



Titre: Modelling of Artificial Neural Networks with Integrated Photonics
Title:

Auteur: Océane Destras
Author:

Date: 2020

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Destras, O. (2020). Modelling of Artificial Neural Networks with Integrated Photonics [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/5553/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/5553/>
PolyPublie URL:

**Directeurs de
recherche:** Gabriela Nicolescu
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Modelling of Artificial Neural Networks with Integrated Photonics

OCEANE DESTRAS

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie informatique

Décembre 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Modelling of Artificial Neural Networks with Integrated Photonics

présenté par **Oceane DESTRAS**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Guy BOIS, président

Gabriela NICOLESCU, membre et directrice de recherche

Tarek OULD BACHIR, membre

*To my companion and my accomplice,
I can never thank you enough for all the support and tenderness you give me.*

*To my family,
thank you for your guidance and love, I miss you . . .*

DÉDICACE

*À mon compagnon et mon complice,
je ne te remercierai jamais assez pour tout le soutien et l'affection que tu me portes.*

*À ma famille,
merci pour votre appui et votre amour, vous me manquez ...*

ACKNOWLEDGEMENTS

First of all, I would like to thank my research director, Gabriela Nicolescu, for giving me the opportunity to lead this research project, as well as post-doctoral fellow Felipe Gohring de Magalhaes for his help and guidance.

I would also like to thank the members of the jury Mr. Guy Bois and Mr. Tarek Ould-Bachir for agreeing to examine my work.

I want to express my gratitude to FRQNT for funding me and, at the same time, offering me the opportunity to produce a research paper.

I would like to thank my laboratory colleagues for advising me but also for making me laugh during these two years.

Finally, I give thanks to my family and friends for encouraging me throughout the dissertation of my thesis.

REMERCIEMENTS

Je remercie tout d'abord ma directrice de recherche, Gabriela Nicolescu, pour m'avoir donné l'occasion de diriger ce projet de recherche, ainsi que le post-doctorant Felipe Gohring de Magalhaes pour m'avoir aidée et guidée.

Je tiens aussi à remercier les membres du jury Mr Guy Bois et Mr Tarek Ould-Bachir pour avoir accepté d'examiner mon travail.

Je souhaite exprimer ma reconnaissance envers l'organisme FRQNT pour m'avoir financée et, par la même, offert la possibilité de produire un travail de recherche.

Je remercie mes collègues de laboratoire pour m'avoir conseillée mais aussi faite rire durant ces deux années.

Finalement, je souhaite remercier ma famille et mes amis pour m'avoir soutenue tout au long de mon mémoire.

RÉSUMÉ

Les algorithmes d'apprentissage profond sont des éléments clés d'interconnection des objets du 21^{ème} siècle. Ils permettent de traiter de grands ensembles de données rapidement et de manière automatique. Cependant, la quantité de données à notre disposition ne fait qu'augmenter, et avec elle la latence et la consommation d'énergie de ces algorithmes. Avec la fin de la loi de Moore, l'application du Deep Learning sur ordinateur électronique devient un défi important à relever. La photonique, permettant une grande puissance de calcul et le traitement de données à haute vitesse, apparaît comme une potentielle alternative. Nous étudions donc la possibilité de reproduire les différentes fonctions d'un algorithme d'apprentissage profond avec des composants optiques. Ces fonctions peuvent être résumées en quatre catégories :

1. Pré-traitement des données d'entrée
2. Propagation avant des données
3. Analyse des données en sortie
4. Optimisation du réseau de neurones

Une première étape vers l'utilisation de réseaux optiques d'apprentissage profond, et l'objectif de ce mémoire, est la production de circuits optiques conçus pour une tâche précise, telle que la reconnaissance d'images. Nous nous retrouvons donc dans le cas de figure suivant : le pré-traitement des données d'entrées est effectué en dehors du réseau optique, et permet, en partie, la conversion des données d'entrée en signaux optiques. L'analyse des données en sortie est exécutée en dehors du circuit, grâce à un photodétecteur qui permet de transformer la lumière en une grandeur mesurable. L'optimisation du réseau de neurones se fait quant à elle au préalable de la fabrication du réseau optique, sur un ordinateur électronique conventionnel. Une fois optimisé, le réseau de neurones doit pouvoir être reproduit avec des composants optiques à partir des résultats obtenus. Dans le cas d'un perceptron multicouche, la propagation des données à travers le réseau de neurones combine deux principales opérations :

1. La multiplication matricielle
2. La fonction d'activation

La faisabilité de la multiplication matricielle optique a déjà été démontrée par le biais d'interféromètres de Mach-Zendher. La fonction d'activation quant à elle est plus complexe à reproduire optiquement de part sa non-linéarité.

Dans ce mémoire, nous proposons en premier lieu une méthodologie pour optimiser un réseau de neurones composé de fonctions entièrement réalisables optiquement. Ensuite, nous suggérons l'utilisation de fonctions polynomiales optiques comme fonctions d'activation. Finalement, nous utilisons la méthodologie choisie pour expérimenter ces différentes fonctions d'activations polynomiales et étudier leur capacité à remplacer les fonctions d'activations originelles.

Nous effectuons différents tests en utilisant l'architecture simple d'un perceptron multicouches: un test de classification de données et un test de classification d'images. Dans le premier cas, nous obtenons des résultats similaires à ce qu'un algorithme traditionnel pourrait trouver. Ce n'est pas le cas pour le deuxième test. Il semblerait qu'une architecture plus complexe soit nécessaire pour une reconnaissance d'images effective. En effet, avec l'utilisation de réseaux convolutionnels, nous obtenons des résultats supérieurs à ceux des réseaux de neurones optiques de l'état de l'art.

Les fonctions polynomiales semblent être une bonne alternatives aux fonctions d'activations déjà existantes et peuvent être introduites dans un réseau de neurones optiques. L'implémentation complète de réseaux optiques convolutionnels reste encore en phase de développement. Elle est la prochaine étape essentielle à la production de réseaux de neurones optiques efficaces pour la reconnaissance d'images.

ABSTRACT

In the information age, and particularly now in the 21st century, mankind has seen a radical shift in the world. There has never been so much data, and its rate of growth keeps climbing. We can safely say that objects are now deeply interconnected. A key tool to handle this huge amount of data and complex connections is deep learning. Deep learning algorithms help us process sizeable data sets rapidly and without human intervention. However, as we accumulate more and more data, even deep learning algorithms are not enough: their power usage is too high, and as the data sets get bigger, so too does the processing time. The limitation of Moore's law by physical barriers, makes the application of deep learning on electronic computers very challenging. Current research points to the usage of alternative technologies in order to overcome these limitations, whereas integrated photonics emerges as one viable solution. Indeed, optical components would allow a bigger computing power and better processing speed. Here, we study the possibility of replicating the functions we use in traditional deep learning algorithms with optical components. We can classify these functions into four different categories:

1. Pre-processing of input data
2. Forward propagation of data
3. Output data analysis
4. Neural network optimization

A first step towards creating optical deep learning networks is the production of optical networks designed to fulfill specific tasks, such as image recognition. In this thesis, we will study the feasibility of programming optical networks to create deep learning algorithms, by following the four categories presented before. Thus, we proceed as follows: first, the input data is pre-processed outside of the circuit. Then, the output of the pre-processing is converted into optical signals, which are fed into the optical circuit. Afterwards, the output data is received by a photodetector that converts the light into a measurable quantity and analyzed outside of the circuit. Lastly, the optimization of the neural network, the backward propagation, is done on a conventional electronic computer. Once optimized, the neural network itself should be buildable using optical components only, thanks to the data obtained earlier. In the case of a multilayered perceptron, the propagation of data through a neural network is made of two main operations:

1. Matrix multiplication
2. Activation function

Matrix multiplication has already been successfully recreated using Mach-Zendher interferometers. However, because of its nonlinearity, the activation function is more difficult to reproduce optically.

In this thesis, we first offer a methodology to optimize a neural network made solely of optically replicable functions. Secondly, we suggest a way to implement activation functions optically with a polynomial approximation. Finally, the methodology is used to test the proposed activation functions and study their ability to replace their conventional versions.

Using the simple architecture of a multilayered perceptron, we run two series of tests. The first being tests of data classification, and, the second, tests of image classification. The data classification task gives as good results as those obtained by a conventional algorithm. As for the image classification, it seems that the chosen architecture is not appropriate to the dataset and not sufficient to obtain competitive results. Indeed, when we reiterate the experiment using convolutional networks, we obtain better results, on the MNIST dataset, than the optical neural networks reported in the state-of-the-art.

Obtained results indicate that polynomial functions are a good alternative to the activation functions found in conventional neural networks, and they can be replicated in optical neural networks. The implementation of complete convolutional optical networks is still in development but is the next step towards producing optical neural networks that can compete with their more conventional counterparts.

TABLE OF CONTENTS

DÉDICACE	iv
ACKNOWLEDGEMENTS	v
REMERCIEMENTS	vi
RÉSUMÉ	vii
ABSTRACT	ix
TABLE OF CONTENTS	xi
LIST OF TABLES	xiv
LIST OF FIGURES	xv
LIST OF SYMBOLS AND ACRONYMS	xvii
LIST OF APPENDICES	xix
CHAPTER 1 INTRODUCTION	1
1.1 Context and Motivations	1
1.1.1 The Internet of Things	1
1.1.2 Key technologies and architectures for the IoT	1
1.1.3 Optical Neural Networks	3
1.2 Objectives & Contributions	4
1.3 Document Organization	4
CHAPTER 2 BASIC CONCEPTS	5
2.1 Artificial Intelligence (AI)	5
2.1.1 Perceptron	7
2.1.2 Structure of a Multiple Layer Perceptron	8
2.1.3 Activation function	10
2.1.4 Backward propagation	13
2.1.5 Other types of Neural Networks	14
2.2 Photonics	16

2.2.1	General information	16
2.2.2	Mach-Zehnder Interferometer (MZI)	17
2.2.3	Micro-Ring Resonator (MRR)	18
2.2.4	Common concepts	19
CHAPTER 3 LITERATURE REVIEW		21
3.1	Electronic Implementation of Artificial Neural Networks	21
3.1.1	Accelerated deep neural networks	21
3.1.2	Spiking neural networks	21
3.2	Photonic Implementation for Artificial Neural Networks	23
3.2.1	Semiconductor Optical Amplifier (SOA)-based neural networks	24
3.2.2	Coherent nanophotonic circuits	24
3.2.3	Optical activation function	26
3.3	Conclusion	26
CHAPTER 4 METHODOLOGY FOR A DEEP LEARNING MODEL IMPLEMENTATION		28
4.1	Data Processing	28
4.1.1	Choosing the dataset	29
4.1.2	Preprocessing the dataset	30
4.2	Architecture of the Model	31
4.3	Training the Model	33
4.3.1	Optimization	33
4.3.2	Evaluation metrics	35
4.4	Summary	37
CHAPTER 5 OPTICAL NEURAL NETWORK MODEL		38
5.1	Data Processing	38
5.2	Model's Architecture	38
5.2.1	Optical matrix multiplication	39
5.2.2	Optical activation function	43
5.3	Training the Model	48
5.4	Summary	48
CHAPTER 6 METHODOLOGY APPLICATION TO CLASSIFICATION PROBLEMS		50
6.1	Experimental Setup	50
6.2	Classification of the Iris Dataset	50

6.3	Classification of the MNIST Dataset	56
6.4	Scalability of Our Polynomial Activation Function in Convolutional Neural Networks	61
6.5	Discussion	62
CHAPTER 7 CONCLUSION		63
7.1	Summary of Works	63
7.2	Limitations	63
7.3	Future Research	64
REFERENCES		65
APPENDICES		75

LIST OF TABLES

Table 2.1	Advantages and drawbacks of activation functions	12
Table 3.1	Comparison table of Spiking Neural Network (SNN) chips performance	23
Table 3.2	Comparison table of photonic neural networks	25
Table 4.1	One-hot encoding	31
Table 4.2	Domain of definition of the hyperparameters of our model	33
Table 5.1	Monthly wage of 19 people, before and after normalization	38
Table 6.1	Keras functions used in our model, and their arguments	50
Table 6.2	Domain of definition of the hyperparameters of our model	51
Table 6.3	Configuration of the deepest model to select the number of epochs . .	51
Table 6.4	Configuration of the model when the highest accuracy is reached . . .	52
Table 6.5	Comparison tables	53
Table 6.6	Comparison table when the activation function is Tanh	53
Table 6.7	Configuration of the final model	54
Table 6.8	Comparison table with weight constrictions of 1	55
Table 6.9	Domain of definition of the hyperparameters of our model	56
Table 6.10	Configuration of the deepest model to select the number of epochs : MNIST	57
Table 6.11	Comparison table	58
Table 6.12	Results on MNIST for one hidden layer	58
Table 6.13	Results on MNIST for two hidden layers	59
Table 6.14	Results on MNIST for three hidden layers	59
Table 6.15	Accuracy obtained on MNIST for state-of-the-art models and the one proposed in this thesis	62

LIST OF FIGURES

Figure 2.1	Simplified version of a single perceptron	8
Figure 2.2	A fully connected Neural Network	9
Figure 2.3	Sigmoid function	10
Figure 2.4	Hyperbolic Tangent Function	11
Figure 2.5	ReLU Function	11
Figure 2.6	Derivative of the Sigmoid Function	12
Figure 2.7	Evolution of the loss with the gradient descent method	13
Figure 2.8	Example of a convolution layer for horizontal edge detection	15
Figure 2.9	Recurrent Neural Network	16
Figure 2.10	Mach-Zehnder Interferometer	17
Figure 2.11	Phase shift of the sine function	18
Figure 2.12	Micro-Ring Resonator (MRR)	19
Figure 4.1	Main steps of the optimization methodology	28
Figure 4.2	Processing flowchart	29
Figure 4.3	Case of underfitting	36
Figure 4.4	Case of overfitting	37
Figure 5.1	Optical representation of a matrix multiplication using an Mach-Zenhder Interferometer (MZI)	40
Figure 5.2	Schematic representation of a MZI (a) and a mesh of MZIs (b)	40
Figure 5.3	Mesh of MZIs to realize a 3×3 unitary matrix	43
Figure 5.4	Sigmoid activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$	45
Figure 5.5	Smooth ReLU activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$	45
Figure 5.6	Hyperbolic tangent activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$	46
Figure 5.7	Output values of $B_6(ReLU)$ with and without $WC = 1$	47
Figure 5.8	Output values of $B_6(Sigmoid)$ with and without $WC = 1$	47
Figure 5.9	Flowchart of a neural network's forward propagation with an optical model and an electronic model	49
Figure 6.1	Plot of the training loss and validation loss	52
Figure 6.2	Measurement of the accuracy with different value of batch size and the activation function Tanh	54

Figure 6.3	Measurement of the accuracy over 50 tests	55
Figure 6.4	Plot of the training loss and validation loss	57
Figure 6.5	Plot of the training loss and validation loss on MNIST with a Multiple Layer Perceptron (MLP)	60
Figure 6.6	Plot of the training loss and validation loss on MNIST with a MLP and dropout=0.1	60
Figure 6.7	Plot of the training loss and validation loss on MNIST with a MLP and dropout=0.2	61
Figure 6.8	Plot of the train and validation loss on MNIST with a Convolutionnal Neural Network (CNN)	61

LIST OF SYMBOLS AND ACRONYMS

AI Artificial Intelligence

FPGA Field-Programmable Gate Array

ML Machine Learning

DL Deep Learning

DNN Deep Neural Network

NN Neural Network

ANN Artificial Neural Network

MLP Multiple Layer Perceptron

ReLU Rectifier Linear Unit

CNN Convolutionnal Neural Network

RNN Recurrent Neural Network

PNN Photonic Neural Network

SOA Semiconductor Optical Amplifier

ONN Optical Neural Network

SNN Spiking Neural Network

MZI Mach-Zenhder Interferometer

MRR Micro-Ring Resonator

CPU Central Processing Unit

GPU Graphics Processing unit

IoT Internet of Things

FPGA Field-Programmable Gate Array

STDP Spike-Timing Dependent Plasticity

LIST OF APPENDICES

Appendix A	Clement's optical model for matrix multiplication	75
------------	---	----

CHAPTER 1 INTRODUCTION

This chapter discusses the challenges and benefits of having an all Optical Neural Network (ONN) and then introduce the objectives and contributions of the thesis.

1.1 Context and Motivations

1.1.1 The Internet of Things

Internet of Things (IoT) are networks composed of different types of objects interconnected thanks to the internet. It is expected that these systems will grow more independent as time goes by, thanks to the integration of smaller and more intelligent components [1]. Each of these objects is part of a bigger network and communicates with other objects, collecting and exchanging information. As a consequence, this network generates enormous amounts of data [2], which may be a challenge to store and process. The idea of the IoT is to build intelligent networks that process data in a decentralized way, not relying on a single, centralized, processor [2].

The IoT can possibly enhance every product, in a variety of fields ranging from connected houses [3] equipped with smart devices, to autonomous cars [4] and smart cities [5]. It is particularly important for the development of Industry 4.0 [6]: for example, all-connected systems can help manage the supply chain. Intelligent equipment such as sensors and communication devices may detect an anomaly and act on it quicker than a human. The possibilities offered by the IoT makes it attractive for most of the key actors in the field [7] [8].

1.1.2 Key technologies and architectures for the IoT

In the context of growing connection between the internet and objects, Artificial Intelligence (AI) algorithms show promise for the future of the IoT. They offer the possibility to use interconnected systems for smart cities such as smart traffic lights [9] with tools to manage their energetic sustainability [10] and help the development of smart factories [11]. AI algorithms do not require a lot of bandwidth and have a low latency [12]. Recently, Machine Learning (ML), a subfield of AI, has experienced a strong growth thanks to the use of Graphics Processing units (GPUs) [13].

Before the 21st century, AI algorithms were running on Central Processing Units (CPUs). The limitation on using CPUs lies in the computational behavior which is sequential and

that the AI algorithms needed to perform a lot of mathematical operations. One solution would have been to use CPUs with multi-cores. However, parallelization capacities of CPUs are not as high as GPUs' [14]. GPUs can perform a lot of mathematical operations in parallel, therefore making it feasible to use MLPs, Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) algorithms [15]. Field-Programmable Gate Array (FPGA) is also a technology that evolved a lot, enabling the integration of AI algorithms in embedded systems. Thanks to that, new technologies such as IBM TrueNorth [16] and Google TPU [17] could emerge. ML is now one of the most promising fields of AI, thanks to the evolution of Deep Learning (DL), a subset of ML. Many ML applications have been considered for IoT systems [18], such as network intrusion prediction [19] or self-organizing cellular networks [20]. In fact, DL is already used by Big Data companies as to better handle their large volumes of data [21].

Recently, interest in studying the power consumption of neural networks has been emerging [22]. Indeed, for a long time researchers were mainly concerned by the improvement of their models' accuracy [23]. In 2018, the amount of data shared worldwide reached 33 Zettabytes [24]. With the explosion of available digital data, deep learning algorithms will have more and more information to process, which means the training of these algorithms will consume more and more time and power. Researchers continuously work to build new architectures to solve this issue using electronic components, like the late GPUs of NVIDIA [25].

In 1965, in an article about the future of transistor technologies, [26], it was predicted that the capacity of an integrated circuit would double yearly for at least a decade: this was named Moore's law. However, since the beginning of the 21st century, this law has been called into question. Indeed, in 2002, a paper was published saying that Moore's law would potentially no longer be applicable in six to eight years [27]. During the Turing lecture of 2018, John L. Hennessy and David A. Patterson announced the end of Moore's law, saying that "single core performance improved only 3%" in 2017 [28].

Nowadays, DL algorithms are already facing a big challenge: the exponential growth of data to process. With the end of Moore's law, state-of-the-art architectures for AI algorithms will not be enough to process and store such amounts of data [29]. Different approaches can be taken to meet this issue. Knowing that the first objective of a DL algorithm is to imitate the functioning of a brain thanks to Artificial Neural Networks (ANNs), one solution to consider is creating ANNs closer in behavior to biological neural networks, such as SNNs. Another idea is to forego integrated electronic circuits entirely. Some researchers have already started to look for the possibility of producing Optical Neural Networks (ONNs) [30] [31].

1.1.3 Optical Neural Networks

Silicon photonics enable the integration of photonic and electronic devices on the same platform [32]. It is said to be the technology with the most potential regarding the production of photonic circuits, promising satisfactory performance at low cost [33]. One important feature of a photonic network is its process of communication: parallelism, which means that several packets of information can cross the network simultaneously without interfering with each other. In the case of an ANN, it allows the processing of all the data at the same time. Another promising feature is the fact that ONNs would be able to overcome electronics solution in terms of speed and energy efficiency [34]. A heavy operation for a conventional computer, such as a matrix multiplication [35], can be computed at the speed of light if implemented in a photonic network [36]. Mach-Zehnder Interferometers (MZIs) and Micro-Ring Resonators (MRRs) are both promising and mature technologies for optical processing and integration [37]. MZIs are robust, withstand changes of temperature well and can handle large amounts of data [38].

One of the biggest challenges in replicating ANNs with ONNs is to be able to implement optically every step of a classic ANN. The implementation of an optical matrix multiplication has been proven already in [30] [39] but the activation function, which is an essential function in an ANN, was not fully addressed. The existing contributions offer either ONNs implementations where the activation function is performed electrically or, when implemented optically, use non-linearities. In the first case, the conversion of information from the optical circuit is achieved with the activation function on a computer and then the output is converted back to the optical circuit. However this means that the circuit can be limited by electronic clock rates and ohmic losses [30]. In the second case, these complex non-linearities are not comparable to those used by the field of AI. Indeed, there is a general difficulty in linking the fields of AI and photonics. To overcome this challenge, we need to reduce the gap between what is produced optically and what can be implemented in an ANN.

This thesis aims to close the gap between the fields of photonics and AI and build a model from which future interdisciplinary projects could evolve. We propose a solution to implement all optical activation functions similar to those already used in ANNs. We build and train an approachable AI model that can potentially be produced optically thereafter. In order to do that, we use the Keras¹ library [40] to simulate our model, as it sits as a user-friendly library.

¹This library allows to build deep neural networks and machine learning models. Its documentation is accessible online.

1.2 Objectives & Contributions

The objective of this thesis is to showcase the benefits of using optical components, instead of the usual computer architectures. They would allow to increase the speed of calculation without having to deal with a potential increase in power consumption, thus making the use of artificial intelligence algorithms more accessible.

Research question: How can integrated photonic be used to imitate the functionality of an ANN?

To answer this question we target these main objectives :

- Define an AI model enabling the implementation of an ONN;
- Define an all-optical activation function with a behavior similar to conventional activation functions, and;
- Adapt the different method of forward propagation used in an ANN to be as effective when used in an optical circuit.

Following the previous objectives, the contributions of this thesis are:

1. Proposition of optical polynomial activation functions that mimics the behavior of conventional activation functions (Sigmoid, Tanh and the Rectifier Linear Unit (ReLU)) using well established optical technologies (MZIs and MRRs), and;
2. Proposition of a methodology to build and train an AI model, with the Keras library, that can then be implemented optically.

1.3 Document Organization

Chapter 2 presents concepts in AI and photonic, more precisely the characteristics of a neural network and the functioning of MRRs and MZIs. In Chapter 3, we present a literature review of the works aiming to produce faster Neural Networks (NNs), electronically and optically. This chapter allows us to justify our choice to study ONNs and at the same time present its limitation regarding the possibility of an all-optical activation function. A general methodology to optimize a generic ANN is explained in Chapter 4, followed by a study around the possibility of implementing it optically, and the activation function, in Chapter 5. Finally, Chapter 6, presents the performance of several of our models comparing them with the state-of-the-art.

CHAPTER 2 BASIC CONCEPTS

This chapter presents the key notions of this thesis. Section 2.1 introduces Artificial Intelligence (AI) concepts, defining the difference between AI, Machine Learning (ML) and Deep Learning (DL) as well as the requirements of a well-functioning Artificial Neural Network (ANN). Section 2.2 describes the characteristics of the optical components we need to generate our Optical Neural Network (ONN).

2.1 Artificial Intelligence (AI)

Artificial Intelligence is the field of research containing the technologies and algorithms showing some kind of intelligence in solving problems humans could encounter (i.e., image recognition and language processing) [41]. Indeed, these difficulties are easily solved by humans but implementing the same kind of problem-solving ability in machines is very complex. One way to address this challenge is through Machine Learning (ML), a subgroup of AI [42].

Broadly speaking, ML corresponds to the computer-based algorithms for AI. The goal of the ML algorithms is to build a mathematical model able to predict accurate outputs for a given task. The structure of the model is first defined, and its parameters are established randomly. The ML algorithm receives as input a dataset of the information it has to process and learn from, called the training set. This set of data then undergoes a series of transformations, among them are the matrix multiplications and activation functions introduced in Section 2.1.3. Finally, the algorithm modifies the parameters of the model (Section 2.1.4) so that the difference between the obtained result and the expected result decreases [43]. For example, for an image recognition task, the dataset would be an assortment of pictures. Then the algorithm trains itself to recognize each image of the training set of this dataset. The quality of the algorithm will be defined by its accuracy, the percentage of correct answers in relation to the total number of inputs, on another subset of the dataset called the test set [44].

There are four main types of ML algorithms:

Supervised learning: In a case of supervised learning, both the input vector, X , and the expected output vector, Y , of the training set are known. It is said that the data are labeled. The algorithm approximates a function, f , that associates X and Y with the relation $f(X) \simeq Y$, by studying the training set [45]. To determine in advance how it would perform on an unknown dataset, the function is used to predict the labels of the test set. The obtained

results are then compared to the expected results.

Unsupervised learning: This algorithm is used when the training set is not labeled. Therefore there is no expected output vector Y . To overcome this lack of information, the algorithm might look for patterns in the training set to organize the data into subcategories, or infer the distribution of the dataset in the space [46].

Semi-supervised learning: In some cases, obtaining a large enough number of labeled data can cost a lot of time and computing power. A solution to that is to only label a small portion of the dataset and employ a semi-supervised learning algorithm. In this case, the training set is composed of labeled and unlabeled data. The algorithm uses both a supervised learning algorithm on the labeled data of the training set, and an unsupervised learning algorithm on the unlabeled data of the training set, to predict the output of the test set [47].

Reinforcement learning: This algorithm defines the behavior of a software agent. A software agent is a computer program made to execute a task autonomously. This agent has an objective, and each of its action has a certain cost. The algorithm will search for the least expensive sequence of actions to achieve the given objective “through trial-and-error interactions with a dynamic environment” [48]. A concrete example is: there is a car located at a point A . It has to go to a point B . The actions are the motions of the car. The agent slowly goes from point A to point B and repeat the operation several times until it finds the best combination of moves to minimize the distance traveled.

These algorithms can be implemented following different prediction models, for example with decision trees or genetic algorithms. One of the most popular prediction models in recent years is the Deep Learning (DL) algorithm. DL refers to Deep Neural Networks (DNNs), meaning Artificial Neural Networks (ANNs) with a lot of layers. An ANN is a system inspired by biological neural networks that is evaluated on its ability to extract information from its environment and act accordingly. It is composed of at least three layers of “artificial neurons” linked between each other by edges [49]. We define in detail how these elements work in Section 2.1.2. Sections 2.1.1, 2.1.2, 2.1.3 and 2.1.4 describe the different steps of a DL algorithm implementation, each with its own flow of functions.

2.1.1 Perceptron

A single perceptron is said to be the most basic representation of an ANN. Specifically, it is used as a binary classifier [50]. The role of a binary classifier is to separate a dataset into two classes. The perceptron algorithm is a supervised learning algorithm. Therefore its goal is to approximate the function $f : f(X) = y$, that transforms X , the input vector, i.e., an element of the dataset, into y , the value of the class to which this element belongs. The output value, y , is often defined by 0 or 1. The perceptron accepts n inputs. These inputs refer to n characteristics, called features, of a data item, X , in the dataset. For example, in a case where we would be trying to classify flowers, one feature may represent the color. Ideally, these characteristics help differentiate the two classes.

The functioning of a perceptron is derived from biological neural networks. We can compare it to a neuron. If a neuron receives enough signals in a certain amount of time then it fires. A perceptron is similar in a way that it too has a threshold function, called the activation function, to determine in which class the element will be categorized. Figure 2.1 illustrates the behavior of a perceptron. The perceptron input is a vector, X , of n features. This vector is multiplied by a weight vector, W , also of size n . Each value of the weight vector, w_i , defines the importance accorded to an input feature, x_i . After the multiplication, every element obtained is summed and a bias, b , is added. Finally, we transform the acquired value Z with the activation function, σ , to obtain the output O . The bias offers the possibility to shift the activation function along the x-axis. A simple perceptron uses the Heaviside step function as the activation function :

$$\sigma(z) = \begin{cases} 0 & Z < threshold \\ 1 & threshold < Z \end{cases} \quad (2.1)$$

To use this step function one needs to define a threshold. If Z is above this threshold, the current example of the database running through the perceptron, is defined as belonging to the first class. If Z is below, it belongs to the second class.

Figure 2.1 illustrates essentially what is called, in an ANN, the forward propagation. In the case of the perceptron, it represents the transformation of the input vector into the output one by going through a vector multiplication and an activation function [51]. The perceptron also needs a learning algorithm. Every time a data item undergoes the forward propagation transformation, the perceptron computes the output error and update the weights of the perceptron in order to minimize this error. The speed at which the weight vector is updated depends on a variable, α , called the learning rate. The function used to update the weights

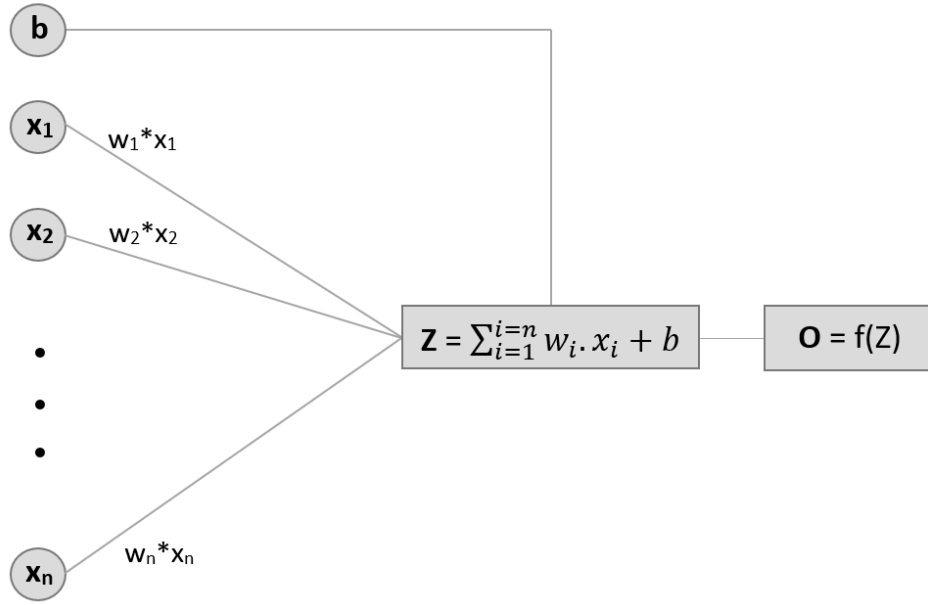


Figure 2.1 Simplified version of a single perceptron

is: $w_i = w_i + \alpha(O_t - O)x_i$.

Unfortunately, a perceptron is limited to linearly solvable problems. For example, it can be configured to represent different logical gates such as OR, AND or NAND, but not the NOR gate. To solve more complex problems we need a Multiple Layer Perceptron (MLP).

2.1.2 Structure of a Multiple Layer Perceptron

A Multiple Layer Perceptron (MLP) is a perceptron composed of at least three layers. An input layer, an output layer and one or more hidden layers. These layers are composed of what we call artificial neurons or, in a simpler way, neurons. Figure 2.2 is referred to as a fully connected neural network because all the neurons of one layer are connected to all the neurons of the next layer. Each neuron of a layer, except for the neurons of the input layer, is defined by its weight and activation function [13].

Same as in the single perceptron, the forward propagation of a MLP corresponds to the transformation of an input vector X into an output vector Y by a succession of matrix multiplications and activation functions [52]. If the algorithm is applied to a classification problem, the output vector represents the probability of each class to be the right one. Here are the two steps happening in the j^{th} neuron of the i^{th} layer during the forward propagation:

1. The first step is the weighted sum of the outputs of the previous layer. If this layer is

the input layer: $z_{i,j} = \sum_{j=1}^n w_{i,j} \times x_j$, or else: $z_{i,j} = \sum_{j=1}^n w_{i,j} \times a_{(i-1),j}$, and;

2. The second step is the application of the activation function that we will call σ_i : $a_{i,j} = \sigma_i(z_{i,j})$,

with x_j being the j^{th} element of the input vector X , $w_{i,j}$ being the weight applied on the j^{th} neuron of the $(i-1)^{th}$ layer, and $a_{(i-1),j}$ being this neuron.

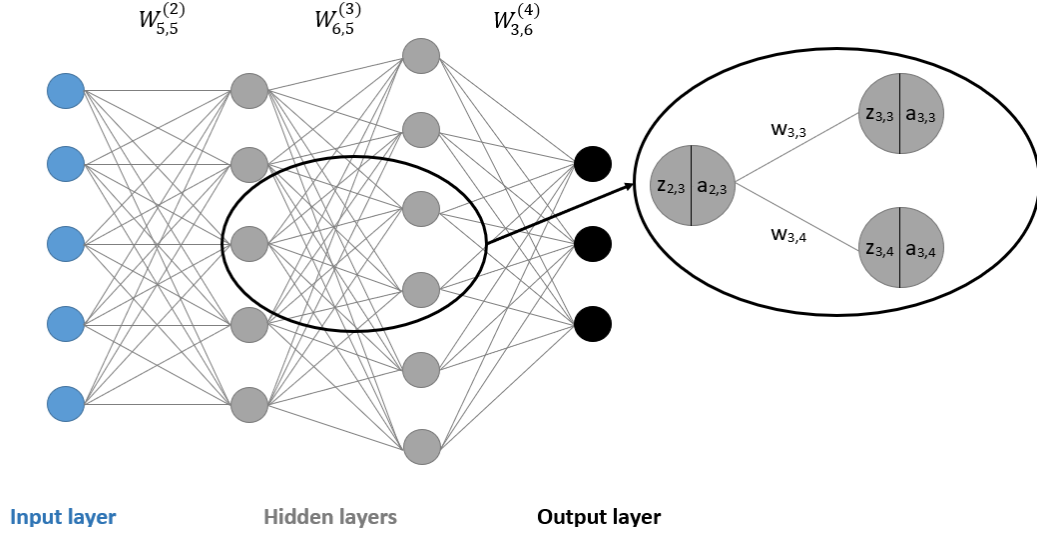


Figure 2.2 A fully connected Neural Network

As the first step happens on all the neurons of a given layer at the same time, its operation can be represented by a matrix multiplication. Each layer i is then linked to a weight matrix $W_{m \times n}^{(i)}$ of size $m \times n$ with m being the size of the i^{th} layer and n the size of the $(i-1)^{th}$ layer. If m is the size of the layer, Z_i and A_i are vectors of the i^{th} layer containing respectively the elements $z_{i,j}$ and $a_{i,j}$, for j ranging from 1 to m . Given that, here is how we can represent the forward propagation of figure 2.2 :

1. $Z_2 = W_{5 \times 5}^{(2)} \times X$
2. $A_2 = \sigma_2(Z_2)$
3. $Z_3 = W_{6 \times 5}^{(3)} \times A_2$
4. $A_3 = \sigma_3(Z_3)$
5. $Z_4 = W_{3 \times 6}^{(4)} \times A_3$

6. $Y = \sigma_4(Z_4)$

The activation function is usually identical for all the neurons of the same layer, but it can change from layer to layer. As for the perceptron, a MLP needs a learning algorithm to update its weights and optimize its future predictions. This algorithm is explained in detail in Section 2.1.4.

The activation function used in the perceptron is the Heaviside function. However with the evolution of DNNs, different activation functions were invented and shown to be very effective [53].

2.1.3 Activation function

The activation function is an essential step of a NN. Without it the input will only undergo a linear transformation which limits its potential. A MLP with non-linear activation functions can classify more complex datasets.

As we see in detail in the subsection 2.1.4, it is important for a NN to have differentiable activation functions to be able to update its weights. The activation functions employed in this thesis are:

- The sigmoid function: It is a non-linear activation function whose outputs range between 0 and 1. Its mathematical formula is:

$$f(x) = \frac{1}{1 + e^{-\beta x}} \quad (2.2)$$

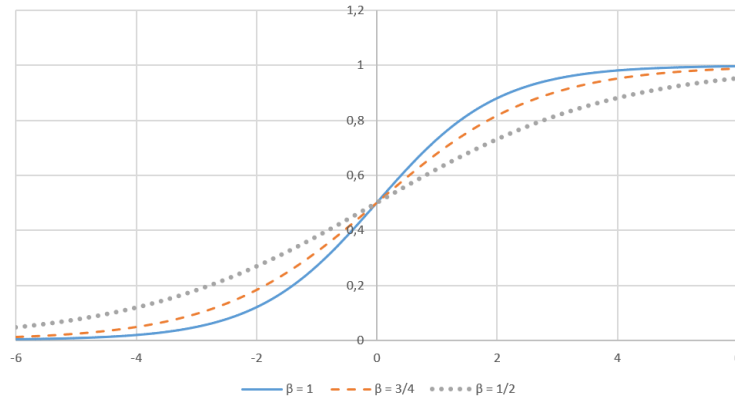


Figure 2.3 Sigmoid function

- The hyperbolic tangent function: This is a variant of the sigmoid function whose output values range from -1 and 1. Its mathematical formula is:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (2.3)$$

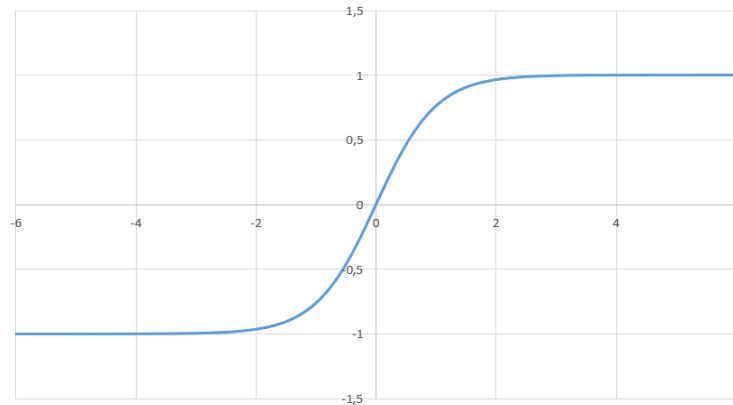


Figure 2.4 Hyperbolic Tangent Function

- The ReLU function: This non-linear activation function is different from the previous one. It is very interesting because it deactivates neurons that have a negative value, those testing if each neuron is essential. Moreover, it lessens the computation cost.

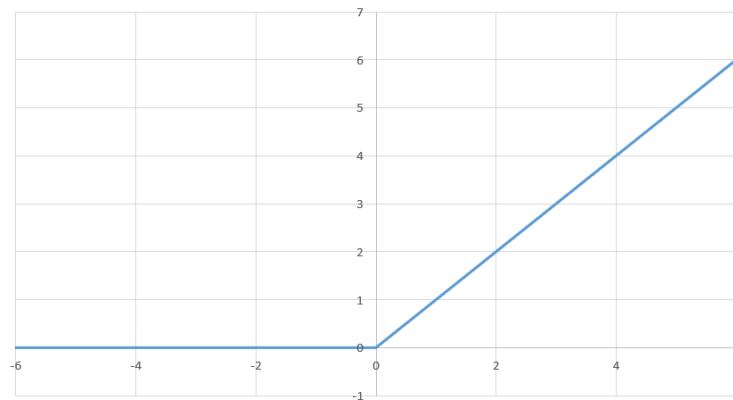


Figure 2.5 ReLU Function

The sigmoid and ReLU functions are two types of activation function appearing very often in DNNs architectures. Variant of them are used a lot too, such as the Leaky ReLU derived from the ReLU, the hyperbolic tangent (tanh) and the hard hyperbolic tangent (hardtanh), both variants of the sigmoid [53].

The advantage of sigmoid-like functions is that they tend to “push” the output value to be either 0 or 1, or -1 and 1, which is useful in a classification problem. However, the two horizontal asymptotes may lead to a vanishing gradient problem. A vanishing gradient problem can be encountered when the model is updated. As described in Section 2.1.4, the learning algorithm uses the derivative of the activation functions to update the weights. We can see in Figure 2.6 that the derivative of the sigmoid function obtains increasingly small values when the input decreases or increases. If the values get too small, the weights cannot be updated. The deeper the network, the greater the chance is of the gradient to vanish [44].

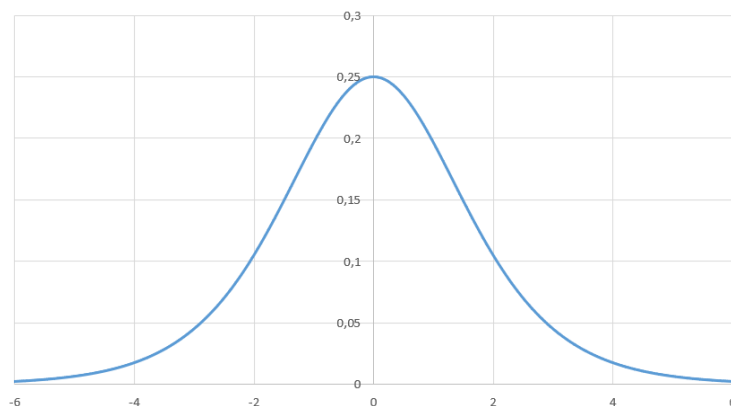


Figure 2.6 Derivative of the Sigmoid Function

ReLU-like functions do not face the problem of vanishing gradient. The derivative is either 0 for $x \leq 0$ or 1 for $x > 0$, meaning the output value does not converge towards zero. It is also a simpler function than the sigmoid which leads to a lesser cost of computing power. However, a simpler function may lead to less accurate learning. Table 2.1 summarizes the pros and cons of using sigmoid-like or ReLU-like activation functions.

Table 2.1 Advantages and drawbacks of activation functions

Activation Function	Advantages	Drawbacks
Sigmoid	• Help to generate probabilities	• Vanishing gradient problem
ReLU	• Low computing power cost • No vanishing gradient	• Less accurate learning

At this point, we have seen the different steps of the forward propagation of an MLP. In the next section, we describe its training algorithm and updating rules.

2.1.4 Backward propagation

When we introduce the functioning of the perceptron, we present a learning algorithm used to update the weights. The role of the backward propagation, or backpropagation [54], is to use the learning algorithm to update the parameters of the perceptron after each forward propagation in order to minimize the output error. The term backpropagation is used in opposition to the forward propagation. The information of the output error travels back through the NN, updating each parameter on its way. As NNs deepen, the weight complexity increases. Here, we introduce a learning algorithm capable of coping with this difficulty, called the gradient descent [55].

The idea is to minimize the error between the obtained output and the expected one by slowly updating the weights. This is similar to the perceptron's learning algorithm. However, to update its weights, the gradient descent algorithm uses the partial derivative of the output error, also called the loss, with respect to each weight. We need to find the value of w_i which minimizes the loss. In Figure 2.7, this value is the abscissa of point C . To get from a random value of w_i , for example the abscissa of point A , to C , we use the slope of the curve. The slope of a curve at a certain point is the derivative of the function at this point. Knowing that, if we subtract the partial derivative $\frac{\delta loss}{\delta w_i}$ to the weight w_i we move slowly from point A to point B . Repeating this operation eventually gets us close enough to the optimal value.

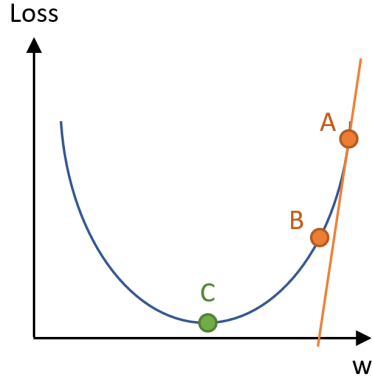


Figure 2.7 Evolution of the loss with the gradient descent method

The equation to update a weight is the following:

$$w_i = w_i - \alpha \frac{\delta loss}{\delta w_i}, \quad (2.4)$$

α is the learning rate, like in the perceptron's algorithm. It defines the speed at which the NN model is updated. It must not be too fast or the training will be unstable, nor too slow

or it will never reach the ideal solution.

Most of the times, to make the algorithm more effective, the input values are normalized. Indeed, those values being different features, they are usually on a complete different scale and this can slow down the training.

Until then, we have been presenting the different step of an MLP, however this is not the only existing architecture used as a DL algorithm.

2.1.5 Other types of Neural Networks

Some neural networks architectures have been designed to optimize their response for a specific case. We present two of them: the Convolutional Neural Network (CNN) and the Recurrent Neural Network (RNN).

Convolutional neural networks (CNN)

This type of NN is mostly used to analyze pictures. Their architecture is composed of fully connected layers, like the one used in a MLP (see Section 2.1.2), placed at the end of the model to classify the dataset. However, a CNN is defined by an addition of two other types of layers capable of extracting more relevant features and at the same time reducing the number of inputs [44]. The two new types of layers used in a CNN are:

1. The convolutional layer: The goal of a convolutional layer is to extract new features, called convolved features, from an input image. The image is represented by a matrix of pixels. A filter, also called kernel, is slid over the input image with a stride of n , meaning that the filter is moved n pixels at a time. At each position, an element-wise matrix multiplication is performed between a subset of the input image and the filter. The elements of the resulting matrix are summed and the output is added to the final matrix of convolved features. Figure 2.8 illustrates how to extract these features. The kernel, in this particular case, is able to highlight the horizontal edges of a picture. Its displacement is executed with a stride of one. The values of a filter can be defined in advance to perform a specific task such as horizontal edge detection, or, in the same way as the weights matrices in classical NNs, can be learned during the training phase.
2. The pooling layer: Its objective is to reduce the size of the array of pixels by only keeping the most important information. The process is analogous to the operation of convolution, and indeed can be seen as a special case of convolution. Usually, a

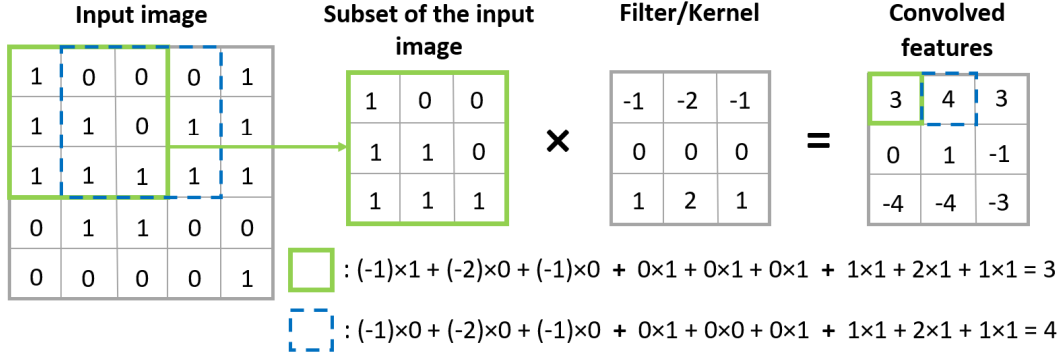


Figure 2.8 Example of a convolution layer for horizontal edge detection

stride of at least two is used to slide the filter over the input image. The higher the stride, the smaller the output will be. The two most common operations of pooling are max-pooling and average-pooling. In the first case, the filter retrieves the pixel of highest value in the image's subset, in the second case it computes the average value of the pixels of the subset. In each case, the output value is transcribed into a matrix which becomes the new input image of the next layer.

A CNN is composed of several alternating layers of convolution, followed by an activation function, and pooling. When the image is considered small enough or the feature extraction seems sufficient, the image is converted to a simple vector to finally be processed and classify by fully connected layers [44].

Recurrent neural networks (RNNs)

RNNs are used to study, translate or understand logical sequences of words, numbers or symbols printed or heard, such as sentences or pieces of music [44]. Each input may be a word or a musical note. When translating a text or a speech, it is necessary to consider each sentence entirely and the context of these sentences. Thus, the idea of an RNN is that at each word of a sentence will be taken into account to process the following one.

Figure 2.8 illustrates the functioning of a basic RNN. Assuming we are performing a translation task of a text of n words, each input x_i is a word and each output o_i is the present translation of the i^{th} first words. The first input is undergoing the same type of transformation we have seen until now : $o_1 = \sigma_1(U_1 \times x_1)$, with σ_1 the activation function of the 1^{st} layer. In Figure 2.9, we can see that each neuron weighs the word preceding the current one to make a prediction. The operations are:

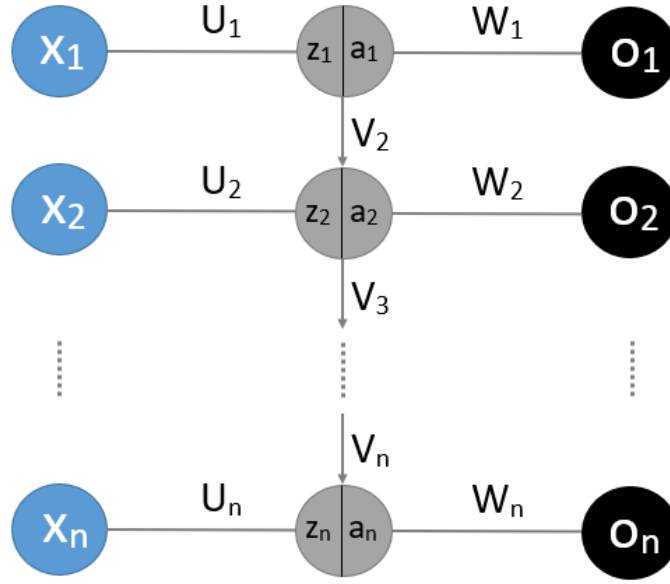


Figure 2.9 Recurrent Neural Network

- $z_i = U_i \times x_i + V_i \times a_{i-1}$
- $a_i = \sigma_i(W_i \times z_i)$

Following this approach, the last output, o_n , of the RNN is the translation of the whole text of n words.

2.2 Photonics

As the goal of this thesis is to build an all-Optical Neural Network (ONN) we need to implement non-linearities, for the activation functions, as well as matrix multiplications, optically. Therefore this section gives a general understanding of photonic circuits and explains the functioning of the optical components used in the simulation of our ONN.

2.2.1 General information

In an optical network, it is the light that is used to transmit information. Instead of using electrical wires, as in electrical devices, the light travels through a waveguide [56].

Optical components can be divided into two main categories [57]:

1. The passive components (e.g., couplers/splitters and filters): They can only apply a

transformation to an input light. They cannot emit light on their own and therefore do not need any power source.

2. The active components (e.g., optical amplifiers and modulators): They need power to execute their function.

In this thesis, we favor the use of passive optical components, as they bring the overall power consumption of the optical circuit closer to zero.

2.2.2 Mach-Zehnder Interferometer (MZI)

A Mach-Zehnder Interferometer (MZI) is an optical device that enables us to modify the amplitude of an input optical wave [56]. Figure 2.12 illustrates the functioning of a MZI. It is composed of a splitter and a combiner (blue circle). The input optical signal is first split into two beams of the same amplitude, then a phase shift is imposed to one of them. This phase shift implies that the signal has been delayed. Finally, the two beams are recombined at the output.

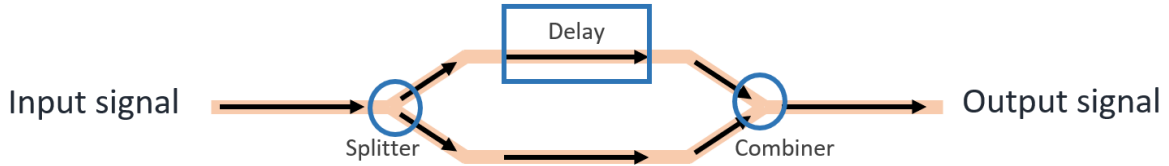


Figure 2.10 Mach-Zehnder Interferometer

Figure 2.11 shows the impact of a phase shift of 1.5 on the sine function. The original amplitude is, therefore, modified in the combiner due to constructive or destructive interference between the two beams of light.

An MZI switch receives two inputs. These two optical signals both get split in two, delayed in one arm of the MZI and recombined [58]. The difference of phase between the two arms, $\delta\Phi$, defines how much power we get in the two output ports [59] :

$$P_{out1} = \sin^2\left(\frac{\delta\Phi}{2}\right); \quad (2.5)$$

$$P_{out2} = \cos^2\left(\frac{\delta\Phi}{2}\right); \quad (2.6)$$

The delay can be fixed, once and for all, by simply using different lengths for each arm of

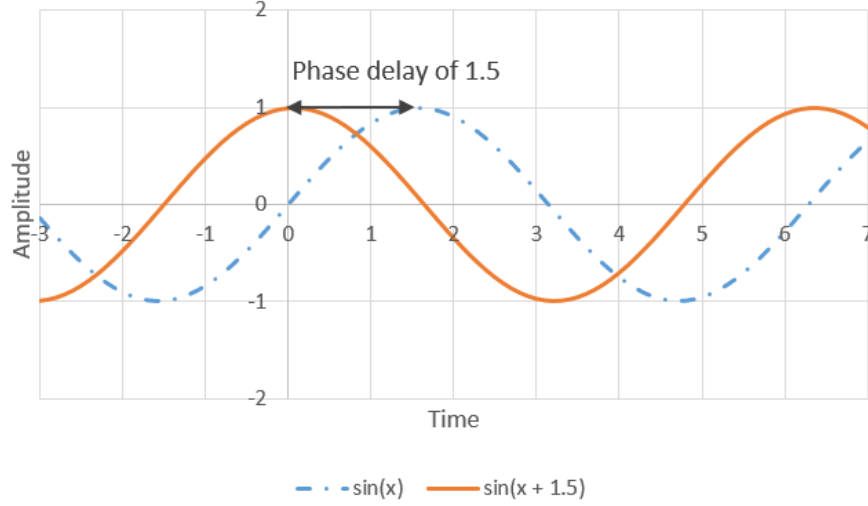


Figure 2.11 Phase shift of the sine function

the MZI. There are also other techniques that allow the modification of the decay after the fabrication of the MZI:

1. Heat variation: if a “temperature difference is applied between the arms,” for example by using a resistive heater on one branch, it influences the propagation speed [57]. Thus delaying one signal compared to the other.
2. Voltage-biased: The existence of an electric field can impact the propagation of the signal in the waveguide. This electric field can be created by putting two electrodes on either side of one arm and delay one signal [60].

The method of application of the delay defines if the MZI is either passive or active. A passive MZI has its arms length defined during design time and never changes, whereas an active one allows the delay to be dynamically tuned during the run-time.

2.2.3 Micro-Ring Resonator (MRR)

A Micro-Ring Resonator (MRR) is used when we need to recover a specific wavelength from an input optical wave [56]. Figure 2.12 shows how an MRR works. The configuration of the loop resonator (the “ring”) defines which part of the optical input is redirected to the output (“Drop”). We can also choose to add another optical signal to the redirected one using the “Add” port. The add port can be used to combine two waves and produce constructive or destructive interferences. The action of redirecting a wavelength through the ring is called coupling.

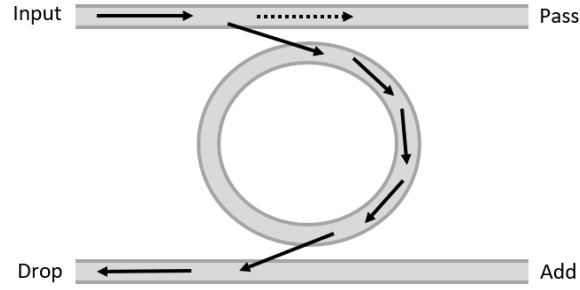


Figure 2.12 Micro-Ring Resonator (MRR)

The selection of the wavelength to redirect depends on the size of the loop. As for the MZI, we can define in advance which wavelength is going to be redirected by fixing the size of the ring. Altering the properties of the waveguide, by heating it or immerse it in an electric field, would allow us to choose another wavelength. Another important characteristic influencing the coupling is the air gap between the ring and the straight waveguide. Usually, the closer the ring is to the waveguide, the easier it is to couple the desired wavelength.

2.2.4 Common concepts

When simulating our ONN, the input features are represented by light beams of different optical power, which corresponds to the energy of light per unit time [61]. MZIs and MRRs are used to transform these values as an ANN would. However, different interactions in the optical circuit might impact its accuracy.

Optical loss

When an optical signal goes through the multiple transformations of an optical circuit, it can be subject to a loss of optical power. The possible causes of losses are the bending of a waveguide, the waveguide itself, the coupling of the input signal into the waveguide and the decoupling of the output signal [56].

The propagation loss in a medium is proportionate to its length. The longer the waveguide, the more losses the circuit may suffer [62]. As a consequence, it is important to limit the size, also called optical depth, of the circuit.

Optical noise

Optical noises are undesired disturbances of the signal in the network. We present here different sources of noise encountered in this thesis.

Photodetector: The use of a photodetector is essential to be able to measure the output optical power. Unfortunately, the interfaces necessary to make the signal detection on the photodetector introduce noises to the network [57].

Crosstalk: This is a phenomenon which refers to a noisy signal interfering with a propagating wave. It happens when a signal is transmitted, or coupled, into a waveguide unintentionally, and, as a consequence produces an unwanted effect in this waveguide [56].

Amplifier: An optical amplifier is a device used to amplify the power. In addition to that, all optical amplifiers add excess noises to the output [63].

In this thesis, we aim to reduce the noises in our ONN, by limiting the implementation of optical amplifiers whenever it is possible. Besides that, we avoid unnecessary or excessive use of signal conversions, via photodetectors, as that would affect the signal quality and hence the model accuracy.

CHAPTER 3 LITERATURE REVIEW

This chapter presents the state-of-the-art research aiming to implement fast ANNs with low energy consumption. It presents the limitations of using electronic computers and promising results concerning ONNs.

3.1 Electronic Implementation of Artificial Neural Networks

In this section, we summarize the various scientific advances that allow the implementation of faster NNs in Subsection 3.1.1. A new type of network is also introduced in Subsection 3.1.2.

3.1.1 Accelerated deep neural networks

With the explosion of Big Data, producing fast and energy efficient architectures for DNNs on chip is a real challenge. Different computing platforms are used to integrate these NNs and optimize the speed and power consumption:

1. FPGA-based neural networks: A FPGA is a chip programmed by the customer, after it has been manufactured [64]. Different FPGA-based accelerators have been introduced in the past years, such as [65] and [66]. A study in [67] showed that FPGA could potentially “achieve more than $10\times$ better speed and energy efficiency than state-of-the-art GPU.”
2. Tensor Processing Unit (TPU): A TPU is a custom Application-Specific Integrated Circuit (ASIC), developed by Google [17]. In one of their studies, Google “found that the TPU delivered $15\text{--}30\times$ higher performances and $30\text{--}80\times$ higher performance-per-watt than contemporary CPUs and GPUs” [68].

3.1.2 Spiking neural networks

Spiking Neural Networks (SNNs) are attractive due to their similarities with biological neural networks. Like them, SNNs process different signals, or spikes of energy. Even further, the time between spikes is taken into account. This means that, a neuron, in addition to reacting according to the intensity of the spike, has more chances of firing back the more it has been triggered in a define period of time. Overall, SNNs allow low power consumption and

parallelism: processing of several data item at the same time [69]. Their biological fidelity gives them the ability to process the complexity of real-time actions [70].

However, they have one major drawback: their activation functions are most of the time non-differentiable [71] which prevents them from using the common methods of learning in ANN. As classic gradient descent cannot be used with SNNs, different alternatives of backpropagation methods have been used, like offline learning [72], [73]. Offline learning means that the learning function is not integrated in the SNN. The idea is to convert the different functions of an ANN to transform it into a SNN. As the ANN chosen has already learned how to perform a certain task, we do not need to use the backpropagation on the SNN. Up to 2018, those SNNs converted directly from existing ANNs gave the best accuracy, with 99.42% [72] and 99.44% [73] on the MNIST dataset [74]. The best result on this dataset with an ANN is 99.79% [75]. The model which achieved this accuracy is composed of a fully connected neural network, a CNN and an RNN. SNNs are getting higher and higher accuracy but still does not reach the results of ANNs.

In 2019, one learning method appeared to be a potential solution to improve the accuracy of SNNs models: the Spike-Timing Dependent Plasticity (STDP). It is originally “a biological process that adjusts the strength of connections between neurons in the brain” [76]. This technique was already used in early paper, such as [77] in 2006. It achieved state-of-the-art result for SNNs in 2019 in [78] with the accuracy of 99.44% on the MNIST dataset.

One issue that remains with the use of SNNs is that, to our knowledge, we do not own any dataset created with the only purpose of being used in SNNs. Most of the time, we convert the dataset used for ANNs to spikes. This is not an optimal solution, as it may cause loss of information.

Several neuromorphic hardwares for SNNs have been produced. They present interesting properties, such as low power consumption and fast inference and are essential in order to generalize the use of SNNs for real-world applications [69]. However, for now, only three platforms seem to “have shared results for classification tasks with the complexity of at least MNIST” [69]: TrueNorth [16], SpiNNaker [79] and BrainScaleS [80]. We present their results in Table 3.1 inspired by [69].

Another neuromorphic chip using SNNs, called Loihi [81], was produced by Intel recently. It is showing promising results. Its energy consumption is only 23.6 pJ per synaptic operations compared to 26 pJ for TrueNorth.

In all the works about SNNs, none seems to have really explored the “real-time action” aspect. Review papers all conclude on the fact that this point would be a major advantage

Table 3.1 Comparison table of SNN chips performance inspired by [69]

Neuromorphic chip	Model	Dataset	Accuracy on test set	Power consumption (per image)
TrueNorth	CNN	MNIST	99.4%	108 μ J
		CIFAR 10	92.7%	0.268 μ J
SpiNNaker	Graphical model	MNIST	95%	3.3 mJ
BrainScaleS	MLP	Downscaled MNIST	95%	7.3 mJ

for SNNs, but it has not been studied enough yet. In [69], they say that the “greatest impact of deep SNNs is expected in the processing of inputs from event-based sensors, because only SNNs are able to fully exploit the precise temporal information such sensors offer” and in [78] that “knowledge representation in time and space makes SNNs unique to perform brain-like computations and to understand the brain data/activity in a spatiotemporal pattern.” These advantages of SNNs could be combined with the architecture of (RNNs) to have a complete contextualization of an environment. It was shown in [73] that RNNs with spiking neurons are able to do probabilistic planning by generating mental plans.

Accelerated DNNs and SNNs both aim to improve the speed and energy efficiency of ANNs. However, with the end of Moore’s law the future of ANNs implemented electronically becomes uncertain, forcing the study of another type of integrated circuit to stay competitive: the analog circuit. In [82], an analog NN is used to classify the dataset MNIST, reaching the accuracy of 95%. Analog SNNs are also emerging. In [83], is proposed a SNN using analog synapses devices which reaches 97.64% accuracy on MNIST. However, even if those architectures are promising, they do not yet perform as well as conventional circuits. In the meantime, the field of silicon photonics opened a new door for ONNs to shine.

3.2 Photonic Implementation for Artificial Neural Networks

Photonic Neural Networks (PNNs) show a great potential regarding the production of fast and energy efficient neural networks [34]. Optical processors can run ten thousand times faster than their electronic counterpart with a potential lower energy consumption [36]. In this section we review the state-of-the-art literature about Photonic Neural Networks (PNNs) and the possibility to produce neuromorphic photonic platforms to run deep learning algorithms.

3.2.1 SOA-based neural networks

In [84], the authors present a PNN architecture based on Semiconductor Optical Amplifiers (SOAs) to classify the Iris dataset. A SOA is an amplifier that uses a semiconductor as a gain medium [85]. As seen in Subsection 2.2.4, it introduces excess noise to the system, and, therefore, may impact the output optical power and at the same time the accuracy. The Iris dataset regroups 150 iris flowers from three species that represent the three classes: setosa, versicolor and virginica. Each flower is defined by four characteristics: sepal length, sepal width, petal length and petal width. The network is a fully connected network composed of an input layer, two hidden layers and an output layer. There are four laser sources emitting four wavelengths to represent the characteristics of the dataset. Every hidden layer possesses four neurons and the output layer possesses three, one for each class. For each input signal to go through each neuron of the first hidden layer, they multiplex them to form a unique input signal, send the signal to each neuron and finally de-multiplex it. Then every input of every neuron is weighted with a SOA. The activation function is the tangent function and is performed in the electronic domain. This means that a photodetector is used to detect the optical power, introducing noise too in addition to the SOA (see Subsection 2.2.4). They reach 95% accuracy on the Iris dataset during their simulation and 85.8% with the photonic integrated chip. The difference of accuracy can be explained by distortions caused by conversions between the electronic and the optical domain [34] and the noise introduced by photodetectors and SOAs.

3.2.2 Coherent nanophotonic circuits

In [30], a new architecture for DL with a coherent nanophotonic circuit is discussed. It is composed of an Optical Interference Unit (OIU) and an Optical Non-linearity Unit (ONU). The OIU is able to realize any matrix multiplication thanks to a combination of 53 MZIs and 213 phase shifters. The ONU, for its part, is supposed to simulate an activation function, but is not yet implemented. The researchers suggest using graphene layers to implement non-linearities. However, for their simulation the non-linearity function is implemented in the electronic domain. They applied their architecture on a vowel recognition task. In [30], they only reached the accuracy of 76.7%, compared to 91.7% on a computer, but, in [86], they achieved the accuracy of 95% on the MNIST dataset [34].

In [87], the model developed is inspired by the one in [30]. They configure two types of fully connected, MZI-based ONNs: GridNet and FFTNet. The first one is a combination of universal unitary multipliers, build with MZIs in a grid-like shape. The FFTNet architecture is more compact and build with less MZIs because a “low number of MZIs leads to reduced

overall loss and noise,” [34]. These two architectures are simulated and tested on the MNIST dataset, assuming the optical components they contain are ideal. GridNet reaches 97.8% accuracy against 94.8% for FFTNet. In a second test, they added imprecision to the simulation that can be caused by the fabrication or the calibration of the optical components. That imprecision is modeled by an independent zero-mean Gaussian noise. During this test, FFTNet shows to be a robust architecture as its accuracy remains constant while GridNet’s accuracy drops below 50%.

These two configurations, [87], [30], do not take into consideration the production of an optical backpropagation. For both of them, the training is done on a computer. In [88] is introduced a way to realize the backpropagation of a ANN with a photonic circuit. They call their procedure the Time-Reversal Inference Method (TRIM). It is an adaptation of the gradient descent methodology for PNNs based on in-situ intensity measurements. This work is a first step towards the integration of deep learning algorithms in a photonic circuit.

Table 3.2 Comparison table of photonic neural networks

Contributions	Architecture	Technology	Dataset	Accuracy ideal - real conditions
[86]	MLP	MZI	MNIST	X - 95%
[87]	MLP	MZI	MNIST	97.8% - 94.8%
[88]	MLP	MZI	XOR gate	100% - X
[84]	MLP	SOAs	Iris dataset	95% - 85.8%

Table 3.2 displays the results obtained by each coherent ONNs presented above. As they are nearly all applied on different datasets, they are difficult to compare between each other. However, we know that the complexity of each dataset is different. A XOR gate is an easy task for a conventional computer reaching every time 100% accuracy. The Iris dataset is more complex but can also be classified with 100% accuracy with a classical NN. The classification of the MNIST dataset, as for it, has reached its highest accuracy in [75]. We observe that, except for the XOR gate, none of the optical applications in ideal conditions reaches a high enough accuracy, and, given the simplicity of the task, we do not consider it a sufficient proof to justify the use of photonic instead of conventional computers. We decide to concentrate on the possible amelioration of the architectures displayed in [86] and [87].

The main issue comes from the noises of the signal detection and its conversion. Indeed, as we are not able to integrate the non-linearity function, this must be done on a conventional computer, which means we need to convert the signal. Converting an optical signal to an electrical, or vice versa, is costly (Subsection 2.2.4) and must be avoided if possible. In [34]

the digital implementation of the non-linear unit is considered partly responsible for the decrease of the accuracy. One solution is to be able to integrate the activation function optically.

3.2.3 Optical activation function

In the different proposition of fully connected neural networks, we have seen none integrate an all-optical activation function. In [30], they propose to implement the non-linearity characteristic of the activation with a saturable absorber. They use the mathematical function of the saturable absorber as their activation function in their ANN model. In [87] they also propose to use a saturable absorber, such as graphene layers, and use the softplus function in their model to approximate it. However, they also announce that softplus's performances do not reach ReLU's.

Some works tried to implement activation functions such as ReLU, sigmoid and their variants [89], [90]: we introduce them in the next paragraphs. Studies of optical non-linearities such as optical bistability in MRRs [91] and the two-photon absorption phenomena [92] are promising but their implementation as an activation function does not exist yet.

In [89], they present the first implementation of an optical sigmoid function. Its solution combines the use of a deeply saturated differentially biased Semiconductor Optical Amplifier (SOA) - MZI with a SOA-Cross-Gain-Modulation (XGM) gate. Again in [90] is presented two optical activation functions with a shape similar to the shape of ReLU and sigmoid. However, as those solutions require the use of state-of-the-art components in silicon photonics, their technology is not as mature as others, such as MZIs and MRRs. They are yet to be integrated in bigger configurations of ONNs to be able to study their scalability. To the best of our knowledge, there is no theoretical study about their feasibility.

3.3 Conclusion

With the end of Moore's law, alternative NNs are emerging, such as SNNs and ONNs. A lot of advances have been made on SNNs, producing neuromorphic chips with a much lower power consumption than their electronic counterpart. Unfortunately, they do not perform as well on classification tasks. ONNs promise low power consumption and high computing speed. However, they are subject to various losses and noises introduced mainly by signal conversions and optical amplifiers that limit their performance and drop the model's accuracy. To avoid this, we propose to simulate an all-ONN. This means that each function of the forward propagation (Subsection 2.1.2) of the NN can be implemented optically, limiting the number

of optical-to-electrical conversion to only one at the output of the network. We also present a solution to avoid using SOAs.

In the following chapters, we present how to integrate Sigmoid and ReLU like activation functions with MZIs and MRRs. We propose a methodology for an optimized implementation of these functions in an ANN in Chapter 4. In Chapter 5, we specify essential characteristics of optical components, and how they can be represented in an ANN, to make the production of an equivalent ONN possible.

CHAPTER 4 METHODOLOGY FOR A DEEP LEARNING MODEL IMPLEMENTATION

In this chapter we present the overall methodology of the project and how we implement our AI model. We follow the same steps as presented in [93], to build a deep neural network for supervised learning in a classification task, leaving aside the business-centered point of view.

Figure 4.1 represents the main steps of the optimization methodology, which are employed in most of the state-of-the-art contributions [94].

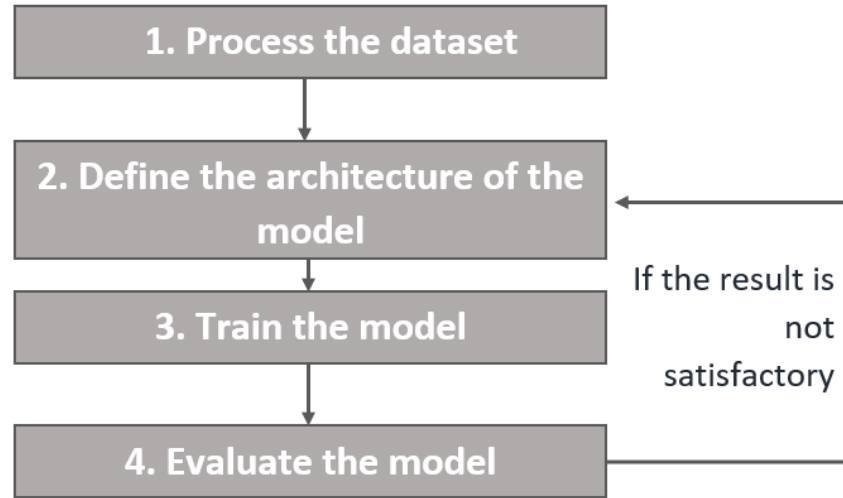


Figure 4.1 Main steps of the optimization methodology

This chapter discusses the NN implementation, without taking into account the optical specificities. Chapter 5 explains in detail how the results of our model can be used to build an all ONN by, for example, implementing restrictions on the parameters of the network and using an activation function feasible optically.

Figure 4.2 is a processing flowchart of our methodology. We use it as a guideline throughout the sections of this Chapter.

4.1 Data Processing

This section introduces the dataset used to build our model and why and how it needs to be preprocessed before training the neural network.

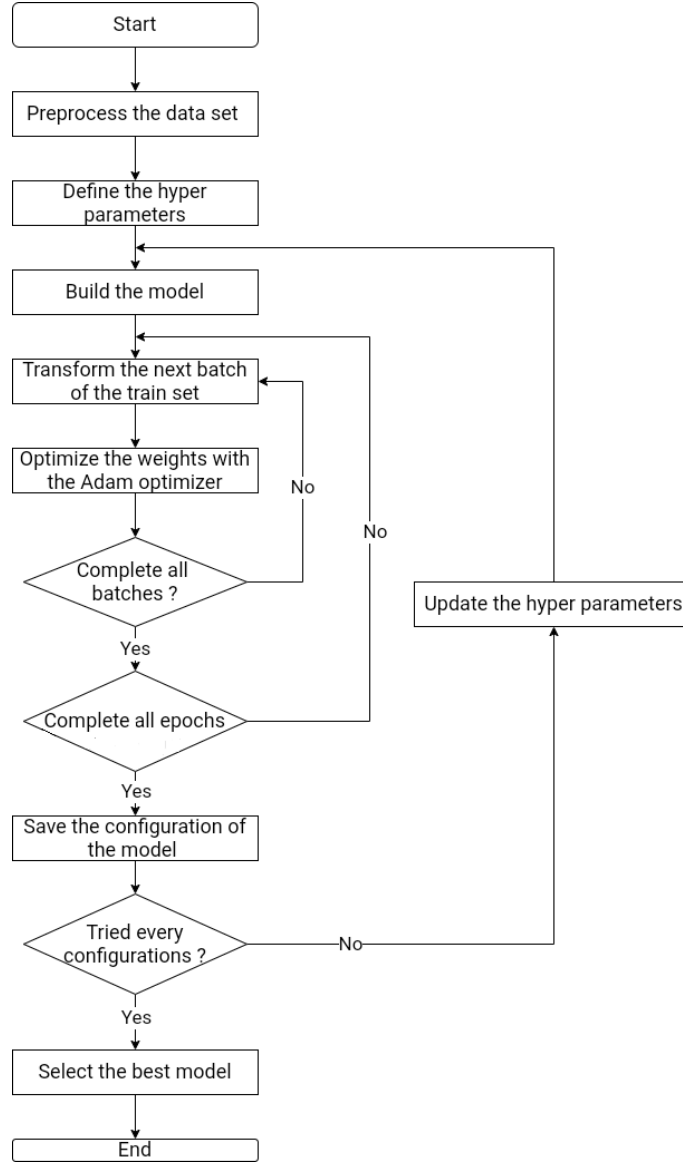


Figure 4.2 Processing flowchart

4.1.1 Choosing the dataset

Following the dataset to process, different architectures of ANNs may be favored, such as RNNs for processing sequential data [44]. The added value of an ONN is to process large databases quickly, potentially consuming less power than a classic computer. Thus, it may be interesting to see how it can perform on a task that needs to be run in real time and manage a large amount of data. In this thesis, we concentrate on the possibility to use an ONN for an image classification task, as they often require a quick response and deal with an ever increasing amount of data.

Usually, an image classification process in ML is dealt with a CNN. It is adapted to the shape of an image which is usually of $N \times N$ pixels. In this thesis, we only assume being able to represent optically a matrix multiplication and a polynomial activation function. A CNN is not reduced to these two operations. More specifically, its pooling layers might need to compute the average, or select the higher value, of a given group of inputs, which cannot be done by a simple matrix multiplication or polynomial function. However, it is still possible to perform an image classification task with a simple MLP by changing the shape of each data item and form a 1-dimensional array of shape $(1, N \times N)$. This operation is called flattening. In order to test the functioning of our network, we first use a small dataset as a crash test. We then select a larger dataset of images to study its scalability. Those are two well-known datasets so as to be able to compare our results with state-of-the-art ones:

1. Fisher’s iris data set [95]: This data set is divided in three classes, which are three types of iris flowers: iris setosa, iris versicolor and iris virginica. It is composed of a total of 150 flowers, 50 of each class, and for each flower 4 features are defined: petal length, petal width, sepal length and sepal width, and;
2. MNIST data set: This data set consists of black and white images of handwritten digits, written by 250 different people. Each image has 28×28 pixels, corresponding to 784 features, and the data set is composed of 60,000 training images and 10,000 test images. MNIST is broadly used to assess the overall performance of neural network architectures. The best results on MNIST were achieved by a CNN with the accuracy of 99.77% [96].

4.1.2 Preprocessing the dataset

Following the choice of the data set, modifications might be needed to adapt it to the input of the AI model and to accelerate its training. The preprocessing phase simplifies the values of the dataset (e.g., conversion from categorical to numerical values with one-hot encoding [97]) and highlights the different features and classes of the neural network.

One-hot encoding

In some data set, there are features defined by categorical values, such as a color or a name. ANNs requires its input to be numerical, meaning that the categorical features need to be converted into numerical values. We do it with a method called one-hot encoding [44]. It consists in decomposing the categorical variable into n binary variables, n being the number

of different values the categorical variable can take. It can also be referred to as a number of categories [98].

An example of this method is presented in Table 4.1. Table 4.1a represents the original data set. The ID is the identification number of a data item and each data item is defined by a color and a price. We want to convert the categorical values of this data set, which are the various colors of the data items. For that we decompose the categorical variable “Color” into three binary columns which are the three categories : red, blue and green, illustrated in Table 4.1b.

Table 4.1 One-hot encoding

(a) Original dataset			(b) One hot encoded dataset				
ID	Color	Price	ID	Red	Blue	Green	Price
1	red	25	1	1	0	0	25
2	blue	10	2	0	1	0	10
3	blue	5	3	0	1	0	5
4	green	30	4	0	0	1	30

Scaling the data

If a model does not seem to give satisfactory results on a dataset, regardless of its architecture, it is a good practice to scale the data before training the model again, as it can help reduce the training time and even obtain a higher accuracy [93]. Two common scaling techniques are:

1. Normalization: rescales the data between 0 and 1, and;
2. Standardization: rescales the data so that they have their mean equal to 0 and a unit standard deviation.

4.2 Architecture of the Model

The next step following the dataset processing is building the DL model. We choose a MLP architecture with one or more hidden layers to begin with. To obtain the best results, there are different parameters, called hyperparameters, that we need to optimize and considered different from the learnable parameters (weights). Here are the different hyperparameters we need to define following each layer :

1. Input layer: number of neurons;
2. Hidden layer(s): number of hidden layers and hidden neurons and their activation function, and;
3. Output layer: number of neurons and its activation function.

Defining the input and output layer is easier as their number of neurons depends essentially on the number of features (input) and number of classes (output) of the dataset. For the output activation function of a classifier, it is important to use a softmax function as it represents the probability distribution over n different classes [44].

For the characteristics of the hidden layers, we use a systematic experimentation to find which values give the best results. The strategy used is called Grid search [99]. It consists in choosing in advance which values we want our hyperparameters to take, test every combinations when running the program and keep the best model. As the number of combinations of hyperparameters is potentially unlimited, conditions are fixed concerning the values they can each take.

Number of hidden layers

It is important to note that too many hidden layers or neurons might lead to the model overfitting and, at the same time, take too much time to compute. However, not enough hidden layers and neurons lead to the model underfitting, and, therefore, a drop in the accuracy [100]. For that it is difficult to choose in advance an appropriate number of hidden layers, N_{hl} , and neurons, N_{hn} , as it varies depending on the dataset [44]. However, in a majority of cases, one or two hidden layers are enough for the classification task [100]. We decide to use up to three hidden layers in our neural network to be sure to obtain the best results, even with more complex datasets such as MNIST.

Number of hidden neurons

To fix the number of hidden neurons, N_{hn} , of a layer, we use two rules that are quite often followed in neural networks:

1. The number of hidden neurons must be between the number of input neurons and output neurons [101], and;
2. As, given the size of the dataset, and as such the number of input neurons, adding one neuron at a time can be very time consuming. To limit the amount of values

to test we use the binary search mode [102], where, the number of hidden neurons: $N_{hn} = [2^n, \text{ for } n \in \mathbb{N}]$.

Activation function

For the activation function, we use polynomial approximations of conventional activation functions (ReLU, Sigmoid and Tanh), that we call: $B_n(ReLU)$, $B_n(Sigmoid)$ or $B_n(Tanh)$, with a polynomial degree $n = [4, 5, 6]$ (see more details in Subsection 5.2.2). One function is selected at a time and used for every hidden layer, to be able to compare their performance.

Table 4.2 resumes the different configurations to test with our model, with N_{hn}^i the number of hidden neurons of the i^{th} hidden layer.

Table 4.2 Domain of definition of the hyperparameters of our model

Hyperparameters	Values to test
Number of hidden layers	$N_{hl} = \{1, 2, 3\}$
Number of hidden neurons	$N_{hn}^i = [2^n, \text{ for } n \in \mathbb{N}] \text{ and } N_{hn}^{i-1} < N_{hn}^i < N_{hn}^{i+1}$
Activation functions	$B_n(ReLU)$, $B_n(Sigmoid)$, $B_n(Tanh)$ for $n = \{4, 5, 6\}$

4.3 Training the Model

This section regroups the different methods of optimization of our model such as choosing the length and speed of the training. It also introduces the different metrics necessary to compare the performance of each combination of hyperparameters.

4.3.1 Optimization

The first step in training a model is to find an algorithm which updates the weights of the system so as to decrease the output loss as much as possible. In our case, we use a stochastic gradient-based optimization, as introduced in Section 2.1.4. The number of times we use the gradient descent is defined by the number of epochs and batches. As presented in Section 4.2, those two variables can be referred to as hyper parameters:

- Batches are subdivisions of the data set. During training, every data item of the same batch must be processed before updating the weights of the model.
- An epoch is completed as soon as all the batches have been processed and the model updated.

To select the number of batches, we decide to test different values with the strategy Grid search, like the previous hyperparameters. For the sake of time optimization, we do not try different number of epochs with Grid search. Instead, before configuring the model as defined in Section 4.2, we use a unique configuration that allows us to select the number of epochs. We choose the configuration that potentially is the longest to train, with the maximum number of hidden layers and hidden neurons, and run it with a high number of epochs. We plot the loss obtained at each epoch and observe when the function converges. We use this value as the number of epochs for each future configuration.

The implementation of the Grid search method with the Keras library [40] uses the k-fold cross validation method to select the best model.

K-fold cross validation

The interest of the cross validation is to be able to select the best configuration for our model by running all of them on a dataset that is neither the train nor test set [103]. This third set is called the validation set and is formed from elements of the train set. In the k-fold cross validation algorithm, the train set is divided into k subsets and k models are built. Each model uses one subset (different for each model) as a validation set and the others as $k - 1$ train sets. Finally, the result kept is the highest accuracy obtained on one of the validation sets.

Other methods can be used to accelerate the optimization of a neural network, such as momentum [104] and adaptive learning rate, which we introduce below.

Momentum

The principle of the momentum, m , appears when updating the weights of the model (see equation 2.4). Each time the weights are updated, the previous weight increment, δW_{t-1} , is taken into account. As a result, the new formula to update the weights is:

$$W_{ij}^{t+1} = W_{ij}^t - [\delta W_{ij}^t + m \times \delta W_{ij}^{t-1}] \quad (4.1)$$

with,

$$\delta W_{ij}^t = \alpha \frac{\delta loss}{\delta W_{ij}^t}. \quad (4.2)$$

This means that, over time, it accumulates a decaying percentage of past gradients and continue to move in their directions. This can drastically increase the speed of training of the model [44]. However, this technique has one disadvantage. If the momentum increases

too much, it could miss a local minimum of the loss function. To avoid this problem, an adaptive learning rate can be used.

Adaptive rate

The notion of adaptive learning rate means that the learning rate evolves over time [105]. Usually, the learning rate decreases with time, allowing the learning algorithm to slow down when it potentially gets closer to a local minimum of the loss function, preventing the algorithm from missing it [106].

The adaptive learning rate, when combined with momentum, can help the algorithm from missing a local minimum. Different optimizers combine this two methods, or similar ones, to train models, such as RMSProp [107], AdaGrad [108] or Adam [109]. We decide to use Adam to optimize our model as it shows good results for MLPs in term of learning speed and accuracy [109].

4.3.2 Evaluation metrics

As the optimum architecture of our model is not known in advance, we have to optimize various models and compare them between each other. To decide which configuration to keep or not we use two standard evaluation metrics: the accuracy and the loss of the system. Once we decide the configuration of our model with the method Grid search, we train and test it several times to study its stability. In this thesis, we use the term stability to define the capacity of a model to produce similar results every time it is trained. To select the best configuration during the training, we use a validation set, a subset of the train set. Each epoch, the accuracy and loss are computed over the validation set. If the accuracy increases, the model is kept in memory. At the end of the training, the accuracy and loss of the test set are computed with the model giving the best accuracy on the validation set.

Loss

The loss is used in the optimization process to update the weights. To compute the loss we use the categorical cross-entropy as it “is often used for classification problems when outputs are interpreted as probabilities of membership in an indicated class” [110]. The categorical cross-entropy is the combination of the softmax function and the cross-entropy loss :

$$Cross-Entropy = \sum_{x \in C} y_{i,x} \times \log(p_{i,x}) \quad (4.3)$$

The cross-entropy loss is summed over the number of classes, C , of the data set. $y_{i,x}$ is the correct prediction of the data item i for the class x and $p_{i,x}$ is the output prediction of our model, and of the softmax layer, of the data item i the x^{th} class.

It is interesting to plot the loss of the train and validation set over time to prevent overfitting or underfitting. If the loss of the train set stagnates or is still decreasing when the algorithm stops, it means that we are in a case of underfitting (Figure 4.3). If the loss of both sets decreases and then at some point, the validation loss increases again it means the model is overfitting (Figure 4.4). In some cases, it is possible to reduce the issue of overfitting by using a method called dropout [44]. Dropout consists in “removing” a given percentage of the weights of the networks temporary. The weights dropped are chosen randomly and change at each epoch. A dropout of 0.2 means that at each epoch, 20% of the weight is chosen randomly to be dropped.

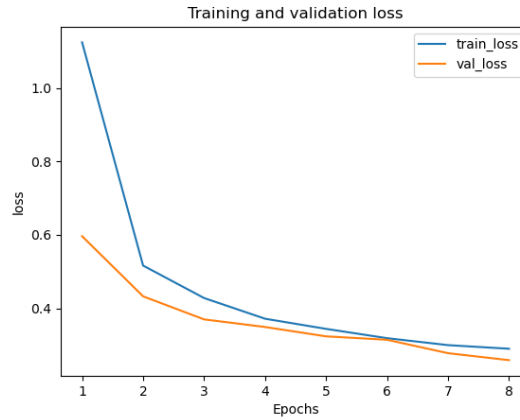


Figure 4.3 Case of underfitting

Accuracy

The accuracy indicates the number of correct predictions of the model in contrast to the total number of predictions:

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}. \quad (4.4)$$

For example, with a dataset of 150 examples, if 120 data items are correctly predicted, the accuracy is 0.8. We can also say that we obtained 80% accuracy on the dataset. The accuracy obtained on the test set represents how well the model would perform on an unknown dataset. Unlike the loss function, it is a binary metric. It takes into account the number of

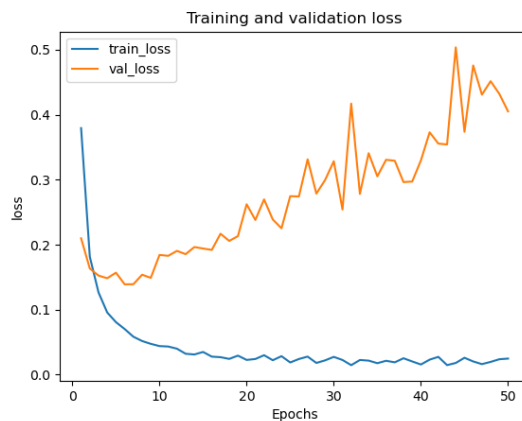


Figure 4.4 Case of overfitting

good predictions and does not indicate how close the predictions of the model are from their expected value.

4.4 Summary

The processing flowchart, illustrated in Figure 4.2, resumes our methodology from the data preprocessing to the selection of our model. The first step of our methodology is to preprocess the dataset, prepare the data so that it can be used by our neural network (one-hot encoding). Then we select a base model from which we can define our hyperparameters and configure our first model. The algorithm transforms each batch of the train set, one at a time, and optimize the weights with the Adam optimizer. This represents one epoch. Once every epoch has been completed, or the early stopping condition has been reached, we save the configuration of the model selected as well as its accuracy. We repeat this step for every predefined configuration of our model and finally compare all the results to select the best model.

CHAPTER 5 OPTICAL NEURAL NETWORK MODEL

In this chapter, we explain the particularities of an ONN and how we can use them to implement the different steps of a DL model. Our DL model is built following the methodology presented in Chapter 4 and taking into account the specificities of an ONN so that once trained, the selected architecture can be built as an optical circuit. The major modifications, compared with a classic NN, concern the architecture of the model.

5.1 Data Processing

In an ONN, the value of the features of each data item is represented by a light intensity. As this can never take negative values, we rescale our dataset with the normalization technique so that each data value is between 0 and 1 (see Subsection 4.1.2). The usual formula employed for the normalization is:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}. \quad (5.1)$$

It is interesting to note that using normalization is not always feasible. Indeed, if the dataset's distribution is not similar to a normal distribution it will not be possible to rescale it uniformly and we might lose critical information. Table 5.1, illustrates this issue. It shows the monthly wage of 19 people, and the values obtained after normalization. The fact that the salary of some individuals is much higher than most of the population leads us to obtain a majority of normalized value very close to zero. The Iris and MNIST datasets both have features bound between a lower and higher value, limiting this effect.

Table 5.1 Monthly wage of 19 people, before and after normalization

Monthly wages	1500	2000	1750	10000	2200	1350	1200	15000
Monthly wages normalized	0,069	0,10	0,086	0,66	0,12	0,059	0,048	1
Monthly wages	500	2000	1800	1850	1600	2150	2100	1670
Monthly wages normalized	0	0,10	0,090	0,093	0,076	0,11	0,11	0,081
Monthly wages	11000	1000	1900					
Monthly wages normalized	0,72	0,034	0,097					

5.2 Model's Architecture

This section explains how we can produce the main steps of a neural network optically: (1) the matrix multiplication, and; (2) the activation function.

5.2.1 Optical matrix multiplication

As presented earlier, an ONN can be produced using meshes of MZIs [30]. These meshes are able to perform any unitary matrix transformation between a set of optical channels if designed properly. There are two methods proposed in the state-of-the-art to choose the parameters of the MZIs, we call them Reck's method [111] and Clement's method [35] following the name of their first author. Clement's method produces a mesh of MZIs with a shorter optical depth than Reck's method, implying that it suffers less propagation loss [35]. Therefore, in this section, we introduce the decomposition technique to build optical matrices proposed in [35].

Clement's method

In [35], the authors explain how to configure a layout of MZIs and phase shifters in order to perform any unitary transformation U . Here is how is defined the unitary group of degree N , $U(N)$:

$$U(N) = [For\ i, j \in [1, N], \exists a_{i,j}, b_{i,j} \in \mathbb{R},\ such\ that$$

$$A = \begin{pmatrix} a_{1,1} + jb_{1,1} & \dots & a_{1,n} + jb_{1,n} & \dots & a_{1,N} + jb_{1,N} \\ \vdots & \ddots & & & \vdots \\ a_{m,1} + jb_{m,1} & & a_{m,n} + jb_{m,n} & & a_{m,N} + jb_{m,N} \\ \vdots & & & \ddots & \vdots \\ a_{N,1} + jb_{N,1} & \dots & a_{N,n} + jb_{N,n} & \dots & a_{N,N} + jb_{N,N} \end{pmatrix} \quad (5.2)$$

$$and\ AA^H = I]$$

Any matrix of the unitary group $U(2)$ can be produced with:

- one MZI, with 50:50 beam splitters and a phase shift $2 \times \theta$, and;
- one phase shifter, placed on one of the input branches of the MZI, with a phase shift ϕ .

Figure 5.1 illustrates the transformation of two optical inputs of light intensity, z_1 and z_2 , into z'_1 and z'_2 , with this architecture. The corresponding formula is:

$$\begin{pmatrix} z'_1 \\ z'_2 \end{pmatrix} = \begin{pmatrix} e^{j\phi} \cos \theta & -\sin \theta \\ e^{j\phi} \sin \theta & \cos \theta \end{pmatrix} \times \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}. \quad (5.3)$$

To reproduce any unitary matrix of size $N \times N$, we need to combine $\frac{N \times (N-1)}{2}$ MZIs. Figure 5.2 shows a schematic illustration of a mesh of 36 MZIs representing a unitary matrix of size 9×9 .

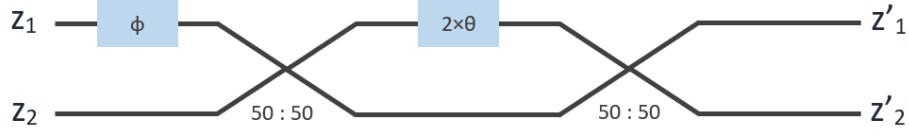


Figure 5.1 Optical representation of a matrix multiplication using an MZI

Figure 5.1 Scheme of a MZI with a phase shift of $2 \times \theta$ and 50:50 beam splitters crossed by two optical waveguides (black lines). A phase shifter, with a phase shift ϕ , is placed on one of the input branches of the MZI.

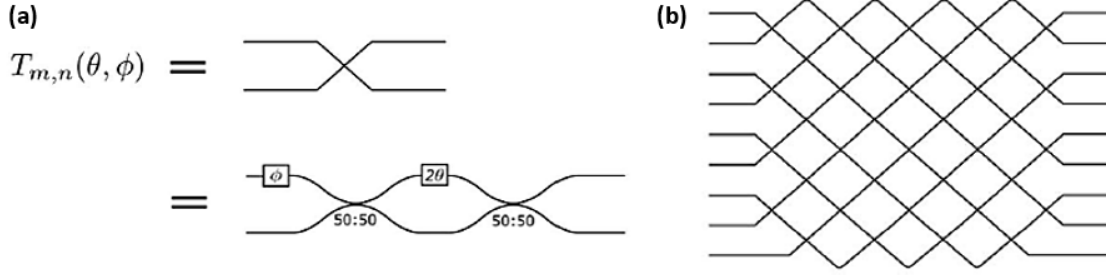


Figure 5.2 Schematic representation of a MZI (a) and a mesh of MZIs (b) from Clement's paper [35]

Given a matrix $U \in U(N)$ that we want to perform optically, the goal is to multiply U by a sequence of matrices, $T_{m,n}(\theta, \phi)$ and $T_{m,n}^{-1}(\theta, \phi)$, so that, at the end, all of the elements of the matrix U equal zero, except for the ones of the diagonal. These $T_{m,n}(\theta, \phi)$ matrices, or $T_{m,n}$ to lighten the expression, correspond to the $U(2)$ transformation seen above (Equation 5.3). They have the same size as U and are similar to an identity matrix except for the fact that their elements $t_{m,m}$, $t_{m,n}$, $t_{n,m}$ and $t_{n,n}$ are respectively equal to $e^{j\phi} \cos \theta$, $-\sin \theta$, $e^{j\phi} \sin \theta$ and

$\cos \theta$, with $t_{m,n}$ being the element of the m^{th} row in the n^{th} column of the matrix $T_{m,n}$:

$$T_{m,n} = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & & & & & & \vdots \\ \vdots & & \ddots & & & & & \vdots \\ \vdots & & & e^{j\phi} \cos \theta & -\sin \theta & & & \vdots \\ \vdots & & & e^{j\phi} \sin \theta & \cos \theta & & & \vdots \\ \vdots & & & & & \ddots & & \vdots \\ \vdots & & & & & & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & 1 \end{pmatrix}. \quad (5.4)$$

A triangular unitary matrix is a diagonal matrix. Therefore nulling half of the matrix is enough to obtain a diagonal one. We proceed by successively nulling the consecutives diagonals of the lower half of U . Each element of a diagonal, i , is nulled by multiplying U by a $T_{m,n}(\theta, \phi)$ matrix and choosing θ and ϕ properly. If i is odd and the element to null is located on the j^{th} column, U is multiplied on the right by $T_{j,j+1}^{-1}$. If i is even and the element to null is located on the j^{th} row, U is multiplied on the left by $T_{j-1,j}$:

$$\left(\prod_{i=M}^1 \prod_{j=0}^{2i-1} T_{N-j-1, N-j} \right) \times U \times \left(\prod_{i=1}^{M'} \prod_{j=0}^{2(i-1)} T_{2i-1-j, 2i-j}^{-1} \right) = D \quad (5.5)$$

If N is even, $M = \frac{N}{2} - 1$ and $M' = \frac{N}{2}$. If N is odd, $M = M' = \frac{N-1}{2}$. We can determine all the parameters θ_i and ϕ_i of the optical circuit like this. All the steps of the method are presented in Appendix A.

Realization of an optical 3×3 unitary matrix with Clement's method

We explain here how to represent a 3×3 unitary matrix, U , with optical components, following the methodology we just explained. We express U as:

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ u_{3,1} & u_{3,2} & u_{3,3} \end{pmatrix} \quad (5.6)$$

The first step is to null the first diagonal containing the single element: $u_{3,1}$. To do that we multiply U on the right by the matrix $T_{1,2}^{-1}(\theta_1, \phi_1)$ and select θ_1 and ϕ_1 so that the element

$\hat{u}_{3,1}$ of the obtained matrix $\hat{U}$ equals zero (see Appendix A):

$$\hat{U} = U \times T_{1,2}^{-1}(\theta_1, \phi_1) = \begin{pmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{pmatrix} \quad (5.7)$$

At the end of this step, we can begin to build our optical model. We represent a three by three matrix multiplication so we need three input waveguides and three output waveguides. The MZI represented by the expression $T_{1,2}^{-1}(\theta_1, \phi_1)$ is placed on the left of the circuit in order to combine the input waveguides 1 and 2 (blue square on Figure 5.3).

For the second step we null the second diagonal of $\hat{U}$ containing the elements: $\hat{u}_{2,1}$ and $\hat{u}_{3,2}$. This time we multiply $\hat{U}$ on the left by $T_{1,2}(\theta_2, \phi_2)$ and then $T_{2,3}(\theta_3, \phi_3)$:

$$T_{2,3}(\theta_3, \phi_3) \times T_{1,2}(\theta_2, \phi_2) \times \hat{U} = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} \quad (5.8)$$

We continue to build our optical circuit. First by adding the MZI, $T_{1,2}(\theta_2, \phi_2)$, on the output waveguides 1 and 2 (green square on Figure 5.3), and secondly by inserting the MZI, $T_{2,3}(\theta_3, \phi_3)$, before $T_{1,2}(\theta_2, \phi_2)$, on the two output waveguides 2 and 3 (orange square on Figure 5.3).

After Equation 5.8, we obtain a upper triangular matrix. As U is unitary, the obtained matrix is therefore a diagonal matrix called D . Finally, we can summarize the succession of multiplications as follows:

$$T_{2,3}(\theta_3, \phi_3) \times T_{1,2}(\theta_2, \phi_2) \times U \times T_{1,2}^{-1}(\theta_1, \phi_1) = D \quad (5.9)$$

This result could be obtained directly with the Equation 5.5 we propose here. Following [35], we can simplify this equation such as:

$$U = D' \times T_{1,2}(\theta_2, \phi_2) \times T_{2,3}(\theta_3, \phi_3) \times T_{1,2}(\theta_1, \phi_1) \quad (5.10)$$

The successive $T_{m,n}$ matrices can be represented directly by a succession of MZIs as in Figure 5.3. The elements of the resulting diagonal matrix D' have a modulus equal to one [35]. In our AI model, we use real matrices. The decomposition of matrices with real numbers by the Clement's method results in a combination of matrices with real numbers too. If D is real then the elements of its diagonal are all equal to 1 or -1 . This can be represented by

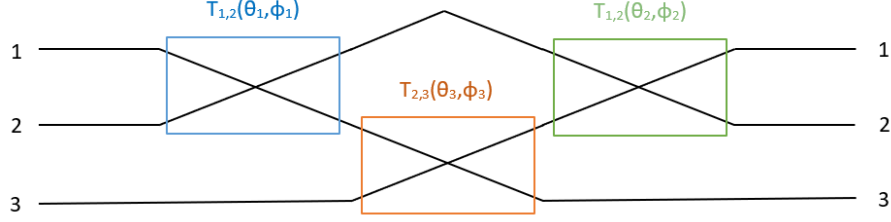


Figure 5.3 Mesh of MZIs to realize a 3×3 unitary matrix

adding a phase shifters on the output of the interferometers [35].

Clement's method allows us to produce any unitary matrix $U(N)$ optically. However, in a neural network the matrix multiplication is not necessarily unitary.

Singular value decomposition

The singular value decomposition states that any real matrix M can be factorized in the form $U\Delta V^T$, so that $M = U\Delta V^T$, with U and V^T being two real unitary matrices and Δ a diagonal matrix of non-negative real numbers [112]. Both U and V^T can be produced optically thanks to Clement's method. Δ can be decomposed into $\beta \times \Delta'$, with β being a compensating scaling factor [87] and Δ' a diagonal matrix with the modulus of its elements below 1. The matrix Δ' can be modeled with phase shifters. For β it is necessary to implement it with a semiconductor optical amplifiers [85] as MZIs and phase shifters can only reduce the input signal.

5.2.2 Optical activation function

A major drawback with ONNs is that an all-optical activation function, designed with MZIs and MRRs, has yet to be implemented. For now, the activation function is usually processed on a computer. It needs a conversion of signals between the photonic and electronic field. This not only slows down the system but also introduces noises. To solve this problem we propose in this section an all-optical activation function and how to implement it optically.

Our choice is to concentrate on efficient activation functions thanks to polynomial approximation. The Weierstrass approximation theorem states that for any real and continuous function f on the interval $[a; b]$, we can find a sequence of real polynomials that converge uniformly to f [113].

In [114], they achieve the production of an all-optical Bernstein polynomial function, with a combination of MRRs and MZIs. They are inspired by the design of stochastic computing

architectures, more specifically the ReSC architecture [115].

In the case of the Bernstein polynomial, it has been proven that it is possible to find a sequence of Bernstein polynomials to fit any real function continuous in the interval $[0; 1]$ [113]. The formula to approximate a function f by a Bernstein polynomial of degree n is:

$$B_n(f) = \sum_{k=0}^n f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k} \quad (5.11)$$

The element of degree j of a Bernstein polynomial can be expressed as follows:

$$B_{k,j}(x) = \binom{j}{k} x^k (1-x)^{j-k} \quad (5.12)$$

The Bernstein approximation limits the use of an efficient polynomial activation function between 0 and 1. This is not usually the case in ANNs. Activation functions are designed to normalize an input that potentially belongs to the entire set of real numbers, and produce an output that is usually between 0 and 1 (Sigmoid and ReLU), which is very useful when we need to produce a probability, or -1 and 1 (Tanh).

To study the relevance of an optical polynomial activation function, we implement the Bernstein approximation of several existing activation functions, for a degree $n = [4, 5, 6]$ (6 being the highest degree of the polynomial studied in [114]), in our neural network. We decide to study the implementation of the three most common activation functions: Sigmoid, ReLU and Tanh.

The sigmoid function is continuous on the interval $[0, 1]$ and thus can be approximated by a Bernstein polynomial of degree n on the same interval. On Figure 5.4, we can see that the sigmoid, the continuous line, possesses two horizontal asymptotes $x = 0$ and $x = 1$ that are harder to reproduce by the polynomial approximation. The Bernstein sigmoid seems unlikely to perform as well as the sigmoid if the input gets too big or too small. However, it is interesting to see how it can behave when fed inputs of small value (in the interval $[-3; 3]$).

For a given x , the ReLU function gives us the maximum value between 0 and x , meaning that on the interval $[0, 1]$ the function can be expressed as $f(x) = x$ and that is the function that the Bernstein polynomial will approximate. To get around this problem, we use a variant of the ReLU called the smooth ReLU or Softplus:

$$Softplus = \ln(1 + e^x) \quad (5.13)$$

Even if ReLU is still one of the most used activation functions, Softplus may give better result

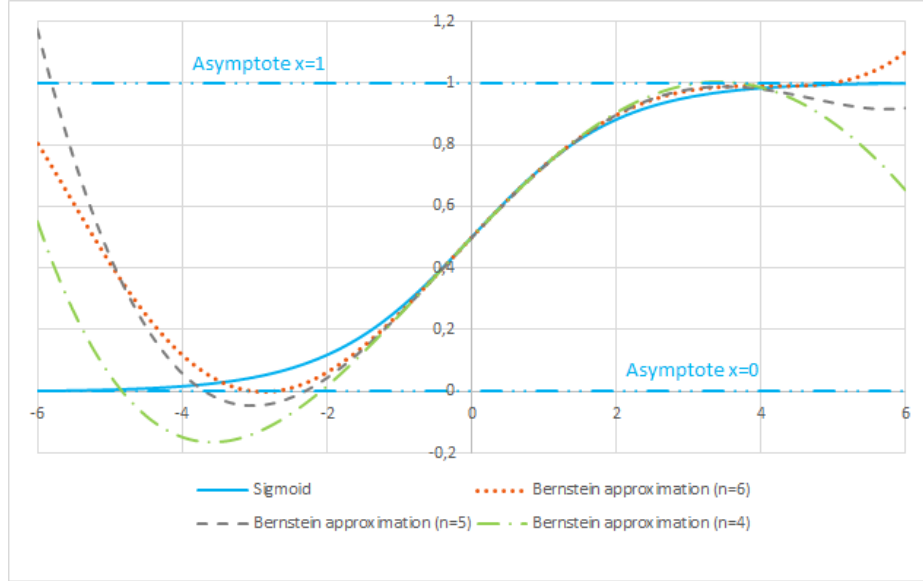


Figure 5.4 Sigmoid activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$

in some applications [116]. In Figure 5.5, we can observe that the Softplus approximations' curve by the Bernstein polynomial has a small plateau at 0 around $x = -3$ (zone indicated by a circle in the figure), that is similar to the curve of ReLU when $x \leq 0$. Overall, in the interval $[-4, 6]$, the different approximations seem to fit quite well the Softplus' curve.

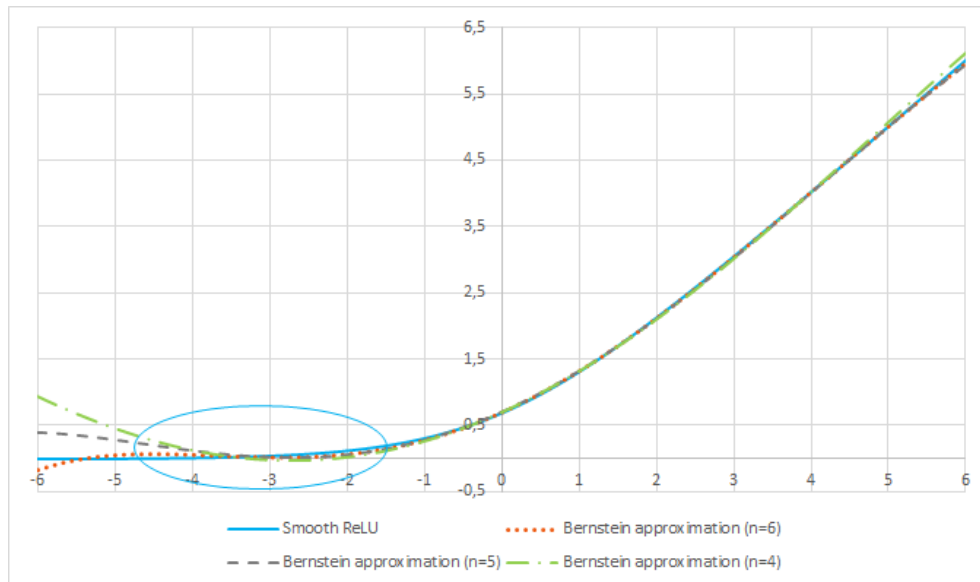


Figure 5.5 Smooth ReLU activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$

The last function we approximate is the hyperbolic tangent, Tanh. This allows us to try another sigmoid-like function that produces output in the interval $] - 1; 1[$. As for the sigmoid approximations, the Bernstein approximations of Tanh, represented in Figure 5.6, might output values too high when $x \in] - \infty; -2] \cup [4; +\infty[$. As for the sigmoid function, this is due to the fact that we cannot represent the asymptotes of Tanh, $x = 1$ and $x = -1$, properly with a polynomial function.

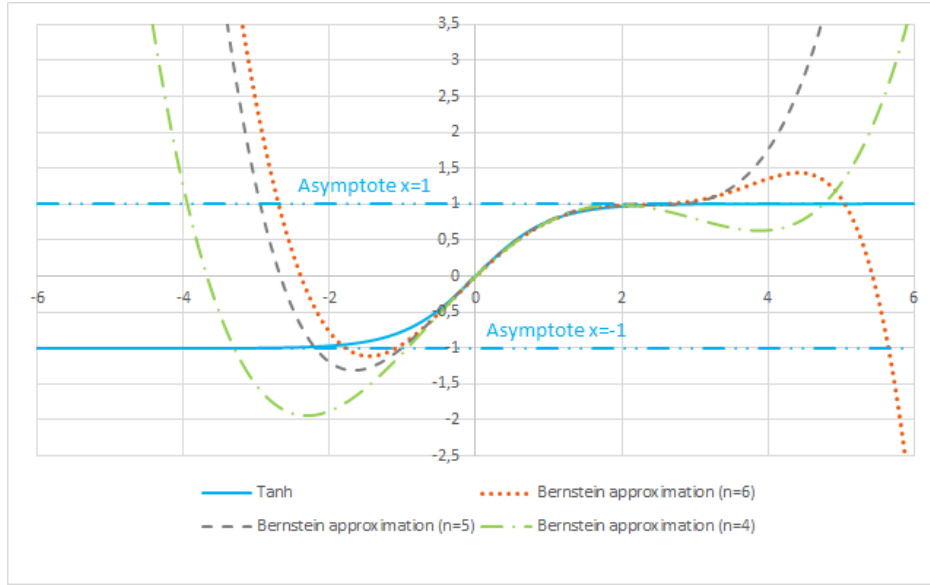


Figure 5.6 Hyperbolic tangent activation function and its approximation by the Bernstein polynomials for $n = 4$, $n = 5$ and $n = 6$

The main issue with the Bernstein approximations are their possibility to produce high values outside the intervals $[0; 1]$ and $[-1; 1]$, which are the characteristic output ranges of the activation functions. The neural network can be configured in a way to limit this effect by reducing the input value of the activation functions, h . h is defined by the data item processed by the neural network, x_i , the number of features, N , and the weights, w_i ,

$$h = \sum_{i=1}^N w_i x_i. \quad (5.14)$$

The normalization of the dataset already allows to reduce the value of h . The weights of the neural networks may also be constrained (see Subsection 5.3) to diminish the output result. We decide to test these two types of functions with and without a weight constraint of 1, $WC = 1$, on our more complex dataset MNIST. We build a model with one hidden layer of 120 neurons and normalize the input between 0 and 1. We display the results of the ReLU transformation, in Figure 5.7, and the Sigmoid transformation, in Figure 5.8, of each neuron

of the hidden layer for three elements of the dataset. We observe both for the Sigmoid and ReLU that the WC decreases a bit the divergence of the values. However it seems more efficient in the case of the ReLU.

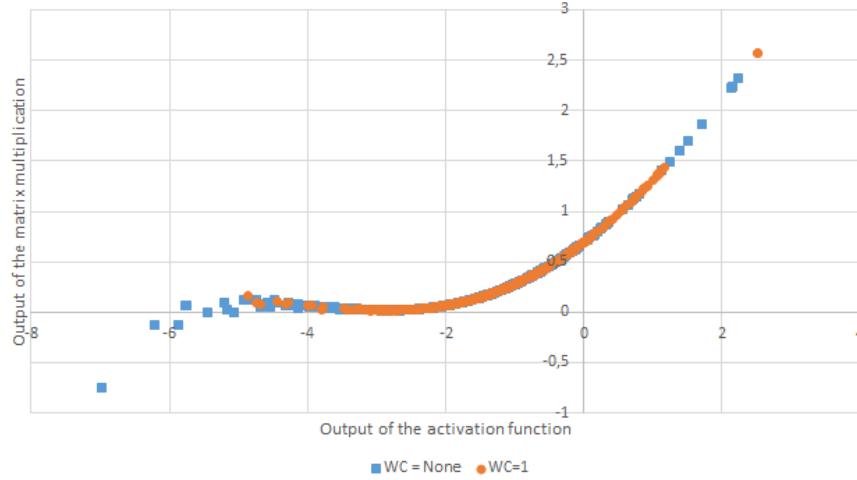


Figure 5.7 Output values of $B_6(ReLU)$ with and without $WC = 1$

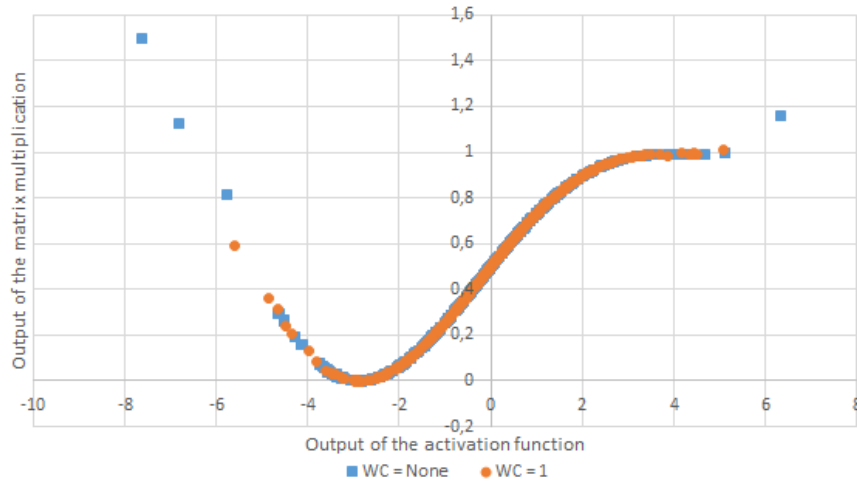


Figure 5.8 Output values of $B_6(Sigmoid)$ with and without $WC = 1$

Output activation function

Usually in a MLP, the output activation function is a softmax function. This function's purpose is to give for a data item its probability of belonging to each class. It cannot be approximated by a polynomial. During the test phase, it is used only once at the end of

the network. Therefore, it can be implemented after the photo-detection, on a computer, without adding losses to the system. However this also means that the input of the softmax function is not directly the light intensity but a measurement of this power.

The power measurement by a photodetector can be expressed as $|z|^2$, with z being the output light intensity of the ONN [117]. We can simulate this measurement in our ANN by adding an activation function $f(x) = |x|^2$. Also if x is a real number, we can simplify the expression to a square function: $f(x) = x^2$. We do not need to use a Bernstein approximation for either the softmax function and the squared function as they are performed electronically.

5.3 Training the Model

This section presents an optimization method that can help stabilize the training process of our model when we use polynomial activation functions.

We observed in Subsection 5.2.2 that having weights of value too high could cause an issue when using polynomial activation functions. A solution is to constrain the norm of the weights, w_i , to be inferior to a constant, c , when they are being updated [118], such that $\|w_i\|_2 \leq c$. It is important to note that constraining the norm of the weights to a value of one would allow us to perform an optical matrix multiplication without SOAs. We decide not to use this method initially and see during the test if its implementation is feasible or even necessary.

5.4 Summary

To build an optical circuit from an AI model, we need to implement directly on our model some of its specification. Figure 5.9 visualizes the design flow of a potential ONN (in white) in parallel with the AI model we use (in blue). The dataset processed must be normalized to have positive values so that they can be represented by a light intensity. The weight matrices of the system can directly be implemented thanks to Clement's method described in the Subsection 5.2.1. The activation functions are Bernstein approximations of common activation function (ReLU, Sigmoid and Tanh) which can be implemented later on with MRRs and MZIs [114]. Finally, the effect of the photodetector can be simulated in our network using a square function.

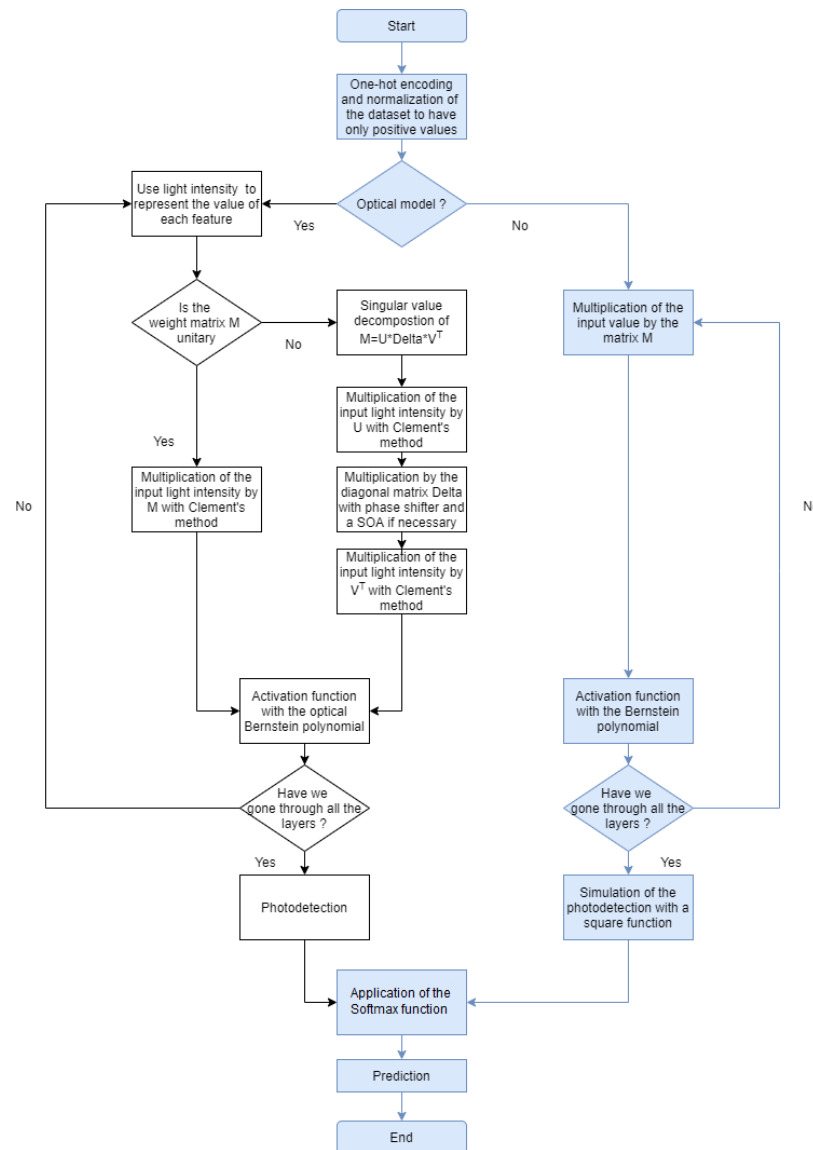


Figure 5.9 Flowchart of a neural network's forward propagation with an optical model and an electronic model

CHAPTER 6 METHODOLOGY APPLICATION TO CLASSIFICATION PROBLEMS

This chapter applies the methodology presented in Chapter 4 to classify two datasets: MNIST and Iris. Its objective is to study the efficiency of a polynomial activation function, taking into account the characteristics of an ONN, described in Chapter 5, when building our model.

6.1 Experimental Setup

Before going into details about the classification of the Iris and MNIST dataset, we use this section to briefly introduce our experimental setup. To build our model, we use the library Keras with the language Python. The Keras library offers the majority of the functions needed to define our model. Table 6.1 summarizes these functions and their arguments, that is the values that we need to provide. We can notice that every function in the Table has been discussed in Chapter 4. The choice of value for each argument is discussed throughout this Chapter.

Table 6.1 Keras functions used in our model, and their arguments

Keras Function	Arguments
Normalization of the dataset	The dataset
Create a model	"None"
Add a layer to the model	Type of layer Activation function Weight constraint
Build a customized activation function f	The function f
Use the Adam optimizer	Learning rate
Compile the model	Loss function Evaluation metrics
The Grid search algorithm	The hyperparameters The value to test for each hyperparameter
Plot the evaluation metrics	"None"

6.2 Classification of the Iris Dataset

The Iris dataset is composed of 150 data items. We use respectively 80% and 20% of the dataset to make our train and test sets. It also has three classes and four features, which corresponds respectively to three outputs, i.e., three neurons on our output layer, and four

inputs. Following the methodology presented in Chapter 4, we define in Table 6.2 the different configurations we want to test. As the number of neurons of a hidden layer, N_{hn} , is stuck between four and three, due to the characteristics of the dataset, and has to be a power of two, we define $N_{nh} = 4$ the only value to test for each layer.

Table 6.2 Domain of definition of the hyperparameters of our model

Hyperparameters	Values to test
Number of hidden layers	$N_{hl} = \{1, 2, 3\}$
Number of hidden neurons	$N_{hn}^i = 4$ for each layer
Activation functions	$B_n(ReLU)$, $B_n(Sigmoid)$, $B_n(Tanh)$ for $n \in [4, 5, 6]$

To define the batch size of our model, we use the binary search mode, beginning with batch size $= 2^3$ and upper bounded by the size of the train set. A hundred and twenty elements of the dataset are used for the train set and we use a 5-fold cross validation over it, meaning that the final size of the train set is always 4/5 of the original train set, leaving us 96 elements. As a consequence, the selection of batch sizes to train the model with is [8, 16, 32, 64].

To select the number of epochs, we do a first test with the biggest and deepest model to see how much time it needs, at least, to converge on the validation set. This model is defined in Table 6.3 and is composed of three hidden layers, each with four neurons, with a random batch size, polynomial order and activation function.

Table 6.3 Configuration of the deepest model to select the number of epochs

Activation function	Batch size	Number of hidden layers	Number of hidden neurons	Polynomial order	Epochs
ReLU	16	3	4/4/4	6	100

We plot the loss of the validation and train set (see Figure 6.1). The validation loss seems to oscillate around the same value starting from the 60th epoch. Given the fluctuation of the curve, we decide to fix the number of epochs at 80. Fixing it at 60 might give underfitted results for some models, and fixing it at 100 might cause overfitting for smaller models.

Now that the different hyperparameters have been chosen, we try all the configurations denoted in Table 6.2. The highest accuracy obtained is 97.5% and is reached three times, every time with the activation function Tanh. The details of the three configurations are presented in Table 6.4.

Despite for the activation function, this table does not help us choose the best configuration for our model. Table 6.5 displays the mean value and standard deviation of the accuracy

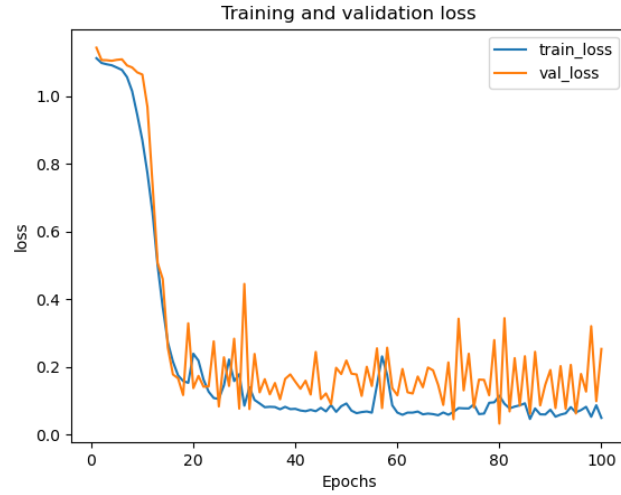


Figure 6.1 Plot of the training loss and validation loss

Table 6.4 Configuration of the model when the highest accuracy is reached

Accuracy	Loss	Activation function	Batch size	Number of hidden layers	Polynomial order
0.975	0.02	Tanh	8	1	5
0.975	0.02	Tanh	64	3	5
0.975	0.02	Tanh	16	3	6

measured for each value taken by the hyperparameters. For example, the mean value of the accuracy measured every time we use the ReLU activation function is 94 % with a standard deviation of 0.0167. In average, Tanh has the highest accuracy and it seems to be the most stable activation function, as the standard deviation is the lowest, which is consistent with our previous observation.

To select our final model we select the hyperparameters' value giving the best accuracy, and if possible the lowest standard deviation. If we were to select them from Table 6.5, the values would be:

1. Activation function = Tanh ;
2. Number of hidden layers = 1 ;
3. Batch size = 16 (lowest standard deviation) or 32 (highest mean value), and ;
4. Polynomial order = 6.

Table 6.5 Comparison tables

Accuracy		Mean value	Standard deviation
Activation functions	ReLU	0.940	0.0167
	Sigmoid	0.944	0.0144
	Tanh	0.946	0.0143
Number of hidden layers	1	0.945	0.0109
	2	0.941	0.0201
	3	0.944	0.0129
Batch size	8	0.943	0.0133
	16	0.950	0.00597
	32	0.951	0.00641
	64	0.928	0.0209
Polynomial order	4	0.944	0.0168
	5	0.940	0.0181
	6	0.946	0.00899
Total		0.943	0.0263

However, given how close the mean values are, each chosen hyperparameter might not give the best results with the selected configuration. We decide to fix activation function = Tanh, as it is the activation function that gives the highest accuracy, and build a new table of comparison, Table 6.6, for this particular case.

Table 6.6 Comparison table when the activation function is Tanh

Accuracy		Mean value	Standard deviation
Number of hidden layers	1	0.953	0.00496
	2	0.934	0.0123
	3	0.951	0.00538
Batch size	8	0.931	0.0123
	16	0.953	0.00423
	32	0.951	0.00344
	64	0.949	0.00502
Polynomial order	4	0.949	0.00433
	5	0.941	0.0126
	6	0.948	0.00532

Now when we select the best values we obtain:

1. Number of hidden layers = 1 ;
2. Batch size = 16 (highest mean value) or 32 (lowest standard deviation), and ;

3. Polynomial order = 4.

As the choice of the batch size is still delicate, we plot the accuracy in function of the batch size: Figure 6.2. We observe that for a batch size of 16, the lowest and highest accuracy are both respectively above the lowest and highest accuracy for every other batch sizes. As a consequence we decide to fix batch size = 16.

Finally, the configuration selected for our model is displayed in Table 6.7.

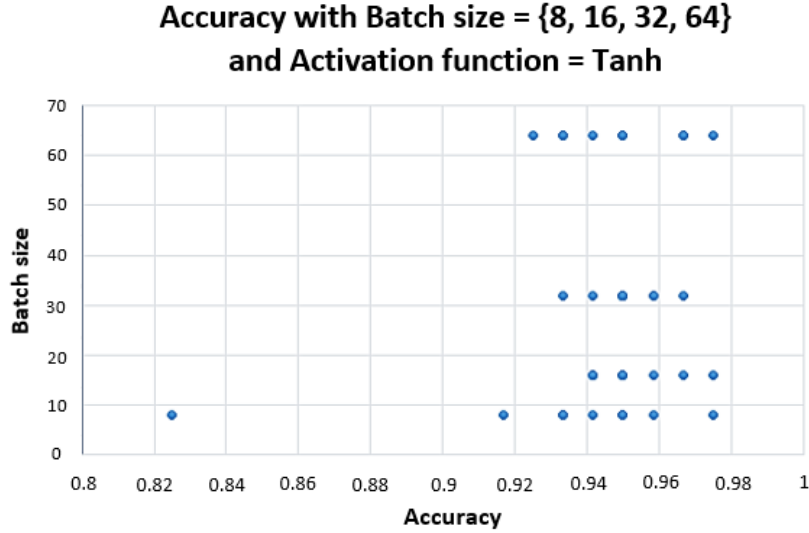


Figure 6.2 Measurement of the accuracy with different value of batch size and the activation function Tanh

Table 6.7 Configuration of the final model

Activation function	Batch size	Number of hidden layers	Polynomial order	Epochs
ReLU	16	3	6	100

To study the stability of this model, we train it 50 times. Each resulting accuracy is plotted on Figure 6.3. We obtain the average accuracy of 98.1%. The accuracy of one means that all 30 items of the test set were correctly predicted, the accuracy of 0.967 represents 1 error, 0.933, 2 errors and 0.900, 3 errors.

Given the stability of the model, we try to constrain the norms of the weights to one, as this would allow us to create an ONN without SOA. We build a new comparison table, Table 6.8, from where we select the best configuration for our network, which is:

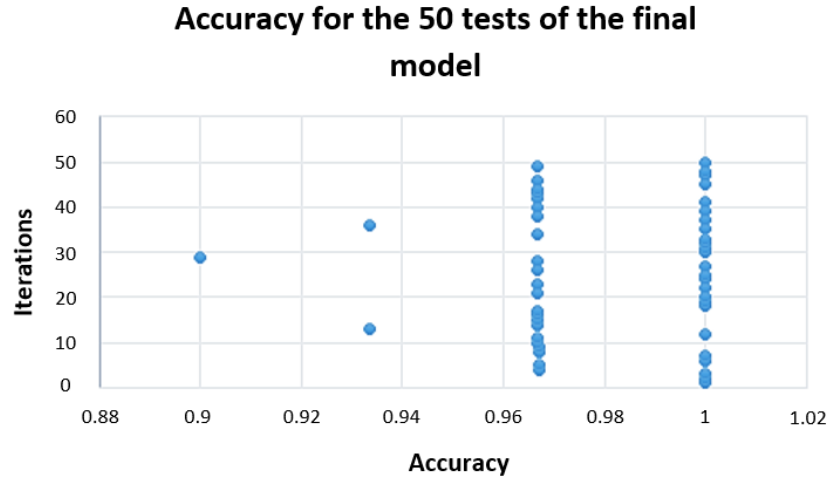


Figure 6.3 Measurement of the accuracy over 50 tests

1. Activation function = Tanh
2. Number of hidden layers = 1
3. Number of hidden neurons = 4
4. Batch size = 8
5. Polynomial order of the activation function = 5

Table 6.8 Comparison table with weight constrictions of 1

Accuracy		Mean value	Standard deviation
Activation functions	ReLU	0.898	0.0448
	Sigmoid	0.676	0.138
	Tanh	0.950	0.0631
Number of hidden layers	1	0.9415	0.0530
	2	0.833	0.0732
	3	0.776	0.130
Batch size	8	0.862	0.0622
	16	0.862	0.0716
	32	0.842	0.0790
	64	0.800	0.0995
Polynomial order	4	0.834	0.102
	5	0.848	0.0895
	6	0.843	0.0819
Total		0.846	0.159

As we did earlier, we train the selected model 50 times to study its stability. The model obtains every time the accuracy of 93.3%. We decide to repeat the operation, now using a smaller learning rate. The model then correctly predicts the entire test set every time.

The results of the classification of the Iris dataset seems promising. We could properly classify a dataset with a polynomial activation function, taking into account the characteristics of an ONN without using SOA. The next step of this chapter is to reproduce this study on a more complex classification test: an image recognition task.

6.3 Classification of the MNIST Dataset

The MNIST dataset is composed of 60,000 items. Each of them is defined by 784 pixels which corresponds to the 784 inputs of the NN. As the size of the dataset and the number of features are important, it leaves us with a lot of configurations to test with our model if we strictly follow the methodology presented in Chapter 4. For a reason of time limitations, we decide to take several initiatives:

1. We fix the polynomial order to $n=4$, as we saw in the previous experiment that they did not have a major impact on the accuracy.
2. We reduce the search of number of hidden neurons to the set $[32, 128, 512]$, except when we use only one hidden layer.
3. We reduce the search of the batch size to the set $[32, 128, 512]$.
4. We divide our research of hyperparameters in two. First, using one hidden layer to select the batch size and activation function, and second, testing different configurations of hidden neurons and hidden layers.

Finally, the domains of definition of the hyperparameters of our model are presented in Table 6.9.

Table 6.9 Domain of definition of the hyperparameters of our model

Hyperparameters	Values to test
Number of hidden layers	$N_{hl} \in [1, 2, 3]$
Number of hidden neurons	$N_{hn}^i \in [32, 128, 512]$ and $N_{hn}^{i+1} \leq N_{hn}^i \leq N_{hn}^{i-1}$
Activation functions	$B_4(\text{ReLU})$, $B_4(\text{Sigmoid})$, $B_4(\text{Tanh})$
Batch size	$[32, 128, 512]$

To fix the number of epochs, we train a deep configuration of three layers with each 512 neurons (Table 6.10). Unfortunately, the loss measured explodes. We try to define a smaller value of 0.0001 for the beginning learning rate of the Adam optimizer. However, even though the loss does not explode anymore, its value equals NaN abruptly during the 9th epoch. We call this later the "NaN" issue. Indeed, we can see in Figure 6.4 that the loss of both the train and validation set were still slowly decreasing when the algorithm was stopped. This can be caused by too much complexity in the network, i.e., the number of hidden layers and their neurons. However, here the NaN issue must be due to the use of polynomial activation functions. Indeed, we saw in Subsection 5.2.2 that for high and low input values the output results might diverge critically. To solve this issue, we could find a way of implementing an optical normalization between the layers of the network.

Table 6.10 Configuration of the deepest model to select the number of epochs : MNIST

Activation function	Batch size	Number of hidden layers	Number of neurons	Polynomial order	Epochs
ReLU	128	3	512 / 512 / 512	4	60

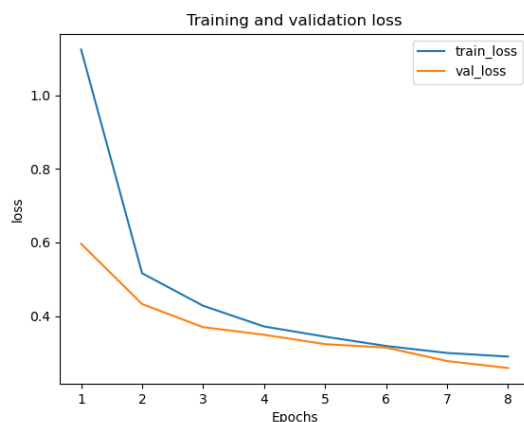


Figure 6.4 Plot of the training loss and validation loss

Given the instability of the network, we decide to study the prediction of the MNIST dataset a bit differently from the Iris one. Instead of fixing a number of epochs at which we save the model to use on the test set, we use a high number of epochs and select the architecture that gives the best results on the validation set during the training. Like this, even if the loss equals NaN at some point, we still get results¹.

¹We note that this is just a way of working around the problem and that solving the NaN issue directly might allow some models to predict more accurately with more training.

Selection of the batch size and activation function

To select the batch size and activation function, we use a configuration of one hidden layer, with 16 neurons. As we did earlier, we build a comparison table, Table 6.11, which indicates that the accuracy is higher in average when the activation function is ReLU and the batch size equals 32. As a consequence, those are the values we keep for our final model.

Table 6.11 Comparison table

Accuracy		Mean value	Standard deviation
Batch size	32	0.913	0.0250
	128	0.906	0.0211
	512	0.825	0.0717
Activation function	ReLU	0.898	0.0284
	Sigmoid	0.868	0.0577
	Tanh	0.877	0.0457

Selection of the number of hidden neurons and hidden layers

We repeat the process, this time to select the number of hidden neurons, N_{hn_i} , of the i^{th} hidden layer. We can fix the activation function to ReLU and the batch size to 32 as those are the values we selected for our final model.

We build three comparison tables, for when the number of hidden layer(s) equals 1, in Table 6.12, equals 2, in Table 6.13, and 3 in Table 6.14.

Table 6.12 Results on MNIST for one hidden layer

Accuracy	Mean value	Standard deviation
N_{HN_1}	16	0.936
	32	0.941
	64	0.961
	128	0.948
	256	0.955
	512	0.954

Over all the tests, the average accuracy when one hidden layer is used is 94.9%, when there are two hidden layers it is 93.2% and when there are three hidden layers it is 93.4%. Furthermore, we can observe that the highest accuracy is reached for one hidden layer of 64 neurons, with a mean value of 0.961 and a standard deviation of 0.00562. As a result, we define the number of hidden neurons and hidden layers respectively to 64 and 1.

Table 6.13 Results on MNIST for two hidden layers

Accuracy		Mean value	Standard deviation
N_{HN_2}/N_{HN_1}	32/32	0.945	0.00748
	32/128	0.953	0.00967
	32/512	0.919	0.0175
	128/128	0.924	0.0139
	128/512	0.925	0.00964
	512/512	0.929	0.0106

Table 6.14 Results on MNIST for three hidden layers

Accuracy		Mean value	Standard deviation
$N_{HN_3}/N_{HN_2}/N_{HN_1}$	32/32/32	0.942	0.00563
	32/32/128	0.932	0.00820
	32/32/512	0.948	0.00654
	32/128/128	0.905	0.0151
	32/128/512	0.933	0.00378
	32/512/512	0.946	0.00582
	128/128/128	0.921	0.0128
	128/128/512	0.920	0.00783
	128/512/512	0.946	0.00581
	512/512/512	0.939	0.00453

Test of the Selected Model on the MNIST Dataset

As we use a smaller network, more stable, for the final model, we put the starting learning rate of the Adam optimizer at 0.001, to its original recommended value, to accelerate the training. We measure the train and validation loss over the epochs and display the results in Figure 6.5. We can observe that the model is in a case of overfitting. The train loss keeps decreasing as the validation loss increases.

Overfitting issues can be limited by reducing the number of weights of the network. To do that, we use a function, called dropout, that can remove a given percentage of the weights of the networks. The weights dropped are chosen randomly at each epoch. We decide to add a small dropout of 0.1, meaning dropping 10% of the weights, and plot the train and validation loss again. We can observe a substantial improvement of the overfitting in Figure 6.6.

The model obtains 97.2% accuracy on the test set. We decide to try to add more dropout by defining the drop rate at 0.2 and obtain this time an accuracy of 97.4% (Figure 6.7).

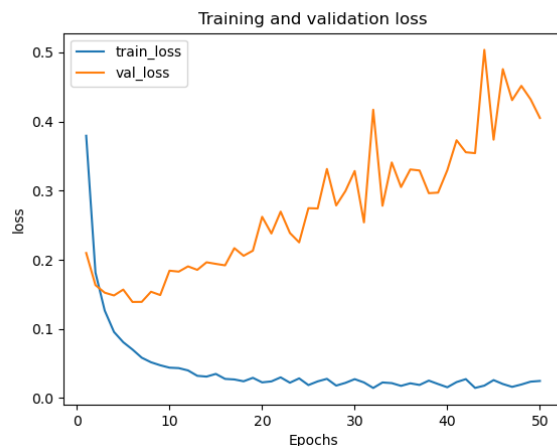


Figure 6.5 Plot of the training loss and validation loss on MNIST with a MLP



Figure 6.6 Plot of the training loss and validation loss on MNIST with a MLP and dropout=0.1

The highest accuracy reached on the MNIST dataset is 99.79 % [75], which is far from the one we have here. The main reason is that we only use a MLP and that model that reach more than 99% accuracy on this dataset are CNNs. We only use a MLP here because we do not possess all the pieces to build an all-optical CNN.

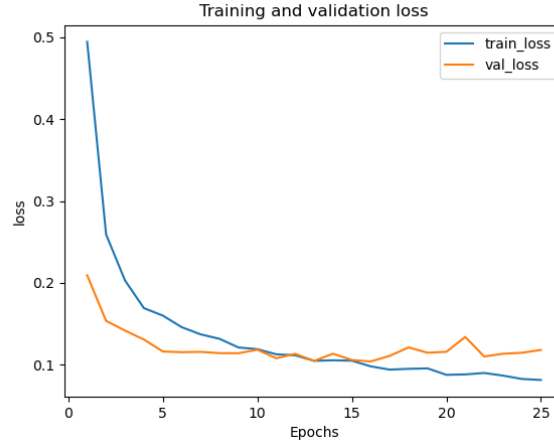


Figure 6.7 Plot of the training loss and validation loss on MNIST with a MLP and dropout=0.2

6.4 Scalability of Our Polynomial Activation Function in Convolutional Neural Networks

Even if we cannot yet produce an optical CNN with MZIs and MRRs, we can study how well our polynomial activation function would perform in these types of networks. We use one of the models of kaggle reaching the highest accuracy with a CNN on MNIST [119] without the data augmentation. We run it once on our computer with conventional activation functions, reaching 99.6% accuracy, and then replace all the activation function by its polynomial approximation. We display the train and validation loss in Figure 6.8. We reach 99% accuracy on the test set.

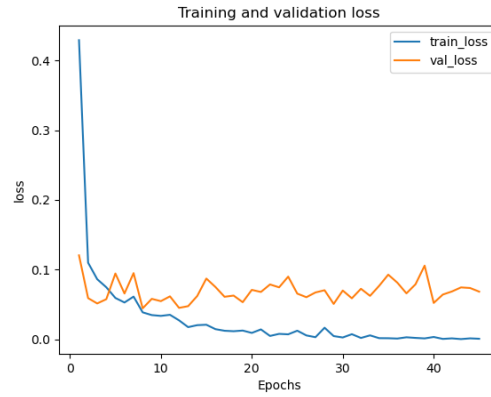


Figure 6.8 Plot of the train and validation loss on MNIST with a CNN

6.5 Discussion

The first series of tests carried out on the Iris dataset showed great results and let us think that an all-optical forward propagation seems feasible. To challenge the model and study its scalability, we performed a second series of tests on a more complex dataset: MNIST. We display all the results of our experiment with the MNIST dataset in Table 6.15. We can observe that when our model is implemented with a MLP architecture it gives lower results than the optical model proposed in [87]. However, the training seems always to be stopped prematurely, which makes us think that solving this issue would allow a rise of the accuracy. We also tested our model with a CNN architecture. This time the results are more promising and the accuracy even surpasses the one of the state-of-the-art optical models, but still staying inferior to the results obtained by more conventional models. Furthermore, the CNN model here is not entirely implementable optically. Indeed, there is no example of optical pooling layers in the state-of-the-art yet.

Table 6.15 Accuracy obtained on MNIST for state-of-the-art models and the one proposed in this thesis

Paper	Architecture	Proposition of an optical implementation	Accuracy
[86]	MLP	Yes	95%
[87]	MLP	Yes	97.8%
[120]	MLP	No	99.65%
[75]	CNN	No	99.79%
Proposed model	MLP	Yes	97.4%
Proposed model	CNN	No	99%

CHAPTER 7 CONCLUSION

This chapter concludes the thesis by presenting a summary of our work and its limitation. It finishes on a proposition of future research.

7.1 Summary of Works

This thesis presented a study about the relevancy of an optical polynomial activation function. It legitimates the use of integrated photonic to accelerate ANNs. We built a model of a NN with specific limitations making its optical conversion feasible. To do that, we first defined a methodology to optimize a MLP in Chapter 4. Then, we specified several characteristics of ONNs in Chapter 5 that need to be incorporated in the network. To test our model, we used it to classify two datasets: Iris and MNIST.

The model performed well on Iris, obtaining the accuracy of 100% in one out of two cases, without getting more than three errors. We took advantage of the model's stability and tried to constrain the norm of the weight to a value of one, as seen in the subsection 5.3. This made it possible to achieve an optical matrix multiplication without SOA. This time the model obtained the accuracy of 100% in all the cases. The results obtained on the Iris dataset appear very promising. However, it is a small dataset of only 150 elements and an easy classification problem. That is why we tried to classify a bigger dataset.

When used on the MNIST dataset, the highest accuracy obtained by our model is of 97.4%, which stands 2.39% below the best accuracy reached on this dataset in the state of the art. It seems that, for the given architecture, an ONN is not the best fit. We decided to use a more adapted configuration such as a CNN architecture. However, some functions of the CNN are not yet represented optically. In the idea that this will be done in the near future, we used a CNN with conventional activation functions, then replaced them by their polynomial estimate. We obtained 99.6% accuracy in the first case, and 99.0% in the second. Using polynomial activation functions gave us a lower accuracy. But it is still a higher one than the results obtained by the photonic network of [86] and [87] with ideal conditions.

7.2 Limitations

The first limitation of our solution is the instability of our model. When applied on MNIST, the algorithm would, sometimes, freeze abruptly during the training and return a NaN value for the loss, especially with deeper networks. This impacts the selection of the top configura-

tion for our model and is preventing us from obtaining better results. For the same reason, the test on the weight constriction could not be performed on MNIST as the training kept stopping. Thus we cannot conclude on the possibility of only using weights with a norm of one.

Another issue with our solution is that it does not take into consideration the instability and loss that can be observed when using optical components.

7.3 Future Research

The first phase to improve the results of our solution is to solve the “NaN” issue and train the model without being interrupted. As it is a common issue in neural networks, there are different solutions offered online. We propose to follow the one presented in [121] which introduces a step-by-step method to debug a ML written in Keras.

The second phase that seems essential if we want to obtain state-of-the-art results with ONNs is to be able to build a convolutional optical neural network. As we know, the pooling layers characteristic of its architecture seem hard to implement optically. However, in [122], they show that it appears possible to replace the pooling layers by convolutional ones. As a convolutional operation can be represented by a matrix multiplication, this would allow us to offer an all-optical CNN.

Finally, in the long run we hope to be able to measure the losses introduced by optical components and build ONNs that limit them such as [87]. We want to do that by building a simulation tool that consider the imprecision of an ONN when this network is being optimized.

REFERENCES

- [1] G. Fortino, “Agents meet the iot: Toward ecosystems of networked smart objects,” *IEEE Systems, Man, and Cybernetics Magazine*, vol. 2, no. 2, pp. 43–47, 2016.
- [2] J. Gubbi *et al.*, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future Gener. Comput. Syst.*, vol. 29, pp. 1645–1660, 2013.
- [3] A. Meola, “How iot & smart home automation will change the way we live,” *Business Insider*. Available online: <http://www.businessinsider.com/internet-of-things-smart-home-automation-2016-8> (accessed on 11 May 2017), 2016.
- [4] K. Mahmud *et al.*, “Integration of electric vehicles and management in the internet of energy,” *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 4179 – 4203, 2018.
- [5] J. M. C. Rodríguez, “Efficiency and reliability in bringing ai into transport and smart cities solutions,” 2019.
- [6] T. Alladi *et al.*, “Blockchain applications for industry 4.0 and industrial iot: A review,” *IEEE Access*, vol. 7, pp. 176 935–176 951, 2019.
- [7] A. W. Services. (2020) Amazon simple notification service. [Online]. Available: <https://aws.amazon.com/sns>
- [8] Cisco. (2020) Cisco iot. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html>
- [9] S. S. Zia *et al.*, “Smart traffic light system by using artificial intelligence,” *Sindh University Research Journal*, vol. 50, pp. 639–646, 2019.
- [10] K. T. Chui, M. D. Lytras, and A. Visvizi, “Energy sustainability in smart cities: Artificial intelligence, smart monitoring, and optimization of energy consumption,” *Energies*, vol. 11, p. 2869, 2018.
- [11] J. Lee *et al.*, “Industrial artificial intelligence for industry 4.0-based manufacturing systems,” *Manufacturing letters*, vol. 18, pp. 20–23, 2018.
- [12] M. Chen *et al.*, “Artificial neural networks-based machine learning for wireless networks: A tutorial,” *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 3039–3071, 2019.

- [13] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks : the official journal of the International Neural Network Society*, vol. 61, pp. 85–117, 2015.
- [14] E. Buber and B. Dirir, “Performance analysis and cpu vs gpu comparison for deep learning,” *2018 6th International Conference on Control Engineering Information Technology (CEIT)*, pp. 1–6, 2018.
- [15] D. Steinkraus, I. Buck, and P. Y. Simard, “Using gpus for machine learning algorithms,” in *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*, 2005, pp. 1115–1120 Vol. 2.
- [16] P. Merolla *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, pp. 668–673, 2014.
- [17] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 1 – 12, 2017.
- [18] L. Cui *et al.*, “A survey on application of machine learning for internet of things,” *International Journal of Machine Learning and Cybernetics*, vol. 9, pp. 1399–1417, 2018.
- [19] S. Suthaharan, “Big data classification: problems and challenges in network intrusion prediction with machine learning,” *SIGMETRICS Perform. Evaluation Rev.*, vol. 41, pp. 70–73, 2014.
- [20] P. V. Klaine *et al.*, “A survey of machine learning techniques applied to self-organizing cellular networks,” *IEEE Communications Surveys Tutorials*, vol. 19, pp. 2392–2431, 2017.
- [21] X. wen Chen and X. Lin, “Big data deep learning: Challenges and perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014.
- [22] Y.-H. Chen *et al.*, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, 2016.
- [23] E. García-Martín *et al.*, “Estimation of energy consumption in machine learning,” *J. Parallel Distributed Comput.*, vol. 134, pp. 75–88, 2019.

- [24] D. Reinsel, J. Gantz, and J. Rydning, “The digitization of the world from edge to core,” *IDC White Paper*, 2018. [Online]. Available: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>
- [25] J. Burgess, “Rtx on—the nvidia turing gpu,” *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.
- [26] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. 114–117, April 1965.
- [27] L. B. Kish, “End of moore’s law: thermal (noise) death of integration in micro and nano electronics,” *Physics Letters A*, vol. 305, no. 3, pp. 144 – 149, 2002. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0375960102013658>
- [28] D. A. Patterson and J. L. Hennessy, “Turing lecture: A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development,” June 2018.
- [29] I. Stoica *et al.*, “A berkeley view of systems challenges for ai,” 2017.
- [30] Y. Shen *et al.*, “Deep learning with coherent nanophotonic circuits.” *Nature Photon*, no. 11, pp. 441–446, June 2017.
- [31] A. N. Tait, “Silicon photonic neural networks,” 2018.
- [32] B. Bai *et al.*, “Towards silicon photonic neural networks for artificial intelligence,” *Science China Information Sciences*, vol. 63, 2020.
- [33] G. T. Reed and A. P. Knights, “Silicon photonics: An introduction,” 2004.
- [34] L. De Marinis, P. Cococcioni, and N. Andriolli, “Photonic neural networks: A survey,” *IEEE Access*, vol. 7, pp. 175 827–175 841, 2019.
- [35] W. R. Clements *et al.*, “Optimal design for universal multiport interferometers,” *Optica*, vol. 3, no. 12, pp. 1460–1465, Dec 2016.
- [36] T. F. de Lima *et al.*, “Machine learning with neuromorphic photonics,” *Journal of Lightwave Technology*, vol. 37, pp. 1515–1534, 2019.
- [37] K. Shiflett *et al.*, “Pixel: Photonic neural network accelerator,” *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 474–487, 2020.

- [38] B. G. Lee and N. Dupuis, “Silicon photonic switch fabrics: Technology and architecture,” *Journal of Lightwave Technology*, vol. 37, pp. 6–20, 2019.
- [39] T. W. Hughes *et al.*, “Training of photonic neural networks through in situ backpropagation and gradient measurement,” *Optica*, vol. 5, no. 7, pp. 864–871, Jul 2018.
- [40] F. Chollet *et al.*, “Keras. github; 2015,” 2015.
- [41] “Artificial intelligence,” in *The American Heritage® Dictionary of the English Language, Fifth Edition*. Houghton Mifflin Harcourt Publishing Company, 2020. [Online]. Available: <https://www.ahdictionary.com/word/search.html?q=artificial+intelligence>
- [42] “Machine learning,” in *Collins English Dictionary*. HarperCollins Publishers, 2020. [Online]. Available: <https://www.collinsdictionary.com/dictionary/english/machine-learning>
- [43] E. Alpaydin, “Introduction to machine learning,” in *Adaptive computation and machine learning*, 2004.
- [44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [45] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach. Third Edition*. Prentice Hall, 2009.
- [46] G. Hinton and T. Sejnowski, *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999.
- [47] X. Zhu and A. B. Goldberg, *Introduction to Semi-Supervised Learning*. Morgan Claypool, 2009.
- [48] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.
- [49] A. K. Jain, J. Mao, and K. M. Mohiuddin, “Artificial neural networks: A tutorial,” *IEEE Computer*, vol. 29, pp. 31–44, 1996.
- [50] Y. Freund and R. E. Schapire, “Large margin classification using the perceptron algorithm,” *Machine Learning*, vol. 37, pp. 277–296, 1998.
- [51] W. Hu *et al.*, “Deep convolutional neural networks for hyperspectral image classification,” *J. Sensors*, vol. 2015, pp. 258 619:1–258 619:12, 2015.

- [52] R. Collobert and S. Bengio, “Links between perceptrons, mlps and svms,” in *ICML '04*, 2004.
- [53] C. Nwankpa *et al.*, “Activation functions: Comparison of trends in practice and research for deep learning,” *ArXiv*, vol. abs/1811.03378, 2018.
- [54] “Backpropagation,” in *The American Heritage® Dictionary of the English Language, Fifth Edition*. Houghton Mifflin Harcourt Publishing Company, 2020. [Online]. Available: <https://www.ahdictionary.com/word/search.html?q=backpropagation>
- [55] R. Rojas, “The backpropagation algorithm,” in *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 149–182. [Online]. Available: https://doi.org/10.1007/978-3-642-61068-4_7
- [56] F. Gohring de Magalhaes, “High-level modelling of optical integrated networks-based systems with the provision of a low latency controller,” Ph.D. dissertation, 2017.
- [57] L. Chrostowski and M. Hochberg, “Silicon photonics design: From devices to systems,” 2015.
- [58] X. Tu *et al.*, “State of the art and perspectives on silicon photonic switches,” *Micromachines*, vol. 10, no. 1, p. 51, Jan 2019. [Online]. Available: <http://dx.doi.org/10.3390/mi10010051>
- [59] R. Khatun *et al.*, “Optimization of 22 mzi electro-optic switch and its application as logic gate,” in *2015 18th International Conference on Computer and Information Technology (ICCIT)*, 2015, pp. 294–299.
- [60] A. W. MacKay, “Complex phase biasing of silicon mach-zehnder interferometer modulators,” 2014.
- [61] P. Rüdiger. (2019) Optical power. [Online]. Available: https://www.rp-photonics.com/optical_power.html
- [62] ——. (2019) Propagation losses. [Online]. Available: https://www.rp-photonics.com/propagation_losses.html
- [63] ——. (2019) Amplifier noise. [Online]. Available: https://www.rp-photonics.com/amplifier_noise.html
- [64] “Field programmable gate array,” in *The Computer Language Company Inc.*, 2020. [Online]. Available: <https://www.computerlanguage.com/results.php?definition=FPGA>

- [65] N. Suda *et al.*, “Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks,” in *FPGA '16*, 2016.
- [66] S. Srinivasan *et al.*, “High performance scalable fpga accelerator for deep neural networks,” *ArXiv*, vol. abs/1908.11809, 2019.
- [67] K. Guo *et al.*, “[dl] a survey of fpga-based neural network inference accelerators,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, pp. 2:1–2:26, 2019.
- [68] P. D. Sato Kaz, Young Cliff. (2017) An in-depth look at google’s first tensor processing unit (tpu). [Online]. Available: <https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>
- [69] M. Pfeiffer and T. Pfeil, “Deep learning with spiking neurons: Opportunities and challenges,” *Frontiers in neuroscience*, vol. 12, pp. 774–774, October 2018.
- [70] S. J. Kim *et al.*, “Spiking-yolo: Spiking neural network for real-time object detection,” *ArXiv*, vol. abs/1903.06530, 2019.
- [71] A. Tavanaei *et al.*, “Deep learning in spiking neural networks,” *Neural Networks*, vol. 111, pp. 47 – 63, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608018303332>
- [72] P. M. J. V. A. D. S. M. S. K. Esser, R. Appuswamy, “Backpropagation for energy-efficient neuromorphic computing,” *Advances in Neural Information Processing Systems*, p. 1117–1125, 2015.
- [73] B. Rueckauer *et al.*, “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” *Frontiers in Neuroscience*, vol. 11, 2017.
- [74] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [75] L. Wan *et al.*, “Regularization of neural networks using dropconnect,” ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. [Online]. Available: <http://proceedings.mlr.press/v28/wan13.html>
- [76] “Spike-timing-dependent plasticity,” in *Wikipédia*, 25 april 2020. [Online]. Available: https://en.wikipedia.org/wiki/Spike-timing-dependent_plasticity

- [77] G. Indiveri, E. Chicca, and R. J. Douglas, “A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity,” *IEEE Transactions on Neural Networks*, vol. 17, pp. 211–221, 2006.
- [78] A. Tavanaei *et al.*, “Deep learning in spiking neural networks,” *Neural networks : the official journal of the International Neural Network Society*, vol. 111, pp. 47–63, 2019.
- [79] S. B. Furber *et al.*, “The spinnaker project,” *Proceedings of the IEEE*, vol. 102, pp. 652–665, 2014.
- [80] J. Schemmel *et al.*, “A wafer-scale neuromorphic hardware system for large-scale neural modeling,” *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, 2010.
- [81] M. Davies *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, pp. 82–99, 2018.
- [82] Y. Du *et al.*, “An analog neural network computing engine using cmos-compatible charge-trap-transistor (ctt),” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1811–1819, 2019.
- [83] D. Kwon *et al.*, “On-chip training spiking neural networks using approximated back-propagation with analog synaptic devices,” *Frontiers in Neuroscience*, vol. 14, 2020.
- [84] B. Shi, N. Calabretta, and R. Stabile, “Image classification with a 3-layer soa-based photonic integrated neural network,” *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*, pp. 1–3, 2019.
- [85] M. Connelly, *Semiconductor Optical Amplifiers*, 01 2004.
- [86] Y. Shen and Y. Bai, “Statistical computing with integrated photonics system,” *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*, pp. 1–1, 2019.
- [87] M. Y.-S. Fang *et al.*, “Design of optical neural networks with component imprecisions,” *Optics express*, vol. 27 10, pp. 14 009–14 029, 2019.
- [88] T. W. Hughes *et al.*, “Training of photonic neural networks through in situ backpropagation,” *2019 Conference on Lasers and Electro-Optics (CLEO)*, pp. 1–2, 2018.

- [89] G. Mourgias-Alexandris *et al.*, “An all-optical neuron with sigmoid activation function.” *Optics express*, vol. 27 7, pp. 9620–9630, 2019.
- [90] A. N. Tait *et al.*, “Silicon photonic modulator neuron,” *Physical review applied*, vol. 11, p. 064043, 2018.
- [91] Q. Xu and M. Lipson, “Optical bistability based on the carrier dispersion effect in silicon ring resonators,” 2006.
- [92] M. Babaeian *et al.*, “Nonlinear optical components for all-optical probabilistic graphical model,” *Nature Communications*, vol. 9, 2018.
- [93] J. Moolayil, J. Moolayil, and S. John, *Learn Keras for Deep Neural Networks*. Springer, 2019.
- [94] K. e. a. Katanforoosh. (2019) deeplearning.ai parameter optimization in neural networks. [Online]. Available: <https://www.deeplearning.ai/ai-notes/optimization/>
- [95] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals Eugen.*, vol. 7, pp. 179–188, 1936.
- [96] D. C. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” *CoRR*, vol. abs/1202.2745, 2012. [Online]. Available: <http://arxiv.org/abs/1202.2745>
- [97] F. Harrag and S. Gueliani, “Event extraction based on deep learning in food hazard arabic texts,” *CoRR*, vol. abs/2008.05014, 2020. [Online]. Available: <https://arxiv.org/abs/2008.05014>
- [98] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery, 2020. [Online]. Available: <https://books.google.ca/books?id=uAPuDwAAQBAJ>
- [99] Wikipedia contributors, “Hyperparameter optimization — Wikipedia, the free encyclopedia,” 2020, [Online; accessed 29-September-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=972537954
- [100] B. Macukow, “Neural networks – state of art, brief history, basic models and architecture,” vol. 9842, 09 2016, pp. 3–14.
- [101] K. Sheela and S. N. Deepa, “Review on methods to fix number of hidden neurons in neural networks,” *Mathematical Problems in Engineering*, vol. 2013, 06 2013.

- [102] C. A. Doukim, J. A. Dargham, and A. Chekima, "Finding the number of hidden neurons for an mlp neural network using coarse to fine search technique," in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, 2010, pp. 606–609.
- [103] G. James *et al.*, *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. [Online]. Available: <https://faculty.marshall.usc.edu/gareth-james/ISL/>
- [104] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
- [105] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. [Online]. Available: <http://arxiv.org/abs/1206.5533>
- [106] N. Buduma and N. Locascio, *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.", 2017.
- [107] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
- [108] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [109] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [110] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.
- [111] Reck *et al.*, "Experimental realization of any discrete unitary operator." *Physical review letters*, vol. 73 1, pp. 58–61, 1994.
- [112] H. Yanai, K. Takeuchi, and Y. Takane, *Singular Value Decomposition (SVD)*. New York, NY: Springer New York, 2011, pp. 125–149. [Online]. Available: https://doi.org/10.1007/978-1-4419-9887-3_5

- [113] K. M. Levasseur, “A probabilistic proof of the weierstrass approximation theorem,” *American Mathematical Monthly*, vol. 91, pp. 249–250, 1984.
- [114] H. El-Derhalli, S. Le Beux, and S. Tahar, “Design space exploration of stochastic computing architectures implemented using integrated optics,” *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2020.
- [115] W. Qian *et al.*, “An architecture for fault-tolerant computation with stochastic logic,” *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.
- [116] Hao Zheng *et al.*, “Improving deep neural networks using softplus units,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–4.
- [117] L. Chrostowski and M. Hochberg, *Silicon Photonics Design: From Devices to Systems*, 03 2015.
- [118] N. Srivastava *et al.*, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.
- [119] “How to choose cnn architecture mnist,” <https://www.kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist>, accessed: 2020-11-01.
- [120] D. C. Cireşan *et al.*, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.
- [121] “Debugging a machine learning model written in tensorflow and keras,” <https://towardsdatascience.com/debugging-a-machine-learning-model-written-in-tensorflow-and-keras-f514008ce736>, accessed: 2020-11-02.
- [122] J. T. Springenberg *et al.*, “Striving for simplicity: The all convolutional net,” *CoRR*, vol. abs/1412.6806, 2015.

APPENDIX A CLEMENT'S OPTICAL MODEL FOR MATRIX MULTIPLICATION

MZI matrix :

$$MZI(\phi, \theta) = \begin{bmatrix} e^{i\phi} \cos \theta & -\sin \theta \\ e^{i\phi} \sin \theta & \cos \theta \end{bmatrix} \quad (\text{A.1})$$

$$T_{i,j}(\phi, \theta) = \begin{bmatrix} 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & & & & & \vdots \\ \vdots & & \ddots & & & & \vdots \\ \vdots & & & e^{i\phi} \cos \theta & -\sin \theta & & \vdots \\ \vdots & & & e^{i\phi} \sin \theta & \cos \theta & & \vdots \\ \vdots & & & & \ddots & & \vdots \\ \vdots & & & & & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 & 1 \end{bmatrix} \quad (\text{A.2})$$

We use $T_{m,n}$ when we need to obtain a null value on a diagonal with an even number. For example if we want to nulls the element $U[p, q]$ we use the matrix $T_{p-1,p}$.

$$\bar{U}[p, q] = 0 \leftrightarrow U[p, q] \times \cos \theta + U[p-1, q] \times e^{i\phi} \sin \theta = 0 \quad (\text{A.3})$$

We define $U[p, q] = r_1 e^{i\phi_1}$ and $U[p-1, q] = r_2 e^{i\phi_2}$.

$$\begin{aligned} \bar{U}[p, q] = 0 &\leftrightarrow r_1 e^{i\phi_1} \cos \theta + r_2 e^{i\phi_2} e^{i\phi} \sin \theta = 0 \\ &\leftrightarrow r_2 e^{i\phi_2} e^{i\phi} \sin \theta = -r_1 e^{i\phi_1} \cos \theta \\ &\leftrightarrow \frac{r_2 e^{i\phi_2} e^{i\phi} \sin \theta}{r_1 e^{i\phi_1} \cos \theta} = -1 \\ &\leftrightarrow e^{i(\phi_2 + \phi - \phi_1)} = -\frac{r_1}{r_2 \tanh \theta} \end{aligned} \quad (\text{A.4})$$

$-\frac{r_1}{r_2 \tanh \theta}$ is a real number so the real part of the exponential is null. We obtain the following system of equations :

$$\begin{cases} \sin(\phi_2 + \phi - \phi_1) = 0 \\ \cos(\phi_2 + \phi - \phi_1) = -\frac{r_1}{r_2 \tanh \theta} \end{cases} \quad (\text{A.5})$$

$$(\text{A.6})$$

$$\begin{cases} \phi_2 + \phi - \phi_1 = 0 \mod \pi \end{cases} \quad (\text{A.7})$$

$$\begin{cases} \cos \phi_2 \cos (\phi - \phi_1) - \sin \phi_2 \sin (\phi - \phi_1) = -\frac{r_1}{r_2 \tanh \theta} \end{cases} \quad (\text{A.8})$$

$$\begin{cases} \phi = \phi_1 - \phi_2 \mod \pi \end{cases} \quad (\text{A.9})$$

$$\begin{cases} \cos^2 (\phi_2) + \sin^2 (\phi_2) = -\frac{r_1}{r_2 \tanh \theta} \end{cases} \quad (\text{A.10})$$

$$\begin{cases} \phi = \phi_1 - \phi_2 \mod \pi \end{cases} \quad (\text{A.11})$$

$$\begin{cases} \theta = -\arctan\left(\frac{r_1}{r_2}\right) \end{cases} \quad (\text{A.12})$$

We use T^{-1} when we need to obtain a null value on a diagonal with an odd number.

$$MZI^{-1}(\phi, \theta) = \begin{bmatrix} e^{-i\phi} \cos \theta & e^{-i\phi} \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (\text{A.13})$$

If we want to null the value $U[p, q]$ we use the matrix $T^{-1}[q, q+1]$.

$$\bar{U}[p, q] = 0 \leftrightarrow U[p, q] \times e^{-i\phi} \cos \theta - U[p, q+1] \times \sin \theta = 0 \quad (\text{A.14})$$

We define $U[p, q] = r_1 e^{i\phi_1}$ and $U[p, q+1] = r_2 e^{i\phi_2}$.

$$\begin{aligned} \bar{U}[p, q] = 0 &\leftrightarrow r_1 e^{i\phi_1} e^{-i\phi} \cos \theta - r_2 e^{i\phi_2} \sin \theta = 0 \\ &\leftrightarrow r_1 e^{i\phi_1} e^{-i\phi} \cos \theta = r_2 e^{i\phi_2} \sin \theta \\ &\leftrightarrow \frac{r_1 e^{i\phi_1} e^{-i\phi} \cos \theta}{r_2 e^{i\phi_2} \sin \theta} = 1 \\ &\leftrightarrow e^{i(\phi_1 - \phi - \phi_2)} = \frac{r_2 \tanh \theta}{r_1} \end{aligned} \quad (\text{A.15})$$

$\frac{r_1 \tanh \theta}{r_2}$ is a real number so the real part of the exponential is null. We obtain the following system of equations :

$$\begin{cases} \sin (\phi_1 - \phi - \phi_2) = 0 \end{cases} \quad (\text{A.16})$$

$$\begin{cases} \cos (\phi_1 - \phi - \phi_2) = \frac{r_2 \tanh \theta}{r_1} \end{cases} \quad (\text{A.17})$$

$$\begin{cases} \phi_1 - \phi - \phi_2 = 0 \mod \pi \end{cases} \quad (\text{A.18})$$

$$\begin{cases} \cos \phi_1 \cos (\phi + \phi_2) + \sin \phi_1 \sin (\phi + \phi_2) = \frac{r_2 \tanh \theta}{r_1} \end{cases} \quad (\text{A.19})$$

$$\begin{cases} \phi = \phi_1 - \phi_2 \mod \pi \\ \cos^2(\phi_1) + \sin^2(\phi_1) = \frac{r_2 \tanh \theta}{r_1} \end{cases} \quad \begin{matrix} \text{(A.20)} \\ \text{(A.21)} \end{matrix}$$

$$\begin{cases} \phi = \phi_1 - \phi_2 \mod \pi \\ \theta = \arctan\left(\frac{r_1}{r_2}\right) \end{cases} \quad \begin{matrix} \text{(A.22)} \\ \text{(A.23)} \end{matrix}$$