Working Title: Reverse Design of Meta Surface Stacks

Bachelor Thesis

Friedrich-Schiller-Universität Jena
Physikalisch-Astronomische-Fakultät
Institute of Quantum and Nano-Optics

Tim Luca Turan

April 3, 2020

Evaluators

First Assessor:	Prof. Dr. rer. nat. Thomas Pertsch Institute of Quantum and Nano Optics Friedrich-Schiller-Universität Jena
Second Assessor:	M.Sc. Jan Sperrhake Friedrich-Schiller-Universität Jena Institute of Quantum and Nano Optics

1 Introduction

Metamaterials are materials whose physical properties emerge not from the kind of material they are but from their internal geometry. Optical metamaterials are mostly made of repeating sub-wavelength structures. They can be engineered to exhibit properties not found in nature like negative refractive indices [1]. In recent years the focus has shifted to flat 2D metamaterials dubbed metasurfaces. Normal optical devices like lenses or phase plates rely on distances much larger than one wavelength to change the light's properties in contrast metasurfaces have sub-wavelength thickness and enable thin components not possible before [2]. **1.too close to source?** These metasurfaces promise custom components which can be tailored to a wide range of applications just by changing the surface geometry but because of their small size metasurfaces with complex geometry are quite hard to manufacture. Also predicting the optical behavior of metasurfaces cannot be done analytically and involves computationally intensive simulations which slows the design process.

2016 [3] presented an algorithm called SASA to analytically calculate the optical behavior of metasurface stacks if one knows how the surfaces behave individually. This allows for a different approach when designing an optical component. The idea here is to create a target behavior not by using complex geometry but by using simple metasurfaces and stacking them ontop of each other. Designing a component becomes an optimization problem of choosing the right surfaces and then tuning stack parameters like the distance between layers and their rotation angle. This approach is possible because SASA is analytical in nature and many different stacks can be considered very quickly but it also poses some challenges when trying to automate the process. Conventional optimization methods rely on the underlying process being continuous so that if a parameter is tuned slightly it is possible to tell weither that was a step in the right direction. This is a problem because some of the choices involved when building a stack are categorical. For example, choosing the material of the layers or the kind of geometry.

Another recent development has produced a very general and powerful way of dealing with categorical decision problems. Deep Neural Networks have found their way into many different areas and given enough data it should be possible to train a Neural Network to choose the discrete stack parameters to a given optical target. Again SASA being analytical is useful because it can generate a lot of training stacks very quickly. We can now sketch out an algorithm which takes a transmission spectrum as a target and outputs a metasurface stack to reproduce this target:

Preperation:

1. Conventionally simulate a database of metasurfaces with simple geometries
2. Use SASA to generate training stacks based on this database
3. Train a Neural Network on these stacks

Algorithm:

1. Input the transmission spectrum into the Neural Network and receive the design parameters
2. Use a conventional optimization method to tune the continuous parameters
3. Use the database in conjunction with SASA to determine the optical behavior of the current stack.
4. Repeat 2. and 3. until the target accuracy is reached

In section 3 we will first develop the physical theory to understand metasurface stacks and learn about their limitations. Then we will look into the computer science background of Neural Networks and which network architecture to choose for the problem at hand. The Algorithm is described in detail in section 4 and is evaluated in section 6. All the code written for this project is open source and documented here: https://github.com/TimLucaTuran

Contents

1 Introduction **3**

2 Notation **6**

3 Background **7**
- 3.1 Scatter Matrix Calculus . . . 7
- 3.2 SASA . . . 13
- 3.3 Neural Networks . . . 15

4 Algorithm **22**
- 4.1 Implementation . . . 24
- 4.2 Network . . . 25
- 4.3 Database . . . 28
- 4.4 Optimizer . . . 29

5 The Inverse Problem **31**
- 5.1 Forward Model . . . 32

6 Junk **33**

7 Sources **34**

ToDo

P.

1. too close to source? . . . 3
2. fix cursive? . . . 13
3. needs cite . . . 13
4. add forumla to update weights based on gradient . . . 17
5. explain co-adapting . . . 21
6. ask jan how these are made . . . 22
7. add this in Appendix? . . . 31

2 Notation

Symbols	Explanation
$\mathbf{E}$, $\mathbf{B}$, $\mathbf{k}$	Vectors are written bold
E, B, k	Magnitudes of vectors are written non-bold so that $\lvert\mathbf{E}\rvert = E$
$\hat{S}$, $\hat{J}$, $\hat{w}$	Matrices have an operator hat
$\hat{w}^2_{1,2}$, E^{in}_x	Super and sub indices are used to specify certain elements
$(y_i - y_i')^2$, $(n)^2$	All exponents are outside parentheses to differentiate between index and exponent
$\mathbb{1}$	Unity matrix sized accordingly e.g. in the context of 2×2 matrices $\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
∇	Nabla operator $\nabla = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix}$
Δ	Laplace operator $\Delta = \begin{pmatrix} \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \\ \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \\ \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \end{pmatrix}$
$\odot$	Hadamard product $\mathbf{a} \odot \mathbf{b} := \begin{pmatrix} a_1\, b_1 \\ a_2\, b_2 \\ \vdots \end{pmatrix}$

3 Background

3.1 Scatter Matrix Calculus

To implement the desired algorithm we need to be able to calculate the optical behavior of stacked metasurfaces. The mathematical framework we will use is called scatter matrix calculus and this section will give some insight into its physical origin and how to use it. We will start with the Maxwell Equations in matter.

Maxwell Equations

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\frac{\partial}{\partial t}\mathbf{B}(\mathbf{r},t) \tag{3.1}$$

$$\nabla \cdot \mathbf{D}(\mathbf{r},t) = \rho_{\text{ext}}(\mathbf{r},t) \tag{3.2}$$

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \mathbf{j}(\mathbf{r},t) + \frac{\partial}{\partial t}\mathbf{D}(\mathbf{r},t) \tag{3.3}$$

$$\nabla \cdot \mathbf{B}(\mathbf{r},t) = 0 \tag{3.4}$$

The four involved fields are: $\mathbf{E}$ electric field, $\mathbf{B}$ magnetic flux density, $\mathbf{D}$ electric flux density and $\mathbf{H}$ magnetic field and the sources are the external charges ρ_{ext} and the macroscopic currents $\mathbf{j}$. All the material properties are captured by the $\mathbf{D}$ and $\mathbf{H}$ fields which are defined as

$$\begin{aligned} \mathbf{D}(\mathbf{r},t) &= \varepsilon_0 \mathbf{E}(\mathbf{r},t) + \mathbf{P}(\mathbf{r},t) \\ \mathbf{H}(\mathbf{r},t) &= \frac{1}{\mu_0}[\mathbf{B}(\mathbf{r},t) - \mathbf{M}(\mathbf{r},t)] \end{aligned} \tag{3.5}$$

where $\mathbf{P}$ is the dielectric polarization and $\mathbf{M}$ is the magnetic polarization. One can read equation (3.5) in the following way: when the electric field $\mathbf{E}$ interacts with matter it exerts a force on all its charges and displaces them by a small amount. This separation of charges results in a counter field $\mathbf{P}$ and the total field $\mathbf{D}$ is now a superposition of $\mathbf{E}$ and $\mathbf{P}$. This set of equations describes the whole electromagnetic spectrum. However in this work we are only interested in visible (VIS) and near infrared (NIR) light where we can make some simplifications. Generally in optics materials are non-magnetizable so $\mathbf{M} = 0$ and there are no free charges $\rho_{\text{ext}} = 0$. Inserting these assumptions into maxwell's equation gives

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\mu_0 \frac{\partial}{\partial t}\mathbf{H}(\mathbf{r},t) \tag{3.6}$$

$$\varepsilon_0 \nabla \cdot \mathbf{E}(\mathbf{r},t) = -\nabla \cdot \mathbf{P}(\mathbf{r},t) \tag{3.7}$$

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \mathbf{j}(\mathbf{r},t) + \frac{\partial}{\partial t}\mathbf{P}(\mathbf{r},t) + \varepsilon_0 \frac{\partial}{\partial t}\mathbf{E}(\mathbf{r},t) \tag{3.8}$$

$$\nabla \cdot \mathbf{H}(\mathbf{r},t) = 0 \tag{3.9}$$

Light in Vacuum

Now we can derive the wave equation by applying $\nabla\times$ to (3.6) which yields

$$\begin{aligned} & \nabla \times \left[\nabla \times \mathbf{E}\right] = \nabla \times \left[-\mu_0 \frac{\partial}{\partial t}\mathbf{H}\right] \\ \iff \quad & \nabla(\nabla \cdot \mathbf{E}) - \Delta \mathbf{E} = -\mu_0 \frac{\partial}{\partial t} \nabla \times \mathbf{H} \quad \Big| \quad \text{subs. (3.8) and (3.7)} \\ \iff \quad & \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta \mathbf{E} = -\mu_0 \frac{\partial}{\partial t}\mathbf{j} - \mu_0 \frac{\partial^2}{\partial t^2}\mathbf{P} + \frac{1}{\varepsilon_0}\nabla(\nabla \cdot \mathbf{P}) \end{aligned} \tag{3.10}$$

In vacuum ($\mathbf{P} = 0$ and $\mathbf{j} = 0$) the right side of this equation vanishes and we are left with $\frac{1}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta \mathbf{E} = 0$ which is solved by the plane wave $\mathbf{E} = \mathbf{E}_0 e^{i(\mathbf{kr}-\omega t)}$ where $\frac{\omega}{k} = c$. This describes the propagation of light through empty space: a sinusoidal oscillation in time and space along the $\mathbf{k}$ direction where $\mathbf{E}$, $\mathbf{B}$ and $\mathbf{k}$ are all perpendicular to each other.

Light in homogeneous, isotropic materials

The next question we can answer is how light propagates through a homogeneous and isotropic material. For us, the dielectric polarization is some linear function of the electric field so $\mathbf{P}(\mathbf{r},t) = \hat{\chi}(\omega,\mathbf{r})\mathbf{E}(\mathbf{r},t)$. An isotropic material behaves the same for all orientations of $\mathbf{E}$ that means $\hat{\chi}(\omega,\mathbf{r})$ becomes a scalar function $\chi(\omega,\mathbf{r})$. If the material is additionally homogeneous, that is the same everywhere independent of $\mathbf{r}$, then $\boldsymbol{\nabla}\chi(\omega,\mathbf{r}) = 0$. With equation (3.7) this gives us $\boldsymbol{\nabla}\cdot\mathbf{P} = 0$ and the wave equation simplifies to

$$\begin{aligned} \frac{\varepsilon}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E} - \Delta\mathbf{E} &= 0 \\ \text{where} \quad \varepsilon\,\mathbf{E} := (1+\chi)\mathbf{E} &= \mathbf{E} + \mathbf{P} \end{aligned} \tag{3.11}$$

That means light behaves in these materials exactly as it would in vacuum we only have to account for a decreased speed of light $c = c_0/\sqrt{\varepsilon} =: c_0/n$ with the refractive index n. This is equivalent to a decreased wavelength $\lambda =: \lambda_0/n$ or an increased wave number $k = 2\pi/\lambda = n\,k_0$ where c_0, λ_0 and k_0 are the quantities in vacuum. The wave equation is also solved by a complex valued $n := \eta + i\kappa$ which describes the behavior of the exponentially decaying field commonly found in metals

$$\mathbf{E} = \mathbf{E}_0 e^{i(n\mathbf{k}\mathbf{r}-\omega t)} = \mathbf{E}_0 \underbrace{e^{-\kappa\mathbf{k}\mathbf{r}}}_{\text{decay}} \underbrace{e^{i(\eta\mathbf{k}\mathbf{r}-\omega t)}}_{\text{oscillation}} \tag{3.12}$$

Interfaces

The metasurface stacks we want to understand are obviously not one homogeneous material. Rather they contain many interfaces between different materials and we can again use the maxwell equations to predict how light will behave at such an interface.

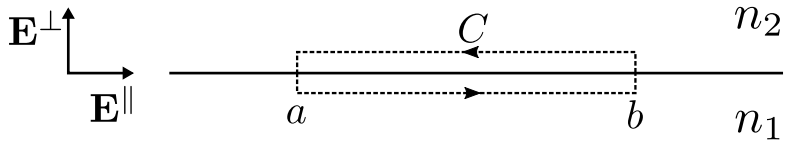


Figure 1: An interface of two materials with different refractive indices n_1 and n_2 and a closed contour C which is tangent to the interface between the points a and b. A field vector within the depicted plane can be decomposed into a tangential and a orthogonal component to the interface, $\mathbf{E}^{\parallel}$ and $\mathbf{E}^{\perp}$, respectively.

Let us consider a closed contour C around an interface as seen in Figure 1 and integrate the first maxwell equation (3.6) over the surface A enclosed by that contour

$$\begin{aligned} \int_A \boldsymbol{\nabla}\times\mathbf{E}(\mathbf{r},t)\,\mathrm{d}\mathbf{A} &= -\mu_0\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\mathrm{d}\mathbf{A} \\ \overset{\text{stokes}}{\Longleftrightarrow} \qquad \int_{C=\partial A} \mathbf{E}(\mathbf{r},t)\mathrm{d}\mathbf{r} &= -\mu_0\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\mathrm{d}\mathbf{A} \end{aligned} \tag{3.13}$$

But now we can bring the contour infinitesimally close to the interface and thus reduce the right hand side of the equation, the total magnetic flux, through the surface, to zero. That leaves us with:

$$\begin{aligned} \int_a^b \mathbf{E}_1\,\mathrm{d}\mathbf{r} \;+ \int_b^a \mathbf{E}_2\,\mathrm{d}\mathbf{r} &= 0 \\ \Longleftrightarrow \qquad \int_a^b \mathbf{E}_1\,\mathrm{d}\mathbf{r} &= \int_a^b \mathbf{E}_2\,\mathrm{d}\mathbf{r} \end{aligned} \tag{3.14}$$

Because a and b were chosen arbitrarily that means that the transverse field components along the path need to be continuous so $\mathbf{E}_1^{\parallel} = \mathbf{E}_2^{\parallel}$. The analog expression $\mathbf{H}_1^{\parallel} = \mathbf{H}_2^{\parallel}$ can be shown by starting with the third maxwell equation (3.8) instead. We can now use these continuity conditions to learn more about the behavior of light at these interfaces.

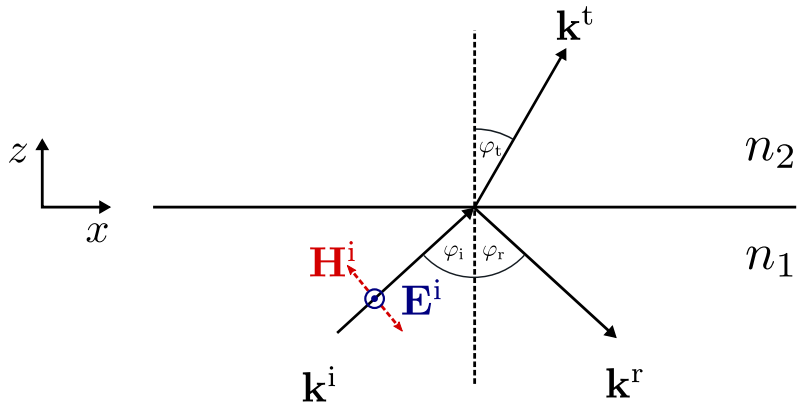


Figure 2: Interface between two materials of different refractive indices n_1 and n_2. The incident light has a wave vector $\mathbf{k}^{\mathrm{i}}$ and is partially transmitted and partially reflected. The fields are shown in transverse electric (TE) polarization where $\mathbf{H}$ is in the x-z plane and $\mathbf{E}$ oscillates in the y direction. In this polarization state the $\mathbf{E}$ field is fully tangential to the interface that means $\mathbf{E} = \mathbf{E}^{\parallel}$ and $\mathbf{E}^{\perp} = 0$. For the $\mathbf{H}$ field only the x component is tangential to the interface and the z component is orthogonal to it, so $\mathbf{H}_x = \mathbf{H}^{\parallel}$ and $\mathbf{H}_z = \mathbf{H}^{\perp}$.

Let us consider the same interface from before and have an incident field $\mathbf{E}^{\mathrm{i}}$ interact with it. From figure 2 we can see:

$$\mathbf{k}^{\mathrm{i}} = k_0 n_1 \begin{pmatrix} -\sin\varphi_{\mathrm{i}} \\ 0 \\ -\cos\varphi_{\mathrm{i}} \end{pmatrix}, \quad \mathbf{k}^{\mathrm{r}} = k_0 n_1 \begin{pmatrix} \sin\varphi_{\mathrm{r}} \\ 0 \\ -\cos\varphi_{\mathrm{r}} \end{pmatrix}, \quad \mathbf{k}^{\mathrm{t}} = k_0 n_2 \begin{pmatrix} \sin\varphi_{\mathrm{t}} \\ 0 \\ \cos\varphi_{\mathrm{t}} \end{pmatrix} \tag{3.15}$$

For the derivation it is useful to decompose the fields into orthogonal polarization components. Here we are going to use the transversal electric (TE) and transversal magnetic (TM) polarizations. The TE component can be seen in figure 2 and in TM polarization $\mathbf{E}$ and $\mathbf{H}$ fields are simply interchanged. If we apply the continuity condition from before to the TE component we get that $\mathbf{E}_1 = \mathbf{E}_2$ at $z = 0$

$$\begin{aligned} & \mathbf{E}^{\mathrm{i}} + \mathbf{E}^{\mathrm{r}} = \mathbf{E}^{\mathrm{t}} \\ \iff \quad & E^{\mathrm{i}} e^{i(k_x^{\mathrm{i}} x)}\, \vec{\mathbf{e}}_y + E^{\mathrm{r}} e^{i(k_x^{\mathrm{r}} x)}\, \vec{\mathbf{e}}_y = E^{\mathrm{t}} e^{i(k_x^{\mathrm{t}} x)}\, \vec{\mathbf{e}}_y \end{aligned} \tag{3.16}$$

Where $E^{\mathrm{i}}, E^{\mathrm{r}}, E^{\mathrm{t}} \in \mathbb{R}$ are the measurable real field magnitudes. This is only possible for all x if

$$E^{\mathrm{i}} + E^{\mathrm{r}} = E^{\mathrm{t}} \quad \text{and} \quad k_x^{\mathrm{i}} = k_x^{\mathrm{r}} = k_x^{\mathrm{t}} \tag{3.17}$$

Two basic laws of optics lie in this relation: The law of reflection

$$k_x^{\mathrm{i}} = k_x^{\mathrm{r}} \quad \Rightarrow \quad k_0 n_1 \sin\varphi_i = k_0 n_1 \sin\varphi_r \quad \Rightarrow \quad \varphi_i = \varphi_r := \varphi_1 \tag{3.18}$$

and the Snells law of refraction

$$k_x^{\mathrm{i}} = k_x^{\mathrm{t}},\ \varphi_t := \varphi_2 \quad \Rightarrow \quad k_0 n_1 \sin\varphi_1 = k_0 n_2 \sin\varphi_2 \quad \Rightarrow \quad n_1 \sin\varphi_1 = n_2 \sin\varphi_2 \tag{3.19}$$

Fresnel Equations

To fully describe an interface we also need to know the transmitted and the reflected field amplitudes, $t := E^t/E^i$ and $r := E^r/E^i$, respectively. Again using the TE polarization shown in figure 2 we see

$$\mathbf{H}^{\mathrm{i}} = H^{\mathrm{i}} \begin{pmatrix} -\cos\varphi_{\mathrm{i}} \\ 0 \\ -\sin\varphi_{\mathrm{i}} \end{pmatrix}, \quad \mathbf{H}^{\mathrm{r}} = H^{\mathrm{r}} \begin{pmatrix} \cos\varphi_{\mathrm{r}} \\ 0 \\ \sin\varphi_{\mathrm{r}} \end{pmatrix}, \quad \mathbf{H}^{\mathrm{t}} = H^{\mathrm{t}} \begin{pmatrix} -\cos\varphi_{\mathrm{t}} \\ 0 \\ \sin\varphi_{\mathrm{t}} \end{pmatrix} \tag{3.20}$$

The H_x component is tangent to the interface so it also needs to be continuous across the boundary

$$\begin{aligned} H_x^{\mathrm{i}} + H_x^{\mathrm{r}} &= H_x^{\mathrm{t}} \\ \iff \qquad H^{\mathrm{i}} \cos\varphi_1 - H^{\mathrm{r}} \cos\varphi_1 &= H^{\mathrm{t}} \cos\varphi_2 \end{aligned} \tag{3.21}$$

If we assume the described plane waves $\mathbf{E} = \mathbf{E}_0 e^{i(n\mathbf{kr}-\omega t)}$ and $\mathbf{H} = \mathbf{H}_0 e^{i(n\mathbf{kr}-\omega t)}$ Maxwell's first equation (3.6) allows us to connect the magnitudes of $\mathbf{H}$ and $\mathbf{E}$ via the refractive index $H \sim n\,E$. This changes (3.21) to

$$n_1\, E^{\mathrm{i}} \cos\varphi_1 - n_1\, E^{\mathrm{r}} \cos\varphi_1 = n_2\, E^{\mathrm{t}} \cos\varphi_2 \tag{3.22}$$

Substituting E^{r} or E^{i} with (3.17) and rearranging we get

$$t = \frac{2n_1 \cos\varphi_1}{n_1 \cos\varphi_1 + n_2 \cos\varphi_2} \quad \text{and} \quad r = \frac{n_1 \cos\varphi_1 - n_2 \cos\varphi_2}{n_1 \cos\varphi_1 + n_2 \cos\varphi_2} \tag{3.23}$$

These are called the Fresnel equations for the TE component. The TM component can be treated analog which yields

$$t = \frac{2n_1 \cos\varphi_1}{n_2 \cos\varphi_1 + n_1 \cos\varphi_2} \quad \text{and} \quad r = \frac{n_2 \cos\varphi_1 - n_1 \cos\varphi_2}{n_2 \cos\varphi_1 + n_1 \cos\varphi_2} \tag{3.24}$$

For perpendicular incident light at $\varphi = 90°$ these should describe the same situation as we can no longer differentiate between TE and TM components and indeed for this angle they are equivalent if we consider that the TE factors describe the electric field and the TM factors describe the magnetic field. The TE factors become

$$t = \frac{2n_1}{n_1 + n_2} \quad \text{and} \quad r = \frac{n_1 - n_2}{n_1 + n_2} \tag{3.25}$$

Stacking

We can take this one step further by including interfaces which have incident light from both sides. Let $\mathbf{E}_{\mathrm{out}}$ be the light coming out of the interface and $\mathbf{E}_{\mathrm{in}}$ be the light going into the interface as seen in figure 3. Using the factors from the Fresnel equations we get

$$E^b_{\mathrm{out}} = E^f_{\mathrm{in}}\, t^f + E^b_{\mathrm{in}}\, r^b \quad \text{and} \quad E^f_{\mathrm{out}} = E^b_{\mathrm{in}}\, t^b + E^f_{\mathrm{in}}\, r^f \tag{3.26}$$

and this can be expressed more concisely using matrices and vectors

$$\begin{pmatrix} E^f_{\mathrm{out}} \\ E^b_{\mathrm{out}} \end{pmatrix} = \underbrace{\begin{pmatrix} t^f & r^b \\ r^f & t^b \end{pmatrix}}_{=:\hat{S}} \begin{pmatrix} E^f_{\mathrm{in}} \\ E^b_{\mathrm{in}} \end{pmatrix} \tag{3.27}$$

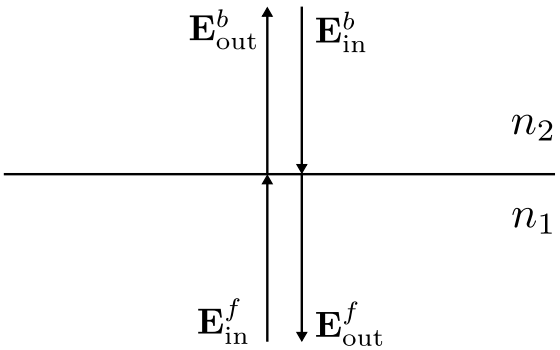


Figure 3: Interface between two materials of different refractive indices n_1 and n_2 were incident light is coming from the front and the back.

We call these matrices mapping field-in to field-out scattering matrices or short S-matrices. They can be used to describe more than just interfaces. As shown in the paragraph Light in materials when light propagates through homogeneous isotropic material just the phase changes by a factor of e^{ik_0nd} where d is the distance traveled and $\mathbf{k}_0$ the wave vector in vacuum. We can express this propagation by allowing complex valued t and r

$$\hat{S}_{n,d} = \begin{pmatrix} e^{ik_0nd} & 0 \\ 0 & e^{ik_0nd} \end{pmatrix} \tag{3.28}$$

and analog the S-matrix for an interface from n_1 to n_2 is

$$\hat{S}_{n_1,n_2} = \begin{pmatrix} \frac{2n_1}{n_1+n_2} & \frac{n_2-n_1}{n_1+n_2} \\ \frac{n_1-n_2}{n_1+n_2} & \frac{2n_1}{n_1+n_2} \end{pmatrix} \tag{3.29}$$

Say we have a stack of different homogeneous isotropic materials. We are now are now able to write down an S-matrix for every part of the stack that is for every interface and every propagation between interfaces as seen in figure 4, but we cannot predict the behavior of the stack as a whole.

The combined S-matrix $\hat{S}$ consisting of $\hat{S}_1$ and $\hat{S}_2$ cannot be obtained by simply using the matrix product. The result would be

$$\hat{S}_1\,\hat{S}_2 = \begin{pmatrix} t_1^f\,t_2^f + r_1^b\,r_2^f & t_1^f\,r_2^b + r_1^b\,t_2^b \\ r_1^f\,t_2^f + t_1^b\,r_2^f & r_1^f\,r_2^b + t_1^b\,t_2^b \end{pmatrix} \neq \begin{pmatrix} t^f & r^b \\ r^f & t^b \end{pmatrix} = \hat{S} \tag{3.30}$$

and this would imply that the transmission from the front t^f increases when the internal reflections of the system r_1^b and r_2^f increase which is the opposite of the behavior we expect. In the 60s Redheffer [4] developed a linear transformation called star product $\star$ which yields the correct combined S-matrix

$$\hat{S}_1 \star \hat{S}_2 := \begin{pmatrix} t_2^f(1-r_1^br_2^f)^{-1}t_1^f & r_2^b + t_2^fr_1^b(1-r_2^fr_1^b)^{-1}t_2^b \\ r_1^f + t_1^br_2^f(1-r_1^br_2^f)^{-1}t_1^f & t_1^b(1-r_2^fr_1^b)^{-1}t_2^b \end{pmatrix} \tag{3.31}$$

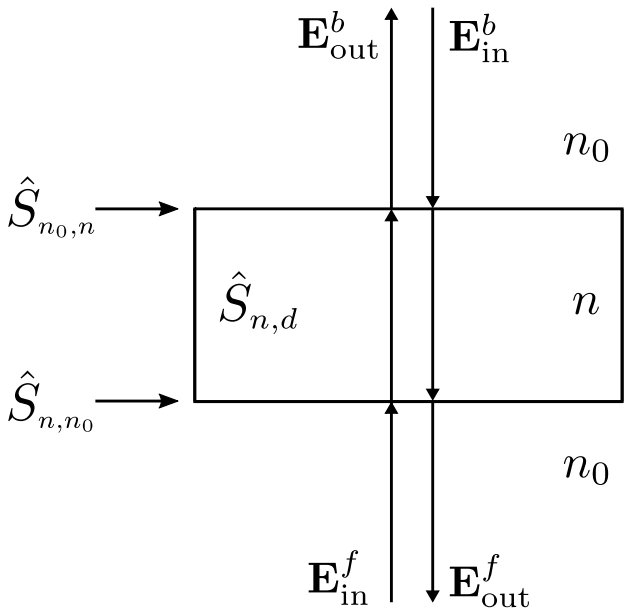


Figure 4: A slab of one homogeneous isotropic material with thickness d in air (n_0). This setup is fully described by two interface S-matrices $\hat{S}_{n_0,n}$, $\hat{S}_{n,n_0}$ and one propagation S-matrices $\hat{S}_{n,d}$.

For this operation to produce physically valid results certain conditions have to be met. We will discuss these in detail in the paragraph Conditions of section 3.2. Applied to the example of figure 4 the star product gives

$$\hat{S} = \hat{S}_{n_0,n} \star \hat{S}_{n,d} \star \hat{S}_{n,n_0} \tag{3.32}$$

Polarization

Up to this point the S-matrix is made of the four complex numbers t^f, r^b, r^f and t^b. That means when a S-matrix is applied to the input vector containing the total incoming light $\begin{pmatrix} E^f_{\text{in}} \\ E^b_{\text{in}} \end{pmatrix}$ E^f_{in} and E^b_{in} are also restricted the complex numbers and can only describe linear polarized light in one direction. The last generalization we want to make is to extend the scatter matrix calculus to all polarizations. As shown in the paragraph Light in materials a planar light wave propagating along the z axis through a homogeneous isotropic material can be described as

$$\mathbf{E} = \begin{pmatrix} E_x e^{i(kz-\omega t+\varphi_x)} \\ E_y e^{i(kz-\omega t+\varphi_y)} \\ 0 \end{pmatrix} = \left(E_x\, e^{i\varphi_x}\, \vec{\mathbf{e_x}} + E_y\, e^{i\varphi_y}\, \vec{\mathbf{e_y}}\right) e^{i(kz-\omega t)} \tag{3.33}$$

the waves polarization is determined by the scaling factors of $\vec{\mathbf{e_x}}$ and $\vec{\mathbf{e_y}}$ and can be expressed as a *Jones Vector* $\mathbf{j} \in \mathbb{C}^2$

$$\mathbf{j} = \frac{1}{\sqrt{E_x^2 + E_y^2}} \begin{pmatrix} E_x \\ E_y\, e^{i\delta} \end{pmatrix} \quad \text{with} \quad \delta := \varphi_y - \varphi_x \tag{3.34}$$

Now all linear operations on the polarization are matrices $\hat{M} \in \mathbb{C}^{2\times 2}$. That means all passive components have a corresponding matrix. Examples for components aligned with the x axis are

$$\begin{aligned} \text{linear polarizer:} \quad & \hat{M} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\ \lambda/4 \text{ plate:} \quad & \hat{M} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} e^{-\frac{i\pi}{4}} \\ \lambda/2 \text{ plate:} \quad & \hat{M} = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \end{aligned} \tag{3.35}$$

We can now generalize the S-matrix calculus simply by allowing Jones matrices in place of the Fresnel coefficients: $t \to \hat{T}$ and $r \to \hat{R}$. The star product of two S-matrices becomes

$$\hat{S}_1 \star \hat{S}_2 = \begin{pmatrix} \hat{T}_1^f & \hat{R}_1^b \\ \hat{R}_1^f & \hat{T}_1^b \end{pmatrix} \star \begin{pmatrix} \hat{T}_2^f & \hat{R}_2^b \\ \hat{R}_2^f & \hat{T}_2^b \end{pmatrix} = \begin{pmatrix} \hat{T}_2^f (\mathbb{1} - \hat{R}_1^b \hat{R}_2^f)^{-1} \hat{T}_1^f & \hat{R}_2^b + \hat{T}_2^f \hat{R}_1^b (\mathbb{1} - \hat{R}_2^f \hat{R}_1^b)^{-1} \hat{T}_2^b \\ \hat{R}_1^f + \hat{T}_1^b \hat{R}_2^f (\mathbb{1} - \hat{R}_1^b \hat{R}_2^f)^{-1} \hat{T}_1^f & \hat{T}_1^b (\mathbb{1} - \hat{R}_2^f \hat{R}_1^b)^{-1} \hat{T}_2^b \end{pmatrix} \tag{3.36}$$

So the generalized S-matrix is a 2×2 block matrix of four Jones matrices and the input vector $\begin{pmatrix} \mathbf{E}^f_{\text{in}} \\ \mathbf{E}^b_{\text{in}} \end{pmatrix}$ is a 2×1 block vector of two Jones vectors.

3.2 SASA

The S-matrix calculus enables us to calculate the optical behavior of a stack if we know the optical behavior of all its layers in the form of their S-matrices. In section 3.1 we derived the S-matrices of propagation through and interfaces between homogeneous isotropic materials. This calculation is based on the Fresnel equations and is therefore analytic. However we cannot calculate the S-matrix of a metasurface analytically. This results in a semi-analytic Stacking Algorithm where the S-matrices of the metasurfaces need to be provided but the remaining S-matrices can be calculated based on geometric parameters.

A core idea of this algorithm is the fact that geometric transformations like rotation and mirroring can be applied directly to Jones matrices. For example let $\hat{M}$ be the Jones matrix of an optical component then the Jones matrix of the rotated component $\hat{M}_\varphi$ is calculated as:

$$\hat{M}_\varphi = \hat{\Theta}(-\varphi)\,\hat{M}\,\hat{\Theta}(\varphi) \quad \text{where} \quad \hat{\Theta}(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \tag{3.37}$$

On this basis we can rotate S-matrices in a similar manner:

$$\hat{S}_\varphi = \begin{pmatrix} \hat{\Theta}(-\varphi)\hat{T}^f\hat{\Theta}(\varphi) & \hat{\Theta}(-\varphi)\hat{R}^b\hat{\Theta}(\varphi) \\ \hat{\Theta}(-\varphi)\hat{R}^f\hat{\Theta}(\varphi) & \hat{\Theta}(-\varphi)\hat{T}^b\hat{\Theta}(\varphi) \end{pmatrix} \tag{3.38}$$

Flipping and mirroring of S-matrices is done analogously. With all mathematical operations well defined we can write down SASA in pseudo code.

Input: Stack = [Layer 0, Layer 1, ...]
Output: Stack S-matrix
S-mats = [ ]
for $i = 0$; $i <$ len(Stack); $i = i + 1$; **do**
  if *Stack*[i] **is** *metasurface* **then**
    add layer $\hat{S}$ to S-mats
  else
    calculate propagation S-matrix $\hat{S}_{n_i,d_i}$
    add $\hat{S}_{n_i,d_i}$ to S-mats
  end
  apply geometric transformations to added S-matrix
  calculate interface S-matrix $\hat{S}_{n_i,n_{i+1}}$
  add $\hat{S}_{n_i,n_{i+1}}$ to S-mats
end
$\hat{S} = \nVdash$
for $i = 0$; $i <$ len(*S-mats*); $i = i + 1$; **do**
  $\hat{S} = \hat{S} \star S$-mats[$i$]
end
return $\hat{S}$

2.fix cursive?

SASA pseudo code: A Layer variable holds all the information of that layer, that is the S-matrix if its a metasurface and the refractive index n and thickness d otherwise.

Conditions

For this algorithm to produce physical valid output some conditions need to be met. Equations that are boundary conditions to the algorithm will be marked bold. It has been shown **3.needs cite** that the S-matrix calculus can only be used when the meta materials in the stack do not interact with each other via the near field. That is, only the zeroth diffraction order of a metasurface can be non-evanescent and higher orders need to have decayed enough when they reach the next metasurface.

First we need to ensure that only the zeroth diffraction order is non evanescent by requiring

$$\lambda > \max(n^f, n^b) \cdot \Lambda \tag{3.39}$$

Additionally the distance between metasurfaces needs to be large enough so that all higher order modes have decayed. It was shown in [3] that if we demand

$$e^{\mathrm{Im}(k_z)d} < e^{-2} \approx 0.14 \quad \text{then} \quad d > \frac{\Lambda}{\pi\sqrt{1 - \frac{\Lambda^2 n^2}{\lambda^2}}} \tag{3.40}$$

The factor e^{-2} was chosen arbitrarily. A smaller value increases the agreement of the S-matrix calculus and more rigorous numerical methods which also take into account the near field. One of these, the Fourier Modal Method will be described in more detail in section 4.4.

Symmetries

We can use geometric transformations to examine the S-matrices of metasurfaces which satisfy certain symmetries. Lets start with mirror symmetry. The Jones matrix $\hat{M}'$ of a mirrored component $\hat{M}$ can be calculated as

$$\hat{M}' = \hat{F}^{-1}\hat{M}\hat{F} \quad \text{where} \quad \hat{F} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.41}$$

If a metasurface has mirror symmetry the S-matrix needs to satisfy $\hat{S} = \hat{S}'$ and that means all the contained Jones matrices need to satisfy the condition too

$$\begin{gathered} \text{Let} \quad \hat{T}^f = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad \text{then} \\ \left(\hat{T}^f\right)' = \hat{F}^{-1}\hat{T}^f\hat{F} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \overset{!}{=} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \hat{T}^f \\ \Rightarrow \qquad \hat{T}^f = \begin{pmatrix} A & 0 \\ 0 & D \end{pmatrix} \end{gathered} \tag{3.42}$$

We can also examine rotational symmetry with the rotation matrix $\hat{\Theta}(\varphi)$. A metasurface of C_4 symmetry, for example square shaped meta particles, maps onto itself if rotated 90°. That means the S-matrix should also remain equal, so $\hat{S}_{\pi/2} \overset{!}{=} \hat{S}$ and

$$\begin{gathered} \hat{T}^f_{\pi/2} = \hat{\Theta}(-\pi/2)\,\hat{T}^f\,\hat{\Theta}(\pi/2) \overset{!}{=} \hat{T}^f, \quad \hat{\Theta}(\pi/2) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \\ \Rightarrow \qquad \hat{T}^f = \begin{pmatrix} A & B \\ -B & A \end{pmatrix} \end{gathered} \tag{3.43}$$

So if a metasurface has both mirror and C_4 symmetry its $\hat{T}^f$ matrix has the form

$$\hat{T}^f = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix} \tag{3.44}$$

That means in this case the star product is commutative and the stack will behave the same for light coming from the top and bottom. And it should be noted that these considerations apply to all Jones matrices and not only to $\hat{T}^f$.

3.3 Neural Networks

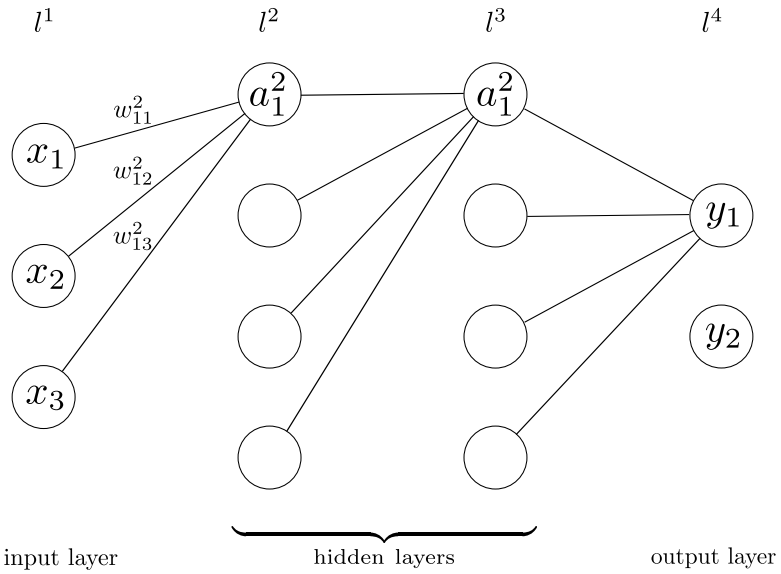


Figure 5: The most simple kind of NN is called densely connected or multilayer perceptron. For clarity only connections to the top most node of each layer are shown.

Artificial Neural Networks (ANN's or short NN's) are a kind of data structure inspired by the biological neurons found in nature. They can be used to find a wide range of input output relations. One classic example is mapping pictures of hand written digits to the actual digits. Rather than explicitly programmed, NN's are trained on a dataset (X, Y) of correct input output pairs.

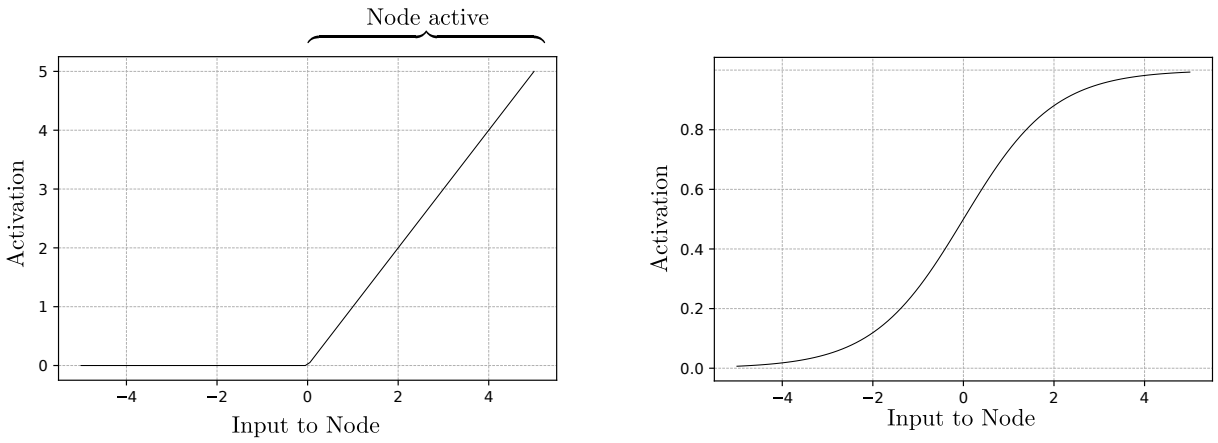
Multilayer Perceptron This kind of classic NN consist of single nodes or neurons which are organized into layers. The terms node and neuron can be used interchangeably. Every node is connected to all the nodes of the previous and the next layer. For this reason the network is called dense or densely connected. Each node holds a value called activation a where the activation to the first layer is the input to the network, here: (x_1, x_2, x_3). The nodes are connected by weights w which specify how much one node should influence the next and every node has a bias b to control at what total input activation the node itself should become active. To calculate the activation of a node one has to multiply all the activations of the previous layer with their respective weights w, add the bias b and finally apply a non-linear activation function σ. To describe this process mathematically we are going to use the usual index notation where superscripts specify the layer and subscripts the node. So a^2_1 is the activation of the first node in the second layer. To characterize a weight two subscripts are needed for the end and beginning of the connection. For the example in figure 5 that means

$$a^2_1 = \sigma\left(\sum_i w^2_{1i}\, x_i + b^2_1\right) \tag{3.45}$$

However it is more convenient to stop considering every node individually and to view the involved quantities as vectors and matrices. So that (3.45) can be written as

$$\mathbf{a}^l = \sigma\underbrace{\left(\hat{\mathbf{w}}^l \mathbf{a}^{l-1} + \mathbf{b}^l\right)}_{:=\,\mathbf{z}^l} \tag{3.46}$$

That means the activation function σ maps the total input to a neuron $\mathbf{z}^l$ to the output/activation of that neuron $\mathbf{a}^l$. Two examples for activation functions can be seen in Figure 6.



(a) rectified linear unit (ReLu)

(b) sigmoid

Figure 6: Two examples of activation functions σ. Especially the ReLu function has a destinct on and off state similar to a biological neurons.

Training

During training the networks output $\mathrm{NN}(\mathbf{x}) := \mathbf{y}'$ is calculated through repeated use of (3.46) and is then compared with the known correct output $\mathbf{y}$ by a cost function $C = C(\mathbf{y}, \mathbf{y}')$. The goal of the training is to minimize this function C. The cost function might simply be the mean squared difference between $\mathbf{y}$ and $\mathbf{y}'$

$$C_{\mathrm{mse}}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{3.47}$$

but there are different cost functions for different kind of outputs. For example a network which predicts continuous values needs a different cost function than one predicting categories. More on this in section 4.2. Now we can quantify how well the NN is performing and are able to use this information to train the network.

As stated before the goal of the training is to minimize the cost function C by changing the weights and biases. That means we are looking for the partial derivatives $\partial C/\partial \hat{w}^l_{j,k}$ and $\partial C/\partial b^l_j$. To find these we need the concept of the *error of a single neuron* δ^l_j

$$\delta^l_j := \frac{\partial C}{\partial z^l_j} \tag{3.48}$$

This property expresses how much the final cost is affected by a change to the input of neuron j in layer l. For the last layer L we can find a simple expression for this property by using the chain rule

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\,\frac{\partial a^L_j}{\partial z^L_j} \overset{(3.46)}{=} \frac{\partial C}{\partial a^L_j}\,\sigma'(z^L_j) \tag{3.49}$$

Remember z^l_j is the input to a neuron and a^l_j is the output. That means δ^L_j is determined by a combination of how much the last output changes the cost function and how much the last input changes the last output. Notice how both these terms are easily accessible. z^l_j was already calculated on the forward pass and the derivatives of the cost and activation functions can be found analytically. By using the gradient operator $\boldsymbol{\nabla}$ and the Hadamard product $\odot$ where

$$\mathbf{a} \odot \mathbf{b} := \begin{pmatrix} a_1\, b_1 \\ a_2\, b_2 \\ \vdots \end{pmatrix} \tag{3.50}$$

we can return to the more convenient vector notation

$$\boldsymbol{\delta}^L = \boldsymbol{\nabla}_a C \odot \sigma'(\mathbf{z}^L) \tag{3.51}$$

Now we know $\boldsymbol{\delta}$ for the last layer but not for the rest of the network but we can express the error of an arbitrary layer $\boldsymbol{\delta}^l$ by the error in the next layer $\boldsymbol{\delta}^{l+1}$

$$\boldsymbol{\delta}^l = \left[(\hat{w}^{l+1})^\top \boldsymbol{\delta}^{l+1}\right] \odot \sigma'(\mathbf{z}^l) \tag{3.52}$$

This equation can be intuitively understood as moving the error $\boldsymbol{\delta}^{l+1}$ back one layer by applying $(\hat{w}^{l+1})^\top$ and then through the activation function of layer l by applying σ'. In a sense its again "simply" a chain rule. With these two equations (3.51) and (3.52) all the errors $\boldsymbol{\delta}$ in the network are known. Just start with the last layer and work your way backwards. This idea of moving the error backwards is the reason why the algorithm is called *Backpropagation*.

The last thing to do is to relate $\boldsymbol{\delta}$ back to the original derivatives $\partial C/\partial \hat{w}^l_{j,k}$ and $\partial C/\partial b^l_j$. Using equation (3.46) gives

$$\partial C/\partial b^l_j = \delta^l_j \quad \text{and} \tag{3.53}$$

$$\partial C/\partial \hat{w}^l_{j,k} = a^{l-1}_k \, \delta^l_j \tag{3.54}$$

Now we could modify the weights and biases after every forward pass by

$$\dots \tag{3.55}$$

4.add forumla to update weights based on gradient

But it is more efficient to average the changes and apply them at once after a small batch of training samples. This concludes the training of a Multilayer Perceptron but other Networks are trained in a similar manner. An excellent comprehensive explanation was published by Nielsen [5].

Convolutional Neural Networks

An area where NNs have been very successful is image recognition or more general computer vision but the described multilayer perceptron has a number of weaknesses for this kind of task. Let's say our input is a n by n gray scale image. This can be expressed as a $n \times n$ matrix, flattened and fed into the input layer as seen in figure 7. But now the number of weights to the next layer $\hat{\mathbf{w}}^2$ is $n \cdot n \cdot |\boldsymbol{l}^2|$ which soon becomes unfeasible. As described in the section on Notation $\boldsymbol{l}^2$ is here the vector of the second layer and not $\boldsymbol{l}$ squared.

Computational limits aside there is another problem. Imagine an image with the letter T in the top right corner. If this letter moves to a different position as in figure 8 the networks reaction will be completely different because the weights and biases involved are completely different. So the NN cannot learn the concept "letter T" independent of its position in the picture. The information about the distance between pixels is lost.

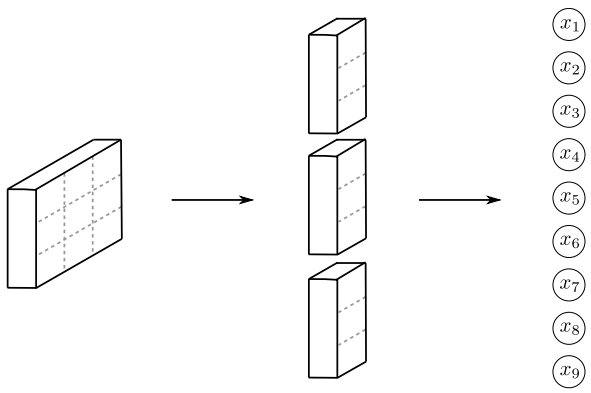


Figure 7: Flattening of a 3×3 matrix to fit the input of a multilayer perceptron.

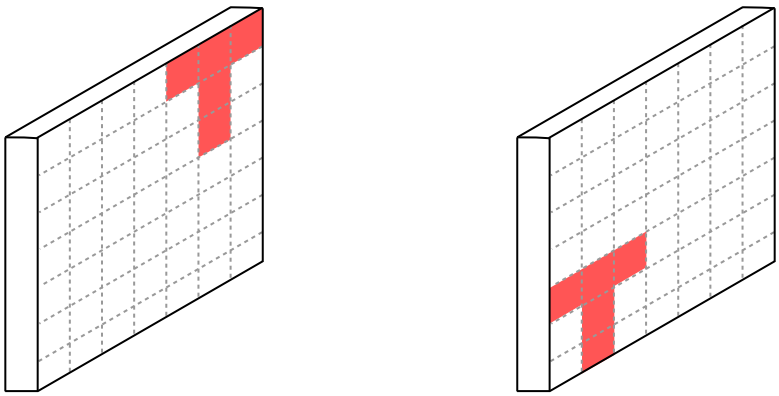


Figure 8: Two pictures of a T at different positions where the red color signifies a high value in the grayscale image. After the flatten operation seen in figure 7 very different nodes are active.

These problems led to the development of a new kind of layer called *Convolution*. A fixed size squared matrix called *kernel* is shifted over the matrix and at every position the point wise product between kernel and matrix is calculated and summed as shown in figure 9

The result of this operation called feature map of that kernel is shown in figure 10. Notice how the greatest value of the feature map is at the position of the letter T. So with only a small number of weights the convolution is able to detect the T independent of its position in the image. This is still slightly misleading because this "T kernel" was intentionally constructed to find the T. In a real convolutional layer the kernel values are trained via Backpropagation similar to the weights of a Multilayer Perceptron as described in the paragraph Training.

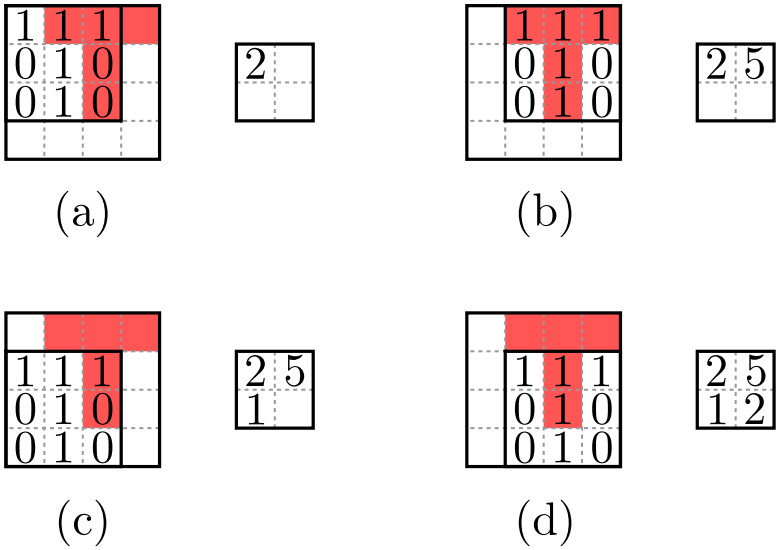


Figure 9: Example of a convolution. The 3×3 kernel is shifted over the image one step at the time. The red color in the image represents a pixel value of 1. For example in picture (a) the point wise product between kernel and image is zero everywhere except at two positions where a one in the kernel meets a one in the image. Because there are less valid positions for the kernel than pixels in the image the result is smaller in size.

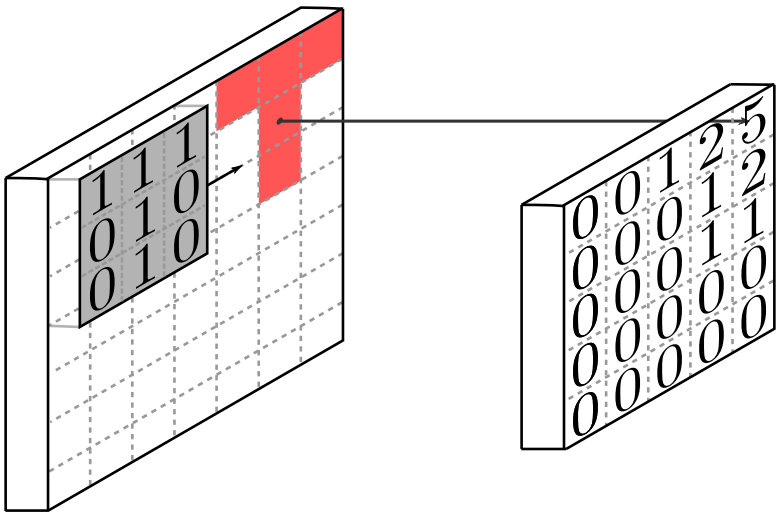


Figure 10: Example of a convolution where white pixel are 0 and red pixel are 1. The 3×3 kernel is shifted over the image and the point wise product between kernel and image is calculated at every step as described in figure 10. The result is the greatest when the kernel is directly over the letter T.

One convolutional layer contains not only one but a number of different kernels k. The resulting k feature maps are stacked in the "z direction" so that the shape of the $n \times n$ matrix transforms to $(n-2) \times (n-2) \times k$ when convolved with a 3×3 kernel. In a convolutional network (ConvNet) multiple of these layers are used so that it can find "patterns in patterns". For the letter detection example one could imagine the first layer to detect various edges and the next layer to detect letters in the position of these edges.

Pooling Layers

For a big image and a large number of kernels the output shape of a convolutional layers is still $\mathcal{O}\big((n)^2 \cdot k\big)$, so quite large. Also notice how in figure 10 the "T kernel's" feature map is not only active at the exact position of the T but in the general region. The solution to this is to downsample the output with a *Pooling Layer*. Here a smaller kernel, usually 2×2 is shifted over the matrix two steps at a time and at every position an operation is performed to reduce the number of values to one. This could be taking the maximum or the average of that 2×2 region. This operation reduces the matrix in the x and y dimension by a factor of 2 as shown in figure 11.

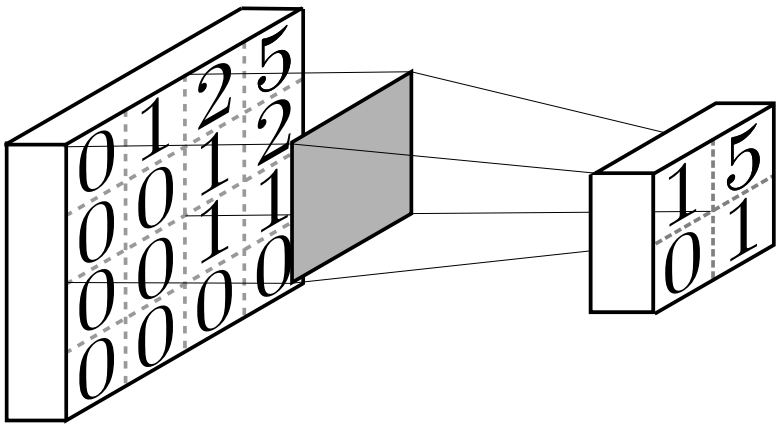


Figure 11: Example of a Max Pooling Layer. For every 2 by 2 field the maximum is calculated. After applying first the convolution and then the pooling layer the information "T in the top right corner" is still there and size of the resulting matrix is very manageable.

Example Network Architecture

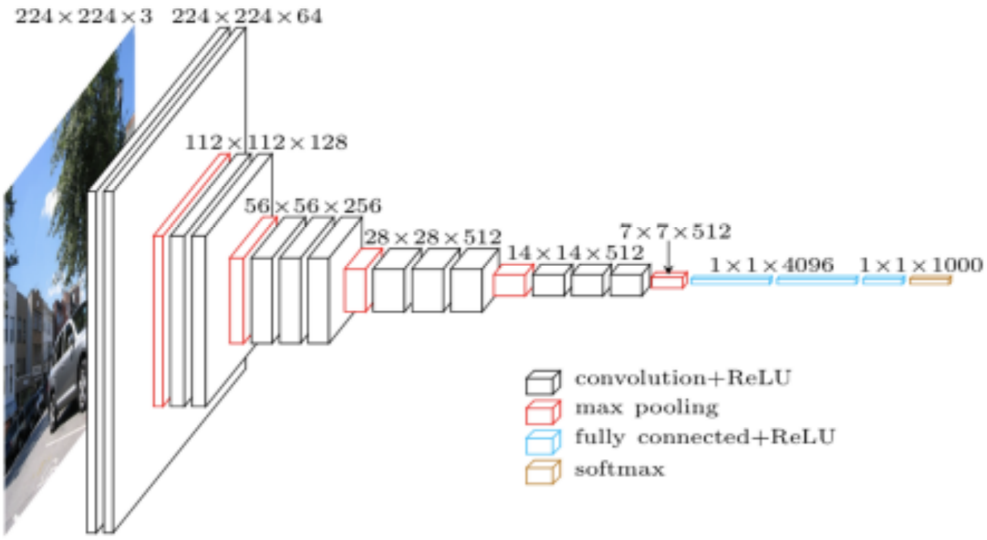


Figure 12: Example for a complete ConvNet. Note the input is in this case an RGB image so there are three layers in the z dimension corresponding to the different colors. In this case the first layer has 64 kernels of size $3 \times 3 \times 3$. The next convolution then has 128 kernels of size $3 \times 3 \times 64$. The final softmax activation rescales the output so that its sum is 1. [citation needed]

Now all the building blocks for a complete ConvNet are available. Repeatedly alternating convolution and pooling layers changes the input from wide in x and y dimension and narrow in z to a long z-strip. At the very end this strip is fed into one densely connected layer which is in turn connected to the output neurons. An example architecture of this kind is shown in figure 12.

1D ConvNets

The input to the desired algorithm is a target spectrum to which the Network should output some parameters (more on this in the section 4.2). This input data is a function $I(\lambda)$ so only one dimensional in contrast to an image which is a function $f(x, y)$ but all the same ideas apply. Convolutional kernels are sized $1 \times 3 \times z$ and pooling kernels are $1 \times 2 \times z$. Both are only shifted in one direction as see in figure 13. These 1D convolutions might detect features like rising and falling edges and the later layers might combine these features into concepts like peaks and troughs but as always in machine learning what the network actually does to reach its objective is not controlled by the programmer.

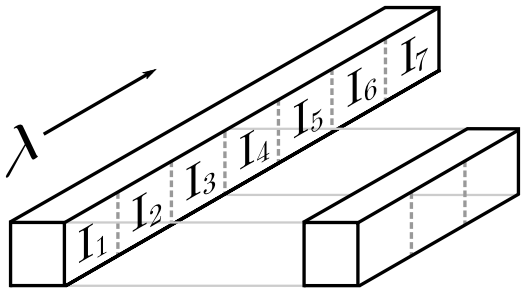
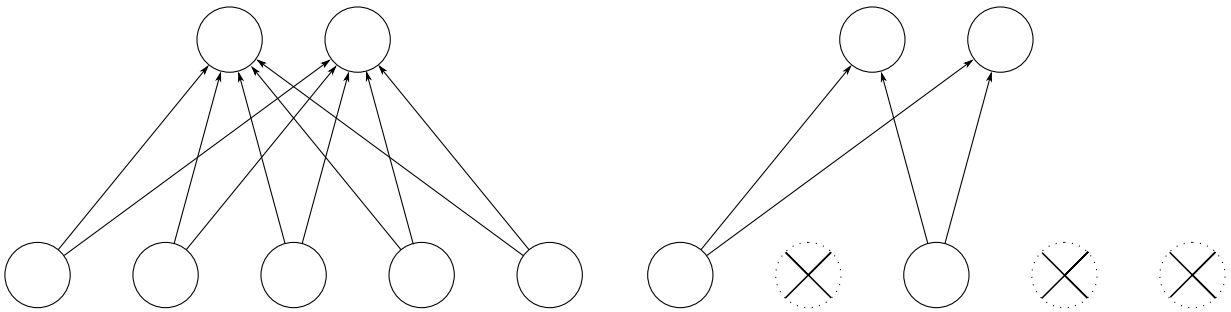


Figure 13: Example of a 1D convolution. A 1×3 kernel is shifted over a spectrum I discretized at 7 wavelengths

Dropout Layer

In 2014 Srivastava et al.[6] presented a method to prevent overfitting and speed up the training process of large Neural Networks. During training they randomly drop a number of neurons in a layer along with all connections to and from these neurons. This prevents the neurons from co-adapting **5.explain co-adapting** and because there are less weights and biases to tune for each step the training becomes overall faster. The process of dropping some neurons is shown in figure 14.



(a) no dropout (b) with dropout

Figure 14: Example of a dropout applied to the bottom layer. Only two of the neurons remain active and only their weights and biases are modified during this training step. [6] (modified)

4 Algorithm

Before starting to implement anything we need to define which metasurface stacks the algorithm is allowed to produce. This was already partly motivated in the introduction: We want easy to manufacture metasurfaces which can produce a high variety of transmission spectra when stacked.
Additionally, to keep things simple we want the transmission spectra to be the same for every point of entry along the base of the stack. Transmission spectra which also depend on the point of entry so $I = I(\lambda, x, y)$ can also be done [7] but functions $I = I(\lambda)$ are already useful and can represent for example a custom filter (the directions x and y are shown in figure 15).
With this in mind we are going to use a structure which is periodic in the x and y direction and consists of one repeating meta-atom. These kind of surfaces can be manufactured via ... **6.ask jan how these are made**.

As shown in section 3.2 paragraph Symmetries we cannot use meta-atoms of C_4 symmetry to produce different transmissions for the x and y polarization. The next simplest geometry is the rectangle. Another limitation given by the manufacturing process is the number of metasurface layers in a stack. Stacks with more than two layers become increasingly hard to manufacture that is why we are going to limit the algorithm to two-layer stacks even though theoretically arbitrary stacks could be used.

The last decision about the metasurfaces geometry is a little heuristic. A large variety of transmission spectra include some that are very absorbent everywhere except for specific wavelengths and some where almost all wavelengths are transmitted. A metal metasurface absorbs light when the electrons in the meta-atoms can oscillate in resonance. For rectangular meta-atoms this is the case for very specific wavelengths so metasurfaces made of these produce the former kind of spectrum. The opposite can be achieved when using rectangular holes. These metasurfaces have many different resonance frequencies and only the wavelengths that would have "fit" into the holes are not absorbed. That means the hole geometry can be used to produce the latter kind of spectrum and we have to include both geometries to reach the greatest variety.

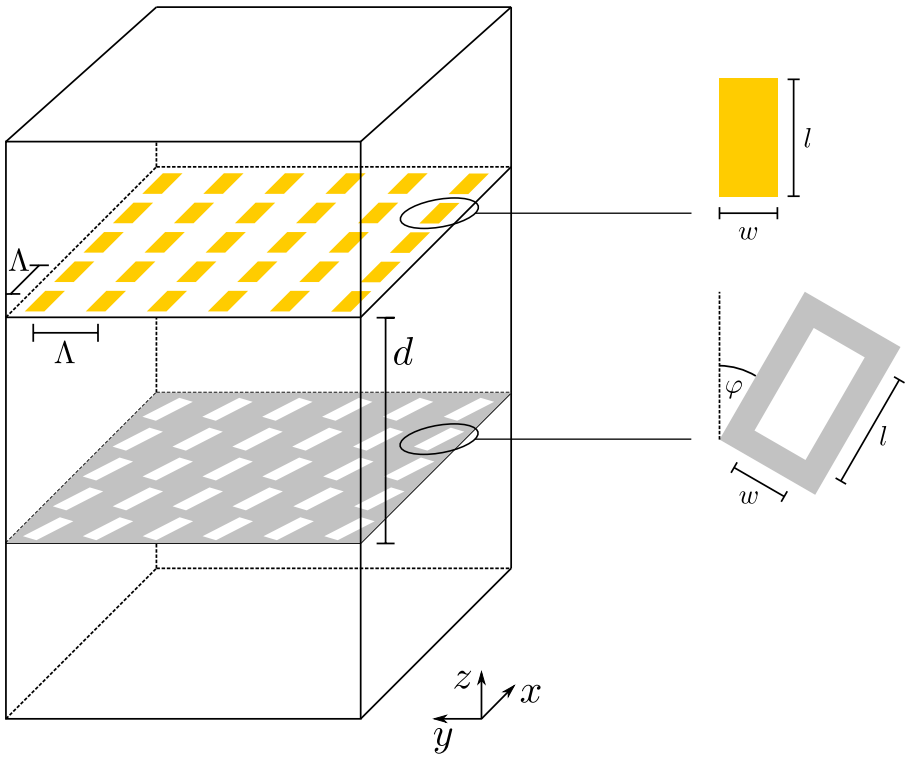


Figure 15: Schematic of a two layer metasurface stack to visualize all the design parameters $\mathcal{D}$ which can be set by the algorithm. These include stack parameters $\mathcal{S}$ and two sets of layer parameters $\mathcal{L}_1$ and $\mathcal{L}_2$. So $\mathcal{D} = (\mathcal{S}, \mathcal{L}_1, \mathcal{L}_2)$. The stack parameters consist of the distance between the metasurfaces d and their rotation angle φ so $\mathcal{S} = (d, \varphi)$. The layer parameters are the width w, length l and thickness t of the meta-atom, the period of meta-atoms Λ the material of the layer m and the kind of geometry g. So $\mathcal{L} = (w, l, t, \Lambda, m, g)$. All of the parameters are continuous except for the choice about the material and geometry.

As for the material we are going to limit the algorithm to Aluminum and Gold this is not physically motivated but we have to remember that every material we add multiplies the amount of metasurfaces that need to be simulated in preparation (see *curse of dimensionality*) and more materials are only useful when the corresponding parameter space is sufficiently. This ends the design considerations, a visualization of all parameters and the notation we are going to use referring to them can be found in figure 15.

4.1 Implementation

Having done the necessary preliminary considerations we can now start implementing the algorithm. It is organized into separate software modules defined by their inputs and outputs. The composition of these modules to the complete algorithm is shown in figure 16 and the individual modules will be discussed in the following sections. Each section will start with the input and output to that module.

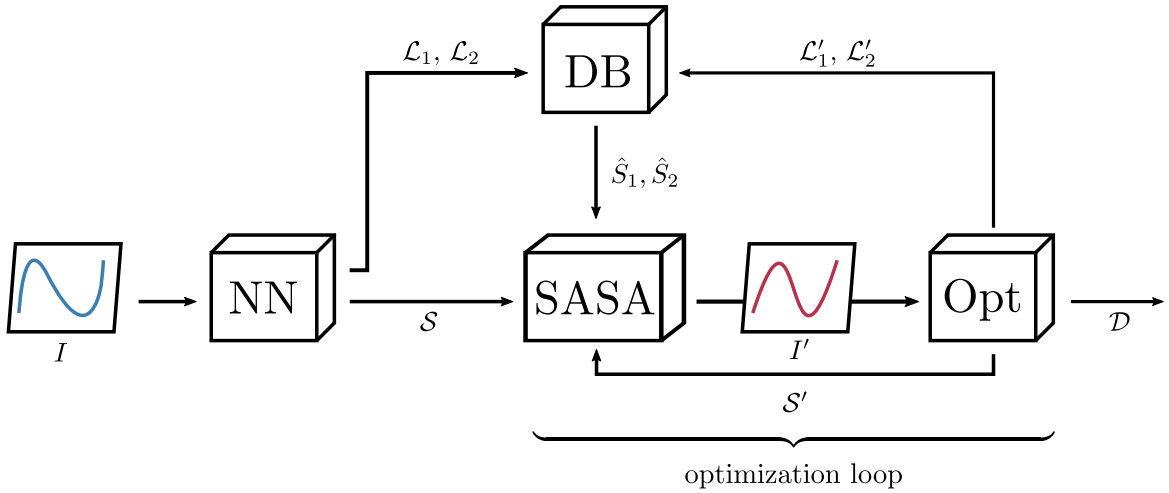


Figure 16: The algorithm tries to find to a given transmission spectrum I the design parameters $\mathcal{D}$ of a two layer meta surface stack which can reproduce this target. The input spectrum is passed to a convolutional neural network which outputs its guess for the stack and layer parameters. The Database module looks at the two sets of layer parameters and interpolates between stored S-matrices to give an estimate for the two S-matrices describing the layers. The SASA module calculates the resulting current transmission spectrum and passes it to the optimizer. This last module compares it to the target spectrum and adjusts the continuous parameters to minimize the difference between the two spectra.

NN:	convolutional neural network trained to map spectra to stack and layer parameters
DB:	database of pre-simulated single layers
SASA:	module calculating $\hat{S}_{\text{stack}} = \hat{S}_{\text{stack}}(\hat{S}_1, \hat{S}_2, \varphi, h)$
Opt:	optimizer changing the continuous parameters to minimize the difference between the current and target spectrum
$\hat{S}_1$, $\hat{S}_2$	S-matrices of the top and bottom layer
$\mathcal{L}_1$, $\mathcal{L}_2$	two sets of layer parameters where $\mathcal{L} = (w, l, t, \Lambda, m, g)$ w...width, l...length, t...thickness, Λ...Period, m...material, g...geometry
$\mathcal{S}$	stack parameters $\mathcal{S} = (d, \varphi)$ where d...distance between layers, φ...rotation angle
optimization loop	this loop is repeated until the target accuracy is reached
I	input transmission spectrum
$\mathcal{D}$	output design parameters

4.2 Network

Input:	transmission spectrum $I = (I_\mathrm{x}, I_\mathrm{y})$ a $\lambda \times 2$ array
	$I_{\mathrm{x/y}}$ ... X- and Y-transmission spectra, λ ... number of wavelengths
Output:	two sets layer parameters $\mathcal{L}_1$ and $\mathcal{L}_2$, stack parameters $\mathcal{S}$

Network Architecture This module is a 1D Convolutional Neural Network instead of the basic Multi Layer Perceptron. It was chosen to utilize the translational invariance of ConvNets. For example the concept "peak" should be learned independent of its position in the spectrum. As described in section 3.3 a ConvNet provides this functionality. Another constraint on the network architecture arises from the different kind of outputs. Most of the outputs are continuous but the choices about material m and geometry g are discrete/categorical. These need different activation functions σ to reach the different value ranges. The continuous outputs are mostly bounded by physical constraints and m, $g \in [0, 1]$ as they are *one hot encoded*, meaning $1 \rightarrow$ "The layer has this property" and $0 \rightarrow$ "The layer does not have this property".

The different outputs also need different cost functions $C(y, y')$ during training where y' is the networks output and y is the known solution. For the continuous output one can simply use the mean squared error

$$C_\mathrm{mse}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{4.56}$$

as all outputs are equally important and the cost function should be indifferent on whether the networks prediction is over or under target. For the categorical output the network learns quicker with the *Categorical Cross-Entropy* error.

$$C_\mathrm{ce}(\mathbf{y}, \mathbf{y}') = -\sum_i y_i \log y_i', \tag{4.57}$$

This error treats false positives ($y_i = 0$, $y_i' = 1$) and false negatives ($y_i = 1$, $y_i' = 0$) differently. A false positive does not increase the overall cost as $y_i = 0 \Rightarrow C_\mathrm{ce} = 0$ but for a false negative $C_\mathrm{ce} \rightarrow \infty$. This is wanted behavior because it does not matter if the network outputs some probability for a wrong class as long as it outputs a higher probability for the correct class. The final architecture is similar to the example given in figure 12 while meeting the above-mentioned constraints:

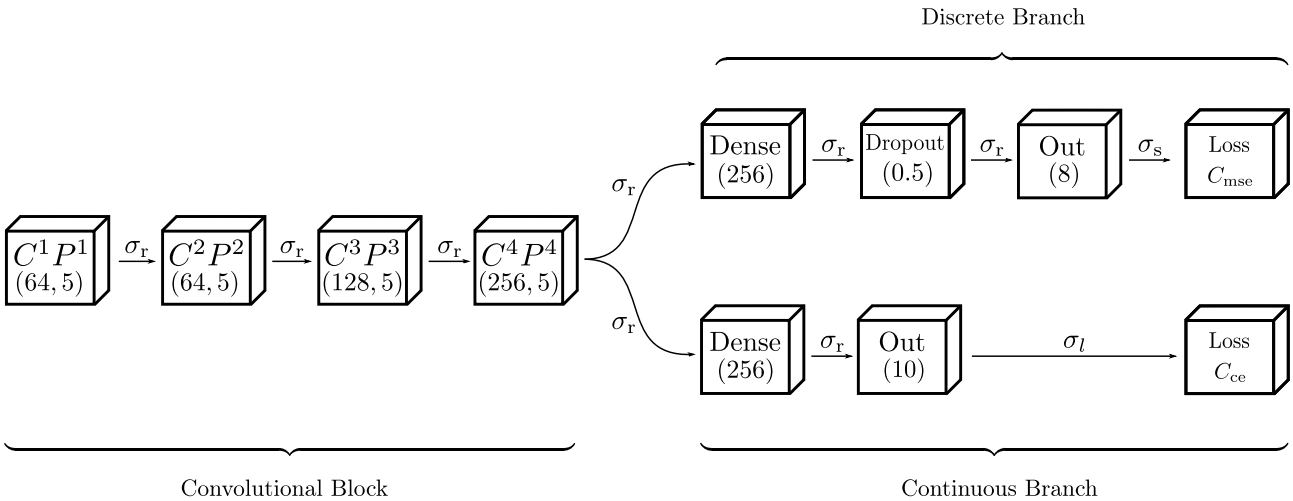
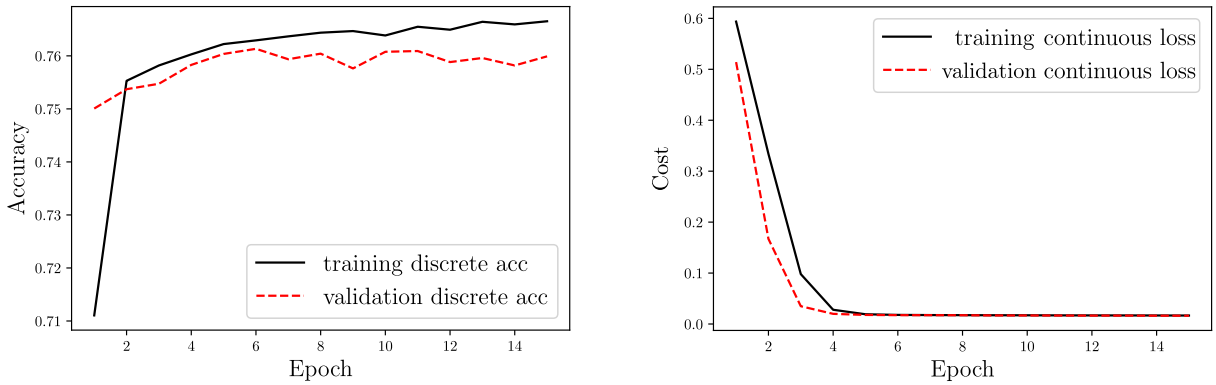


Figure 17: The network starts with 4 pairs of convolutional and pooling layers C^iP^i. The convolutions are characterized by (*number of kernels*, *kernel size*). The kernel size is always 5 and the number of kernels is gradually increased. Then the Network splits into a discrete and a continuous branch via two Dense layers with (*number of neurons*). In the discrete branch a dropout is applied to the dense layer where (0.5) is (*fraction of neurons to drop*). All the internal activations σ_r are ReLu's and the final activations σ_s and σ_l are a sigmoid and a linear function.

Network Training

To train a Neural Network one needs a training set (X, Y) of known input output pairs. In this case they are generated using the pre-simulated single layers in the database which are randomly combined into stacks. Then the stacks X- and Y-transmission spectra (I_x, I_y) are calculated via SASA. That means $I = (I_x, I_y) \in X$ are the networks input and the random design parameters $\mathcal{D} = (\mathcal{S}, \mathcal{L}_1, \mathcal{L}_2) \in Y$ are the output. For the first test we used squares and square holes of Aluminum and Gold. During the training one epoch is defined as one loop over all training samples. After every epoch the network is validated on a data set $(X_\mathrm{v}, Y_\mathrm{v})$ of samples it has not seen before. This is done to check wether the network actually learns something rather than just memorizing the input data. The results of this first training on the square geometry can be seen in figure 18:



(a) (b)

Figure 18: Training only with squares. (a) shows the accuracy of the discrete branch that is what percentage of choices for material and geometry were correct. In (b) we can see the performance of the continuous branch evaluated directly through its cost function C_mse.

Training and validation results are very similar which indicates that there is no overfitting or memorization. The discrete accuracy quickly reaches a maximum of $\sim 76\%$ but the speed at which this value is reached is suspicious. The network does not improve much after the fourth epoch and this does not change by tuning the network architecture. That is because the issue lies not within the architecture but in the training data. In section 3.2 we have shown that for the used two layer stacks the transmission spectrum is reciprocal that is the same for both directions. That means the data generation can result in two different stacks which produce the same spectrum. Consider a stack where one layer is Aluminum and the other is Gold as seen in figure 19 . As both of them produce the same spectrum one time the network is taught that the first layer is Gold and another time its taught the complete opposite. Actually if the network is trained this way it only ever predicts stacks with layers of equal materials because this is the only setup it can get right.

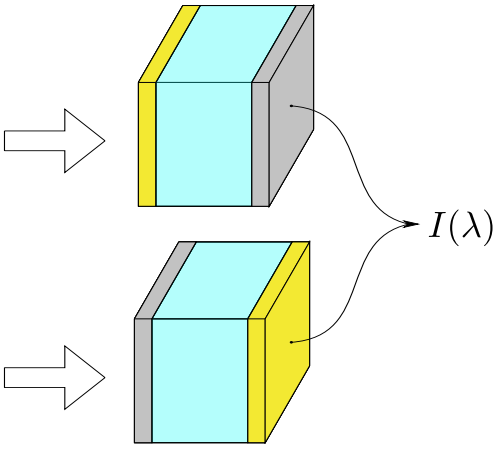
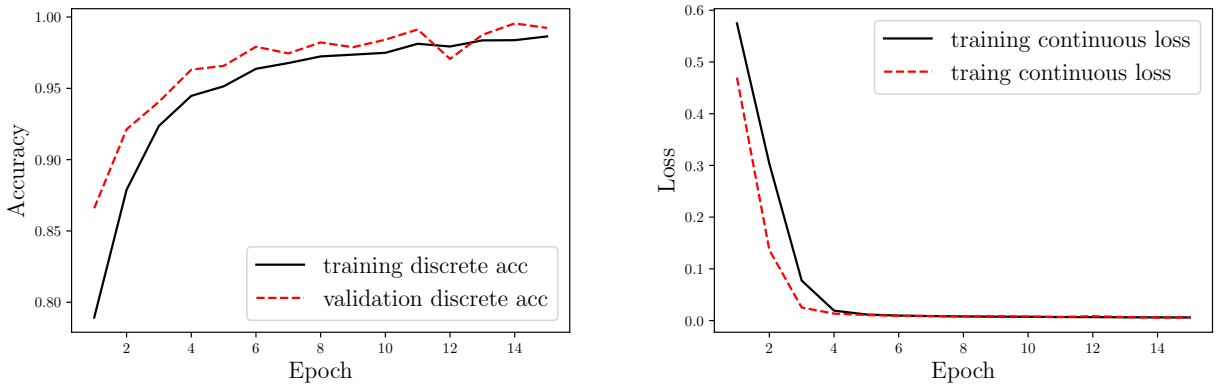


Figure 19: A stack of one Gold and one Aluminium layer separated by a glass spacer. Both stacks produce the same spectrum which leads to issues when using completely random stacks to train the network.

This is a well known problem when trying to solve an inverse problem. In this case the function $\mathcal{D} \to I$ is well defined as one stack can only produce one spectrum but for the inverse problem we are trying to solve $I \to \mathcal{D}$ and there might be multiple designs which produce the same spectrum. We can solve the issue described in figure 19 simply by allowing only one of the orientations into the training data. By doing this the training results change as seen in figure 20:



(a) (b)

Figure 20: Training on the square geometry after only allowing one of the equivalent stacks shown in figure 19 into the training data. Again, (a) shows the accuracy of the discrete branch and (b) the average cost of the continuous branch. The discrete accuracy has improved significantly compared to figure 18 but the continuous loss remains similar.

The discrete accuracy is much better at $\sim 98\%$ and the training curve looks more natural in the sense that improvements diminish over time but do not hit a sudden barrier as they did in figure 18. However, the performance of the continuous branch could not be improved in this manner and we will discuss the reason for this in chapter 5.

4.3 Database

Input:	Layer parameters $\mathcal{L} = (w,\, l,\, t,\, \Lambda,\, m,\, g)$
Output:	Interpolation of the S-matrix for this layer $\hat{S} = \hat{S}(\mathcal{L})$

Fourier Modal Method

The database consist of ~ 5000 S-matrices of single layers which were simulated with the Fourier Modal Method (FMM) on a compute cluster. This method is applicable to all surface structures which are periodic in x and y direction and constant in z direction. It works by expanding the involved fields into their diffraction orders via a Fourier expansion. For example for the reflected electric field is:

$$\mathbf{E}_\mathrm{r} = \sum_{m,n} \mathbf{R}_{mn} e^{i\mathbf{k}_{mn}\mathbf{r}} \tag{4.58}$$

Applying Maxwell's equations and the continuity conditions described in section 3.1 results in an eigenvalue problem which can be reformulated into a linear set of equations. The unknown properties like R_{mn} are found by solving this set of equations. Computationally a Fourier series like (4.58) has to be truncated at some order. This order determines the accuracy of the FMM and the matrix which represents the set of linear equations is sized order $\times$ order. Because of that the computational complexity of this method increases rapidly with the order. The method was first introduced by Noponen and Turunen [8].

Interpolation

To find the S-matrix to layer parameters $\mathcal{L}$ which are not already stored in the database this module has to interpolate between pre simulated S-matrices. First it looks for the n closest neighbors of $\mathcal{L}$. To do that the continuous input is normalized:

$$\bar{\mathcal{L}}_i = \frac{\mathcal{L}_i - \mathcal{L}_i^\mathrm{min}}{\mathcal{L}_i^\mathrm{max} - \mathcal{L}_i^\mathrm{min}},\ i \in 1...4 \quad \text{so that} \quad \bar{\mathcal{L}}_i \in [0,\, 1] \tag{4.59}$$

Now the distance d to every entry in the database satisfying the material geometry combination is calculated and the n entries with the smallest distance are selected where:

$$d(\mathcal{L}^1,\, \mathcal{L}^2) := \sum_{i=1}^{4} \left|\mathcal{L}_i^1 - \mathcal{L}_i^2\right| \tag{4.60}$$

The output $\hat{S}(\mathcal{L})$ is calculated via Inverse Distance Weighting [9] so that more distant entries have a smaller effect on the result. Let $(\mathcal{L}^1,\, \ldots,\, \mathcal{L}^n)$ be the n closest neighbors to $\mathcal{L}$ with stored S-matrices $(\hat{S}_1,\, \ldots,\, \hat{S}_n)$ then:

$$\begin{gathered} \hat{S}(\mathcal{L}) = \sum_{j=1}^{n} w_j\, \hat{S}_j \quad \text{where} \quad w_j = \frac{1/d_j^2}{\sum_i\ 1/d_i^2} \\ \text{so that} \quad \sum_j w_j = 1 \end{gathered} \tag{4.61}$$

4.4 Optimizer

Input:	Current spectrum I_c, Current design parameters $\mathcal{D}$
Output:	Improved design parameters $\mathcal{D}'$

The optimizer is at the core a Downhill-Simplex [10] tuning the continuous design parameters to minimize the mean-squared-difference between the current spectrum I' and target spectrum I we defined as $C_{\text{mse}}(I, I')$. However, the standard approach is unable to follow the physical constraints discussed in section 3.2 and has to be modified in that regard. To achieve this we can introduce a distance to the boundary D. Let p be a single continuous parameter with lower bound p^l and upper bound p^u, then:

$$D\Big(p, p^l, p^u\Big) = \begin{cases} p^l - p, & \text{for } p < p^l \\ 0, & \text{for } p^l \leq p \leq p^u \\ p - p^u, & \text{for } p^u < p \end{cases} \tag{4.62}$$

In this way one can penalize the simplex for stepping over the set boundaries by using a total loss L which depends on the sum of all distances D_i:

$$L(I_c, I_t, p) = \underbrace{C_{\text{mse}}(I_c, I_t)}_{\text{find target}} + \underbrace{\left[\sum_i D\Big(p_i, p_i^l, p_i^u\Big)\right]^3}_{\text{stay within bounds}} \tag{4.63}$$

The choice of exponent, 3 in this case, depends on how much the simplex should be penalized for stepping over a boundary. All our conditions are requirements for physical approximations and these approximations do no break completely when a boundary is only slightly violated. That justifies this approach where the simplex might step over a boundary but is then "pushed back" in the right direction. In figure 21 we can see the optimizer in operation:

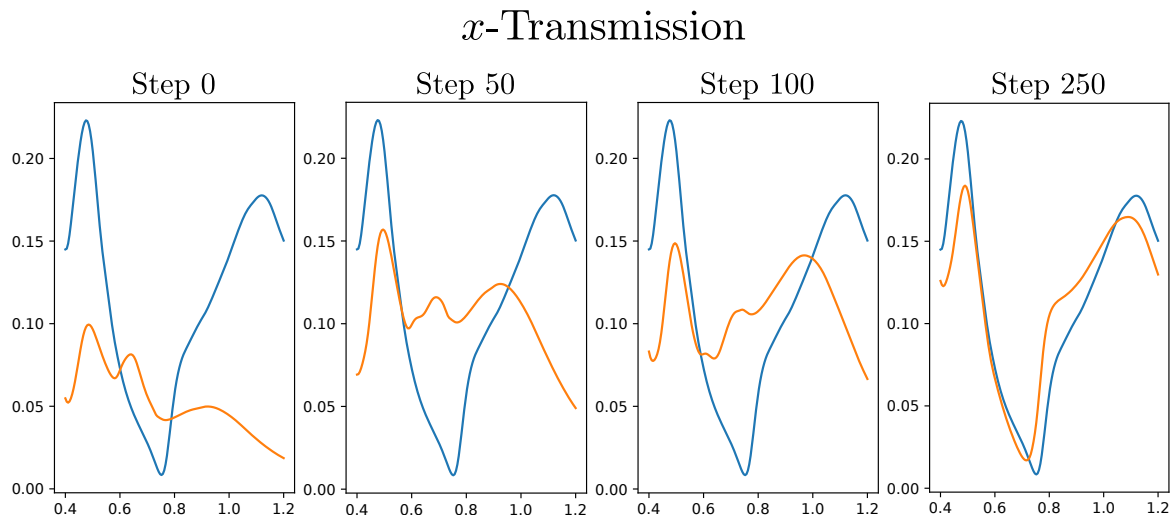


Figure 21: The fraction of transmitted light polarized in the x direction over the wavelength in μm at optimization steps $(0, 50, 100, 250)$ left to right. Blue is the target spectrum I and orange is the current spectrum I'.

In the very left graph of figure 21 at step 0 the optimizer has not jet tuned any parameters and the current and target spectrum are quite different. The blue target spectrum is in this case produced by a random stack from the validation data. That means there is a set of parameters which could reproduce this spectrum perfectly. By step 250 the optimizer has found a design which comes close to the original.

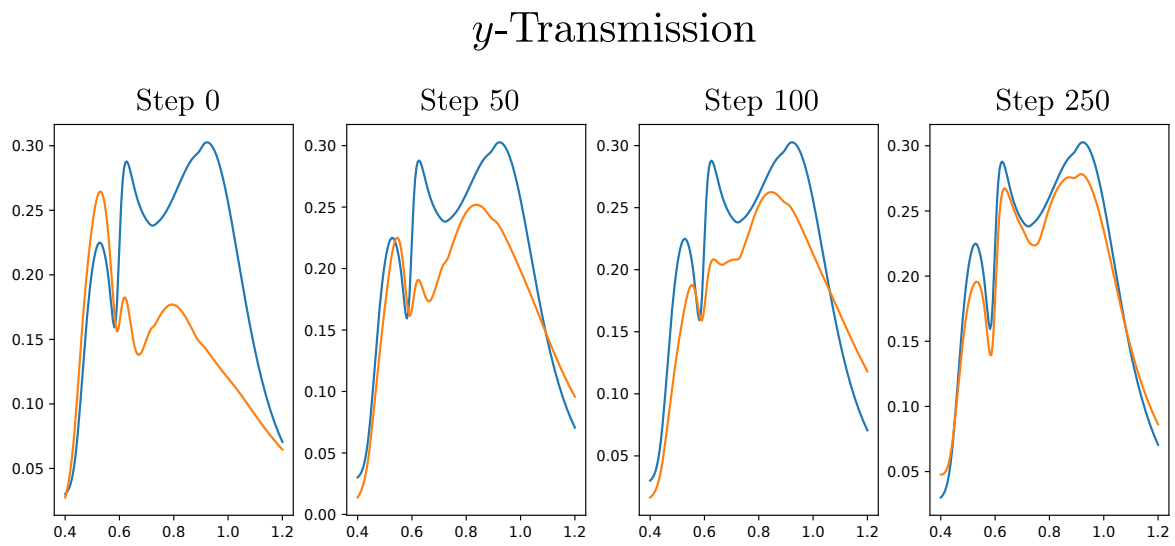


Figure 22: Simultaneous to the optimization of the x polarization shown in figure 21 the optimizer also tunes the y polarization.

To gain more insight we can consider how the parameters changed during the optimization as seen in figure 23. As designed, the optimizer did not change the material and geometry parameters and they were chosen correctly by the Neural Network. This is in accordance with the high discrete accuracy seen in section 4.2. Some of the continuous parameters are close to the original but others like the rotation angle φ are completely different. This shows that even though we have removed the equivalent stacks described in figure 19 there are still pairs of designs $\mathcal{D}_1, \mathcal{D}_2$ with different continuous parameters which produce very similar spectra $I(\mathcal{D}_1) \simeq I(\mathcal{D}_2)$ and these make it impossible for a network of this architecture to learn about the continuous parameters with good accuracy.

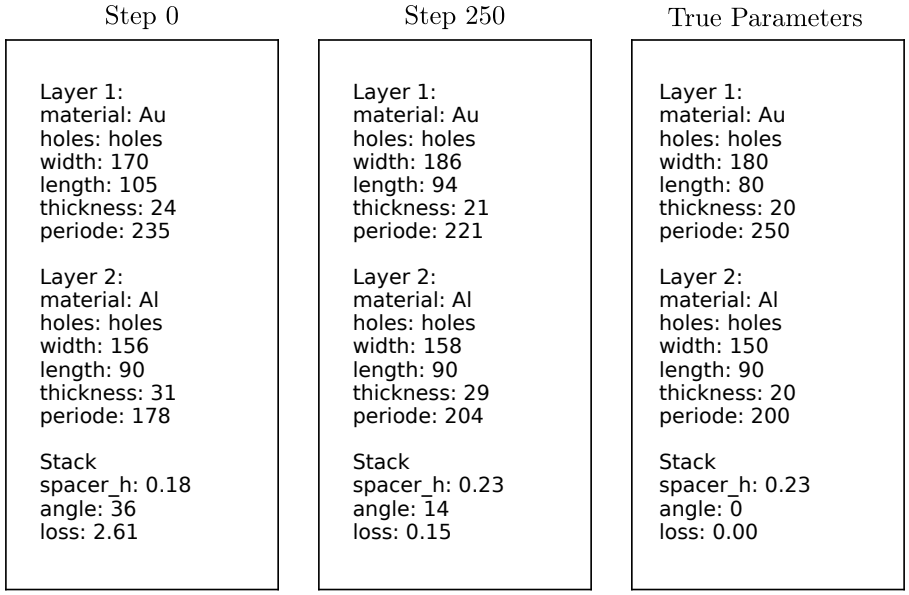


Figure 23: The design parameters $\mathcal{D}$ to the optimization shown in figure 21 and 22 at the beginning and end compared to the true parameters which produced the target spectrum.

5 The Inverse Problem

In section 4 we have seen that it is very difficult or even impossible to solve the inverse problem $I \to \mathcal{D}$ directly with a neural network because of multiple designs mapping to a single spectrum. For the algorithm we solved this by adding a conventional optimization method after the network to tune the parameters the network was not able to set correctly. This had some success in the sense that it was able to reproduce know stacks and even some more general functions **7.add this in Appendix?** but this approach is also a big concession because we loose many of the advantages neural networks bring. Conventional optimization methods are way more computationally expensive than a single forward pass in a neural network and more importantly they are very prone to getting stuck in local minima. Having one network being able to solve $I \to \mathcal{D}$ directly would be the best solution.

As this is a well known problem numerous solutions have been proposed. A very interesting route was taken by Liu et al. [11]. They approach the inverse problem by first solving the forward problem in our case $\mathcal{D} \to I$ and then building a combined network in a kind of tandem structure $I \to \mathcal{D} \to I'$ where the cost function is $C = C(I, I')$. That means during training the network is given a target spectrum I and predicts the corresponding design parameters $\mathcal{D}$ this design is then fed into the forward model and produces a spectrum I' and the cost function depends on the difference in I and I'. This approach completely circumvents the issue of multiple designs mapping to a single spectrum because it only depends on the differences between the target and the produced spectrum. We can already solve the forward problem via the DB and SASA modules as seen in figure 24.

This combined model can easily be implemented as all the necessary modules are already there but it can not be trained. To understand why we need to go back to the concept of Backpropagation. During training the gradient of the cost function needs to propagate backwards through the network. To capture this mathematically we will identify the forward model in figure 24 as the last layer L in a combined network. Equation (3.49) tells us we need to calculate:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \tag{5.64}$$

The first part $\partial C/\partial a_j^L$ is simple and only depends on the cost function. For example using $C_{\text{mse}} = \sum_j \left(a_j^L - y_j\right)^2$ the derivative is $\partial C/\partial a_j^L = 2\left(a_j^L - y_j\right)$ However, the second part $\partial a_j^L/\partial z_j^L$, the output of the last layer derived by the input to the last layer, is not easily accessible. Maybe not accessible at all if we consider the calls to the database and interpolation that happen during this step. The only way to train this combined tandem model is by replacing the forward model with something where we can access the gradient.

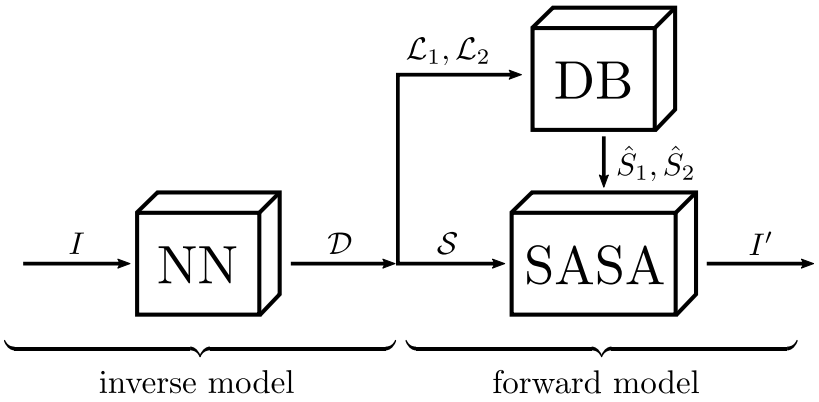


Figure 24: A part of the algorithm described in section 4 can be understood as a tandem model $I \to \mathcal{D} \to I'$. This combined model receives a target spectrum and outputs its best reproduction of that spectrum I'. The notation is the same as in figure 16.

5.1 Forward Model

6 Junk

A bunch of collected junk

7 Sources

References

[1] R. A. Shelby. Experimental verification of a negative index of refraction. *Science*, 292(5514):77–79, apr 2001.

[2] Nanfang Yu and Federico Capasso. Flat optics with designer metasurfaces. *Nature Materials*, 13(2):139–150, jan 2014.

[3] C. Menzel, J. Sperrhake, and T. Pertsch. Efficient treatment of stacked metasurfaces for optimizing and enhancing the range of accessible optical functionalities. *Physical Review A*, 93(6), jun 2016.

[4] R. M. Redheffer. On a certain linear fractional transformation. *Journal of Mathematics and Physics*, 39(1-4):269–286, apr 1960.

[5] Michael Nielsen. http://neuralnetworksanddeeplearning.com/chap2.html, December 2019.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[7] J. P. Balthasar Mueller, Noah A. Rubin, Robert C. Devlin, Benedikt Groever, and Federico Capasso. Metasurface polarization optics: Independent phase control of arbitrary orthogonal states of polarization. *Physical Review Letters*, 118(11), mar 2017.

[8] Eero Noponen and Jari Turunen. Eigenmode method for electromagnetic synthesis of diffractive elements with three-dimensional profiles. *Journal of the Optical Society of America A*, 11(9):2494, sep 1994.

[9] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference on -*. ACM Press, 1968.

[10] R. Mead J. A. Nelder. A simplex method for function minimization. *The Computer Journal*, 8(1):27–27, apr 1965.

[11] Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu. Training deep neural networks for the inverse design of nanophotonic structures. *ACS Photonics*, 5(4):1365–1369, feb 2018.