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

February 26, 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

I hope this all works jada jada jada Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

2 Notation

3 Physical Background

3.1 S-Matrix Calculus

To implement the desired algorithm we need to be able to calculate the optical behavior of stacked Meta Surfaces. 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 at the very beginning 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: **E**...electric field, **B**...magnetic flux density, **D**...electric flux density and **H**...magnetic field and the sources are the external charges ρ_{ext} and the macroscopic currents j. All the material properties are captured by the **D** and **H** fields which are defined as follows:

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

Where **P** is the dielectric polarization and **M** is the magnetic polarisation. One can read equation 3.5 in the following way: When the electric field **E** interacts with matter it exerts a force on all its charges and displaces them by a small amount. The separation of charges results in a counter field **P** and the total field **D** is now a superposition of **E** and **P**. This set of equations describes the whole electromagnetic spectrum, in this work however we are only interested in visible (VIS) and near infra red (NIR) light so we can make some simplifications. $\rho_{\text{ext}} = 0$ and $\mathbf{M} = 0$ why?. Inserting these assumptions into the maxwell 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 famous wave equation by considering $\nabla\times$ (3.6):

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

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 occilation in time and space along the **k** direction where **E**, **B** and **k** are all perpendicular to each other. show figure?

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 property $\chi(\omega,\mathbf{r})$. If the material is additionally homogeneous, that is the same everywhere independent of $\mathbf{r}$, then $\boldsymbol{\nabla}\cdot\chi(\omega,\mathbf{r}) = 0$. With equation (3.7) this gives us $\boldsymbol{\nabla}\cdot\mathbf{P} = 0$ and the wave equation simplifies to why is j=0:

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

So light behaves in these materials exactly as it would in vacuum we only have to account for a decreased speed of light $c' = \frac{c}{\sqrt{\varepsilon}} =: \frac{c}{n}$ with the refractive index n. A complex valued $n := \eta + i\kappa$ even captures the possibility of a decaying field:

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

Fresnel Equations

The meta surface stacks we want to understand are obviously not one homogeneous material. 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.

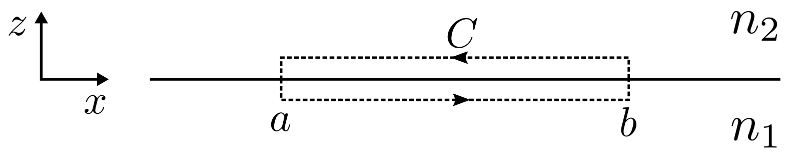


Figure 1: An interface of two materials with different refractive indices n_1 and n_2

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:

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

Now we can bring the contour infinitely 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 also chosen arbitrarily that means that the transverse field components along the path need to be continuous so $\mathbf{E}_1^{\parallel} = \mathbf{E}_2^{\parallel}$.

3.2 Neural Networks

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 then explicitly programmed NN's are trained on a dataset (X, Y) of correct input output pairs.

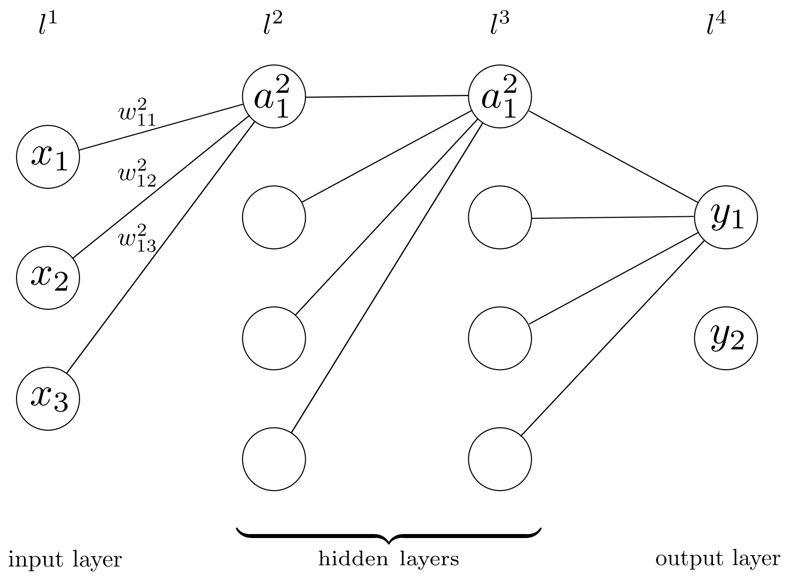


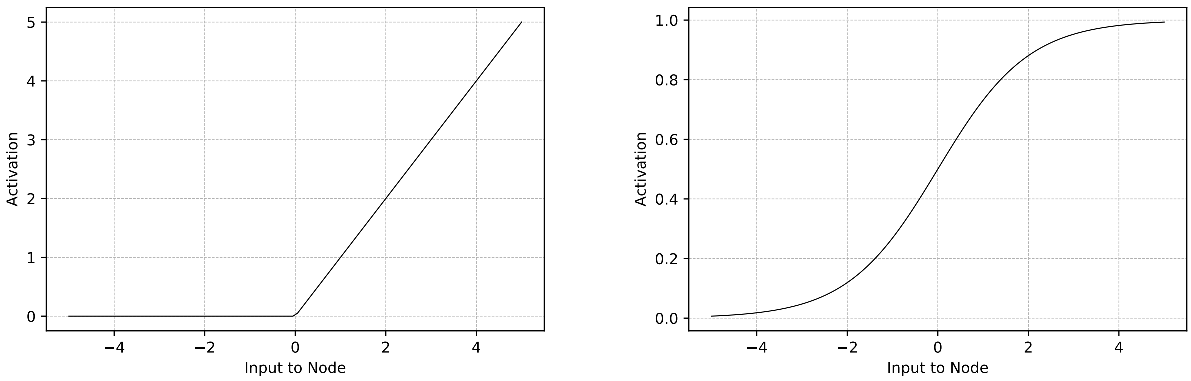
Figure 2: The most simple kind of NN is called densely connected or multilayer perceptron. For clarity only connections to the top most node of each layer are shown.

Multilayer Perceptron This kind of classic NN consist of single nodes which are organized into layers. Every node is connected to all the nodes of the previous and next layer thus they are called dense. 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). 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. For the index notation 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 2 that means:

$$a^2_1 = \sigma\left(\sum_i w^2_{1i}\, x_i + b^2_1\right) \tag{3.15}$$

However it is more convenient to stop considering every node individually and to view the involved quantities as vectors and matrices. So that (3.15) can be written as:

$$\mathbf{a}^l = \sigma\Big(\hat{\mathbf{w}}^l \mathbf{a}^{l-1} + \mathbf{b}^l\Big) \tag{3.16}$$



(a) ReLu (b) sigmoid

Figure 3: 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 network output $\mathrm{NN}(\mathbf{x}) := \mathbf{y}'$ is calculated though repeated use of (3.16) and is then compared with the known correct output $\mathbf{y}$ by a cost function $C = C(\mathbf{y}, \mathbf{y}')$. This might simply be the mean squared difference between $\mathbf{y}$ and $\mathbf{y}'$:

$$C_{\mathrm{mse}}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{3.17}$$

but there are more sophisticated cost functions for different kind of outputs. Now we can quantify how well the NN is performing but how should the weights and biases be changed to improve this performance? Here the very important Algorithm *Backpropagation* is used and allows a efficient calculation of $\boldsymbol{\nabla} C_{\mathbf{b}^l}$ and $\boldsymbol{\nabla} C_{\hat{\mathbf{w}}^l}$. These are used to gradually change the weights and biases to minimize the cost function. A very comprehensive explanation of Backpropagation can be found here: [1].

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 (see figure 4). But now the number of weights to the next layer $\hat{\mathbf{w}}^2$ is $n \cdot n \cdot l^2$ which soon becomes unfeasible.

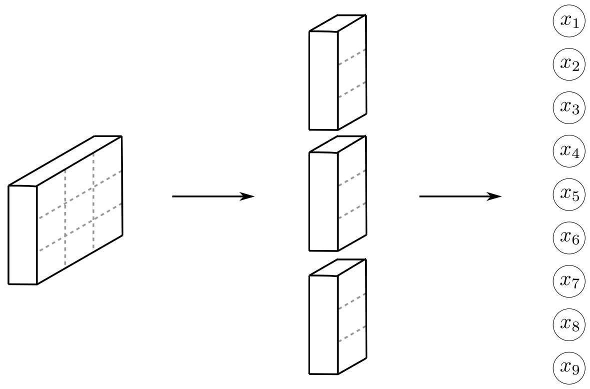


Figure 4: Flattening of a 3×3 matrix to fit the input of a multilayer perceptron.

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 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. Also the information about the distance between pixels is lost.

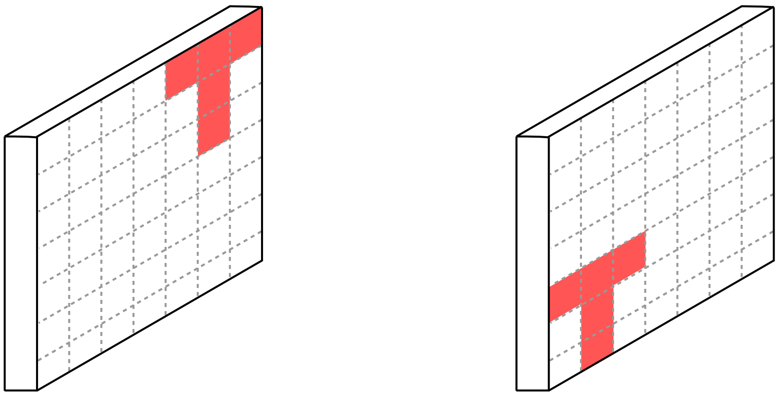


Figure 5: Two pictures of a T at different positions where the red color signifies a high value in the grayscale image. After the flatten operation seen in figure 4 very different nodes are active.

These problems led to the development of a new kind of layer called *Convolution*. A fixed size *kernel* is shifted over the matrix and at every position the the point wise product between kernel and matrix is calculated and summed. The result is called feature map of that kernel (see figure 6). 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. One convolutional layer contains not only a single but a number of different kernels k so that the shape of the $n \times n$ matrix transforms to $(n-1) \times (n-1) \times k$. 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.

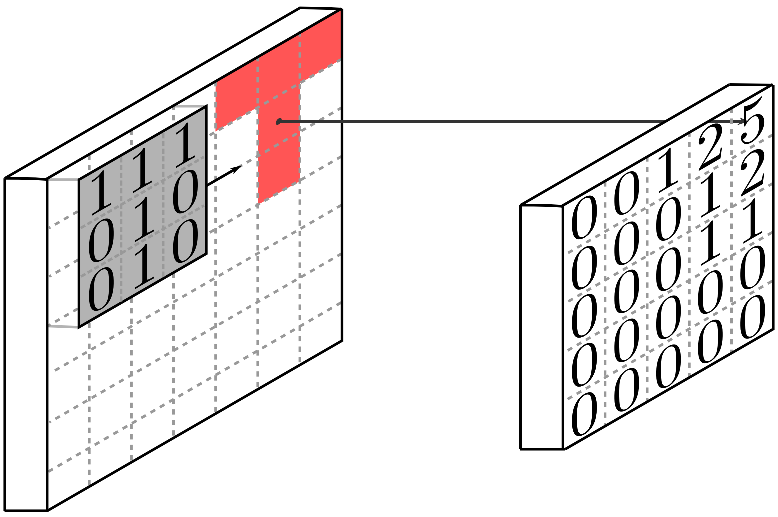


Figure 6: Example of a convolution where white pixel are 0 and red pixel are 1. The 3×3 Kernel is shifted over the image and when its directly over the The "T kernels" feature map is greatest at the position of the original letter.

Pooling Layers For a big image and a large number of kernels the output shape of a convolutional layers is still $n \times n \times k$, so quite large. Also notice how 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. This operation reduces the matrix in the x and y dimension by a factor of 2.

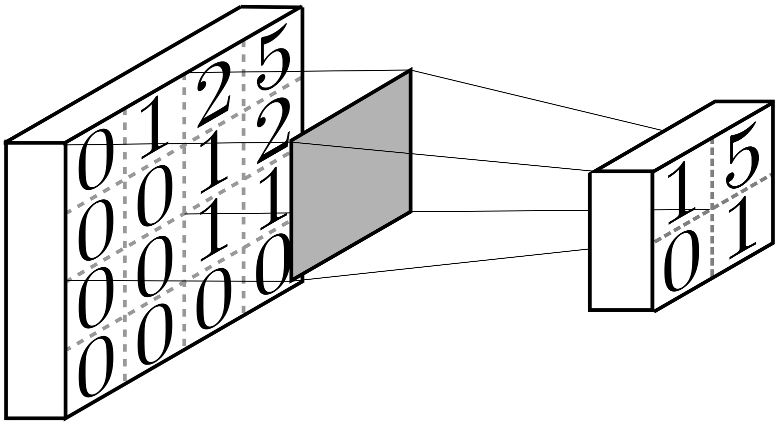


Figure 7: Example of a Max Pooling Layer. For every 2 by 2 field the maximum is calculated. After applying first the convolution and then the pooling layer the information "T in the top right corner" is still there and size of the resulting matrix is very manageable

Example Network Architecture 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.

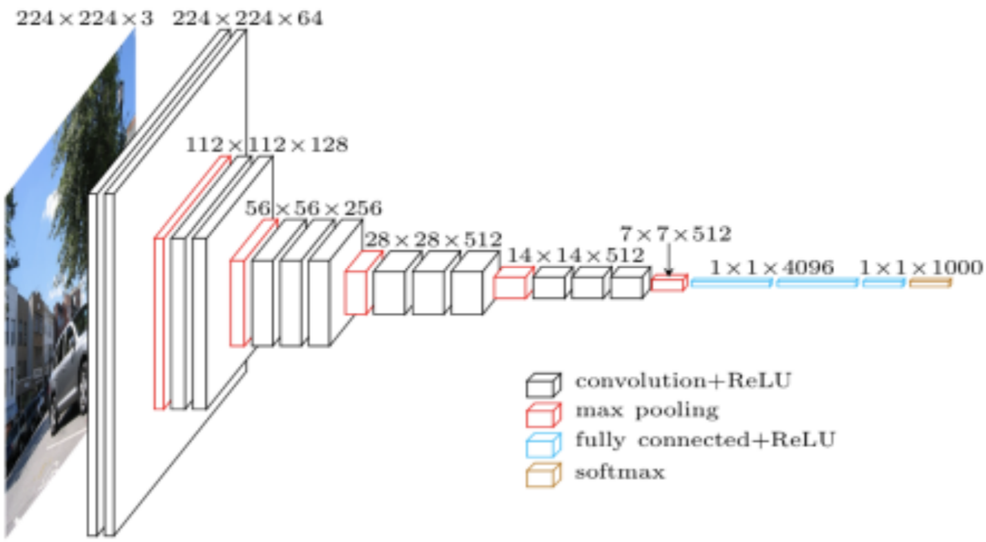


Figure 8: Example for a complete ConvNet. Note the input is in this case an RGB image so there are three layers in the z dimension corresponding to the different colors. In this case the the first layer has 64 kernels of size $3 \times 3 \times 3$. The next convolution then has 128 kernels of size $3 \times 3 \times 64$. [citation needed]

1D ConvNets The algorithm is applied to spectra, so a functions $I(\lambda)$. This data is only one dimensional but all the same ideas apply. Kernels are here sized $1 \times 3 \times z$ and are shifted in one dimension. Same goes for the pooling. These 1D convolution might detect features like rising and falling edges.

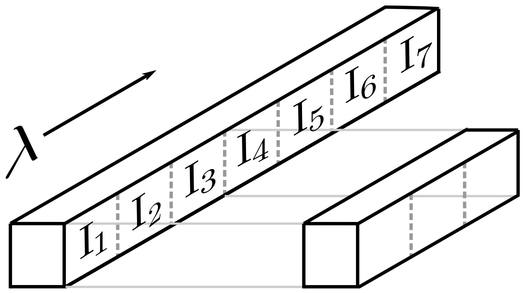
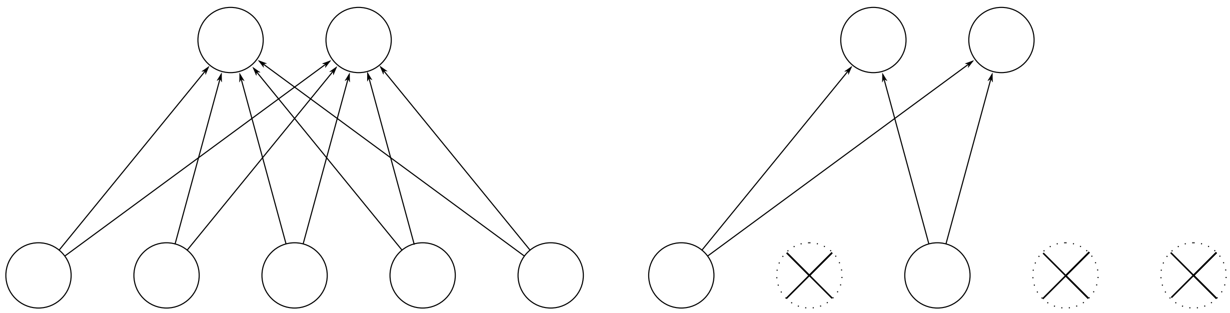


Figure 9: Example of a 1D convolution. A 1×3 kernel is shifted over a spectrum I discretized at 9 wavelengths

Dropout Layer In 2014 Srivastava et al.[2] 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 ? and because there are less weights and biases to tune for each step the training becomes overall faster.



(a) no dropout (b) with dropout

Figure 10: 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. Figure found in [2] (modified)

4 The Algorithm

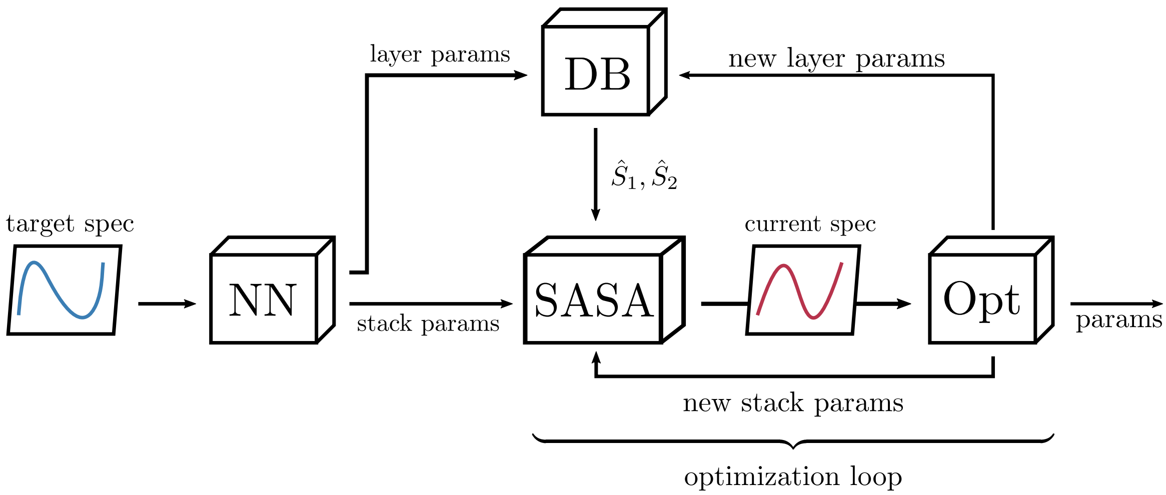


Figure 11: The algorithm tries to find to a given transmission spectrum the kind of two layer meta surface stack which can reproduce this target. The input spectrum is passed to a convolutional neural network which outputs its guess for the stack and layer parameters. The Database module looks at the two sets of layer parameters and interpolates between stored S-matrices to give an estimate for the two S-matrices describing the two layers. The SASA algorithm 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. Find the details to all parts of the algorithm in the sections below.

NN:	convolutional neural network trained to map spectra to stack and layer parameters
DB:	database of FMM simulated single layers
SASA:	algorithm calculating $\hat{S}_{\text{stack}} = \hat{S}_{\text{stack}}(\hat{S}_1, \hat{S}_2, \varphi, h)$
Opt:	optimizer changing 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
layer params	two sets of continuous parameters: $p = (w, l, t, \Lambda)$ and discrete parameters: $m \in$ (Al, Au), $g \in$ (rectangles, rectangular holes) w...width, l...length, t...thickness, Λ...Period, m...material, g...geometry
stack params	φ...rotation angle, h...distance between layers
new params	the Opt. only changes the continuous parameters, the discrete ones, e.g. material, remain unchanged
optimization loop	this loop is repeated until the target accuracy is reached

4.1 Network

Input:	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 $p = (w,\, l,\, t,\, \Lambda)$, m, g and stack parameters φ, h w...width, l...length, t...thickness, Λ...Period, m...material, g...geometry φ...rotation angle, h...distance between layers

Network Architecture This module is a 1D Convolutional Neural Network instead of the simple 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 3.2 a ConvNet provides this functionality. Another constraint on the network architecture arises from the different kind of outputs. $p = (w,\, l,\, t,\, \Lambda)$, φ and h are continuous and m, 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

$$C_{\mathrm{mse}}(\mathbf{y},\, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{4.18}$$

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.

$$C_{\mathrm{ce}}(\mathbf{y},\, \mathbf{y}') = -\sum_i y_i \log y_i', \tag{4.19}$$

Using this error if the network predicts $y_i' = 0$ for all categories then $C_{\mathrm{ce}} \rightarrow \infty$. why is this better?

The final architecture is similar to the example given in figure 8 while meeting the above-mentioned constraints:

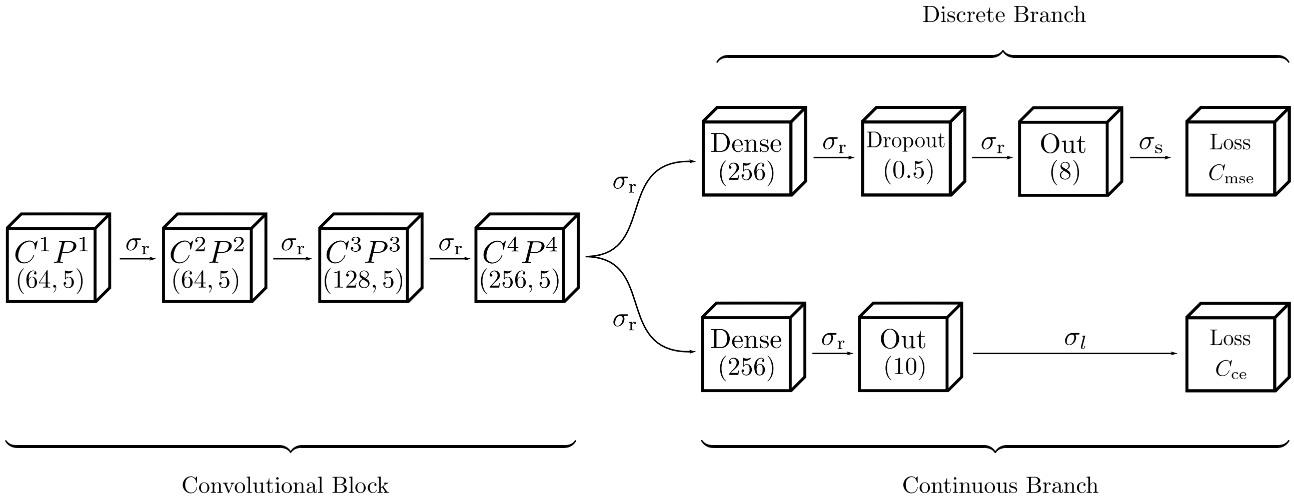


Figure 12: The network starts with 4 pairs of convolutional and pooling layers. The convolutions are characterized by (*number of kernels*, *kernel size*). The kernel size is always 5 and the number of kernels is gradually increased. Then the Network splits into a discrete and a continuous branch via 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 a stacks. Then this stacks X- and Y-transmission spectra (I_x, I_y) are calculated via SASA. That means $(I_x, I_y) \in X$ are the networks input and the random parameters $(p_1, m_1, g_1, p_2, m_2, g_2) \in Y$ are the output. Using this approach the following accuracy is reached:

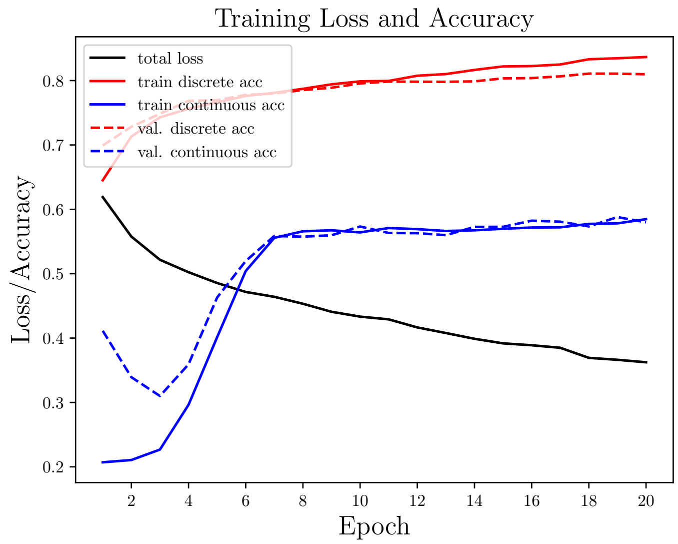


Figure 13: Shown are the total loss and the training and validation accuracy for both the continuous and the discrete branch. After each epoch the network is validated on data not used in training to check for overfitting.

Training and validation accuracy are very similar which indicates that there is no overfitting. The discrete accuracy quickly reaches a maximum of $\sim 80\%$ which is less than expected for this classification problem. The issue here lies not in the networks arichtecture but in the data generation process. In missing we have shown that for a two layer stack the transmission spectrum is the same for both directions. That means the data generation can result in two different stacks which produce the same spectrum. Lets consider a stack where one layer is Aluminium and the other is Gold. 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 constellation it can get right.

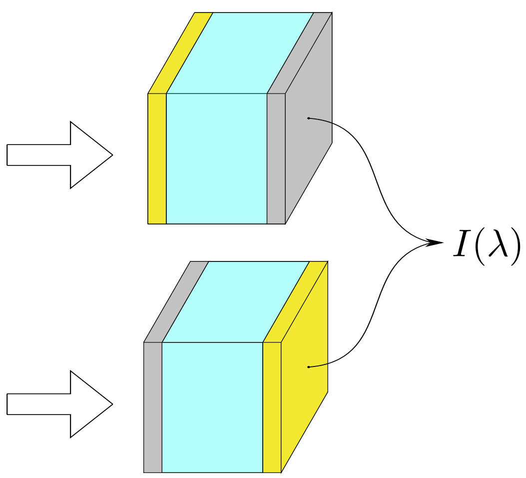


Figure 14: A stack of one Gold and one Aluminium layer separated by a glass spacer. Both orientations produce the same spectrum which leads to issues when using completely random stacks to train the network.

One can prevent this issue simply by allowing only one of the equivalent orientations into the training data. This does not limit the capabilities of the trained network as the variety of input spectra stays the same.

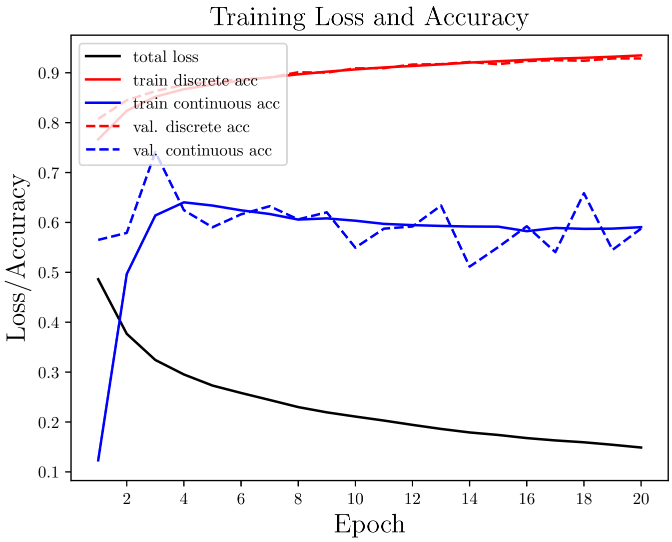


Figure 15: Network loss and accuracy after reducing equivalent stacks to one orientation. The discrete accuracy has improved and

4.2 Database

Input:	layer parameters $p = (w, l, t, \Lambda)$, material m geometry g
Output:	approximation of the S-matrix of this layer $\hat{S} = \hat{S}(p, m, g)$

The database consist of $\sim$ 5000 S-matrices of single layers which were simulated with the Fourier Modal Method (FMM) on a compute cluster. Its main input are the layer parameters width, length, thickness and period, so $p = (w, l, t, \Lambda)$. This module first looks for the n closest neighbors of p. To do that the input is scaled:

$$\tilde{p}_i = \frac{p_i - p_i^{\min}}{p_i^{\max} - p_i^{\min}} \quad \text{so that} \quad \tilde{p}_i \in [0, 1] \tag{4.20}$$

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:

$$d(p, q) = \sum_i |p_i - q_i| \tag{4.21}$$

The output $\hat{S}(p)$ is calculated via Inverse Distance Weighting [3] so that more distant entries have a smaller effect. Let $(q_1, \ldots, q_n)$ be the n closest neighbors to p with stored S-matrices $(\hat{S}_1, \ldots, \hat{S}_n)$ then:

$$\begin{gathered} \hat{S}(p) = \sum_{j=1}^{n} w_j \, \hat{S}_j \quad \text{where} \quad w_j = \frac{1/d_j^2}{\sum_i 1/d_i^2} \\ \text{so that} \quad \sum_j w_j = 1 \end{gathered} \tag{4.22}$$

To have this interpolated approximation $\hat{S}(p)$ be close to the result of rigoros simulation FMM(p) the simulated grid of parameters needs to be sufficiently dense. Insert some calculation of what is sufficiently dense

4.3 Optimizer

Input:	current spectrum I a $\lvert\lambda\rvert \times 2$ Array,
	current continuous parameters $p = (w, l, t, \Lambda, h, \varphi)$
	h...spacer height, φ...layer rotation
Output:	improved continuous parameters $\tilde{p}$

The optimizer is at the core a Downhill-Simplex [4] minimizing the mean-squared-difference between the current spectrum I_c and target spectrum I_t so it minimizes $C_\mathrm{mse}(I_\mathrm{c}(p),\, I_\mathrm{t})$ This standard method is however unable to follow the constraints and has to be modified in that regard. To achieve this one can introduce a distance to the boundary D. Let p be a single parameter with lower bound p^l and upper bound p^u, then:

$$D\left(p,\, p^l,\, p^u\right) = \begin{cases} p^l - p, & \text{for } p < p^l \\ 0, & \text{for } p^l \leq p \leq p^u \\ p - p^u, & \text{for } p^u < p \end{cases} \tag{4.23}$$

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:

$$L(I_\mathrm{c},\, I_\mathrm{t},\, p) = \underbrace{C_\mathrm{mse}(I_\mathrm{c},\, I_\mathrm{t})}_{\text{find target}} + \underbrace{\left[\sum_i D\left(p_i,\, p_i^l,\, p_i^u\right)\right]^2}_{\text{stay within bounds}} \tag{4.24}$$

align underbraces

The choice of power depends on how much the simplex should be penalized for stepping over a boundary. All our conditions are requirements for physical approximations and
Maybe 3D plot of an example loss function

5 Junk

A bunch of collected junk

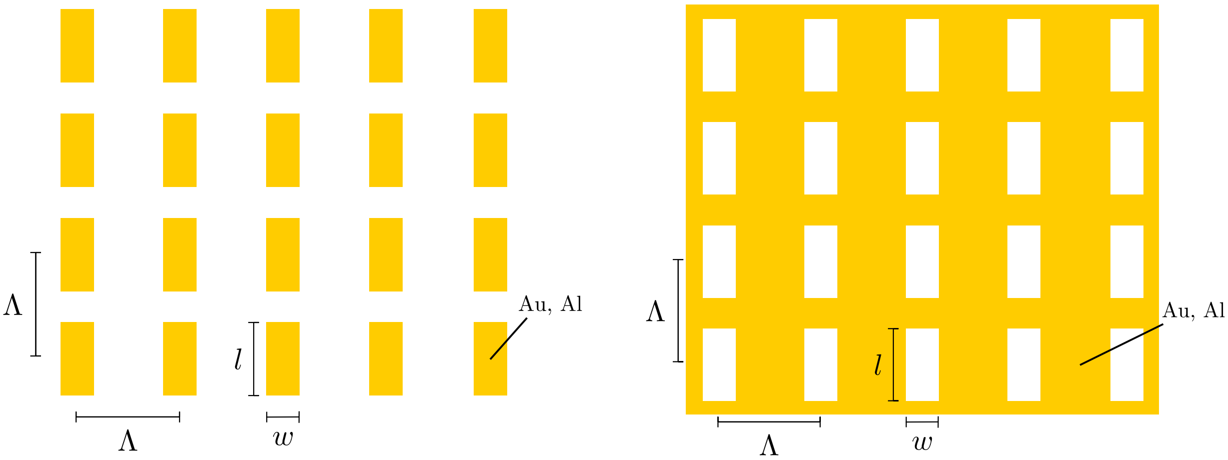
5.1 Meta Surfaces

Definition

Stacks

Geometry

The idea behind choosing a type of meta surface was: To use a simple and easy to manufacture geometry and achieve complex transmission spectra by stacking these simple layers on top of each other. It is essential to use particles of at least C^4 rotational symmetry to enable the layer to affect x and y polarizations differently. A fitting geometry are rectangular meta surface particles.



(a) rectangles (b) rectangular holes

Figure 16: The rectangular particles of width w and length l are arranged on a square matrix with Periode Λ in x and y direction. They are made of Gold or Aluminium. The second geometry (b) is inverse to the first in that wherever there was material now light can transmit freely. Both layers also have thickness t.

5.2 S-Matrix Calculus

Jones Formalism A planar lightwave propagating along the z axis trough a homogeneous material can be described as:

$$\mathbf{E} = \begin{pmatrix} E_x e^{i(kz-\omega t+\varphi_x)} \\ E_y e^{i(kz-\omega t+\varphi_y)} \\ 0 \end{pmatrix} = \left(E_x\, e^{i\varphi_x}\, \vec{\mathbf{e_x}} + E_y\, e^{i\varphi_y}\, \vec{\mathbf{e_y}}\right) e^{i(kz-\omega t)} \tag{5.25}$$

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* $\mathbf{j} \in \mathbb{C}^2$.

$$\mathbf{j} = \frac{1}{\sqrt{E_x^2 + E_y^2}} \begin{pmatrix} E_x \\ E_y\, e^{i\delta} \end{pmatrix} \quad \text{with} \quad \delta := \varphi_y - \varphi_x \tag{5.26}$$

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. A couple examples for components in horizontal position:

$$\begin{aligned} \text{polarizer:} \quad \hat{M} &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\ \lambda/4 \text{ plate:} \quad \hat{M} &= \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} e^{-\frac{i\pi}{4}} \\ \lambda/2 \text{ plate:} \quad \hat{M} &= \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \end{aligned} \tag{5.27}$$

By using matrices one can easily compute the effect of multiple components using the matrix product and find the behavior of a rotated component $\hat{M}_\varphi$ using the standard rotation matrix.

$$\hat{M}_\varphi = \hat{R}(-\varphi)\, \hat{M}\, \hat{R}(\varphi) \quad \text{where} \quad \hat{R}(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \tag{5.28}$$

SASA

$$\hat{S} = \begin{pmatrix} \hat{T}^f & \hat{R}^b \\ \hat{R}^f & \hat{T}^b \end{pmatrix} \tag{5.29}$$

How do symmetries in the MS result in S-Matrix symmetries.

Applying Jones calculus to MS

Here: Under which conditions can one describe the effect of a MS with a jones matrix?

Boundary Conditions

Maybe mark equations used as boundary conditions for the algorithm differently(e.g. fat)

5.3 SASA

Here I want
-Intuition for single Layers: how does the transmission change for different Λ, l, w, ϕ
-Why is the S matrix needed (Fabry-Perot-Effects, ...)

6 Sources

References

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

[2] 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, January 2014.

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

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