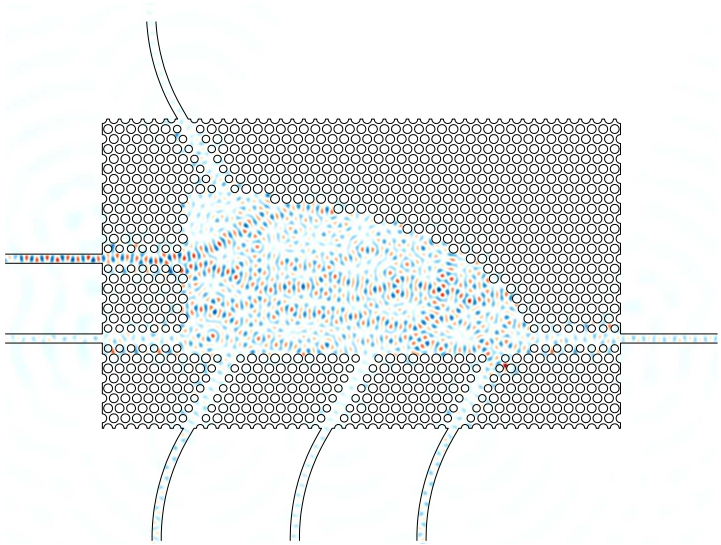
UNIVERSITEIT GENT

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Novel architectures for brain-inspired photonic computers

Nieuwe architecturen voor brein-geïnspireerde fotonische computers

Floris Laporte



FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Proefschrift tot het bekomen van de graad
Doctor in de ingenieurswetenschappen:
fotonica
Academiejaar 2019-2020

UNIVERSITEIT GENT

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Promotoren:

Prof. Dr. Ir. Peter Bienstman
Prof. Dr. Ir. Joni Dambre

Examencommissie:

Prof. Dr. Ir. Filip De Turck (voorzitter)	Universiteit Gent, INTEC
Prof. Dr. Ir. Peter Bienstman (promotor)	Universiteit Gent, INTEC
Prof. Dr. Ir. Joni Dambre (promotor)	Universiteit Gent, IDLab
Prof. Dr. Ir. Francis wyffels	Universiteit Gent, IDLab
Prof. Dr. Ir. Dries Vande Ginste	Universiteit Gent, IDLab
Prof. Dr. Ir. Wim Bogaerts	Universiteit Gent, INTEC
Prof. Dr. Ir. Serge Massar	Université Libre de Bruxelles, LIQ
Dr. Ir. Martin Fiers	Luceda Photonics

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
Technologiepark-Zwijnaarde 126, 9050 Gent, België

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Proefschrift tot het behalen van de graad
Doctor in de ingenieurswetenschappen:
fotonica
Academiejaar 2019-2020

Acknowledgment

Photonic Neuromorphic computing, ofwel *Photonics + machine learning*. Het was van in het begin een doctoraat dat mij aansprak. Een combinatie waarvan elk deel gerepresenteerd werd door mijn twee promotoren: prof. Peter Bienstman en prof. Joni Dambre. Ik zou hen willen bedanken voor het verder aanflakkeren van de interesse in elk van beide gebieden afzonderlijk en me te helpen ze in een interessante manier te combineren tijdens dit doctoraat. Bovendien apprecieer ik de hoeveelheid vrijheid, vertrouwen en bijsturing die ik van hen kreeg. Het liet me toe mijn eigen ding te doen zonder het grotere geheel uit het oog te verliezen.

Peter, bedankt om altijd beschikbaar te zijn voor eender welke vraag op eender welk moment, voor je geduld als ik weer eens iets nieuws probeerde en voor je waardevolle input tijdens het schrijven van dit werk.

I would also like to thank the members of the jury, whose many valuable comments definitely improved the final version of the manuscript.

Each member of the *Photonic Reservoir Computing* group also had an important impact on my research these last four years. Andrew, you taught me to work with high speed photonic reservoir setup. Bendix, you planted the seed for the Photontorch simulator, Alessio who experimented with it first and Emmanuel, who started using it extensively. Matthias, your machine learning insights were very valuable. Stijn and Chonghuai for continuously improving the reservoir setup and sharing your knowledge with the rest of us.

Working in a large research group, like the *Photonic Research Group*, has the advantage that you're being aided, getting influenced and inspired by many people during the period of the PhD. I'd like to especially thank my office mates Chonghuai, Umar and "office responsible" Anton. I really appreciate discussing both our photonics-related problems and life experiences together. Alexandros, for helping me with photonic crystals. Fabio, for directing me away from photonic crystals. Jasper and Michael, your continued support in the measurement labs is really appreciated. Alejandro, Alessio, Irfan, I really apprciate you guys spearheading the ping-pong group (I should work on my presence). Wim, Antonio, Umar, Lukas, Mi, Xiangfeng, Hong, thanks for letting me tag along with the programmable photonics team in Florida. I had a great time. Also, thanks for coming to the parties Alejandro, Alessio, Ana, Andrew, Anton, Artur, Camiel, Chonghuai, Grigorij, Kasper, Mahmoud, Nina, Sanja, ...

Even though I have always tried to spend as little time as possible there, a big thank you is in place for the people who regularly helped me in the cleanroom. Sören and Muhammad in particular, for your patience helping me fabricate yet another batch of experimental cavities, but also Liesbet for the occasional SEM image.

Tijdens het werken aan een doctoraat kan de boog echter niet altijd gespannen staan. Daarom wil ik graag mijn vrienden en familie bedanken die mij daar dagelijks aan herinneren. Mijn vrienden van de scouts, volleybal, squash, het "elite" ski team, de Musketeers, de "schoolkaaifamilie" en recentelijk ook de vrienden van onze nieuwe roundnet club. Jullie hielpen mij de dagelijkse beslommeringen van een doctoraat met gemak relativeren.

In het bijzonder wil ik ook graag mijn ouders, broer en zus bedanken. Mama, papa, Jeroen en Jana, merci voor het aanhoudende geloof en de niet aflatende steun die ik kon genieten tijdens de laatse vier jaar én ervoor.

And of course, I'd like to thank my girlfriend Eva, I wouldn't have been able to do this without you. Many thanks for your patience and understanding while I was writing this book. Your support and love mean a lot to me.

Gent, maart 2020
Floris Laporte

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

ANN	Artifical Neural Network

C

CMOS	Complementary Metal-Oxide-Semiconductor

D

DC	Directional Coupler
DSP	Digital Signal Processor

E

ESN	Echo State Network

F

FCNN	Fully-Connected Neural Network
FDTD	Finite-Difference Time Domain
FFNN	Feed-Forward Neural Network
FIR	Finite Impulse Response

G

GC	Grating Coupler
GD	Gradient Descent
GPU	Graphical Processing Unit

I

IIF	Infinite Impulse Response

L

LASSO	Least Absolute Shrinkage and selection operator
LDA	Linear Discriminant Analysis
LSM	Liquid State Machine
LSTM	Long Short Term Memory

M

MC	Memory Containing
ML	Memory Less
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MZI	Mach-Zehnder Interferometer

N

NN	Neural Network
NRMSE	Normalized Root Mean Squared Error
NRZ	Non-Return to Zero
MSE	Mean Squared Error

O

ODE	Ordinary Differential Equation
OSA	Optical Spectrum Analyzer
OTF	Optical Tunable Filter

P

PC	Polarization Controller
PCA	Principle Component Analysis
PIC	Photonic Integrated Circuit
PRBS	Pseudo-Random Bit Stream
PRC	Photonic Reservoir Computing

Q

QAM	Quadrature Amplitude Modulation

R

RIE	Reactive Ion Etching
RC	Reservoir Computing
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RTO	Real Time Oscilloscope
RZ	Return to Zero

S

SEM	Scanning Electron Microscope
SLM	Spatial Light Modulator
SNR	Signal to Noise Ratio

SOA	Semiconductor Optical Amplifier
SOI	Silicon on Insulator
SVD	Singular Value Decomposition

T

TE	Transverse Electric (mode)
TM	Transverse Magnetic (mode)

U

UNN	Unitary Neural Network
URNN	Unitary Recurrent Neural Network

X

XE	Cross Entropy

Nederlandse Samenvatting

In deze tijd van het internet leven we in een wereld die volledig vertrouwt op digitale informatieverwerking. De hoeveelheid informatie tot onze beschikking is vrijwel onbeperkt en groeit nog steeds exponentieel tot het punt waarop sommige van de meest de belangrijke aspecten van ons leven *online* worden geregeld.

Deze honger naar informatie vraagt elke dag meer en meer van onze infrastructuur. Tot nu toe is deze vraag gepaard gegaan met een toename in rekenkracht die, volgens de wet van Moore, stelt dat ongeveer om de 18 maanden het aantal transistors — en dus het rekenvermogen — op een computerchip verdubbelt. Het einde van deze wet is echter in zicht, omdat we tegen de fundamentele grenzen aan de grootte van de transistors stoten. Ondertussen groeit de vraag naar informatie nog steeds aan een oogverblindend tempo. Een compleet nieuw rekenmodel — vooral voor telecom — is nodig.

1 Fotonisch neuromorf rekenen

Inspiratie voor dit nieuwe computermodel kan gevonden worden door te kijken naar onze eigen hersenen, die berekeningen volledig anders doen vergeleken met traditionele computers. Zo een model heeft daarom de mogelijkheid om bepaalde taken een grootteorde efficiënter uit te voeren, denk bijvoorbeeld aan *autorijden* of *gezichten herkennen*.

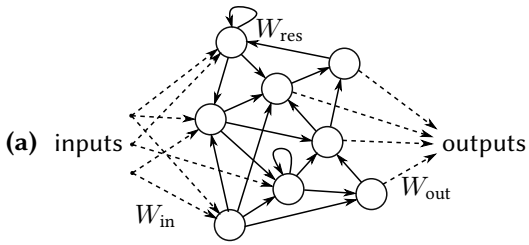
Typische neuromorfe computers volgen dezelfde basisstructuur als onze hersenen: neuronen die onderling verbonden zijn door synapsen. Deze breingeïnspireerde *neurale netwerken* zijn zeer succesvol in software, waar ze naar een geheel nieuw gebied van machine learning genaamd *deep learning* leidden. Ondanks hun successen zijn deze software-gebaseerde neurale netwerken echter fundamenteel nog steeds gebonden aan dezelfde limieten van traditioneel computergebruik.

In dit doctoraat proberen we *fotonica* te gebruiken om deze *neuromorfe* structuren te maken in *hardware*. De keuze voor fotonica heeft twee redenen. Allereerst maakt licht als drager voor informatie ultrasnelle berekeningen mogelijk ("aan de snelheid van het licht"). Ten tweede richten we ons op *telecom*-applicaties, waarvoor de meeste signalen zich al in het optische domein bevinden. Het is daarom logisch om de berekening optisch te houden om het aantal elektro-optische conversies te beperken.

In dit werk zullen we bestaande fotonische neuromorfe architecturen verbeteren, met de nadruk op het uitbreiden van de *fotonische reservoircomputer* naar meer algemene neuromorfische architecturen. We zullen ook de interessante eigenschappen van licht beter proberen te benutten door bijvoorbeeld quasi instantane analoge vermenging van het licht in photonische caviteiten, alsook interessante niet-lineaire interacties in fotorefractieve materialen te gebruiken.

1.1 Reservoir computers

Het idee van een reservoircomputer, zoals geïllustreerd in Fig. 1, is gebaseerd op een zeer dynamisch systeem — het *reservoir* — dat een tijdsafhankelijk ingangssignaal transformeert door temporele en ruimtelijke menging naar een hoogdimensionale interne reservoirtoestand. Deze toestand wordt dan *geïnterpreteerd* door de *readout*, die in feite een lineaire combinatie op de interne toestand uitvoert. De gewichten die de lineaire combinatie bepalen zijn dus geoptimaliseerd om opkomende patronen in het reservoir te herkennen in plaats van in het signaal zelf.



Figuur 1: Een laag-dimensionaal tijdsafhankelijk ingangssignaal wordt gemengd met vroegere versies van zichzelf vanwege de zeer dynamische architectuur van het reservoir. Telkens wordt (een deel van) de reservoirstatus uitgelezen om een tijdsafhankelijke voorspelling te maken.

Het reservoir — dat in feite werkt als een preprocessor voor de uitlezing — heeft twee fundamentele eigenschappen nodig om te correct te werken: de genoemde spatio-temporale *vermenging* en een soort van *vervagend geheugen*, wat betekent dat het asymptotisch eerdere signalen moet vergeten. Merk op dat deze vereisten niet erg beperkend zijn en dat er dus een hele reeks verschillende reservoirimplementaties bestaan, variërend van software reservoirs tot zeer performante hardwarereservoirs in elektronica [1]. In deze thesis ligt de focus op optisch geïmplementeerde reservoirs.

[1] Zelfs reservoircomputers die gebruik maken van een emmer water als reservoir zijn al gerapporteerd.

1.2 Optimalizatie van het reservoir

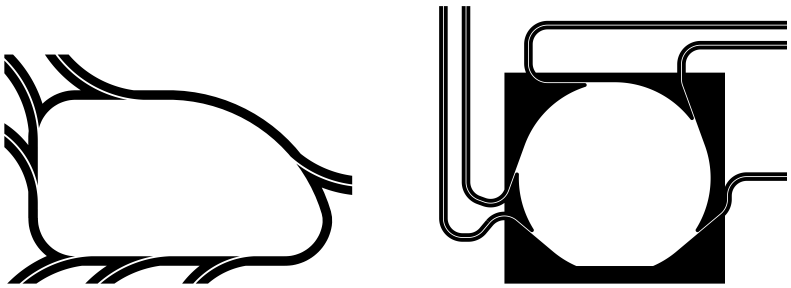
Hoewel normaal gezien het reservoir niet wordt geoptimaliseerd, zijn er toch enkele voordelen van *enige* optimalisatie in het reservoir. Bovendien laten huidige optimalisatietechnieken die zijn voortgekomen uit de snelle groei op het gebied van *deep learning* toe om gemakkelijker dergelijke zeer recurente systemen te optimaliseren.

Door *Photontorch*[2], een op maat gemaakte fotonische circuitsimulator, te gebruiken kunnen deze technieken tijdens de simulatie en optimalisatie van de fotonische circuits worden geleend. Dit laat toe om te experimenteren met uitbreidingen op de traditionele fotonische reservoircomputer.

1.3 Caviteiten voor continue mixing in het reservoir

Een andere uitbreiding op de traditionele reservoircomputer kan ontstaan door een meer *continu* mengen van het ingangssignaal toe te staan. Een traditionele reservoircomputer maakt gebruik van de standaard neuromorfe architectuur: *neuronen* die verbonden zijn door *synapsen*. Licht laat echter wat meer esoterische structuren toe, zoals caviteiten, waar het licht vrij is om te mengen in een meer continue manier, waardoor een heel complexe vermenging van het ingangssignaal mogelijk wordt.

Een typische caviteit die voor dit doel kan worden gebruikt, moet een grote spatio-temporale *vermenging* van het ingangssignaal vertonen. Mogelijke vormen met deze eigenschap zijn bijvoorbeeld de zogenaamde *kwartstadium* of een circelvormige schijf met een recht stuk zoals weergegeven in Fig. 2.



(a) Kwartstadium **(b)** Schijf

Figuur 2: Twee caviteiten die interessante menging vertonen. Beide vormen zijn gebaseerd op indexcontrast: een eenvoudige geul werd geëtst rond de caviteit om het licht binnen te houden.

[2]Een portmanteau voor “Photon” en “PyTorch”, de machine learning backend die deze simulator gebruikt.

1.4 Optimalizeerbare reservoircaviteiten

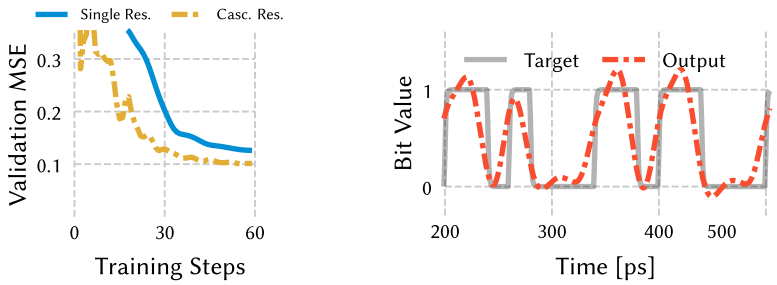
Beide uitbreidingen van reservoir computing waar we naar op zoek zijn kunnen in feite *gecombineerd* worden. Inderdaad, met behulp van een niet-lineair effect, zoals bijvoorbeeld het *photorefractieve effect*, kan enige *plasticiteit* in de caviteiten worden ingebouwd. Om dit aan te tonen gebruiken we een aangepaste FDTD simulator, gebouwd om het fotorefractieve effect nauwkeurig te kunnen simuleren. We zien dat deze fotorefractieve caviteiten zichzelf reorganiseren volgens de doorgestuurde lichtpatronen. Deze reorganisatie kan worden beschouwd als een vorm van zelf-leren.

2 Resultaten

Omdat de genoemde architecturen bedoeld zijn om te worden gebruikt in telecom, werden hun prestaties gemeten op telecom-gerelateerde taken, zoals header herkenning en Booleaanse logica.

2.1 Aaneengeschakelde reservoirs

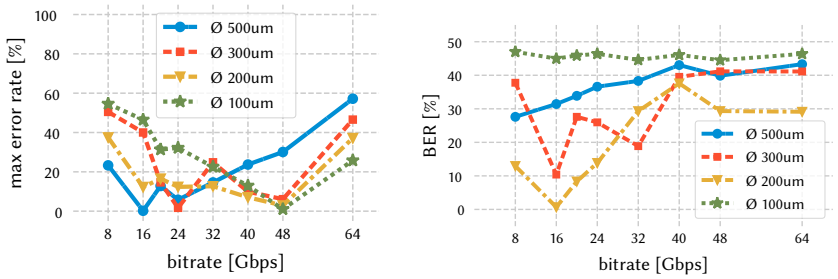
De meest voor de hand liggende uitbreiding op de traditionele reservoirarchitectuur kan worden bereikt door twee reservoirs te combineren. We gebruiken de Photontorch-simulator om zowel de readout als de tussenverbindingen tussen de reservoirs te optimalizeren voor een betere prestatie op de XOR-taak. De prestatieverbetering door het gebruik van twee aaneengeschakelde reservoirs tijdens training kan gezien worden in Fig. 3a. Een typisch signaal dat de readout verlaat wordt getoond in Fig. 3b.



Figuur 3: (a) Leercurve voor het trainen van de readout voor een alleenstaand reservoir vergeleken met de leercurve voor het trainen van de readout *en* de tussenverbindingen van twee aaneengeschakelde reservoirs. (b) Resulterend uitgangssignaal voor het aaneengeschakelde reservoir voor de XOR-taak.

2.2 Fotonische caviteiten

De fotonische caviteiten getoond in Fig. 2 werden gemaakt in veel verschillende formaten op het 220 nm siliciumfotonicaplatform. De prestaties voor elk van de vervaardigde caviteiten op een 3-bit headerherkenningstaak en op de XOR-taak werd gemeten, zoals geïllustreerd in Fig. 4. De resultaten laten zien dat de schijf-caviteit 3-bit headers kan herkennen voor ten minste één bitrate voor elke geteste diameter van de schijf. De XOR-taak lijkt echter moeilijker te zijn voor de caviteiten: alleen de schijf met een diameter van 200 μm kan deze taak uitvoeren.



(a) **(b)**

Figuur 4: Gemeten prestaties van fotonische caviteiten van verschillende groottes en bij een groot bereik van bitrates op (a) de 3-bit headerherkenningstaak en (b) de XOR-taak.

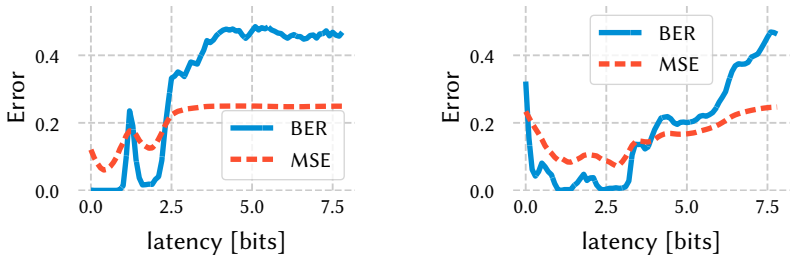
2.3 Fotorefractief reservoir

Het gebruik van fotorefractieve kristallen voor neuromorfisch computergebruik is gemotiveerd door het feit dat dit effect mogelijk de prestaties van het reservoir kan verbeteren. Door het kristal in een caviteit te plaatsen en er een signaal door te sturen zullen interferentie-effecten beginnen op te bouwen. Het idee is nu dat verschillende bit-deelreeksen van het ingangssignaal op verschillende manieren met elkaar zullen interfereren, wat resulteert in een kristal dat zichzelf reorganiseert om deze reeksen beter te herkennen.

Deze systemen werden gesimuleerd met een speciale FDTD-simulator ontwikkeld om specifiek het fotorefractieve effect te kunnen simuleren. In Fig. 5 is duidelijk te zien dat het *primen* van het kristal met zo een bitstroom het werkingsbereik van het systeem verbreedt en dus het geheugen vergroot.

3 Conclusies

Concluderend hebben we laten zien dat verder gaan dan de traditionele fotonische reservoircomputer, door de definitie van een fotonisch reservoir uit te brei-



(a) Before priming **(b)** After priming

Figuur 5: Prestaties van het fotorefractief reservoir op de XOR-taak voor en na het primen.

den om caviteiten er in op te nemen *of* door enige optimalisatie toe te staan in het reservoir, de prestaties op verschillende telecom-gerelateerde taken verbetert.

Concreet toonden we aan dat *aaneengeschakelde* fotonische reservoirs met een optimaliseerbare tussenlaag beter presteren op de XOR-taak. We konden deze prestatieverhoging alleen aantonen dankzij onze speciale fotonische circuit optimizer, Photontorch, waarmee een zeer efficiënte optimalisatie van de tussenliggende verbindingsgewichten mogelijk werd.

Bovendien, als alternatief voor de op neuronen gebaseerde architecturen, experimenteerden we met reservoircomputing op basis van fotonische caviteiten. Deze caviteiten vertonen een *continu* mengen in tegenstelling tot een meer (ruimtelijk) discreet mengen binnen de knooppunten van een netwerk. We hebben experimenteel aangetoond dat deze caviteiten een levensvatbaar platform bieden voor hoge snelheid fotonische reservoircomputing in het telecomveld. Ten slotte hebben we aangetoond dat het fotorefractieve effect kan benut worden om het kristal te *primen* om terugkerende patronen in willekeurige bitreeksen beter te herkennen.

English Summary

In this day and age of the internet, we are living in a world that completely relies on digital information processing. The amount of information at our disposal and being exchanged is virtually unlimited and still growing exponentially to a point where major aspects of our life take place *online*.

However, this rapid growth in appetite for information demands more and more from our infrastructure every day. So far this demand has been matched by an increase in computing power according to Moore's law, which states that about every 18 months the number of transistors — and hence the computing power — on a computer chip doubles. However, this law is starting to crumble, as fundamental limits on the size of the transistors are being reached. Meanwhile, the demand for information keeps growing at a dazzling pace. A new computing paradigm — especially for telecom — is needed.

1 Photonic neuromorphic computing

Inspiration for this new computing paradigm can be found by looking at our own brain, which does computations completely differently compared to computers and hence has the ability to do certain tasks orders of magnitude more efficiently, for example *driving a car* or *recognizing faces*.

Typical neuromorphic computers follow the same basic structure of our brain: neurons interconnected by synapses. These brain-inspired *neural networks* have been very successful in software, giving rise to a whole new field of machine learning called *deep learning*. However, despite their successes, these in-software neural networks are fundamentally still bound by the same limits of traditional computing.

In this PhD we attempt to use *photonics* to create these *neuromorphic* structures in *hardware*. The choice for photonics has two reasons. First of all, using light as a carrier for information enables ultra-fast computations ("at the speed of light"). Second, we will be targeting *telecom* applications, for which most of the signals are already in the optical domain. It makes therefore sense to keep the computation optical to reduce the number of electro-optical conversions.

We will improve on existing photonic neuromorphic computing paradigms, with the main focus on extending the principle of *photonic reservoir computing* to more general neuromorphic architectures. We will also try to better better exploit the interesting properties of light, by using it's quasi instantaneous mix-

ing in photonic cavities as well as using the interesting nonlinear interactions in photorefractive materials.

1.1 Reservoir computing

The idea of a reservoir computer, as illustrated in Fig. 6, is based on a highly dynamical system — the *reservoir* — which transforms a time-dependent input signal through temporal and spatial mixing to a high-dimensional internal reservoir state. This state is then *interpreted* by a *readout*, which basically performs a linear combination on this internal state. The weights defining the linear combination are thus optimized to recognize emerging patterns in the reservoir instead of in the signal directly.

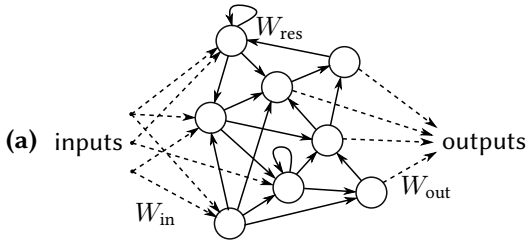


Figure 6: A low-dimensional time-dependent input signal gets mixed with previous versions of itself due to the highly dynamic architecture of the reservoir. At each time, (a part of) the reservoir state is read out by the readout to make a time dependent prediction

The reservoir — which basically acts as a preprocessor to the readout — needs two fundamental properties to operate properly: the mentioned spatio-temporal *mixing* and a *fading memory*, which means that it should asymptotically forget past inputs. Notice that these requirements are not very restrictive and hence a whole range of different reservoir implementations exist, ranging from in-software reservoirs to highly performant hardware reservoirs in electronics[3]. In this thesis, the focus lies on optically implemented reservoirs.

1.2 Optimizing the reservoir

Although the main idea of reservoir computing is that the reservoir itself is not being optimized, some advantages from allowing *some* optimization in the reservoir still exist. Moreover, current optimization techniques which have arisen from the rapid growth in the field of *deep learning* have made it easier to optimize such highly recurrent systems.

By using a custom-built photonic circuit simulator, which we call *Photontorch*[4], these optimization techniques can be borrowed during the simulation and

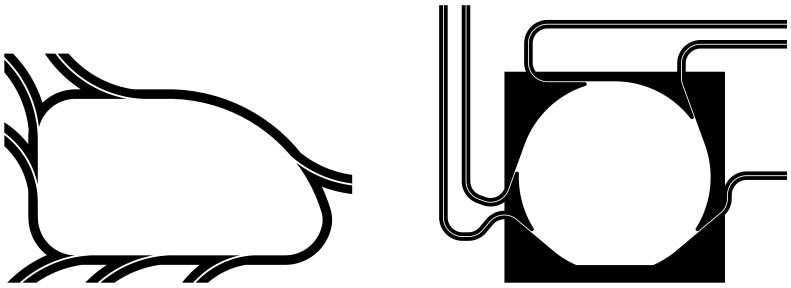
[3] Even reservoir computing using a bucket of water has already been reported.
[4] A portmanteau for "Photon" and "PyTorch", the machine learning backend this simulator uses.

optimization of photonic circuits, enabling experimenting with extensions to the photonic reservoir computer paradigm.

1.3 Cavities for continuous reservoir mixing

Another extension to the traditional reservoir computer is allowing a more *continuous* mixing between the input states. A traditional reservoir computer uses the standard neuromorphic architecture: *neurons* connected recurrently by *synapses*. However, light allows for some more esoteric structures, such as cavities, where light is free to mix in a more continuous matter after which very complex mixing of the input signal becomes possible.

A typical cavity to be used for this purpose should have high spatio-temporal mixing. Possible shapes like this are for example the so-called *quarter-stadium* or a disk-shaped cavity with a chamfer as illustrated in Fig. 7.



(a) Quarter stadium **(b)** Disk with chamfer

Figure 7: Two cavity shapes that induce interesting mixing dynamics. Both shapes are based on index contrast: a simple trench was etched around the cavity to keep the light in.

1.4 Optimizable reservoir cavities

Finally, both extensions of reservoir computing that we are looking into can in fact be *combined*. Indeed, using a non-linear effect, like the *photorefractive effect*, some *plasticity* can be built into the cavity. We use a custom FDTD simulator, built to accurately simulate the photorefractive effect, to show in a theoretical viability study that photorefractive cavities reorganize themselves according to the light patterns sent through them. This reorganization can be considered a form of self-learning.

2 Results

As the said architectures are intended to be used in telecom, their performance was measured on telecom-related tasks, such as header recognition and Boolean

logic.

2.1 Cascaded reservoirs

The most straightforward extension to the traditional so-called swirl reservoir can probably be achieved by cascading two of them together. We use the custom Photontorch simulator to optimize both the readout and the intermediate connections for better performance on the XOR task. The performance increase from using two cascaded reservoirs during training is visualized by the learning curve in Fig. 8a. A typical output for the cascaded reservoir after training is visualized in Fig. 8b.

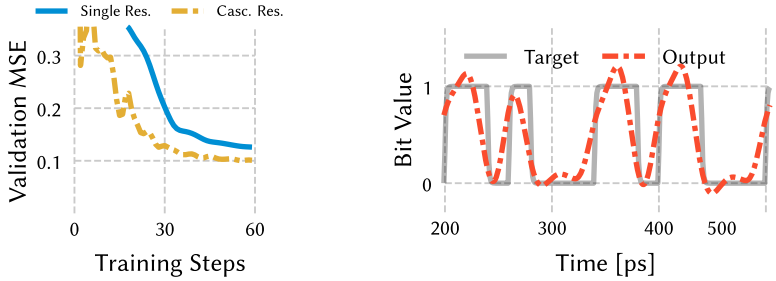


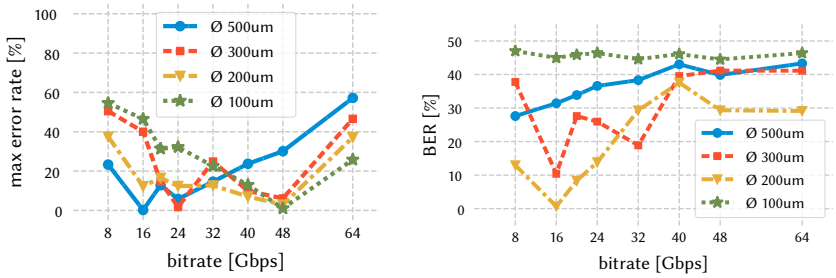
Figure 8: (a) Learning curve for training the readout of a single reservoir compared to the learning curve for training *both* the readout *and* intermediate connections in the cascaded reservoir on the XOR Task (b) Resulting outputs for the cascaded reservoir for the XOR target.

2.2 Photonic cavities

The photonic cavities shown in Fig. 7 were fabricated for many different sizes on the 220 nm silicon photonics platform. The performance for each of the fabricated cavities on both a 3-bit header recognition task and on the XOR task was tracked, as illustrated in Fig. 9. The measurements show that the chamfer cavity was able to perform the 3-bit header recognition task for the whole range of diameters at at least one bitrate. The XOR-task seems to be more difficult for these cavities: only the 200 μm diameter chamfer cavity can perform this.

2.3 Photorefractive reservoir

Using photorefractive crystals for neuromorphic computing is motivated by the fact that this effect can possibly be exploited for better reservoir performance. By placing the crystal in a cavity and sending a signal through the cavity and the crystal, interference effects start to build up. The idea is now that different bit

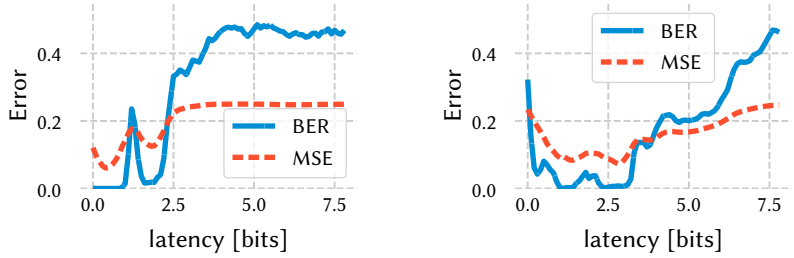


(a) **(b)**

Figure 9: Measured performance of photonic cavities of different sizes and at a large range of bitrates on (a) the 3-bit header recognition task and (b) the XOR task.

subsequences of the signal beam will interfere with each other in different ways, resulting in a crystal that reorganizes itself towards better recognizing these sequences.

These systems were simulated with a dedicated FDTD simulator, which was developed to specifically be able to simulate the photorefractive effect. In Fig. 10 it can clearly be seen that *priming* the crystal first with such a bit stream widens the operation range (increases the memory) of the system.



(a) Before priming **(b)** After priming

Figure 10: Performance of the photorefractive reservoir on the XOR task before and after priming.

3 Conclusions

In conclusion, we showed that going *beyond* traditional photonic reservoir computing, by either expanding the definition of a photonic reservoir to include cavities *or* by allowing some optimization in the reservoir, improves the performance

on several telecom-related tasks.

Concretely, we showed that *cascaded* photonic reservoirs with an optimizable intermediate layer perform better on the XOR task. We were only able to show this increase in performance due to our dedicated photonic circuit optimizer, Photontorch, which enables very efficient optimization of the intermediate connection weights.

Moreover, as an alternative to the node-based architecture, we experimented with photonic reservoir computing based on photonic cavities. These cavities exhibit a *continuous* mixing in contrast to the more (spatially) discrete mixing inside the nodes of a network. We experimentally showed that these cavities provide a viable platform for high-speed photonic reservoir computing in the telecom field. Finally, we showed that the photorefractive effect can be exploited to *prime* the crystal to better recognize returning patterns in random bit sequences.

0 Introduction

In this day and age we are living in a world that completely relies on digital information processing. According to Moore's law, digital computing power has been doubling every 18 months since the invention of the transistor in the 1960s. However, with the advent of the Internet, the bandwidth requirements on our infrastructure have been catching up at a dazzling pace while Moore's law is starting to falter.

This race between a stagnating computing power and ever increasing bandwidth requirements has sparked renewed interest in alternative computing paradigms for a different, smarter and faster processing scheme of data. One of these alternative computing paradigms is *analog* or *physical computing*, which relies on using the analog response of a physical system to an appropriately encoded input signal. The idea here is that the physical system is used for the computationally intensive part of the calculation at hand, while a simple electronic circuit can then be used for translating the analog response back into a digital format.

A popular implementation of these physical computers relies on physical architectures which are inspired by the *human brain*: instead of the traditional transistors used in digital computing, these *neuromorphic* computers consist of *neurons* interconnected by *synapses*. These *artificial neural networks* thus rely on a more biologically inspired way of computing which is in stark contrast to the very systematic and deterministic approach used in digital computers.

To see the possible advantages of this approach, one only has to observe the vast amount of tasks a human is good at, such as recognizing faces, driving a

car and listening to conversations at a busy party with lots of background noise, to just name a few. All these tasks which come naturally to us are incredibly difficult for traditional computers. The main reason for this is that it is very hard to break down what we are actually doing in clear step-by-step tasks required to program digital computers.

Indeed, what is actually happening in our brain is much more primitive: we do not distinguish a cat from a dog by following a flowchart, we just *know*: we have seen cats and we have seen dogs before in our life and either animal activates our brain slightly differently. It is very hard to translate this into a logical decision making process as no such explicit process ever took place in our brain.

The promise of neuromorphic computing being successful in totally different areas from traditional computing has attracted the attention of many researchers since the 1970s. However, it took until the early 2010s before the first impressive developments started to appear. Ironically, this was for neural networks implemented in software on digital computers...

This branch of machine learning has taken traditional computing by storm and the results are impressive. Image recognition? A computer can do it better than you. Games like chess, Go or even Poker? A computer beats you at it. Recognizing your friend's voice? Alexa or Siri can do it too. Soon, cars will not need drivers anymore and packages will be delivered at your doorstep by an unmanned drone.

These achievements all serve to show that this new computing paradigm is very successful and is here to stay. However, these in-software *deep neural networks* are still constricted by the same limits on Moore's law. Although the massive parallel nature of these *deep neural networks* allows them to be easily parallelized on Graphical Processing Units (GPUs) — which probably buys them a few more years — they too will eventually reach their limits. Moreover, *simulating* such *analog* systems like neural networks on a *digital* computer is inherently inefficient and leads to immense power consumption[1].

This all leads to the question if these inherently *analog* neural networks should be implemented on *digital* computers at all. In this thesis we will try to tackle this problem head on. In fact, we'll do this by applying neuromorphic architectures to solve the mentioned scaling issue in telecom: how can analog systems, like artificial neural networks, be used to increase the speed and efficiency of common telecommunication operations such as header recognition and boolean logic.

The most obvious information carrier for these tasks is of course light. First

[1] AlphaZero, Google's AI that beats *both* Go-players and chess players needs about 250 GPUs, just for *playing* the game (*inference*). At a power consumption of about 300 W per GPU, this results in a total power consumption of 75 kW, about 4000 times more than what a human brain needs (20W). This is not even considering the training of this massive neural network, which on its own consumes even *more* than inference (playing the game).

of all because most telecommunication signals are already in the optical domain, so it makes sense to keep them there. Second, light propagation is a massively parallel scheme which allows to easily perform the analog operations necessary to mimic a neural network. Additionally, light has an amplitude and a phase, giving two independent degrees of freedom that can be exploited for better computations. Moreover, the available bandwidth of a photonic chip is several orders of magnitude larger than that of an equivalent electronic chip as it is not limited by the movement of charge carriers. Finally, some materials allow for very fast optical non-linearities that can possibly be exploited.

The way to implement these photonic neural networks is *less clear*. Over the years, several different implementations have been tried and tested, ranging from photonic activation functions [1] to photonic *deep* neural networks [2], to photonic reservoirs computers [3–9]. This thesis aims to expand on these artificial neural network implementations by searching for new architectures that are effective for the mentioned telecom applications.

The most obvious choice for optical neural networks would be *integrated silicon photonics*, which in addition to the mentioned advantages of using light as an information carrier also benefits from a very small footprint (typical photonic chips have a footprint of the order of a few mm^2) and an incredibly mature CMOS fabrication platform.

However, even on the silicon photonics platform there are several possibilities for implementing neural networks on a chip. One could stick to primitive photonic elements like splitters and combiners to construct a similar mapping of the structure of our brain, consisting of neurons and synapses. However, light allows for some more esoteric structures such as cavities, where light is free to mix in a more continuous matter at no computational overhead.

Moreover, if one looks beyond the silicon photonics platform, one gains access to a whole new world of optical effects that cannot be exploited on a silicon chip. One of these effects is the non-linear *photorefractive* effect, which we will also briefly touch upon in later chapters. Obviously, leaving the integrated photonics platform to incorporate photorefractive materials also introduces new challenges: slower operating speeds and less mass-producible structures to name the most important ones.

1 Optical Information Processing

We are currently living in an age of *Big Data*. The amount of information at our disposal and being exchanged is virtually unlimited and still growing exponentially. Data-heavy services like *Netflix*, *YouTube* and *Skype* demand more and more of our infrastructure every day.

A large part of this growing information exchange relies on an ever expanding optical infrastructure for telecommunication. Indeed, today we experience instant communication over 1000s of miles across the globe. Light is an indispensable factor in this link: from large fiber-optical cables connecting continents on the bottom of the ocean to small on-chip optical interconnects increasing the operation speed of traditionally electronic chips.

The fact that fiber optical communication can take advantage of huge bandwidth, low electromagnetic crosstalk and low power loss is of course well known. Today fiber-optical communication links often reach data transmission rates well above 100 Gbps. These high optical data transmission rates have had a direct influence on the importance of on-chip optical interconnects.

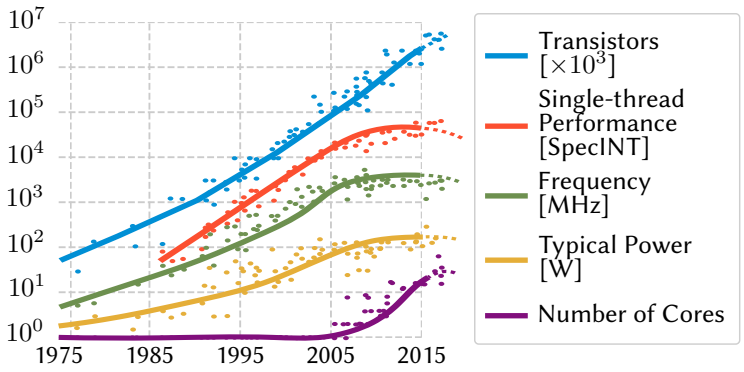


Figure 1: Moore's law is stagnating. Single-thread computing power has not been increasing since about 2010, while the sole reason the number of transistors has kept on increasing is by the increasing number of cores per chip (parallelization). Original data from [10]

Indeed, up until recently, we have been able to match the growing demand in bandwidth by an ever increasing processing power in our computers. This exponential growth in computing power, known as Moore's law, states that roughly every 18 months the number of transistors on a computer chip doubles, as is illustrated in Fig. 1. However, to accommodate this increasing chip-density — which currently in 2019 lies at about 10 billion transistors per chip — these transistors have to be made smaller and smaller to the point where they have reached their quantum limit: making them any smaller will make their bit-value unstable due to thermal effects.

This fundamental limit on the processing power of electronics impedes the processing of the incoming optical signals beyond certain data transmission rates and hence alternatives — preferably in the optical domain — have to be found.

A lot has already been improved in the optical domain. Over the years the

optical communication research has reduced the fiber losses, reduced the cost of optical sources and amplifiers and so on. However, a large part of optical communication still relies on various manipulations in the electrical domain to ensure correct routing, conditioning and alignment. Traditionally these manipulations are carried out by converting the light into an electrical signal, which is then electronically processed by a Digital Signal Processor (DSP) and subsequently converted back into the optical domain if needed.

Apart from the obvious power considerations related to this transformation to the electrical domain and back, there are also the fundamental scaling issues for digital processing that limit the DSP technology to signals well below the limits of optical communication[2].

It is therefore key that optical methods are found to do the most common optical signal processing tasks related to optical telecommunication. Using photonic neuromorphic computing can possibly allow for such all-optical signal processing, removing most bandwidth limitations by keeping the processing entirely in the optical domain.

2 Silicon Photonics

The choice for silicon as an optical medium is not so much guided by its optical properties but rather by the advanced CMOS[3] fabrication techniques it enables. Indeed, CMOS fabrication is an incredibly mature manufacturing technique used in micro electronics to create electronic structures with sub-micrometer precision. Borrowing this technology enables the manufacturing of nanophotonic structures with the same precision, offers a cost-effective way of mass producing silicon photonic chips and at the same time opens up possible opportunities for co-integration of electronics and photonics on the same silicon chip.

However, silicon photonics would not be a viable platform if silicon was not at least *a bit* transparent at optical wavelengths used for telecommunication. Silicon is transparent for infrared light in the so-called C-band around $1550\,\mathrm{nm}$ and the O-band around $1300\,\mathrm{nm}$. Although the losses at these wavelengths are still a few $\mathrm{dB/cm}$ — a few orders of magnitude larger than losses in optical fibers — the small size of these silicon photonic structures makes up for a lot.

To create light-guiding structures on a silicon chip, trenches need to be etched on a $220\,\mathrm{nm}$ silicon-on-insulator (SOI) slab as illustrated in Fig. 2. By sending light through such an entrenched structure, light gets laterally trapped and is *guided* forward creating a *waveguide*. It turns out that the guiding properties

[2] As a comparison: digital signals in the electrical domain become notably difficult to process beyond 20 Gbps, while single-wavelength optical data transmission rates easily reach 100 Gbps and more.

[3] CMOS stands for Complementary Metal-Oxide-Semiconductor, which is the most common type of micro-transistor used today.

rely heavily on the width of the waveguide, for which *single-mode* propagation lies around $450\,\text{nm}$. Slight manufacturing variations can have a measurable influence on the phase of the mode through surface roughness and hence high manufacturing precision is required for most applications.

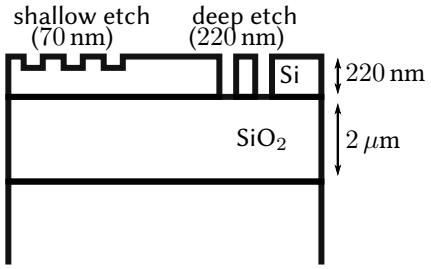


Figure 2: Most silicon photonic chips consist of structures patterned on a $220\,\text{nm}$ thick silicon-on-insulator (SOI) structure. Generally speaking, deeply etched trenches in the Si layer create light-guiding structures called *waveguides*, while shallow etched trenches are used to couple light into the waveguides from above the chip.

3 Photonic Reservoir Computing

Reservoir Computing (RC) is a well-established machine-learning concept, first proposed in the early 2000s [11, 12] as a brain-inspired computing mechanism to process temporal signals in real time. In RC, a highly dynamical and non-linear system is used as a signal-mixing *reservoir*, which essentially produces a number of non-linearly mixed versions of an input signal. Those resulting *reservoir states* are then *read out* by performing an application-dependent linear combination of these states [13].

In the case of *photonic* RC, this reservoir can be created by interconnecting many optical *neurons* (often called *nodes* in this context). While traditionally, those neurons are implemented by a non-linear activation function, previous work has shown that, in the case of light, a non-linear detection operation at the readout suffices. This is a direct result of the fact that light carries an amplitude and a phase, resulting in a non-linearly mixed cross-term at the readout [9].

We have shown before that this conceptually simple approach to reservoir design leads to fewer parameters to tune and more energy-efficient designs. The work in this thesis will pick up where the research has brought us so far and look in which ways this traditional passive design can be improved by possibly cascading or micro-optimizing the photonic reservoir. To do these micro-optimizations, a dedicated simulator will be necessary which can optimize (back-propagate) through the physical structure of the reservoir itself.

Then, a different approach will be followed, where the concept of a passive reservoir which follows the traditional neuronal structure of our brain (neurons connected by synapses) is expanded to a more continuous representation using photonic cavities. In these cavities, the light is free to mix in a more *continuous* manner at no computational overhead.

4 Photorefractive effect

As an alternative medium for optical neuromorphic computing, photorefractive materials will be explored. As the photorefractive effect is mostly a phenomenon appearing in bulk dielectric materials, exploring this route requires us to look beyond silicon photonics.

The photorefractive effect relies on a careful interplay between photons and electrons. A photorefractive material consists of a crystalline structure with many photon traps present. Photons entering the photorefractive material will thus very often excite electrons which will become free to diffuse through the material. As on average more electrons are freed at locations of high light intensity, the global tendency of the electrons will be to move towards locations of lower light intensity where the chance of being trapped is higher. This means that if an interference pattern is applied in the material, a permanent displacement of charges can be created which in turn creates a quasi permanent space-charge field which will remain present, even when the interference pattern is removed. When the material is then also susceptible to an electro-optical effect like the Pockels or Kerr effect, this material is said to be *photorefractive*, as due to the *photon*-induced space-charge field a *refractive* index profile can be created.

This phenomenon is obviously very interesting for several reasons. The first of which is the *plasticity* of the material: different *holograms* can be imprinted into the photorefractive crystal for different applications bringing us closer to an all-optical multi-functional processing unit. Moreover, the 3D structure of the holograms allows for a much richer interconnection topology that far exceeds any structure that can be made on a 2D chip.

However, an expensive price has to be paid for these advantages: the ease and cost of manufacturing.

5 Objectives

As we have laid out before, one of the major goals of this thesis is replacing the DSP with an all-optical processor. Due to the highly analog nature of light, one promising way to achieve this is by adopting a neuromorphic architecture on a silicon photonics chip.

How this architecture is supposed to look is one of the main items that will be looked into in this dissertation. We will focus in particular on simple feed-forward structures pre-trained in software, as well as on recurrent reservoir architectures and photonic cavities all integrated on-chip. However, also photorefractive systems will be considered due to their interesting plasticity which possibly can be exploited for training.

Moreover, to efficiently obtain and test out new designs, a machine-learning driven design philosophy will be followed. For this, a completely new photonic simulator was created, which allows to efficiently simulate *and* optimize arbitrary on-chip photonic circuits through a machine learning algorithm called *backpropagation*.

An obvious advantage of these on-chip designs is a possible co-integration with the electronics into a opto-electronic transceiver. Therefore, especially for these kind of structures the *reservoir computing* approach can be particularly interesting as that approach relies on a simple electronic readout circuit that can relatively easily be implemented on a chip.

6 Thesis outline

As this thesis is about novel photonic neuromorphic computing architectures, we will gradually build up the content towards more complex unconventional ways of doing neuromorphic computing with light.

Chapter 1 covers the basics of machine learning (ML) and neuromorphic computing. The goal of this chapter is to introduce most of the prerequisites that will be used in later chapters.

Then, in chapter 2, we will cover the simulation and efficient optimization through *backpropagation* of large photonic circuits. For this, a dedicated simulator — Photontorch [14] — was designed. We will also use this simulator to improve the *photonic swirl reservoir* [9] on certain benchmark tasks.

In chapter 3, we will then cover the simulation of on-chip photonic cavities used in the context of reservoir computing. We will simulate their performance on the same benchmark tasks as before and compare it with experimental results.

Finally, in chapter 4, we will explore how *photorefractive* crystals can be used to do neuromorphic computing. The photorefractive effect in these materials enables some kind of plasticity in the reservoir: the reservoir is able to learn For the photorefractive effect to be strong enough, however, these systems have to be considered *in bulk*, i.e. not on-chip.

7 Publications

Publications in international journals

[1] Floris Laporte, Andrew Katumba, Joni Dambre, and Peter Bienstman. *Numerical demonstration of neuromorphic computing with photonic crystal cavities.* Optics express, 26(7):7955–7964, 2018.

[2] Floris Laporte, Joni Dambre, and Peter Bienstman. *Highly parallel simulation and optimization of photonic circuits in time and frequency domain based on the deep-learning framework PyTorch.* Scientific reports, 9(1):5918, 2019.

[3] Andrew Katumba, Matthias Freiberger, Floris Laporte, Alessio Lugnan, Stijn Sackesyn, Chonghuai Ma, Joni Dambre, and Peter Bienstman. *Neuromorphic computing based on silicon photonics and reservoir computing.* IEEE Journal of Selected Topics in Quantum Electronics, 24(6):1–10, 2018.

[4] Chonghuai Ma, Floris Laporte, Joni Dambre, and Peter Bienstman. *Addressing Limited Weight Resolution in a Fully Optical Neuromorphic Reservoir Computing Readout.* arXiv preprint arXiv:1908.02728, 2019.

[5] Alessio Lugnan, Andrew Katumba, Floris Laporte, Matthias Freiberger, Stijn Sackesyn, Chonghuai Ma, Joni Dambre, and Peter Bienstman. *Photonic Neuromorphic Information Processing and Reservoir Computing.* APL Photonics *(accepted)*, 2020.

Publications in international conferences

[6] Floris Laporte, Joni Dambre, and Peter Bienstman. *Photonic Reservoir Computing (invited).* Photonic Integration Week, 2020

[7] Floris Laporte, Joni Dambre, and Peter Bienstman. *Photontorch: Simulation and Optimization of Large Photonic Circuits Using the Deep Learning Framework PyTorch.* In 2019 IEEE Photonics Society Summer Topical Meeting Series (SUM), pages 1–2. IEEE, 2019.

[8] Floris Laporte, Joni Dambre, and Peter Bienstman. *Neuromorphic Computing with Signal-Mixing Cavities.* In 2018 IEEE International Conference on Rebooting Computing (ICRC), pages 1–4. IEEE, 2018.

[9] Peter Bienstman, Joni Dambre, Andrew Katumba, Matthias Freiberger, Floris Laporte, and Alessio Lugnan. *Photonic reservoir computing: a brain-inspired approach for information processing (invited).* In Optical Fiber Communication Conference, pages M4F–4. Optical Society of America, 2018.

[10] Peter Bienstman, Joni Dambre, Andrew Katumba, Matthias Freiberger, Floris Laporte, and Alessio Lugnan. *Silicon photonics for neuromorphic information processing*. In Optical Data Science: Trends Shaping the Future of Photonics, volume 10551, page 105510K. International Society for Optics and Photonics, 2018.

[11] Floris Laporte, Alessio Lugnan, Joni Dambre, and Peter Bienstman. *Novel photonic reservoir computing architectures*. In Workshop on Dynamical systems and Brain-Inspired Information Processing, page 1, 2017.

[12] Andrew Katumba, Floris Laporte, Alessio Lugnan, Joni Dambre, and Peter Bienstman. *Integrated-photonics implementation of reservoir computing neural networks (invited)*. In Workshop Machine Learning@ ECOC 2017, pages 1–1, 2017.

[13] Floris Laporte, Joni Dambre, and Peter Bienstman. *Reservoir computing with signal-mixing cavities (invited)*. In 2017 19th International Conference on Transparent Optical Networks (ICTON), pages 1–4. IEEE, 2017.

[14] Floris Laporte, Joni Dambre, and Peter Bienstman. *Header recognition with signal-mixing cavities*. In Workshop on Dynamical Systems and Brain Inspired Computing, pages 1–1, 2017.

[15] Floris Laporte, Joni Dambre, and Peter Bienstman. *Photorefractive crystals as brain-inspired photonic reservoir computing systems*. In Proceedings Symposium IEEE Photonics Society Benelux, pages 151–154, 2016.

Patents

[16] Peter Bienstman, Floris Laporte, and Alessio Lugnan. *Mixing wave based computing*. Patent Application No. PCT/EP2018/063855, Filed May 26, 2018

[17] Peter Bienstman, Alessio Lugnan, and Floris Laporte. *Object classification system and method*. Patent Application No. PCT/EP2018/063854, Filed May 26, 2018

Book Chapters

[18] Andrew Katumba, Matthias Freiberger, Floris Laporte, Alessio Lugnan, Stijn Sackesyn, Chonghuai Ma, Joni Dambre, and Peter Bienstman *Integrated on-chip reservoirs*. Photonic Reservoir Computing: Optical Recurrent Neural Networks, De Gruyter, 2019.

References

[1] Alexander N Tait, Mitchell A Nahmias, Bhavin J Shastri, and Paul R Prucnal. *Broadcast and weight: an integrated network for scalable photonic spike processing.* Journal of Lightwave Technology, 32(21):4029–4041, 2014.

[2] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. *Deep learning with coherent nanophotonic circuits.* Nature Photonics, 11(7):441, 2017.

[3] Quentin Vinckier, François Duport, Anteo Smerieri, Kristof Vandoorne, Peter Bienstman, Marc Haelterman, and Serge Massar. *High-performance photonic reservoir computer based on a coherently driven passive cavity.* Optica, 2(5):438–446, 2015.

[4] Lennert Appeltant, Miguel Cornelles Soriano, Guy Van der Sande, Jan Danckaert, Serge Massar, Joni Dambre, Benjamin Schrauwen, Claudio R Mirasso, and Ingo Fischer. *Information processing using a single dynamical node as complex system.* Nature communications, 2:468, 2011.

[5] Yvan Paquot, Francois Duport, Antoneo Smerieri, Joni Dambre, Benjamin Schrauwen, Marc Haelterman, and Serge Massar. *Optoelectronic reservoir computing.* Scientific reports, 2, 2012.

[6] Laurent Larger, Miguel C Soriano, Daniel Brunner, Lennert Appeltant, Jose M Gutiérrez, Luis Pesquera, Claudio R Mirasso, and Ingo Fischer. *Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing.* Optics express, 20(3):3241–3249, 2012.

[7] Daniel Brunner, Miguel C Soriano, Claudio R Mirasso, and Ingo Fischer. *Parallel photonic information processing at gigabyte per second data rates using transient states.* Nature communications, 4:1364, 2013.

[8] Kristof Vandoorne, Wouter Dierckx, Benjamin Schrauwen, David Verstraeten, Roel Baets, Peter Bienstman, and Jan Van Campenhout. *Toward optical signal processing using Photonic Reservoir Computing.* Opt. Express, 16(15):11182–11192, Jul 2008.

[9] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman. *Experimental demonstration of reservoir computing on a silicon photonics chip.* Nature communications, 5, 2014.

[10] Chuck Moore. *Data processing in exascale-class computer systems.* 2011.

[11] H. Jaeger. *The 'echo state' approach to analyzing and training recurrent neural networks.* Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148:34, 2001.

[12] W. Maass, T. Natschläger, and H. Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations.* Neural computation, 14(11):2531–2560, 2002.

[13] David Verstraeten, Benjamin Schrauwen, Michiel d'Haene, and Dirk Stroobandt. *An experimental unification of reservoir computing methods.* Neural networks, 20(3):391–403, 2007.

[14] Floris Laporte, Joni Dambre, and Peter Bienstman. *Highly parallel simulation and optimization of photonic circuits in time and frequency domain based on the deep-learning framework PyTorch.* Scientific reports, 9(1):5918, 2019.

1 Machine Learning & Neuromorphic Computing

Computers have been used to find relations in data for as long as they were around. The models used to approximate and find those relations have grown in complexity with increasing computing power. During this chapter we'll try to introduce a small subset of these models along with some key concepts used in *Machine Learning* with a strong focus on the most relevant ones used in later chapters: *linear models*, *gradient descent*, *backpropagation*, *dimensionality reduction* and *neural networks*, eventually leading to an introduction to *Photonic Neuromorphic Computing*.

1.1 Linear models

Of all machine learning concepts introduced here, *Linear Models* remain relevant as being the most interpretable and easy to implement, yet they remain incredibly powerful in combination with other paradigms or even on their own.

1.1.1 Linear regression

Linear regression is traditionally defined as the method that finds the best fitting straight line through a set of points by minimizing the *y-distance* between the points and the line. In the example depicted in Fig. 1.1, this amounts to finding

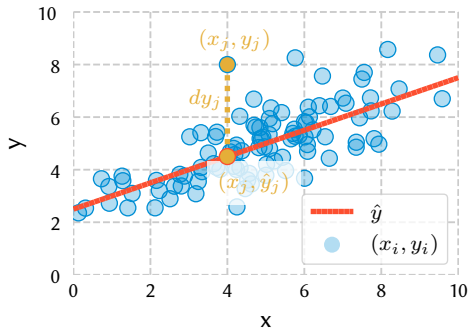


Figure 1.1: A point cloud that seems to have a linear relationship

a line defined by

$$\hat{y} = ax + b \tag{1.1}$$

Where $\hat{y}$ is the *predicted* y-value for the given x-values. This equation can be generalized in matrix form as follows:

$$\hat{y} = \begin{pmatrix} b & a \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{1.2}$$

This matrix form for the straight line is preferred over the explicit form as it generalizes better to higher dimensional problems, where x belongs to a N-dimensional vector space (called the *feature* space), instead of being just a scalar value:

$$\hat{y} = w^T x = \begin{pmatrix} w_0 & w_1 & \cdots & w_N \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{pmatrix} \tag{1.3}$$

In this case a vector of $N + 1$ elements (a bias term 1 and N *features*) gets weighted by $N + 1$ weights (often w_0 is called the *bias*). Let us define X as the composite matrix consisting of B observations of the vectors x_i, each of $(N+1)$ elements (the features) forming a $(N + 1) \times B$ matrix[1].

[1] In this thesis, we'll consistently denote the number of samples by the letter B, which stems from the *batch size*: a term that is traditionally only used during iterative optimization methods like *gradient descent* (see 1.2), where a *batch* means a subset of the total number of samples (observations) available. Note that in the case of regression tasks, however, *all* data is used.

To each of vectors x_i in the batch X belongs a scalar data-point y_i, such that the equation for the predicted points $\hat{y}_i$ can be written as

$$\begin{aligned}\begin{pmatrix}\hat{y}_1 & \cdots & \hat{y}_B\end{pmatrix} = \hat{y} &= w^T X \\ &= \begin{pmatrix}w_0 & w_1 & \cdots & w_N\end{pmatrix} \cdot \begin{pmatrix}1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1B} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NB}\end{pmatrix}\end{aligned} \tag{1.4}$$

Obviously, the predicted value $\hat{y}$ has to be as close as possible to the real value y. In the case of *linear regression*, this is achieved by minimizing the *mean squared error* (MSE) *loss*[2] L:

$$L = \frac{1}{2m} |y - \hat{y}|^2 \tag{1.5}$$

This term is often also called the L_2-loss as it purpose is to minimize the average L_2-*norm*[3] between prediction and target. Going on, this loss term can be rewritten as follows:

$$\begin{aligned} L &= \frac{1}{2m} |y - \hat{y}|^2 && (1.6)\\ &= \frac{1}{2m} \left|y - w^T X\right|^2 && (1.7)\\ &= \frac{1}{2m} \left(y - w^T X\right) \left(y - w^T X\right)^T && (1.8)\end{aligned}$$

To minimize this expression, the derivative (gradient with respect to w) has to be zero:

$$\begin{aligned}&\nabla_w \left(yy^T - yX^T w - w^T X y^T + w^T X X^T w\right) \\ &\quad = -2Xy^T + 2XX^T w \\ &\quad = 0\end{aligned} \tag{1.9}$$

Which yields for the weights:

$$w = (XX^T)^{-1} X y^T = X^+ y^T \tag{1.10}$$

Here, the expression for the *pseudo-inverse* of X, denoted by X^+ can be recognized in the last expression[4]. Note that this closed-form solution for finding the

[2] *Loss*, in machine learning, is generally a cumulative error between prediction and target. The loss function can be chosen freely and its choice only depends on the application domain and the goal one wants to achieve.

[3] $\sqrt{\sum_i x_i}$

[4] In fact, this is basically how the pseudo-inverse was defined in the first place.

weights given the sample points X and the target y relies on the fact that the expression XX^T is invertible. This is usually not a problem, as for the $(N+1) \times B$ matrix X, the number of samples B is typically much larger than the number of features N, resulting in a dense $(N+1) \times (N+1)$ matrix XX^T that is almost always invertible. In other words: given enough sample points ($B \gg N$), you would be very unlucky to have them all be linearly dependent.

1.1.2 Regularization and overfitting

Often, for high-variance data, the closed-form solution defined above yields weights that vary significantly in size. This is not typically what you want, as this leads to *overfitting*. Overfitting happens when the model is *too specialized* on the data it was presented with during training and as a result does not generalize well to unseen data. The term originally comes from fitting a polynomial to a set of data points, as can be seen in Fig. 1.2 Overfitting generally is a result of

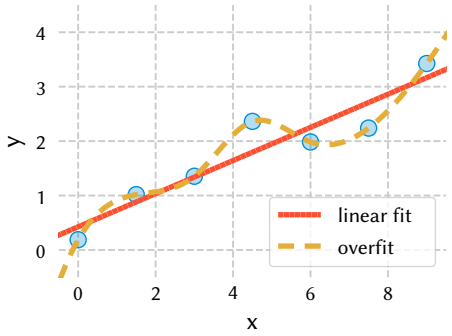


Figure 1.2: For the data points presented here, a simple linear fit is probably sufficient. The fitted line going through all the data points is clearly overfitting.

using too many parameters in the model at hand. The traditional way of dealing with this is often to reduce the number of parameters. However, a more general approach is to use *regularization* of the parameters of the model, which often yields better results.

Regularization penalizes models using too large weights. This penalty is enforced by adding an extra term to the loss function. This term is usually proportional to the norms of the weights. Generally speaking, the MSE loss would be extended as follows:

$$L = \frac{1}{2m} |y - \hat{y}|^2 + \lambda |w|^p \tag{1.11}$$

The two most common regularization methods are L_1-regularization or LASSO (Least Absolute Shrinkage and Selection Operator) regularization, for which $p = 1$, and L_2 regularization or Tikhonov regularization, for which $p = 2$. The application of Tikhonov regularization to linear regression is often called *ridge regression* and results in a modification of the closed-form solution (1.10) for the weights:

$$w = (XX^T - \lambda I)^{-1} X y^T \tag{1.12}$$

The expression for the weights under LASSO regularization is not as simple, but can be shown to equal [1]

$$w = \text{sign}\left(X^+ y^T\right)\left(\left|X^+ y^T\right| - \lambda I\right)^+ \tag{1.13}$$

1.2 Loss minimization by gradient descent

In the regression examples above, a closed-form solution for each of the loss functions was found. However, this is often more the exception than the norm. For more complicated non-linear models or for more specialized loss functions, finding an analytic solution is almost never possible.

A naive solution is to just calculate the loss for every parameter combination possible and see where the minimum lies. This parameter sweep approach works well for models with only a few parameters, but unfortunately becomes completely unfeasible for models existing in a high dimensional feature space. For these kind of problems, a smarter *iterative* approach is necessary. One of the most used algorithms to iteratively find the minimum of the loss function is *Gradient Descent* (GD).

Imagine a ball placed on a hill. When the ball is placed on one of the slopes of the hill, it will roll down. This same principle is applied during GD. When using GD, a random weight vector is chosen (think of it as the coordinates of the ball); this weight vector will have a certain loss associated to it (the altitude of the ball). Just like the ball will roll down the hill in the direction where the hill is the steepest, the weight vector will be updated by GD in the direction where the gradient of the loss points. By taking those steps consistently in the direction of the gradient, you're *guaranteed* to eventually arrive at *a* minimum of the loss function, as illustrated in Fig. 1.3. Note that this minimum might be a *local* minimum.

What is nice about the MSE loss, which defines the closed-form solution for the regression techniques, is that this loss function results in a *convex* surface for linear models. This is a big advantage over (most) other loss functions, as a convex surface only has a single minimum: the *global* minimum and thus, one is guaranteed to find its minimum by gradient descent.

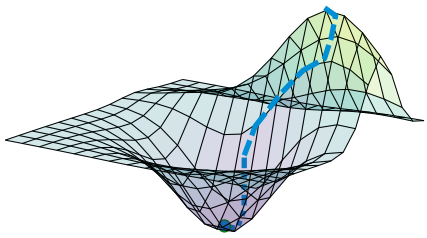


Figure 1.3: During gradient descent, the minimum of the loss function is found by iteratively updating the weights in the direction that the loss decreases. This direction is determined by the gradient of the loss function, hence the name of this algorithm.

The math behind gradient descent is quite easy to understand. The change in loss $\mathrm{d}L$ resulting from a *small change* $\mathrm{d}w$ in the weights can be expressed as follows:

$$\mathrm{d}L = \nabla_w L \cdot \mathrm{d}w \tag{1.14}$$

When the change of the weights itself is chosen in the opposite direction of the gradient of the loss such that

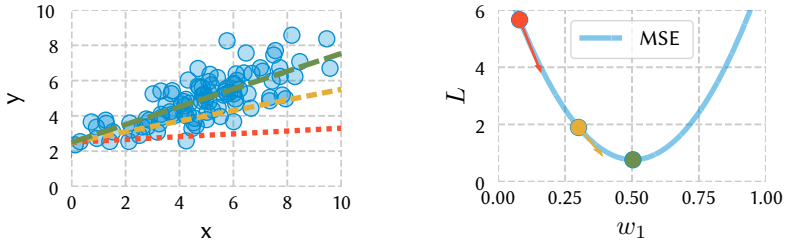
$$\mathrm{d}w = -\eta \nabla_w L, \tag{1.15}$$

the expression becomes

$$\mathrm{d}L = -\eta \left|\nabla_w L\right|^2. \tag{1.16}$$

This means that the weights chosen in such a way will always decrease the value of the loss function. The parameter η is a strictly positive hyper-parameter called the *learning rate* and determines how fast the minimum of the loss function is reached. Choosing a good value for η is more an art than a science: choose it too small and you might never reach the minimum. Choose it too large and equation (1.14) does not hold anymore, which means that the ball might overshoot the minimum of the valley, or worse, overshoot the valley altogether. That said, there are a whole lot of smart optimization techniques such as Adam [2], RMSProp [3], Nesterov Momentum [4] and many others [5] that improve the gradient descent algorithm to reduce (but not remove) the importance of the initial learning rate choice.

As an example, GD was applied on the weights defining the best fitting line through the point cloud from Fig. 1.1. As can be seen in Fig. 1.4a, the line determined by the weights gets better after iteratively changing the weights in the direction of the minimum like presented in Fig. 1.4b. Moreover, the final best fitting line obtained by GD (green) is equivalent to the line obtained by the closed form solution of LR. This is to be expected, as both are found by minimizing the same loss function: the MSE.



(a) **(b)**

Figure 1.4: The line fitting the data points in (a) is iteratively updated by gradient descent, which changes the weights of the line in the direction of the minimum of the MSE loss (b). For illustration purposes, only the slope of the line, w_1, is updated.

1.3 Linear classifiers

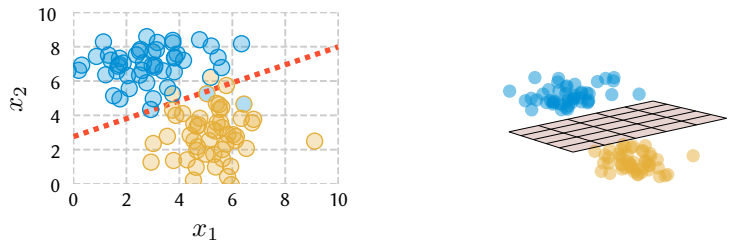
Where regression techniques try to make predictions about continuous target values y given a set of features X, classifiers try to predict discrete target values or *classes*. Although these two problems are clearly distinct, similar techniques as for regression can be applied to find the boundaries between those classes.

Let us consider a binary classification task as an example. Such a classification task can be summarized by modifying (1.3) in such a way that it is *thresholded* around 0.[5]

$$\hat{y} = \text{sign}\left(w^T X\right) \tag{1.17}$$

For such a binary classification task, one can visualize the operation of a linear classifier as dividing the N-dimensional feature space with a *hyperplane*, as can be seen in Fig. 1.5, where the hyperplane is a 1D-line in a 2D feature space (with features x_1, x_2) or a 2D plane in a 3D feature space (with features x_1, x_2, x_3).

[5] Note that a bias term is included in X, which makes a threshold around anything else than 0 redundant.



(a) **(b)**

Figure 1.5: A linear classification boundary tries to find the optimal boundary between the blue class (-1) and the orange class ($+1$), given two features x_1 and x_2. A class is misclassified when its boundary color (the prediction) is different from the inside color (the target). (a) In 2D, this dataset is not linearly separable, however in 3D (b), it is.

Classification mistakes might happen when the parameters (weights) of the hyperplane are not yet optimized to separate the clusters of data points *or* when the dataset is not completely linearly separable, like in Fig. 1.5a, where a decent boundary between the two classes was found, but some mistakes were still made.

It is important to note however, that the higher the dimension of the feature space, the *easier* it will be to find a hyperplane that separates the data points, *even* when the number of data points is still much larger than the number of features [6]. This is exemplified in Fig. 1.5, where in the 2D case - Fig. 1.5a - the dataset is not linearly separable, while in the 3D case - Fig. 1.5b - it is.

1.3.1 Linear regression

The most naive implementation of a linear classifier is probably to just use *linear regression*. Indeed, even though the goal is to predict *discrete* class labels, one can just pretend they are *continuous*. Using this approach, the weight matrix can easily be calculated using (1.10) to predict a *continuous approximation* of the class label. This continuous class value is then thresholded like in (1.17) to obtain the *class label*.

For binary classification tasks, this approach works reasonably well and it is the preferred way to implement the *readout* for *photonic reservoir computing* for *telecom applications*[6], as we will see later in 1.7.2. There are however a whole lot of disadvantages when using linear regression for classification, the main one being that it is very sensitive to unbalanced data and outliers. Indeed, linear regression tries to find a global *relation* between the data points[7], and if many of

[6] Recognizing bits in a bit stream is a binary classification task.
[7] This by itself should already be a reason not to use linear regression for complicated classification

them have class label -1 and only some of them have class label 1, the *bias* term (which is related to the average of all the data points) will be too low. The other weights will try to compensate for this, resulting in an inaccurate representation of the classes.

On the other hand, outliers will influence the regression as well, as they will be counted more heavily due to the quadratic MSE loss.

These problems can partly be solved through *regularization*, but for best performance on *classification* tasks, different loss functions like the *cross-entropy* loss (see 1.3.4) or different algorithms are necessary.

1.3.2 Perceptron model

Let us have another look at (1.17).

$$\hat{y} = \text{sign}\left(w^T X\right) \tag{1.18}$$

This model is often called the *perceptron model* after a 1958 paper by Rosenblatt [7]. In this paper, an iterative technique was proposed to update the weights. The *perceptron algorithm* takes a randomly misclassified data point x_i and uses *that* to update the weights:

$$w' = w + y_i x_i \tag{1.19}$$

Since x_i is misclassified, we have that $y_i = -\hat{y}_i = -\text{sign}(w^T x_i)$: it thus pushes the classification boundary towards a correct classification of x_i.

There is however a problem with this very simple model. Although it works great for linearly separable data, it becomes *unstable* for data like in Fig. 1.5 which is not.

1.3.3 Soft thresholding

It would be great if *gradient descent* could be used on the perceptron model (1.18). However, gradients are not well defined on discontinuous functions like the sign function. Let us instead replace the sign by a continuous *look-a-like*: the hyperbolic tangent function tanh:

$$\hat{y} = \tanh\left(w^T X\right) \tag{1.20}$$

The tanh, which is defined as

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}, \tag{1.21}$$

tasks. Data points related to class A might have nothing in common with data points related to class B. Finding a *global* linear relationship between both does not make much sense.

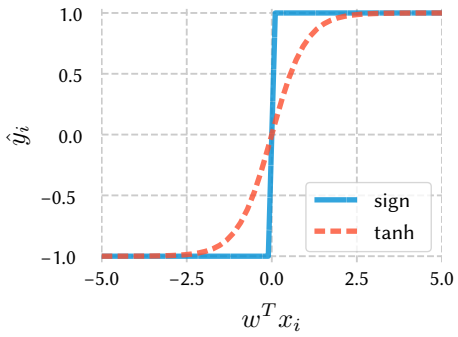


Figure 1.6: Instead of a discontinuous sign function, a continuous tanh can be used to threshold the prediction.

is bounded between -1 and 1 and thus implements a so-called *soft threshold*, as can be seen in Fig. 1.6. Moreover this function is *differentiable*! This allows us to apply *gradient descent* on its parameters and as such find optimal weights in an *iterative* manner. However, as we know by now, we need a *loss* function L to differentiate for gradient descent. We also know that the choice for the loss function is rather arbitrary, but *ideally* we would like one that is convex such that the gradient descent algorithm can easily find the single, global minimum.

1.3.4 Logistic Regression

The nice thing about using such a soft thresholding function is that it can be related to a *probability* $P(y_i = 1|x_i)$: given the feature vector x_i, what is the probability that you will predict y_i to have class label $+1$ (as opposed to class label -1). *Logistic regression* tries to do just that: it tries to have a notion of *probability* in the loss function. One can use the fact that the tanh is bounded between -1 and $+1$ to propose a *proxy* for the probability to guess y_i correctly given x_i:

$$P(y_i|x_i) = \frac{1}{2}\left(\tanh\left(\frac{1}{2}y_i w^T x_i\right) + 1\right) \tag{1.22}$$

$$= \frac{1}{1 + \exp\left(-y_i w^T x_i\right)} \tag{1.23}$$

The second function is often called the *sigmoid* or *logistic* function, hence the name *logistic regression*. Logistic regression now proposes to maximize the *product* of *all* these probabilities for each x_i, i.e. the *joint* probability of guessing all

y_i correctly given x_i[8]:

$$w = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{B} \frac{1}{1 + \exp\left(-y_i w^T x_i\right)} \tag{1.24}$$

One can take the natural log of this expression, normalize it over the number of sample points (batch size) B and multiply it by -1 (loss functions are *minimized*, not *maximized*) to arrive at the loss function used in logistic regression:

$$\begin{aligned} L &= -\frac{1}{B} \sum_{i=1}^{B} \log\left(\frac{1}{1 + \exp\left(-y_i w^T x_i\right)}\right) \\ &= \frac{1}{B} \sum_{i=1}^{B} \log\left(1 + \exp(-y_i w^T x_i)\right) \end{aligned} \tag{1.25}$$

This loss is called the *cross-entropy*[9] loss and is probably the most used loss function in machine learning for classification problems. This loss becomes small when y_i and the un-thresholded prediction $w^T x_i$ have the same sign, i.e. when (1.20) would yield the *correct* prediction, just like we would expect.

Note that often, equation (1.20) is not applied in practice. One can opt to stick to the hard threshold of (1.17) as all the continuous thresholding information is implicitly assumed in the cross-entropy loss (1.25). Consequently, logistic regression *only* differs from linear regression by the chosen loss function.

1.3.5 Regression in the complex domain

Optical signals are generally represented by *complex numbers*. Both the MSE loss (1.8) and the cross-entropy loss (1.25) are quite easily extended to the complex domain by replacing the transpose by a *hermitian transpose*:

$$L_{\text{MSE}} = \frac{1}{2m} \left| y - w^H X \right|^2 \tag{1.26}$$

$$L_{\text{XE}} = -\frac{1}{B} \sum_{i=1}^{B} \log\left(1 + \exp(-y_i w^H x_i)\right) \tag{1.27}$$

Where both w and X are complex-valued. However, having an expression for these losses in the complex domain does not automatically mean that they are well-behaved (i.e. convex). Luckily for the case of those two losses this is the case, as can be seen in Fig. 1.7.

Essentially the same derivation for the closed form solution for linear regression can be applied in the *complex domain* as well, yielding the following form

[8] This is also known as the maximum likelihood.

[9] Which I often write as XE, although that's probably not an accepted acronym.

(a) Complex MSE **(b)** Complex cross-entropy

Figure 1.7: Both the MSE and the cross-entropy loss are convex and well behaved functions in the complex domain.

for the weights at the *global minimum*:

$$w = (XX^H)^{-1}Xy^H \tag{1.28}$$

Note that this equation requires full knowledge of both the amplitudes *and* the phases of each $x_i \in X$. However, for many *real-life* applications, this closed form solution is *too restrictive*.

To see why, we will consider the binary classification task with class labels 0 and 1, which means that the vector y is real-valued. Assuming we have access to the amplitudes and phases for each $x_i \in X$[10] such that we can calculate the complex weight matrix w with (1.28), we can use this weight matrix w to make a prediction $\hat{y}_i$ for a new data point $\tilde{x}_i$:

$$\hat{y}_i = w^H \tilde{x}_i \tag{1.29}$$

However, when we implement a system like this in photonics, the optical measurement will normally not measure the complex-valued $\hat{y}_i$ directly, but rather what the photodiode detector will output is proportional to the following expression:

$$|\hat{y}_i|^2 = |w^H \tilde{x}|^2 \tag{1.30}$$

By using (1.28) to calculate the weights, we implicitly assume that the output should be a real-valued quantity in (1.29). However by squaring this prediction all the phase information gets lost, i.e. we don't actually *care* if $\hat{y}_i$ is almost real: we care if the *norm* corresponds to the *norm* of the class label. This seemingly insignificant detail puts too many constraints on the weight matrix w, possibly eliminating better weight matrices which are not constrained by having to return

[10] This is in practice not always self-evident, depending on the application.

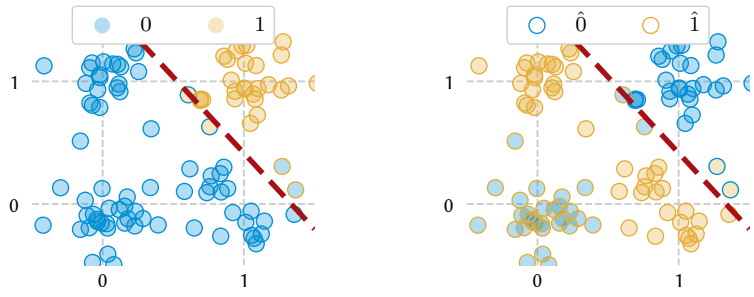
a real-valued prediction. Hence, the *actual* loss function we are interested in this case looks as follows[11]:

$$L_{\text{MSE}} = \frac{1}{2m} \left| y - \left| w^H X \right| \right|^2 . \tag{1.31}$$

By enclosing the expression $w^H X$ into the expression for the complex norm $|\cdot|$, the complex valued weights w are free to take on any phase they need[12]. However, this loss function cannot result in a nice closed form solution as the information loss due to the norm operation makes it non-invertible without assuming a phase. An iterative method like gradient descent is necessary here.

1.3.6 Classification on noisy Boolean problems

As a simple illustration on how these linear classifiers can be used, we apply them on two examples: a *noisy* AND function and a *noisy* XOR function[13]. When using normal *real*-valued regression, one can find a boundary for the AND that linearly separates the classes, as illustrated in Fig. 1.8a. When trying to to the same for the XOR in Fig. 1.8b, we find practically the *same* boundary! This means that at least *one* of the XOR clusters (in this case the cluster at $(0, 0)$) will be *misclassified*.



(a) Noisy AND **(b)** Noisy XOR

Figure 1.8: (a) The noisy AND can be linearly separated. (b) The XOR has two clusters that form a cross and can thus not be linearly separated as can clearly be observed (points with different boundary color (prediction) than the inside color (ground truth) are misclassified)

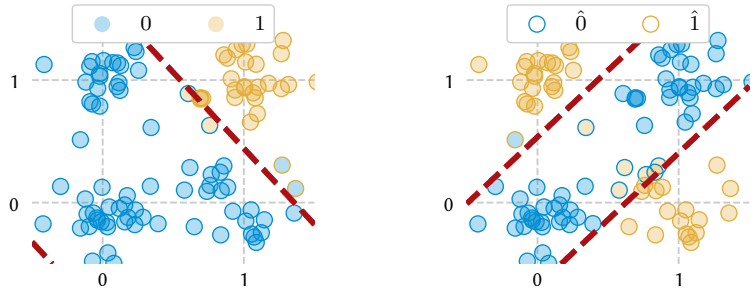
However, when we do the same with *complex* valued linear regression as laid out in the previous section, we *can* find a boundary for the XOR. This is

[11] Note that this loss function obviously only works for $y > 0$: relabeling might be necessary.

[12] Note that this is different from selecting just real-valued weights, as a phase relation *between* the elements of the weight matrix *exists* which *has* an influence on the magnitude $\left| w^H X \right|$

[13] Both of these functions are very relevant for later chapters, as they serve as perfect benchmark tasks for telecommunication applications.

an important observation: the quadratic *measurement* of complex-valued fields (taking the norm) is a *non-linear* operation. As the order of this operation is *quadratic*, we are effectively able to find *two* linear boundaries at once: enough to separate the XOR task!



(a) Noisy AND **(b)** Noisy XOR

Figure 1.9: When using complex weights, a boundary for the XOR *can* be found.

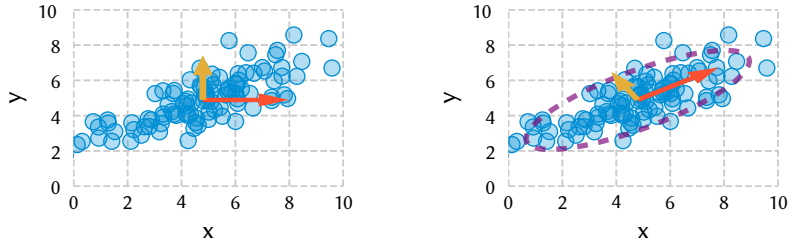
1.4 Dimensionality Reduction

When working with high-dimensional data, it is sometimes important to perform a dimensionality reduction technique to reduce chances of overfitting or to figure out which dimensions (features) of the data are the most important. There are two major *linear* dimensionality reduction techniques: *principal component analysis* (PCA) and *linear discriminant analysis* (LDA). Both techniques try to find a transformation for the features defining the data into as much as possible linearly uncorrelated variables. The approach of both techniques is however different in the sense that PCA is an *unsupervised* technique, i.e. it does not know or use the class labels, while LDA does use this information. Both techniques can be very valuable, as they effectively rank the possible linearly independent linear combinations of the features in order of importance.

1.4.1 Principal Component Analysis

As discussed above, PCA ranks the features in order of importance. But what is *importance*? The PCA algorithm *defines* the most important features (*eigenvectors* in fact) of the data to be the ones with the *highest* variance in the data associated to it. This choice becomes more clear when looking at figure Fig. 1.10a, where the same cloud of points as before is plotted. Looking at this data, we would probably say that the x-axis has more predictive power over the points than the y-axis, as when the data is projected on either of the axes, the cloud

is projected over a much larger range of values on the x-axis than on the y-axis. However, you do not need to stop there. In fact, PCA, looks at *linearly indepen-*



(a) Rank by variance **(b)** PCA

Figure 1.10: (a) We see that the x-axis is more important than the y-axis, as the data shows more variance along the x-axis. (b) However, the PCA algorithm goes a step further and describes the data in a new basis defined by the orthogonal PCA transformation. The order of the basis vectors is described by how well they describe the data.

dent axes of the data, as illustrated in Fig. 1.10b. The first of these axes is the axes that *maximizes* the variance, while the second axis is an axis perpendicular to the first axis maximizing the variance as well given this constraint. For highly dimensional data, this eventually leads to a collection of linearly independent axes for the data with decreasing variance, i.e. *importance*. Each of these axes describes a different linear combination of the initial features of the data.

Given this definition, PCA is often thought of as fitting an ellipsoid to a point cloud. Using this definition, we say that each axis of the ellipsoid corresponds to a *principal component* of the point cloud.

Mathematically, the weight vector defining the first principal component (the component with highest variance) can be described as the weight vector with unit norm that maximizes the covariance matrix [8] of X:

$$w_1 = \underset{w}{\operatorname{argmax}} \left(w^T (X - \bar{X})(X - \bar{X})^T w \right) \qquad \text{with } w^T w = 1 \tag{1.32}$$

$$= \underset{w}{\operatorname{argmax}} \left(w^T \operatorname{cov}(X) w \right) \qquad \text{with } w^T w = 1 \tag{1.33}$$

In what follows, we will assume $\bar{X}$ (the mean of X over the B sample points) to be zero, which can be achieved by a simple translation. The subsequent principal components can then be obtained the same way by first subtracting the first principle component weight vectors from X to ensure orthogonality:

$$X_{w_i} = X - \sum_{j}^{i-1} w_j w_j^T X \tag{1.34}$$

PCA now transforms X as follows:

$$\tilde{X} = \begin{pmatrix} w_0^T & w_1^T & \cdots & w_N^T \end{pmatrix} X \tag{1.35}$$

$$= W^T X \tag{1.36}$$

The transformed matrix $\tilde{X}$ can be related to the singular value decomposition (SVD) [9] of X:

$$X = USV^T \tag{1.37}$$

Where U and V are matrices of which the columns are orthogonal unit vectors, respectively called the left singular vectors and the right singular vectors. S is a diagonal matrix consisting of the singular *values*, which are strictly positive and — just like in PCA — ranked from large to small. Using this knowledge, it is not difficult to see that

$$XX^T = US^2U^T \tag{1.38}$$

It is clear that (1.32) is only satisfied if w_1 corresponds to the first column of U, w_2 corresponds to the second column and so on. Therefore, we have $W = U$ and we get:

$$\tilde{X} = W^T X \tag{1.39}$$

$$= SV^T \tag{1.40}$$

Since the SVD algorithm is a very efficient one, this representation for PCA is preferred.

1.4.2 Linear discriminant analysis

When the class labels of the data points are known, PCA can be extended by replacing the covariance of all the data points X, by the mean of the covariances for each of the data points belonging to a class c:

$$\text{cov}(X) = (X - \bar{X})(X - \bar{X})^T \rightarrow \frac{1}{C}\sum_{c=1}^{C}(X_{y=c} - \bar{X}_{y=c})(X_{y=c} - \bar{X}_{y=c})^T \tag{1.41}$$

This choice of covariance matrix will still optimize for basis vectors that maximize the variance *inside* classes, but the variance *between* classes is not taken into account. This has the emergent result that data points resulting to different classes will (hopefully) best be represented by different axes, subsequently improving classification between the classes.

1.5 Artificial neural networks

Artificial neural networks (ANNs) are a beautiful brain-inspired programming paradigm that allows a computer to learn from observational data. ANNs and in particular its branch of *deep learning* have currently entrenched themselves as *the* go-to method for problems like image recognition, natural language processing and speech recognition.

However, this has not always been the case. Programming has traditionally been a process of telling the computer exactly what it needs to do by breaking the problem at hand up into many smaller well-defined tasks, each of which a computer knows well how to solve.

The contrast with a neural network could not be bigger: when *programming* a neural network, the *programmer* only decides on the *architecture* of the network and then proceeds to give the network a large amount of examples and data, hoping the network will figure out the relations between the data by itself to eventually arrive at a solution to the problem. If the network does not succeed to find a solution to the problem at hand, the programmer proceeds to tweak some hyper-parameters or designs a new network architecture. This leads to a very iterative design approach, which is inherently completely different from traditional programming.

So what is a neural network? As mentioned before, a neural network is a brain-inspired structure consisting of *artificial neurons* — often called the *nodes* of the network — which are interconnected by artificial synapses called the *weights* of the network.

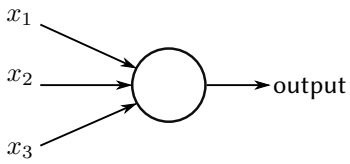


Figure 1.11: An artificial neural network node (neuron) with three inputs. All artificial neurons always have a single output.

This picture of *nodes* and *weights* becomes more clear when looking at a single artificial neuron as depicted in Fig. 1.11. In an ideal case, such a neuron makes a weighted sum of its input signals (x_1, x_2 and x_3 in this case); if the sum is bigger than a certain threshold value t, the output of the neuron is one, else its output is zero. This sounds familiar! It sounds a lot like the *perceptron model* (1.18): defined for a *single* output:

$$\hat{y} = \text{sign}\left(w^T x\right) \tag{1.42}$$

Indeed, the *perceptron* model is one of the most basic ways to implement an *artificial neuron*. This makes sense, as this model can be intuitively understood as if it makes a decision (-1 or 1) depending on the *features* it finds important.

As we did for *logistic regression*, it is often useful to swap out the discontinuous sign-function by a smoothened *activation* function:

$$\hat{y} = \text{act}\left(w^T x\right) \tag{1.43}$$

This activation can be *anything* you want. The most popular neuron activations are the tanh (just like in logistic regression) and the Rectified Linear Unit (ReLU), which is defined as follows:

$$\text{ReLU}(x) = \begin{cases} x & \text{when } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.44}$$

As the mathematical model underlying these artificial neuron is very rudimentary, a single neuron will not be able to make any subtle decisions. However, the idea is that a whole network like in Fig. 1.12, consisting of layers of individual neurons feeding into layers of subsequent neurons will be able to do so.

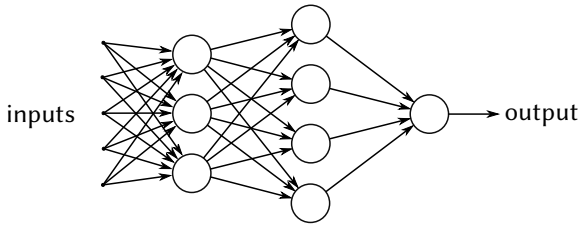


Figure 1.12: An artificial neural network consists of *nodes* representing the neurons and *weights* representing the synapses.

In such a neural network, each individual neuron in the first layer will thus make decisions for the input signals it gets and transmit a signal if it finds the input signals important enough to do so. Each subsequent layer will then too weigh the outputs of previous layers and decide for itself which previous neuron outputs it finds important. This kind of network or artificial neurons, where the output of a previous layer of neurons is fed into the next layer of neurons is called a *Feed-Forward* Neural Network (FFNN). In fact, this network belongs to a certain subclass of FFNNs, called the *Fully-Connected* Neural Networks (FCNN). There, every neuron in one particular layer of the network is connected to every neuron in the next layer of the network. Note that although it looks in Fig. 1.12 like each of these neurons has multiple outputs, this is not the case: the same output of the neuron is transmitted to each of the neurons in the next layer.

Mathematically, the operation of a *layer* of N neurons, i.e. the parallel execution of these neurons, on the output of a previous layer x^{i-1} of M neurons

can be expressed as follows:

$$x^i = \text{act}({W^i}^T x^{i-1}). \tag{1.45}$$

or when acting on a full batch X containing B feature vectors:

$$X^i = \text{act}({W^i}^T X^{i-1}). \tag{1.46}$$

We see that the only difference with (1.43) is the fact that the weight *vector* is replaced by a $(M+1) \times (N+1)$ weight *matrix* working in on the $(M+1)$-dimensional feature *vectors* x^{i-1} in the batch X^{i-1}. Written in terms of the individual matrix components, this equation becomes:

$$x^i_{nb} = \text{act}\left(\sum_{m=0} w^i_{mn} x^{i-1}_{mb}\right) \tag{1.47}$$

$$= \text{act}\left(z^i_{nb}\right) \tag{1.48}$$

Where we defined z^i_{nb} to be the weighted output of neuron n in layer i on the feature vector with index b:

$$z^i_{nb} = \sum_{m=0} w^i_{mn} x^{i-1}_{mb} \tag{1.49}$$

This value is also called the *logit* of the neuron. It is a useful quantity when discussing *backpropagation*.

1.6 Backpropagation

We have seen that for all but the simplest models we have to rely on *iterative* techniques like *gradient descent* to find the minimum of the loss function. We expect this to be the same for these complicated structures of interconnected neurons. Indeed, *gradient descent* is the basis of *neural network* optimization. However, applying gradient descent is not straightforward: how will you find the gradient with respect to each parameter (weight) in the network? This is where the *backpropagation* algorithm comes in.

Backpropagation was first proposed in 1986 [10] to optimize neural networks. Although it was being applied successfully at that time, due to a lack of computing power the neural networks themselves were too small to do anything interesting. It was only much later, in the early 2000s, after the computing power reached a certain threshold that backpropagation became truly successful as the backbone for optimizing very large neural networks in the field of *deep learning*.

The ultimate goal of *backpropagation* is to find an expression $\partial L / \partial w^i_{nm}$ This would allow us to update each of the weights like proposed in (1.15). Assume a small variation $\mathrm{d}z^i_n$ in the logit z^i_n. This variation will propagate to the end of the neural network such that the loss changes by a value

$$\mathrm{d}L = \frac{\partial L}{\partial z^i_n}\mathrm{d}z^i_n \tag{1.50}$$

$$= \delta^i_n \mathrm{d}z^i_n \tag{1.51}$$

Here we defined the value δ^i_n and call it the *error* each neuron makes on the loss L:

$$\delta^i_n = \frac{\partial L}{\partial z^i_n}. \tag{1.52}$$

The value for this error in the last layer of the network can easily be related to the output of the last layer L of the network x^L_i by applying the chain rule:

$$\delta^L_n = \frac{\partial L}{\partial x^L_n}\frac{\partial x^L_n}{z^L_n} \tag{1.53}$$

$$= \frac{\partial L}{\partial x^L_n}\frac{\partial}{z^L_n}\mathrm{act}\left(z^L_n\right) \tag{1.54}$$

It is now key to relate the error in the last layer to the layer before it. More generally, we'd like an expression that relates the error in layer $i+1$ to the error in layer i. This relation can again be found by applying the chain rule:

$$\delta^i_n = \sum_m \frac{\partial L}{\partial z^{i+1}_m}\frac{\partial z^{i+1}_m}{\partial z^i_n} \tag{1.55}$$

$$= \sum_m \delta^{i+1}_m \frac{\partial z^{i+1}_m}{\partial z^i_n} \tag{1.56}$$

$$= \sum_m \delta^{i+1}_m \sum_k w^{i+1}_{km}\frac{\partial x^{i+1}_k}{\partial z^i_n} \tag{1.57}$$

$$= \sum_m w^{i+1}_{nm}\delta^{i+1}_m \frac{\partial}{\partial z^i_n}\mathrm{act}\left(z^i_n\right) \tag{1.58}$$

So it turns out, the way to relate the error at layer i to the error at layer $i+1$ is to multiply the latter with its *non-transposed* weights (in contrast to using the transposed weights in the *forward* pass) and the derivative of the activation.

There is one final piece of the puzzle missing to completely define the back-propagation algorithm: the relation between the error at the neuron and how the weights need to be updated. By a similar application of the chain rule, one can show that:

$$\frac{\partial L}{\partial w^i_{mn}} = x^{i-1}_m \delta^i_n \tag{1.59}$$

The *backpropagation* algorithm can be summarized as follows:

1. calculate the error δ_n^L at each neuron of the output layer using (1.54).
2. calculate the error δ_n^i at each neuron of each layer using all the errors δ_m^{i+1} of all the neurons at the next layer using (1.58).
3. calculate the gradient of the loss with respect to the weight $\partial L/\partial w_{mn}^i$ by using the error in that layer and the activation at the *preceding layer* by using (1.59)
4. update the weights by an algorithm like gradient descent like in (1.15).

1.7 Recurrent neural networks

We introduced neural networks as a *brain-inspired* paradigm. However there is a missing piece... When reading this paragraph, your human brain is capable of relating words with preceding words, sentences with preceding sentences to make up one coherent story in your mind.

One of the main *disadvantages* of feed-forward neural networks discussed up until now is the fact that they cannot easily do this. They cannot find relations in *temporal* data. A solution to this is the so called *Recurrent Neural Network* (RNN), where feedback loops are introduced in the system: neurons in a certain layer can now be connected to neurons in a *previous* layer. This way, information injected into the RNN will stay in the network for a certain time and interact in the neurons of the RNN with new information. This makes RNNs the go-to choice for problems like *speech recognition*, time series prediction, image captioning, text translation, text generation, and so on [11].

Whereas the topology of FFNNs are relatively limited (each layer of neurons is supposed to be connected to the next layer of neurons), the architecture of *recurrent neural networks* can be as complex as you want, as for example in Fig. 1.13.

The *easiest* way to implement a recurrent neural network, however, is to just connect a FFNN onto itself, as is illustrated in Fig. 1.14. This allows data X^t that entered at time step t to mingle with data entering the FFNN on a later time step. In fact, this representation of an RNN is not much different than the representation of a *traditional* feed forward neural network! Consider for example the *rolled-out* version of this network, as illustrated in Fig. 1.15. This *chain* of *subnetworks* reveals how a recurrent neural network is related to sequential data.

However, choosing a recurrent neural network topology brings a lot of problems, especially for the *backpropagation* algorithm. As we have seen, during backpropagation, the weight matrix is applied multiple times to the error at the

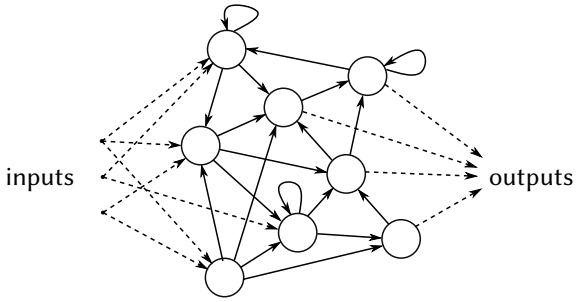


Figure 1.13: A recurrent neural network can have any topology

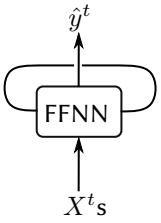


Figure 1.14: Connecting a FFNN onto itself is the easiest way to create a *recurrent neural network*.

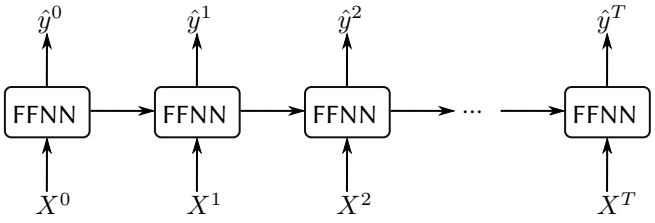


Figure 1.15: When an RNN is *rolled out*, it becomes clear that it in fact has a similar network structure as a *deep* neural network. However, the same subnetwork is used over and over in conjunction to letting new data X^t in every time step.

output of the system. Since a *recurrent* neural network can be interpreted as a neural network with an infinite amount of layers, this leads to either vanishing or exploding gradient updates if the weight matrix is not carefully chosen [12]. This vanishing/exploding gradient problem prevents the network to learn relations on larger timescales [13, 14].

1.7.1 Long Short Term Memory

Long Short Term Memory (LSTM) cells [15], are intricate building blocks (like the FFNN in Fig. 1.15) for constructing RNNs. They are especially engineered to learn relations between input data on a *short* scale (e.g. generating the next word in a sentence) and on a *longer* scale (e.g. generating the next sentence in an essay). It is important to note that the LSTM cell does *not* solve the vanishing/exploding gradient problem. It is just a way of constructing a network that learns on *two* timescales by keeping *two* internal *cell states*. The exact inner workings of the LSTM are beyond the scope of this chapter, but it is important to note that they arguably are the most successful software-based recurrent neural network to date.

1.7.2 Reservoir computing

Reservoir computing (RC) is a technique that has been independently proposed in the early 2000s by Jaeger [16] and Maass [17]. They called it the Echo State Network (ESN) and the Liquid State Machine (LSM) respectively.

RC is considered a solution to the vanishing gradient problem that plagues recurrent neural networks. However, as we will see, it is more of a *workaround*. It is based on the principle that complex classification with a linear classifier becomes *easier in higher dimensions*, a principle we briefly touched upon in 1.3.

The way this works is as follows: consider a recurrent neural network with a completely random topology, like in Fig. 1.13. Instead of trying to train the weights of this network, we just *accept* the fact that it is not feasible to train such a network. However, we *do use* the network. Indeed, the network is very useful because it exhibits a *memory* (due to its recurrent nature). It is this combination of memory and non-linearity that will transform a low-dimensional *time dependent* input signal to a higher dimensional *spatially dependent* output signal. As we know by now, we can use a simple *linear classifier* — which in the context of RC is called the *readout* — to interpret such a high-dimensional output. The advantage of this approach is that it is very general: multiple readouts can be trained to extract different features from the reservoir.

We can formalize the reservoir computing approach [6] by saying that the reservoir needs to exhibit a so-called *fading memory*, which means that the reservoir asymptotically forgets past inputs. On top of that, the reservoir needs to

exhibit a *mixing property*: the internal dynamics of the reservoir should be rich enough to enable a linear classifier to easily separate the different classes.

This recurrent neural network approach was proposed during a time that other solutions like LSTMs were still very difficult to train and hence took off quite rapidly as a valid approach for tackling time-dependent problems with recurrent neural networks without having to deal with vanishing/exploding gradients. However, due to improving computing power, better optimization techniques and improved intricate RNN architectures like the LSTM, training RNNs has now become easy enough that reservoir computing is — in software — not being used all that often anymore. However, when computing power is limited or fast training times are important, reservoir computing remains relevant, as under these constraints it can outperform the more intricate [over-]engineered approaches like LSTMS [18, 19].

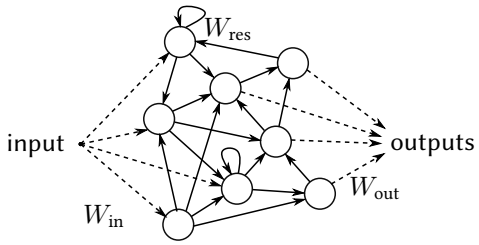


Figure 1.16: A low-dimensional time-dependent input signal X^t gets distributed by an [optional] input layer into the reservoir. Inside the reservoir, the signal mixes with previous versions of itself due to the highly dynamic architecture of the reservoir. At each time, [a part of] the reservoir state is read out by the readout to make a time dependent prediction $\hat{y}^t$

For a reservoir such as the one depicted in Fig. 1.16, operating in discrete time and for a certain sequence of inputs x^t, the resulting reservoir states u_t and the corresponding readout values y_t are given by the following formulas [20]:

$$u_t = W_{\text{in}} x_{t-1} + f_{\text{res}}(W_{\text{res}} u_{t-1}) \tag{1.60}$$

$$\hat{y}_t = f_{\text{act}}(W_{\text{out}} u_t) \tag{1.61}$$

Here, W_{in} and W_{res} perform a fixed random linear combination on the input states and the reservoir states respectively, while W_{out} performs the readout operation by doing an application-dependent linear combination on the reservoir states. Moreover, f_{res} and f_{act} are a non-linear activation functions. Often, the non-linearity of the system is directly related to the difficulty of the tasks it can solve. Luckily, it turns out that for the telecom applications we will be targeting in this thesis not that much non-linearity is necessary. In fact, for the photonic hardware implementations targeted here, it turns out the quadratic non-linearity

of the detector is good enough and the reservoir operation can be described as

$$u_t = W_{\text{in}} x_{t-1} + W_{\text{res}} u_{t-1} \tag{1.62}$$

$$\hat{y}_t = W_{\text{out}} f_{\text{det}}(u_t), \tag{1.63}$$

Where f_{det}, the detector operation, is the only non-linear element in the whole system. These kind of reservoir systems are called *passive* reservoirs as there are no non-linearities in the reservoir itself. Obviously their dynamics are less rich but in general they make up for it in much higher operation speeds as we will see in the following chapters.

The fact that a *reservoir* does not need to be trained makes it an *ideal* candidate for *hardware* implementations. Examples of such reservoirs include memristor networks, mechanical systems, networks of randomly connected boolean logic gates and of course linear and non-linear photonic systems.

1.8 Photonic neuromorphic computing

1.8.1 Photonic reservoir computing

Backed by the promise of the ultra-high-speed and high-bandwidth signal propagation of photonics, the reservoir computer has already found its way into several optical hardware implementations. These *photonic reservoir computers* roughly split into two kinds: the single-node reservoir, based on delayed feedback [20–25] and the photonic reservoir on chip [26–28].

The single-node reservoir is often highly non-linear and uses delayed feedback to get an interesting mixing of the input states. Because of the delayed feedback, a single node can act as multiple *virtual nodes* by time-multiplexing the input signal. Examples of such nodes include lasers, Mach-Zehnder modulators and electronic FPGAs. They have typically quite good performance on several benchmark tasks but are usually much slower than on-chip integrated photonic reservoirs, making them much less suited for telecom applications.

On-chip photonic reservoirs typically consist of an interconnection of multiple nodes on a photonic chip. These nodes can be anything ranging from Semiconductor Optical Amplifiers [26] (SOA) to microring resonators to photonic crystal cavities [28]. Any structure that possesses a *fading memory* and is *mixing* the signal sufficiently can be considered for (photonic) reservoir computing.

One of the advantages of photonic reservoir computing is the possibility of removing the non-linearity inside the reservoir in favor of a non-linear measurement operation at the readout [20, 27]. These kind of reservoir computers are called *passive* photonic reservoir computers, and exhibit ultra-high speed of operation because of the absence of internal non-linearities. In fact, the operation

speed is only limited by the operation speed of the final photodetector. Removing the non-linearities inside the reservoir is of course only possible because we are working with coherent light, which possesses an amplitude and a phase, which — as we have briefly touched upon in 1.3.5 and 1.3.6 — results in a non-linearly mixed magnitude on detection.

One of the main on-chip photonic reservoir architectures is the *Swirl Reservoir* [27], which gets its name from the architecture depicted in Fig. 1.17, for which the node interconnections *swirl* in a clockwise manner. The swirl reservoir is a completely passive reservoir that has been quite successful for some basic telecom benchmark tasks, such as the time-delayed XOR task and header recognition. In the next chapter, we will see how we can improve the performance of the swirl reservoir on these benchmark tasks by slightly breaking the philosophy of reservoir computing and allowing some phases *inside* the reservoir to be optimized by using backpropagation *through* the reservoir; something that will only be possible by optimization through backpropagation.

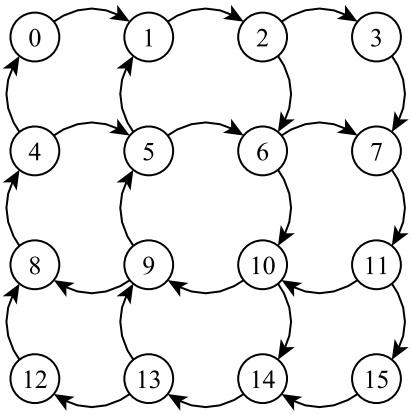


Figure 1.17: Swirl reservoir architecture with 16 nodes.

One of the problems with the swirl reservoir and in fact most physical reservoirs which are built on a 2D surface, is the limited interconnection topology: each node of the reservoir is only connected to its neighbors. One may ask if the conventional node structure [...] is in fact the best topology for such a photonic reservoir, as for example in photonic cavities, light is able to mix continuously, possibly introducing a much richer interconnection topology. This is one of the topics that will be explored in depth in chapter 3 and 4, where we will abandon the node structure of photonic reservoirs in favor of such photonic cavities.

1.8.2 Neuromorphic computing with unitary matrices

Another photonic neuromorphic computing paradigm that has recently emerged is neuromorphic computing with unitary matrices. It is in fact well known that any unitary matrix can be constructed from cascading MZIs in a staggered fashion [29, 30], as is illustrated in Fig. 1.18. However, it is only more recently that

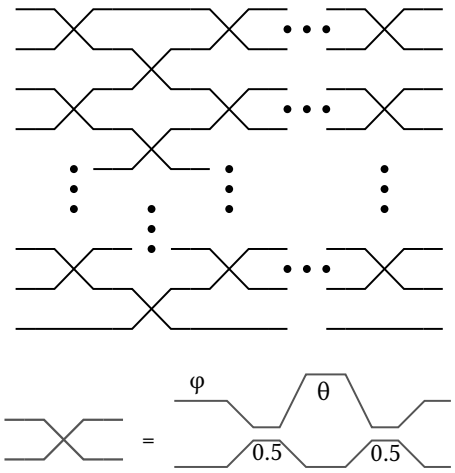


Figure 1.18: Any unitary matrix can be constructed from cascading multiple MZIs together.

this knowledge got transferred to the domain of *deep learning*, were it was shown that these mesh architectures exhibit excellent properties for constructing recurrent neural networks [31], as their unitary nature limits the exploding/vanishing gradient problem. Due to these findings, these structures — although already quite well known in photonics [32–34] — were recycled to be used as deep neural networks in photonic hardware [35, 36]. We will explore these kind of networks in a little more depth in the next chapter.

1.9 Conclusion

In this chapter we introduced most of the machine learning theory that will be used in subsequent chapters. We started by introducing simple linear models and worked our way up through non-linear models to neural networks to finally arrive at recurrent neural networks and reservoir computing. We also briefly touched upon two photonic neuromorphic computing paradigms — photonic reservoir computing and photonic meshes — that will serve as the starting point of the next chapters.

References

[1] Robert Tibshirani. *Regression shrinkage and selection via the lasso: a retrospective*. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 73(3):273–282, 2011.

[2] Diederik P Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

[3] Tijmen Tieleman and Geoffrey Hinton. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural networks for machine learning, 4(2):26–31, 2012.

[4] Yurii Nesterov. *A method for unconstrained convex minimization problem with the rate of convergence O (1/k^ 2)*. In Doklady AN USSR, volume 269, pages 543–547, 1983.

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. arXiv preprint arXiv:1609.04747, 2016.

[6] David Verstraeten, Benjamin Schrauwen, Michiel d'Haene, and Dirk Stroobandt. *An experimental unification of reservoir computing methods*. Neural networks, 20(3):391–403, 2007.

[7] Frank Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological review, 65(6):386, 1958.

[8] Svante Wold, Kim Esbensen, and Paul Geladi. *Principal component analysis*. Chemometrics and intelligent laboratory systems, 2(1-3):37–52, 1987.

[9] Gene H Golub and Christian Reinsch. *Singular value decomposition and least squares solutions*. In Linear Algebra, pages 134–151. Springer, 1971.

[10] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. *Learning representations by back-propagating errors*. Cognitive modeling, 5(3):1, 1988.

[11] Andrej Karpathy. *The unreasonable effectiveness of recurrent neural networks*. Andrej Karpathy blog, 21, 2015.

[12] Sepp Hochreiter. *The vanishing gradient problem during learning recurrent neural nets and problem solutions*. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02):107–116, 1998.

[13] Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. Diploma, Technische Universität München, 91(1), 1991.

[14] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. *Learning long-term dependencies with gradient descent is difficult*. IEEE transactions on neural networks, 5(2):157–166, 1994.

[15] Sepp Hochreiter and Jürgen Schmidhuber. *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997.

[16] H. Jaeger. *The 'echo state' approach to analyzing and training recurrent neural networks*. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148:34, 2001.

[17] W. Maass, T. Natschläger, and H. Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations*. Neural computation, 14(11):2531–2560, 2002.

[18] Herbert Jaeger. *Echo state network*. Scholarpedia, 2(9):2330, 2007.

[19] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. *An overview of reservoir computing: theory, applications and implementations*. In Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007, pages 471–482, 2007.

[20] Quentin Vinckier, François Duport, Anteo Smerieri, Kristof Vandoorne, Peter Bienstman, Marc Haelterman, and Serge Massar. *High-performance photonic reservoir computer based on a coherently driven passive cavity*. Optica, 2(5):438–446, 2015.

[21] Lennert Appeltant, Miguel Cornelles Soriano, Guy Van der Sande, Jan Danckaert, Serge Massar, Joni Dambre, Benjamin Schrauwen, Claudio R Mirasso, and Ingo Fischer. *Information processing using a single dynamical node as complex system*. Nature communications, 2:468, 2011.

[22] Yvan Paquot, Francois Duport, Antoneo Smerieri, Joni Dambre, Benjamin Schrauwen, Marc Haelterman, and Serge Massar. *Optoelectronic reservoir computing*. Scientific reports, 2, 2012.

[23] Laurent Larger, Miguel C Soriano, Daniel Brunner, Lennert Appeltant, Jose M Gutiérrez, Luis Pesquera, Claudio R Mirasso, and Ingo Fischer. *Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing*. Optics express, 20(3):3241–3249, 2012.

[24] Daniel Brunner, Miguel C Soriano, Claudio R Mirasso, and Ingo Fischer. *Parallel photonic information processing at gigabyte per second data rates using transient states*. Nature communications, 4:1364, 2013.

[25] Laurent Larger, Antonio Baylón-Fuentes, Romain Martinenghi, Vladimir S Udaltsov, Yanne K Chembo, and Maxime Jacquot. *High-speed photonic reservoir computing using a time-delay-based architecture: million words per second classification.* Physical Review X, 7(1):011015, 2017.

[26] Kristof Vandoorne, Wouter Dierckx, Benjamin Schrauwen, David Verstraeten, Roel Baets, Peter Bienstman, and Jan Van Campenhout. *Toward optical signal processing using Photonic Reservoir Computing.* Opt. Express, 16(15):11182–11192, Jul 2008.

[27] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman. *Experimental demonstration of reservoir computing on a silicon photonics chip.* Nature communications, 5, 2014.

[28] Martin Fiers, Thomas Van Vaerenbergh, Francis Wyffels, David Verstraeten, Benjamin Schrauwen, Joni Dambre, and Peter Bienstman. *Nanophotonic reservoir computing with photonic crystal cavities to generate periodic patterns.* IEEE transactions on neural networks and learning systems, 25(2):344–355, 2013.

[29] Michael Reck, Anton Zeilinger, Herbert J Bernstein, and Philip Bertani. *Experimental realization of any discrete unitary operator.* Physical review letters, 73(1):58, 1994.

[30] William R Clements, Peter C Humphreys, Benjamin J Metcalf, W Steven Kolthammer, and Ian A Walmsley. *Optimal design for universal multiport interferometers.* Optica, 3(12):1460–1465, 2016.

[31] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. *Tunable efficient unitary neural networks (EUNN) and their application to RNNs.* arXiv preprint arXiv:1612.05231, 2016.

[32] David AB Miller. *Perfect optics with imperfect components.* Optica, 2(8):747–750, 2015.

[33] Jacques Carolan, Christopher Harrold, Chris Sparrow, Enrique Martín-López, Nicholas J Russell, Joshua W Silverstone, Peter J Shadbolt, Nobuyuki Matsuda, Manabu Oguma, Mikitaka Itoh, et al. *Universal linear optics.* Science, 349(6249):711–716, 2015.

[34] Antonio Ribeiro, Alfonso Ruocco, Laurent Vanacker, and Wim Bogaerts. *Demonstration of a 4× 4-port universal linear circuit.* Optica, 3(12):1348–1357, 2016.

[35] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. *Deep learning with coherent nanophotonic circuits*. Nature Photonics, 11(7):441, 2017.

[36] Nicholas C Harris, Gregory R Steinbrecher, Jacob Mower, Yoav Lahini, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Seth Lloyd, and Dirk Englund. *Bosonic transport simulations in a large-scale programmable nanophotonic processor*. arXiv preprint arXiv:1507.03406, 2015.

2

Photontorch

Optimizing photonic circuits is hard. When designing photonic circuits one often has to take into account imperfect component models and variation in the components, an effect that quickly grows when many of these components are interconnected. Having a simulator that can deal with these imperfections and can possibly compensate for such variation could undeniably be a great asset.

For this reason, a new photonic circuit simulator was developed, called *Photontorch*, which was built on top of the machine learning library PyTorch[1] [1], which tries to at least partially resolve these issues by enabling very efficient optimization of photonic circuits by backpropagation through their physical parameters.

In this chapter, we will gradually construct the building blocks for creating a photonic circuit simulator (and optimizer). The main focus of this chapter is to introduce Photontorch as a framework to implement photonic models on top of a machine learning library like PyTorch, in order to get speed-ups due to GPU acceleration, as well as the capability to use machine-learning optimization techniques for circuit design. The main aim of this chapter is not to implement the most accurate dispersive circuit-solver. Rather, we will follow an approach similar to that of the commercially available simulator Caphe [2].

[1] PhotonTorch = Photon + PyTorch.

2.1 The wave equation

Every electromagnetic phenomenon can be described by Maxwell's equations:

$$\nabla \times \mathbf{E} = -\mu_0 \mu_r \frac{\partial \mathbf{H}}{\partial t} \qquad \nabla \cdot (\epsilon_0 \epsilon_r \mathbf{E}) = \rho \tag{2.1}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \epsilon_0 \epsilon_r \frac{\partial \mathbf{E}}{\partial t} \qquad \nabla \cdot (\mu_0 \mu_r \mathbf{H}) = 0, \tag{2.2}$$

where — in the most general case — ϵ_r and μ_r are the relative permittivity and permeability *tensors* respectively[2]. However, when working with simple linear optics in a dielectric material (such as for most integrated optics), μ_r can be considered to be 1, as a nonzero μ_r would imply a magnetic material. Similarly, the charge and current density ρ and $\mathbf{J}$ can also be set to zero. Moreover, for most materials and photonic structures, ϵ_r can be represented by a (piecewise) constant[3] scalar (instead of a tensor). Knowing all this, Maxwell's equations can be significantly simplified:

$$\nabla \cdot \mathbf{E} = 0 \tag{2.3}$$

$$\nabla \cdot \mathbf{H} = 0 \tag{2.4}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} \tag{2.5}$$

$$\nabla \times \mathbf{H} = \epsilon_0 \epsilon_r \frac{\partial \mathbf{E}}{\partial t} \tag{2.6}$$

By taking the curl of the last two identities (2.5) and (2.6), and using the first two identities (2.3) and (2.4), we get the electromagnetic wave equations:

$$\nabla^2 \mathbf{E} = \mu_0 \epsilon_0 \epsilon_r \frac{\partial^2 \mathbf{E}}{\partial^2 t} \tag{2.7}$$

$$\nabla^2 \mathbf{H} = \mu_0 \epsilon_0 \epsilon_r \frac{\partial^2 \mathbf{H}}{\partial^2 t} \tag{2.8}$$

These wave equations imply a propagation speed of

$$v = 1/\sqrt{\mu_0 \epsilon_0 \epsilon_r} \tag{2.9}$$

$$= c/n, \tag{2.10}$$

where we substituted $c = 1/\sqrt{\mu_0 \epsilon_0}$ as the speed of light in vacuum ($\epsilon_r = 1$). Performing this substitution implicitly also yields the relation between the refractive index of the material — which is defined as the factor with which the speed of light in a certain material is slower compared to the speed of light in vacuum — and its relative permittivity ϵ_r:

$$n = \sqrt{\epsilon_r} \tag{2.11}$$

[2] More on that in Chapter 4

[3] This implies that $\nabla \epsilon_r = 0$

This yields for the wave equations in a dielectric material:

$$\nabla^2\mathbf{E} = \frac{n^2}{c^2}\frac{\partial^2\mathbf{E}}{\partial^2 t} \tag{2.12}$$

$$\nabla^2\mathbf{H} = \frac{n^2}{c^2}\frac{\partial^2\mathbf{H}}{\partial^2 t}. \tag{2.13}$$

2.2 Waveguide modes

Although the wave equations seem decoupled, they are not: $\mathbf{E}$ and $\mathbf{H}$ are coupled through the curl equations (2.5) and (2.6). In waveguides and in general all structures with an axis of invariance (the propagation axis), the above wave equations yield orthogonal modes, for which however at a discontinuity boundary conditions apply. These boundary conditions arise from the Maxwell equations at the interface between two different refractive indices.

$$E_{\parallel}^a - E_{\parallel}^b = 0 \tag{2.14}$$

$$H_{\parallel}^a - H_{\parallel}^b = 0 \tag{2.15}$$

$$\epsilon^a E_{\perp}^a - \epsilon^b E_{\perp}^b = 0 \tag{2.16}$$

$$H_{\perp}^a - H_{\perp}^b = 0 \tag{2.17}$$

where $\parallel$ denotes a component parallel to the interface and $\perp$ denotes a component perpendicular to the interface.

The general idea for solving these coupled wave equations in this case is by finding the (eigen)modes Ψ which still satisfy the (uncoupled) wave equation:

$$\nabla^2\Psi(x,y,z,t) = \frac{n(x,y,z)^2}{c^2}\frac{\partial^2\Psi}{\partial^2 t}(x,y,x,t) \tag{2.18}$$

Let us consider for example a waveguide, i.e. a structure that is invariant along a certain direction. A general solution for a waveguide mode propagating in the z-direction is:

$$\Psi(x,y,z,t) = A(x,y)\exp\left(i(kz-\omega t)\right) \tag{2.19}$$

k is called the *propagation constant* or the wave number of the mode and is more often expressed in terms of the *effective index* and the *wavelength* of the light:

$$\beta = \frac{2\pi}{\lambda}n_{\text{eff}}. \tag{2.20}$$

Finding the *modes* and the corresponding effective indices analytically is beyond the scope of this chapter. It suffices to know that for simple waveguide structures, viewing the waveguide as an approximate *slab*-structure is often sufficient to find an approximate value for the effective index. For a more accurate value for the effective index, or for more complicated waveguide structures, dedicated numerical mode solvers are necessary.

2.3 Scattering matrices for linear components

In the previous section a short overview was given on how Maxwell's equations are used to describe optical systems. We briefly touched upon how this works for a waveguide and — even without going into too much detail — we saw that this is a quite tedious approach.

In fact, only for the simplest (and smallest) systems this is a feasible approach. If the system becomes more complex, one often resorts to numerical approximations, like the Finite-Difference Time Domain (FDTD) method, which discretizes Maxwell's equations on a grid[4]. However, when simulating a whole circuit of optical components, even the FDTD method becomes inconceivable.

However, this does *not* mean that the only option to figure out the behavior of a large photonic circuit is to measure it. One can use the measured or simulated properties of single component in the circuit to obtain the behavior of the larger circuit. One way to do this is by the *scattering* matrix (S-matrix) formalism.

The S-matrix formalism considers each component in the circuit as a *black box*, which relates the fields x' leaving the component to the fields x entering the component by a *linear* S-matrix. These S-matrices are then interlinked according to the circuit topology[5] to obtain the S-matrix of the larger circuit.

Consider the complex-valued optical *state* vector x, which describes the phase and amplitude in each *port* of a component, i.e. in each input/output mode of the component. If the component is *linear*, then — as also illustrated in Fig. 2.1 — the S-matrix accurately describes how the fields entering the component x are related to the fields exiting the component x'.

$$x' = Sx. \tag{2.21}$$

However, this S-matrix cannot take any arbitrary form. There are in fact quite some constraints on the S-matrix for most optical components.

The most important property of optical components is *reciprocity*. A component is reciprocal if it is made from materials for which the permittivity ϵ and the permeability μ are symmetric. Most — if not *all* — optical components satisfy this requirement[6]. Reciprocity implies that transmission between port i and port j does not depend on the propagation sense, hence we have:

$$S = S^T \tag{2.22}$$

[4] More on this in Chapter 4

[5] How this exactly works will be discussed later in 2.4 and 2.5.4

[6] Materials which are *not* reciprocal *need* to have magnetic properties and are often used for creating optical isolator: components that only transmit light in a single direction. However, the study of these kind of components is beyond the scope of this work, hence reciprocity is assumed throughout.

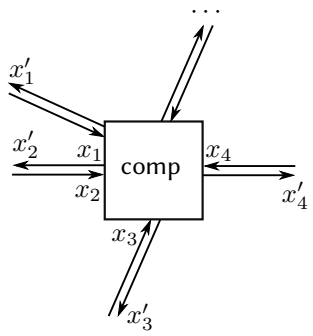


Figure 2.1: In the S-matrix formalism, a component is considered a black box which changes the input fields x_i to the output fields x'_i.

Furthermore, for *passive* components, i.e. components that do not add energy to the circuit, we have that $|x'| < |x|$, and thus:

$$x^H x - x'^H x' = x^H (I - S^H S) x \geq 0, \tag{2.23}$$

which implies that the matrix $I - S^H S$ is semi-positive definite. In the special case of lossless components, the requirement (2.23) turns into an equality

$$S^H S = I, \tag{2.24}$$

implying that the action of a lossless passive component is *unitary*.

2.3.1 Waveguide S-matrix

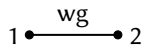


Figure 2.2: Waveguide schematic

The S-matrix of a lossless waveguide without reflection can be represented by the following S-matrix, which follows from (2.19):

$$S^{\text{wg}} = \begin{pmatrix} 0 & \exp(\frac{2\pi i}{\lambda} n_{\text{eff}} L) \\ \exp(\frac{2\pi i}{\lambda} n_{\text{eff}} L) & 0 \end{pmatrix}, \tag{2.25}$$

where L is the length of the waveguide, λ is the wavelength of the light and n_{eff} is the (possibly wavelength-dependent) *effective index* of the waveguide; a quantity that can be obtained from an eigenmode solver.

Looking at the form of the S-matrix, we see it makes sense: the field x_1 at port 1 becomes the output field x'_2 at port 2 and vice-versa. Moreover, a phase

factor proportional to the length of the waveguide is introduced, just like we would expect.

If loss has to be taken into account, the S-matrix can be multiplied by a global *attenuation* factor A. For example, for a waveguide with N dB/m loss[7]:

$$A(L) = 10^{-NL/20} \tag{2.26}$$

2.3.2 Directional coupler S-matrix

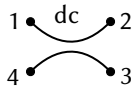


Figure 2.3: Directional coupler (dc) schematic

A directional coupler is defined as a component that *couples* two waveguides together. In terms of the four ports of this device, the S-matrix can be written down as[8]

$$S^{\text{dc}} = \begin{pmatrix} 0 & \tau & i\kappa & 0 \\ \tau & 0 & 0 & i\kappa \\ i\kappa & 0 & 0 & \tau \\ 0 & i\kappa & \tau & 0 \end{pmatrix} \tag{2.27}$$

This S-matrix satisfies (2.23). The parameters κ and τ can be related to the *coupling* and *transmission* of the directional coupler. Although these parameters could in principle chosen to be complex-valued, they are often represented by real numbers, as — when building larger circuits — any additional phase can be can be absorbed by a waveguide S-matrix. For the same reason, the directional coupler is often considered to be *lossless*, which implies due to (2.24) that

$$\tau^2 + \kappa^2 = 1, \tag{2.28}$$

just like we would expect.

In practice, the phenomenological parameters κ and τ depend on all kinds of different physical parameters, such as the coupling length of the directional coupler, the gap between the waveguides and so on. Just like for the *effective index* of a waveguide, it is often useful to abstract away a lot of those details and distill the S-matrix down to its core properties, knowing that these abstract parameters can always be related to physical properties if need be.

[7] The factor $1/20$ is coming from the fact that we are working with light *amplitude*, not the *intensity* (which would give a factor $1/10$).

[8] Note that the port order as illustrated in Fig. 2.3 is important, as a different port order will result in a differently ordered S-matrix. The two should always be defined together.

2.4 Circuits of linear components

When defining a circuit of N components with S-matrices $S_1, S_2, \ldots, S_N$ acting on the incoming fields represented by the vectors $x_1, x_2, \ldots, x_N$, one can write the collective action of these components as:

$$\begin{cases} x'_1 &= S_1 x_1 \\ x'_2 &= S_2 x_2 \\ &\vdots \\ x'_N &= S_N x_N \end{cases} \tag{2.29}$$

This can be put into matrix form by defining the joint S-matrix of all the components to be the block-diagonal matrix with each block being the S-matrix of each component individually.

$$S = \begin{pmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_N \end{pmatrix}, \tag{2.30}$$

In that way, when acting on all the fields at the same time, one retrieves (2.21), i.e. the same S-matrix equation as for a single component:

$$x' = \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_N \end{pmatrix} = \begin{pmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = Sx \tag{2.31}$$

Note that S_n and x_n for $n \in [1, N]$ are matrices and vectors themselves, and thus we have for the total number of elements of the combined S-matrix, i.e. the total number of *ports* P that $P \geq N$.

However, this is not the complete story, as the above definition for the S-matrix for multiple components does not include any *connectivity* information *between* the components. To achieve this, a *connection matrix* C has to be defined as well. The connection matrix describes instantaneous interconnections between the ports of individual components, as is illustrated in Fig. 2.4. The C-matrix thus describes how the output field vector x' gets transformed back into an input field vector x. Furthermore, it also describes how fields that do not belong to the circuit, the external fields x_{ext}, are connected to the circuit:

$$\begin{pmatrix} x \\ x'_{\text{ext}} \end{pmatrix} = C \begin{pmatrix} x' \\ x_{\text{ext}} \end{pmatrix} = C \begin{pmatrix} Sx \\ x_{\text{ext}} \end{pmatrix} \tag{2.32}$$

Per definition, we have that $C = C^T$ as each connection is assumed bidirec-

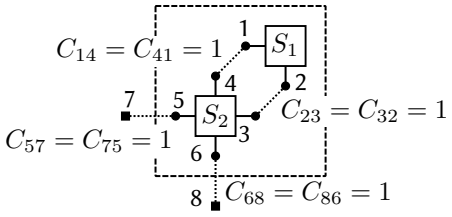


Figure 2.4: A visual representation of (2.32): two components with S-matrices S_1 (ports 1,2) and S_2 (ports 3,4,5,6) are interconnected by a connection matrix C, which also connects the rest of the ports to the output ports (7,8).

tional. Non-bidirectional connections can be implemented with non-reciprocal S-matrices. Furthermore, the connection matrix can be split up into a part C_{int} responsible for the *interconnections* between the individual components and a part C_{ext} that connects the leftover internal ports (which are not interconnected to other components of the circuit) to the external circuit ports . This leads to the following equation in terms of the split-up connection matrix:

$$\begin{pmatrix} x \\ x'_{\text{ext}} \end{pmatrix} = \begin{pmatrix} C_{\text{int}} & C_{\text{ext}} \\ C_{\text{ext}}^T & 0 \end{pmatrix} \begin{pmatrix} Sx \\ x_{\text{ext}} \end{pmatrix}. \tag{2.33}$$

Note that the input fields of the circuit are the output fields of the external ports and vice versa, hence the apparent (but correct) reversal of x_{ext} and x'_{ext} which we define here to relate to the circuit (not to the output ports). This yields two matrix equations, the first of which can be inverted such that we can find a relation between the input and output fields of the circuit:

$$x'_{ext} = C_{\text{ext}}^T S (I - C_{\text{int}})^{-1} C_{\text{ext}} x_{\text{ext}} \tag{2.34}$$

This equation is entirely independent of the internal fields x of the circuit, and relates the fields coming into the circuit x_{ext} to the fields going out of the circuit x'_{ext} in a linear way. This allows us to define the reduced S-matrix of a linear circuit as follows:

$$S_{\text{circuit}} = C_{\text{ext}}^T S (I - C_{\text{int}} S)^{-1} C_{\text{ext}} \tag{2.35}$$

This formula basically solves the circuit in the frequency domain and is an alternative to the more traditional *transfer matrix* method [3]. Moreover, this is a very useful formula as it allows to reduce the number of nodes (and thus the memory requirements in simulation) of any linear photonic circuit.

2.4.1 Ring resonator S-matrix

To see how this scattering matrix formalism can be useful, let us consider a ring resonator. A ring resonator can be considered to be a directional coupler which is connected onto itself by a waveguide as illustrated in Fig. 2.5.

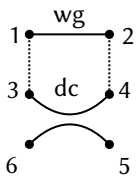


Figure 2.5: A ring resonator consists of a directional coupler connected onto itself by a waveguide.

Let us first address the more traditional way of approaching this problem. In this ring resonator, light resonates inside the ring. As discussed in 2.3.1, the change of amplitude of the light after transmission through the waveguide can be described as

$$\phi = S_{21}^{\text{wg}} = S_{12}^{\text{wg}} = A(L) \exp\left(\frac{2\pi i}{\lambda} n_{\text{eff}} L\right) \tag{2.36}$$

with $A(L)$ a loss-factor depending on the length of the waveguide. Using this knowledge, assume light is inserted into the ring resonator along port 6 in the ring resonator illustrated in Fig. 2.5; the light will be coupled to port 4 with a coupling efficiency $i\kappa$. After this, the light will enter the waveguide (along port 2). After transmission through the waveguide — resulting in the extra phase ϕ from (2.36) — the light leaves the waveguide (at port 1) and enters the directional directional coupler again through port 3, after which it gets transmitted to port 4 with a transmission efficiency τ. This process repeats over and over, which yields for the light amplitude and phase at port 3 the following infinite sum:

$$x_3 = x_6 i\kappa\phi(1 + \tau\phi + (\tau\phi)^2 + \cdots) \tag{2.37}$$

$$= x_6 i\kappa\phi \frac{1}{1 - \tau\phi}, \tag{2.38}$$

which means that, to have the amplitude at the output of the ring resonator — port 5 — this quantity has to be multiplied one last time by $i\kappa$, while also adding the direct straight connection ($6 \to 5$), introducing a separate factor τ:

$$x_5 = -x_6\kappa^2\phi\frac{1}{1 - \tau\phi} + \tau x_6 \tag{2.39}$$

$$= x_6 \frac{\tau - \phi}{1 - \tau\phi} \tag{2.40}$$

When substituting ϕ, this is recognized as the widely known transmission of an all-pass filter:

$$x_{\text{out}} = \frac{\tau - A(L)\exp\left(\frac{2\pi i}{\lambda} n_{\text{eff}} L\right)}{1 - \tau A(L)\exp\left(\frac{2\pi i}{\lambda} n_{\text{eff}} L\right)} x_{\text{in}} \tag{2.41}$$

This approach is, however, tedious and prone to errors. Moreover, it does not generalize well to more complicated circuits. Let us see how the reduced S-matrix approach handles this in a more general and systematic way.

Before we do anything, we need to modify Fig. 2.5 a bit. As discussed before, the approach requires the unconnected internal ports to be connected to *output ports* of the circuit, which in the new figure Fig. 2.6, are labeled as port 7 and 8.

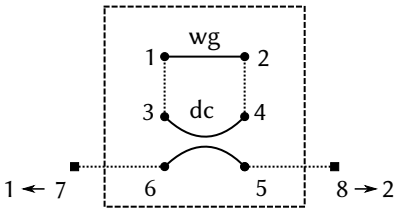


Figure 2.6: A ring resonator consists of a directional coupler connected onto itself by a waveguide. The outputs of the directional coupler are coupled to the output ports 7 and 8, which after reducing the S-matrix can be relabled as port 1 and 2 of the ring resonator circuit.

When the circuit is defined like this, we have a circuit with 8 ports: 6 internal ports (the actual circuit) and 2 external ports. The joint S-matrix is defined as the block-diagonal matrix of the 2 component S-matrices:

$$S = \begin{pmatrix} 0 & \phi & 0 & 0 & 0 & 0 \\ \phi & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \tau & i\kappa & 0 \\ 0 & 0 & \tau & 0 & 0 & i\kappa \\ 0 & 0 & i\kappa & 0 & 0 & \tau \\ 0 & 0 & 0 & i\kappa & \tau & 0 \end{pmatrix}, \tag{2.42}$$

The complete connection matrix of the circuit is defined as follows:

$$C = \left(\begin{array}{cccccc|cc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right) = \left(\begin{array}{c|c} C_{\text{int}} & C_{\text{ext}} \\ \hline C_{\text{ext}}^T & 0 \end{array}\right) \tag{2.43}$$

When the equation for the reduced S-matrix (2.35) is applied, we get after some derivations:

$$S_{\text{ring}} = \begin{pmatrix} 0 & -\frac{\kappa^2\phi}{1-\phi\tau} + \tau \\ -\frac{\kappa^2\phi}{1-\phi\tau} + \tau & 0 \end{pmatrix}, \tag{2.44}$$

which corresponds perfectly with (2.39).

It is important to note however, that although this S-matrix is certainly correct in the frequency domain, it might be an incorrect representation in the time domain. This is because when reducing the S-matrix, one implicitly assumes no delays in the ring. When the delay of the ring itself is important, i.e. when the bandwidth of the time-dependent signal is comparable to the bandwidth of the ring resonator, a more elaborate port-reduction is necessary, which will be introduced in 2.5.4.

2.4.2 Mach-Zehnder Interferometer S-matrix

The Mach-Zehnder Interferometer (MZI) is a typical building block of many photonic circuits. It is often used as a tunable directional coupler, as tuning the coupling of a directional coupler directly is often infeasible to perform accurately.

The most general MZI is often defined as two $50/50$ directional couplers connected to each other by two waveguides with a mutual phase difference of 2θ, while sometimes an additional input phase ϕ is also taken into account, as illustrated in Fig. 2.7.

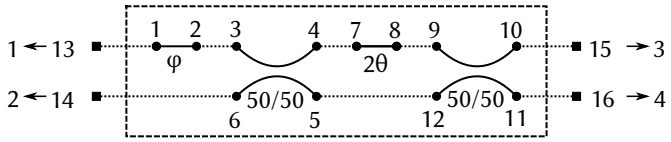


Figure 2.7: In simulation, a general MZI can be constructed from two waveguides and two $50/50$ directional couplers.

Applying the S-matrix reduction (2.35) on this circuit yields for its S-matrix:

$$S_{\text{MZI}} = -ie^{i\theta} \begin{pmatrix} 0 & 0 & -e^{i\phi}\sin(\theta) & e^{i\phi}\cos(\theta) \\ 0 & 0 & \cos(\theta) & \sin(\theta) \\ -e^{i\phi}\sin(\theta) & \cos(\theta) & 0 & 0 \\ e^{i\phi}\cos(\theta) & \sin(\theta) & 0 & 0 \end{pmatrix}. \tag{2.45}$$

Ignoring the global phase factor $-ie^{i\theta}$ and just focusing on the matrix transformation M from ports $(1,2) \to (3,4)$, we have

$$M = \begin{pmatrix} -e^{i\phi}\sin(\theta) & e^{i\phi}\cos(\theta) \\ \cos(\theta) & \sin(\theta) \end{pmatrix}, \tag{2.46}$$

which is known as the most general representation of the $SU(2)$ group [4], which means that by cascading these components, one can create any unitary matrix [5, 6], a property we will use later in section 2.8.4 to simulate a *Unitary Recurrent Neural Network* (URNN) with photonic components in Photontorch.

2.5 Towards a general circuit

2.5.1 Delay-introducing linear components

When considering linear photonic components in the time domain, it is very often the case that — although they act linearly — some components introduce a delay. The simplest example of this is a waveguide, for which the S-matrix accurately describes the phase change this component introduces, but not the time delay.

To accurately simulate such components in the time domain, in principle, proper Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) methods should be used. However, for many low-dispersive circuits, the approach followed by Photontorch is to model each non-dispersive component at hand by a single delay per port. The choice of only using a single delay per port of course has the consequence that any Group Velocity Dispersion (GVD) and by extension any higher order dispersion effects are not taken into account. To accurately and efficiently model dispersive circuits, methods like *vector fitting* (VF) [7, 8] should be used. As mentioned in the upcoming section 2.5.2, Photontorch is flexible enough to incorporate alternative governing equations by an approach like VF and to define a different non-linear component implementing those equations. However, this was not the main focus of this chapter.

The most simple relation for a time-delaying component that can still be described by an S-matrix is the following, where we assume the component or network of components introduces a time-delay dt:

$$x'(t) = Sx(t - dt). \tag{2.47}$$

However, this equation is not general enough, as in general each port $p \in [1, P]$ of a component or network of components can have a different delay:

$$\begin{pmatrix} x'_1(t) \\ x'_2(t) \\ \vdots \\ x'_P(t) \end{pmatrix} = S \begin{pmatrix} x_1(t - dt_1) \\ x_2(t - dt_2) \\ \vdots \\ x_P(t - dt_P) \end{pmatrix} \tag{2.48}$$

However, this is something that is quite easily solved by keeping a buffer in memory from where the relevant fields are sampled before applying the S-matrix. Let us define the buffer operation to be:

$$B \star x(t) = \begin{pmatrix} x_1(t - dt_1) \\ x_2(t - dt_2) \\ \vdots \\ x_P(t - dt_P) \end{pmatrix}. \tag{2.49}$$

Then, equation (2.48) becomes:

$$x'(t) = S\left(B \star x(t)\right) \tag{2.50}$$

In fact, one can argue that even this relation is not general enough, as one can imagine a different delay for each *interconnection* between two ports. This can be represented by a delay matrix D with a similar form as the S-matrix:

$$x'(t) = S\left(D \star x(t)\right) \tag{2.51}$$

This delay matrix would then basically introduce a different delay for each non-zero element of the S-matrix.

In Photontorch by default, delays are implemented with the buffer operation (2.50), as this operation can much more easily be parallelized and applied for the whole circuit simultaneously (see also 2.6). Moreover, in most cases, the node-based delay on the *component* level is sufficient anyway.

As a shorthand notation, we will write

$$x'(t) = S \star x(t) \tag{2.52}$$

Were $S\star$ is a general notation for either one of the previous two operations. Note that this operation obviously reduces to the instantaneous S-matrix operation when every $dt_p = 0$.

2.5.2 Non-linear components

So far, we have a decent framework for simulating linear components with and without time delay. We can extend this even further to include non-linear time-dependent components, i.e. components for which the action changes with time. These kind of components can often be described by an internal state u, which is governed by an ordinary differential equation (ODE):

$$\frac{\partial u}{\partial t}(t) = f(t, u, x) \tag{2.53}$$

This ODE depends on time, the internal state u and the incoming fields x. An equation like this can be turned into an update equation by using Euler or Runge-Kutta integration. On top of the ODE, one needs a second relation, which relates the internal state of the component to the outgoing field amplitudes x':

$$x' = g(t, u, x). \tag{2.54}$$

For many non-linear components, such as Semiconductor Optical Amplifiers (SOA) the relation g turns out to be an exponential relationship which only depends on u [9]. This, of course, has the nice interpretation that the internal state can be interpreted as the *gain* of the SOA.

2.5.3 Network terminations

When discussing the circuit of linear components in 2.4, we introduced the concept of network output nodes. In the discussion there, they were treated differently from the *real* network nodes, which are part of the components of the network. However, it is sometimes useful to think of these output nodes as components themselves, which are described by the following relation:

$$x'(t) = s(t). \tag{2.55}$$

Here, $s(t)$ is a time-varying *source*-function. This, in fact, looks like a simplified version of equation (2.54) above with the sole difference that it does not depend on an internal state, nor does it depend on incoming or outgoing fields, which means that the S-matrix describing such a component is zero. We'll call these kind of components network *terminations* as they act as a sink to all fields coming in due to their zero S-matrix and as a source due to the possibly non-zero source term $s(t)$.

2.5.4 A general circuit

Taking everything we have seen so far into account, we can propose a general relation between input fields and output fields of a possibly non-linear and time-dependent circuit:

$$x'(t) = S \star x(t) + g\left(t, x(t), x(t-dt), \ldots\right) \tag{2.56}$$

Also, just like for the linear circuit, we can define the connection matrix C to map the outgoing fields back onto the ingoing fields:

$$x(t) = Cx'(t) \tag{2.57}$$

When using terminations for every open port in the circuit, we must have that

$$\sum_i C_{ij} = \sum_i C_{ji} = 1 \quad \forall j, \tag{2.58}$$

which means that the network cannot be connected to other components, as it has no free ports left. We will call a network like this *fully-connected*. For these kind of networks, we can find a rearrangement of the ports in such a way that they are split up into memory-less (ML) ports, i.e. ports that only rely on an instantaneous S-matrix operation and memory-containing (MC) ports, which are all other ports belonging to components which are time-delayed and/or non-linear:

$$\begin{pmatrix} x^{\text{ML}} \\ x^{\text{MC}} \end{pmatrix} = \begin{pmatrix} C^{\text{MLML}} & C^{\text{MLMC}} \\ C^{\text{MCML}} & C^{\text{MCMC}} \end{pmatrix} \begin{pmatrix} S^{\text{MLML}} & 0 \\ 0 & S^{\text{MCMC}} \end{pmatrix} \star \begin{pmatrix} x^{\text{ML}} \\ x^{\text{MC}} \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix} \tag{2.59}$$

$$= \begin{pmatrix} C^{\text{MLML}} & C^{\text{MLMC}} \\ C^{\text{MCML}} & C^{\text{MCMC}} \end{pmatrix} \begin{pmatrix} S^{\text{MLML}} x^{\text{ML}} \\ S^{\text{MCMC}} \star x^{\text{MC}} \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix} \tag{2.60}$$

For these kind of top-level networks, the number of ports can be reduced just like in the purely linear case by using the fact that the ML part of the equation is independent of time and thus can be inverted into an equation which only describes the connectivity behavior between the MC ports to which a generalized source term g is added:

$$\begin{aligned} x^{\text{MC}} &= \left(C^{\text{MCMC}} + C^{\text{MCML}} \cdot S^{\text{MLML}} \cdot (1 - C^{\text{MLML}} S^{\text{MLML}})^{-1} C^{\text{MLMC}}\right) S^{\text{MCMC}} \star x^{\text{MC}} + g \\ &= \tilde{C} S^{\text{MCMC}} \star x^{\text{MC}} + g, \end{aligned} \tag{2.61}$$

Here we can define the reduced connection matrix $\tilde{C}$ as:

$$\tilde{C} = \left(C^{\text{MCMC}} + C^{\text{MCML}} \cdot S^{\text{MLML}} \cdot (1 - C^{\text{MLML}} S^{\text{MLML}})^{-1} C^{\text{MLMC}}\right), \tag{2.62}$$

which describes all the instantaneous connections in the circuit[9]. Note that although the name implies that it is a connection matrix, this reduced connection matrix is not a binary matrix: it contains all the S-matrix information of the ML nodes as well. We have now that $S^{\text{MCMC}}\star$ only encodes delayed interactions. Moreover, if we assume that g also only encodes delayed interactions[10], then the right hand side of (2.61) only depends on times in the past:

$$x^{\text{MC}}(t) = \tilde{C} S^{\text{MCMC}} \star x^{\text{MC}}(t) + g(t, x(t - dt), ...) \tag{2.63}$$

This means that we have found an update equation for time domain simulations. In the following, we will drop the superscripts and work only with the MC nodes, assuming the reduction is already performed.

2.5.5 Carrier Modulation

A simplification that is often made is to simulate only the envelope of the signal by ignoring the carrier frequency $e^{i\omega t}$. Especially for single-frequency signals, this is a practical approach as it allows for a much larger time step to be used in simulation. For signals with a certain bandwidth, these carrier frequencies should be included as otherwise the carrier frequency beating will be ignored; an effect that is important when studying non-linear optical effects such as four-wave mixing.

2.5.6 A double ring in the time domain

To see how accurate the currently implemented circuit approach is in the presence of group velocity dispersion, we simulate the double-ring circuit visualized

[9] This expression contains an inverse, which is almost never a good idea to perform explicitly. Instead a (differentiable) solver is used.

[10] Most realistic non-linear photonic components introduce a delay anyway.

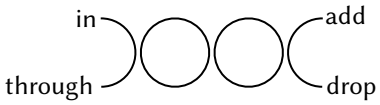


Figure 2.8: A double ring add-drop filter. The first ring has a circumference of $20\,\mu$m, while the second ring has a circumference of $20.01\,\mu$m.

in Fig. 2.8 in the time domain and compare the time traces to the time traces of an identical circuit simulated with Lumerical Interconnect, which uses an FIR-based approach.

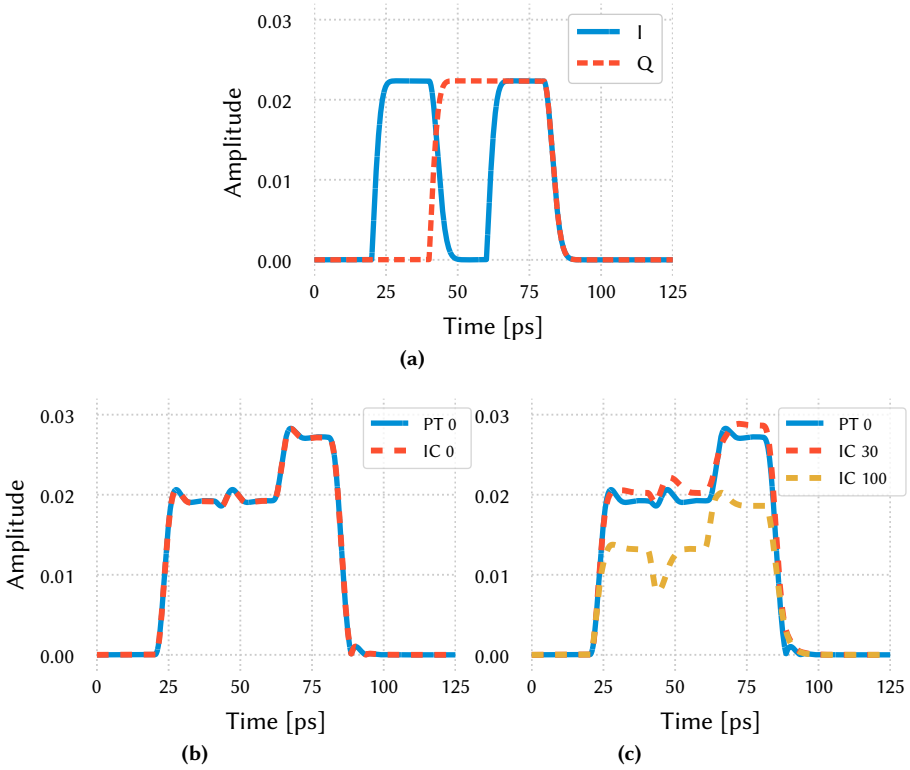


Figure 2.9: (a) 4-QAM modulated input sent through the double ring circuit. (b) Response without GVD for Photontorch (PT) and Interconnect (IC). (c) Response with GVD for Interconnect (30 and $100\,\mathrm{ps/(nm \cdot km)}$) compared to the Photontorch response (no GVD).

The structure simulated has two rings with slightly different circumferences ($20.00\,\mu$m and $20.01\,\mu$m respectively). The rings have an effective index of 2.35 and a group index of 4.3 at a wavelength of $1550\,$nm. A 4-QAM modulated input

signal (shown in Fig. 2.9a) is sent through the circuit and the responses are shown, respectively for no GVD (Fig. 2.9b) and GVD $= 30\,\mathrm{ps}/(\mathrm{nm}\cdot\mathrm{km})$ and GVD $= 100\,\mathrm{ps}/(\mathrm{nm}\cdot\mathrm{km})$ (Fig. 2.9c).

It is clear that, for a dispersive circuit like the ring-circuit here, having a simulator that accurately models the group velocity dispersion is important for higher values of the dispersion.

2.6 Highly parallel simulations with Photontorch

As of 2019, there are a handful of simulators for designing photonic integrated circuits, such as Aspic [10], Luceda Caphe [2], Lumerical Interconnect [11] and VPI Photonics [12]. All are excellent circuit simulation tools for their particular purpose. However, some of these photonic circuit simulation tools are not well suited for parallel simulations and for many of them, optimizing a circuit means nothing more than just sweeping the parameters, which quickly becomes unwieldy when the number of parameters or components in the circuit starts to grow.

Photontorch, however, is written in Python and uses PyTorch tensors [13] to describe the parameters and S-matrices of the components. PyTorch tensors are highly optimized arrays, which, as opposed to the more commonly used Numpy ndarrays [14], can be placed on the Graphical Processing Unit (GPU) of a computer, automatically enabling highly parallelizable simulation of photonic circuits simulations.

However, to harness the true power of those GPUs, some modifications to the update equation (2.63) need to be implemented. The main difference being that the field vector x inside the circuit will be defined in Photontorch as a multidimensional *tensor* with two extra dimensions: the number of wavelengths or *modes* m and the number of simulations done in parallel, i.e. the number of *batches* b. Think of the latter as the result of multiple different input waveforms, computed in parallel.

Furthermore, the wavelength dependence of the S matrix requires us to define a different S matrix for the circuit for each wavelength or mode m, resulting in a 3d tensor with one of the dimensions the number of wavelengths. The parallelized version of the update equations (2.63) for a network with N memory containing nodes becomes in this case:

$$x_{(q+1)mnb} = \sum_{i}^{N}\sum_{j}^{N} \tilde{C}_{mni} S_{mij} \star x_{qmjb} + g(qdt, x_{(q-1)mnb}, \ldots), \tag{2.64}$$

Here, we made the additional discretization in time such that

$$x_{qmnb} = x_{mnb}(qdt). \tag{2.65}$$

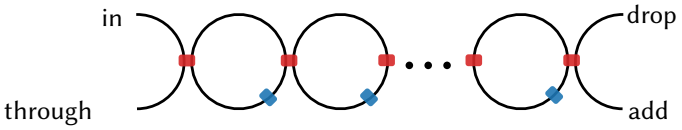


Figure 2.10: A CROW is an add-drop filter with extra rings. Each CROW with n rings has $n + 1$ couplings (red) and n phase shifts (blue).

This parallelization means that as long as the GPU-memory is not full, one can increase the number of simulations done in parallel both in the frequency domain (number of wavelengths/modes) and in the time domain (number of distinct input waveforms or batches) without much overhead, which is especially important during optimization of circuits.

Note furthermore that one can parallelize both over the modes and batches at the same time, which is especially useful for time domain simulations if the responses of different waveforms at different wavelengths need to be evaluated at the same time. Furthermore, since wavelengths are evaluated simultaneously, nothing is stopping us from defining wavelength-mixing (non)linear components.

2.7 Performance metrics

In this section, the performance of CPU-based computation will be compared to the performance of GPU-based computation. We will make a distinction between parallelized execution, where the update equations (2.64) are followed, and non-parallelized execution, where the update equations (2.63) are followed. In the latter case, each additional wavelength or batch has to be sent through the circuit in sequence, which will increase the simulation time linearly with a unit slope.

We choose to benchmark the performance by simulating the large Coupled Resonator Optical Waveguide (CROW) illustrated in Fig. 2.10. A CROW is a good circuit for benchmarking simulation speed, as it allows to easily add new rings to increase the difficulty of the simulation. Other parameters that can be tweaked during a CROW circuit simulation are the number of wavelengths simulated simultaneously and the number of parallel simulations performed at once (batched execution). All simulations were performed on a normal desktop computer with an Intel i7-4790K CPU with 16GB RAM. For the GPU simulations we used an Nvidia GTX-1060 (6GB) GPU.

We first simulate a CROW for multiple wavelengths in the frequency domain. In Fig. 2.11a, we clearly see an almost linear behavior in terms of the GPU performance after adding additional rings to the network, whereas CPU simulation

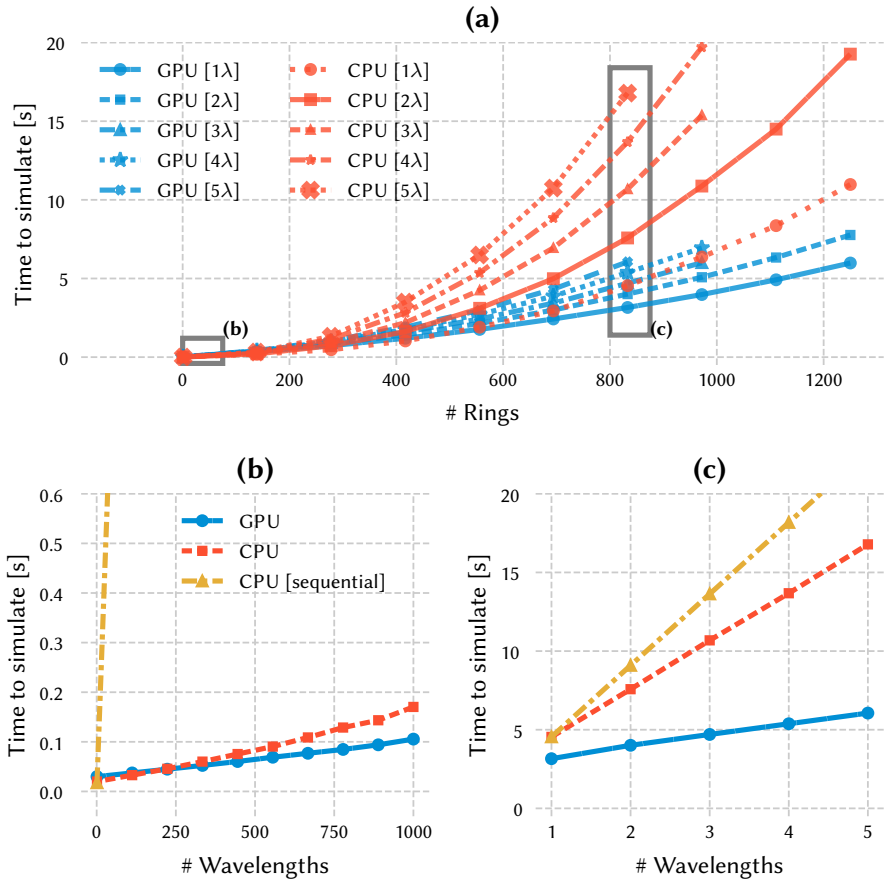


Figure 2.11: Simulation times to simulate a CROW circuit in the frequency domain. (a) A CROW simulated on a GPU shows an almost linear increase in simulation times, whereas CPU simulation times increase much faster. (b) We can zoom in on the beginning of this graph, where we simulate a CROW with just 10 rings, but for many waveguides simultaneously. We see that especially in this regime, being able to simulate for many wavelengths concurrently yields enormous benefits over the sequential simulation approach often used by other frameworks. (c) Even when the number of rings increases to 850 it stays more interesting to use the concurrent approach.

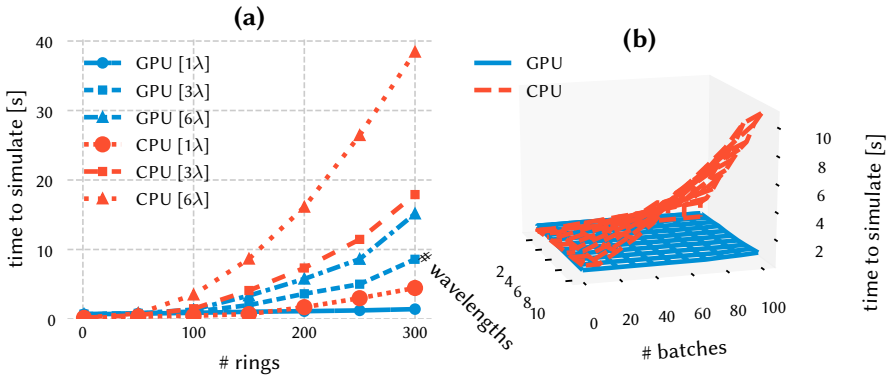


Figure 2.12: Using a GPU becomes even more appropriate when simulating in the time domain. (a) Here, the performance was tracked for a single simulation (batch) of 2000 time steps for 1, 3 and 6 wavelengths at once respectively. (b) The performance for simulating a 10-ring CROW for multiple wavelengths and multiple parallel simulations.

time increases much faster. This is of course because simulating the frequency domain response of a circuit basically comes down to calculating the reduced connection matrix (2.62) of the circuit, something that becomes quadratically more difficult for an increasing number of nodes. Due to its inherent parallel nature, this operation can be done more efficiently on a GPU. Moreover, simulating additional wavelengths at once is always faster than the sequential simulation, as can be seen in Fig. 2.11b and Fig. 2.11c.

To benchmark the time-domain performance, we chose to simulate a CROW for 2000 time steps for an increasing number of rings and wavelengths, as can be seen in Fig. 2.12a. It is clear from this figure that simulation times for the CPU start to approach the unit slope early on, while for the GPU this transition happens a lot later.

When simulating just 10 rings for multiple batches and multiple wavelengths, we clearly see in Fig. 2.12b that the GPU performance is barely affected.

The simulation time for the 10-ring CROW for a single wavelength and single batch on a CPU using just one core is about $200\,\mathrm{ms}$. Assume now that we want to perform 100 simulations (batch size of 100) at once, each for 10 different wavelengths. The naive sequential execution would result in a $200\,\mathrm{s}$ execution time, while the parallelized execution times clock down at $10\,\mathrm{s}$ and $1\,\mathrm{s}$ for the CPU and GPU respectively, as can be seen in Fig. 2.12b. This means that we get up to a $200\times$ speed-up for the GPU, while even for the CPU a speed-up of $20\times$ can be achieved because of the efficient multi-threading.

The performance of Photontorch (both on CPU and GPU) for simulating such

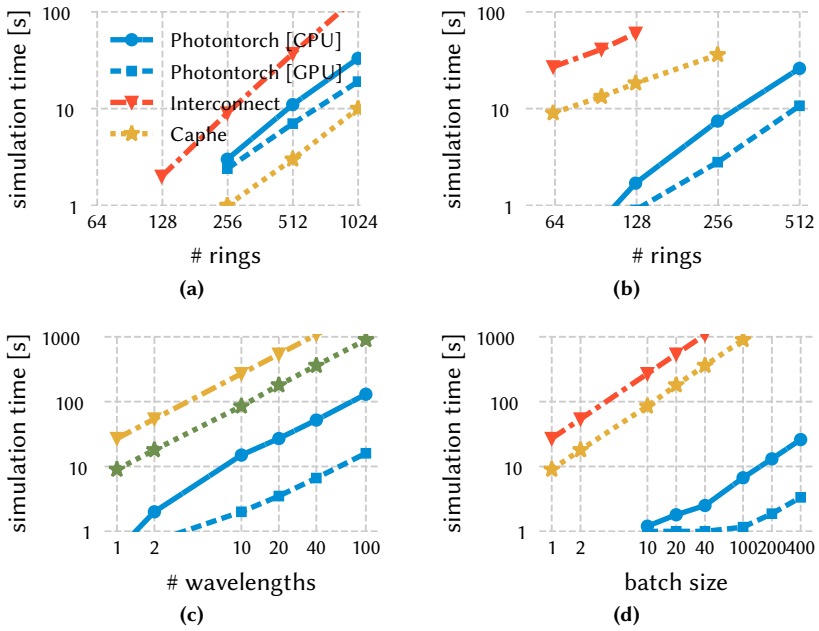


Figure 2.13: The performance for Photontorch simulating a CROW, both in the frequency domain and the time domain, was also compared to *Lumerical Interconnect* and *Caphe*. (a) The time needed to find the frequency response for a CROW of increasing number of rings. The performance of Photontorch lies somewhere in between the Caphe and Interconnect. (b) The time needed to do a time-domain simulation of 3000 time steps for an increasing number of rings. The simulation time of Photontorch is practically zero up to about 100 rings. (c) Performance for a multi-mode time-domain simulation for a CROW of 64 rings and an increasing number of wavelengths. (d) Performance for a time-domain simulation of a CROW with 64 rings for a single wavelength but for an increasing number of input waveforms (batch size).

a CROW was also compared to other photonic simulators, such as *Lumerical Interconnect* and *Luceda Caphe*.

First, the response of a CROW in frequency domain was calculated. For this task, Photontorch is outperformed by Caphe, but performs significantly better than Interconnect, as can be seen in Fig. 2.13a. Caphe performs better in this regard due to its possibly more efficient solver to solve for a large system of equations necessary to find the reduced connection matrix of the CROW[11]. This solver utilizes a factorization method for sparse systems [2], which is currently not available in Photontorch's PyTorch backend, but could conceivably be added.

However, once the reduced connection matrix for the circuit is found, Photontorch vastly outperforms both Caphe and Interconnect in time-domain simulations of the CROW, as can be seen in Fig. 2.13b-d, where a CROW was simulated for 3000 time steps. Indeed, in Fig. 2.13b, one sees that Photontorch outperforms both Caphe and Interconnect for a time-domain simulation of a CROW with an increasing number of rings. Moreover, simulating additional wavelengths at once for a CROW with 64 rings is always faster than the sequential simulation required by Caphe and Interconnect, as can be seen in Fig. 2.13c. Similarly, simulating multiple input wave forms at once (batched execution) for a CROW with 64 rings at a single wavelength generates almost no overhead in Photontorch, especially on a GPU, as can be seen in Fig. 2.13d. Note however that Interconnect uses an FIR-based approach, which — as we've seen in 2.5.6 — can be more accurate than the Caphe/Photontorch approach.

2.8 Optimization of photonic circuits through backpropagation

Apart from its parallel nature, Photontorch can also be used to efficiently optimize large photonic circuits through backpropagation. As we have seen in 1.6, backpropagation is a well-established optimization method which is traditionally used to optimize the many parameters of large deep neural networks. Many deep learning frameworks exist today to help with the process of backpropagation. Generally speaking, these deep learning frameworks keep track of the gradients of each operation and of the order of operations (the computational graph) to enable automatic backpropagation.

Photontorch uses exclusively PyTorch [13] operators and data structures, which means that for each circuit operation, PyTorch will know how to perform the backpropagation through it. This allows us to optimize complex photonic circuits as if they were recurrent neural networks. This is a completely new optimization paradigm for physical systems enabled by the rapid advancement of

[11] To preserve differentiability, Photontorch is required to use a standard PyTorch solver.

deep learning. This way of optimizing photonic circuits is in many cases vastly more efficient than sweeping the parameters of the circuit or optimizing through genetic algorithms, as a much smaller portion of the parameter space has to be explored.

Photonic circuits are typically recurrent in nature, which will have an effect on how effective backpropagation is, as exploding gradients and vanishing gradients are common problems for large recurrent neural networks [15]. In deep learning these problems are often solved by using specialized recurrent modules such as the well-known Long Short-Term Memory (LSTM) cell [16] or the Gated Recurrent Unit (GRU) [17]. However, recent advances have shown that recurrent deep learning with unitary matrices [18, 19] do not suffer from these problems. Lossless photonic components are per definition unitary, which will allow us to still find a suitable optimum for many circuit optimization problems through backpropagation. In the case of lossy structures, the losses are typically low enough to consider the photonic circuit quasi-unitary.

We show the relevance of this optimization scheme by optimizing a CROW in the frequency domain to act like a band-pass filter. We further show that the optimization can also be applied in the time domain. To illustrate this, we optimize some parameters and weights of a photonic reservoir computer [20]. As it will turn out, having a photonic circuit for which the action is differentiable will turn out to be very useful to obtain some connection weights between two cascaded reservoirs. Finally we show that this framework can also be used to optimize optically implemented unitary matrices by training a network of cascaded MZIs [5, 6, 19, 21, 22] to perform the permuted pixel-by-pixel MNIST[12] digit recognition task [19, 23], a well-known machine learning benchmark task for recurrent neural networks. Implementing and tuning large networks like this in photonics is not yet easy, however these photonic structures can serve as inspiration for different architectures of neural networks [19, 22] to be implemented in software, which can be designed by Photontorch.

2.8.1 Optimizing a CROW in the frequency domain

We use the CROW defined before (in 2.7) to create a band pass filter around $\lambda_0 = 1555\,\text{nm}$. We assume the CROW, as illustrated in Fig. 2.14, has 10 rings, each with a radius of $8\,\mu\text{m}$. After optimizing both the phases and the couplings, we find an optimum through gradient descent, as can be seen in Fig. 2.15.

Note that this bandpass filter is probably not the best bandpass filter one can achieve. Due to the recurrent nature of the circuit, there is a very real chance

[12] This task is named after the MNIST (Modified National Institute of Standards and Technology) database of over 50,000 handwritten digits. It's arguably the most well-known machine learning dataset to date and (in it's permuted pixel-by-pixel form) an excellent benchmark task for recurrent neural networks.

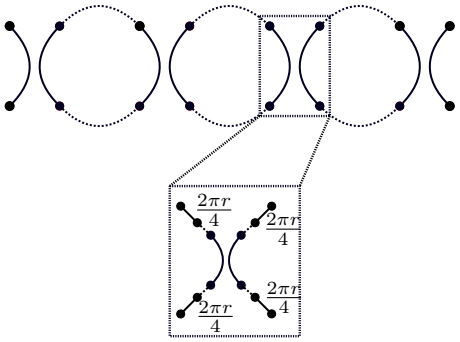


Figure 2.14: A CROW circuit in Photontorch with ring radius r is built up from several directional couplers for which each arm has a length of $2\pi r/2$. These directional couplers with non-zero arm length are basically Photontorch sub-circuits containing 4 waveguides (each with length $2\pi r/4$ connected to each of the ports of the directional coupler *without* length).

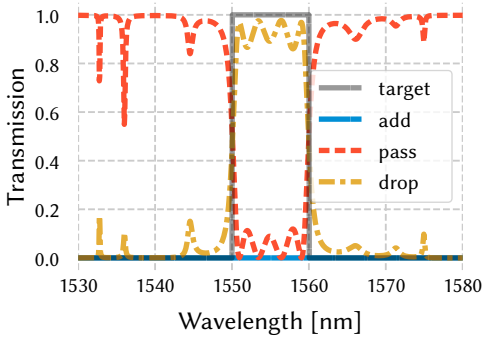


Figure 2.15: The parameters for a CROW-based bandpass filter can be obtained through backpropagation.

that the optimization through backpropagation got stuck in a *local* optimum. However, that is besides the point, as the real power is that a *decent* optimum was found without having to study the rings themselves. Indeed, we do not need to know the Free Spectral Range (FSR) of the rings, neither do we know the Full Width at Half Maximum (FWHM) to carefully craft a solution. We just define a CROW and let the optimizer find the necessary parameters in less than a minute[13]. If we were interested in a different wavelength range or a different bandwidth of the filter, we just need run the optimizer with a different target function.

[13] On a Nvidia GTX160 GPU.

2.8.2 Optimizing a ring network in the frequency domain

The CROW is a photonic circuit for which many analytical solutions exist and hence is probably not the most interesting example. However, the analytical approach becomes significantly more cumbersome when the network gets more complex. A step up in complexity from the 1D CROW structure is a for example the 2D ring network[14] shown in Fig. 2.16.

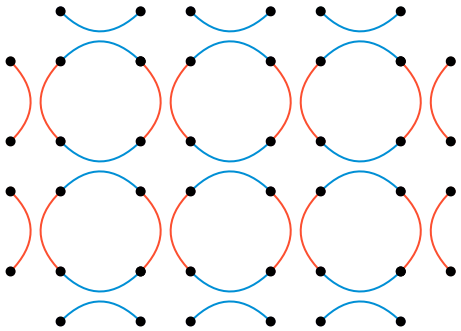


Figure 2.16: A ring molecule on a square lattice with rings of radius r can be built up from the same basic building blocks as a CROW organized in a staggered way. This time, the arm length of each of the directional couplers is $2\pi r/4$.

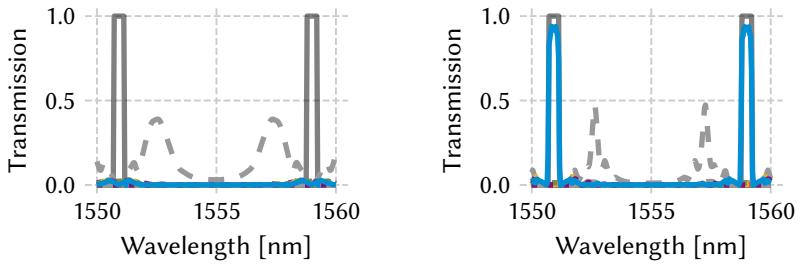
In the ring network, $50\,\mu$m rings with effective index $n_{\text{eff}} = 2.34$ and group index $n_{\text{g}} = 3.4$ are placed on a 6×6 grid. Light is injected anti-clockwise in the top right ring and the the clockwise output at the bottom left ring is optimized to correspond to a bandpass filter. Each directional coupler in the network can be modeled by a coupling and an additional phase factor at one of its 4 arms, which for the 6×6 network yields 168 independent parameters that can be optimized.

Just like for the 1D CROW, the 2D Ring network is quite easily optimized for the target at hand, as can be seen in Fig. 2.17. When looking at the final transmission at a log scale, as illustrated in Fig. 2.18, it is clear that what we have found is *probably* not a global optimum, as the extinction ratio (ER) is definitely not great right next to the pass band. However, here we make the same remark as before: in these kind of optimizations, the universality of the optimization is very powerful; something that probably will be more clear after an optimization in the *time* domain.

2.8.3 Optimizing a ring network in the time domain

Take for example a set of 1000 measured pulses of two distinct types, as illustrated in Fig. 2.19a. Both pulse types have a Gaussian-like profile, but the second one is modulated by a certain high frequency.

[14] Also sometimes called a ring *molecule*.



(a) Initial transmission **(b)** Final Transmission

Figure 2.17: After some optimization, the ring network can easily be optimized as a bandpass filter.

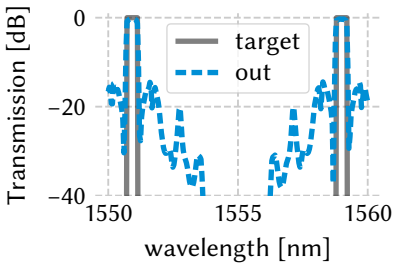
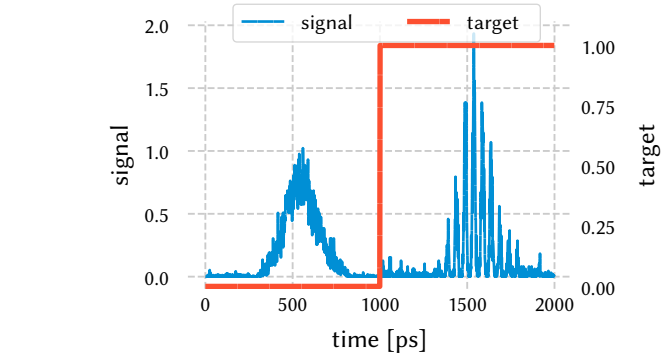
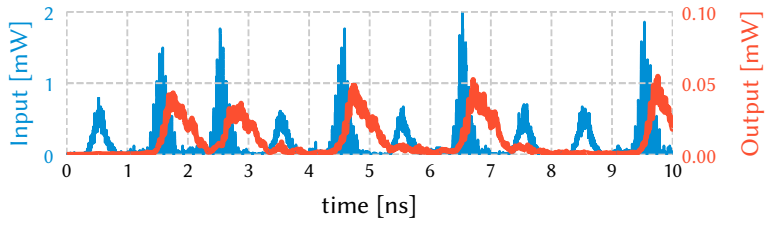


Figure 2.18: Transmission of the output port of the ring network on a logaritmic scale.

The goal is now to *recognize* these pulses by simulating and optimizing a ring network. A traditional way to do this could be with an optimization in the frequency domain which finds a bandpass filter that filters out the modulation frequency of the second pulse. However, this is a highly engineered-approach. Instead, we choose to brute-force optimize this *directly* in the time domain. Indeed, we will simulate a 3×3 ring network for which the ring delay corresponds to the pulse length ($T = 1\,\text{ns}$), which for an effective index of $n_{\text{eff}} = 2.4$ and a group index of $n_g = 4.3$ yields a quite large ring length of $1.4\,\text{mm}$. Simulating the circuit is done with a timestep of $1\,\text{ps}$ as this corresponds to the sampling rate of the input signal. [...] The network is optimized to follow the target function as illustrated in Fig. 2.19a. After some training, the detected signal follows the target function quite well, as can be seen in Fig. 2.19b



(a)



(b)

Figure 2.19: Pulse classification of two types of pulses. (a) The two pulses with their respective target function. (b) A stream of 10 pulses before entering and after leaving the (trained) ring network.

By looking at the resulting response in the frequency domain in Fig. 2.20, we see a whole different response than anything anyone would have strived towards if traditional optimization techniques would have been used. This proves

that Photontorch can be a valuable help during the design and optimization of photonic circuits, as it opens pathways to places in the parameter space that would not have been considered otherwise.

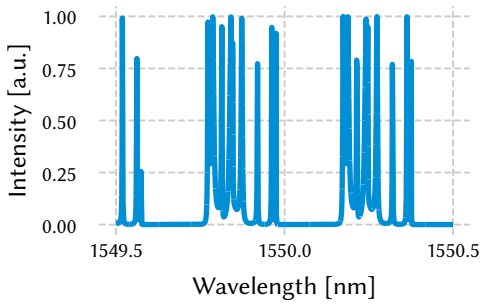


Figure 2.20: Frequency response of the 3×3 ring network optimized to recognize two different pulse types.

2.8.4 Optimizing photonic meshes

Optically implemented unitary matrices were first proposed by Reck et al. [5] in 1994 and were recently further improved upon by Clements et al. [6]. This led to a proof-of-concept of *photonic deep learning* [22], which is not only directly applicable in photonics, but also in the field of deep learning itself, where it was shown that these photonics-inspired *Unitary Recurrent Neural Networks* (URNN) can yield better results in recurrent architectures [19] than the more traditionally used Long Short-Term Memory (LSTM) cells [16], as they do not suffer from common problems in recurrent neural networks, such as vanishing gradients [15].

In 2.4.2, we mentioned that the MZI has an S-matrix which corresponds to the most general representation of the $SU(2)$ group. Let us choose this component as our building block for creating a general unitary matrix. To create such a matrix, we will cascade the MZIs together as illustrated in Fig. 2.21 to create a mesh-like structure, hence their name: photonic meshes. The number of consecutive MZI layers in such a photonic mesh is — in the context of photonic deep learning — sometimes called the *capacity* of the network. The capacity is a free parameter of the system and it turns out that one needs a *full-capacity* network to span the full unitary matrix space. Full capacity means that the number of MZI layers needs to be equal to the rank of the unitary matrix. However, networks with less capacity can also be used for plenty of photonic deep learning applications.

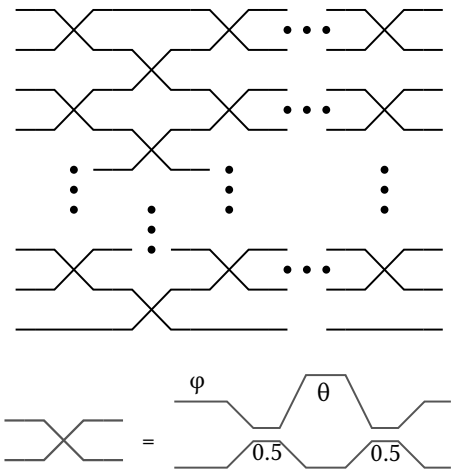


Figure 2.21: Any unitary matrix can be created by cascading several layers of MZIs together in what is called a *photonic mesh*. To span the full unitary matrix space, the number of MZI layers needs to be equal the rank of the matrix to represent.

Pixel-by-pixel MNIST task

By connecting the photonic mesh structure back onto itself, one can create the recurrent structure visualized in Fig. 2.22. This recurrent structure is in fact a photonic implementation of a Unitary Recurrent Neural Network (URNN), which we will use with great results in the permuted pixel-by-pixel MNIST task [18, 19], a common benchmark task for recurrent neural networks, where one tries to perform digit recognition on an image of a digit with 28×28 pixels that is sent pixel-by-pixel through the neural network in a fixed but randomized order as is illustrated in Fig. 2.23.

The architecture of the URNN, illustrated in Fig. 2.22, is defined as follows. We have a single input (which will take the image pixels one by one), which gets transformed to a 256D state by an array of optimizable weights. This 256D state gets then fed into the unitary matrix network of capacity 3, i.e. in three layers of each 128 MZIs (each MZI has two inputs). The outputs of this URNN get split: one part gets fed into the output layer and one part gets sent back to the input of the unitary matrix. The output layer in itself is again an array of 256×10 weights, which makes a linear combination for each of the possible digit responses. The output number with the largest resulting amplitude is the answer of our network.

The input and the output layer can in principle be represented by a photonically implemented unitary matrix as well, but we chose not to do this in this proof of concept application as to not make the model overly complex. The total

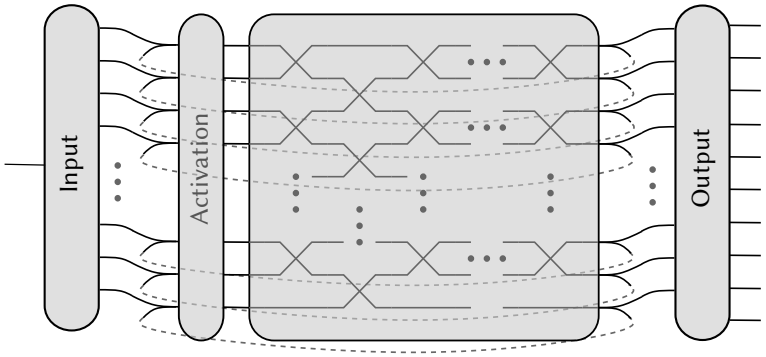


Figure 2.22: By looping the unitary matrix onto itself, one creates a URNN. The network represented here contains an input layer, which transforms the 1D time dependent input data to a 256D state. This state then gets sent through the unitary matrix, which is connected onto itself. The output weights transform the recurrent layer back into a 10D state: one output for each digit to recognize. To boost the power of the recurrent neural network, an activation or non-linear element was added into the recurrent loop.

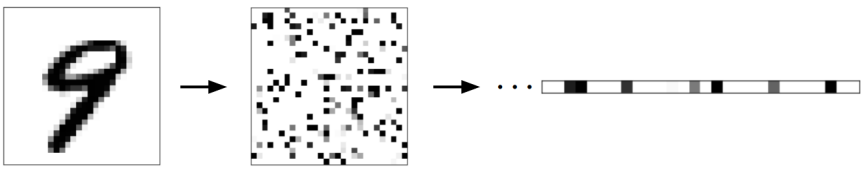


Figure 2.23: An image of a digit consisting of 28×28 pixels is first randomized by a fixed permutation before it is flattened and sent through the network pixel by pixel.

number of parameters represented by photonic components, i.e. the cascade of MZIs, is thus $2 \times 128 \times 3 = 768$, as each MZI contains two optimizable parameters: the input phase difference ϕ and the phase difference between its arms θ. Optimizing this many parameters with a conventional circuit simulator would be a nightmare, however it is quite easily done with Photontorch.

To boost the performance of the network defined above, a non-linear layer has to be added to the recurrent loop. This non-linear element was implemented in simulation by the modrelu [19] function. However, Photontorch allows in principle to easily swap out this non-linearity for a more physically achievable non-linearity, for example implemented by a Semiconductor Optical Amplifier (SOA).

The final accuracy on the MNIST digits for the permuted pixel-by-pixel MNIST task is 92%, as can be seen in Fig. 2.24. This is on par with previously documented results for unitary matrices [18, 19]. However, in this case, the core of the network was defined solely using Photontorch components, which makes it a very modular approach. This allows for example to change the network at certain locations by changing some of the MZIs to more complex components. Moreover, Photontorch allows to easily experiment with completely different photonics-inspired neural network designs that are less easily implemented with conventional modelling tools.

It is important to note that this architecture was not intended to be a realistically realizable photonic architecture, but rather a showcase for the optimization strength of Photontorch. The network is not realistic as, for one, the ModReLU is a non-physical activation function. Moreover, the timestep was way too large for realistic simulations: each timestep corresponds to sending a single (full) pixel sent through the network. Additionally, the 1×256 splitter and the 256×10 combiner were modeled by normal weight matrices. A normal weight matrix can be decomposed with the singular value decomposition into two unitary matrices (and a diagonal matrix of singular values). Hence the splitter and combiner each would need to be composed of *two* MZI meshes (and an array of amplifiers) to be realizable in photonics. This would make the design considerably more complicated. Finally, no delays *inside* the mesh was modeled. The only delays modeled arise from the feedback loop, which corresponds to a single timestep.

Improved optics by component redundancy

One way these photonic meshes can be used is to introduce component redundancy in the network yielding noise-resilient photonic circuits. Take for example a directional coupler. The actual coupling of a directional coupler might vary through process variations, temperature, the wavelength of the light and so on. Designing circuits that are resilient to these changes and variations is a real challenge. As an example on how Photontorch can be used in this regard, we will

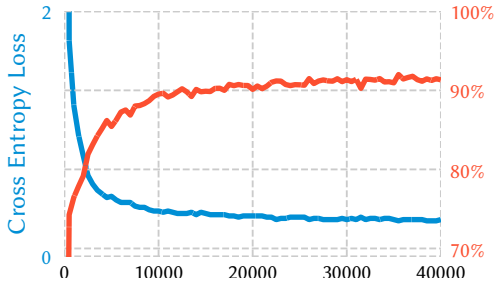


Figure 2.24: Training for the pixel-by-pixel MNIST task with a capacity-3 unitary neural network.

optimize a 4×7 photonic mesh of directional couplers to act as a *single* 50/50 directional coupler.

Such a mesh consisting of 11 directional couplers, will then be used as illustrated in Fig. 2.25: light entering along the first directional coupler is split through the circuit to finally arrive at the final directional couplers labeled 10 and 11. Since we are trying to emulate a 2×2 component, we have two output ports too many: we will try to keep the output at the odd numbered ports at 0, while the output at the even numbered ports will be targeted.

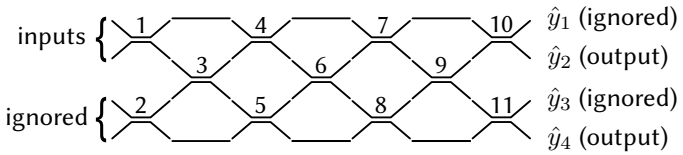


Figure 2.25: A 4×7 mesh of imperfect directional couplers acting as a single tolerant directional coupler with 50/50 coupling.

We furthermore assume that the directional couplers have some fabrication errors which result in a coupling which is normally distributed around the desired coupling with a standard deviation of $\sigma_{\kappa^2} = 0.05$. Moreover, we assume that neither perfect coupling nor zero coupling can be achieved, i.e. $\kappa_{\text{min}} = 0.1; \kappa_{\text{max}} = 0.9$.

We then try to find the parameters which yield the mesh resilient to this noise on the coupling. To do this, the mesh is first optimized *without* any noise, which gives the baseline parameters of the circuit listed in Table 2.1. Then, a large number of meshes is generated with randomly permuted coupling according to the mentioned Gaussian distribution around the baseline. This yields a batch of B meshes with slightly different coupling arising from the same baseline coupling. The baseline parameters of the randomized batch are subsequently optimized on

κ	1	2	3	4	5	6	7	8	9	10	11
baseline	0.63	0.50	0.25	0.68	0.32	0.23	0.67	0.29	0.70	0.76	0.47
adjusted	0.12	0.50	0.76	0.90	0.25	0.10	0.90	0.69	0.77	0.90	0.24

Table 2.1: Coupling coefficients for a noise-resilient mesh

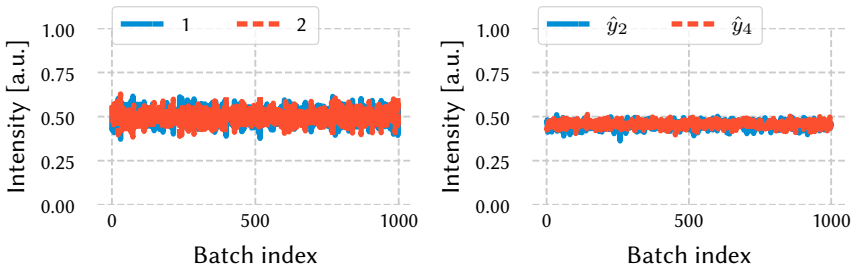
the output of the randomized batch with the following loss function:

$$\begin{aligned} L =& \alpha \left(\frac{1}{B} \sum_{b=1}^{B} \sum_{i=1}^{4} \left(\hat{y}_{bi} - y_i \right)^2 \right) \\ &+ \beta \left(\frac{1}{B} \sum_{b=1}^{B} \left(\hat{y}_{b2} - \frac{1}{B} \sum_{n=1}^{B} \hat{y}_{n2} \right)^2 + \left(\hat{y}_{b4} - \frac{1}{B} \sum_{n=1}^{B} \hat{y}_{n4} \right)^2 \right). \end{aligned} \tag{2.66}$$

The first term is nothing more than the mean squared error between the targets y_i at each of the four output ports $y = (0, 0.5, 0, 0.5)$ and the output $\hat{y}_b$ for each of the networks in the batch, scaled by a hyper-parameter α. The second term describes the variance in the output at the ports we are interested in (2 and 4), scaled by a hyper parameter β. Both these terms are important, as the first term ensures the intended transmission at 50%, while the second term minimizes the variation. The hope is, that by ignoring the variance in the parameters for the odd-numbered output ports, some of the variation in the even-numbered output ports gets transferred to them, reducing the variance in the ports we are interested in. The hyperparameters — which are completely free to choose — were in this case chosen to be $\alpha = 1$ and $\beta = 6$.

By definition, we will not be able to achieve better than 0.05 standard deviation on the coupling for a single directional coupler, as illustrated in Fig. 2.26a but by connecting them together in the mesh structure, we *are* able to achieve less variation, as illustrated in Fig. 2.26b. Indeed, by defining the inputs and outputs of the $50/50$ splitter as illustrated in Fig. 2.25, the standard deviation can be reduced to 0.016.

This improved optics by using more components comes at a price however, which in this case is twofold. First of all, the total output power at the ports at interest will be reduced, as some of the light will inevitably end up at the output ports we are not interested in. This can also be seen somewhat when comparing Fig. 2.26a and Fig. 2.26b. This difference will obviously be greatly amplified when losses are taken into account, which — in a realistic setting — will probably kill this concept. Moreover, the overhead and increased chip real estate of having to fabricate 11 directional couplers instead of a single one is not insignificant. However, it serves as an inspiration of how better optics can possibly be achieved when redundancy is built into the circuit.



(a) **(b)**

Figure 2.26: Transmission for (a) a batch directional couplers with coupling normally distributed around 50% with a standard deviation of 5%; (b) a batch of mesh circuits containing directional couplers with the same deviations but optimized to reduce variation in the output.

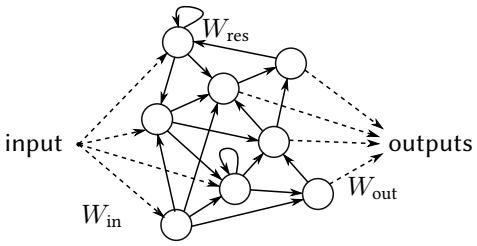


Figure 2.27: A single-input reservoir computer. A single input is distributed over the nodes of a reservoir by a fixed set of input weights W_{in}. The reservoir has a complex recurrent interconnection topology characterized by its intermediate weights W_{int}. The reservoir states are read out by a trainable set of readout weights W_{out}.

2.8.5 Improving the performance of a single passive reservoir

As was explained in the previous chapter, reservoir computing is an almost two-decade-old machine-learning concept [24, 25]. It is defined by distributing an input signal over a series of nodes which are *recurrently* connected, as shown in Fig. 2.30. The connections between the recurrent nodes are not optimized and form the so-called *reservoir*. In fact, only the output connections that combine the states in the recurrent nodes into a useful output signal are optimized for the task at hand. The reservoir is called *passive* if no active elements or non-linearities are present inside it. Such passive reservoirs rely solely on the non-linear operation at the photodetector and are easily implemented in photonic circuits with splitters and combiners [20].

In most on-chip reservoirs, the reservoir states are first detected before they are linearly combined into an output signal (the so-called electrical readout). Although this first-detect-then-weight approach produces good results, it is not very feasible for large reservoirs, as one would need as many detectors as there are reservoir nodes. On top of that, using multiple detectors and analog-to-digital converters goes against the idea of having an energy-efficient solution to many problems. For an all-optical implementation, it is beneficial to move the many detectors at the nodes of the reservoir to one single detector at the output, after an optically implemented weighting procedure, implemented e.g. by amplitude and phase modulators (a so-called optical readout). These complex-valued readout weights should be trained to minimize the Mean Squared Error (MSE) between the detector output and the target signal.

The traditional approach to train reservoirs uses ridge regression [26] to optimize a real-valued sum with real-valued weights [27]. Just like we mentioned in 1.3.5, while it is possible to use a complex extension of linear regression to optimize a complex-valued sum with complex-valued weights, this is not entirely what we need in order to train the optical readout.

Indeed, we only care about the amplitude of the signal after the detector[15], Whereas complex-valued ridge regression would only be able to aim for a given complex summed signal before the detector. However, there are many different complex-valued signals (each with a different phase) before the detector that give rise to the same intensity after the detector. In order to be able to use ridge regression, we would need to arbitrarily fix the phase of the signal before the detector, effectively limiting the space of complex optical weights.

This is where Photontorch can be of invaluable help, as it enables to perform backpropagation through the detector without having to make any assumption on the phase before the detector. Moreover, since Photontorch enables backpropagation through the *whole* circuits, we should also be able to fine tune the reservoir to better perform the task at hand.

This is where Photontorch can be of invaluable help, as it enables to perform backpropagation through the detector without having to make any assumptions on the phase before the detector. Moreover, since Photontorch enables backpropagation through the *whole* circuit, we should also be able to fine-tune the reservoir to better perform the task at hand.

To test this premise, we fine-tune a typical swirl-reservoir, much like the one in Fig. 1.17, but this time with 36 nodes (6×6) using Photontorch on the XOR task, where the output of the reservoir should predict the XOR of two subsequent bits in the input bit stream sent through the reservoir at $50\,\text{Gbps}$. All interconnec-

[15]The detector is, throughout the simulations, modeled with a load resistance $R_L = 1\,\text{k}\Omega$, responsivity $\eta = 0.5A/W$ and frequency cut-off $f_c = 50\,\text{Gbps}$, implemented by an order-4 Butterworth filter.

tions between the nodes of the reservoir correspond to a single bit period of $20\,\mathrm{ps}$ ($1.4\,\mathrm{mm}$). We will quantify how well the MSE between the target stream and the response of the reservoir-readout-detector combination on an input stream of 10^5 bits *improves* by optimizing the phases of *six* randomly chosen interconnections between reservoir nodes compared to having no internal optimization of the reservoir (but *with* optimization of the readout).

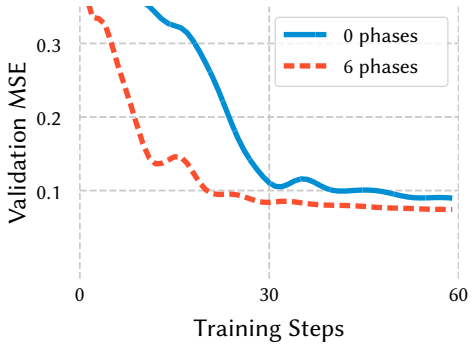


Figure 2.28: Learning curves of the reservoir optimization through backpropagation. A reservoir where only the readout is optimized is compared to a reservoir where *both* the readout *and* 6 internal phases were optimized.

As can be seen in Fig. 2.28, the reservoir performance can be somewhat improved by allowing some fine-tuning of the reservoir. Moreover, we clearly see that the acceptable performance with for example $\mathrm{MSE} < 0.1$ is achieved much faster when fine-tuning is allowed. A typical signal from an optimized fine-tuned reservoir is illustrated in Fig. 2.29

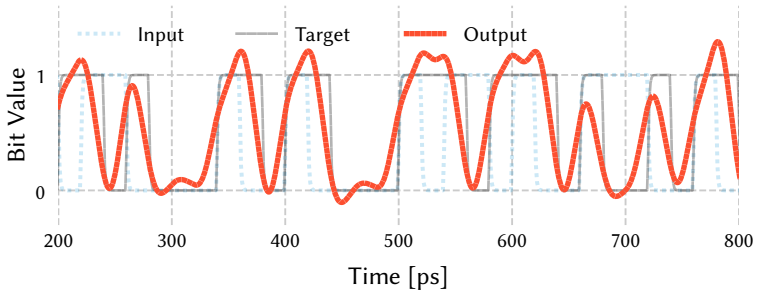


Figure 2.29: Optimal performance on the XOR task where the reservoir was fine-tuned by allowing the optimization of 6 internal phases.

2.8.6 Improving the performance by cascading two passive reservoirs

Because we can now backpropagate through the reservoir, we're now able to create even more interesting structures. In stead of optimizing a single reservoir, one can now for example choose to optimize two *cascaded* reservoirs, as illustrated in Fig. 2.30. This was previously completely impossible as the action of the last reservoir could not be inverted and hence a set of intermediate *connection weights* W_{int} could not be found.

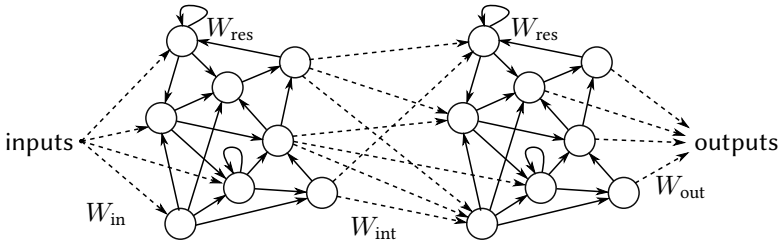


Figure 2.30: Two reservoirs are cascaded by a trainable set of intermediate weights W_{int}.

We combine two reservoirs of half the size of the reservoir introduced in the previous section to still arrive at a total of 36 nodes (2 times 18 nodes) but this time with a set of intermediate weights in between which interconnect each node of the first reservoir to each node of the second reservoir.

Looking at Fig. 2.31a, we see that it does not perform much better than the original 36 node reservoir (without internal phases optimized). This is probably because 2×18 reservoirs inherently possess less richness than a 36 node reservoir. However, we expect it to have more memory, as a signal has to pass through two recurrent reservoir blocks before being detected.

Two justify this claim, we perform the more difficult task where the XOR of two bits with one bit in between is taken as a target. For this task, more memory is needed, as the reservoir needs to retain 3 bits of information in stead of 2. The learning curve for this task is visualized in Fig. 2.31b. Here we clearly see better performance for the cascaded reservoir (but only if we allow the set of intermediate weights to be optimized).

2.9 Conclusion

Grown from the necessity to simulate (and optimize) photonic neuromorphic architectures going beyond traditional reservoir computing, the presented Photontorch framework is the first photonic simulator to our knowledge that enables

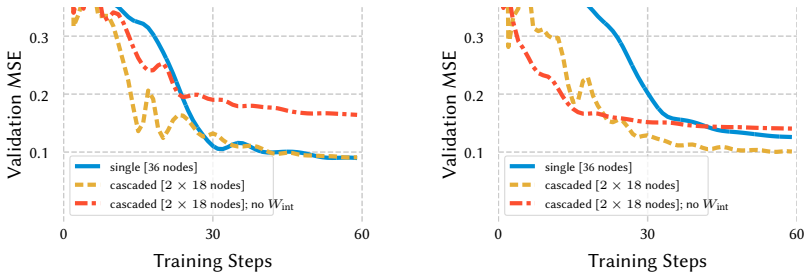


Figure 2.31: Learning curves obtained by optimizing the cascaded reservoir optimization through backpropagation. A reservoir where only the readout is optimized is compared to a cascaded reservoir where *both* the readout *and* the intermediate weights are optimized. (a) Performance on the XOR of two adjacent bits. (b) Performance on the XOR of two bits with one bit in between.

true optimization of large photonic circuits by backpropagation through its *physical parameters*.

The simulator thereby adds a completely new approach to the photonic simulation landscape in two major ways: it facilitates the simulation of large photonic circuits in a truly parallel way on a GPU and — perhaps more importantly — it enables a completely new way of optimizing photonic circuits through backpropagation. Moreover, by relying on a well-established machine learning library like PyTorch, potentially new (optimization) techniques discovered for deep-learning can now be instantly applied to the optimization of photonic circuits.

We demonstrated by providing concrete examples that this deep-learning based photonic circuit simulator can be of great value to optimize photonic circuits. Indeed, Photontorch shows a lot of promise for such photonic circuit simulation and optimization. It is an ideal choice when simulating low-dispersive, passive circuits for multiple wavelengths in the time domain. Additionally, the inherent parallel nature also allows to simulate the batched response to different independent input waveforms simultaneously at almost no overhead.

The main feature of Photontorch is its close relation to PyTorch autograd tensors, allowing it to leverage backpropagation through each photonic component to optimize the parameters of large photonic circuits. We expect this to be incredibly useful for prototyping photonic circuits, as well as for optimizing the parameters in arbitrary photonic circuits containing both passive and active elements, such as - but definitely not excluded to - the large mesh network used for the state-of-the art performance on the pixel-by-pixel MNIST task discussed in this chapter.

This feature might act as a double-edged sword, however, as having to de-

scribe each operation in terms of differentiable PyTorch tensors inherently limits what kind of computations can be done efficiently, while in addition, GPUs generally are more efficient for linear operations. This means that - although Photontorch is certainly capable of doing so - circuits with many active components will not be simulated as efficiently as the highly optimized CPU-code found in some other simulators.

Apart from the simple examples given in this chapter to show how the presented simulator can be used, Photontorch was also used to improve and extend the traditional on-chip passive photonic reservoir architecture by enabling optimization *inside* the recurrent circuit and by *connecting* two non-optimized recurrent circuits through an optimizable intermediate connection. Both designs performed better than the original reservoir after the optimization: an optimization that was only possible by backpropagation through the physical parameters of the circuit.

That is probably the final strength of the Photontorch framework: although it grew from the necessity to simulate photonic *neuromorphic* structures in the time domain, today it is certainly capable to simulate *any* photonic circuit whatsoever, both in the frequency domain *and* in the time domain.

References

[1] Floris Laporte, Joni Dambre, and Peter Bienstman. *Highly parallel simulation and optimization of photonic circuits in time and frequency domain based on the deep-learning framework PyTorch.* Scientific reports, 9(1):5918, 2019.

[2] Martin Fiers, Thomas Van Vaerenbergh, Ken Caluwaerts, Dries Vande Ginste, Benjamin Schrauwen, Joni Dambre, and Peter Bienstman. *Time-domain and frequency-domain modeling of nonlinear optical components at the circuit-level using a node-based approach.* JOSA B, 29(5):896–900, 2012.

[3] Max Born and Emil Wolf. *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light.* Elsevier, 2013.

[4] Bernard Yurke, Samuel L McCall, and John R Klauder. *SU (2) and SU (1, 1) interferometers.* Physical Review A, 33(6):4033, 1986.

[5] Michael Reck, Anton Zeilinger, Herbert J Bernstein, and Philip Bertani. *Experimental realization of any discrete unitary operator.* Physical review letters, 73(1):58, 1994.

[6] William R Clements, Peter C Humphreys, Benjamin J Metcalf, W Steven Kolthammer, and Ian A Walmsley. *Optimal design for universal multiport interferometers.* Optica, 3(12):1460–1465, 2016.

[7] Bjorn Gustavsen and Adam Semlyen. *Rational approximation of frequency domain responses by vector fitting.* IEEE Transactions on power delivery, 14(3):1052–1061, 1999.

[8] Yinghao Ye, Domenico Spina, Yufei Xing, Wim Bogaerts, and Tom Dhaene. *Numerical modeling of a linear photonic system for accurate and efficient time-domain simulations.* Photonics Research, 6(6):560–573, 2018.

[9] Govind P Agrawal and N Anders Olsson. *Self-phase modulation and spectral broadening of optical pulses in semiconductor laser amplifiers.* IEEE Journal of quantum electronics, 25(11):2297–2306, 1989.

[10] Daniele Melati, Francesco Morichetti, Antonio Canciamilla, Davide Roncelli, Francisco M Soares, Arjen Bakker, and Andrea Melloni. *Validation of the building-block-based approach for the design of photonic integrated circuits.* Journal of Lightwave Technology, 30(23):3610–3616, 2012.

[11] Lumerical. *A commercial-grade circuit simulator for the design, simulation and analysis of photonic integrated circuits.* https://www.lumerical.com/tcad-products/interconnect/. Accessed: 2019-11-01.

[12] VPI. *Photonics design automation*. http://www.vpiphotonics.com. Accessed: 2018-12-10.

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. *Automatic differentiation in PyTorch*. Neural Information Processing Systems, 2017.

[14] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[15] Sepp Hochreiter. *The vanishing gradient problem during learning recurrent neural nets and problem solutions*. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02):107–116, 1998.

[16] Sepp Hochreiter and Jürgen Schmidhuber. *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning phrase representations using RNN encoder-decoder for statistical machine translation*. arXiv preprint arXiv:1406.1078, 2014.

[18] Martin Arjovsky, Amar Shah, and Yoshua Bengio. *Unitary evolution recurrent neural networks*. In International Conference on Machine Learning, pages 1120–1128, 2016.

[19] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. *Tunable efficient unitary neural networks (EUNN) and their application to RNNs*. arXiv preprint arXiv:1612.05231, 2016.

[20] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman. *Experimental demonstration of reservoir computing on a silicon photonics chip*. Nature communications, 5, 2014.

[21] David A. B. Miller. *Perfect optics with imperfect components*. Optica, 2(8):747–750, Aug 2015.

[22] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. *Deep learning with coherent nanophotonic circuits*. Nature Photonics, 11(7):441, 2017.

[23] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. *A Simple Way to Initialize Recurrent Networks of Rectified Linear Units*. CoRR, 2015.

[24] H. Jaeger. *The 'echo state' approach to analyzing and training recurrent neural networks*. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148:34, 2001.

[25] W. Maass, T. Natschläger, and H. Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations*. Neural computation, 14(11):2531–2560, 2002.

[26] Arthur E Hoerl and Robert W Kennard. *Ridge regression: Biased estimation for nonorthogonal problems*. Technometrics, 12(1):55–67, 1970.

[27] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. *An overview of reservoir computing: theory, applications and implementations*. In Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007, pages 471–482, 2007.

3

On-chip Reservoir Computing with Photonic Cavities

So far, when talking about neuromorphic computing, we have only considered conventional photonic integrated circuits built from conventional components such as waveguides and directional couplers. In this chapter we will explore a more unconventional route: on-chip photonic cavities as reservoir mixing units [1]. We will explore these cavities both through thorough numerical simulations and in measurements.

3.1 Introduction

Although photonic on-chip reservoirs like the swirl reservoir discussed in 1.8.1 have some major advantages over other photonic reservoir architectures, such as ultra-high speed operation, they still suffer from some serious drawbacks. Those drawbacks include a very limited interconnection topology, a low density of nodes per chip surface area and high losses due to the many 3 dB-combiners. To address these issues, we propose a new passive design on a silicon photonics chip that seeks to improve on the reservoir-on-chip technology. The proposed reservoir consists of a cavity with a special shape tuned to foster interesting mixing dynamics [2–5]. Two types of cavities will be explored: photonic crystal cavities, which exhibit low losses and high Q-factors and — for ease of manufacturing — cavities based on dielectric index contrast at a single interface.

One of the advantages of choosing a cavity as photonic reservoir is the extremely small on-chip footprint, with dimensions smaller than $0.1\,\mathrm{mm}^2$, which is at least an order of magnitude smaller than the previous reservoir-on-chip design discussed in 1.8.1. As we will see below, this photonic crystal design promises very low loss combined with excellent performance on the XOR Task and header recognition task while still accepting bitrates in a wide region of operation.

3.2 Reservoir designs

3.2.1 Photonic crystal cavity reservoir

The first incarnation of the design consists of an on-chip photonic crystal cavity in the shape of a quarter-stadium resonator as illustrated in Fig. 3.1 and Fig. 3.2. This specific shape is known to foster interesting mixing of the fields in an almost chaotic manner [2–4]. The use of a photonic crystal cavity has the potential for low loss and with it a long reservoir memory, but at the cost of a more difficult fabrication process. This is why this cavity type was only studied in simulation. It serves however as a good test case to later on compare other designs to.

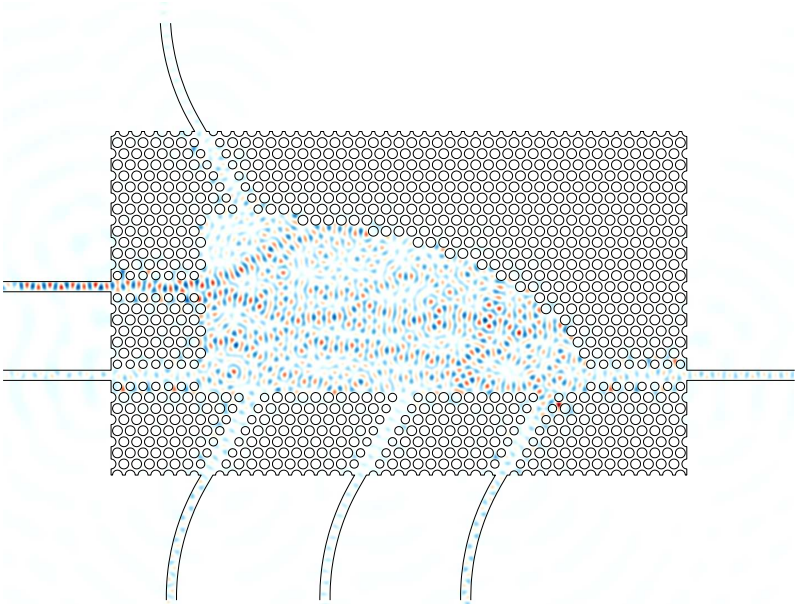


Figure 3.1: Snapshot of the field profile in $10\,\mu\mathrm{m} \times 5\,\mu\mathrm{m}$ photonic crystal cavity. The mixing of the signal can clearly be witnessed by inspecting the field profiles.

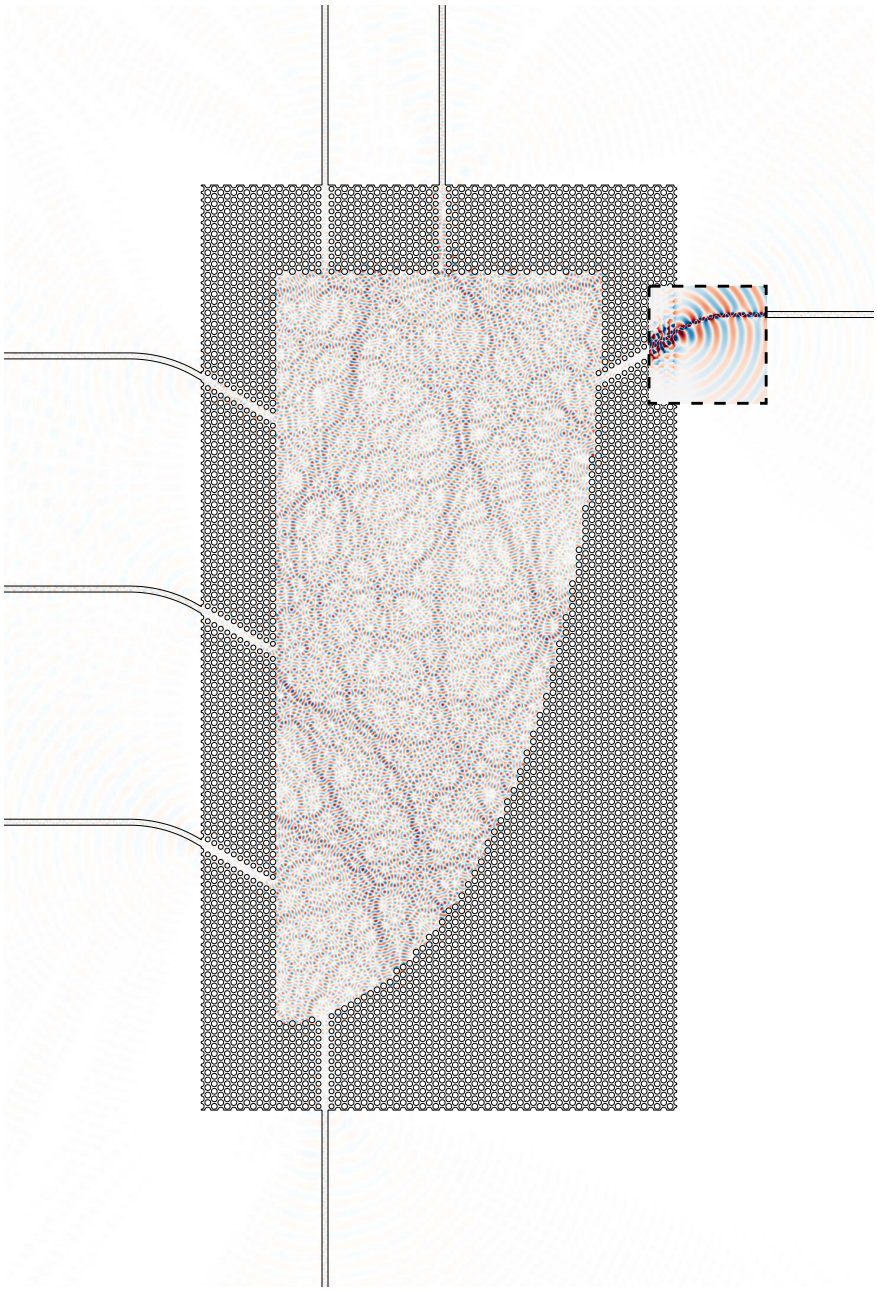
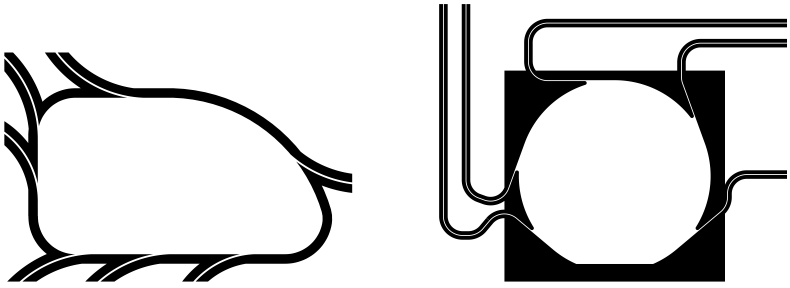


Figure 3.2: Snapshot of the field profile in $60\,\mu\mathrm{m} \times 30\,\mu\mathrm{m}$ photonic crystal cavity. The mixing of the signal can clearly be witnessed by inspecting the field profiles. At one of the arms, the color map range was decreased by a factor ten to better show the radiation losses due to mode mismatch between the W1-defects and the waveguide.

The photonic crystal cavity which was simulated for different sizes ranging from $20\,\mu\text{m} \times 10\,\mu\text{m}$ to $100\,\mu\text{m} \times 50\,\mu\text{m}$ was designed for the $220\,\text{nm}$ silicon photonics platform, consisting of holes etched in a $220\,\text{nm}$ silicon slab with radius $r = 0.27a$, with $a = 420\,\text{nm}$ the pitch of the photonic crystal. The light is sent through one of the seven standard standard $450\,\text{nm}$ wide waveguides, which are connected to W1-defects in the wall of the photonic crystal cavity. The light inside the cavity subsequently leaks out of the cavity via all of the defects. The six other defects are used for readout. It is clear that this cavity is a great candidate for a reservoir, because alongside its *mixing property*, it also possesses a *fading memory*: the signal is bound to remain in this cavity for a certain amount of time directly proportional to the Q-factor and the dimensions of this cavity. Moreover, light is trapped inside the cavity and can only leak out using the wall defects, where they contribute to the useful output signal. This results in potential low losses in the system.

3.2.2 Cavities based on index contrast

Other cavity types that will be explored are the cavities based on index contrast as illustrated in Fig. 3.3. One of these cavities, Fig. 3.3a, has a similar quarter stadium shape as the photonic crystal cavity, while the other, Fig. 3.3b is based on a disk with a chamfer [5]. As these cavities only depend on the etching of the trenches around them, they will be a lot easier to manufacture. This ease of manufacturing might come at a price though. Since we no longer have the omnidirectional reflection of the photonic crystal, losses will be higher and thus the cavities might exhibit less memory. As these cavities rely less on a specific etch-



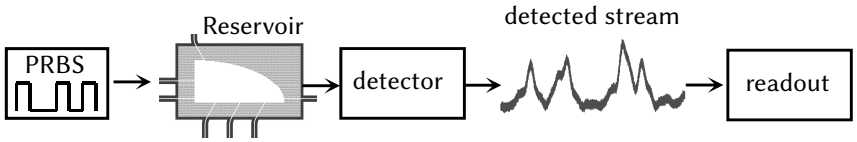
(a) Quarter stadium shape **(b)** Chamfer design

Figure 3.3: Two cavity shapes that induce interesting mixing dynamics. Both shapes are based on index contrast: a simple trench was etched around the cavity to keep the light in.

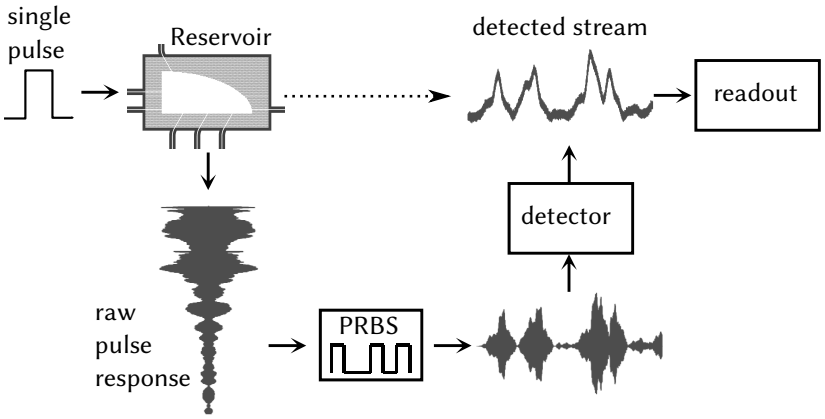
ing process, multiple silicon photonic platforms could be more easily targeted. We made cavities for the 400nm and the 220nm silicon photonic platform in two

different shapes: the quarter stadium shape [2–4] just like for the photonic crystal cavities and the circle with a chamfer [5].

3.3 Simulations



(a) Idealized measurement setup



(b) Simulation approximation

Figure 3.4: (a) Measurement setup and (b) the approximation in simulation: the response of a single bit is recorded and is coherently added together according to a PRBS.

To obtain the response of the reservoirs to an arbitrary bit stream, the most time-consuming step of the calculation is simulating the propagation of the light through the cavity. Indeed, training the reservoir readout relies on retrieving the response of a full bit stream as illustrated in Fig. 3.4a. However, as most of the simulations rely on a 3D Finite Difference Time Domain[1] (FDTD) simulation, which is a slow numerical method for solving electromagnetic problems, sending the whole bit stream through the system is simply unfeasible.

[1] More on the FDTD method in Chapter 4

3.3.1 Pulse composition

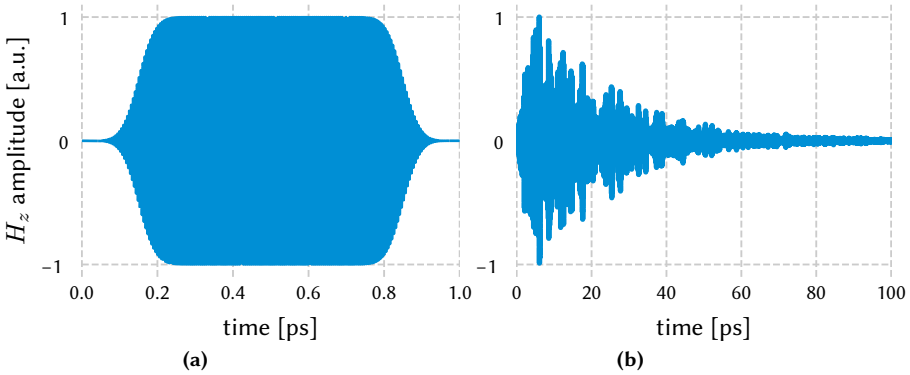


Figure 3.5: (a) A normalized 1 ps input pulse with smoothed rising and falling edges. (b) Normalized response to the input pulse at one of the output arms.

We therefore do not simulate the propagation of the complete bit stream, as this would result in enormous calculation times[2]. Instead, we use an alternative approach illustrated in Fig. 3.4b. Suppose we have a pseudo-random bit stream (PRBS) consisting of the bits $b_1, b_2, ..., b_N \in \{0, 1\}$, for which the nonzero bits are specified by a smoothed pulse $\mathbf{u}(t)$ as shown in Fig. 3.5a:

$$\mathbf{u}(t) = \begin{pmatrix} \mathbf{E}_{\text{in}}(t) \\ \mathbf{H}_{\text{in}}(t) \end{pmatrix} \qquad \text{with } \mathbf{u}(t) = \mathbf{0} \text{ if } t < 0 \text{ or } t > T, \tag{3.1}$$

with T the bit period of the signal. Then the value of the bit stream at time t is given by[3]

$$\mathbf{x}(t) = \begin{pmatrix} \mathbf{E}_{\text{in}}(t) \\ \mathbf{H}_{\text{in}}(t) \end{pmatrix} = \sum_{n=1}^{N} b_n \mathbf{u}(t - nT) = b_k \mathbf{u}(t - kT) \quad \text{with } k = \left\lceil \frac{t}{T} \right\rceil \tag{3.2}$$

Each of the exit waveguides i will have an exponentially decaying response $\mathbf{U}_i(t)$ to the single bit pulse $\mathbf{u}(t)$ as for example the response shown in Fig. 3.5b. This means that the response $\mathbf{X}_i(t)$ to the total bit stream at waveguide i can be described by

$$\mathbf{X}_i(t) = \begin{pmatrix} \mathbf{E}^i_{\text{out}}(t) \\ \mathbf{H}^i_{\text{out}}(t) \end{pmatrix} = \sum_{n=1}^{N} b_n \mathbf{U}_i(t - nT). \tag{3.3}$$

[2] simulating a single bit takes between 12-24 hours

[3] Note that this results in return-to-zero (RZ) encoding for the smoothed pulses defined here.

Note that by only looking at $\mathbf{X}_i(t)$, the individual responses cannot be decoupled anymore since $\mathbf{U}_i(t) \neq 0$ for $t > T$. The state of the reservoir depends linearly on the previous input values, just like one would require from the passive reservoir described in 1.7.2.

In practice, the responses $u_i(t)$ at each of the waveguides to a single pulse are recorded from an FDTD simulation, performed by the *Lumerical FDTD* software. Then the response of a complete bit stream (typically a PRBS signal of 10^5 bits) is calculated by coherently adding together the individual bit responses for each channel in the way described above. Note that this described procedure is just a "bit-level" version of the impulse response method, where the response of an arbitrary system is found by convolving the function with the response of an ultrashort impulse. Here, we chose to work with the "bit-level" response instead of the true impulse response because of numerical rounding errors.

3.3.2 Convergence analysis

A simple convergence analysis can be performed where two subsequent bits are either simulated directly either via FDTD or by the coherent composition technique defined above. As can be seen in Fig. 3.6, the resulting pulses obtained by either technique are quite similar, but not identical. The residual difference between the two is probably due to rounding errors and the influence of the tail of the impulse response which is truncated at some point in the coherent composition technique. The normalized root mean squared error (NRMSE) between both is 0.16. However, this should not be a big problem, as similar errors are obtained between two identical streams detected with the noisy detector, which will be introduced next.

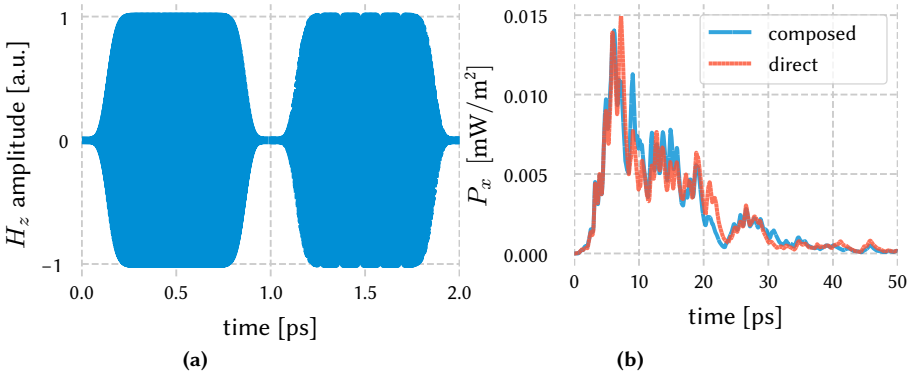


Figure 3.6: (a) Two subsequent 1ps input pulses. (b) Responses (Poynting vector projected in the direction of propagation) obtained by composition and direct simulation.

3.3.3 Photodetector

Finally, the photodetector performs the non-linear operation described in (1.63) by detecting the light. This detector is simulated by a detector model with similar parameters as the detector in our labs, which has a load resistance $R_L = 1\,\mathrm{k\Omega}$, a bandwidth $f_c = 25\,\mathrm{GHz}$ and a responsivity $\eta = 0.5\,\mathrm{A/W}$. The detector noise introduced by the amplification of the photo-generated current is described by white thermal noise I_{tn}, modeled as a Nyquist process and shot noise I_{sn}, modeled as a Poisson process:

$$I_n = \sqrt{I_{\mathrm{tn}}^2 + I_{\mathrm{sn}}^2} = \sqrt{\left(\frac{4kTf_c}{R_L}\right)^2 + 2qIf_c}. \tag{3.4}$$

3.3.4 Readout

In the readout, the output streams leaving the arms of the reservoir, of which two are shown in Fig. 3.7a are sampled a fixed number of times per bit period. After this, a linear combination of the sampled values is made according to weights that are specifically trained for the intended application. Increasing the number of sampling points generally improves the performance of the reservoir. However, since we are already working at very high bitrates, we usually choose to sample only once per bit.

To obtain the weight matrix W_{out} for the linear combination acting on the output states of the reservoir, two different kinds of training algorithms are used: *ridge regression* (linear regression with a regularization parameter) for the binary classification tasks such as the XOR task, and *linear discriminant analysis* (LDA) [6] for multi-class classification tasks such as header recognition.

3.3.5 Benchmark tasks

Finally, three different tasks will be considered for these cavities. The first, easiest task will be the *copy* task, where the reservoir attempts to reproduce the input signal with a certain *latency*. The latency is usually expressed as a multiple of the bit period, i.e. the number (or fraction) of bit periods one has to wait after the last relevant bit has completely entered the cavity before you can reproduce the target signal. The XOR task, also described in 2.8.5, where the reservoir tries to perform the XOR of two subsequent bits and with a certain latency[4]. Finally the performance on a header recognition task will be assessed. All three of these tasks will be assessed at different bitrates to find the operating range of the cavity.

[4] In fact some latency is always necessary as the signal needs to traverse the cavity before it will be detected.

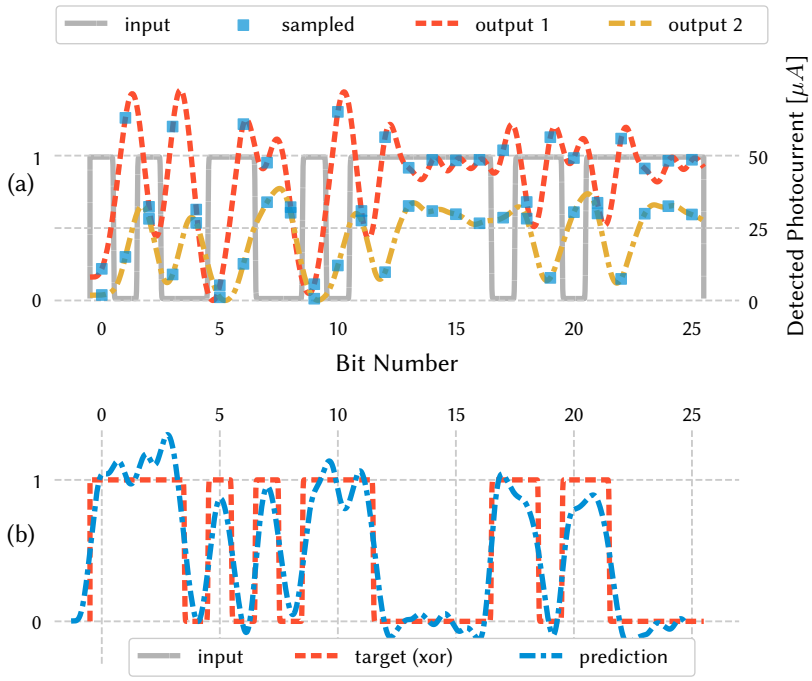


Figure 3.7: (a) Waveforms detected at two of the exit waveguides as the result of a certain 50 Gbps bit sequence input. The outputs are sampled at least once per bit period. (b) After the readout, the prediction approximates the desired XOR target. The prediction and the target were aligned by shifting the prediction backwards in time according to the optimal latency of 0.8 bits.

3.4 Cavity parameters

Obviously, even when sticking to the shapes introduced above[5], there are still a very large number of free parameters of the cavity that can be changed. The most important ones are the size of the cavity and the number of waveguides connecting into it. Indeed, changing either of those parameters will have a direct effect on the Q-factor and hence on the memory of the system. Moreover, changing the number of connected waveguides will *also* have an effect on the complexity of the tasks the reservoir can solve, as fewer waveguides equate to less rich linear combinations at the readout. Choosing the right combination of parameters it thus far from trivial.

3.4.1 Power budget of the reservoir

An important metric for reservoirs is how power efficient it is. The simulated total energy measured at the exits of the photonic crystal cavity is about $75\,\%$ of the total energy inserted, as can be seen in Fig. 3.8. Looking deeper into the source of the losses, we find that there is a slight mode mismatch between the access waveguides and the W1-defect photonic crystal waveguides. When coupling out the light from the cavity, this causes scattering out of the exit waveguides, resulting in lost power[6], as can also be seen in Fig. 3.2. Contrast this with a cavity of the same shape and size but which just relies on index contrast: only 25% of the input power is retrieved in that case. This is a direct consequence of the lack of omni-directional reflection at the cavity boundaries.

3.4.2 Q-factor and pulse half life

Apart from the power budget, which quantifies how much of the input power can be retrieved at the output, it is also interesting to quantify how long a signal is *"stuck"* in the cavity. This property is quantified by the *quality factor* (Q-factor) of the cavity.

To calculate the Q-factor, the response, as visualized in Fig. 3.2 of the $30\,\mu\mathrm{m} \times 60\,\mu\mathrm{m}$ photonic crystal cavity to a $1\,\mathrm{ps}$ pulse was simulated. When the light source is turned off, the field amplitude in the photonic crystal cavity decays exponentially as $\exp(-mt)$ as illustrated in Fig. 3.9, for which the envelope of the amplitude has a slope $m = -0.037\,\mathrm{ps}^{-1}$. This yields for the Q-factor at

[5] The reason these shapes were chosen is because they performed well for the specific applications in the mentioned reference works. Their choice is therefore rather arbitrary and it certainly could be that a different shape works better.

[6] Obviously this mode mismatch can probably be further reduced by a thorough optimization of the waveguide to W1-defect transition.

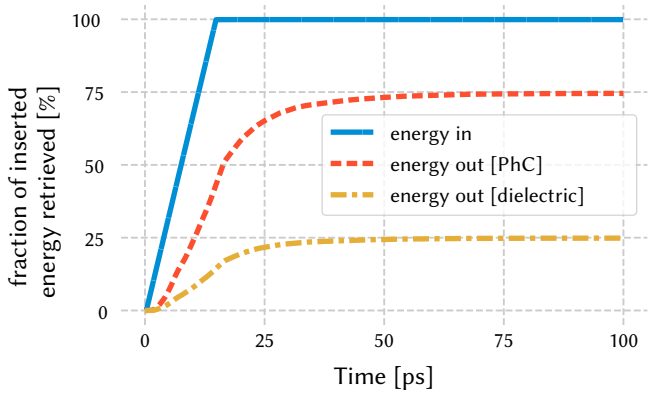


Figure 3.8: When inserting a $10\,\mathrm{ps}$ pulse into the $60\,\mu\mathrm{m} \times 30\,\mu\mathrm{m}$ photonic crystal cavity, about 75% of the total inserted energy is retrieved at the output waveguides. This corresponds to about $0.8\,\mathrm{dB}$ loss. Compare this to a cavity of the same shape and size but which just relies on index contrast: only 25% of the input power is retrieved.

$\lambda = 1550\,\mathrm{nm}$:

$$Q = -\frac{\pi c}{\lambda m} = 16400. \tag{3.5}$$

Perhaps a more useful value is the *half-life* $T_{1/2}$ of the pulse, as it provides a more tangible metric for the memory of the reservoir.

$$T_{1/2} = -\frac{\log(2)}{m} = 18\,\mathrm{ps}. \tag{3.6}$$

Since the losses are so much higher in the dielectric cavity, because the walls of the cavity cannot contain the light as well as in the photonic crystal case, we expect to need a larger cavity to obtain a higher Q-factor. Indeed, a large cavity will create longer propagation times *inside* the cavity and thus contain the signal for a longer period of time. For the dielectric chamfer cavity with a diameter of $100\,\mu\mathrm{m}$ the decay of the pulse is illustrated in Fig. 3.10. We get a pulse with a similar decay time as for the photonic crystal cavity:

$$T_{1/2} = 14\,\mathrm{ps}. \tag{3.7}$$

This half life is still worse than the one of the photonic crystal cavity, even though the cavity is about 4 times bigger (in area). The pulse itself also seems to be less *"rich"*, possibly due to the high power loss during reflections at the walls of the cavity.

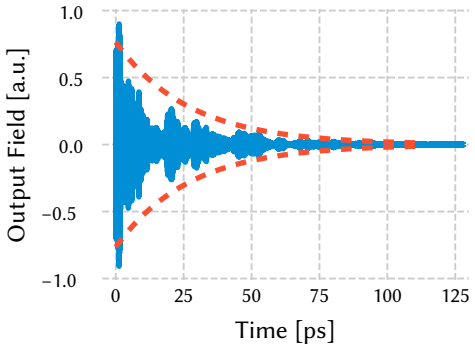


Figure 3.9: Decay of the field amplitude in the photonic crystal cavity. The amplitude decays with a half life $T_{1/2} = 18\,\mathrm{ps}$.

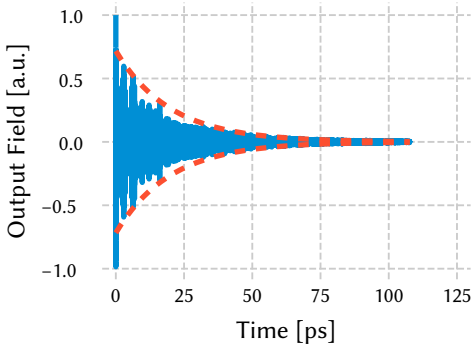


Figure 3.10: Decay of a pulse in the dielectric chamfer cavity with diameter $100\,\mu\mathrm{m}$. Note that this cavity, which in area is about 4 times bigger than the photonic crystal cavity has a *worse* half life and hence Q-factor.

3.5 Simulated boolean tasks

In our simulations, a PRBS of 10^5 bits is sent through one of the connected arms of the cavity. The responses of the other waveguides is then recorded. Finally, on the recorded output stream, the readout weights are trained to follow the intended target function, which can be the same as the input for the *copy* task, the XOR for two subsequent bits for the XOR task or any more complicated bit-level function on the stream.

The weights of the readout are chosen to minimize the mean squared error, as is show in Fig. 3.5. After performing a threshold, the bit error rate (BER) is calculated. Since we use 10^5 bits, the general guideline is to crop the BER at 10^{-3}, i.e. 2 orders of magnitude higher than the lowest BER one can find in the simulation [7]. This is obviously orders of magnitude higher than usual targets in telecom, which aim for a maximum BER of 10^{-9}. However, it is important to note that whenever a cropped BER of 10^{-3} is shown, very often that means no errors were made in during the processing of *all* the 10^5 bits.

To be able to show lower BERs, more bits should be simulated. The main reason no more bits were simulated in the case of the cavities is that the simulation procedure described previously — even though it is quite efficient, only having to simulate a single bit with FDTD — arguably will not yield much more interesting bit combinations: the pulse response for a single bit is *at most* about 15 times longer than the bit itself, giving on average only 2^{15} unique bit-combinations. Even after adding additional noise, simulating 10^5 bits is already a stretch.

We will proceed to go over the different tasks the reservoir will solve. We only send the bit stream through once and for each task a different *linear* readout is trained. This is the real power of reservoir computing: one reservoir can target many different applications by just changing the readout.

3.5.1 Copy task

The copy task is possibly the *easiest* task one can think of: send a bit stream through the reservoir and try to retrieve the same bit stream with a certain delay. Even though no special calculations need to be performed to do this operation, the copy task still serves as the prime measure for the *memory* of the reservoir.

We attempt to retrieve the original bit stream at different delays or *latencies* for the $60\,\mu\text{m} \times 30\,\mu\text{m}$ cavity, as illustrated in Fig. 3.11. We see in the figure that at the measured bitrate of $50\,\text{Gbps}$, the reservoir can remember the stream for *almost* 3 bits.

In fact, we can do the same copy operation for a whole range of bitrates, ranging from $15\,\text{Gbps}$ to $200\,\text{Gbps}$. We see in Fig. 3.12 that we get a wide region of operation. Moreover, as we would expect, the reservoir can successfully remember more bits for an increasing bitrate. The reason the reservoir stops working

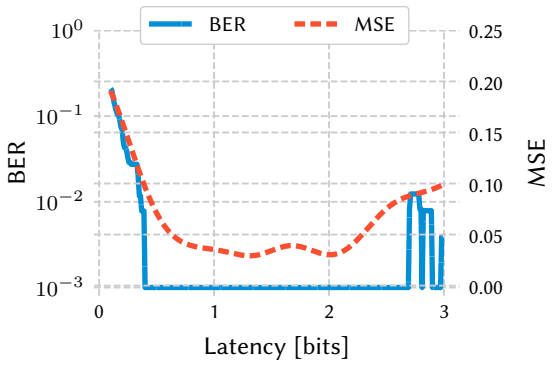


Figure 3.11: copy task at 50 Gbps performed with an increasing latency.

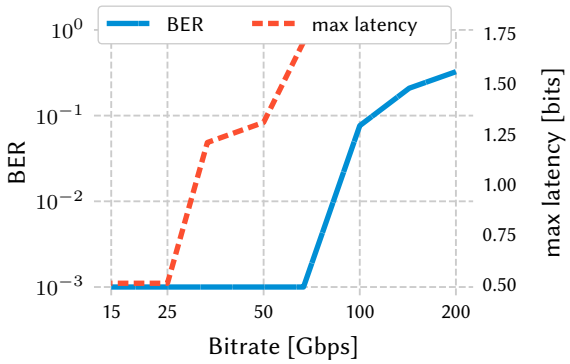


Figure 3.12: Sweep of the best copy-task performance for the $60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity at different bitrates. To save time, the sweep over the bitrates was done with 2D FDTD simulations.

at bitrates higher than 67 Gbps is probably because the reservoir remembers *too many* previous bits, which increases the signal-to-noise ratio on the most recent bits, which are relevant for the operation.

3.5.2 Header recognition

Obviously, the copy task is an incredibly easy task as at each point in time only one bit has to be recollected, no classification of or operation on the bits has to be performed. We therefore advance to a more difficult task: header recognition, which — for applications in telecom — is an incredibly useful task to be able to solve fast and accurately.

We first simulate the propagation of the bit stream at 50 Gbps through the

$60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity, then the readout weights are trained to recognize all the different headers present in the bit stream. Concretely, all different headers were searched for simultaneously in the random bit stream. For each bit in the bit stream, a class label was given corresponding to the header of length L consisting of the current bit and the $L-1$ previous bits. This procedure is shown in Table 3.1

L	...	1	0	1	1	0	1	1	1	...
2		...	2	1	3	2	1	3	3	...
3			...	5	3	6	5	3	7	...
4				...	11	6	13	11	7	...

Table 3.1: Labeling a random bit stream for different header lengths L.

Linear Discriminant Analysis (LDA) [6] was then used to find a different weight vector for each of the different classes, resulting in a weight matrix W_{out}. As can be seen in Fig. 3.13, the header recognition task at $50\,\text{Gbps}$ works up to 6 bit headers in a wide region of operation.

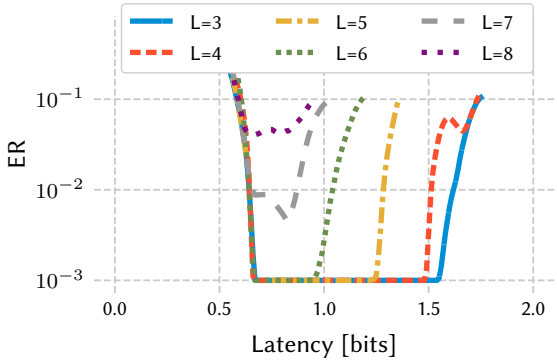
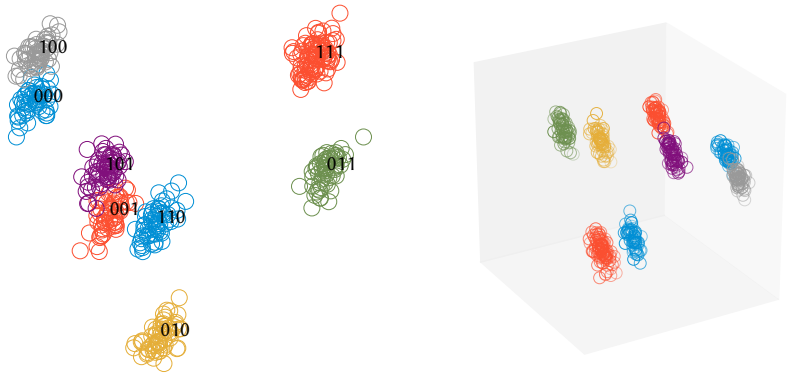


Figure 3.13: Error Rate (ER) for the worst performing header at each latency. The reservoir can distinguish headers of up to L=6 bits without error at the optimal bitrate of $50\,\text{Gbps}$. To reduce simulation times, the sweep over the latencies was stopped when the ER became higher than 10^{-1}.

To see the separation of the headers visually, we can make a projection from the 2^L-dimensional header-space to a lower dimensional space. This is illustrated in Fig. 3.14. This figure is a good illustration of the value of a projection to a higher dimension. Indeed, seeing the 2D and the 3D figures next to each other shows how a higher dimensional problem gets easier to separate: the locations of similar headers are clearly easier to separate in 3D than in 2D. Remember that the 3-bit header space is 8-dimensional. This high dimensionality is an important



(a) **(b)**

Figure 3.14: The separation of 3-bit headers can be visualized by projecting on the (a) two primary LDA axes or (b) three primary LDA axes. A nice separation for all different headers can be observed while similar headers are located closer together. Seeing the 2D and the 3D figures next to each other also serves as a good example on how a higher dimensional problem gets easier to separate: the locations of similar headers are clearly easier to separate in 3D than in 2D.

aspect of the perfect separation as seen in Fig. 3.13 and Fig. 3.15.

Finally, the same header recognition task is performed at different bitrates, as illustrated in Fig. 3.15. Here we clearly see that that longer headers can more easily be recognized at higher bitrates. This is unsurprising, as for longer headers, the reservoir needs to keep more bits in memory, therefore, the bitrate needs to be higher to accommodate this.

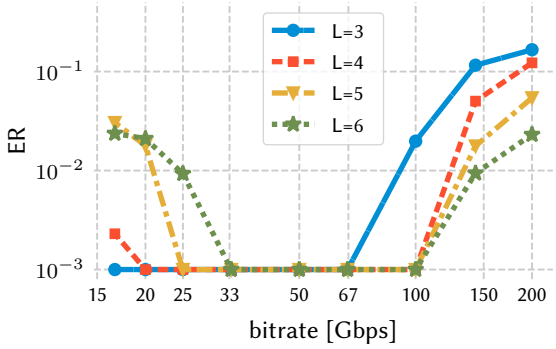


Figure 3.15: By sweeping over the bitrate to find the operation range, we find that the reservoir can distinguish headers up to a header length of $L = 6$ bits without error at a bitrate of up to $100\,\mathrm{Gbps}$. To save time, the sweep over the bitrates was done as a 2D FDTD simulation.

3.5.3 AND task

A slight step up from just *copying* the input and *recognizing* headers is to actually *operate* on the bits. One of the easiest bitwise operations one can perform is the AND operation on two subsequent bits of the bit stream[7]. Though still a linearly separable task, it is interesting to first check the performance of the cavity on this task before we move to the non-linear XOR task.

As usual, we first simulate the propagation of the bit stream at $50\,\mathrm{Gbps}$ through the $60\,\mu\mathrm{m} \times 30\,\mu\mathrm{m}$ photonic crystal cavity. As can be seen in Fig. 3.16, the reservoir is clearly able to perform the AND on two subsequent bits. The range of latencies for which this works is comparable to the range of the copy task, indicating that the AND task is in fact not much more difficult than just remembering the bits: not much non-linearity is necessary.

A slightly more difficult task[8] is the AND task of two bits with one bit in between (as opposed to the AND task of two subsequent bits). A different readout

[7] Actually, the AND operation is very similar to 2-bit header recognition hence the similar performance.

[8] In terms of memory; obviously it is still a linear task.

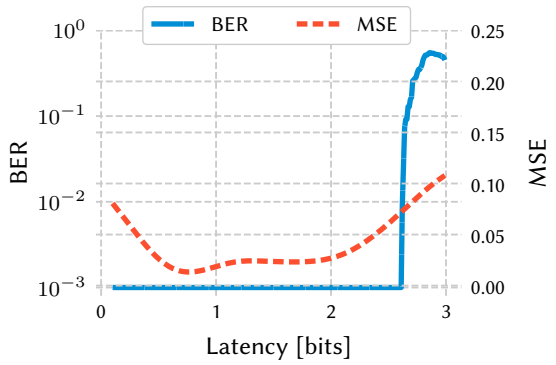


Figure 3.16: AND of two subsequent bits at 50 Gbps performed with a certain latency.

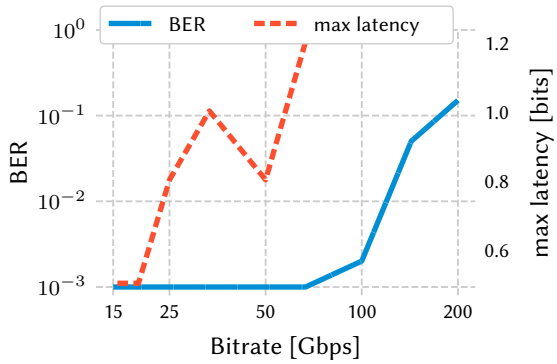


Figure 3.17: Sweep of the best AND performance for the $60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity at different bitrates. To save time, the sweep over the bitrates was done with 2D FDTD simulations.

was trained on the output of the 50 Gbps bit stream that went through the photonic crystal. The performance of this specific AND task is worse, but perfect classification can still be observed, as can be seen in Fig. 3.18

However, looking at the performance vs bitrate, we see that we actually got quite lucky with the bitrate choice at 50 Gbps for the photonic crystal cavity. Indeed, doing a sweep again over the bitrates we see that all the other bitrates perform worse: the wide region of operation is gone.

3.5.4 XOR task

Another binary task is the XOR of two consecutive bits. The XOR is known in machine learning to be a hard, non-linear task due to the fact that, as we have

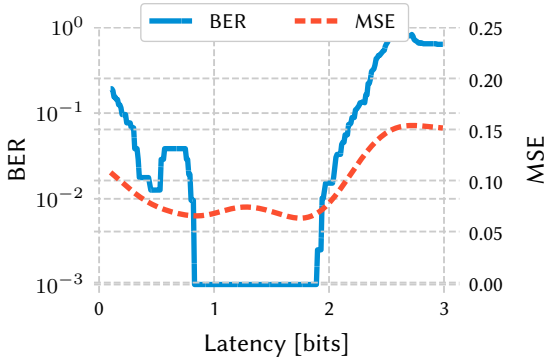


Figure 3.18: AND of two bits with one bit in between at 50 Gbps performed with a certain latency.

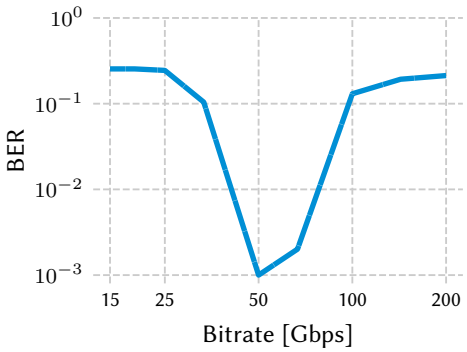


Figure 3.19: The AND performance of two bits with a bit in between. Performance is noticeably worse than for two subsequent bits. The wide region of operation is gone: this task only works at 50 Gbps. To save time, the sweep over the bitrates was done with 2D FDTD simulations.

touched upon in 1.3.6, the output cannot be found by just performing a linear classification algorithm such as linear regression on the inputs. At least *some* non-linearity in the system is necessary. As we know from 1.7.2, by relying on the non-linearity in the detector, we expect a passive system like the cavity discussed here to still be able to do this operation.

We first simulate the propagation of the bit stream at 50 Gbps through the $60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity. As can be seen in Fig. 3.20, the reservoir is clearly able to perform the XOR on two subsequent bits. The range of latencies for which this works is however quite small, another indication of the difficulty of the task: a certain alignment of the signals is required before the XOR operation

can be performed successfully.

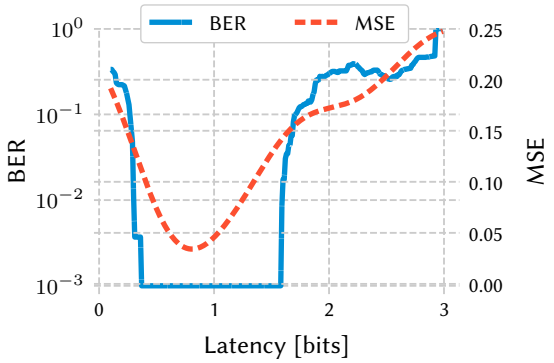


Figure 3.20: XOR of two subsequent at 50 Gbps bits performed with a certain latency.

Doing the same for bitrates ranging from 15 Gbps to 200 Gbps, we see that we get a wide region of operation for the photonic crystal cavity. Just like for the

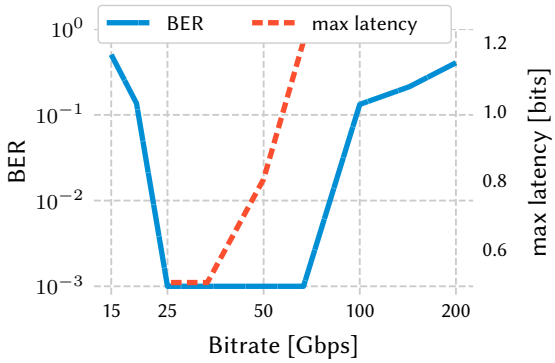


Figure 3.21: Sweep of the best XOR performance for the $60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity at different bitrates. To save time, the sweep over the bitrates was done with 2D FDTD simulations.

AND bitrate, the reservoir has a quite wide range of operation between 25 Gbps and 67 Gbps. The fact that the XOR task is more difficult than the linear AND task is however again reflected here, in the region of operation for the XOR, which is noticeably smaller than for the AND task.

Now that we have found the operating range of the $60\,\mu\text{m} \times 30\,\mu\text{m}$ cavity it is probably an interesting question to ask ourselves how to shift the operating range to higher bitrates. The most obvious way to do this is to reduce the size of the

cavity. Indeed, reducing the size of the photonic crystal cavity up to $6\,\mu\text{m} \times 4\,\mu\text{m}$ allows the operating range to shift to around $2\,\text{Tbps}$[9], as can be seen in Fig. 3.22.

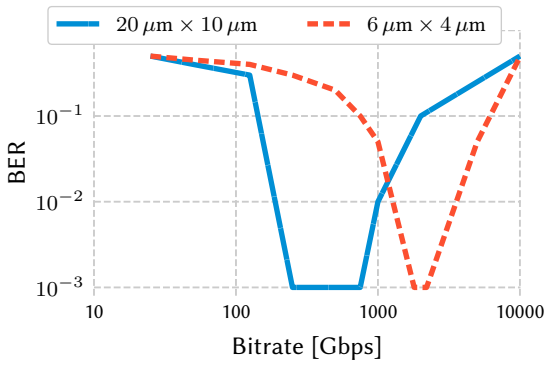


Figure 3.22: XOR vs bitrate for two smaller photonic crystal cavities of smaller size. Due to the smaller cavity size, these two sweeps were completely performed with 3D FDTD simulations.

Returning to the $60\,\mu\text{m} \times 30\,\mu\text{m}$ we can focus on the XOR task with a bit in between. However, as the maximum latency for which the normal XOR operation can successfully be applied nowhere exceeds two bits, we expect this task to fail for the cavity the cavity at hand. Indeed, this is confirmed by Fig. 3.23 and Fig. 3.24: the wide region of operation which could be observed earlier is completely gone and although the best performing bitrate is still at $50\,\text{Gbps}$, the minimal BER of 2.6% is just not good enough.

The interesting result here is that we succeeded to compute a highly non-linear function such as XOR by using a completely passive device. This is of course only possible because of the non-linearity of the photodetector, which takes the magnitude of the complex-valued field at the exits of the reservoir.

3.5.5 Number of arms

The cavities considered so far have always had 7 waveguides. However, taking 7 connected waveguides (1 input, 6 outputs) still seems as an arbitrary choice. Therefore, the Q-factor calculation of 3.4.2 was done again for a changing number of connected waveguides. As one would expect, reducing the number of arms will have a direct influence on the Q-factor and hence the memory of the system: we expect the cavity to retain the signal longer. However, with this larger memory comes a reduced complexity the reservoir is able to solve as a less rich linear

[9] For this the realistic detector model was disabled for obvious reasons.

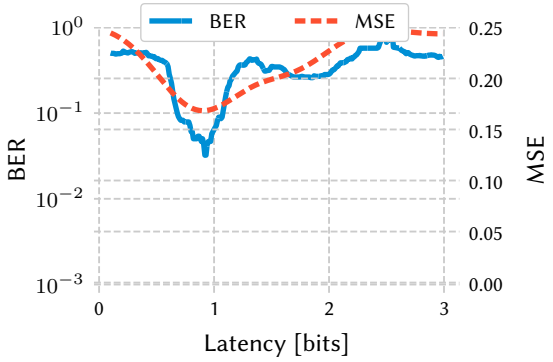


Figure 3.23: XOR of two bits with one bit in between at 50 Gbps performed with a certain latency.

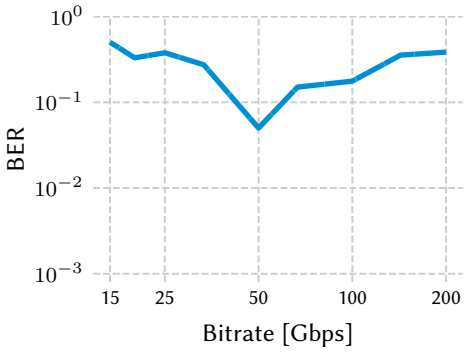


Figure 3.24: The XOR performance of two bits with a bit in between. Just like for the AND task, the performance is noticeably worse than for two subsequent bits. The wide region of operation is completely gone and although the best operating bitrate is still at 50 Gbps, the performance is still not good enough. To save time, the sweep over the bitrates was done with 2D FDTD simulations.

combination of the fields at the output waveguides can be taken. To quantify this trade-off, both the maximum latency for the copy task as the performance on the XOR task were tracked.

What we see in Fig. 3.25 is — apart from the expected decay of the Q-factor — that the XOR performance seems to have an abrupt *phase transition*-like performance increase when going from 5 to 6 connected waveguides. This basically means that the XOR problem becomes linearly separable in the 6-dimensional *cavity* output space.

Moreover, we also clearly see that the memory for the copy-task is highest

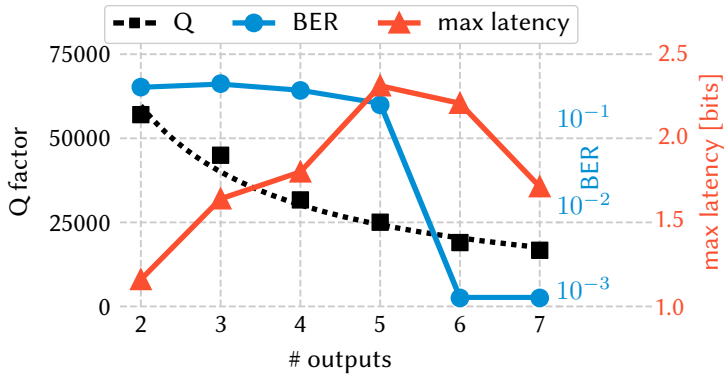


Figure 3.25: The Q-factor decays for an increasing number of output arms. Moreover, the BER on the XOR task seems to drastically increase when transitioning to six output waveguides. We can also see that a higher Q does not automatically relate to a longer memory capacity (expressed in maximum latency) for retrieving the original bit stream (copy task), possibly because the memory of the cavity fades *too slow*.

for the cavity with the 5 waveguides. This is counter-intuitive as naively one would think that the memory would be highest for a lower number of attached waveguides. A possible explanation for this could be that the fields are not fading away fast enough, resulting in *too* chaotic or unpredictable mixing with too many bits in the past.

The Q-factor decay of the photonic crystal cavity can be compared to the Q-factor decay of the dielectric chamfer cavity shown in Fig. 3.26. One immediately sees that first of all the Q-factor is lower for the dielectric cavity (as expected) and that the Q-factor is not as influenced by the number of arms connected into the cavity. This is of course also expected as most of the power gets lost at the interface of the cavity and hence the lost power by adding more arms is less dominant.

3.6 Fabrication

All fabricated cavities were created with an electron-beam (e-beam) lithography process, which works generally speaking as follows: a 240 nm thick layer of ARP-6200.09 resist is spin-coated on a silicon substrate and is then covered by a 60 nm Electra resist. The resist is then patterned by our Voyager 2 e-beam at 50 kV, 0.4622 nA, 20 mm working distance and 60 μm aperture at a beam speed of 58.8 mm/s and a nominal dose of 160 $\mu C/cm^2$.

After patterning, the resist is developed for 1 minute at room temperature with 99 % n-amylacetate. The chip is then etched until the required depth by Reactive Ion Etching (RIE) with a CF4 SF6 Ge mixture.

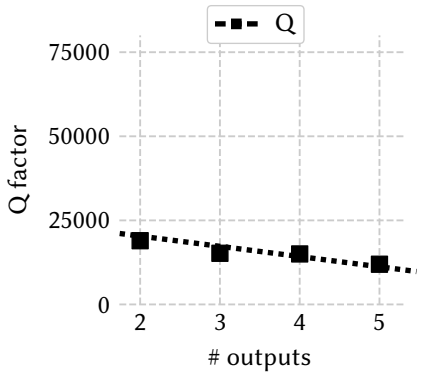


Figure 3.26: Q factor decay for an increasing number of output waveguides for a dielectric cavity. Since the base loss is already quite high, adding additional arms to the cavity will not have a big influence on the Q-factor.

There are two typical "*platforms*" for working with silicon photonics, named after the thickness of the crystalline silicon (Si) layer grown on the glass (SiO2) substrate: the 220 nm and the 400 nm platform as illustrated in Fig. 3.27. The 200 nm platform requires two etch steps: a deep etch (220 nm) for the trenches around guiding structures like waveguides, while the shallow etch (70 nm) is used for coupling light into the waveguides through Grating Couplers (GCs). Fabrication of structures for the 400 nm platform is generally speaking easier, as only a single etch step (180 nm) is needed to create *both* waveguides and grating couplers. However, due to these shallow-etched waveguides, only a single mode (the TE mode) is well confined, other modes leak out quite rapidly[10].

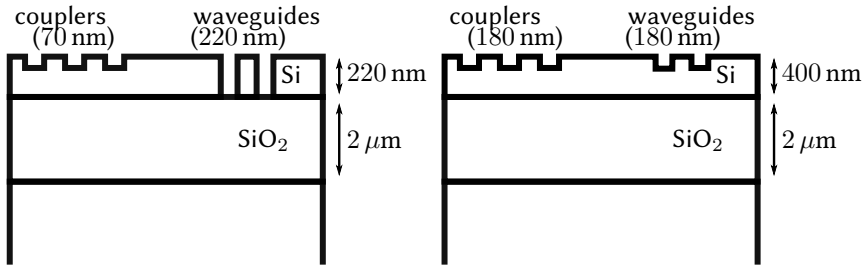
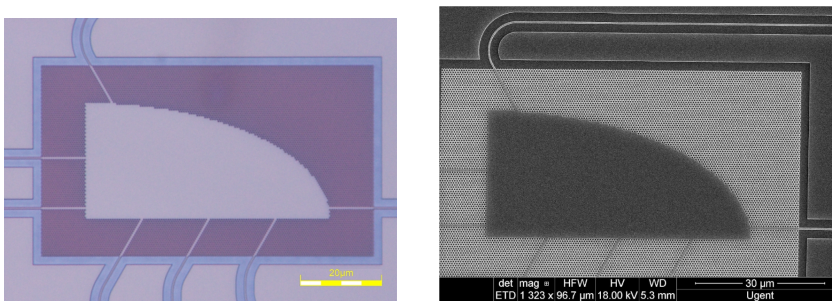


Figure 3.27: Most silicon photonic chips consist of structures patterned on (a) a 220 nm or (b) a 400 nm thick silicon-on-insulator (SOI) structure.

[10] a problem we encountered while creating cavities on the 400 nm platform, as will be discussed later.

During the course of this PhD, several incarnations of cavities were made. The first generation of cavities followed the photonic crystal design shown in Fig. 3.28. The cavities were made on the 220 nm platform by a two step process. For our initial attempt, the waveguides and photonic crystal holes were fully etched into the 220 nm thick silicon slab and then the grating couplers were shallowly etched 80 nm deep. However, due to the large cavity size, the decision was made not to under-etch the photonic crystal. The obvious drawback of this is that this breaks the confinement of the light in the z-direction due to asymmetry in refractive index below (SiO_2) and above (air) the photonic crystal. Therefore, we expect the losses to be rather high. Additionally, the e-beam fabrication procedure for photonic crystals is rather challenging, and was still being developed in our lab at the time. As expected in this first attempt, losses in the



(a) **(b)**

Figure 3.28: (a) Microscope image and (b) Scanning electron microscope image of the $60\,\mu\text{m} \times 30\,\mu\text{m}$ photonic crystal cavity fabricated with electron beam lithography.

cavity were way too high as illustrated in Fig. 3.29, where a typical transmission profile for the cavity at one of the arms is pictured. Generally speaking, at least -40 dBm is necessary: around the same amount as the total amplification available in the high-speed setup, as the high-speed photodetector expects a 0 dBm input (see also 3.7.1).

The decision was then made to considerably simplify the fabrication process: the second generation of cavities manufactured were of the dielectric type as illustrated in Fig. 3.30. However, this time, the cavity was fabricated on the 400 nm platform. The advantage of this platform is the ease of manufacturing: all structures are etched 180 nm deep into the 400 nm silicon slab. This only requires a single etch step instead of the two on the 220 nm platform. However, the shallowly etched structures on the 400 nm platform provide no confinement for TM modes and due to mode-mixing in the cavity, we expect a non-trivial amount of loss. Simulating the exact loss values for large 3D structures is rather challenging, therefore we decided to proceed with the fabrication anyway. However, as

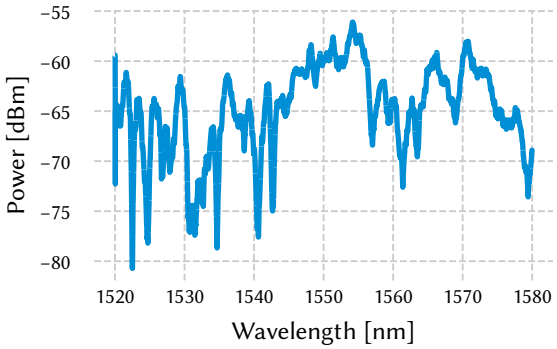


Figure 3.29: Measured transmission of the photonic crystal cavity. Losses in the manufactured photonic crystal cavities were way too high to advance to high speed measurements. (Not normalized with respect to grating coupler loss $2 \times 7.5\,\mathrm{dB}$.)

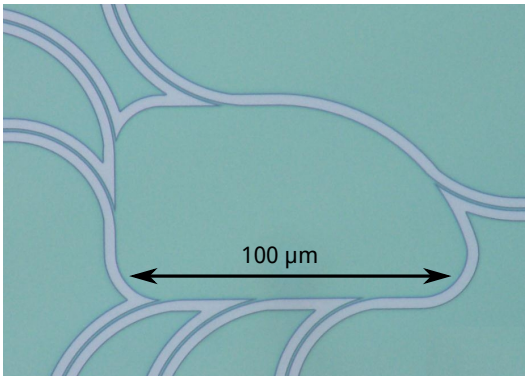


Figure 3.30: Microscope image of a $100\,\mu\mathrm{m} \times 50\,\mu\mathrm{m}$ cavity made on the $400\,\mathrm{nm}$ platform.

reflected in the transmission curves for this cavity in Fig. 3.31, the losses are too high to qualify for high-speed measurements.

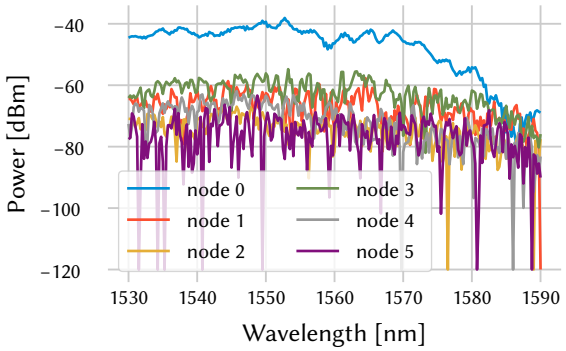


Figure 3.31: Measured transmission for all the arms of the dielectric cavity made on the 400 nm platform. None of these transmissions qualify for a high speed measurement. (Not normalized with grating coupler loss 2×7.5 dB.)

For the third generation of cavities, we returned to the 220 nm platform for better confinement of the modes; a choice motivated by a higher confinement of the TM mode. Moreover, to play it extra safe, a shape was chosen for which

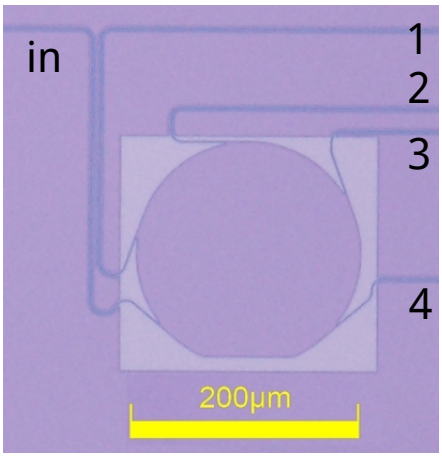


Figure 3.32: Microscope image of a dielectric chamfer cavity with 200 μm diameter. The cavity has a single input arm and four output arms which we label accordingly.

we knew it should have low enough losses [5]: the chamfer cavity visualized in Fig. 3.32. This generation of chamfer cavities were made in many different sizes, ranging from 30 μm diameter up to 500 μm diameter. A typical transmission

measurement in the frequency domain for the $100\,\mu$m cavity is shown in Fig. 3.33. As typical transmission is above 40 dBm, these cavities qualify for high speed

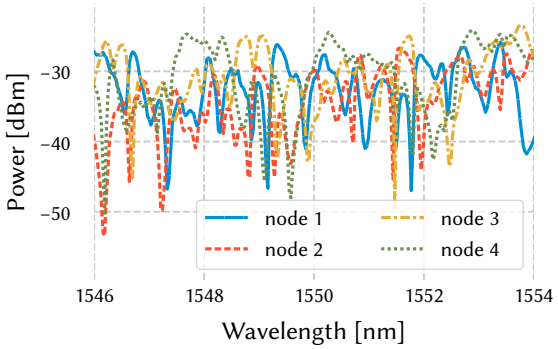


Figure 3.33: Measured transmission around 1550 nm for all the output arms in the $100\,\mu$m diameter chamfer cavity. (Not normalized with respect to grating coupler loss: 2×7.5 dB.)

measurements.

3.7 High speed measurements

3.7.1 High speed setup

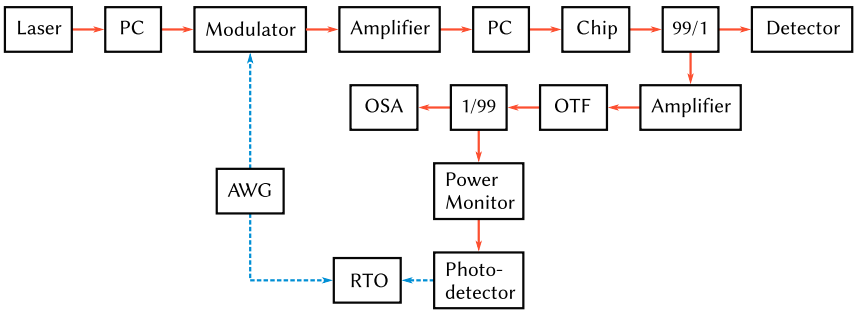


Figure 3.34: Diagram of the high speed measurement setup.

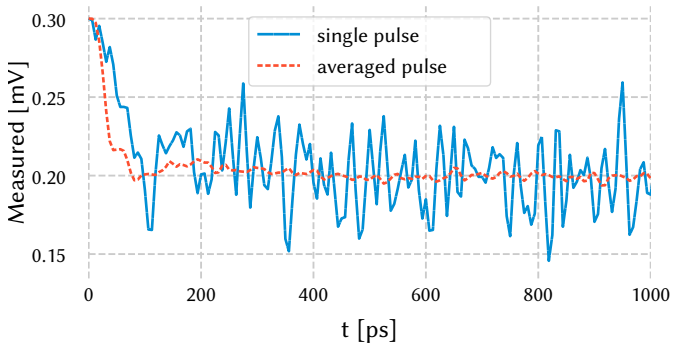
The high-speed measurement process is illustrated in Fig. 3.34. During this process, 1550 nm laser light is modulated by a Keysight M8195A Arbitrary Waveform Generator (AWG) according to a PRBS of 10^5 bits[11] with a bit rate of up to

[11] Just like before, using only 10^5 bits limits the minimum achievable BER to 10^{-3}. However, this

64 Gbps and a sample rate of 160 GHz. The laser beam is brought into the correct polarization with a Polarization Controller (PC), after which it is modulated by the signal coming from the AWG. The signal is then pre-amplified by a Keopsys CEFA-C-HG amplifier (+15 dB). The light is then brought again in the correct polarization by a second PC and coupled into the chip through an on-chip grating coupler (−7.5 dB). After transmission through the cavity (−15 dB), the light is coupled out of the chip by another grating coupler (−7.5 dB). The signal then passes another Keopsys CEFA-C-HG amplifier (+15 dB). Due to both amplifications, the spectrum is significantly broadened, which is partially cleaned up by an Optical Tunable Filter (OTF). The spectrum can be monitored by an Optical Spectrum Analyzer (OSA) and finally the light is detected with a high-speed photodiode. The electrical signal is sent to a Keysight DSA-Z 634A Real Time Oscilloscope (RTO) with a sample rate of 160 Gbps where it is compared to the original signal coming from the AWG.

3.7.2 Pulse response

Just like during the simulation, it is instructive to first look at the pulse response to see how long a signal remains inside the cavity. As can be seen in Fig. 3.35, the



.

Figure 3.35: Measured pulse response of the 100 μm diameter chamfer cavity. Blue line: single pulse; Red line: when averaging 10 pulses a small significant bump reveals itself around 200 ps

pulse remains about 100 ps inside the cavity before it disappears into the noise.

The noise of the signal was uncharacteristically high, even for reference measurements. Meanwhile, the cause of this has been identified as a defective am-

time, the bits are standard NRZ encoded, in contrast to the RZ encoding we were restricted to during the pulse composition technique used in simulation.

plifier. As a temporary stop-gap measure, multiple pulses can be averaged together to reduce the noise. This has the effect that a small bump becomes visible at about 200 ps after the pulse has arrived. In all future experiments, we will perform an average over 10 pulses, while we are waiting for a better amplifier.

The averaged pulse response for each of the cavity sizes can also be compared, as illustrated in Fig. 3.36, where the response to a 30 ps pulse measured at arm 1 of each of the cavities is visualized. Here we clearly see that the responses get richer with increasing cavity size.

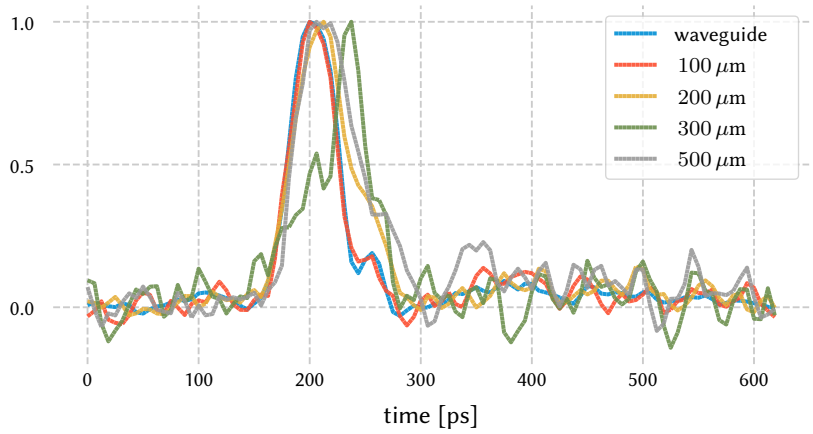


Figure 3.36: Comparison between the normalized 30 ps pulse responses for each of the cavities measured at arm 1 and averaged over 20 streams. The 100 μm response looks almost indistinguishable from the response of the waveguide, indicating that the dynamics in this cavity are probably not rich enough for the any of the tasks at hand. The responses of the $200 - 500$ μm cavities are a bit stretched out.

3.7.3 Copy task

Just as during the simulations, the memory of the system can be quantified by the performance on the copy task. As the 100 μm diameter chamfer cavity has a similar Q-factor as the photonic crystal cavity, we expect the memory of this cavity to be similar as long as the signal does not get lost into the noise.

As can be seen in Fig. 3.37, the memory of the 100 μm diameter cavity at 48 Gbps is about two bits.

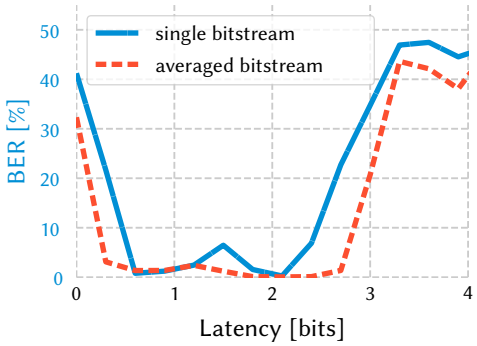


Figure 3.37: Measured copy task performance for the $100\,\mu$m diameter chamfer cavity. To reduce the noise on the measurement, the performance of the readout on the average of 10 bit streams (dashed red line) is also included. At the measured bitrate of 48 Gbps the reservoir has a memory of about two bits.

3.7.4 Header recognition

The performance on recognizing 3-bit headers was measured for different cavity sizes ranging from $100\,\mu$m diameter to $500\,\mu$m diameter. The performance is quantified by the error rate on the *worst* performing header at each bitrate. As can be seen in Fig. 3.38, each of the cavities can achieve near perfect performance for at least one bitrate. We can also see that larger cavities have an operating

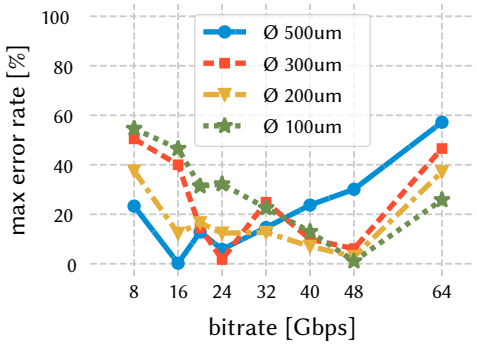


Figure 3.38: 3-bit header recognition. Performance on the worst performing header at each bitrate for a sweep of cavity sizes.

range at lower bitrates, just like one would expect. Just like we did before during the simulations, we can plot the separation of the headers on a 2D-projection as

illustrated in Fig. 3.39. We see that such a 2D projection is clearly not enough to separate them, although a rough grouping can be observed. This figure serves as a nice qualitative visual representation of which headers get confused with each other the most.

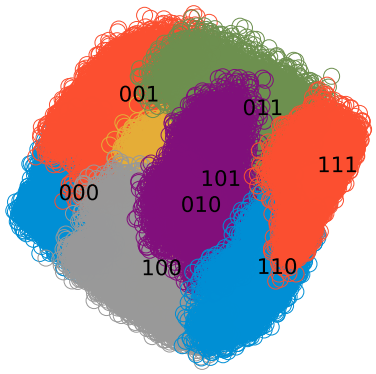


Figure 3.39: Although the measured headers for the 200 μm diameter cavity are not completely separable as there is a max error rate of 2%, we still can see distinct regions for each header. Moreover, similar headers seem to be closer together.

The performance for 4 bit headers is clearly worse: as can be seen in Fig. 3.40 the best max error rate is around 20%, clearly too high.

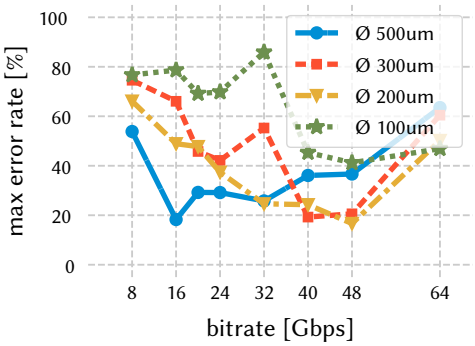


Figure 3.40: 4-bit header recognition. Performance on the worst performing header at each bitrate for a sweep of cavity sizes.

3.7.5 AND Task

Just like in the simulations, the maximum latency for the AND task is about 2 bits, as can be seen in Fig. 3.41.

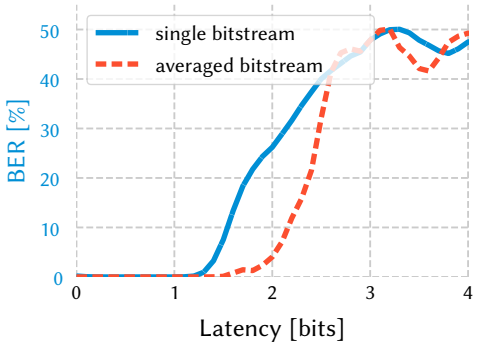


Figure 3.41: performance on the AND task for the 500 μm cavity at 16Gbps

3.7.6 XOR Task

As usual we start of the discussion with the BER vs latency plot, as illustrated in Fig. 3.42. Here we clearly see the advantage of reducing the noise by averaging multiple bit streams: the minimal BER drops from about 20% to about 0.6%.

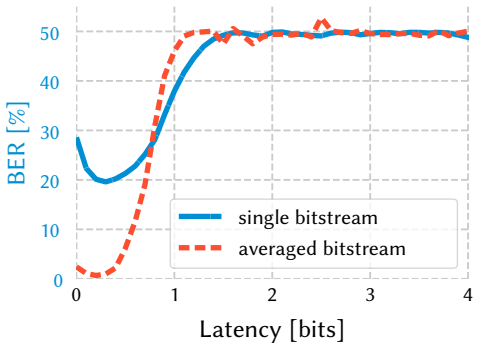


Figure 3.42: performance on the XOR task for the 200 μm cavity at 16 Gbps. The minimal bit error rate is 0.6% for the 10$\times$ averaged bit stream.

When looking at the bit error rate of the dielectric chamfer cavity at different bitrates and at different diameters of the cavity, we see that we have found a

sweet spot at 16 Gbps for the 200 μm diameter cavity. Both the smaller cavity *and* bigger cavities perform worse.

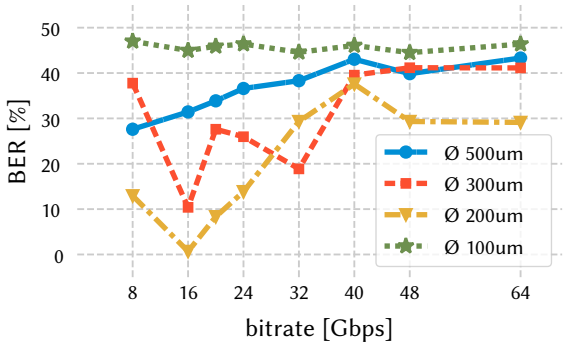
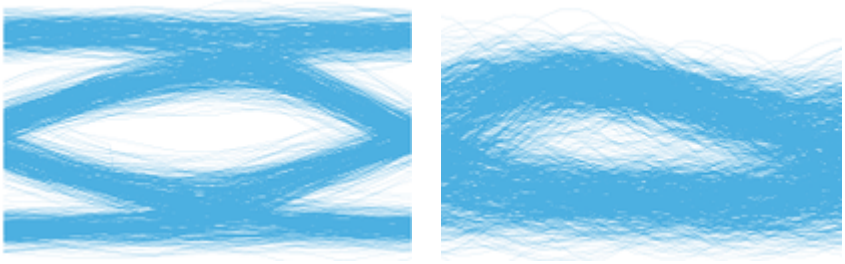


Figure 3.43: Measured BER on the xor task at different bitrates. The error rate was reported on a 10× averaged bit stream.

The reason the large cavities perform worse can be attributed to noise. Whereas the average power coming through an arm from the 100 μm diameter cavity is about -30 dBm, transmission for the largest cavity (500 μm diameter) approaches -40 dBm, the lower limit of the simulation.

The reason the smaller cavities perform worse is twofold. First of all, the smallest cavities ($< 200\,\mu$m) do not have rich enough field profiles: it is very hard to see the difference between the cavity response and the response of a normal waveguide. Indeed, looking at the averaged response at arm 1 of each of the cavities, we see that the 100 μm cavity response is almost indistinguishable from the waveguide response, as can be seen in Fig. 3.36. Moreover, the operation range of the smaller cavities lies at the higher bitrates, which — even though we measured up until 64 Gbps — leave the AWG noticeably deteriorated above 40 Gbps.

The recovered BER for the XOR task of 0.6% at the optimal bitrate and at the optimal cavity is still high for normal telecommunication standards. This is also reflected in the eye-diagram, as illustrated in Fig. 3.44, which becomes significantly more closed after the XOR operation. It has to be noted, however, that the fact that a linear system like the cavity can perform this non-linear XOR task at all (with a little non-linear help from the detector) is at least remarkable. Moreover, to perform this operation on a fully co-integrated electro-optical chip, no high-speed electronics are necessary as the linear combination necessary for the readout operation can possibly be implemented by a static and analog electric circuit



(a) (b)

Figure 3.44: Eye diagrams for (a) the incoming bit stream before the XOR operation and (b) the outgoing bit stream after the XOR operation at 16 Gbps. The difficulty of the XOR operation is reflected in the eye diagram which is significantly more closed after the operation.

3.8 Conclusions

This chapter introduced a completely new photonic reservoir computer on a silicon photonic chip. These cavities were benchmarked for typical telecommunication problems. They are able to perform basic telecom tasks such as *header recognition* and *boolean logic*.

In simulation, we were able to perform most of these tasks at the minimum determinable BER (about 10^{-3}). During the measurements, however, even higher bit error rates are observed. Moreover, the wide range of operation seems to be gone. A large part of this worse performance can probably be attributed to a broken amplifier. Still, probably more work has to be performed to decrease the bit error rates even further, e.g. by designs with lower loss and better fabrication techniques.

That said, some conclusions on the design of cavity-based reservoir computers can be made. first of all, the cavity size has an immediate effect on the optimal operating range of the reservoir. This is something that was both confirmed in simulation and experiment. Moreover, during the measurements, we found an upper limit for the dielectric cavity size above which the losses in the cavity become too high for the reservoir operation to work. This is something that can probably be achieved by falling back on a photonic crystal cavity.

During the measurements, we seem to have found a lower lower limit on the cavity size. However, this lower limit probably has to be attributed to the limitations of the measurement equipment to date, which becomes too noisy both in generation and detection at high bitrates. This premise is in fact confirmed when looking at the simulation results, which easily allows reservoir operation up to 2 Tbps when the limitations of the setup are not taken into account.

References

[1] Floris Laporte, Andrew Katumba, Joni Dambre, and Peter Bienstman. *Numerical demonstration of neuromorphic computing with photonic crystal cavities.* Optics express, 26(7):7955–7964, 2018.

[2] H.-J. Stockmann and J. Stein. *Quantum chaos in billiards studied by microwave absorption.* Phys. Rev. Lett., 64:2215–2218, May 1990.

[3] M Sieber, U Smilansky, SC Creagh, and RG Littlejohn. *Non-generic spectral statistics in the quantized stadium billiard.* Journal of Physics A: Mathematical and General, 26(22):6217, 1993.

[4] Changxu Liu, Ruben EC Van Der Wel, Nir Rotenberg, L Kuipers, Thomas Fraser Krauss, Andrea Di Falco, and Andrea Fratalocchi. *Triggering extreme events at the nanoscale in photonic seas.* Nature Physics, 11(4):358–363, 2015.

[5] Brian C Grubel, Bryan T Bosworth, Michael R Kossey, Hongcheng Sun, A Brinton Cooper, Mark A Foster, and Amy C Foster. *Silicon photonic physical unclonable function.* Optics express, 25(11):12710–12721, 2017.

[6] Alan Julian Izenman. *Linear discriminant analysis.* In Modern multivariate statistical techniques, pages 237–280. Springer, 2013.

[7] M. Jeruchim. *Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems.* IEEE Journal on Selected Areas in Communications, 2(1):153–170, Jan 1984.

4

Neuromorphic Computing with Photorefractive Materials

In this chapter we will explore how *photorefractive* materials can be used as a medium for neuromorphic computing. These kind of materials change their refractive index as a reaction to the light propagating through them, which in turn has an effect on the light itself. We will explore how this interesting effect could be used in the context of neuromorphic computing.

A large part of this research relies on accurately simulating this interaction. Because of the vast timescale differences between the governing phenomena in a photorefractive crystal, a dedicated *Finite-Difference Time-Domain* (FDTD) simulator was necessary. A large part of this chapter will thus also be focused on explaining the implementation of the FDTD method with a specific focus on the photorefractive effect.

4.1 The Finite-Difference Time-Domain Method

The Finite Difference Time Domain (FDTD) method [1, 2] is one of the most-used ways to accurately simulate electromagnetic phenomena. In this part of the chapter, we'll introduce the basics of the FDTD simulator that was built to simulate photorefractive crystals. A large part of the FDTD simulator created for this work was originally inspired by [3], however the final implementation differs quite a bit.

4.1.1 Electromagnetism background

As we have touched upon in 2.1, any electromagnetic phenomenon is governed by Maxwell's equations:

$$\nabla \times \mathbf{E} = -\mu_0 \mu_r \frac{\partial \mathbf{H}}{\partial t} \qquad \nabla \cdot (\epsilon_0 \epsilon_r \mathbf{E}) = \rho \tag{4.1}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \epsilon_0 \epsilon_r \frac{\partial \mathbf{E}}{\partial t} \qquad \nabla \cdot (\mu_0 \mu_r \mathbf{H}) = 0, \tag{4.2}$$

for which the permittivity ϵ_r and the permeability μ_r are — in their most general form — tensors that act on the electric field $\mathbf{E}$ and the magnetic field $\mathbf{H}$.

Furthermore, at any moment in time, the energy density in the electromagnetic field can be written as

$$\mathcal{E} = \frac{1}{2}\epsilon_0 \epsilon_r E^2 + \frac{1}{2}\mu_0 \mu_r H^2 \tag{4.3}$$

If one wants a measure for the energy flow in the system, the *Poynting Vector* is used:

$$\mathcal{S} = \mathbf{E} \times \mathbf{H}$$

which describes the energy flow through a surface. Indeed: the Poynting vector $\mathcal{S}$ and the energy density have the following relationship:

$$\frac{d\mathcal{E}}{dt} = -\nabla \cdot \mathcal{S}. \tag{4.4}$$

4.1.2 Simulation units

We can choose a different definition for the fields $\mathbf{E}$ and $\mathbf{H}$ such that

$$\tilde{\mathbf{E}}(\mathbf{r}, t) = \sqrt{\epsilon_0}\mathbf{E}(\mathbf{r}, t) \tag{4.5}$$

$$\tilde{\mathbf{H}}(\mathbf{r}, t) = \sqrt{\mu_0}\mathbf{H}(\mathbf{r}, t), \tag{4.6}$$

Additionally, Maxwell's equations can be written as update equations[1]:

$$\tilde{\mathbf{H}}(\mathbf{r}, t + dt) = \tilde{\mathbf{H}}(\mathbf{r}, t) - cdt\mu_r^{-1}\left(\nabla \times \tilde{\mathbf{E}}(\mathbf{r}, t)\right), \tag{4.7}$$

$$\tilde{\mathbf{E}}(\mathbf{r}, t + dt) = \tilde{\mathbf{E}}(\mathbf{r}, t) + cdt\epsilon_r^{-1}\left(\nabla \times \tilde{\mathbf{H}}(\mathbf{r}, t)\right) \tag{4.8}$$

which have a nice symmetrical form in $\tilde{\mathbf{E}}$ and $\tilde{\mathbf{H}}$. Making such a substitution like (4.5) and (4.6) is important for methods like the FDTD method for which accuracy of the fields is important, as it makes sure both fields are of a similar magnitude close to one[2], which increases numerical stability.

[1] We are ignoring the current density J for now.

[2] In SI units, the relative magnitude difference between the fields is related by the electromagnetic impedance of free space $\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}}$, which for the current choice of simulation units equals 1 per definition.

Note that this particular choice of simulation parameters also changes the form of the energy density:

$$\mathcal{E} = \frac{1}{2}\left(\epsilon_r \tilde{E}^2 + \mu_r \tilde{H}^2\right), \tag{4.9}$$

while the expression for the Poynting vector becomes:

$$\mathcal{S} = c\tilde{\mathbf{E}} \times \tilde{\mathbf{H}}$$

4.1.3 Yee grid discretization

By defining the grid spacing du (the same in all three spatial dimensions) and the time step dt, the update equations can be discretized in both space and time. However, doing only that is not sufficient, as the *curl* of the fields $\tilde{\mathbf{E}}$ and $\tilde{\mathbf{H}}$ is not well defined for any discretized space. In 1966, Yee et al. found a solution to this, which is now widely known as Yee-discretization [1].

According to the Yee discretization, there are inherently two types of fields on the grid: E-type fields, for which the x, y and z components have two integer coordinates and one half-integer coordinate, and H-type fields, for which the x, y and z components have two coordinates on half-integer grid locations and one coordinate on integer grid locations. By placing the 6 components of the electromagnetic fields each at a different location, one arrives at the staggered grid cell shown in Fig. 4.1, for which the E-type fields are defined on the edges

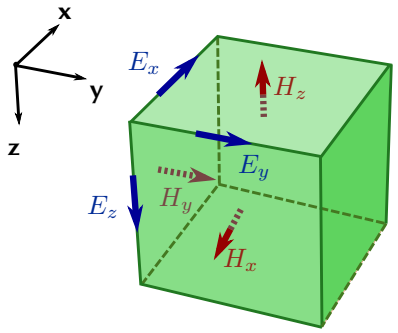


Figure 4.1: A unit cell on a Yee-grid. The E-fields are on the edges of the unit cell; the H-fields are on the faces of the unit cell.

of each unit cell of the grid and the H-type fields are defined on the faces of the unit cell. Furthermore, apart from the staggering in space, we also choose the coordinates to be staggered in time, such that we can write for the final Yee-

coordinates:

$$\mathbf{H}[m,n,p,q] = \begin{pmatrix} H_x\left(mdu,(n+\frac{1}{2})du,(p+\frac{1}{2})du,qdt\right) \\ H_y\left((m+\frac{1}{2})du,ndu,(p+\frac{1}{2})du,qdt\right) \\ H_z\left((m+\frac{1}{2})du,(n+\frac{1}{2})du,pdu,qdt\right) \end{pmatrix} \tag{4.10}$$

$$\mathbf{E}[m,n,p,q] = \begin{pmatrix} E_x\left((m+\frac{1}{2})du,ndu,pdu,(q+\frac{1}{2})dt\right) \\ E_y\left(mdu,(n+\frac{1}{2})du,pdu,(q+\frac{1}{2})dt\right) \\ E_z\left(mdu,ndu,(p+\frac{1}{2})du,(q+\frac{1}{2})dt\right) \end{pmatrix} \tag{4.11}$$

The beauty of these interlaced coordinates is that they enable a very natural way of writing the curl for the electric and magnetic fields: the curl of an H-type field will be transformed to an E-type field and vice versa. Defined on these coordinates, the curl of E can be written as:

$$\Phi_{\mathbf{E}}[m,n,p] = du\nabla\times\mathbf{E}[m,n,p] = du\begin{pmatrix} \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \\ \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} \\ \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \end{pmatrix}[m,n,p] \tag{4.12}$$

$$= \begin{pmatrix} (E_z[m,n+1,p] - E_z[m,n,p]) - (E_y[m,n,p+1] - E_y[m,n,p]) \\ (E_x[m,n,p+1] - E_x[m,n,p]) - (E_z[m+1,n,p] - E_z[m,n,p]) \\ (E_y[m+1,n,p] - E_y[m,n,p]) - (E_x[m,n+1,p] - E_x[m,n,p]) \end{pmatrix}$$

In which the half-integer indices are implicitly assumed. The curl of H can be obtained in a similar way, but due to their position in the grid cell the right difference $(E[m+1]-E[m])$ turns into a left difference $(H[m]-H[m-1])$:

$$\Phi_{\mathbf{H}}[m,n,p] = du\nabla\times\mathbf{H}[m,n,p] = du\begin{pmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{pmatrix}[m,n,p] \tag{4.13}$$

$$= \begin{pmatrix} (H_z[m,n,p] - H_z[m,n-1,p]) - (H_y[m,n,p] - H_y[m,n,p-1]) \\ (H_x[m,n,p] - H_x[m,n,p-1]) - (H_z[m,n,p] - H_z[m-1,n,p]) \\ (H_y[m,n,p] - H_y[m-1,n,p]) - (H_x[m,n,p] - H_x[m,n-1,p]) \end{pmatrix}$$

4.1.4 Update equations

Using the Yee discretization for the electromagnetic fields and the above definitions for the curl of the fields, yields the following update equations which leap-frog each other (we leave out the spatial dependency, as they are all contained within the curls Φ_E and Φ_H.):

$$\tilde{\mathbf{H}}[q+1] = \tilde{\mathbf{H}}[q] - s_c\mu_r^{-1}\Phi_{\tilde{\mathbf{E}}}[q] \tag{4.14}$$

$$\tilde{\mathbf{E}}[q+1] = \tilde{\mathbf{E}}[q] + s_c\epsilon_r^{-1}\Phi_{\tilde{\mathbf{H}}}[q+1], \tag{4.15}$$

where we defined the dimensionless number

$$s_c = \frac{cdt}{du}, \tag{4.16}$$

called the *Courant number* s_c. For stability reasons, the Courant number should always be smaller than $1/\sqrt{D}$, with D the dimension of the simulation. This can be intuitively be understood as the condition that information should always travel slower than the speed of light through the grid. In the FDTD method described here, information can only travel to the neighboring grid points (through application of the curl). It would therefore take D time steps to travel over the diagonal of a D-dimensional cube (square in $2D$, cube in $3D$). The Courant condition follows then automatically from the fact that the length of this diagonal is $du\sqrt{D}$:

$$\frac{cDdt}{du\sqrt{D}} < 1 \Rightarrow s_c = \frac{cdt}{du} < \frac{1}{\sqrt{D}} \tag{4.17}$$

4.1.5 Sensible defaults

In order to have an accurate simulation a good rule of thumb is to choose the grid spacing of the FDTD simulation 10 times smaller than the *smallest* wavelength in the grid. When working with $1550\,\mathrm{nm}$ light in silicon ($n \approx 3.1$) this would result in a grid spacing of approximately $50\,\mathrm{nm}$.

Moreover, for stability reasons it is recommended to choose the time step to exactly equal the Courant condition, which in 3D results in a time step of about $0.1\,\mathrm{fs}$.

4.1.6 Sources

So far, we have ignored the current density in the update equations. However, without having a way to handle this, it would be impossible to introduce energy into the grid. We can include the current density $\mathbf{J}$ into the update equation for $\tilde{\mathbf{E}}$ as follows:

$$\begin{aligned}\tilde{\mathbf{E}}[q+1] &= \tilde{\mathbf{E}}[q] + s_c\epsilon_r^{-1}\left(\Phi_{\tilde{\mathbf{H}}}[q+1] - \tilde{\mathbf{J}}[q+1]\right) &(4.18)\\ &= \tilde{\mathbf{E}}[q] + s_c\epsilon_r^{-1}\Phi_{\tilde{\mathbf{H}}}[q+1] + \tilde{\mathbf{E}}_s[q+1], &(4.19)\end{aligned}$$

where we defined the current density in simulation units to be $\tilde{\mathbf{J}} = \mathbf{J}du/\sqrt{\epsilon_0}$. Furthermore, we introduced $\tilde{\mathbf{E}}_s$ to be the electric field source term.

Moreover, it is often useful to also define a *magnetic field source term* $\tilde{\mathbf{H}}_s$, which would be derived from the *magnetic current density* $\mathbf{J}_m$ if it were to exist. In the same way, the update equation for $\mathbf{H}$ can be rewritten as:

$$\begin{aligned}\tilde{\mathbf{H}}[q+1] &= \tilde{\mathbf{H}}[q] - s_c\mu_r^{-1}\left(\Phi_{\tilde{\mathbf{E}}}[q] - \tilde{\mathbf{J}}_m[q]\right) &(4.20)\\ &= \tilde{\mathbf{H}}[q] - s_c\mu_r^{-1}\Phi_{\tilde{\mathbf{E}}}[q] - \tilde{\mathbf{H}}_s[q+1]. &(4.21)\end{aligned}$$

In practice, it is easier to directly define the source terms $\tilde{\mathbf{E}}_s$ and $\tilde{\mathbf{H}}_s$ instead of their respective current densities. The sources defined in the way described above are called *soft* sources due to their additive nature. For stability reasons, one might choose to implement a *hard* source instead by setting the value of the fields fixed to the source value. When defined like this, the field value inside the source is *only* defined by the source itself, not by the fields around it. An unfortunate effect of choosing a hard source is however that the fields around them reflect from them as they are not allowed to travel through.

4.1.7 Lossy Medium

When a material has a *electric conductivity* σ, a conduction-current $\mathbf{J}_\sigma = \sigma\mathbf{E}$ will ensure that the medium is lossy. Plugging this conduction current into (4.18) yields:

$$\tilde{\mathbf{E}}[q+1] = \tilde{\mathbf{E}}[q] + s_c\epsilon_r^{-1}\left(\Phi_{\tilde{\mathbf{H}}}[q+1] - \frac{\tilde{\sigma}}{2}\left(\tilde{\mathbf{E}}[q] + \tilde{\mathbf{E}}[q+1]\right)\right) \tag{4.22}$$

Where we defined $\tilde{\sigma} = \sigma du/\epsilon_0$ and where we interpolated $\sigma\tilde{\mathbf{E}}$ to be defined on an integer time step such that it is aligned in time with $\Phi_{\tilde{\mathbf{H}}}$. We can define the parameter f:

$$f = \frac{1}{2}s_c\epsilon_r^{-1}\tilde{\sigma}, \tag{4.23}$$

such that the update equation for $\mathbf{E}$ becomes:

$$\tilde{\mathbf{E}}[q+1] = \frac{1-f}{1+f}\tilde{\mathbf{E}}[q] + \frac{1}{1+f}s_c\epsilon_r^{-1}\Phi_{\tilde{\mathbf{H}}}[q+1] \tag{4.24}$$

Note that the above equation is only valid in the case of a diagonal inverse permittivity tensor ϵ_r^{-1}. If this is not the case — as is common for some photorefractive and electro-optical materials — the above equation becomes even more complicated and time-consuming to evaluate. It is therefore sometimes a good approximation to transfer the electromagnetic absorption to the magnetic domain by introducing a (nonphysical) magnetic conductivity σ_m, because for non-magnetic materials the permeability tensor μ_r is just equal to one. A similar substitution yields in that case:

$$f = \frac{1}{2}s_c\mu_r^{-1}\tilde{\sigma}_m, \tag{4.25}$$

such that the update equation for H becomes:

$$H[q+1] = \frac{1-f}{1+f}H[q] - \frac{1}{1+f}s_c\mu_r^{-1}\Phi_{\tilde{\mathbf{E}}}[q+1] \tag{4.26}$$

Later on, we will need to keep track of the absorption in the photorefractive material to calculate the strength of the resulting photorefractive effect. The easiest way to calculate the absorption is by keeping track of the induced change in energy density during the update of the fields:

$$\begin{aligned} d\mathcal{E} &= \frac{d\mathcal{E}}{dt}dt = c\tilde{\mathbf{H}} \cdot \frac{\partial \tilde{\mathbf{H}}}{\partial t}dt \\ \Rightarrow d\mathcal{E}_{\text{abs}} &= c\tilde{\mathbf{H}} \cdot \left(\left.\frac{\partial \tilde{\mathbf{H}}}{\partial t}\right|_{\sigma_m=0} - \left.\frac{\partial \tilde{\mathbf{H}}}{\partial t}\right|_{\sigma_m\neq 0} \right) dt \\ &= c\tilde{\mathbf{H}} \cdot d\tilde{\mathbf{H}}_{\text{abs}}, \end{aligned} \tag{4.27}$$

where $d\tilde{\mathbf{H}}_{\text{abs}}$ is obtained by subtracting (4.26) from (4.21). This expression is useful, as in photorefractive systems, the absorbed energy directly relates to the number of electrons freed (see 4.2.1). Note however that the inner product of two fields is not well defined, as each component has a different position in the grid cell. Therefore, the fields are first interpolated to the *corner* of the grid cell before the product is calculated. Moreover, since $d\tilde{\mathbf{H}}_{\text{abs}} \propto \Phi_{\tilde{\mathbf{E}}}$, which is only defined on half-integer time steps, $\tilde{\mathbf{H}}$ needs to be interpolated in time as well.

Another question we may ask ourselves is how the absorption introduced by a magnetic conductivity σ_m relates to the general description of absorption in a dielectric material:

$$I = I_0 \exp\left(-\alpha r\right) \tag{4.28}$$

Looking at the Maxwell equations, we have:

$$\nabla \times \mathbf{E} = -\mu_0\mu_r \frac{\partial \mathbf{H}}{\partial t} - \sigma_m \mathbf{H} \tag{4.29}$$

The right hand side becomes in the frequency domain:

$$\text{RHS} = -i\omega\mu_0 \left(\mu_r - i\frac{\sigma_m}{\mu_0\omega}\right)\mathbf{H} \tag{4.30}$$

$$= i\omega\bar{\mu}\mathbf{H}, \tag{4.31}$$

where we defined the complex relative permeability as $\bar{\mu}_r = \mu_r - i\frac{\sigma_M}{\mu_0\omega}$. This in turn leads to a complex refractive index:

$$\bar{n} = n - in_i = \sqrt{\epsilon_r\bar{\mu}_r} = \sqrt{\epsilon_r\mu_r - i\frac{\epsilon_r\sigma_m}{\mu_0\omega}} \tag{4.32}$$

Assuming μ_r to be a scalar, we get using $n = \sqrt{\epsilon_r\mu_r}$:

$$n^2 - n_i^2 - 2inn_i = n^2 - in^2\frac{\sigma_m}{\mu_r\mu_0\omega} \tag{4.33}$$

This is a quadratic equation in n_i, which yields:

$$n_i = n\frac{\sigma_m}{2\mu_r\mu_0\omega} = \sqrt{\epsilon_r}\frac{\sigma_m}{2\mu_0\omega}$$

Which gives us for the absorption α:

$$\alpha = \frac{2\pi\sqrt{\epsilon_r}\sigma_m}{\lambda\mu_0\omega} = \frac{\sqrt{\epsilon_r}\sigma_m}{\mu_0 c} = \frac{\sqrt{\epsilon_r}\sigma_m}{\eta_0}, \tag{4.34}$$

with $\eta_0 = \sqrt{\mu_0/\epsilon_0}$ the impedance or impermeability of free space.

Note also that if we would have done a similar derivation for the *electric* conductivity σ, the expression for the complex refractive index would be

$$\bar{n} = n - in_i = \sqrt{\epsilon_r\bar{\mu}_r} = \sqrt{\epsilon_r\mu_r - i\frac{\mu_r\sigma}{\mu_0\omega}} \tag{4.35}$$

The two expressions (4.32) and (4.35) are interchangeable if $\mu_r\sigma = \epsilon_r\sigma_m$, giving us a relationship between the physical *electric* conductivity and the unphysical *magnetic* conductivity. An anisotropic ϵ_r thus makes the effective loss due to a magnetic conductivity anisotropic as well. However, this effect is often a small price to pay for much more efficient update equations.

4.2 Simulating the photorefractive effect

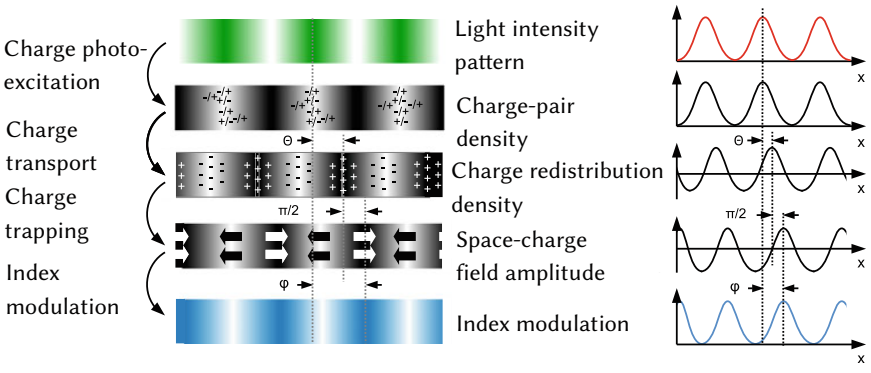


Figure 4.2: Left: the photorefractive effect defined by its microscopic processes. Right: amplitude of the different functions according to the space and phase shift between them. Figure from [4].

The photorefractive effect is a peculiar response some materials[3] have to an applied optical field. When illuminated with light, these materials develop a

[3] Quite a few inorganic materials exhibit the photorefractive effect. The most important materials being $LiNbO_3$, $KNbO_3$, $BaTiO_3$, Bi_12Si_20 and GaAs.

change in refractive index. The effect was first observed in the 1960s by Ashkin et al. [5] in what they called *optically induced lensing*. The effect was not well understood, until three years later an explanation of the microscopic processes governing the effect was given by Chen et al. [6]. This microscopic process relies on a careful interplay between the photons and the charges in the material and can be summarized by the following four-step process which is also illustrated in Fig. 4.2:

1. Photons excite charges in the illuminated regions of the material
2. The charges are now free to move through the bulk of the material
3. The charges are captured again in the dark regions of the material, giving rise to a inhomogeneous charge distribution
4. The induced charge distribution gives rise to a space-charge electric field modulating the refractive index of the material through the Pockels effect.

As this effect relies on the diffusion of electrons through the material, is said to be non-local. It is also a reversible effect, as the induced charge density can be removed by applying a different optical field or by a thermal process.

To include the Photorefractive effect into the FDTD simulator we have introduced previously, these 4 steps need to be accurately modeled. For this, we'll use the well-known Kukhtarev equations [7] in 4.2.1 to model the excitation and trapping of the electrons, we'll then describe how we'll model the diffusion in 4.2.2, followed by a discussion on how to find the space charge electric field in 4.2.3 and the resulting index change due the the pockels effect in 4.2.4. Although all of these phenomena are well-known, the way they are integrated into the FDTD method, as discussed in 4.2.6 and 4.2.7 has — to our knowledge — never been done before.

4.2.1 Kukhtarev equations

When numerically modeling the process above, we will need to make some generally accepted assumptions and simplifications to the problem. First, no difference is made between traps and donors in the photorefractive material: we assume any unfilled trap is positively charged and any filled trap is a (neutral) donor, while in reality there may be some irreversible effects. Second, it is assumed that all traps have the same energy. Finally, we assume that electrons can not be excited from inside the valence band — only from within the traps described above. These assumptions were first proposed by Kukhtarev in 1978 [7],

who also proposed the excitation and diffusion equations resulting from them:

$$\frac{dn}{dt} = \frac{dN_D^+}{dt} + \nabla \cdot \mathbf{J} \tag{4.36}$$

$$\frac{dN_D^+}{dt} = (sI + \beta)(N_D - N_D^+) - \gamma n N_D^+ \tag{4.37}$$

$$\mathbf{J} = \frac{\mu k T}{e}\nabla n - \mu n \mathbf{S} \tag{4.38}$$

The first equation describes how the change in free electron density n is related to the change in excited donor density N_D^+ and the spatial gradient of the current density J.

For the second equation, the change in excited donor density N_D^+ can be split into a *generative* term and a *recombination* term. The generative term will be proportional (through a photo-ionization cross-section s) to the intensity of the light I, which in this case is defined in terms of the energy density $I = c\mathcal{E}$. Assuming the only absorption in the photorefractive material is due to the photo-ionization, we can propose a relation between the *photoionization* s and the absorption coefficient α[4]:

$$\alpha = s\frac{\hbar c}{\lambda}(N_D - N_D^+) \tag{4.39}$$

The generative term will also be proportional to a thermal excitation rate β. Both the photo-ionization and the thermal excitation will obviously only be able to excite from the non-excited donors $N_D - N_D^+$. On the other hand, the recombination term will be proportional to the number of free electrons n and the number of excited donors N_D^+ through a recombination rate γ.

Finally, the current density $\mathbf{J}$ in the third equation is related to the gradient of the electron density ∇n *and* to the actual electron density n multiplied by the Poynting vector $\mathbf{S}$ through the mobility μ[5].

4.2.2 Electron diffusion

Consider the diffusion equation:

$$\frac{\partial n}{\partial t} = \left.\frac{\partial n}{\partial t}\right|_{\text{diff}} + \left.\frac{\partial n}{\partial t}\right|_{\text{drift}} = D\nabla^2 n - \nabla \cdot \mathbf{F} \tag{4.40}$$

with D the diffusion constant:

$$D = \frac{kT}{e}\mu \tag{4.41}$$

[4]The assumption that the absorption is completely due to the photo-ionization is an approximation. It gives the lower bound for the absorption given the photo-ionization cross-section s.

[5]obviously different from the magnetic permittivity $\mu_r\mu_0$

and $\mathbf{F}$ the electron flow which is proportional to a present electric field $\mathbf{E}$:

$$\mathbf{F} = n\mu\mathbf{E} \tag{4.42}$$

Let us focus first on the diffusion part of the equation in 2D[6]:

$$\left.\frac{\partial n}{\partial t}\right|_{\text{diff}} = D\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) n \tag{4.43}$$

Using symmetric differences, we get:

$$\left.\frac{\partial n^{m,n}}{\partial t}\right|_{\text{diff}} = \frac{D}{du^2}\left(n^{m+1,n} + n^{m-1,n} + n^{m,n+1} + n^{m,n-1} - 4n^{m,n}\right) \tag{4.44}$$

Let us focus now on the drift part of the equation:

$$\left.\frac{\partial n^{m,n}}{\partial t}\right|_{\text{drift}} = -\nabla \cdot \mathbf{F}, \tag{4.45}$$

We will now perform a scheme like in [8], which uses symmetric differences in space:

$$\left.\frac{\partial n^{m,n}}{\partial t}\right|_{\text{drift}} = -\frac{1}{du}\left(\frac{F_x^{m+1,n}(q) - F_x^{m-1,n}(q)}{2} + \frac{F_y^{m,n+1}(q) - F_y^{m,n-1}(q)}{2}\right)$$

This gives for the rate of change of n:

$$\begin{aligned}\frac{\partial n^{m,n}}{\partial t} =& \frac{D}{du^2}\left(n^{m+1,n} + n^{m-1,n} + n^{m,n+1} + n^{m,n-1} - 4n^{m,n}\right) \\ &- \frac{1}{du}\left(\frac{F_x^{m+1,n}(q) - F_x^{m-1,n}(q)}{2} + \frac{F_y^{m,n+1}(q) - F_y^{m,n-1}(q)}{2}\right)\end{aligned} \tag{4.46}$$

This yields the following update equations:

$$\begin{aligned}n'^{m,n} =& n^{m,n} + \frac{Ddt}{du^2}\left(n^{m+1,n} + n^{m-1,n} + n^{m,n+1} + n^{m,n-1} - 4n^{m,n}\right) \\ &- \frac{dt}{du}\left(\frac{F_x^{m+1,n}(q) - F_x^{m-1,n}(q)}{2} + \frac{F_y^{m,n+1}(q) - F_y^{m,n-1}(q)}{2}\right)\end{aligned}$$

By carefully choosing the time step for this update equation as

$$dt = \frac{du^2}{4D} = \frac{edu^2}{4kT\mu}, \tag{4.47}$$

[6] The 3D discussion is exactly the same and is left as an exercise to the reader.

the update equation gets considerably simplified:

$$n'^{m,n} = \frac{1}{4}\left(n^{m+1,n} + n^{m-1,n} + n^{m,n+1} + n^{m,n-1}\right) - \frac{edu}{4kT\mu}\left(\frac{F_x^{m+1,n}(q) - F_x^{m-1,n}(q)}{2} + \frac{F_y^{m,n+1}(q) - F_y^{m,n-1}(q)}{2}\right)$$

In the case of a uniform electric field, one gets the typical Lax-Friedrich scheme:

$$n'^{m,n} = \frac{1}{4}\left(n^{m+1,n} + n^{m-1,n} + n^{m,n+1} + n^{m,n-1}\right) - \frac{edu\,|E|}{4kT}\left(\frac{n^{m+1,n}(q) - n^{m-1,n}(q)}{2} + \frac{n^{m,n+1}(q) - n^{m,n-1}(q)}{2}\right)$$

A von Neumann-stability analysis [8] then yields a stable update equation if:

$$s_c = \frac{e\,|E|}{4kT} du < \frac{1}{\sqrt{2}} \tag{4.48}$$

Where we — just like for the FDTD update equations — defined a courant number s_c, this time for the diffusion equation. As for the FDTD updates, in general, s_c must be smaller than $1/\sqrt{D}$, with D the dimension of the simulation.

However, in our case, the space-charge field S is not uniform and hence, the Courant number is not well-defined. However, for most realistic space-charge fields present in the material, which in for example $LiNbO_3$ is limited to maximally 100kV/cm [9], stability on the diffusion is trivially obtained.

4.2.3 Space Charge Electric Field

The diffusion of the electrons through the material depends on a present electric field $\mathbf{E}$ through (4.42). In the case of a photorefractive crystal, the only electric field present is the so-called *space charge field* $\mathbf{S}$, which is induced by the charge distribution ρ in the material[7]:

$$\rho = e(N_D^+ - n - N_c) \tag{4.49}$$

where N_c is the compensating charge, defined as the uniformly distributed charge necessary to make the whole crystal charge neutral:

$$N_c = N_D^+(t=0) - n(t=0) \tag{4.50}$$

[7] Notice the interplay: the diffusion and hence the resulting charge density depends on the space charge field and the space charge field in turn depends on the charge density. Both effects enforce each other.

From Maxwell's static equations in a dielectric, we get

$$\nabla \cdot \mathbf{S} = \frac{\rho}{\epsilon_s} \tag{4.51}$$

$$\nabla \times \mathbf{S} = \mathbf{0} \tag{4.52}$$

Here, we defined ϵ_s to be the *static* permittivity of the photorefractive material, which usually is vastly different than the permittivity at optical wavelengths. Moreover, we also assumed that S varies slowly enough to allow the second equation to equal zero.

The charge density ρ is directly related to the absorbed energy density $d\mathcal{E}_{\text{abs}}$, which we solved in (4.27) on the corners of the grid cell, hence (4.51) also needs to be solved on the corners of the grid cell:

$$\begin{aligned}\frac{\rho[m,n,p]}{\epsilon_s} &= S_x[m,n,p] - S_x[m-1,n,p] + S_y[m,n,p] - S_y[m,n-1,p] \\ &\quad + S_z[m,n,p] - S_z[m,n,p-1]\end{aligned} \tag{4.53}$$

Similarly, $\mathbf{S}$ is an E-type field located on the edges of the grid cell and the application of the curl (4.52) needs to be solved on the faces of the grid cell (H-component locations):

$$0 = S_z[m,n+1,p] - S_z[m,n,p] - S_y[m,n+1,p] + S_y[m,n,p] \tag{4.54}$$

$$0 = S_x[m,n,p+1] - S_x[m,n,p] - S_z[m+1,n,p] + S_z[m,n,p] \tag{4.55}$$

$$0 = S_y[m+1,n,p] - S_y[m,n,p] - S_x[m,n+1,p] + S_x[m,n,p] \tag{4.56}$$

Taking all the equations together for each grid point gives us $4MN$ equations for $3MN$ unknowns, which is an overdetermined linear system.

$$A\mathbf{x} = \mathbf{b} \tag{4.57}$$

with $\mathbf{x}$ the vector of unknowns:

$$\mathbf{x} = \begin{pmatrix} S_x[1,1,1] \\ \vdots \\ S_x[M,N,P] \\ S_y[1,1,1] \\ \vdots \\ S_z[M,N,P] \end{pmatrix} \tag{4.58}$$

and **b** the targets:

$$\mathbf{b} = \begin{pmatrix} \rho[0,0,0]/\epsilon_s \\ \vdots \\ \rho[M,N,P]/\epsilon_s \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.59}$$

and A the sparse matrix containing the coefficients of (4.53) and (4.56). Although this system overdetermined, it turns out a solution to this linear system of equations can still be found by calculating the left pseudo-inverse of A:

$$A^+ = (A^T A)^{-1} A^T \tag{4.60}$$

In this way, the solution to this system can be found as:

$$\mathbf{x}^+ = A^+ \mathbf{b} \tag{4.61}$$

Solving any system by directly inverting a matrix is of course never a good idea because of how expensive the operation of inverting is. However, one could argue that A does in fact not change during the propagation of the light and hence the inversion of the matrix only has to happen once. While this might be more efficient for small grids, inverting A is generally speaking still not the right choice. This is because the inverse of a sparse matrix is almost never sparse itself, which would introduce an expensive dense matrix multiplication after the inversion of the matrix. It is therefore recommended to use a sparse solver to solve for x numerically, especially when the grid size grows. However, even when using a numerical solver for the sparse system $Ax = b$, calculating the space charge field is still a serious bottleneck of a photorefractive simulation as it requires us to *recalculate* the whole space charge electric field from scratch every diffusion time step.

A solution to this problem is to use the biconjugate gradient method [10], which actually solves for the pseudo-inverse, i.e. the system

$$A^T A x = A^T b \tag{4.62}$$

Solving this system numerically is on its own already more efficient as $A^T A$ is an orthogonal matrix. However, in addition to that, the biconjugate gradient method also allows to initialize the system with an estimate of x for which we can of course use the value of x at the previous diffusion time step. This way of solving for the space charge field works at least 10 times faster than a traditional solver solving $Ax = b$.

4.2.4 Electro-optic effect

So far, we have figured out how the light creates a space charge electric field $\mathbf{S}$ by exciting electrons that diffuse to the crystal and get trapped elsewhere. The only piece of the puzzle that is still missing is the relation between this space charge electric field $\mathbf{S}$ and the optical properties of the material.

Generally speaking, the electro-optic effect is described as a dependence of the *impermeability* tensor $\eta = \mu_r \epsilon_r^{-1}$ of the material on a present electric field $\mathbf{E}$. When this electric field is small — as is the case for the space charge electric field $\mathbf{S}$ — we can apply a first-order series expansion:

$$\eta_{ij}(\mathbf{S}) = \eta_{ij}(0) + r_{ijk} S_k. \tag{4.63}$$

This first-order dependency on the electric field is called the Pockels effect. In the case of non-magnetic materials ($\mu_r = 1$), this equation can be rewritten as:

$$(\epsilon_r^{-1})_{ij}(\mathbf{S}) = (\epsilon_r^{-1})_{ij}(0) + r_{ijk} S_k. \tag{4.64}$$

This equation is used when implementing the Pockels effect in simulation. Note that often when talking about Pockels coefficients, the Kleinman convention [11, 12] is used. The Kleinman convention is a notational convention which uses the fact that, for reciprocal materials, we must have that $r_{ijk} = r_{jki}$ and thus only 18 coefficients need to be known instead of 27. The Kleinman-convention now defines the Pockels coefficients in a 6×3 matrix as follows:

$$\begin{pmatrix} r_{11} = r_{111} & r_{12} = r_{112} & r_{13} = r_{113} \\ r_{21} = r_{221} & r_{22} = r_{222} & r_{23} = r_{223} \\ r_{31} = r_{331} & r_{32} = r_{332} & r_{33} = r_{333} \\ r_{41} = r_{231} & r_{42} = r_{232} & r_{43} = r_{233} \\ r_{51} = r_{311} & r_{52} = r_{312} & r_{53} = r_{313} \\ r_{61} = r_{121} & r_{62} = r_{122} & r_{63} = r_{123} \end{pmatrix} \tag{4.65}$$

Note however that almost no material has 18 independent Pockels coefficients. This is because the Pockels tensor is related to the crystal deformations resulting from a present electric field. Hence, the symmetry elements of the Pockels tensor should reflect the symmetry of the crystal, resulting in the fact that most Pockels coefficients will be zero for most crystalline materials.

Furthermore, the r_{ij} parameters are usually very small: they are expressed in picometer per volt. Typical Lithium Niobate has maximal values for r_{ij} around 30pm/V [13], while the best photorefractive crystals have values of around 100pm/V. Usually values for the space charge field are limited by the breakdown voltage, which is (up to an order of magnitude) about 100kV/cm [9]; therefore, the maximal refractive index change is limited by the value of $r_{ij} S_j$, which usually will not be higher than 10^{-5}.

4.2.5 Lithium Niobate

One photorefractive material that is often used is Lithium Niobate. General photorefractive parameters can be found in several reference works, like [14, 15]. The values used for the simulations in this thesis are taken directly from these works and are summarized in Table 4.1. Note that $LiNbO_3$ has a trigonal (3mm) crystal structure and thus only needs 4 Pockels coefficients, r_{22}, r_{13}, r_{33}, r_{42}, ordered in the following way:

$$\begin{pmatrix} 0 & -r_{22} & r_{13} \\ 0 & r_{22} & r_{13} \\ 0 & 0 & r_{33} \\ 0 & r_{51} & 0 \\ r_{51} & 0 & 0 \\ -r_{22} & 0 & 0 \end{pmatrix} \tag{4.66}$$

	Parameter	Value	Unit
s	Photo-ionization cross section	0.0025	m^2/J
β	Thermal excitation rate (300K)	1.0	s^{-1}
γ	Recombination rate	10^{-15}	m^3/s
μ	Mobility	0.0015	m^2/Vs
N_D	Donor density	$6.6 \cdot 10^{24}$	m^{-3}
N_D^+	Initial excited donor density	$3.3 \cdot 10^{24}$	m^{-3}
n	Initial free electron density	$1 \cdot 10^{17}$	m^{-3}
α	Absorption coefficient	20	m^{-1}
ϵ	Static relative permittivity	32	1
ϵ	Relative permittivity @ 1500 nm	4.9 - 4.6	1
$\mathbf{S}$	Space Charge Electric Field	$< 10^5$	$\mathrm{V/m}$
r_{22}	Electro optic coefficient	7	$\mathrm{pm/V}$
r_{13}	Electro optic coefficient	10	$\mathrm{pm/V}$
r_{33}	Electro optic coefficient	32	$\mathrm{pm/V}$
r_{42}	Electro optic coefficient	32	$\mathrm{pm/V}$

Table 4.1: Typical photorefractive parameters for $LiNbO_3$. [13–15]

4.2.6 FDTD update equations for the electric field

Writing the perturbation on the inverse permittivity $r_{ijk}S_k$ in matrix form gives (after applying the Kleinman convention):

$$r_{ijk}S_k = \begin{pmatrix} r_{11}S_x + r_{12}S_y + r_{13}S_z & r_{61}S_x + r_{62}S_y + r_{63}S_z & r_{51}S_x + r_{52}S_y + r_{53}S_z \\ r_{61}S_x + r_{62}S_y + r_{63}S_z & r_{21}S_x + r_{22}S_y + r_{23}S_z & r_{41}S_x + r_{42}S_y + r_{43}S_z \\ r_{51}S_x + r_{52}S_y + r_{53}S_z & r_{41}S_x + r_{42}S_y + r_{43}S_z & r_{31}S_x + r_{32}S_y + r_{33}S_z \end{pmatrix}$$

Note that $\mathbf{S}$ is an E-type field, i.e. its components are located on the edges of the Yee cell. This is a problem, as the above equation clearly shows that components on *different* edges of the cell get mixed. Therefore, each S_k in $r_{ijk}S_k$ needs to be interpolated to the *corner* of the grid cell *first* before they can be added together. This has the added result that ϵ_r^{-1} also lives on the corner of the grid cell. This introduces another problem of its own, because in the update equation (4.15), ϵ_r^{-1} multiplies the E-type field Φ_H, which itself is again defined on the edges of the cell.

The naive solution to this is to interpolate ϵ_r^{-1} back onto the edges after applying the perturbation. However, this operation is not well defined as it is not clear where to place the non-diagonal components of ϵ_r^{-1}. It turns out that this is a difficult problem to solve and the only way to solve it is to adapt the update equation [16]:

$$\begin{aligned}
\tilde{E}_x \text{ += } & \left[\epsilon_r^{-1} \begin{pmatrix} 0 \\ (\Phi_{\tilde{\mathbf{H}}})_y^{\{c\}} \\ (\Phi_{\tilde{\mathbf{H}}})_z^{\{c\}} \end{pmatrix} \right]_x^{\{E_x\}} + (\epsilon_r^{-1})^{\{E_x\}} \begin{pmatrix} (\Phi_{\tilde{\mathbf{H}}})_x \\ 0 \\ 0 \end{pmatrix} \\
\tilde{E}_y \text{ += } & \left[\epsilon_r^{-1} \begin{pmatrix} (\Phi_{\tilde{\mathbf{H}}})_x^{\{c\}} \\ 0 \\ (\Phi_{\tilde{\mathbf{H}}})_z^{\{c\}} \end{pmatrix} \right]_y^{\{E_y\}} + (\epsilon_r^{-1})^{\{E_y\}} \begin{pmatrix} 0 \\ (\Phi_{\tilde{\mathbf{H}}})_y \\ 0 \end{pmatrix} \\
\tilde{E}_z \text{ += } & \left[\epsilon_r^{-1} \begin{pmatrix} (\Phi_{\tilde{\mathbf{H}}})_x^{\{c\}} \\ (\Phi_{\tilde{\mathbf{H}}})_y^{\{c\}} \\ 0 \end{pmatrix} \right]_z^{\{E_z\}} + (\epsilon_r^{-1})^{\{E_z\}} \begin{pmatrix} 0 \\ 0 \\ (\Phi_{\tilde{\mathbf{H}}})_z \end{pmatrix},
\end{aligned} \tag{4.67}$$

where $(\cdot)^c$ is defined as an interpolation of a field component to the corner of the grid cell and $(\cdot)^{E_i}$ is defined as an interpolation of a field component to the E_i-edge of the grid cell.

Obviously, this is a very expensive operation and should be avoided as much as possible. One way of dealing with this is to only use this update equation *inside* the photorefractive crystal, while using the normal update equation everywhere else in the FDTD grid.

4.2.7 Bringing it all together

So far, we have discussed *three* different mechanisms that are important for simulating a photorefractive system, each of which operates at its own timescale. Bringing them together in a meaningful and efficient way is not easy. The general principle is laid out in the flowchart in Fig. 4.3.

The optical propagation is performed by an FDTD simulation, which is characterized by the FDTD time step, which we derived in 4.1.5 to be around $dt_{\text{fdtd}} \approx 0.1\,\text{fs}$. Depending on the size of the grid and the symbol rate of the signal, the FDTD simulation is run for a few thousand time steps. As a general rule we assume it takes about 1000 FDTD time steps to simulate the propagation of a single pulse through the structure, hence after the FDTD simulation about $100fs$ has passed.

However, the diffusion in the crystal happens at a much slower pace. Using (4.47) we can find that for a typical photorefractive material like $LiNbO_3$, with a mobility $\mu = 0.0015\,\text{m}^2/\text{Vs}$, the diffusion time step is about $dt_{\text{diff}} \approx 15\,\text{ps}$ — about 5 orders of magnitude *larger* than the time step of the FDTD simulation. During this characteristic time for the diffusion the refractive index of the material can of course be considered constant. To save time, we arbitrarily multiply the absorption profile with a factor 100, which would physically be equivalent to sending the same signal 100 times through the crystal.

The absorption profile can then be converted into free carriers through (4.39). These are then free to diffuse through the crystal with the mentioned time step. Typically we will update the space charge field about every 100 diffusion steps and repeat this process 1000 times.

This means that the refractive index is updated every 10^{-6} s before the process is started over again. It is however very often the case that after 10^{-6} s the induced refractive index profile has still not changed enough to have a measurable influence on the propagation of the light, which allows us to recycle the previous absorption profile without having to redo the FDTD simulation.

Now that we have obtained a method for simulating the photorefractive effect, we will proceed to apply it on some examples.

4.2.8 A note on stability

Although for some parts of the photorefractive simulation, we briefly mentioned stability contraints, such as for the FDTD update equations and for the diffustion equations, for other parts — like for example the calculation of the space charge field — the stability of the simulation was not considered.

Moreover, even if all *subsystems* of the photorefractive simulation were proven to be stable on their own, it's not a given that their combination as laid out in this chapter still is.

That said, the simulations performed in the rest of this chapter seemed for sure to be stable, as some simulations of were run that took more than a week to finish.

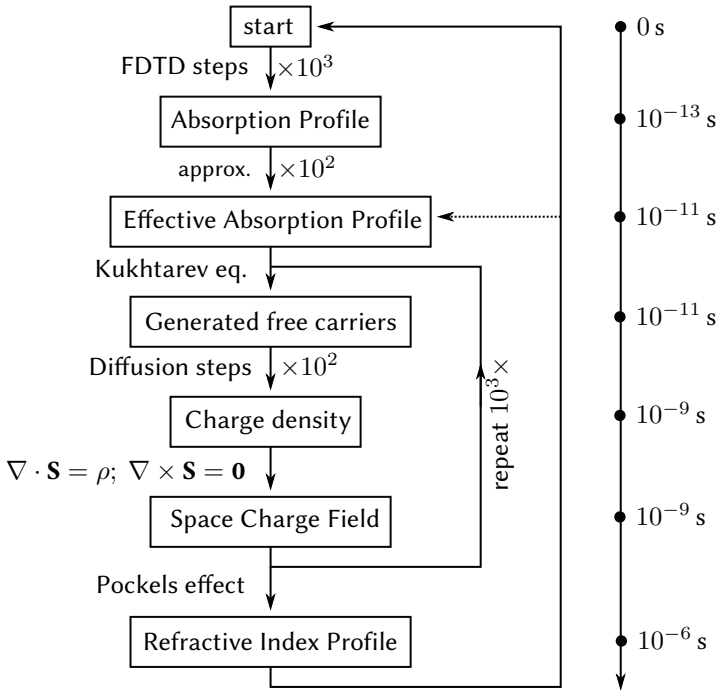


Figure 4.3: A flowchart of a typical photorefractive FDTD simulation together with typical times in the physical process. Each iterative operation is accompanied by a typical number of iterations.

4.3 Holographic storage in photorefractive crystals

Soon after their discovery, photorefractive crystals were hyped to be the *next-generation* storage devices based on holographic patterns written into those crystals [17, 18]. The idea was simple: make a spatially modulated laser beam interfere inside the crystal with a reference beam. The interference between these two beams creates a hologram inside the crystal. The hologram can then simply be read out by illuminating the crystal with the reference beam, as it will be refracted on the hologram.

4.3.1 Beam coupling

The easiest way to understand why the spatially modulated beam can be read out by the reference beam is by looking at a simple interference of two sinusoidal plane waves, which will create a sinusoidal interference pattern in itself. As discussed before, the electrons will be freed at the location where the light intensity is highest, i.e. the locations of constructive interference. They will then diffuse to the dark locations in the crystal: the locations of destructive interference. This results into a space charge electric field $\mathbf{S}$ which will have a maximal value right *inbetween* a maximum and a minimum of the light intensity. This means that the refractive index change — the grating in the crystal written by the two beams interfering — will be shifted by a quarter period in space, which turns out to be the optimal configuration for energy exchange between the two beams.

The process described above is called *beam coupling*, and is the underlying principle of how holographic storage works. It would thus be a good indicator for the simulator described above if the said effect can accurately be simulated. This is what is illustrated in Fig. 4.4: two beams entering the photorefractive material perpendicular to each other (respectively from the north and the west) create an interference pattern oriented along a $45°$ angle. In the beginning, the beams propagate nicely through each other and almost unchanged beam profiles, shown in blue on the figure, are detected at the east and south detector. After a while, the induced grating becomes strong enough to have an effect on the beams. When now one of the sources is turned off, the grating present in the photorefractive material will refract the other beam. This is shown on the figure by the green beam profiles, which are the result of the transmission of the west beam when the beam coming from the north was turned off. The beam is reflected towards the south instead of transmitted towards the east: the induced grating *couples* the east beam towards the south.

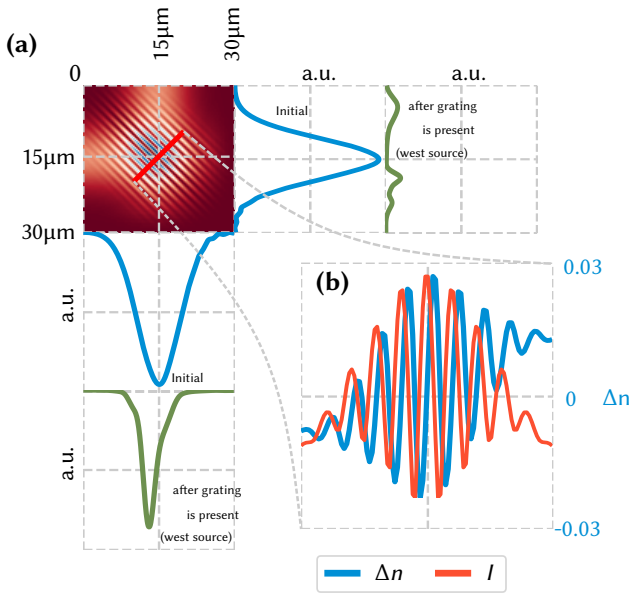


Figure 4.4: (a) Beam coupling can be induced in a photorefractive crystal by letting two perpendicular beams interfere. At first, the beams propagate through each other, resulting in the blue curves on the figure. However, the resulting interference pattern creates an index contrast that after a few seconds of illumination becomes big enough to have an influence on the light. When now one of the sources is turned off, the light will be refracted on the induced grating in the material, as can be seen in the green beam profile (where the north source was turned off). (b) The induced index variation n and the intensity profile I of the light are shifted by a quarter period: perfect for energy exchange between the beams. The Pockels effect was exaggerated by a factor 1000 to account for the small crystal size in simulation.

4.3.2 Holographic storage

One could in principle use multiple beams to encode information in the photorefractive crystal. A way to do this is to have each beam encode a single bit of information. However, it is clear to see that this would be a very impractical approach, as aligning multiple beams at exactly the right positions in the crystal is a very hard task.

In reality, a different approach is used: to store information in the crystal, one returns to the two-beam approach. However, this time, in order to encode more information into the crystal, one beam — the object beam — is spatially modulated by a Spatial Light Modulator (SLM). After passing through a Fourier lens the spatially modulated object beam is Fourier transformed and focused in the center of the crystal where it interferes with a Gaussian reference beam, as illustrated in Fig. 4.5.

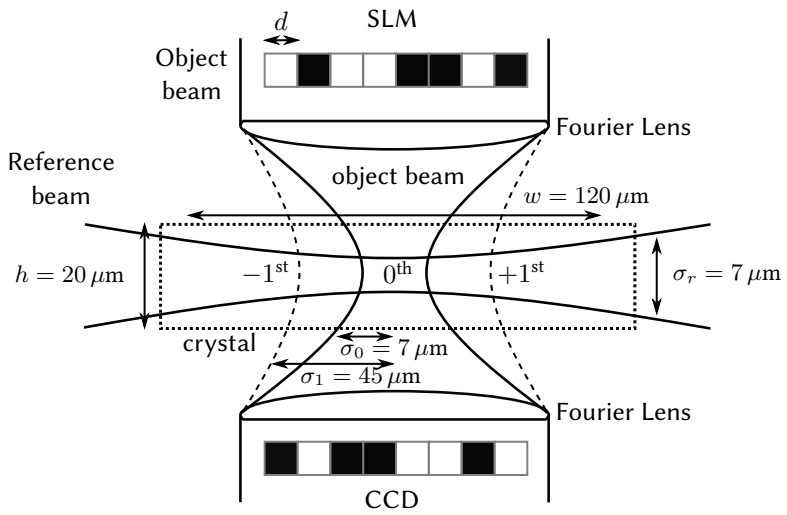


Figure 4.5: Simulated holographic setup

This process was simulated for an SLM with pixel pitch $d = 15\,\mu$m at an optical wavelength of $\lambda = 633\,$nm. The light from the SLM interferes with a Gaussian reference beam with beam waist $\sigma_r = 7\,\mu$m. We want the focused object beam to have the same beam waist $\sigma_0 = \sigma_r$ as the SLM beam. The necessary focal length of the Fourier lens to obtain such a beam waist can be obtained from the diffraction limit:

$$\sigma_0 = 1.22\frac{\lambda f}{Nd} \Rightarrow f \approx 1\,\text{mm}, \tag{4.68}$$

where we used $N = 8$ as the number of pixels in the SLM.

Naively, one could assume that the necessary crystal width for such a beam waist would thus be around $2\sigma_0 \approx 14\,\mu$m. However, the diffraction limit for

$N = 8$ only describes the width of the 0^{th}-order Fourier peak. To have enough resolving power to see each individual pixel of the SLM, higher-order Fourier peaks need to be included as well. A good estimation for the necessary width to have enough resolving power is to set $N = 1$, which will give us the crystal width necessary to resolve a single pixel of the SLM:

$$\sigma_1 = 1.22\frac{\lambda f}{d} \approx 45\,\mu\text{m}, \tag{4.69}$$

Since the pixel pitch defines the spatial frequency that occurs the most in the SLM, this beam width defines the 1^{st} order Fourier peak. The absolute minimum width for the crystal necessary to resolve a single pixel and hence to retain enough information to read it out without error would thus be around $w = 2\sigma_1 \approx 90\,\mu\text{m}$. To be sure we will use a bit more: $w \approx 3\sigma_1 \approx 120\,\mu\text{m}$. Since there is no information encoded into the reference beam, the height of the crystal can just be $h \approx 3\sigma_r \approx 20\,\mu\text{m}$.

We use the above parameters in simulation and see in Fig. 4.6 that this approach works quite well: the binary signal that was encoded into the crystal can be retrieved without much issue.

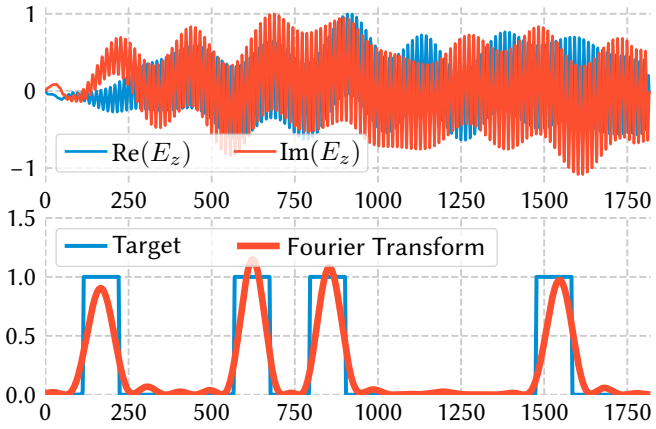


Figure 4.6: Detected hologram before second Fourier lens (top) and after second Fourier lens (bottom).

Note however that one is not bound to use binary data to encode information into the crystal. Binary pixels were used in the above example to better show how well the system can retrieve the data. However, in many applications, such as in 4.4.1, we will only be interested in writing a *random* hologram into the crystal without much interest in the content written into it.

4.4 Artificial neural networks with photorefractive crystals

As soon as the holographic properties of photorefractive crystals were well understood, this property has been exploited for countless of applications. A particular application domain of interest to us, is its use as an artificial neural network [19, 20]. The traditional idea here is to write the weights of the neural network as a hologram into the crystal. As laid out in the previous chapter, once the weights are written into the crystal, one basically has a neural network with ultra-fast inference.

Moreover, it turns out that *optical* training algorithms akin to backpropagation can be derived for such systems. However, due to the inherently slow process of diffusion inside the crystal, the whole idea never really took off, among others because of the very long training times.

A new way of approaching photorefractive crystals for neuromorphic computing is to view them as reservoirs by placing them in a cavity as shown in Fig. 4.7. In that case, we do not need to train the crystal, but can simply randomly initialize it and use the random index distribution to continuously mix the incoming signal, much like we did with the on-chip cavities in the previous chapter. The randomly mixed signal then gets detected by a camera for which each pixel gets weighted by a readout layer. We will explore this possibility in 4.4.1.

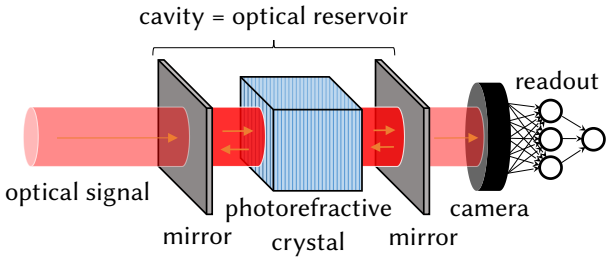


Figure 4.7: A photorefractive reservoir computer.

Additionally, another new and potentially more interesting aspect of these photorefractive structures, is that they might show a form of self-learning, i.e. the ability to reconfigure themselves due to time-dependent training signals. Indeed, the mobile charges freed by the signal will roam around inside the crystal until they reach an equilibrium position. It might be possible to exploit this in a way similar to Hebbian learning [21, 22], which is best characterized by the catchphrase *"neurons that fire together wire together"*. In the case of photorefractive crystals, this could take the form of common patterns and correlations in the training input to be more expressed in the crystal, at the expense of less frequently occurring patterns, which is what we will explore in 4.4.2.

Still, even without self-learning, using a photorefractive crystal for neuromor-

phic computing has other advantages. The biggest one being that a 3D structure allows a much richer interconnection topology that cannot be achieved with the 2D on-chip systems discussed in the previous chapters. Of course, it has the disadvantage of needing a more bulky, free-space setup.

The free-space setup where light propagates through scatterers for computational purposes is also reminiscent of the work in [23]

4.4.1 Reservoir computing with a fixed hologram

Let us first target the randomly initialized photorefractive crystal cavity, as illustrated in Fig. 4.7. We will assume the crystal is pre-initialized with a random hologram and that the intensity of the light is low enough to not have any influence on the refractive index.

This description of the problem is a *classic* reservoir computing problem and resembles the cavities approach of the previous chapter. However, this time the light is leaking out of the cavity through a semi-transparent mirror and the readout is trained on the pixel values detected in the camera behind the mirror. Just like in the previous chapter, we will study the performance of the reservoir under several benchmark tasks [24].

We would like the cavity round trip time to equal the length of a single bit. Using the typical $LiNbO_3$ parameters at a bitrate of $10\,\text{Gbps}$. this would result in a cavity of about $0.7cm$. However, simulating this size with FDTD for a wavelength of $1550\,\text{nm}$ is just plain impossible. Therefore, for computational reasons the cavity is made $1000\times$ smaller than what we would use in practice. This means that we are targeting a $10\,\text{Tbps}$ signal in a $7\,\mu\text{m}$ cavity.

Copy Task

Just like for the on-chip cavity discussed in the previous chapter, we will start of with the copy task, where the performance on replicating the input bit stream with a delay is measured. This obviously is a good benchmark for the memory-capacity of the network. As can be seen in Fig. 4.8, depending on when you sample, the reservoir has a memory of about 2 bits. The BER jumps up and down so much due to the fact that the bit length (in s) of the bit stream sent in is smaller than the round trip propagation time in the cavity[8]. Therefore, there are moments when the information requested is just not present at the camera/detector.

This problem can be circumvented by choosing a camera for which the integration time matches the bitrate of the signal, as can be seen in Fig. 4.9. This approach clearly makes the BER graph much smoother, but the performance is not necessarily better. However, if the sampling position can not be clearly chosen up front, this is a valid approach.

[8] An artifact from the return-to-zero (RZ) encoding used in this case.

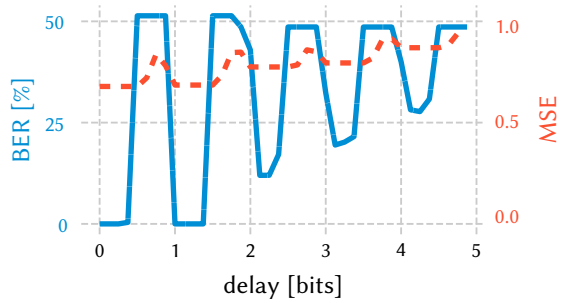


Figure 4.8: BER and MSE for the copy task measured at 8 sample points per bit.

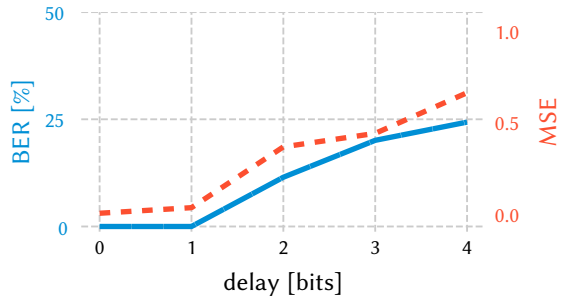


Figure 4.9: BER and MSE for the copy task measured with a camera integration time equal to the bit pulse length of the signal.

XOR Task

A similar approach can be taken for the XOR task, which for 8 samples per bit looks — surprisingly — very similar to the copy task as can be seen in Fig. 4.10.

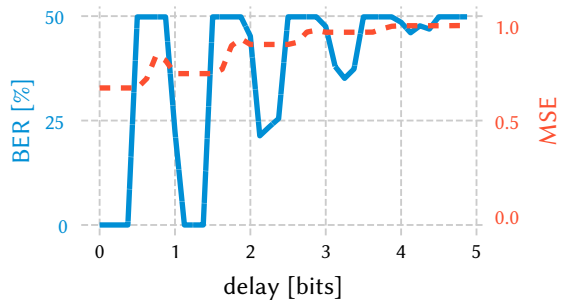


Figure 4.10: BER and MSE for the XOR task at 8 samples per bit.

Increasing the integration time of the camera here is not recommended though, as it clearly degrades the performance at longer delays.

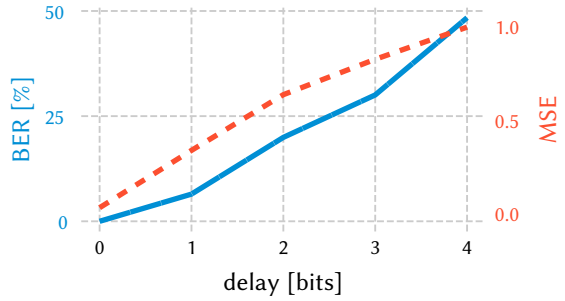


Figure 4.11: BER and MSE for the XOR task measured with a camera integration time equal to the bit pulse length of the signal.

4.4.2 Reservoir computing with a changing hologram

Alternatively one can try to exploit the holographic properties of the photorefractive crystal. We will do this in two steps: we will first *prime* the photorefractive crystal with a random bit stream and then send through the crystal the actual bit stream on which we want to operate. The hope is that this priming will improve the performance on the task.

The priming of the crystal happens by continuously sending a random bit stream through a cavity containing the crystal while a reference beam is active.

The signal beam enters the cavity through a 50/50 mirror and will reflect at the other side of the cavity on a fully reflecting mirror. Moreover, the light is sent into the cavity under a 10° angle, ensuring the light will leave the cavity after a few round trips at the top of the crystal.

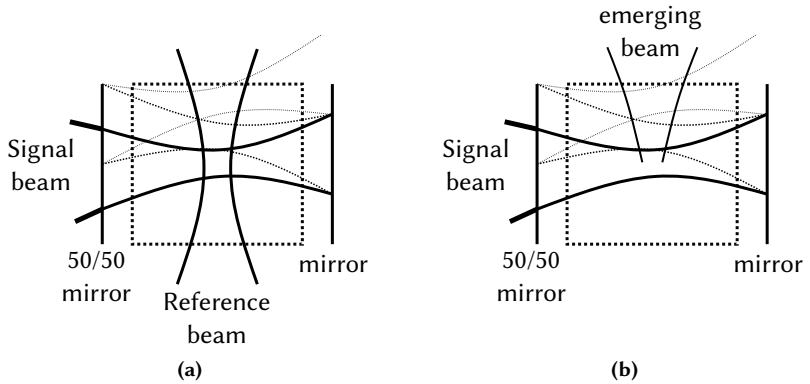


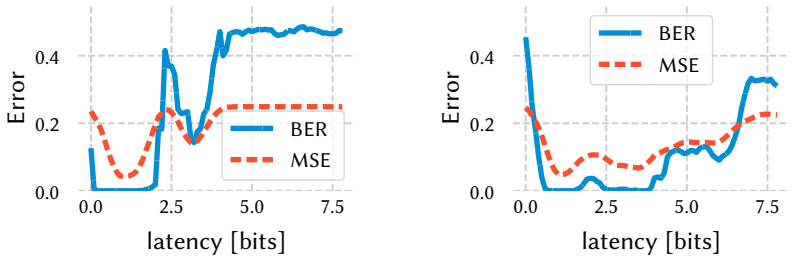
Figure 4.12: (a) The photorefractive crystal is first primed by enabling multiple interference locations with itself *and* with a reference beam. (b) After the priming step different emerging beams form for different bit sequences. These beams take "shortcuts" through the cavity, ensuring proper mixing of the input signal.

The idea is now that, due to the reflections at the mirrors, light is kept inside the crystal and different bit subsequences of the signal beam will interfere with each other *and* with the reference beam, resulting in different types of gratings present in the crystal. When the reference beam is then turned off and the same bit sequence appears again, it will activate that grating and result in (hopefully) a unique emergent behavior due to beam coupling effect.

In simulation, we choose the size of the crystal equal such that the propagation time to traverse the crystal equals a single bit. For the typical parameters of $LiNbO_3$ and a bitrate of $10\,\text{Gbps}$, this would correspond to a cavity size of $1.4\,\text{cm}$. However, just like before, simulating this size with FDTD is just plain impossible. Therefore, we will reduce the cavity size again by a factor 1000 and increase in turn the Pockels effect and the bitrate in simulation by the same factor. Although this yields unphysical parameters, the general idea of the simulation still holds: we are basically trading off grating *length* for grating *strength*.

Copy task

Again, we will go over the copy task first and compare the result on the task before and after priming in Fig. 4.13. We clearly see that priming the crystal with a random bit stream increases the memory. A possible explanation for this behavior can be found in how gratings tend to work: it typically takes a while before the interference effect in the grating which directs the emerging beam out



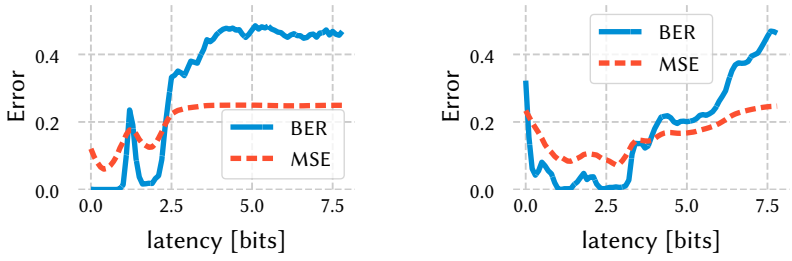
(a) Before priming **(b)** After priming

Figure 4.13: Performance on the copy task before and after priming. Although in both cases the incoming bit stream can be recovered without error, the priming seems to increase the memory of the reservoir.

of the cavity gets established. This means that light gets *stuck* in the grating for a while before it is reflected outward.

XOR task

Just like for the Copy task, priming increases the memory of the reservoir. However, it does not necessarily improve the *quality* of the XOR operation, as the operation can be performed by both the primed reservoir and the non-primed reservoir (Fig. 4.14).



(a) Before priming **(b)** After priming

Figure 4.14: Performance on the XOR task before and after priming. Just like for the copy task, priming seems to improve the memory of the reservoir.

4.5 Conclusion

In this chapter, common descriptions of well-known processes, like the Kukhtarev equations, diffusion equations and the Pockels effect, were combined to create an FDTD Description that enables us to accurately simulate the photorefractive effect.

Some examples, such as beam coupling and holographic storage, were provided to clearly show that the photorefractive effect can be simulated quite well. Morover, these examples laid out the groundwork for the photorefractive neuromorphic computing tasks.

To be able to actually simulate these photorefractive neuromorphic systems in a reasonable amount of time, some approximations were necessary. The absorption in the crystal was increased by a factor 100 and it was assumed this is equivalent to exciting the crystal about 100 times longer. Moreover, the Pockels effect is made stronger by a factor 1000 while the size of the cavity is reduced and the bitrate is increased by the same factor. This is quite a leap of faith and hence the results obtained should just be considered as a *qualitative* indication that such neuromorphic systems, like the reservoir computer and the primed crystal, *could* work in principle.

The simulations we did this way, indicate that a randomly initialized photorefractive crystal could be a viable option as a photonic reservoir computer. Moreover, the self-organizing photorefractive crystals, where the crystal is *primed* with a random bit stream, show promising results as the memory of the system is clearly improved.

References

[1] Kane S Yee et al. *Numerical solution of initial boundary value problems involving Maxwells equations in isotropic media.* IEEE Trans. Antennas Propag, 14(3):302–307, 1966.

[2] Allen Taflove and Susan C Hagness. *Computational electrodynamics: the finite-difference time-domain method.* Artech house, 2005.

[3] John B Schneider. *Understanding the finite-difference time-domain method.* 2010.

[4] Pierre-Alexandre Blanche and Brittany Lynn. *Introduction to the Photorefractive Effect in Polymers.* pages 1–63, 2016.

[5] A Ashkin, G Boyd, and J Dziedzic. *Optically induced refractive index inhomogeneities in $LiNbO_3$ and $LiTaO_3$.* Appl. Phys., pages 5–7, 1966.

[6] FS Chen. *Optically induced change of refractive indices in LiNbO3 and LiTaO3.* Journal of applied physics, 40(8):3389–3396, 1969.

[7] Nikolaiv Kukhtarev et al. *Holographic storage in electrooptic crystals.* Ferroelectrics, 22(1):949–960, 1978.

[8] Luciano Rezzolla. *Numerical Methods for the Solution of Hyperbolic Partial Differential Equations.* 2005.

[9] Pietro Ferraro, Simonetta Grilli, and Paolo De Natale. *Ferroelectric crystals for photonic applications: including nanoscale fabrication and characterization techniques*, volume 91. Springer Science & Business Media, 2013.

[10] Roger Fletcher. *Conjugate gradient methods for indefinite systems.* In Numerical analysis, pages 73–89. Springer, 1976.

[11] DA Kleinman. *Nonlinear dielectric polarization in optical media.* Physical Review, 126(6):1977, 1962.

[12] Woldemar Voigt et al. *Lehrbuch der kristallphysik*, volume 962. Teubner Leipzig, 1928.

[13] Vladimir M Fridkin. *Photoferroelectrics*, volume 9. Springer Science & Business Media, 2012.

[14] O Beyer, D Maxein, K Buse, B Sturman, HT Hsieh, and D Psaltis. *Femtosecond time-resolved absorption processes in lithium niobate crystals.* Optics letters, 30(11):1366–1368, 2005.

[15] NA Gusak and NS Petrov. *On the dependence of the free carrier concentration on light intensity in photorefractive crystals.* Technical Physics, 46(5):635–637, 2001.

[16] Gregory R. Werner and John R. Cary. *A stable FDTD algorithm for non-diagonal, anisotropic dielectrics.* Journal of Computational Physics, 226(1):1085 – 1101, 2007.

[17] D Von der Linde and AM Glass. *Photorefractive effects for reversible holographic storage of information.* Applied physics, 8(2):85–100, 1975.

[18] K Buse, A Adibi, and D Psaltis. *Non-volatile holographic storage in doubly doped lithium niobate crystals.* nature, 393(6686):665, 1998.

[19] Demetri Psaltis, David Brady, and Kelvin Wagner. *Adaptive optical networks using photorefractive crystals.* Applied Optics, 27(9):1752–1759, 1988.

[20] Demetri Psaltis, David Brady, Xiang-Guang Gu, and Steven Lin. *Holography in artificial neural networks.* In Landmark Papers On Photorefractive Nonlinear Optics, pages 541–546. World Scientific, 1995.

[21] Donald Olding Hebb. *Distinctive features of learning in the higher animal.* Brain mechanisms and learning, 37:46, 1961.

[22] Bernard Widrow, Youngsik Kim, and Dookun Park. *The Hebbian-LMS learning algorithm.* ieee ComputatioNal iNtelligeNCe magaziNe, 10(4):37–53, 2015.

[23] Alaa Saade, Francesco Caltagirone, Igor Carron, Laurent Daudet, Angélique Drémeau, Sylvain Gigan, and Florent Krzakala. *Random projections through multiple optical scattering: Approximating kernels at the speed of light.* In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6215–6219. IEEE, 2016.

[24] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. *Machine learning and the physical sciences.* Reviews of Modern Physics, 91(4), Dec 2019.

5 Conclusions

5.1 Summary

The main goal of this work was to examine the viability of light as a carrier for *computation* as opposed to just *communication*. The premise of this work was that, for this to work, *probably* a different computing paradigm is needed. The computing paradigm researched here was *neuromorphic computing*.

This dissertation gave an overview into novel optical neuromorphic computing architectures applied to the telecom field. During this work, roughly three types of architectures were investigated.

We started off with the most traditional on-chip neuromorphic architecture, which depends on photonic nodes interconnected in a feed-forward or recurrent way. To more easily research non-reservoir computing architectures, a new photonic circuit simulator, Photontorch, was created, which focuses specifically on aiding the design process of large photonic circuits through optimization by backpropagation. To our knowledge, the presented framework is the *first* photonic simulator that enables true optimization of large circuits by backpropagation through its *physical parameters*.

Using this simulator, traditional on-chip reservoir architectures were improved upon and extended in simulation by enabling optimization *inside* the recurrent circuit and by *connecting* two non-optimized recurrent circuits through an optimizable intermediate connection. Both designs performed better than the original reservoir after the optimization: an optimization that was only possible by backpropagation through the circuit. However, also other photonic neuromorphic architectures, such as photonic meshes, were explored with state-of-the

art performance in simulation on the pixel-by-pixel MNIST task.

We then proceeded by turning to a slightly more esoteric on-chip photonic reservoir computing design based on photonic cavities. These cavities exhibit a *continuous* mixing in contrast to the more (spatially) discrete mixing inside the nodes of a network. Although initially intended for *photonic crystal* cavities, fabrication constraints pushed towards a fabrication method that is more suitable for mass-integration. Although the power budget can still be improved, these cavities provide a viable platform for high-speed photonic reservoir computing in the telecom field.

Indeed, we have shown both in simulation and in experiment that these photonic cavities are able to perform important tasks such as *header recognition* and *boolean logic* due to their interesting mixing dynamics. We confirmed by experiment particular successes on the nonlinear XOR task and up to 3-bit header recognition.

Finally, we concluded this work with a theoretical viability study on neuromorphic computing with *photorefractive crystals*. For this, a dedicated FDTD-simulator was created which was specifically designed to accurately simulate the careful interplay between photons and electrons in a photorefractive material. By placing these photorefractive materials in a cavity, a similar (but orders of magnitude larger) system as the on-chip cavities can be created. However, this time one has a fine-grained control over the hologram *inside* the cavity, which influences the mixing of the light. Moreover, it turns out that the photorefractive effect can be exploited to *prime* the crystal to better recognize patterns in random bit sequences.

5.2 Perspectives

This work attempted to expand the notion of photonic *reservoir* computing. This was done by either aiding the simulation through Photontorch, which enabled more complex *neuromorphic* architectures, but also by expanding the definition of photonic reservoir computing to include *photonic cavities* or by leaving the integrated photonic platform to study *photorefractive crystals*. All three of the directions explored open up viable paths for future research by themselves.

Photontorch could for example enable experimentation with hierarchic neuromorphic architectures more akin to the current in-software *deep-learning* architectures or aid with training of physical devices by optimizing an in-software surrogate circuit. More generally, the novel approach to optimizing photonic circuits Photontorch has to offer, may have a considerable impact given the growing importance of integrated optics due to the significantly more complex circuit it allows to optimize. Additionally, there is room for implementing more accurate simulation models in the case of dispersion. A Vector Fitting model on top of PyTorch would be very interesting.

With respect to the on-chip cavity reservoirs, the shapes chosen here — although chosen for their known mixing properties — are rather arbitrary: there

is an opportunity to investigate different shapes, sizes and fabrication methods. Also, further research is necessary on how to use several of these cavities in a hierarchy for possibly better results.

Additionally, due to fabrication differences, the cavity response between similarly designed cavities can change considerably. For consistent performance between separately fabricated cavities or even between different environments (temperature, wavelength, ...), a highly dynamical readout is necessary. This, of course, is not *only* true for the cavities. In general, further research into the photonic reservoir readout (optically implemented or electronically) should be a priority.

Moreover, although researching the photorefractive effect was interesting as a case-study, there are an abundance of other on-chip nonlinearities and plastic materials that could be explored as well.

Although the cavity reservoirs (both the on-chip cavity and the photorefractive crystal cavity) yield some interesting results, reported bit error rates remain — even though some of it can be attributed to the simulation technique and measurement equipment — still quite high. One of the key factors that probably needs to be addressed are the high losses in the cavities, especially the on-chip dielectric cavities based on index-contrast, which currently have an insertion loss of about 80%.

Moreover, most tasks discussed here were clearly benchmark tasks. It would therefore be interesting to see how current or future incarnations of the designs discussed in this work will do on more challenging tasks that are also industrially relevant. One of these tasks that stands out in importance is (non-linear) signal equalization tasks that are currently performed by a DSP: any improvement that removes the need for such an expensive electronic component (and accompanying light-electricity-light conversion) would be a welcome development for the telecom sector.