstract

In this work, exploratory research using artificial neural networks is conducted to automate the placement task of analog integrated circuit layout design by creating a model which can generate a valid layout at push-button speed. The used methodology explores a semi-supervised approach to the problem where no guidelines are used during part of the training of the network and instead, the network learns to fulfill the most general topological constraints (symmetry, self-symmetry and current flow) which are evaluated directly in the loss function. Additionally, an input vector containing encoded the constraints to which each device is subject to is developed. This new input vector, along with a methodology for padding it for cases with a different number of devices, allows the prediction for multiple topologies using a single network. The result is a model capable of interpreting topological constraints which results in a clear increase in generalization of the knowledge trained. The focus on evaluating the topological quality of the predictions leads to the production of a multitude of novel placements, even for the same sizing scenario, giving the designer the ability to choose between several valid placements generated at push-button speed.

Keywords: Artificial Neural Networks, Analog Integrated Circuit Design, Electronic Design Automation, Placement Optimization
## Contents

Declaration ............................................................. iii
Acknowledgments ....................................................... v
Resumo ................................................................. vii
Abstract .............................................................. ix
List of Tables .......................................................... xiii
List of Figures .......................................................... xv
Nomenclature .......................................................... xix
Acronyms .............................................................. xix

### 1 Introduction

1.1 Motivation ......................................................... 1
1.2 Topic Overview .................................................... 2
1.3 Objectives ......................................................... 5
1.4 Thesis Outline ..................................................... 5

### 2 Related Work

2.1 State-of-the-Art in Analog IC Placement ......................... 7
2.1.1 Placement Constraints ........................................ 7
2.1.2 Placement Representation ..................................... 10
2.2 Automatic Layout Generation Approaches ....................... 12
2.2.1 Placement Optimization Considering Topological Constrains 12
2.2.2 Layout Migration and Retargeting ............................ 13
2.2.3 Layout Synthesis through Knowledge Mining ................ 14
2.2.4 ANNs Towards Automatic Layout Generation ............... 15
2.3 Machine Learning Overview ...................................... 17
2.3.1 Supervised/Unsupervised Learning .......................... 17
2.3.2 The Five Tribes of Machine Learning And Choice of Model 23
2.4 Neural Network Overview ........................................ 24
2.4.1 General Architecture and Behavior ........................ 25
2.4.2 Backpropagation Algorithm .................................. 26
2.4.3 Optimizers .................................................... 29
List of Tables

2.1 Device Sizing info Dataset used as input in [8] .............................................. 15
2.2 Device Positioning per Template ................................................................. 17

3.1 SSA Circuit Device’s Current Flow Constraints Description Vector .................. 45
3.2 CFSSA Circuit Device’s Current Flow Description ........................................... 46
3.3 SSA Circuit Device’s Symmetry Description .................................................... 47
3.4 CFSSA Circuit Device’s Symmetry Description ................................................. 47
3.5 SSA Circuit Device’s Sizing Description and Normalization ......................... 48
3.6 SSA Circuit Device’s Padded Current Flow Constraints Description Vector ........ 50
3.7 Model’s hyperparameters values ................................................................... 57

4.1 MSE Models Input Vector Structure Variants .................................................. 60
4.2 Mean Topological Error of MSE-NP and MSE-P Models .............................. 61
4.3 Template Selection Rate for Minimum Area Training ...................................... 61
4.4 Mean Topological Error of MSE Models with Scrambling of the Device’s Order ... 65
4.5 Mean Topological Error of Topological Models ........................................... 68
4.6 Mean Topological Error of Fully Topological Model with Test Set Augmentation (×100) ... 71
4.7 Folded Circuit Device’s Padded Current Flow Description ............................. 77
4.8 Mean Topological Error of the Top 9 Folded Circuit Predictions (Out of 1,000) .... 77
List of Figures

1.1 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. Reprinted from [3] .......................................................... 2
1.2 High-level view of the analog or mixed-signal IC design process. Reprinted from [6] .... 3

2.1 Two types of symmetry composed by symmetric pairs and self-symmetric devices. (a) Symmetric placement along the vertical axis. (b) Symmetric placement along the horizontal axis. Reprinted from [12] .................................................. 8
2.2 Two symmetric-placement examples of a symmetry group $S_1 = \{(b_1, b'_1), (b_2, b'_2)\}$. (a) $S_1$ forms a symmetry island. (b) $S_1$ cannot form a symmetry island. Reprinted from [12] ... 8
2.3 Example of successful common centroid layouts: (a) and (c), And an unsuccessful example in (b). Adapted from [14] .......................................................... 9
2.4 Regular shapes of placement. Reprinted from [16] .......................................................... 9
2.5 Two symmetric placements of the op-amp. The module with input ports is placed (a) inside the symmetry island (b) on the top boundary of the symmetry island. Reprinted from [17] .................................................. 9
2.6 (a) A placement with the consideration of the current flow constraints. (b) A placement without the consideration the current flow constraints. In blue: Monotonic current paths. In red: Non-monotonic current paths. Reprinted from [19] .................................................. 10
2.7 Example of a absolute coordinates defined placement. Reprinted from [22] ............. 11
2.8 (a) An admissible placement. (b) Its B*-tree representation. The binary tree is built in such a way that the relation between any two nodes contains information about their relative positioning in the produced layout, e.g. If node $n_j$ is the left child of node $n_i$, module $b_j$ must be located on the right-hand side and adjacent to module $b_i$. Reprinted from [23] .................................................. 11
2.9 Design flow of the major analog layout synthesis methodologies, distinguished by the number of previously existing solutions used during the process. Reprinted from [9] .... 12
2.10 Process of extracting hierarchical constraints from a legacy layout through the construction of a topology slicing tree. Reprinted from [26] .................................................. 14
2.11 Flow for layout migration. Reprinted from [26] .................................................. 14
2.12 (a) A circuit and (b) Its graph representation where the weights of the connections encode the type of connection between any two devices. Reprinted from [27] ................. 15
2.13 Single stage amplifier with gain enhancement: schematic with current paths highlighted. Adapted from [8] .............................................................. 16
2.14 Three different templates for the SSA circuit, each described by the relative positioning of each device. Adapted from [8] ................................. 16
2.15 An example of a supervised image classification problem where each image represents an example and the label's consist in the name of the fruit (e.g. plum). Reprinted from [32] 18
2.16 An example of a supervised classification problem where each example has two features \((x, y)\) and are labeled as either red or blue. Reprinted from [33] ................. 19
2.17 Visual representation of the sigmoid function \(S(x)\). ........................................ 20
2.18 An example of a supervised linear regression problem where each example has two features \((X_1, X_2)\) and are labeled with their \(Y\) value. The model learns the parameters that define the surface that best explains the behavior observed, minimizing the error between the labels and the model’s outputs. Reprinted from [30] .......................... 20
2.19 Simulated data in the plane, clustered into three classes (represented by orange, blue and green) by the K-means clustering algorithm. Reprinted from [30], .............. 22
2.21 An example of an ANN comprised of an input, an output and \(n\) hidden layers. Reprinted from [37] ................................................................. 25
2.22 Elementary artificial neuron overall functioning ............................................. 26
2.23 Network consisting of two hidden layers with a single neuron each. The network’s parameters consist in \((w_1, b_1, w_2, b_2, w_3, b_3)\). \(y_3\) is the network’s output and \(x\) is the network’s input. ................................................................. 27
2.24 Visualization of the influence of a model’s complexity in the overfitting phenomenon. A very simple model is not capable of describing the intricacies of the input/output data relation, however a very complex model creates an unrealistic description of the data. The best model lies somewhere in between. Reprinted from [45] ................. 31
2.25 General relation between training/test error and model complexity. Reprinted from [46] 31
2.26 General relation between training/validation error and training iterations (epochs). Reprinted from [47] ................................................................. 32
2.27 Standard neural network with and without dropout applied. Crossed units represent dropped out units. Reprinted from [49] ................................. 33
2.28 Linear activation function, used in the output layer in regression problems. .................. 35
2.29 Plot of the Rectified Linear Unit function ..................................................... 36
2.30 Plot of the Exponential Linear Unit function ................................................. 37
2.31 Map of California, vertical lines indicate constant longitude, horizontal lines indicate constant latitude ................................................................. 38
3.1 Cascode Free Single Stage Amplifier: schematic with current paths highlighted. Adapted from [58] ........................................... 44
3.2 Example of the scaled, descriptive vector of the device PM0 in the single stage amplifier circuit. ......................................................... 48
3.3 Structure of the input vector for the SSA topology. .......................... 49
3.4 Example of the, not yet scaled, padded descriptive vector of the device PM0 in the single stage amplifier circuit. .......................... 50
3.5 Structure of the padded input vector for the SSA topology. ............... 51
3.6 Device scramble transformation example. ..................................... 51
3.7 Device scramble at input vector level. ........................................ 52
3.8 Example of a placement for the SSA topology ............................... 55
3.9 Example of a placement for the single-stage amplifier with overlap area between devices in gray ....................................................... 56

4.1 MSE Models’ Without Device Scrambling Training and Validation Error Evolution. The error is evaluated using the MSE loss function. .................. 60
4.2 Example of test placement solution for the MSE-NP model with the mean topological error ......................................................... 62
4.3 Distribution of selected target template and the predictions’ most similar template. Each column $x$ is normalized so that the sum of the values for all $y$’s is 1. White indicates no value because either the template was not selected for any example or because no examples with a given target template were never mistargeted to a specific template. . . . 62
4.4 Average error for given target template/most similar template pairs. ................................. 63
4.5 Example of a prediction with high MSE error but low TLF error. .......... 64
4.6 Evolution of training and validation error for an MSE model with sizing based input vector with the order of devices scrambled for each example ....................... 65
4.7 Topological models’ training and validation Error Evolution. The error is evaluated using the TLF loss function. ................................. 66
4.8 TLF error components’ evolution throughout training. The error is evaluated using the TLF loss function. ........................................... 67
4.9 Sampling of the fully topological model’s test predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure. .......... 69
4.10 Sampling of the MSE initialized topological model’s test predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure. ................................. 70
4.11 Pearson correlation analysis between the components of the TLF. The $p$-values for each pair are shown in green. For the MSE initialized model, the analysis was made for the samples after the change in loss function. ...................... 71
4.12 Normalized, through minmax error components’ evolution during training for the fully topological model ....................................... 72
4.13 Sampling of the topological model's augmented test set predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure. ................................................................. 73
4.14 9 different predictions for a single sizing case of the SSA topology after 1000 scrambled copies of the same example were tested ................................................................. 74
4.15 9 different predictions for a single sizing case of the CFssa topology after 1000 scrambled copies of the same example were tested ................................................................. 75
4.16 Post-processed top 9 CFSSA predictions for a single sizing scenario ...................................................... 76
4.17 Folded Single Stage Amplifier: schematic with current paths highlighted .................................................. 76
4.18 9 different predictions for a single sizing case of the FSSA topology after 1,000 scrambled copies of the same example were tested ................................................................. 78
4.19 9 different predictions for a single sizing case of the FSSA topology after 1000 scrambled copies of the same example were tested ................................................................. 79
Acronyms

AI  Artificial Intelligence.
ANN  Artificial Neural Networks.
CAD  Computer Assisted Design.
CDV  Constraint Descriptive Vector.
CFSSA  Cascode Free Single Stage Amplifier.
EDA  Electronic Design Automation.
FSSA  Folded Single Stage Amplifier.
IC  Integrated Circuit.
MDP  Markov Decision Process.
ML  Machine Learning.
MS  Mixed-Signal.
MSE  Mean Squared Error.
MSE-NP  Mean Squared Error - Non Polynomial Features.
MSE-NPS  Mean Squared Error - Non Polynomial Features with Device Scrambling.
MSE-P  Mean Squared Error - Polynomial Features.
MSE-PS  Mean Squared Error - Polynomial Features with Device Scrambling.
RL  Reinforcement Learning.
SoC  Systems-on-Chip.
SSA  Single Stage Amplifier.
TLF  Topological Loss Function.
Chapter 1

Introduction

In this chapter an introduction to analog Integrated Circuit (IC) design is given with special focus given to automatic device placement in analog IC layout generation and the limitations that the current design flow faces. The standard procedures are presented and its limitations are discussed. Furthermore, the concept of Machine Learning (ML) is introduced and the possible use of this branch of Artificial Intelligence (AI) as a step towards the production of Electronic Design Automation (EDA) tools is analyzed.

1.1 Motivation

In the last decades, very large scale integration technologies have been widely improved, the semiconductors and systems market is expected to continue growing at an annual growth rate of almost 7% over the next 5 years. The percentage of analog and mixed-signal circuits in comparison to digital and discrete components has increased in the automotive sector since 2014 with below 20% to nearly 25% in 2018 and is expected to grow even further. The analog components market in all semiconductor segments is expected to grow until 2023 with a compound annual growth rate of 7.4% [1]. This market growth was deemed possible through the advancements made in the IC technology, exponentially increasing the density of transistors while inversely reducing its cost as described by Moore’s law [2]. However, the design of such complex multi-million transistors ICs is only possible due to the use of Computer Assisted Design (CAD) and Electronic Design Automation (EDA) tools that support the design process.

As analog ICs are difficult to design and reuse, designers have been replacing analog circuits by digital computing, however analog circuitry is needed to interface with the real world as some functionalities are inherently analog (e.g. sensing a system’s inputs like the signals from a microphone). As a result, a lot of real world implementations are the result of a mix between digital (used in most of the high level functions) and analog circuits, denominated as Mixed-Signal (MS) Systems-on-Chip (SoC).

While in most MS SoC the area occupied by the analog components is lesser than its digital counterparts, the effort put into its design is much higher as show in Fig. 1.1.

Due to this difference in required effort, a lot of research has recently been put into the development of new methods and approaches towards the automation of the design process of analog ICs.
Despite the considerable evolution of the tools used in automatic analog ICs design, these still have a long way to go to reach the levels of automation observed in digital circuit design automation tools [4]. Since the CAD tools for EDA have not yet reached the desired level of maturity, human intervention during all stages of the process is very much a given since the exploration of the solution space is done manually. The design flow is thus characterized by time-consuming, non-systematic and error-prone interactions between the designer and commonly used CAD tools such as circuit simulators, layout editing environment and verification tools [5]. Meanwhile, the problem is aggravated through the increase in demanded complexity of the systems, resulting in a higher number of transistors which in turn increase the complexity of the design problem by increasing the number of interactions which should be taken into account.

Overall, the difference in difficulty when designing analog ICs can be explained by the lack of a systematic design flow supported by mature EDA tools, the integration of analog circuits using technology optimized for digital circuits and the continuous nature of analog circuit’s signals’ which make them extremely sensible to parasitic disturbances, crosstalk, thermal noise, process variability, aging, device modeling accuracy and layout dependent effects which makes the design process very problem specific and greatly restricts the use of previous solutions. As such, there is high demand for analog EDA tools being met with intensive research efforts. Still, no automation solution for analog IC design matured to be widely used or recognized in the industry. Indicating that there is still long way to go in the making of analog IC design automation solutions that meet designer needs and increase their productivity.

1.2 Topic Overview

The design of analog IC systems can be split into seven stages as represented in Fig. 1.2 [6].

1. **Conceptual Design**: This is typically the product conceptualization stage, where the specifications for a design are gathered and the overall product concept is developed.

2. **System Design**: In this stage, the overall architecture of the system is designed and partitioned. Hardware and software parts are defined, additionally, the latter’s interfaces have to be specified.
3. **Architectural Design**: This stage is concerned with the decomposition of the hardware part in functional blocks (analog and digital) which, when connected, fulfill the specified behavioral specifications. Each of the composing blocks is defined through specifications it must fulfill and described through the use of hardware description language such as VHDL. The high-level architecture is then tested through the use of simulations to check whether the specifications defined in the first stage are fulfilled.

4. **Cell Design**: For analog blocks, a detailed implementation of its components is made. More complex blocks are further decomposed into a set of sub-blocks but ultimately, every block is described by a device-level circuit schematic with the corresponding sizing of the circuit parameters. During the process, manufacturability considerations (tolerances and mismatches) are taken into account in order to guarantee a high yield and/or robustness. The resulting circuit design is tested using SPICE-type circuit simulations.

5. **Cell Layout**: This stage translates the electrical schematic of the different analog blocks into a geometrical representation while minimizing the area of the resulting layout and reducing any parasitics which have a negative impact on the circuit’s performance.

6. **System Layout**: The generation of system-level layout of an IC includes system-level block placement and routing, power-grid routing and a description of any shielding and guarding which will act as a measure to prevent crosstalk and substrate coupling. The IC becomes testable and a detailed verification is performed with the embedded software.

7. **Fabrication and Testing**: This is the processing stage where the masks are generated and the ICs fabricated. Defective devices are rejected through ongoing tests during and after the fabrication.
Note that each of these stages is a complex process in itself and the process of analog IC design is usually a result of a lot of backtracking and verification which, even though represented in the figure, appear highly simplified. The purpose of the figure and the description of the several stages is to describe the milestones and requirements which have to be fulfilled during the process and does not describe the design flow in its entire complexity.

Currently, a lot of the EDA tools research has been made in an effort to automate the stages of both cell design [7] and cell layout [8]. This thesis addresses a task in the cell layout stage, specifically the placement of analog devices on the circuit's layout. For an effective placement, the location of the devices must follow several constraints. Symmetry, matching, grouping, current flow aware placement are of the utmost importance to minimize the impact of the layout in the circuit's desired performance. It is possible to distinguish between eight different constraints which should be taken into account during the development of the layout [9] resulting in a complex problem of multiple interconnected restraints.

Current approaches in the development of EDA tools towards analog IC layout generation can be split into three main categories differentiated by the amount of previously existing solutions (commonly denoted as legacy designs or legacy layouts) used as reference during the process of generating a solution [8]. Layout generation considering placement and routing constraints (without considering legacy designs), layout migration and retargeting from a previous legacy design or template, and layout synthesis with knowledge mining from multiple previously designed layouts. In practice, the different approaches all have the objective of fulfilling the eight constraints aforementioned and diverge on whether they do it explicitly (when no legacy layout is considered) or indirectly through the use of pre-existing solutions which fulfill these constraints and as such have the knowledge of the original designer embedded into them. The recent development in AI and ML methods, and available computational capabilities can be used to forge new approaches for the automation of analog IC placement.

ML is the process through 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 either showing it examples of desired input-output behavior or by rewarding system's (initially) random actions that achieve the desired behavior can be easier to program than to anticipate all possible responses for all inputs[10]. 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. A variety of techniques have been developed in an effort to optimize the process of pattern detection and in general, the optimal description of the problem in order to achieve the best results is usually the challenging task itself. Some of the most popular ML models nowadays are Artificial Neural Networks (ANN) which has regained a lot of its once lost popularity due to the increasing computational power widely available, allowing the increasing complexity of the networks which, in a general matter, results in better performance. Some approaches already make use of ML models such as reinforcement learning [11] and ANNs[8], the latter which serves as starting point for this thesis.
1.3 Objectives

The primary goal of this work is to accelerate and facilitate the process of analog IC placement through ML methodologies, more specifically, through the use of ANNs. The main objectives for this work are detailed below:

- Research the use of ANNs towards a push-button speed solution for the development of analog IC placement;
- Create an initial, relatively basic abstract description of some of the topological constraints of a circuit which can then be expanded to include all the remaining restrictions;
- Develop a versatile model capable of developing good placements for circuit's with a variety of different topologies and number of components;
- Study the possibility of an unconventional loss function which is not reliant on past examples and instead directly evaluates the prediction made in an effort to produce a model more robust which produces novel and better solutions.

In sum, a novel approach to both the network's loss function and the description of the circuit is tested with the objective of serving as proof of concept to show the potential of the approach which may be generalized into a variety of different problems and contexts.

1.4 Thesis Outline

The document is organized as follows:

- Chapter 2 presents the state-of-the-art in analog IC layout synthesis, namely on the placement part of the layout generation process. Moreover, an ML overview that explains the underlying nature of these systems is given. The chapter is concluded with an explanation of how ANNs work and some considerations that should to be taken into account when developing one;
- Chapter 3 starts by discussing the dataset used for the training of the network. Additionally, both the proposed input vector structure and the topological evaluation loss function are thoroughly explained. Finally, the overall structure of the network, as well as the chosen hyperparameters are described;
- In chapter 4, the tests done to the several considered models are explained and their results shown and discussed.
- Chapter 5 presents the conclusion drawn from this work and outlines future developments that can be done to further optimize the performance of the methodology presented in this work.
Chapter 2

Related Work

In this chapter, the constraints which influence the process of layout generation are explained along with the description of three main approaches of existing EDA tools. After that, an overview in ML is given, describing and comparing its main methods focusing on their strong points in order to frame ANNs in the wider landscape of possible models for analog IC placement. Finally, ANNs are more thoroughly studied through the deconstruction of the model and description and study of its several parts as well as comparing different used methods for each of these components.

2.1 State-of-the-Art in Analog IC Placement

Determining the location of each device or device group in the actual layout is one of the most critical steps in analog layout synthesis due to all of the constraints that should be taken into account since devices interfere with each other in ways that may affect the circuit performance. Furthermore, since the placement task precedes the routing generation, which has a definitive impact in the attainable interconnect quality, most of parasitic effects and consequent post-layout circuit's performance degradation are established once a placement is fixed. In this section, these constraints are described in detail, additionally, the two different approaches to the representation of a placement are described, as well as their strengths and weaknesses.

2.1.1 Placement Constraints

Recent studies address eight placement constraints to be taken into account during the development of a analog IC placement[9]:

1. Symmetry: A symmetry constraint can affect either a single device, a pair of devices or a group of them as represented in Fig. 2.1. It states that the centroid of the involved devices should be placed along a symmetry axis. This reduces the effect of parasitic mismatches as well as the circuit's sensitivity to process variations;
2. **Symmetry Island**: This is a stricter restriction which can be put on symmetry groups. It was shown in [13] that the larger the distance between symmetric devices, the greater the difference between their electrical properties resulting in performance hindering phenomena. As such, a symmetry island is defined by a group of connected components which share a symmetry axis and should be placed in close proximity in order to improve performance. An example is shown in Fig. 2.2.

3. **Common Centroid**: In order to minimize any stress-induced mismatch of the layout, the devices are decomposed into a set of equal device units and are arranged such that the centroid of the decomposed devices are located along the symmetry axis of both devices. Fig. 2.3 shows an example of two such common centroid layouts.

4. **Proximity**: A proximity constraint for a set of devices restricts the distance between the affected devices during placement. This constraint can help reduce interconnecting wirelength, improve matching among devices among others. One common application of proximity constraints is when considering a hierarchical approach to layout design[15].

5. **Regularity**: According to [16], the arrangement of the layout structures in all levels (device level, cell level and block level) using the regular shapes shown in Fig. 2.4 results in a more compact layout, better routability and less sensitivity to process variation.
6. **Boundary**: The boundary constraint restricts an in-out device to the boundary of a rectangular area around its device group as shown in Fig. 2.5. This results in a reduced critical net wire length between the group and its external connections, leading to less parasitic and better performance [17].

7. **Current/Signal Flow**: As the parasitic in the most critical current/signal paths of an analog circuit usually has great impact on circuit performance, the devices in the same current/signal path should be placed in an order that minimizes the wirelength of the current/signal paths. As such, current paths should be designed monotonically[18] as shown in Fig. 2.6

8. **Thermal**: If not taken into account, the improper placement of power devices close to thermally-...
sensitive devices might considerably reduce the circuit’s performance. As such, thermal radiating devices should be placed taking into account a thermal symmetry axis in order to reduce any thermal gradients on the chip and reduce any mismatch across devices [14].

These constraints make the placement task a complicated one due to the numerous amount of interdevice relationships to be taken into account. In general, existing solutions split into whether these constraints are handled explicitly or implicitly. In this thesis, an explicit approach to the problem is studied (in contrast to the one taken in [8]) with the intention of producing a versatile model capable of recognizing and meet these restrictions.

2.1.2 Placement Representation

In the literature, the many tools that attempt to automate placement generation can be distinguished by the representation of the cells, i.e. by how it encodes and moves the cells: absolute coordinates or topological representation.

- **Absolute Coordinates**: In the absolute representation, the optimizer moves the cells explicitly as each cell is represented by its absolute coordinates in an $\mathbb{R}^2$ coordinate system as done in [20] and shown in Fig. 2.7. Using this representation for the cells which make up the circuit, the previously defined constraints can easily be translated into an optimization problem in function of each cell's positioning, as such, this representation is recurrently used as a practical solution to layout constraint implementation. However, the lack of any restriction on the positioning of each cell allows the creation of invalid layouts with overlap between devices or unfulfilled symmetry constrains, as such, the search space includes both feasible and unfeasible solutions. Some approaches like [21] rely on a post-processing phase which converts the proposed solutions from absolute placement to a relative one where a deterministic cell slide technique is used in order to ensure a valid solution.

- **Topological Representation**: In the topological representation, the placement problem is encoded as a representative structure which details a set of constraints placed on the circuit [23] (visualized
Figure 2.7: Example of an absolute coordinates defined placement. Reprinted from [22]

in Fig. 2.8) [24], e.g. device 1 is to the right of device 2. Thus the tool does not move the
cells explicitly and instead changes the relative positions of cells by perturbing the representing
structure. Due to this difference in representation, the search space is greatly reduced as it only
contains admissible placements, however, it is not guaranteed that the optimal solution is included
in it. Finally, a packing procedure must be performed to transform the relative representation into
a floorplan while avoiding overlaps.

While it may seem that an absolute representation of the placement is more inefficient due to the in-
clusion of unfeasible solutions in the search space, recent efforts on stochastic/evolutionary algorithms
suggest that searching the extensive search space associated to the use of absolute coordinates can
compete with a search in the reduced solution space especially when taking into account the neces-
sity for packing or structural scan techniques as post-processing tools in the topological representation
framework [20]. Additionally, an absolute representation of the placement forms a very intuitive and
general definition of the problem allowing for easy further development and constraint handling.
2.2 Automatic Layout Generation Approaches

In order to increase the amount of automation in the process of analog layout synthesis, many new methodologies have been recently introduced. These can be divided into three main categories based on the number of legacy layouts used as reference when developing the solution for a given circuit. A legacy layout is defined as a previously developed layout solution which has been proven to have good performance and fulfill the circuit’s constraints. These previously developed solutions were a result of careful consideration and a long process of verification, as such, they contain embedded in them the knowledge and effort of their designer, and solutions which make use of these legacy layouts attempt to recycle the work put into these by using them as a guideline for the development of the current layout. These approaches ultimately differ in whether or not the topological constraints of the circuit are handled explicitly.

The three main categories are the following:

- Placement Optimization Considering Topological Constrains
- Layout Migration and Retargeting
- Layout Synthesis through Knowledge Mining

Fig. 2.9 represents the high level differences between the existing approaches.

![Diagram of design flow](image)

Figure 2.9: Design flow of the major analog layout synthesis methodologies, distinguished by the number of previously existing solutions used during the process. Reprinted from [9]

2.2.1 Placement Optimization Considering Topological Constrains

This approach doesn’t make use of any previous templates, instead, the topological constraints mentioned in 2.1.1 are codified in a cost function dependent on the devices’ positioning and are dealt with explicitly. In the absolute coordinate’s framework (introduced in 2.1.2), the circuit’s constraints can be
translated and quantified through a cost function. Take an example from [25] where the cost function to be minimized is defined as

\[ f(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \sum_i \alpha_i f_i(x), \]  

(2.1)

where \( x \) is the design variables vector (i.e. coordinates and, if allowed, the rotation of each cell) and \( \alpha \) is the weight's vector for each of the cost components; \( f_1(x) \) and \( f_2(x) \) are the two fundamental objectives to be achieved/minimized, circuit area and overlap. The remaining \( f_i(x) \) represent any remaining objectives. As such, the absolute representation of the problem allows a simple mapping between device placement and restriction satisfaction. For example, in [25], \( f_2(x) \) is forcibly driven to 0, that is, no overlap is allowed in the final solutions. Once the problem is defined as a cost function, optimization problem’s framework tools can be used to find a solution.

Using a relative description of a placement, these constraints can be enforced through either the penalization of unfeasible solutions, the avoidance of these solutions altogether (through a representation that directly enforces the fulfillment of these constraints) or through post-processing fixes. In [22], each placement proposed by a B*-tree is evaluated for the fulfillment of symmetry constraints and corrections are made such that these constraints are directly taken into account during the exploration of the solution space.

Notice that in [22], symmetry is the only out of the eight constraints mentioned in 2.1.1 which is enforced in the solution, in matter of fact, this methodology requires a very thorough definition of the problem in order to ensure a meaningful solution, i.e. it is needed to make explicit all the constraints which should be taken into account (which might differ on a problem basis) when designing the solution, or else the results might be quite unexpected.

### 2.2.2 Layout Migration and Retargeting

This methodology makes use of the fact that the circuit currently being designed has been designed before, thus assuring the existence of a proven quality layout for the considered topology. As such, the legacy layout can be used as a guideline for device placement in order to fulfill the circuit’s constraints. These approaches start by extracting from the existing layout some features which serve as the layout’s representation and as the guideline for the new layout[26][27]. This extracted representation works similarly to the relative placement representation described in 2.1.2 which encodes in its structure information about the relative placement of the devices. An example from one feature extraction procedure is shown in Fig. 2.10.

However, there are differences between both the circuits which prevent the direct use of the legacy layout, the difference in size of the devices may either make the layout invalid due to unmet constraints or enable further optimization of some metrics like chip area; and the possible use of a different process technology imposes some design rules which must now be taken into account. Fig. 2.11 shows the overall flow of layout migration.

These solutions enable the conservation and re-utilization of the knowledge and careful consider-
2.2.3 Layout Synthesis through Knowledge Mining

The methodology described in the previous section requires the existence of a legacy layout of the exact same circuit which constitutes a limitation to the design of circuits without previously qualified layouts and doesn’t make use of the immense knowledge database which all of the legacy layouts constitute in the scenario of existing matching sub-circuits. To address these limitations, some new solutions based in knowledge mining were proposed in [27] and [28].

In this approach the solution is based on several examples taken from a design repository composed by several quality-approved layouts for many different circuit topologies. The process consists in first matching sub-circuits from the input circuit to the sub-circuits extracted from any of the layouts in the repository, notice that the same sub-circuit can have more than one match in the repository allowing the generation of several possible layouts for each combination of sub-circuits matched. In [27], each layout in the repository is translated into a connected graph which encodes the nature of the connections between its composing devices as shown in Fig. 2.12, with the same being done to the circuit being currently designed. Once the circuit's are translated into graphs the problem to identify all common
sub-circuits which have the same device types, device constraints, and interconnections between the
two layouts becomes a problem of graph matching (i.e. graph isomorphism, sub-graph isomorphism or
sub-graph identification)

Figure 2.12: (a) A circuit and (b) Its graph representation where the weights of the connections encode
the type of connection between any two devices. Reprinted from [27]

This methodology allows the mixture of the spread-out knowledge in the repository. However, com-
parison with every layout of the dataset can prove computationally heavy, additionally conflicts are de-
terministically solved, as the acquired knowledge is not generalized but only applied being impossible to
generate a layout beyond training data.

2.2.4 ANNs Towards Automatic Layout Generation

A recent approach following the Knowledge Mining methodology makes use of ANNs in order to develop,
in theory, a model capable of mixing, for a given circuit topology, different previously produced legacy
layouts at push-button speed[8].

In [8], the dataset consisted in several combinations of sizing characteristics for each of the 12
devices that make up the Single Stage Amplifier (SSA) configuration introduced in [29] which can be
visualized in Fig. 2.13.

For each of the 12 devices of the circuit in each example, 5 sizing variables were stored, the channel’s
width, the channel’s length, the number of fingers and the encapsulated module’s width and height (the
device’s effective width and height in the layout). So, each example in the dataset consists in a specific
combination of values for each of the devices’ sizing characteristics (a sizing scenario). An example of
the device sizing info which is stored for each device in each of the examples is shown in table 2.1, these
were the features used for the model. This input vector is denominated 5-sizing.

Table 2.1: Device Sizing info Dataset used as input in [8]

<table>
<thead>
<tr>
<th>Example #</th>
<th>Device 0</th>
<th>...</th>
<th>Device 11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w_0 [µm]</td>
<td>l_0 [µm]</td>
<td>nf_0</td>
</tr>
<tr>
<td>0</td>
<td>17.5</td>
<td>0.36</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>17.7</td>
<td>0.36</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The solution was designed with the intention of generalizing a designer’s knowledge through the inclusion of a target for each of these examples. For each example, 12 placements were built using current CAD tools. Each of these placements was similar in every example with slight adjustments made in order to accommodate for the sizing differences between them, as such, these are denominated templates since they serve as general guidelines to determine the relative positioning of each device. Fig. 2.14 shows 3 of the 12 templates generated. Take the example of the middle placed template, it can be described by the relative positioning of each device, MNM5 is to the left of MNM4 which is above MNM7, whereas the leftmost template can be described by device MNM7 being above MNM4 and so on.

The process for generating these templates makes use of the layout migration tools explained previously, once 12 different placements are generated for a sizing scenario, through layout migration it is possible to migrate them for every other sizing case in the dataset.

As such, for each example in the dataset, each device’s absolute positioning in each of the 12 generated placements is stored as shown in table 2.2. For each of the examples only one of the generated
placements is used as target, the one with minimum area (in [8] the network is trained to generate three predictions, one with the smallest area, the smallest width and smallest height, however, for simplification purposes, consider only the smallest area output since the process is identical for the three cases changing only the criterion used for template selection).

Table 2.2: Device Positioning per Template

<table>
<thead>
<tr>
<th>Example #</th>
<th>Template 0</th>
<th>...</th>
<th>Template 11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>area_0</td>
<td>x_0</td>
<td>y_0</td>
</tr>
<tr>
<td>0</td>
<td>[µm]</td>
<td>[µm]</td>
<td>[µm]</td>
</tr>
<tr>
<td>1</td>
<td>2052.7</td>
<td>10.71</td>
<td>4.01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Once each sizing scenario has been assigned a placement as target, the model's learning process consists in approximating its predictions to the assigned target through the Mean Squared Error (MSE) loss function which can be seen in equation 2.4 (will be further explained later) where \( \hat{Y} \) is the prediction vector with \( x, y \) coordinates for each of the 12 devices, similarly \( \hat{Y} \) contains the \( x, y \) target coordinates for each device. By selecting only 1 of the 12 templates for each of the examples, the model is punished (through a high loss) for following one of the remaining templates instead. As such, the model is not encouraged to innovate and, for example, mix the existing templates as was intended, instead this behavior results in higher MSE error which the model is trained to minimize. This work will build on top of this developed model and to address this issue, a new loss function is designed, one which encourages the exploration of a multitude of different relative device positioning.

2.3 Machine Learning Overview

In this section, a brief overview in general Machine Learning concepts necessary for the full comprehension of the system developed (i.e. in order to contextualize the choices made) is given. The section starts by describing the distinguishable machine learning methodologies, first in relation to the guidelines given to the model in terms of desired behavior, then in terms of approach used to represent the problem's information and towards the development of a solution. This distinction is made with the purpose of pointing out the strengths of the used methodology.

2.3.1 Supervised/Unsupervised Learning

An ML system is said to be supervised if a predefined solution is imposed on the system, that is if there is an a priori defined expected behavior. Current taxonomies separate the area in three different levels of meta knowledge enforcement, supervised, unsupervised and reinforcement learning:

**Supervised Learning**

In supervised learning, each example fed into the system is inherently attached to its solution, i.e. the expected response of the system given the current example. This solution is denominated the example's
label. Supervised learning can be seen as learning with a teacher where the “student’s” (the model’s) answers \( \hat{y} \) (its output) are compared to the “teacher’s” (the model’s designer) answers \( y \) (the given labels). The model’s output is evaluated in relation to the given labels in the form of an error \( e(\hat{y}, y) \) and the model’s parameters are adjusted to reduce this error in an attempt to extrapolate the “teachers” knowledge to the rest of possible examples[30]. These labels can either be qualitative variables or quantitative, this distinction in output type has led to a naming convention to distinguish these prediction tasks: classification problem if the labels are qualitative and regression problems if the labels are quantitative.

An example of a dataset composed by features/qualitative label pairs is shown in Fig. 2.15, where the input (or features) is the picture of a fruit and its label is the name of the fruit represented. The system’s objective is to correctly “name” the fruits in each image. As such the problem is denominated as a classification problem. In general, a classification problem is one where the objective consists in associating to each example, one or more (in the case of multi-dimensional classification problems [31]) of \( n \) discrete classes existent in the context of the problem.

![Figure 2.15](image.png)

Figure 2.15: An example of a supervised image classification problem where each image represents an example and the label’s consist in the name of the fruit (e.g. plum). Reprinted from [32]

These categorical labels have to be converted to numerical values, one valid approach assigns to each of the problem’s \( n \) labels a specific integer value. However, as mentioned before, some problems might allow a single example to be inserted into various categories. Similarly to the example of fruit classification, consider an animal classification problem in which the system might be expected to classify the picture of a brown dog as a canine, a mammal and brown. An usual and flexible approach consists in defining the output as a vector of length \( n \) where each value is assigned to represent one of the problem’s features so when an example belongs to any given category, its assigned value in the vector is 1 and 0 if not. This implementation easily allows multiple label assignment, so assume this approach from here on.

Each example in the dataset is defined through \( m \) features which can be translated into numeric values, as such, each example can be plotted in an \( \mathbb{R}^m \) plane as a single point, i.e. a specific combination of the \( m \) features considered. In the case of Fig. 2.15, each example’s features can be the RGB value of each of the \( p \) pixels that form the image resulting in a \( m = 3p \) dimensional feature space. Each of these
points is previously assigned to one of the \( n \) classes (by the "teacher", i.e. the developer). As such, the model’s objective can be seen as splitting the feature’s \( \mathbb{R}^m \) space between the \( n \) possible classes acting as a \( \mathbb{R}^m \rightarrow \mathbb{R}^n \) mapping function. The model’s mission is to split the space in such a manner that RGB patterns of, for example agata potatoes, are mapped to the agata potato label assigned space (assigned by the encoding done previously by the developer).

A simpler, easier to visualize problem can be seen in Fig. 2.16 consisting of an input space of \( \mathbb{R}^2 \) with two possible labels, red or blue. The model consists in the hyperplane that splits the feature space in two defined by an equation similar to

\[
T = \langle W, X \rangle + b
\]  

(2.2)

where \( X \) represents the \( \mathbb{R}^2 \) feature vector for each example, \( W \) the weights associated to each feature and \( b \) translates the hyperplane with respect to the origin. As such, when replacing vector \( X \) with a specific example’s features, the resulting value \( T \) should be such that examples belonging to the same label return similar values, and examples of distinct labels return distinct values. Usually, as is the case in Fig. 2.16, two classes can be distinguished by whether the returning value is positive or negative.

This may seem contradicting to the statement that the model acts as a map between \( \mathbb{R}^m \rightarrow \mathbb{R}^n \) spaces since in this case with two labels, the returning value is a singular real value which encodes the class, e.g. 1 for red class, -1 for blue class. However, consider a case where \( T = 0.0005 \), that is, that the example in question is very close to the dividing hyperplane, this case should be signaled with some level of uncertainty. To encode this uncertainty, usually the value \( T \) is post-processed and converted to a \( \mathbb{R}^2 \) value which indicates the probability of the given example being classified as either of the classes. One usual post-processing approach makes use of the sigmoid \( S(x) \) function represented in Fig. 2.17.

Making use of the sigmoid function, the model’s \( \mathbb{R}^2 \) output consisting of the probabilities \( P_r \) and \( P_b \) of the example belonging to either the red or blue class are defined through

\[
P_r = S(T) \quad P_b = S(-T).
\]  

(2.3)

Figure 2.16: An example of a supervised classification problem where each example has two features \((x, y)\) and are labeled as either red or blue. Reprinted from [33]
However, as aforementioned, not all ML problems are classification problems. Another variant of ML problems are regression problems in which the model should be capable of detecting the trend that best explains the observations in the examples given as training data. Here, labels are not discrete as the case of red/blue classes (even though these can be relaxed to continuous values as aforementioned), labels are a real value which contain information about the expected behavior of the system. Take the simple example of a linear regression problem from [30] where examples are defined by two features $X_1, X_2$ and the label consists in a real value $Y$. The model acts as a $\mathbb{R}^2 \rightarrow \mathbb{R}^1$ mapping function, which learns the trend that best explains the examples seen during training phase, extrapolating it for the rest of the feature space. A visual representation of said problem is seen in Fig. 2.18. In a way, the model's behavior is similar to classification problems, mapping from a space of dimension equal to the number of features considered to a space of dimension equal to the number of labels, one in this case, whereas in the case of the classification problem illustrated in Fig. 2.16, two.
The best model is found by a number of iterations which consist in comparing the "student's" knowledge with the known answer and adjusting the model so that the "student's" answers \( Y \) become closer to the "teacher's" \( \hat{Y} \). An usual error function used is the Mean Squared Error (MSE) defined by

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.
\]

Building a model that minimizes the MSE function results in a mapping \( \mathbb{R}^2 \rightarrow \mathbb{R}^1 \) which results in the minimum average difference between the predictions made and the given knowledge.

It is important to keep in mind this effect of supervised models, they map from the feature space to the label space in such a way that minimizes the given error directly related to the given labels. The labels act as a guideline to the models behavior and is expected that this relation, fed to the model in the shape of features, labels pairs, is extrapolated to the rest of the feature space.

Unsupervised Learning

In supervised learning, the goal is to predict the value of an outcome measure based on a number of input measures; in unsupervised learning however, there is no outcome measure, and the goal is to describe the associations and patterns among a set of input measures. As such, in an Unsupervised system, no example is labeled, i.e. the model is not given any guideline as to how to behave in the form of expected "answers" for the given examples. One of the main difficulties of this approach is the fact that, unlike what happens with supervised learning, in this framework there is no direct measure of success due to the lack of given correct answers, making the task to ascertain the validity of the inferences made a matter of opinion which cannot be verified directly [30]. Usual applications of this methodology are \( K \)-means clustering and Association rule analysis.

Clustering, or data segmentation, is related with the task of grouping a collection of objects into subsets, denominated as clusters, in such a way that any object is more closely related to any other belonging to the same cluster than to any other which doesn't.

In order to form these groups, an essential part of the solution consists in using a metric to evaluate the similarity between any two objects of the dataset. As such, clustering methods attempt to group the objects based on the definition of similarity supplied to it [30]. In a way, this similarity function given to the model is similar to the loss function used in the supervised learning framework as they both quantify the knowledge intended to be learned. One popular clustering algorithm consists of the \( K \)-means which separates the data into \( K \) different clusters. The number of intended clusters is previously defined and there are several techniques which can be used in order to define the number of intended clusters[30]. Fig. 2.19 shows simulated data clustered into three different groups through the use of \( K \)-means.

\( K \)-means ultimately defines \( K \) points in the variable space which represent the center of each group considered, i.e. the most representative combination of variables for any given cluster. Each of the initial problem's are assigned to the cluster to which they're closer to (Euclidean distance). This distance is simply a quantification of the similarity between objects. As such, in the case of \( K \)-means, the similarity between any two objects \( i \) and \( j \) is defined through their distance \( d(i, j) \) in the variable space, defined
Figure 2.19: Simulated data in the plane, clustered into three classes (represented by orange, blue and green) by the K-means clustering algorithm. Reprinted from [30].

through, using the example from Fig. 2.19,

\[ d(i,j) = \sum_{k=1}^{2} (X_{ik} - X_{jk})^2 \]

(2.5)

where \( X_{ik} \) denotes the value of feature \( X_k \) for the object \( i \).

For cases where each object’s features are quantitative, both Euclidean distance and absolute difference are generally used as measures of similarity, however, take the case where some features are categorical and present no quantitative relation to each other such as an object’s color (as opposed to other discrete variables which can be ordered somehow, such as degree of preference (hate, dislike, like, love)), in these cases, it is necessary to explicitly define the similarity between any pair of values. Differences in the definition of similarity used might culminate in different results since this is what defines what the model pays attention to. Consider the existing similarities to supervised learning where the defined loss function codifies the intended behavior of the system.

**Reinforcement Learning**

Reinforcement Learning (RL) stands in between supervised and unsupervised learning and deals with learning in sequential decision making problems in which there is limited feedback. RL aims to develop an agent that learns how to behave in a given environment where the only feedback given consists in a scalar reward given after any decision made. The goal of the agent is to perform the actions which maximize its reward in the long run.

Recently, the most common framework towards modeling a sequential decision making problem has been the Markov Decision Process (MDP) [34]. In an MDP, a given environment is defined through states, actions, transitions between states and a reward function definition.
A set of environmental states $S$ is defined as the finite this will be important further on set $\{s_1, ..., s_N\}$ where the size of the state space is $N$, i.e. $|S| = N$. A state consists in an unique representation of all features that are important in the current state of the problem being modelled [34]. In chess, any specific configuration of black and white pieces across the board might represent a state.

The finite set of actions $A = \{a_1, ..., a_N\}$ represents what the agent is capable of doing in order to influence its environment, often resulting in a change of state. Taking the example of chess, the agent's possible actions consist in moving one of its pieces to any of the possible positions (according to the rules associated to each piece).

Consider the chess example, once the agent finds himself in any given state (a specific configuration of the pieces across the board), once it performs an action the state will inherently change since at least one piece of the board will change its position. As such, it is possible to define a state transition function $T(s, a, s') = P(s'|s, a)$ which gives, for any given current state $s$, the probability of transitioning to any state $s'$ given that the agent performed the action $a$. In the chess example, there is no uncertainty associated to any state transition, once a piece is moved, the configuration of the board changes accordingly, as such $T(s, a, s') \in \{0, 1\}$. However, some problems might have some environmental factors (factors out of the agent's control) that may influence the transition between states, in these cases $T(s, a, s') \in [0, 1]$.

The reward function constitutes the feedback given to the agent, this reward might be given in function of either the current state only, of performing a given action in a given state or of transitioning between two states via a specific action (most common approach).

An agent's goal is to develop a policy which indicates which action should be taken in any given state in order to maximize its expected reward [35]. As such, similarly to a supervised model's loss function and a clustering algorithm's similarity function, the reward function acts as the main influence of the agent's behavior.

RL is said to be in between supervised and unsupervised learning because no collection of expected actions is given to the agent (as the targets in supervised learning), instead the agent has the objective of "exploring" and "interpreting" the given reward function and come up with the correct actions for any given state by itself.

### 2.3.2 The Five Tribes of Machine Learning And Choice of Model

Machine Learning is a vast area in itself, in fact, in [36] ML methodologies are split into five different Tribes as illustrated in Fig. 2.20. All of the methodologies have something unique to offer and so the decision to use ML as a tool passes through analyzing the different approaches and selecting the one that better adapts to the problem and is more strongly connected to the objectives placed for the solution.

Once the existing methodologies have been analyzed, it is necessary to contextualize them in the problem domain and according to the project's objectives in order to choose the most appropriate description of the problem and the most appropriate solution's framework.

The problem consists in building a valid placement for a given list of devices, as such, the problem
consists in a regression problem where the model should be able to predict the chip location coordinates for each of the involved devices. Given this property of the problem, Bayesian and the Analogizers’ Support Vector Machine approaches, which are usually (but not only) associated with classification, are discarded.

Some current topological representation EDA tools already make use of symbolic approaches like in [23] which makes use of binary trees which encode the relation between devices, however, for cases where a lot of constraints have to be taken into account, developing the most appropriate and efficient representation might present a great challenge as shown in [22].

One of the main issues with current analog IC design is the time it takes for the development of the solution, as such, the chosen model should be one which once fully developed should be able to quickly provide a solution since this constitutes the main objective of the work. Taking this into account, Evolutionary approaches might be discarded due to the simulation time associated with modeling the natural selection of the different strategies.

The Connectionist's ANNs have been widely researched as tools for solving regression problems and once its layers’ weights have been properly set, it is capable of providing push-button speed solutions, these factors made the methodology the most appropriate for the development of the intended solution. Existing ML applications to the problem such as in [8] have already explored the use of ANNs towards automatic analog IC placement, however, in this work new features are included in order to create a more direct relation between the input and output data as well as a novel RL inspired approach to the development of the loss function in order to develop a more generalizing model.

2.4 Neural Network Overview

The flexibility of ANNs and their adaptability to almost any kind of problem made them the prime choice approach in this work, as well as the fact that they can efficiently infer rules relations from the input and output data. The main disadvantage of ANNs is that they have a high number of parameters to be tuned,
making the development process very time-consuming. However, the low prediction times that are the main goal of this work made them the chosen solution.

In this section the ANNs behavior and associated processes are explained in detail, as well as its hyperparameters, their influence in the model and the impact of optimally selecting the essential features for the description of the problem.

### 2.4.1 General Architecture and Behavior

The central idea behind ANNs is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features. ANNs are constituted by a network of interconnected elementary neurons usually forming \( 2 + n \) or layers, an input layer, an output layer, and \( n \) hidden layers as shown in Fig. 2.21.

![Figure 2.21: An example of an ANN comprised of an input, an output and \( n \) hidden layers. Reprinted from [37].](image)

The input layer has as number of neurons the same number of features selected for each example in the dataset. Similarly, the output layer the same number of neurons as the number of outputs desired. The number of both hidden layers and neurons in each of these is one of the hyperparameters of the model and will be further analyzed in 2.4.5, but in general a higher number of layers and neurons are capable of inferring more complex relations between the network’s input and output.

Each of the network’s elementary neurons is essentially a function which returns an output \( y \) as a function of its input connections \( x_1, \ldots, x_n \) in the form of

\[
y = \phi\left(\sum_{i=1}^{n}(w_ix_i) + b\right),
\]

where \( \phi \) is the neuron’s activation function, which constitutes another hyperparameter which will be further analyzed in 2.4.5, and \( b \) indicates the neuron’s bias which is generally treated as another weight \( w_0 \) associated to a constant input of value \( x_0 = 1 \) resulting in the simpler expression

\[
y = \phi\left(\sum_{i=0}^{n}(w_ix_i)\right),
\]

visualized in Fig. 2.22
Generally in ANNs, for a node in layer $j$, its inputs are the outputs of all neurons in layer $j - 1$ and its output will be an input for all neuron’s in layer $j + 1$. Taking the behavior of each neuron into account and the network’s architecture from Fig. 2.21 which is composed by several sequential layers of interconnected neurons, it is possible to conclude that the network’s output to any given input is the result of the inputs propagation through each layer of the network. The output of a given layer $j$ with $n$ neurons preceded by a layer with $m$ neurons can be described as a $\mathbb{R}^m \rightarrow \mathbb{R}^n$ function where the output of each neuron is given by 2.7. The learning process consists in the iterative adjustment of the network’s total neurons so that the network’s transformation of inputs into outputs optimizes a certain evaluation metric called the loss function.

ANNs are generally used in a supervised learning context, i.e. to each example used during the learning phase (phase where the objective is to adjust the model’s parameters to obtain gradually better predictions as opposed to evaluating its performance as is the case during test phase), there is a desired output associated. In these cases, the model’s outputs are evaluated in comparison to the pre-defined targets through a loss function similar to the MSE function 2.4 described in 2.3.1. The MSE loss function is widely used in supervised regression models and encourages the update of the networks parameters (its weights) such that the transformation of the input to output is as close as possible to the given targets. Notice that this doesn’t influence directly the relation that the network builds between the input and output data, several solutions might result in the same MSE loss and translate to significantly different relations as will be shown in 2.4.4. In extreme cases, the supplying of targets for every point in the input space would result in the development of the desired relation, however this gravely misses the point of the approach as the objective is to easily create a rough approximation of the desired relation by simply feeding to the model some example cases, as opposed to directly describing or even simply inferring this relation in its entirety which constitutes a significantly harder task.

The goal of the model’s designer then is to model both the loss function and input features in such a way that by giving the available examples as references, the model is capable of developing a close approximation to the relation between input and output that is easy to intuitively precept but hard to materialize and describe.

### 2.4.2 Backpropagation Algorithm

The learning phase of the model consists in a series of input-prediction iterations, where in each, the predictions made are evaluated via a defined loss function $L$ and the models parameters (each neuron’s
weights \( \{ w_{00}, \ldots, w_{mn} \} \) where \( w_{ij} \) denotes the weight \( i \) associated to neuron \( j \) are updated in order to minimize this function. In order to ensure that the weights’ update results in a smaller error, it is necessary to define the error’s derivative in relation to each neuron’s weight \( \frac{\delta L}{\delta w_{ij}} \). In order to achieve this, the backpropagation algorithm[38] is used.

Consider a network composed by two hidden layers with a single neuron each as shown in Fig. 2.23. The output for any given neuron of the network is defined in 2.7. Considering a supervised problem where the label associated with a specific example is denominated \( \hat{y} \) then the error between the prediction made and the given label might be defined by

\[
L(y_3, \hat{y}) = (y_3 - \hat{y})^2. \tag{2.8}
\]

Figure 2.23: Network consisting of two hidden layers with a single neuron each. The network’s parameters consist in \( (w_1, b_1, w_2, b_2, w_3, b_3) \). \( y_3 \) is the network’s output and \( x \) is the network’s input.

Through 2.7, it is possible to define the network’s output as the result of applying an activation function to the weighted sum of the last neuron’s inputs. By replacing in 2.8, the network’s loss is thus given by

\[
L(\phi_3(z_3), \hat{y}) = (\phi_3(z_3) - \hat{y})^2, \tag{2.9}
\]

where \( \phi_3 \) denotes the activation function of the last network’s neuron and \( z_3 \) represents the weighted sum of the same neuron’s inputs, defined as

\[
z_3 = w_3y_2 + b_3, \tag{2.10}
\]

where \( y_2 \) represents the output of the second to last neuron which in turn is the input of the last neuron. Remember also that the bias \( b_3 \) can be represented as a weight associated to a constant input of 1, so the task of updating the biases and the weights is technically the same. So in order to calculate \( \frac{\delta L}{\delta w_3} \), one can make use of the chain rule and define the partial derivative as

\[
\frac{\delta L}{\delta w_3} = \frac{\delta L}{\delta \phi_3} \frac{\delta \phi_3}{\delta z_3} \frac{\delta z_3}{\delta w_3}, \tag{2.11}
\]

where note that \( \frac{\delta z_3}{\delta w_3} = y_2 \), i.e. the previous layer’s output.

Through the same process, it is easy to define \( \frac{\delta L}{\delta b_3} \) for which the only change in the partial derivatives that decompose the goal derivative is the last term, which becomes \( \frac{\delta z_3}{\delta b_3} = 1 \). Notice that for this last neuron, no matter the number of inputs it has, the calculation of the loss’s partial derivative in relation to
any of its $n$ associated weights would be given by

$$
\frac{\delta L}{\delta w_{3i}} = \frac{\delta L}{\delta \phi_3} \frac{\delta \phi_3}{\delta z_3} \frac{\delta z_3}{\delta w_{3i}},
$$

(2.12)

where $i = \{0, ..., n\}$ denotes the weight associated to its $i$th input (consider 0 to be the bias). In matter of fact, for any weight of any neuron in the output layer, the process would be exactly the same.

Returning to the simplified example in Fig. 2.23, for the neuron in the previous layer, it is only necessary to continue the backtracking method shown in 2.11. Initially, it is possible to define the partial derivative $\frac{\delta L}{\delta y_2}$ similarly to the cases of $w_3$ and $b_3$, so

$$
\frac{\delta L}{\delta y_2} = \frac{\delta L}{\delta \phi_3} \frac{\delta \phi_3}{\delta z_3} \frac{\delta z_3}{\delta y_2}.
$$

(2.13)

Taking into account the expression for a neuron’s output from 2.7, it is possible to define $y_2$ as

$$
y_2 = \phi_2(z_2)
$$

(2.14)

where $z_2 = w_2y_1 + b_2$, so $\frac{\delta L}{\delta w_2}$ can be calculated through

$$
\frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta \phi_3} \frac{\delta \phi_3}{\delta z_3} \frac{\delta z_3}{\delta \phi_2} \frac{\delta \phi_2}{\delta z_2} \frac{\delta z_2}{\delta w_2}.
$$

(2.15)

However, the expression doesn’t remain the same if a model with $m$ output nodes is considered since the output of any neuron in the second to last layer would propagate to all of the $m$ nodes in the next layer, so when backtracking the error to any of the $n$ weights of any of the $k$ neurons in the second to last layer, $m$ paths should be considered resulting in

$$
\frac{\delta L}{\delta w_{2ik}} = \sum_{j=1}^{m} \left( \frac{\delta L}{\delta \phi_{3j}} \frac{\delta \phi_{3j}}{\delta z_{3j}} \frac{\delta z_{3j}}{\delta \phi_{2k}} \frac{\delta \phi_{2k}}{\delta z_{2k}} \frac{\delta z_{2k}}{\delta w_{2ik}} \right)
$$

(2.16)

This backtracking of the influences in the loss function can be generalized for any number of layers, for any number of neurons in each of them. The important point however, is that all of the involved functions, the neuron’s activation function and the loss function, should be differentiable (the weighted sum of a neuron’s input is inherently differentiable) in order to allow an efficient update of the model’s weights in each iteration.

One other simplification made in the example previously analyzed is the fact that only one example was considered. Take the example of the MSE loss function, it consists in averaging the square sum of the difference to the label across all examples. This doesn’t change the process however, as $\frac{\delta L}{\delta w_{jik}}$ for any weight $i$ for the neuron $k$ in layer $j$ is simply the average across all examples of the process previously explained. However, this fact underlines the fact that the calculation of each weights update is directly tied to the number of examples considered. This will be further discussed in 2.4.5.
2.4.3 Optimizers

The most simple way of updating any model's parameters iteratively in order to minimize a loss function \( L \) is through the Gradient Descent [39]. Consider a model with the parameter's vector \( W \) of length \( d \). Through the Gradient Descent method, the model's parameters can be updated from iteration \( t \) to iteration \( t + 1 \) through

\[
W_{t+1} = W_t - \eta \nabla L(W_t),
\]  

(2.17)

where \( \eta \) represents the method's parameter, the learning rate. The gradient \( \nabla L(W_t) \) gives the direction in the parameter space towards which the loss increases most drastically. As such, the direction \( -\nabla L(W_t) \) gives the direction in the parameter space towards which the loss decreases most drastically, therefore the parameters are updated through a step in the direction indicated by \( -\nabla L(W_t) \). The learning rate \( \eta \) represents the magnitude of the step taken. However, the magnitude of the step should not overshoot the function's minimum value. If the \( \eta \) value is too large, the model might never converge, or worse, diverge and explode. However, too small steps might take a lot of iterations to converge or become stuck in a local minimum of the loss function. Due to this challenge of choosing the correct value \( \eta \), several new optimizers have been proposed.

One very common concept in more complex methods is the concept of momentum introduced in [40] and its objective is to simulate the physical sense of acceleration [41]. It takes into account the past gradients and if its direction remains the same. This concept is mathematically encoded through

\[
W_{t+1} = W_t - \Delta W_{t+1}, \quad \Delta W_{t+1} = \beta \Delta W_t + \eta \nabla L(W_t),
\]  

(2.18)

Where \( \beta \) is usually set to \( \beta = 0.9 \). Notice the recursive nature of the update factor \( \Delta W_{t+1} \) which depends on the update factors of all previous iterations. The update on the parameters gets bigger as the iterations progress consistently, making it easier to escape from plateaus and local minima of the loss function. When the gradient changes its direction between iterations, the algorithm reduces the step taken as if the parameter update was affected by inertia.

A common issue with the mentioned approaches is that the same learning rate applies to all parameter updates. If the data is sparse and its features have very different frequencies, rarely occurring features should result in a larger parameter update[42]. Adaptive Gradient (AdaGrad) [43] addresses this issue: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. As such, the parameter \( \eta \) is now a vector of learning rates \( \eta \), one for each of the model's \( d \) parameters. The values for each of these learning rates also varies in time \( t \). The update of a given \( i \) parameter at instance \( t + 1 \) is defined through

\[
w_{t,i,t+1} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla L(w_{t,i}),
\]  

(2.19)

where \( G_{t,ii} \in \mathbb{R}^{d \times d} \) is a diagonal matrix where each diagonal element \( i, i \) is the sum of the squares of
the gradients with respect to \( w_i \) up to time step \( t \) through

\[
G_{t,ii} = \sum_{j=0}^{t} (\nabla L(w_{j,i}))^2,
\]

(2.20)

and \( \epsilon \) is a smoothing term that avoids division by zero (usually in the order of \( 1 \times 10^{-8} \)). The added adaptive term from 2.20, decreases the learning rate every time \( \nabla L(w_{t,i}) \neq 0 \). Thus, more frequent parameters are updated less drastically than less frequent terms. Notice that for very frequent parameters, the adaptive term keeps growing until the parameter's update is infinitesimally small and no new knowledge is acquired.

To avoid this accumulation in the adaptive term, some methods like RMSprop associate a decaying term \( 0 < \gamma < 1 \) (usually 0.9) to the sum of gradients in order to consider strongly the most recent gradient values [42]. The adaptive term for parameter \( w_i \) in instant \( t \) is thus defined through

\[
G_{t,ii} = \gamma(\nabla L(w_{t,i}))^2 + (1 - \gamma)G_{t-1,ii}.
\]

(2.21)

Finally, the Adaptive Moment Estimation (Adam) optimizer [44] mixes both the concept of the gradient's momentum from [40] and its second momentum, seen in the RMSprop, used to adapt the optimizer's learning rate. The gradient's first momentum in instance \( t \) is defined as

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(W_t)
\]

(2.22)

similarly to 2.18 with the addition of a bias-correcting term[42]. The gradient's second momentum is defined as

\[
v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(W_t))^2
\]

(2.23)

similarly to 2.21, again with the addition of a bias-correcting term. Finally the update of the parameters is given by

\[
W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t.
\]

(2.24)

The authors propose default values of 0.9 for \( \beta_1 \), 0.999 for \( \beta_2 \), \( 10^{-8} \) for \( \epsilon \) and 0.001 for \( \eta \) with which it has been shown to perform well in several context [44], reducing the need of adjustment. The adaptive learning rate makes Adam (and the other optimizers that use it) appropriate for sparse input data, while the use of momentum makes it faster to converge and less likely to become stuck in saddle points or local minima [42]. These factors make it the selected optimizer for the model.

2.4.4 Regularization

One phenomenon to which all ML models are subject to is overfitting. A model is said to be overfit if it is too well adjusted to the given training data. A model’s goal is usually to modulate a relation between the given input data and output data which, in general, represent real data. Relations between real data are usually not very complex, so the development of a very complex relation to explain the given training
data will most likely not describe the real relation existent between the input and output. Fig. 2.24 shows a classic example where the excessive complexity of the system results in a perfect fit for the training data, but is completely different from the real relation that generated the data.

![Figure 2.24: Visualization of the influence of a model's complexity in the overfitting phenomenon. A very simple model is not capable of describing the intricacies of the input/output data relation, however a very complex model creates an unrealistic description of the data. The best model lies somewhere in between. Reprinted from [45]](image1)

One common approach towards evaluating a model's overfit is through the comparison of the error for the training and test data (data which was not used for the model's parameter adjustment). The larger the difference between both errors, the more overfit the model is since it shows that the given knowledge in the form of training data was not properly generalized for the rest of the feature space. Fig. 2.25 shows the general relation between these two errors and the model's complexity. The optimum model complexity results in the smaller difference between training and test error as well as minimum values for each.

![Figure 2.25: General relation between training/test error and model complexity. Reprinted from [46]](image2)

In order to prevent a model from overfitting, several regularization methods were developed. For ANNs, these include early stopping, Lasso, Ridge and Dropout regularization.

As explained in 2.4.2, an ANNs weights are updated in order to minimize the defined loss function for the given training points, thus by nature, as iterations go on, the model is adjusted in order to best fit the training data. As a consequence, the training of the model for too many iterations might cause overfit. One regularization technique usually employed is stopping the training once the validation error begins to increase. Fig. 2.26 shows the general relation between training/validation error and the number of training iterations. This figure might seem very similar to Fig. 2.25 and it is easy to assume that
complexity and number of training iterations are directly related which to some extent is safe to conclude that they are, however it is not completely true that in each iteration the network’s complexity increases. Early stopping tackles the fact that the model becomes ever more adjusted to the training data as iterations go on and this is true regardless of whether its complexity increases or not.

![Figure 2.26: General relation between training/validation error and training iterations (epochs). Reprinted from [47]](image)

Large network weights result in small variations in the input causing large variations in the output, as such, these describe less smooth relations similarly to the overfitted model in Fig. 2.24. Lasso and Tikhonov Regularization are two techniques which, by adding terms to the loss function, limit the magnitude of the network’s weights in order to reduce overfit [48]. Lasso (or $L_1$) regularization adds to the loss function a factor which penalizes in accordance to the absolute value of the network’s weights, resulting in a loss function with a shape

$$L' = L + \lambda \sum_{i=0}^{n} |w_i|,$$  \hspace{1cm} (2.25)

where $L$ represents the previous loss function, $L'$ the new loss function, $n$ the total number of weights of the network, $w_i$ a given weight of the network and $\lambda$ an adjustable parameter. Tikhonov (or $L_2$) adds a penalizing factor in function of the square of the network’s weights through

$$L' = L + \lambda \sum_{i=0}^{n} w_i^2,$$  \hspace{1cm} (2.26)

By penalizing the square of the weight’s values, higher weight values are more heavily penalized by the $L_2$ regularization than by $L_1$, as such it is said that $L_2$ regularization promotes small all around weights. However, $L_2$ regularization penalizes with less severity small weights so it is not usual for weights to be driven to zero, whereas $L_1$ regularization through a heavier penalization of smaller weights, leads to a higher number of null weights, useful for eliminating any possible useless feature in the input vector.

Dropout is becoming one of the most commonly used regularization techniques used in ANNs, because it addresses two major problems in NNs: it prevents overfitting and provides a way of approximately combining different ANN architectures. Dropout is a method that approximates training a large number of neural networks with different architectures in parallel.

During the training of the network, in a given iteration, each neuron is dropped with a probability $1 - p$ where $p$ denotes the neuron’s specific independent probability of being kept (usually the same
probability is applied to all nodes or to all nodes in a specific layer). Dropping a neuron consists in eliminating it temporarily from the network by also eliminating its incoming and outgoing connections. Once the selected nodes are eliminated, the temporary architecture of the network is very different from its original as shown in Fig. 2.27.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more, or less, responsibility for the inputs. This method targets cases where neurons may change in a way that they fix up the mistakes of the other neurons. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data. Both these factors make the model more robust[49].

2.4.5 Neural Network’s Hyperparameters

The main disadvantage of the ANN is the great number of hyperparameters which should be adjusted. In this section, an overview of each hyperparameter’s influence on the model is detailed, and strategies for finding the most appropriate value are briefly described.

Hidden Layers and Number of Neurons

As aforementioned and shown in Fig. 2.21, an ANNs architecture is composed by an input and output layer as well as one or more hidden layers. The number of hidden layers greatly influences the complexity of the modeled input/output relation. A larger number of layers describes a more complex relation and tends to lead to better results. However, this increasing complexity might lead to overfitting as mentioned in section 2.4.4 and increases the time and computational power required during the training stages. In contrast, a small number of layers might not be capable of accurately modulating the relation between the data. Unfortunately there is no formula in choosing the number of layers in the network and current approaches generally resort to either brute force (preforming a search in the depth space) or revolve around intuition and educated adjustments. The used approach consists in gradually increasing the number of layers until the results show no further sign of improvement.

Along with the number of layers, one must also consider the number of neurons in each of them.
For the input layer, the number of neurons is equal to the number of features considered. Similarly, the number of neurons in the output layer is the same as the expected number of outputs, in this work’s case, an \(x\) and \(y\) coordinate pair for each device considered. However, the number of neurons in each of the hidden layers is one other hyperparameter which should be tuned. Again, there is no perfect formula to pick the number of neurons in each layer although there are some heuristics such as the ones developed in [50], [51] and [52]. Ultimately, the optimal solution varies from problem to problem.

One of the simplest methods is through trial and error and it works by training the network with different combinations of number of neurons for the hidden layers and save the one that had the best results. The problem with this method is that it is really time consuming since training an ANN usually requires a long time. This work is done on top of the work done in [8], as such, the initial model was the one used and expanded for the current work. Some pruning techniques can be used after in order to eliminate nodes which, if removed from the network, would not noticeably affect network performance. One pruning technique consists in removing one of the nodes in an edge with weight very close to 0.

**Batch Size**

Consider an ANN which has as loss function the MSE from 2.4. The parameter \(n\) represents the number of examples considered in the calculation of the loss. During the training phase, the network’s parameters are adjusted in order to minimize this loss function. Naturally, one would aim for the minimum MSE error across all of the training examples in order to calculate the loss’s gradient \(\nabla L(W)\) considering all training examples \(n = N\). This is denominated Batch Gradient Descent as a single update of the model’s parameters requires the calculation of the model’s gradients for the whole training set, considering it a single batch of data. Batch Gradient Descent can be very slow and is unfeasible for datasets that do not fit in memory but is certain to converge to the global minimum for convex error surfaces[42].

Stochastic Gradient Descent, in contrast, computes the losses gradients for a single example at a time (i.e. a batch of size 1). This approach results in frequent, high variance updates which results in erratic and noisy parameter updates. However, this erratic behavior might be somehow beneficial since it enables the optimizer to escape local minima while usually performing much faster than Batch Gradient Descent.

Mini-batch gradient descent performs a parameter update for a certain number of training examples \(0 < n < N\). This approach reduces the variance of the parameter updates, which can lead to more stable convergence when compared to Stochastic Gradient Descent and faster than Batch Gradient Descent.

As such, the process of choosing the size of the data batch considered in each update can influence the results and the overall performance. Again, optimal batch size varies from problem to problem and usually trial and error through educated guesses is used to determine the hyperparameter’s value.
Activation Function

As explained in section 2.4.1, each neuron’s output is determined through equation 2.7 where $\phi(z)$ denotes the neuron’s activation function which represents one of the hyperparameters of the network. A neuron’s activation function is what ultimately defines its behavior and capabilities. As such, non-linear activation functions are usually used, since usually the relation between the data is non-linear. Each of the activation functions researched have their purpose, their strengths and weaknesses, but remember that for the backpropagation algorithm to work, each neuron’s activation function should be differentiable.

In regression problems the model’s output is usually a real value, either negative or positive. Therefore, the output layer nodes’ activation function is usually a linear function since its codomain consists in the entire real set. Fig. 2.28 shows the function’s plot. Concerning the backpropagation algorithm, it’s a differentiable function with a simple constant derivative. This constant derivative term constitutes one of the main issues with the linear function (additionally to the fact that it cannot modulate non-linear relations), since no matter the error, it will always contribute with a constant factor to the parameters update.

![Linear Activation Function](image)

Figure 2.28: Linear activation function, used in the output layer in regression problems.

The sigmoid (or logistic activation function) maps the input values to a output range $[0, 1]$, which essentially encodes their probability of belonging to a certain class making it commonly used in multi-class classification problems, it also prevents cases where its output reaches invalid high values which can happen to unbounded cases such as the linear function. Another attractive aspect of the sigmoid for classification problems is how it has a very steep curve in the vicinities of $z = 0$ as seen in Fig. 2.17, this makes it very sensitive to small variations in this area resulting in a clearer separation of the data. Its gradient is smooth as well which makes it well-behaved during the backpropagation algorithm. Notice, however, how in Fig. 2.17 the function seems to have two asymptotes in the far sides of the input axis (hence its bounded values), in these almost horizontal regions the function is not at all sensitive to changes in the input, i.e. Its gradient is very small, approaching 0, which results in a lack of updates for
the model’s parameters which is the same as it stopping its learning.

The Rectified Linear Unit (ReLU) activation function is computed through \( f(z) = \max(0, z) \) and its plot is shown in Fig. 2.29, the function’s output is 0 if its input is less than or equal to 0, otherwise, its output is equal to its input. Although it might not seem like it, ReLU is nonlinear in nature and is thus able to encode non-linear relations. Its simple formula makes it less computationally heavy than the sigmoid function. Notice however, that the function does not have an upper bound making it susceptible to exploding outputs. However it does address the small vanishing gradients from the sigmoid function, even though it brings up another issue. For the region \( z < 0 \), the neuron’s gradient is 0 and thus it will stop responding to changes in the error. This is called the dying ReLU problem. Another problem arises when its derivative is taken into account since it is not continuous in \( z = 0 \), these discontinuations make the optimization process more erratic.

![Figure 2.29: Plot of the Rectified Linear Unit function](image)

In order to address this dying ReLU problem, some variations on the activation function have been introduced such as the Leaky ReLU [53]. However, the most notable is the Exponential Linear Unit (ELU), proposed in [54], whose plot is seen in Fig. 2.30 and its output is determined through

\[
  f(z) = \begin{cases} 
  \alpha(e^z - 1) & \text{if } z \leq 0 \\ 
  z & \text{if } z > 0,
  \end{cases}
\]  

(2.27)

where \( \alpha \) is a parameter in itself. This variation solves the dying ReLU issue and has a smooth derivative given by

\[
  f(z) = \begin{cases} 
  \alpha e^z & \text{if } z < 0 \\ 
  1 & \text{if } z > 0.
  \end{cases}
\]

(2.28)

Notice that as \( z \) approaches 0, the ELU’s derivative approaches 1.
2.4.6 Feature Engineering

One of the main challenges faced when developing an ML system, is the selection of the optimal features and its preprocessing.

The selected features should be such that for the same combination of features, the same output is expected. Consider the problem of predicting California Housing Prices from [55]. Some of the dataset features include, for each house in the dataset (which constitute each example), latitude, longitude and total number of rooms. If a model was trained using only latitude and total number of rooms as features, the same combination of values for the pair of features would not have the same expected output (assuming all examples were used) since intuitively, one knows that the longitude of the house’s location will likely influence its price, i.e. the loss function (which would most certainly be calculated through comparison between the prediction made and the target value) varies along the longitude feature axis. However, if only these two features were available, the model would be feasible if all houses in the dataset had the same longitude value. That is, in order to ensure a capable model, by reducing the number of features, one should reduce the scope of the solution accordingly. This can be interpreted as forcibly reducing the variability of one of the problem’s features through the reduction of the solution’s scope. This example might seem trivial at first but it serves as an example for these two factors which should be taken into account, that the same combination of features should have the same expected output and that in order to reduce the number of necessary features, the scope of the solution should be reduced accordingly, inversely, in order to increase the scope of the solution, it might be necessary to add features.

Consider again the California Housing Prices prediction, Fig. 2.31 shows a grid map of the considered location, the latitude feature enables the modulation of the housing prices along the vertical axis of the map, while the longitude feature does the same for the horizontal axis. However, a synthetic feature \( \text{latitude} \times \text{longitude} \) would enable the division of the space in a grid like manner, distinguishing exam-
amples by the combination of these two features. This is called feature crossing and can provide predictive abilities beyond what those features can provide individually. Polynomial features of degree $n$ consist in crossing features to produce crossings of degree $n$. For example, if the model’s input features were $[a, b, c]$ then its polynomial features of degree $n = 2$ would be $[a, ab, ac, a^2, b, bc, b^2, c, c^2]$. Notice that from the added features, only $[ab, ac, bc]$ were a product of actually crossing features.

![Map of California](image)

Figure 2.31: Map of California, vertical lines indicate constant longitude, horizontal lines indicate constant latitude

The scale and distribution of the data drawn from the domain may be different for each variable, for example input variables may have different units (e.g. kilometers, hours, milligrams) likely meaning that the variables have different scales. These variations in scale may cause the network to learn large weight values, which, as aforementioned, may lead to high sensitivity to input variations which may lead to overfitting. Large scale values as targets may lead to large gradient values during backpropagation which may result in an unstable optimization process. In order to avoid both these issues, both input features and targets should have the same scale and be relatively small (around the interval $[-1, 1]$) which can be done through scaling methods such as min-max normalization and standardization.

Min-max normalization scales the range of a given unscaled feature $x$ to the range $[0, 1]$. A scaled version of the feature $x'$ can be determined through

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}. \quad (2.29)$$

Standardization models a feature’s data as a normal distribution with centered in 0 and standard deviation unitary. It does so through

$$x' = \frac{x - \mu}{\sigma} \quad (2.30)$$

where $\mu$ is the mean of the unscaled feature data and $\sigma$ its standard deviation.
Categorical features are those that take discrete values, such as *year of birth* or *color*. As mentioned in 2.3.1, the feature *color* has to be converted into a numerical representation. One intuitive approach is to model the *color* categories similarly to *year of birth*, assign to each possible color an integer. This is called Ordinal Encoding [56] and even though it is very simple, it naturally implies an order between the possible categories [57], which in this case there is not. One Hot Encoding is the most widely used coding scheme nowadays. It expands the feature into a vector of length equal to the number of possible categories, each of which is assigned to a specific position in the vector. As such, after the conversion the vector is 0 in all positions except for the assigned position for the categorical feature value which is 1. In reality this "vector" is simply the result of creating multiple new features (one for each vector element). This might highly increase the number of features considered if the number of possible categories is very large. Even though the ordinal coding does not expand the number of features, One Hot Encoding has been shown to produce better results [56]. As such, naturally numeric categorical features such as the *year of birth* should also be converted to One Hot Encoding.

In this section, an ANNs learning process was deconstructed and the most influencing parameters were analyzed and its impact explained.

### 2.5 Conclusions

In this chapter, the current approaches towards automatic layout generation were categorized depending on the number of considered pre-existent guidelines. Approaches which consider legacy layouts do not produce novel predictions, instead, pre-existing knowledge is reused and nothing is gained knowledge wise. On the other hand, approaches which rely on the optimization of certain quality metrics are generally time-consuming and computationally heavy.

A novel approach, introduced in [8], implemented ANNs in order to make the process of template selection and migration in practice instantaneous. However, similarly to Knowledge Mining and Layout Migration (in between of which the approach might be placed), this approach does not promote innovation or knowledge generalization and might even punish the behavior. This makes the model behave similarly to classification models, assigning to each example a class in the form of positioning guidelines (template). Additionally, the process of template pre-production might present a challenge in itself, consequently if no template was produced for a given topology, the model would just use another topology's template as target, producing an invalid placement. However this approach reveals the capabilities of the ANNs.

In this work, the use of ANNs is further explored, in an attempt to use its fast, generalizing predictive prowess to produce optimized, topological constrain aware placements inspired by the Placement Optimization approach. Thus attempting to minimize the time cost usually associated with the approach.

Additionally, the most common ML methodologies were briefly presented in order to demonstrate, by comparison, the advantages of the chosen Connectionist approach when taking into account the context and objectives of this work's problem. More importantly, ML systems were categorized in function of the guidelines imposed on the system's behavior. However, more than noticing the differences between
the approaches, one should look into its similarities in order to better understand the underlying nature of ML. All of the defined categories learned by iteratively minimizing a cost function (in the context of supervised learning, maximizing a reward is analogous to minimizing a punishment which could be the inverse of the reward function) which codifies the true intended behavior of the system. In supervised models, the desired behavior is for the model to make predictions as close as possible to previously deemed correct guidelines, this forces the model to understand and replicate, as well as possible, an existent relation (more than one relation might produce the same results, e.g., overfit models produce the same predictions but encode a much more complex relation) between the input data and the given output data. In Unsupervised Learning, the metric which is iteratively minimized is the overall similarity between each point and its assigned cluster center, as such, the provided definition of similarity will influence the models behavior, i.e., the formation of clusters. For Reinforcement Learning agents, its objective is to build a policy which maximizes its expected reward (similar to minimizing an expected punishment), therefore, its overall behavior, which is dictated by its policy, is codified in the developed reward function.

This work takes a somewhat Supervised approach to the problem since there is a target for the network, to predict placements which fulfill all of the circuit’s topological constraints. However, this target is somewhat unclear as there is no single correct answer, instead, a multitude of predictions are allowed, as long as they fulfill the topological constraints of the circuit. This process could be converted to a classic Supervised Learning approach if, for every example in the dataset, all the placements which fulfill the circuit’s topological constraints were generated and the error associated to a specific example’s prediction would be given by the minimum, for example, MSE error across all the generated target placements. In a way, the developed loss function describes abstractly all these possible placements and directs the model to the closest one.

Moreover, the ANNs learning process was deconstructed and analyzed. The process of weight update which constitutes the actual adaptability or learning of the network is based on the backpropagation algorithm which relies on the differentiability of both each neuron’s activation function and the model’s loss function. However, what dictates each weights’ update is ultimately the chosen optimizer. By analyzing the different optimizers, its several components and which problems each address, one becomes aware of the challenges of global convergence. Therefore, it is part of the developer’s job to not only chose the appropriate optimizer but also to design a well-behaved loss function which attempts to minimize any possible challenges during the convergence phase such as designing a continuous, smooth loss function while also avoiding any possible plateaus. In practice it is not possible to completely have control over this due to the enormous scale of the parameter space, however these should be nevertheless taken into account and obvious scenarios should be avoided.

Finally, the importance of the chosen representation for the problem’s features was analyzed. As previously mentioned, this work expands the scope of the solution developed in [8] to produce a model capable of developing placements for a variety of topologies. As such, the problem’s features should be expanded as well in order to properly distinguish between topologies. However, the added information should be optimally added, adding only the necessary information about each problem’s example and
not any more since this might gravely affect the learning process.
Chapter 3

Implementation

The work here done has as basis the approach suggested in [8] which makes use of an ANN model trained to approximate its predictions to previously generated templates with the objective of generalizing the knowledge put into these templates. The approach presented some problems such as being limited to a single circuit topology and that good, innovating predictions were being punished due to the used evaluation (loss) function (the MSE). In this chapter the envisioned improvements to this approach are detailed: The development of the input features in order to expand the solution's scope and increase generalization and the introduction of a new loss function which evaluates the prediction made through the fulfillment of the circuit's topological constraints.

The original approach serves as a reference to evaluate the effectiveness of the improved system. A third model is tested using the original loss function as a model initializer in an attempt to reduce training time and develop a more stable solution.

3.1 Problem Definition and Dataset Description

As explained in section 2.2.4, this work is built on top of the model developed in [8]. The model’s dataset consisted in several sizing examples of the same circuit topology. Each sizing scenario was represented by a 5-sizing input vector similar to each row in table 2.1. For each sizing scenario 12 templates were designed out of which one was used as target placement (the one with least area). The model’s predictions were evaluated by comparing them to these selected targets, if the prediction made was very similar, the error would be low.

While in [8] the scope of the developed solution consisted only of designing different layouts for a constant circuit topology (the SSA from [29]), the solution here developed intends to expand the scope of the solution, making it able to design placements for multiple circuit topologies, as such, a new circuit is introduced in the training data as well as a testing circuit (not included in the training data) in order to evaluate the model’s performance to never seen before topologies. The purpose of introducing this new circuit in the training data is to not only introduce new patterns of topological constraints but as well to test the model’s capability of dealing with topologies with different number of devices.
The newly introduced training circuit, the Cascode Free Single Stage Amplifier (CFSSA), was introduced in [58] and its schematic can be seen in Fig. 3.1.

![Figure 3.1: Cascode Free Single Stage Amplifier: schematic with current paths highlighted. Adapted from [58]](image)

The introduction of these new circuits brings up some issues. The number of devices is different, which would make the input vector’s length different for each case. Also, even if the number of devices is not different, if the circuit topology is different, how would the network distinguish between examples? It is not far fetched to think of another circuit with 12 devices with similar sizing characteristics but very different topology, that is, very different symmetry or current flow constraints (which were explained in section 2.1.1), if so, the templates used for the SSA would most likely be invalid for this new topology. Using only the sizing of the devices to identify the circuit topology is not enough as the same combination of features would have different expected outputs.

### 3.2 Constraint Descriptive Vector

A new input vector is here proposed in order to address both the issue of input examples collision (two different topologies with a different expected prediction having the same input vector) and promote more diverse and tailor made solutions for each example. The objective is to build a model capable of interpreting the topological constraints of the circuit given as input and producing an optimal valid (which fulfills all constraints it’s subject to) placement.

Two topological constraints are considered in the design of this solution, current flow (or monotonic current path) and symmetry. In order to enable the ANN to interpret these constraints, these should be somehow encoded in the input vector, as such, the new input vector now includes not only each devices’ sizing characteristics but also a description of each devices’ symmetry and current flow constraints.


3.2.1 Current Flow Encoding

Current flow constraints describe the path the current takes (through which devices it passes in order) from the power source to the ground. The devices should be placed in such an order that the current flows monotonically in one direction as shown in Fig. 2.6.

In the case of the SSA in Fig. 2.13, it is possible to distinguish six different current paths. As aforementioned, these paths can be described by the list of devices through which the current passes in order, that is, as if one were to trace a path from the power source to the ground and registering through which devices it would pass. For example, one valid description of a current path would be $[PM0, NM10]$ as well as $[PM1, NM4, NM9]$. As such, for each of the 12 devices that make up the circuit, it is possible to save its position in each of the six different current paths, for example, the device $PM0$ appears only in one current flow, the first one (numbering them from left to right and top to bottom), in the first position. Therefore, its current flow constraints can be represented by the vector of length 6, $[1, 0, 0, 0, 0, 0]$. Similarly, the current flow constraints of the device $NM8$ can be represented by the vector $[0, 0, 3, 2, 0, 0]$, meaning the device is a member of two different current paths in which it appears in position 3 and 2 respectively.

As it was explained in section 2.4.6, it has been suggested that the use of one-hot encoding for categorical features has better performance than the ordinal coding aforementioned[56]. In this case, the categorical integer feature can be described, for a specific device, as the device’s position in each of the current paths. Since in the example of the SSA the maximum number of devices which make up a current path is three, then each path should be encoded by a one-hot vector with three categories. Using the example of the device $NM8$ once again, each of the values in the descriptive vector previously mentioned will be extended as a three categories one-hot vector as $[[0, 0, 0], [0, 0, 1], [0, 0, 0], [0, 1, 0], [0, 0, 0], [0, 0, 0]]$, or as a $6 \times 3 = 18$ long vector resulting from the concatenation of all of the one-hot vectors $[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. The full current flow constraints descriptive vector for each of the 12 devices in the SSA topology is shown in table 3.1.

<table>
<thead>
<tr>
<th>Current Path #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM0 (0)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM1 (1)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM2 (2)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM3 (3)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NM4 (4)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NM5 (5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NM6 (6)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM7 (7)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM8 (8)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NM9 (9)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NM10 (10)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM11 (11)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Using the same method, table 3.2 shows the full current flow constraints descriptive vector for each of the devices that make up the CFSSA.
Table 3.2: CFSSA Circuit Device’s Current Flow Description

<table>
<thead>
<tr>
<th>Current Path #</th>
<th>PM0 (0)</th>
<th>PM1 (1)</th>
<th>PM2 (2)</th>
<th>PM3 (3)</th>
<th>NM0 (4)</th>
<th>NM1 (5)</th>
<th>PM4 (6)</th>
<th>NM6 (7)</th>
<th>PM5 (8)</th>
<th>NM7 (9)</th>
<th>PM6 (10)</th>
<th>NM8 (11)</th>
<th>NM9 (12)</th>
<th>NM4 (13)</th>
<th>NM5 (14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 0</td>
<td>0 0 1</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>7</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

The length of this one-hot vector is thus given by

\[ L_{cf} = n_{cf} \times \max_{i}(devs_{cf}(i)), \]  

where \( n_{cf} \) denotes the number of current paths in the circuit and \( devs_{cf}(i) \) represents the number of devices which make up each of the \( n_{cf} \) current paths. So, for each of the two circuit topologies that make up the training dataset, the length of the current flow constraint description vector for each device is given by

\[ L_{cf}(SSA) = 6 \times 3 = 18; \]  
\[ L_{cf}(CFSSA) = 8 \times 3 = 24; \]

3.2.2 Symmetry

A symmetry constraint corresponds to a set of devices and/or device pairs which are symmetrically placed along a symmetry axis as represented in Fig. 2.6.

In the case of the SSA, each of the devices has a symmetric pair, as is the case of the pair of devices \([PM0, PM3]\) and \([NM6, NM7]\). Once again, these constraints can be encoded through the use of a one-hot vector for each of the devices which indicates to which of the circuit’s \( N \) devices it should be symmetric to. As such, the result is, for each device, an \( N = 12 \) long vector, e.g. for device \( PM0 \): \([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]\), or for device \( NM6 \): \([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]\). The full symmetry constraints descriptive vector for each of the devices of the SSA are shown in table 3.3, for the CFSSA in table 3.4.

Notice from table 3.4 that this representation can also encode self-symmetric devices (devices which should be centered along the symmetry axis) as is the case of for example device \( PM4 \) from the CFSSA, these devices’ constraints can be encoded through a 1 in their own index. Devices which are not
Table 3.3: SSA Circuit Device’s Symmetry Description

<table>
<thead>
<tr>
<th></th>
<th>PM0</th>
<th>PM1</th>
<th>PM2</th>
<th>PM3</th>
<th>NM4</th>
<th>NM5</th>
<th>NM6</th>
<th>NM7</th>
<th>NM8</th>
<th>NM9</th>
<th>NM10</th>
<th>NM11</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NM9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NM10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NM11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.4: CFSSA Circuit Device’s Symmetry Description

<table>
<thead>
<tr>
<th></th>
<th>PM0</th>
<th>PM1</th>
<th>PM2</th>
<th>PM3</th>
<th>NM0</th>
<th>NM1</th>
<th>PM4</th>
<th>NM6</th>
<th>PM5</th>
<th>NM7</th>
<th>PM6</th>
<th>NM8</th>
<th>NM9</th>
<th>NM4</th>
<th>NM5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NM5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

constrained whatsoever (i.e. autonomous cells) would be encoded by having no 1 along the vector.

The symmetry constraint descriptive vector’s length for each device is given by

\[ L_{sym} = N \tag{3.4} \]

where \( N \) represents the number of composing devices for any given circuit topology. So for the two circuit topologies used

\[ L_{sym}(SSA) = 12 \tag{3.5} \]
\[ L_{sym}(CFSSA) = 15 \tag{3.6} \]

3.2.3 Sizing

Since now the introduction of both the current flow and symmetry constraints serve as distinguishing factors between two different circuit topologies, the only sizing characteristics of each device that directly influence the placement task are its encapsulation’s width and height. As such, these are the only two
sizing characteristics out of the five shown in Table 2.1 which are used in the constraint description vector. So, for each device, a two values long vector is kept \([\text{width}, \text{height}]\).

As explained in section 2.4.6, quantifiable features should be scaled in order to avoid overfitting. In this case, the fact that the features refer to devices’ sizes on nanometric integration technologies, it is important that these values are scaled in order to avoid any vanishing gradients during the backpropagation phase. Therefore, the sizing characteristics of each device are normalized using equation 2.30 across all examples as shown in Table 3.5.

3.2.4 Zero Padding for Multiple Circuit Design

By aggregating these three components of a device’s description into one vector the result is a vector of length

\[
L_{dev} = n_{cf} \times \max_{i}(\text{devs}_{cf}(i)) + N + 2.
\] (3.7)

In the case of the SSA, the vector’s length is given by \(6 \times 3 + 12 + 2 = 32\). An example of a full, scaled, device descriptive vector for device \(PM0\) in a sizing scenario of the SSA is given in Fig. 3.2

![Example of the scaled, descriptive vector of the device PM0 in the single stage amplifier circuit.](image)

For devices of the CFSSA circuit topologies, through equations 3.1 and 3.4, their descriptive vector’s length is given by

\[
L_{dev}(CFSSA) = 8 \times 3 + 15 + 2 = 41
\] (3.8)

Finally, the complete input vector of a sizing case for the SSA and the CFSSA circuits are the result of the concatenation of the descriptive vectors of all composing \(N\) devices as shown in Fig. 3.3.
Considering the concatenation of $N$ constraint descriptive vectors, the length of a circuit’s input vector is given by

$$L = N[Max(dev_{cf}) + n_{cf} + N + 2],$$ \hspace{1cm} (3.9)

So, replacing by the values of each circuit topology, its specific input vector length is given by

$$L(SSA) = 12[6 \times 3 + 12 + 2] = 384$$ \hspace{1cm} (3.10)

$$L(CFSSA) = 15[8 \times 3 + 15 + 2] = 615$$ \hspace{1cm} (3.11)

Recall from section 2.4.5 that an ANN’s number of nodes in the input layer is usually the number of features considered, that is, the size of the input vector. Notice also that the input vector’s length depends on the input circuit’s topology and since an ANN’s structure is not changeable depending on the input, a general input vector length must be defined. This is necessary because it is the objective of this work to build a network capable of formulating placements for different circuit topologies, otherwise, three different networks with different structures would be more effective for each individual circuit. Naturally, smaller input vectors will be padded with zeros to match the largest input vector’s length.

Consider the current flow constraint descriptive vector for each device explained in section 3.2.1, it consists in the concatenation of $n_{cf}$ one-hot vectors, one for each of the different current paths in the considered topology. The topology with the most different number of current paths is the CFSSA with 8. As such, other topologies add the necessary amount of empty current paths (i.e. current paths of which no device is part of), represented by zeroed one-hot vectors, to match the 8 of the CFSSA. Each of these $max(n_{cf}) = 8$ one-hot vectors can be interpreted as the device’s position in a given current path, as such, the length of each one-hot vector should accommodate for the longest current path in all of the considered topologies. From the analysis of tables 3.1, 3.2 it is possible to conclude that the maximum number of devices in any given current path from any of the topologies is 3. Thus, in general, when considering $T$ different circuit topologies, the minimum length of the current flow constraint descriptive vector is given by

$$L_{cf} = \max_{t_i} n_{cf}(t_i) \times \max_{cf_{ij}} n_{devs}(cf_{ij}),$$ \hspace{1cm} (3.12)

where $t_i, i \in [1, T]$ represents a specific topology, and $cf_{ij}, i \in [1, T], j \in [1, n_{cf}(t_i)]$ represents a specific current path from any one of the considered topologies.

The length of the padded vector considering only the SSA and CFSSA topologies is $8 \times 3 = 24$. 
Therefore, the current flow vectors of the SSA devices are padded with two empty current paths as shown in table 3.6.

### Table 3.6: SSA Circuit Device’s Padded Current Flow Constraints Description Vector

<table>
<thead>
<tr>
<th>Current Path #</th>
<th>PM0 (0)</th>
<th>PM1 (1)</th>
<th>PM2 (2)</th>
<th>PM3 (3)</th>
<th>NM4 (4)</th>
<th>NM5 (5)</th>
<th>NM6 (6)</th>
<th>NM7 (7)</th>
<th>NM8 (8)</th>
<th>NM9 (9)</th>
<th>NM10 (10)</th>
<th>NM11 (11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For the symmetry constraint descriptive vector, it is only necessary to add autonomous cells (devices with no symmetry constraints) in order to match the maximum number of devices in all the considered topologies (15 from the CFSSA). In sum the padding process is similar to introducing in the circuit’s schematic new devices which do not have either width or height nor any topological constraints and two new current paths which no device is part of. The new padded descriptive vector of the device PM0 in the SSA circuit can be seen in Fig. 3.4.

![Figure 3.4: Example of the, not yet scaled, padded descriptive vector of the device PM0 in the single stage amplifier circuit.](image)

Finally, the resulting padded input vector’s length is given by

$$length = \max_{t_i} (N(t_i)) \max_{t_i}(n_{cf}(t_i)) \max_{t_i}(n_{devs}(c_{f ij})) + \max_{t_i}(N(t_i)) + 2,$$

(3.13)

where $N(t_i)$ denotes the number of devices in topology $i$. The final shape of a padded input vector can be visualized in Fig. 3.5.
3.2.5 Device Scrambling

In order to avoid the model to become overfit to the order of the constraints in the input vector, for every example, the order of the devices is scrambled. Considering 15 devices, the number of possible permutations of a single example is $15! \sim 1.31 \times 10^{12}$. Fig. 3.6 shows an example of scrambling the order of 15 devices.

![Device Scramble Diagram](image)

To apply the effects of changing the order of devices at every level, it is necessary to consider its effects on the current flow and symmetry constraints descriptive vectors of each device. Consider the SSA's current flow table 3.6, changing the order of devices corresponds to simply changing the order of the rows since the current path's order remains the same, so in reality each device's current flow constraints vector remains unaltered. For the symmetry constraints, consider the CFSSA's symmetry table 3.4. Changing the order of the rows is not enough and thus the order of the columns should be changed as well in succession. This corresponds to the transformation applied to each device's symmetry constraints vector. At the input vector level, it is only necessary to change the order through which one concatenates the devices' descriptive vectors as illustrated in Fig. 3.7.

3.3 Topological Loss Function

In order to focus the model on learning how to interpret a circuit's topological constraints, it is necessary to change its loss function since, as aforementioned, the one used in [8] (represented in equation 2.4) evaluates the quality of the prediction made by comparing its similarities to the selected legacy layout.
Figure 3.7: Device scramble at input vector level.

This results in the focus being on generating layouts similar to previously designed layouts, which in turn discourages the production of novel designs and creates a model unprepared for new circuit topologies. The use of this loss function also demands the production of several previously tested legacy designs which may prove a challenge in itself. To overcome these issues, a new loss function is proposed, one which evaluates the quality of a prediction made by evaluating whether or not the circuit's placement constraints are met. Therefore, the general shape of the proposed topological loss function is given by

\[
    \text{loss} = w_w W_A + w_s S + w_{cf} C_F + w_o O,
\]

where \( W_A \) denotes the wasted area of the layout, \( S \) represents the deviation from optimal symmetry, \( C_F \) represents the error associated to non-monotonic current flow directions, \( O \) denotes the summed overlap area between all devices and \( w_w, w_s, w_{cf}, w_o \) represent the weights associated to the wasted area, symmetry, current flow and overlap errors respectively.

### 3.3.1 Wasted Area

It is a usual requirement for analog IC technology to have minimal area in order to reduce costs, as such, the first factor taken into account in the new loss function is the layout's compactness.

The area of the layout is defined using the area of the smallest possible rectangle which can encapsulate all of the circuit's devices in full, i.e. the square defined by the minimum and maximum \( x \) and \( y \) coordinates from all of the devices' edges. However, since the examples in the dataset, for the same circuit, are distinguished only by different devices' sizes, it would be unfair to evaluate only the layout's area since examples with bigger devices would inevitably rank lower than those with smaller devices. This difference could highly influence the results if there was a clear disparity in average device's area between the training and validation sets.

To better quantify how compact a layout is, another metric is used, its wasted area defined by

\[
    W_A = \frac{A_L}{\sum_{i=1}^{N} A(dev_i)},
\]

where \( A_L \) denotes the predicted layout's circumventing area, \( N \) the number of devices in the circuit and \( A(dev_i) \) the area of the device \( i \).
3.3.2 Summed Symmetry Axis’ Deviation

The symmetry axis of a pair of symmetric devices can be determined by calculating the centroid of the pair by first calculating the centroid of each individual device. This is defined by

\[
\bar{x} = \frac{1}{2}(x_i + \frac{w_i}{2} + x_j + \frac{w_j}{2}),
\]

where \(\bar{x}\) denotes the \(x\) coordinate of the pair’s symmetry axis, \(x_{i/j}\) the \(x\) coordinate of the bottom-left corner of device \(i/j\) and \(w_{i/j}\) the width of device \(i/j\). For self-symmetric devices \(i = j\) and the equation holds true.

All of the devices’ symmetry axis (if any exists) should have the same \(x\) coordinate, as such, the summed deviation of each pair’s symmetry axis to any reference axis is given by

\[
\sigma_A = \sum_{k=1}^{M} (\bar{x}_k - \bar{x}_r)^2,
\]

where \(M\) denotes the total number of symmetry groups in the circuit, \(\bar{x}_k\) the \(x\) coordinate of the \(k\) group’s symmetry axis and \(\bar{x}_r, r \in [1, M]\) represents a reference axis either chosen randomly from any of the \(M\) existent or set as a parameter.

For normalization purposes the layout is forced to be centered at \(x = 0\) and so, all symmetry axis should be located at \(x = 0\), therefore, taking as reference axis \(\bar{x}_r = 0\)

\[
\sigma_A = \sum_{k=1}^{M} \bar{x}_k^2.
\]

The value of \(\bar{x}_k\) is squared since it is easier to differentiate than the absolute value, thus facilitating the backpropagation of the error.

Additionally, symmetric devices should be located along the same horizontal line, that is, the \(y\) coordinate of both devices which constitute a symmetry group \(k\) should be the same. The difference in \(y\) coordinates between a pair of devices \(i, j\) which belong to symmetry group \(k\) is given by

\[
\Delta y_k^2 = (y_i - y_j)^2.
\]

Similarly to what’s done in 3.18, the difference between the \(y\) coordinates is squared to take into account the backpropagation phase. As such, the error associated to the deviation of the \(y\) coordinates between all the circuit’s symmetric pairs is given by

\[
\sigma_y = \sum_{k=1}^{M} \Delta y_k^2.
\]

Finally, the symmetry error is also normalized since a deviation of 5 nanometers can be significant if the layout’s width is 20 nanometers but not so much for cases where the width is around 1 micrometer. However, a layout’s width is directly dependent on the prediction made by the network, so using it as
a normalization parameter would cause erratic gradients and hinder the convergence process. As an alternative, the average width of all devices in the layout was used. The average width is also squared to match the symmetry axis’ deviation units. The error is also normalized in relation to the total number of devices in the circuit since circuits with higher number of devices would have more devices contributing to the total symmetry axis’ deviation resulting in a higher error. The final symmetry error is given by the expression

$$ S = \frac{\sigma_A + \sigma_y}{N} \times \bar{w}^2 $$

(3.21)

where $\bar{w}$ denotes the average width of all devices and $N$ the total number of devices in the circuit.

### 3.3.3 Current Flow Consistency Error

The direction through which the current flows, abstracting from the wiring between devices, can be determined by the sign of the difference in $y$ coordinates between two consecutive devices in a current path. More specifically, between the top $y$ coordinate of the first device and the bottom $y$ coordinate of the second device.

Take the example of a placement solution in Fig. 3.8 for the SSA topology. one of the circuit’s current paths is defined by [PM0, NM10] (current path number 0 in table 3.6) which corresponds to [dev0, dev10] in the nomenclature used in Fig. 3.8. By calculating the direction of the current through the following equation

$$ C_D = \text{sign}(y_i^+ - y_i^-) $$

(3.22)

where $y_i^+/-$ denotes the top/bottom $y$ coordinate of device $i$, it results in a current direction $C_D = 1$ (assuming the $y$ coordinate is measured along the vertical axis, being 0 at the bottom of the layout limits and maximum at the top of the layout limits), meaning that in order for the layout to fulfill the current flow constraints, all current paths should be directed from top to bottom (also denominated as positive direction). Another current path is defined as [dev1, dev4, dev9] (current path number 1 in table 3.6), it is possible to confirm through the inspection of Fig. 3.8 that for all consecutive pair of devices, the top $y$ coordinate of the first device is always greater than the bottom $y$ coordinate of the second device, therefore, the current flows in a consistent manner and the associated error is null. As such, it is possible to quantify a layout’s current flow consistency error through

$$ C_F = \sum_{i=1}^{n_{cf}} \sum_{j=1}^{d(CF_i)} -\min(0, D_r(y_{ij}^+ - y_{ik}^-)) $$

(3.23)

where $n_{cf}$ denotes the number of current paths in the circuit, $d(CF_i)$ the number of devices that make up current path number $i$, $y_{ij}^+/-$ the top/bottom $y$ coordinate of the $j$th device in the current path $i$ and $D_r$ a reference direction, either chosen randomly from the ones existing in the suggested placement or set as a parameter.

In this work the current was forced to flow in the positive direction in order to avoid erratic gradients,
Notice that since in this model all currents are forced to flow in the positive direction, errors occur only if the current’s direction is negative. If it is positive, then the error is null. This aspect is codified in equation (3.24) through $-\min(0, y_{ij}^+ - y_{ik}^-)$ where positive current directions are ignored and negative directions are considered. These then have to be made positive so they contribute to an increase in the error which will be minimized during the training phase. It is also important to note that the greater the value of $y_{ik}^-$ is in relation to $y_{ij}^+$, the greater the error even though once current flow is not consistent, it doesn’t make it any wronger if the devices are further apart, however this is done so that the gradient of the error pushes $y_{ik}^-$ to lower values.

Similarly to the case of the symmetry error, the current flow consistency error should be normalized in relation to both the number of devices $N$ in the circuit and the average device height $\bar{h}$. The final normalized current flow error is given by

$$C_F = \frac{\sum_{i=1}^{n_{cf}} \sum_{j=1}^{\text{devs}} \sum_{k=j+1} \min(0, y_{ij}^+ - y_{ik}^-)}{N \times \bar{h}}.$$  (3.25)

### 3.3.4 Overlap Between Devices

As explained in section 2.1.2, the use of absolute coordinates allows devices to occupy the same region in the 2D plane as exemplified in Fig. 3.9.

To address this problem, another metric is introduced, total overlap area, which quantifies the area of the intersection between all pairwise combination of devices. The horizontal overlap between any pair of devices $i$ and $j$ (overlap between the two lines which result from the projection of both devices in the horizontal axis) can be calculated through
Figure 3.9: Example of a placement for the single-stage amplifier with overlap area between devices in gray

\[ O_H(i, j) = \max(0, \min(x^+_i, x^+_j) - \max(x^-_i, x^-_j)), \]  
\[ O_V(i, j) = \max(0, \min(y^+_i, y^+_j) - \max(y^-_i, y^-_j)), \]

where \( x^+_i / x^-_i \) represents the right/left \( x \) coordinate of device \( i \). Vertical overlap is calculated through

where \( y^+_i / y^-_i \) represents the top/bottom \( y \) coordinate of device \( i \). Finally, combining both 3.26 and 3.27, the total overlap area for a layout with \( N \) devices is given by

\[ O = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} O_H(i, j) \times O_V(i, j)}{2}. \]

Notice that the overlap between two devices would be accounted twice with the numerator only, as such, the error is divided by 2 to account for that.

Finally, similarly to the wasted area metric, the overlap area should be normalized in relation to the devices’ total area, resulting in the final expression

\[ O = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} O_H(i, j) \times O_V(i, j)}{2 \sum_{i=1}^{N} A(dev_i)}, \]

where \( A(dev_i) \) represents device \( i \)'s area.

### 3.4 Model Structure and Training

In order to compare the new model’s performance with the one used in [8], a lot of the network’s hyperparameters were left untouched at the beginning to specifically evaluate the impact of the changes made in the input vector and loss function. As such the model’s structure and its hyperparameters are the following:
• The network’s initial 3 hidden layers from [8] were not capable of codifying the data relation, as such, 4 hidden layers were used with 2000, 750, 250 and 100 nodes respectively were used;

• The activation function used in all nodes (except in the output layer’s nodes which use a linear function) was kept as ELU due to the advantages described in section 2.4.5;

• Adam was left as the optimizer with the default values $\alpha = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.999$;

• Overfitting was addressed by using dropout in the hidden layers, using a dropping rate of 0.3;

• Initial random weights of the network layers were initialized by means of a normal distribution;

• Since the training dataset contains around 7000 to 14000 examples, the batch size was set to 500 which represents around 3.5% to 6.5% of the dataset which is lower than the 10% threshold to be considered a big batch size. The downside of such a low value is that the training times get higher, while requiring less epochs to achieve convergence.

• The topological loss function’s (equation 3.14) weights were empirically set to $w_a = 1, w_s = 400, w_{cf} = 1 \times 10^{-3}$ and $w_o = 700$. The weights were set in order to make all components similar in magnitude if the level of satisfaction is similar, e.g. for the placement in Fig. 3.9 one would expect a noticeably higher overlap error than symmetry error.

The model’s hyperparameters are summarized in table 3.7.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Layer</td>
<td>1 Layer (615 nodes)</td>
</tr>
<tr>
<td>Hidden Layers</td>
<td>4 Layers (2000, 750, 200, 100 nodes)</td>
</tr>
<tr>
<td>Output Layer</td>
<td>1 Layer (30 nodes)</td>
</tr>
<tr>
<td>Activation Function</td>
<td>ELU (Linear for the output layer)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$)</td>
</tr>
<tr>
<td>Regularizer</td>
<td>Dropout (drop rate=0.3)</td>
</tr>
<tr>
<td>Weights Initializer</td>
<td>Normal distribution</td>
</tr>
<tr>
<td>Loss Function</td>
<td>MSE, Topological Loss Function (equation 3.14)</td>
</tr>
<tr>
<td>Batch Size</td>
<td>500</td>
</tr>
<tr>
<td>Loss Function Weights</td>
<td>$w_a = 1, w_s = 400, w_{cf} = 1 \times 10^{-3}, w_o = 700$</td>
</tr>
</tbody>
</table>

3.4.1 Conclusions

In this chapter, a thorough description of the implemented model was given. In order to expand the scope of the solution, topology identifying features were added to input vector to distinguish between different topologies in the dataset. Additionally, these served, in conjunction with the introduced Topological Loss Function (TLF), to focus the model’s objective on producing any valid placement instead of following predefined guidelines. This chapter shows as well the application of the knowledge exposed in the previous
one. A topological constraint aware approach was used instead of the legacy layout based approaches in order to address the lack of innovation problem from [8]. The developed TLF attempts to explore the possibilities of the Supervised Learning by enabling the model to aim towards one between a variety of possible targets (any valid placement for a given sizing scenario). Finally, the ANNs functioning was taken into account when developing the Constraint Descriptive Vector (CDV) input vector through the use of appropriate categorical features encoding and scaling of the quantifiable ones, as well as during the development of the loss function, making sure it has no significant irregularities or plateaus.
Chapter 4

Results

In this chapter, different ANN models are tested and analyzed. They differ by the format of their input vector or by the loss function used during training, the network’s architecture is kept the same. The objective being to compare the impact of these isolated changes. All ANNs were implemented in Python language with TensorFlow [59] and Scikit-Learn [60].

4.1 MSE Models

Firstly, the MSE models are tested, i.e. models which use as loss function the MSE equation from 2.4. This section describes the structure of the networks and thoroughly analyzes this approach’s shortcomings, serving as a starting point to the development of this work’s suggested solution.

4.1.1 Structure and Training

Four different variants of the MSE model were tested, these vary only in the structure of their input vector, which all consist in variants of the 5-sizing input vector structure described in section 3.1 and illustrated in table 2.1:

- **Mean Squared Error - Non Polynomial Features (MSE-NP):** The input vector is left unaltered (i.e. structure wise, each feature is normalized using 2.30), that is, for each device its five sizing characteristics are stored and the input vector is formed similarly to what’s illustrated in Fig. 3.3, through the successive concatenation of each device’s characteristics. For all of the examples used the order of the devices was kept constant;

- **Mean Squared Error - Polynomial Features (MSE-P):** For this model the order of the devices was kept constant as well, but polynomial features of degree 2, using only the crossed features, were used through the process explained in section 2.4.6. The polynomial features were applied after the concatenation of each device’s sizing characteristics, as such, features between devices were crossed as well;
- **Mean Squared Error - Non Polynomial Features with Device Scrambling (MSE-NPS):** Similarly to MSE-NP, no polynomial features were used but in this model, the order of the devices is scrambled for every example in the dataset;

- **Mean Squared Error - Polynomial Features with Device Scrambling (MSE-PS):** Finally, an MSE model is tested using both polynomial features and scrambling the order of the devices in each example.

Table 4.1 summarizes the differences between the variants as well as the length of the resulting input vector.

<table>
<thead>
<tr>
<th></th>
<th>MSE-NP</th>
<th>MSE-P</th>
<th>MSE-NPS</th>
<th>MSE-PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial Features</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scrambled Device Order</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Vector Length</td>
<td>$12 \times 5 = 60$</td>
<td>1831</td>
<td>60</td>
<td>1831</td>
</tr>
</tbody>
</table>

For the training of this model, only SSA topology sizing cases were used since it wasn’t prepared for multiple topology prediction. As such, the target features consist in the $x, y$ coordinates of the left-bottom corner of each device in the selected SSA template out of the 12 that were generated. In total, 10,422 sizing examples were considered, these were divided in three sets, a training set (70%), a validation set (15%) and a test set (15%). The model’s hyperparameters are the ones defined in table 3.7. Finally, each model was trained for 1,500 epochs.

### 4.1.2 Non-Polynomial/Polynomial Features

The models without scrambling the device’s order are the ones which most faithfully follow the approach presented in [8]. Fig. 4.1 shows the evolution of the average MSE error for the training and validation error throughout the 1,500 training epochs.

![Figure 4.1: MSE Models' Without Device Scrambling Training and Validation Error Evolution. The error is evaluated using the MSE loss function.](image)
By comparing both graphs, it is clear that the MSE-P model is able to further reduce training error but it seems more conducive to overfitting since the validation error seems to increase as the model reduces the training error, suggesting a reduction in generalization. To prevent this, either L1 or L2 regularization could have been used, however these methods were not applied in [8].

To evaluate the model's capability of producing valid placements, the training and test outputs (after the training phase) of both models were tested using the TLF defined in 3.14. The average values for all the four factors that make up the loss function as well as the total error are represented in table 4.2. All of the values are already weighted using the values described in table 3.7 and the total loss is calculated using equation 3.14.

<table>
<thead>
<tr>
<th></th>
<th>Non-Polynomial</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Test</td>
</tr>
<tr>
<td>Wasted Area</td>
<td>1.61</td>
<td>1.44</td>
</tr>
<tr>
<td>Symmetry</td>
<td>1.22</td>
<td>1.42</td>
</tr>
<tr>
<td>Current Flow</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Overlap</td>
<td>73.69</td>
<td>132.48</td>
</tr>
<tr>
<td>Total Error</td>
<td>76.52</td>
<td>135.34</td>
</tr>
</tbody>
</table>

The results show, as expected, a higher topological error in the test sets when compared to the training sets. Also, the polynomial model presents a much lower training error but its test error is comparable to its non-polynomial counterpart, further suggesting the occurrence of overfitting (early stopping and/or L1/2 regularization would certainly reduce this effect, as explained in section 2.4.4, but the purpose of this comparison is to point out the model's tendency to overfit). Furthermore, a reduction in wasted area is generally accompanied by an increase in overlap error.

Notice also that overlap is consistently the highest error component. To further understand this error, take the example of the unweighted average overlap error for the test set of the non-polynomial model, $O = \frac{132.48}{700} = 0.19$, meaning that the overlap between devices sums up to 20% of the area of all devices. For visualization purposes, Fig. 4.2 represents a test prediction with approximately the mean error and the mean overlap error.

To further understand any possible causes for this error, table 4.3 contains information regarding the relative amount of times each of the 12 templates was selected as target (i.e. the selected template produced the layout with smallest area) for all of the 10,422 examples. Notice that 91.33% of the examples are distributed amongst only four templates, in decreasing order of relevance: 5, 7, 9 and 4.

<table>
<thead>
<tr>
<th>Template</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Rate (%)</td>
<td>2.76</td>
<td>0.48</td>
<td>5.35</td>
<td>0.00</td>
<td>18.04</td>
<td>25.84</td>
<td>0.00</td>
<td>25.53</td>
<td>0.01</td>
<td>21.92</td>
<td>0.06</td>
<td>0.00</td>
</tr>
</tbody>
</table>

In order to ascertain the impacts of the unbalanced dataset, Fig. 4.3 shows how likely the model is of mistargeting an example given that it was targeted to a specific template. That is, the horizontal axis represents the template selected as target for any given example, the vertical axis represents with which template the prediction made by the model has minimum MSE error, i.e. to which template is the
prediction made closer to. Each column is normalized in relation to the number of examples in each template. As such, the figure shows the probability of the model mistargeting to any of the templates $y$ (vertical axis) given that the template selected as target is $x$ (horizontal axis).

As expected, the largest values are located mainly in the main diagonal which indicates that the model has correctly chosen to copy the target template. However, clear exceptions in the training set, in Fig. 4.3a, include the horizontal lines for template 7 and 4 which present high values throughout, meaning that the model is incorrectly basing its prediction in template 7 or 4. This error is most likely a consequence of the imbalanced dataset as shown through table 4.3. Fig. 4.3b shows the same distribution but for the test set where clearly, a lot of the examples are being mistargeted towards template 5.

The lack of a balanced dataset (which is not easy to produce) leads to a biased model which doesn’t
predict novel layouts, basing its predictions towards the most common templates across all examples. The change of the loss function to a topological evaluation of the prediction reduces the dependency on a balanced dataset and promotes the creation of novel layouts.

To further justify the change in loss function, Fig. 4.4a shows the distribution of the logarithmic average TLF error for each of the \((\text{target template}, \text{most similar template})\) combinations seen in Fig. 4.3. Similarly, Fig. 4.4b shows the distribution of the MSE error, this figure can be interpreted as how different template \(y\) is from template \(x\).

As expected, the lowest MSE errors are located in the main diagonal, contrarily, for the TLF error, the main diagonal presents some relatively high values (e.g. the square with coordinates \((7, 7)\)) and the lowest values are located in cases of mistargeting as is the case of template 10 being mistargeted as template 4. In this particular example, template 10 is relatively similar to template 4 as Fig. 4.4b shows, presenting an average MSE error of around 1.5. However, notice that when template 9 is mistargeted as template 4, it presents a quite low TLF error and quite high MSE error,

![Figure 4.4: Average error for given target template/most similar template pairs.](image)

As expected, the lowest MSE errors are located in the main diagonal, contrarily, for the TLF error, the main diagonal presents some relatively high values (e.g. the square with coordinates \((7, 7)\)) and the lowest values are located in cases of mistargeting as is the case of template 10 being mistargeted as template 4. In this particular example, template 10 is relatively similar to template 4 as Fig. 4.4b shows, presenting an average MSE error of around 1.5. However, notice that when template 9 is mistargeted as template 4, it presents a quite low TLF error and quite high MSE error,

4.1.3 Device Scrambling

In order to discern the MSE models' ability to deal with never seen before topologies, both MSE-NPS and MSE-PS models are trained with the order of the devices’ sizing information scrambled as demonstrated in Fig. 3.7.

The evolution of the training and validation loss for the two models are represented in Fig. 4.6. In both cases, the validation error is significantly higher than the training error (1.75 to 4 times higher), being higher than the validation error of both models show in Fig. 4.1. This discrepancy between the
training and validation error shows that the model is unable to generalize the knowledge obtained during training due to the fact that there is no identifying feature for each device in the input vector.

A template can be defined as a specific relative positioning of the devices adjusted to their sizing characteristics, for example, the template 9 shown in Fig. 4.5a is defined by having device 7 above device 8 but below device 5, however, if the order of the devices is scrambled, there is no way to distinguish between device 6 or 7, or even between device 7 and 0 since there are sizing examples where device 0 is larger than device 7. In a template, the reason why each device is located somewhere relative to any other is the result of the consideration of its topological constraints, hence, since in a sizing based input vector there is no reference to these constraints, it is impossible to distinguish between devices and thus the necessity of including in the input vector each device’s topological constraints’ description.

Finally, the topological evaluation of the training and test predictions for both of these models is represented in table 4.4. The overall error has consistently increased (at least 4 times) in each case when compared to the non-scrambled counterparts in table 4.2.

Figure 4.5: Example of a prediction with high MSE error but low TLF error.
4.2 Topological Models

In this section two different models that use the TLF as loss function are tested. A model which is initialized through training 100 epochs using the MSE loss function and then switched to the TLF for 1,000 more epochs and a model trained for 1,100 epochs using only the TLF. Additionally, both these models use the CDV described in section 3.2. The hypothesis being tested is that the use of the MSE loss function in the early phases of training might speed up the convergence process and result in a better model.

4.2.1 Structure and Training

Both models are trained with a dataset composed by examples of the SSA and the CFSSA topology, as such, the SSA examples’ input vectors are padded in accordance to the process described in section 3.2.4, additionally, the order of the devices is scrambled for each example as explained in section 3.2.5.

For the SSA topology, 10,422 different sizing cases are used while for the CFSSA circuit only 255 different sizing cases. However, to build a balanced dataset, each CFSSA sizing example is augmented

Overall, the lack of the device’s topological constraints’ identifiers in the input vector hinders the detection of patterns which can be used to determine the relative positioning of each device.
45 times changing the order of the devices in each copy totaling 11,475 total examples. The resulting dataset has a total of 21,897 examples which are split into a training (70%), validation (15%) and test set (15%).

For the MSE initialized model, during the training using the MSE function, one of the 12 previously generated templates (the one with smallest area) is chosen as target placement for SSA topology examples, whereas for the CFSSA only two templates were previously generated, so similarly, for each example the one with smaller area is chosen as target.

The rest of the models' parameters are described in table 3.7 (including the TLF weights).

### 4.2.2 General Results

The evolution of the training and validation error for both topological models are represented in Fig. 4.7.

![Figure 4.7](image-url)

(a) MSE initialized topological model  
(b) Fully topological model  

Figure 4.7: Topological models' training and validation Error Evolution. The error is evaluated using the TLF loss function.

The MSE initialization was introduced in order to initialize the model's weights in a stable region where the restrictions were closer to fulfilled, the TLF would then just make slight adjustments and optimize the results. However, it is clear by analyzing both figures that the MSE initialization seems to delay the process of convergence, and that the loss decreases exponentially once the loss function is changed. To discern the influence of each component in the total error, Fig. 4.8 shows the evolution of the individual error components over the training phase for the two topological models.

Notice that the overlap error is the main component error in both models and while overlap and symmetry error can be easily analyzed, both the current flow and wasted area components are quite small in comparison and as such their evolution cannot be studied through Fig. 4.8 (this will be addressed later).

To allow a better comparison between both models, the final average error for the training and test predictions for each model are shown in table 4.5.

By analyzing table 4.5 it is possible to conclude that the difference between the test and training errors are not as significant as for the MSE models. For the MSE models without device scrambling
(the results shown in table 4.2), the test error is 76% and 425% higher than the training error without polynomial features and with polynomial features respectively. For the topological evaluation based models, the test error is only about 20% higher in both cases. So even though the training error is larger for the topological evaluation based models, there is a clear improvement in the generalization of the knowledge gained. Notice in Fig. 4.7 the similarity between the shapes of the validation and the training curves in both cases, it seems that whenever the model reduces the error for the training set, the effect of the update on the network’s weights is equally felt in the validation set, this further corroborates the hypothesis that the generalization of this model is much higher when compared to its MSE counterparts. Notice also in table 4.5 that the MSE initialized model has a much lower current flow error, this being the only clear advantage of the model over its fully topological counterpart.

In order to better interpret the errors of table 4.5, Fig. 4.9 and Fig. 4.10 show the best, the worst and the three quartile predictions for each of the models, on top of each prediction there is a description of its error components.

By analyzing the predictions made, table 4.5 and the evolution of the distinct error components in

Figure 4.8: TLF error components’ evolution throughout training. The error is evaluated using the TLF loss function.
Table 4.5: Mean Topological Error of Topological Models

<table>
<thead>
<tr>
<th></th>
<th>MSE Initialized</th>
<th>Topological</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Test</td>
</tr>
<tr>
<td>Wasted Area</td>
<td>2.75</td>
<td>2.65</td>
</tr>
<tr>
<td>Symmetry</td>
<td>15.40</td>
<td>18.46</td>
</tr>
<tr>
<td>Current Flow</td>
<td>0.66</td>
<td>0.71</td>
</tr>
<tr>
<td>Overlap</td>
<td>63.74</td>
<td>77.28</td>
</tr>
<tr>
<td>Total Error</td>
<td>82.55</td>
<td>99.1</td>
</tr>
</tbody>
</table>

Fig. 4.8, it is clear that the main difficulty of the model is avoiding overlap between devices, clearly representing the majority of the error. In matter of fact, its high associated weight is a result of attempting to reduce said overlap since a higher weight makes its associated gradient higher than the remaining components’, thus prioritizing its decreasement during training. Notice that even for the worst predictions, the model seems pretty capable of interpreting both symmetry and current flow constraints.

4.2.3 TLF Components Correlation Analysis

One question that should be posed now is what’s the influence of each component on any of the others? For example, it’s intuitive that an increase in the prediction’s wasted area should lead to a decrease in the overlap between the devices, however, the relation between the current flow error and symmetry (if any) is harder to visualize. To address this question, an analysis on the correlation between the components for both models was made (for the MSE initialized model, the correlation was studied once the model had been initialized and the training using the TLF began). The study was made using Pearson’s correlation coefficient. Fig. 4.11 shows the results of the analysis.

As expected, there is a very strong negative correlation between wasted area and overlap, with a coefficient of \( \rho = -0.9 \) for the MSE initialized model and \( \rho = -0.8 \) for the uninitialized one. Notice that in every case, the coefficients are not as pronounced for the uninitialized model, however, the signs are consistent across both examples which strengthens the credibility of the most pronounced results. The already examined overlap/wasted area \( (O_L/W_A) \) correlation for example, presents high negative values in both cases meaning that most likely there is a somewhat strong negative correlation between both even if not as strong as the results in Fig. 4.11a might suggest. Other significant examples are the pairs current flow/wasted area \( (C_F/W_A) \), symmetry/overlap \( (Sym/O_L) \) and \( Sym/W_A \). Notice that in every case the registered \( p \)-values meet the criteria \( p < 0.001 \) meaning the correlations are statistically significant.

The pair \( C_F/W_A \) presents high positive correlation between the two, meaning that an increase in wasted area (which translates in an increase of the prediction’s area) is met with an increase in current flow error, this correlation can be explained through the analysis of equation 3.24, the equation makes it so that if two devices are in the incorrect order, e.g. device A is above device B when it should be below it instead, the error is higher if B maintains its position and A is moved further above (i.e. in the wrong direction). This means that for a given prediction with a pair of devices in the wrong order, if the layout is simply expanded outwards maintaining the relative position of each device, its current flow error
Figure 4.9: Sampling of the fully topological model’s test predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure.

increases. In the same manner, once the layout is shrunken, the current flow error decreases. Notice that if originally the two devices were already in the correct order, the error would remain 0 once the layout was expanded, thus, an increase in wasted area can only lead to either the increase in current flow error or its stagnation which can clearly be seen in the scatter plot of both components in Fig. 4.11a.

There is a strong negative correlation between the pair $\text{Sym}/WA$ meaning that an increase in area is followed by a decrease in symmetry error, this relation can be explained when taking into account the high weight associated to the overlap error and the nature of the symmetry between devices. Symmetry between any two devices is fulfilled if their symmetry axis is centered in $x = 0$ and their $y$ coordinates are the same, this presents a fragile equilibrium where both devices have to be placed taking the other into consideration, as such, finding the perfect symmetric placement is a result of small corrections on each of the involved device’s positioning. These small corrections, when made in a compact layout, may increase significantly the overlap error, as such, these corrections are discouraged by the overlap component’s high gradient, so once the layout is expanded these corrections can be made and the symmetry error decreases. This also explains the existing positive correlation between the pair $\text{Sym}/Ol$, since a decrease/increase in overlap error (increase/decrease in wasted area) is followed by a decrease/increase in symmetry error. If the hypothesis made is true, it should be possible to verify these relations in the evolution of the error components, however, due to the different scale of the error
Figure 4.10: Sampling of the MSE initialized topological model’s test predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure.

components, these cannot be easily compared in Fig. 4.8. In order to visualize these influences, Fig. 4.12 shows the evolution of the different components for the uninitialized model, each normalized using minmax normalization (explained in section 2.4.6) so they’re all in the same scale. The figure corroborates the supposition made as the area of the predictions increase in order to reduce the other error components. There is however a noticeable jump in the current flow error at around 100 epochs which is not easily explainable since no other component suffers such drastic changes which could explain this behavior.

4.2.4 Test Dataset Augmentation

Due to the network being robust to the order of the devices in the input vector, every sizing example can be theoretically augmented $15! \sim 1.3 \times 10^{13}$ by changing the order of the devices in each copy. As the purpose of this work is to develop a push-button speed model capable of producing a valid placement solution for a given sizing example, it could be part of the system’s flow to first augment the given example by changing the order of devices before passing these through the model. In practice, as long as one of these augmented vectors produces a valid placement, the objective is fulfilled. As such, in this subsection, each example in the test set of the fully topological model is augmented and a placement solution is generated for each, the best of which is selected as final placement solution.
In order to fulfill the desired push-button speed each example was only augmented 100 times. To make sure the desired speed is still met, a test was run to compare the time it took to make a single prediction and 100 predictions. A single prediction took 0.0304 seconds while 100 predictions took 0.0396 seconds. This slight increase in time is considered within satisfactory bounds.

The mean topological errors for the augmented test set and comparison with the not augmented test and training sets are represented in table 4.6.

Table 4.6: Mean Topological Error of Fully Topological Model with Test Set Augmentation ($\times 100$)

<table>
<thead>
<tr>
<th>Topological Component</th>
<th>Training</th>
<th>Test</th>
<th>Augmented Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasted Area</td>
<td>2.98</td>
<td>2.85</td>
<td>3.39</td>
</tr>
<tr>
<td>Symmetry</td>
<td>15.58</td>
<td>17.70</td>
<td>11.12</td>
</tr>
<tr>
<td>Current Flow</td>
<td>4.02</td>
<td>4.40</td>
<td>1.09</td>
</tr>
<tr>
<td>Overlap</td>
<td>63.36</td>
<td>74.77</td>
<td>11.04</td>
</tr>
<tr>
<td>Total Error</td>
<td>85.94</td>
<td>99.72</td>
<td>26.65</td>
</tr>
</tbody>
</table>

There is a significant reduction in error, particularly in the overlap component, making it even lower than the training set error. To best compare the results, the predictions were sampled similarly to what’s seen in Fig. 4.9. The results are shown in Fig. 4.13 and the difference in quality is clear when compared to Fig. 4.9. With the augmented test set, the worst prediction in Fig. 4.13b is better than the first quartile prediction from Fig. 4.9c.

4.2.5 Generation of Novel Layouts

It is one of the main goals, with this work’s approach, to create a model capable of generating a variety of placement solutions in order to give the analog IC designer the opportunity to pick a template most suitable to its preferences and the project’s requirements. In order to test the fully topological model’s capability of producing novel layouts, two sizing scenarios were picked at random from the test dataset,
one of the SSA topology and one of the CFSSA topology. Each of these two was augmented (with the order of devices scrambled) 1000 times and predictions were generated for each copy. Out of all the predictions, the 9 with least error are chosen as suggested templates. If the model is capable of producing novel layouts, these 9 should be considerably different from each other.

Fig. 4.14 and 4.15 show the results for the SSA and CFSSA examples respectively.

The results in Fig. 4.14 show a much more noticeable difference between predictions and its safe to say a variety of templates are suggested. The predictions in Fig. 4.15 however, show more similarity between them. This might be due to the fact that the increasing number of devices limits the amount of possible placements due to increasing number of devices. Another possibility might be the lack of different sizing cases for the CFSSA topology. Different sizing cases are the main cause for innovation when considering only one topology since these may lead to overlapping and thus the model is forced to correct these overlaps by searching for a different template. As such, the lack of sizing cases may lead to the formulation of a more general template which works for all the seen sizing cases. Notice that in Fig. 4.14 all predictions have the same sizing characteristics, it means that templates are being attributed to the order of the devices, which is beneficial for the application since it makes results like these and the ones discussed in section 4.2.4 possible.

Finally, to enforce the strict fulfillment of the constraints, the predictions made can be easily post-processed through a simple symmetry enforcing compacting algorithm. The algorithm starts by fixing the minor mismatches in symmetric pairs by placing them at their average \( y \) coordinate and, keeping their distance between one another fixed, shifting their symmetry axis to match the reference axis. After, the \( y \) coordinates of all devices are doubled, thus doubling the \( y \) separation of all devices, after which the \( x \) coordinate can be more easily compacted avoiding overlaps (this compacting is done using an interval tree). Finally, the \( y \) coordinates are also compacted using an interval tree. This simple post-
Figure 4.13: Sampling of the topological model’s augmented test set predictions. The weighted error components of each prediction as well as the total loss value are printed on top of each figure.

processing step took 40 ms. The post-processed compact placements for the top 9 CFSSA placements are presented in Fig. 4.16.

It is clear that this simple post-processing step has not only successfully corrected the slight symmetry mismatches as it produced much more compact placements at a slight computational cost. Note that this fast optimization is only so effective due to the quality of the initial placement. If symmetry mismatches or overlaps were more noticeable, a much more costly process would have to be used.

4.2.6 Folded Single Stage Amplifier Circuit Test

In order to test the model’s performance with the never seen topology, a new circuit is tested, the Folded Single Stage Amplifier (FSSA). Its schematics is presented in Fig. 4.17 with the current paths superimposed.

In order to convert the circuit into an input vector, both its symmetry and current flow constraints should be properly encoded. The circuit is composed by 14 devices which is less than the 15 used in training so the input vector will be padded with an empty device descriptive vector. Notice that the circuit is composed by four different self-symmetric devices which might cause some problems to the network. The circuit is composed by 5 distinct current paths which is less than the limit of 8. Notice though that two current paths are composed by four devices while the remaining topologies only considered a
maximum of three devices per current flow. Therefore, a problem arises when building the current flow constrain vector for each of the FSSA’s devices, the network is trained for the one hot vector to be the concatenation of 8 one hot vectors of length 3 and simply concatenating one hot vectors of length 4 would encode false information. To address this issue, the two current paths which are composed by four devices are split into two current paths each on the third device, this being the last and first device of the now split paths. For example, the following current path \([PM3, PM5, NM5, NM7]\) is split into current paths \([PM3, PM5, NM5]\) and \([NM5, NM7]\). The resulting padded current flow constraint descriptive vector for each device is shown in table 4.7.

One single sizing case was augmented 1,000 times by scrambling the order of the devices in each example, then, similarly to what was done in section 4.2.5, the 9 predictions with smaller error were selected as final output. Table 4.8 shows the mean loss and its weighted components for the top 9 predictions while these can be visualized in Fig. 4.18.

Although the mean loss value for these 9 predictions is quite high, it is still comparable to the test set topological errors for the MSE-NP and MSE-P models shown in table 4.2. However, note that the

Figure 4.14: 9 different predictions for a single sizing case of the SSA topology after 1000 scrambled copies of the same example were tested
Figure 4.15: 9 different predictions for a single sizing case of the CFSSA topology after 1000 scrambled copies of the same example were tested

FSSA circuit topology introduces 4 different self-symmetric devices. Up until this point the model had only ever dealt with one single self-symmetric device and has learned to deal with it accordingly, placing it in the top center of the layout (top due to the current flow constraints). By applying this simple logic to the new topology, all four self-symmetric devices are placed in the top center of the layout resulting in a big overlap between these devices as it can be seen in any of the 9 predictions. Note that this at least means that the model is capable of recognizing self-symmetric devices. In order to better evaluate the predictions made, Fig. 4.19 shows the same predictions but without the self-symmetric devices. While there is still some significant overlap, it is clear that symmetric pairs were clearly identified. Note as well that the current flow error is quite minimal in all predictions.

Overall the model seems to have correctly identified the topological constraints of the circuit and seems to have attempted to satisfy them using the methods which it had learned before.
Figure 4.16: Post-processed top 9 CFSSA predictions for a single sizing scenario.

Figure 4.17: Folded Single Stage Amplifier: schematic with current paths highlighted
Table 4.7: Folded Circuit Device’s Padded Current Flow Description

<table>
<thead>
<tr>
<th>Current Path #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM0 (0)</td>
<td>1 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>PM3 (1)</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>PM4 (2)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>PM1 (3)</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>PM5 (4)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>PM6 (5)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM1 (6)</td>
<td>0 0 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM3 (7)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM4 (8)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM5 (10)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM6 (11)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM7 (12)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>NM8 (13)</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Mean Topological Error of the Top 9 Folded Circuit Predictions (Out of 1,000)

<table>
<thead>
<tr>
<th>Wasted Area</th>
<th>Symmetry</th>
<th>Current Flow</th>
<th>Overlap</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.55</td>
<td>50.34</td>
<td>0.32</td>
<td>110.57</td>
<td>162.78</td>
</tr>
</tbody>
</table>
Figure 4.18: 9 different predictions for a single sizing case of the FSSA topology after 1,000 scrambled copies of the same example were tested.
Figure 4.19: 9 different predictions for a single sizing case of the FSSA topology after 1000 scrambled copies of the same example were tested.
Chapter 5

Conclusions

This chapter presents the conclusions of this work, as well as future directions for further applications of ANNs towards the automation of the placement process of analog layout design.

In this work, a new approach to the use of ANNs towards automatic analog IC placement was tested. The introduction of a loss function which evaluates the quality of a prediction based on the fulfillment of multiple considered constraints allowed the generation of novel placements solutions tailor made for each example, as well as eliminating the need for the pre-production of tested legacy layouts.

The use of a CDV as the input vector allows the distinction between different circuit topologies and consequently, together with the input vector padding method, enables the introduction of multiple circuit topologies using the same model.

The use of a constraint identification and satisfaction approach results in a greater generalization of the acquired knowledge as the model seems to correctly identify constraints in never seen before topologies.

The use of ANNs seems suited to the problem as it enables the production of solutions at push-button speed as well as being especially good at detecting patterns in input/output pairs which are then generalized for the surrounding input space.

However, the results fall somehow short of the necessary performance for the development of a fully automation tool since the model seems incapable of producing valid layouts for never seen before topologies while still producing too many invalid layouts for the training examples. Yet, the results seem promising and serve as a proof of concept for justifying further optimization and research on the proposed approach.

5.1 Future Work

This work served mainly to serve as proof of concept for the implementations done in the input vector and loss function, as such, the necessary focus was never given to the tuning of the hyperparameters which constitutes a pivotal part on the development of any ANN model. The tuning of these parameters may not only lead directly to better performance but might also optimize the learning process enabling the use of
more training examples which should certainly lead to better results. One of these hyperparameters are the loss function’s weights which were set through trial and error, for optimal results some grid search on the parameter’s space (with the use of cross-validation to faithfully compare models) should have been performed, however this would take too much time with the available machines. Note even that the use of ANNs may not have been optimal, other regression models like random forest might have been tested and could result in better performance.

An augmentation of the training set could be done in two different manners: considering the use of the same topologies, each example in the training set could be augmented by creating copies of it with the order of the devices scrambled; or, new circuit topologies with never seen before constraints could easily be introduced, due to the fact that the used loss function doesn’t require the production of target placements, input vectors could be artificially mass generated to create a vast training set. However, these input vectors should have only feasible constraints, unfeasible constraints include a device symmetric to two others or contradictory current flow constraints.

In order to include this model in a EDA tool, fast post-processing methods which would correct slight mistakes in the prediction made should be considered.

Future development of this tool would also most likely pass through the addition of more components to the loss function to encode the remaining topological constraints to which analog IC circuits are usually subjected to such as wiring length. The inclusion of more constraints would make convergence harder, however, it would enable designers to define the weights associated to each in order to get the placement solution more suitable for the problem, here, more constrains considered during training results in more optimal solutions for a bigger variety of problems.

The encoding of each constraint in the input vector could be further researched as well, more efficient representations would result in smaller input space and further optimization of the learning process. Additionally, cross features such as each device’s area or dimensions’ ratio could reduce the predictions’ error, especially the overlap component.

One of the great challenges in creating a model capable of multiple topology layout generation is that when bigger and bigger topologies (composed by more devices) are considered, the expansion of the input space may hinder the learning process especially for smaller topologies, as such, Recurrent Neural Networks may prove an interesting alternative as each device would successively be put into the layout taking into consideration the devices which had already been placed before.
Bibliography


