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 5, 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
	Institute of Quantum and Nano Optics
	Friedrich-Schiller-Universität Jena

1 Abstract

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 Physical Background

2.1 Meta Surfaces

2.2 The S-Matrix Formalism

2.3 SASA and the Star Product

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

$$\mathbf{E} = \begin{pmatrix} E_x e^{i(kz-\omega t+\varphi_x)} \\ E_y e^{i(kz-\omega t+\varphi_y)} \\ 0 \end{pmatrix} = \left(E_x\, e^{i\varphi_x}\, \vec{\mathbf{e_x}} + E_y\, e^{i\varphi_y}\, \vec{\mathbf{e_y}}\right) e^{i(kz-\omega t)} \tag{1}$$

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

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

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 are for components in horizontal position:

$$\begin{aligned} \text{polarizer:} \quad \hat{M} &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\ \lambda/4 \text{ plate:} \quad \hat{M} &= \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} e^{-\frac{i\pi}{4}} \\ \lambda/2 \text{ plate:} \quad \hat{M} &= \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \end{aligned} \tag{3}$$

By using matrices one can easily compute the effect of multiple components using the matrix product or rotated components using the standard rotation matrix.

$$\hat{M}_\varphi = \hat{R}(-\varphi)\, \hat{M}\, \hat{R}(\varphi) \quad \text{where} \quad \hat{R}(\varphi) = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \tag{4}$$

SASA

$$\hat{S} = \begin{pmatrix} \hat{T}^f & \hat{R}^b \\ \hat{R}^f & \hat{T}^b \end{pmatrix} \tag{5}$$

Boundary Conditions

2.4 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 program the relation NN's are trained on a dataset (X, Y) of correct input output pairs.

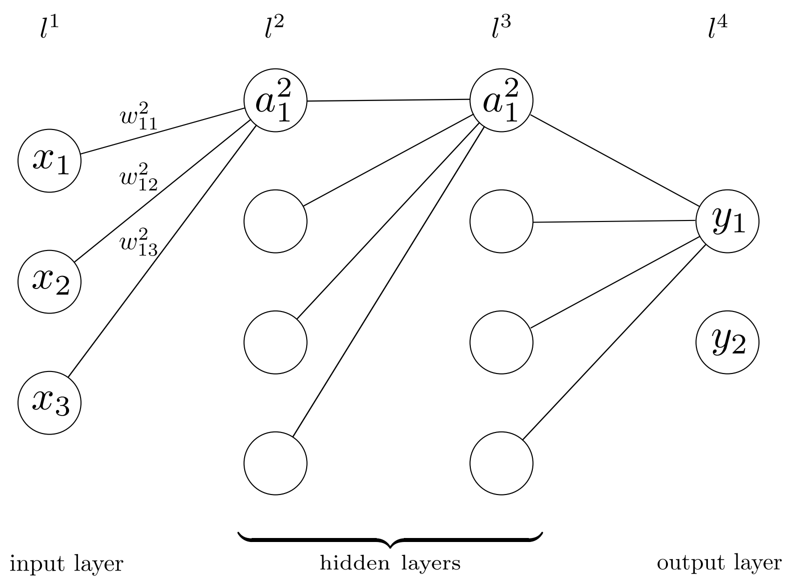


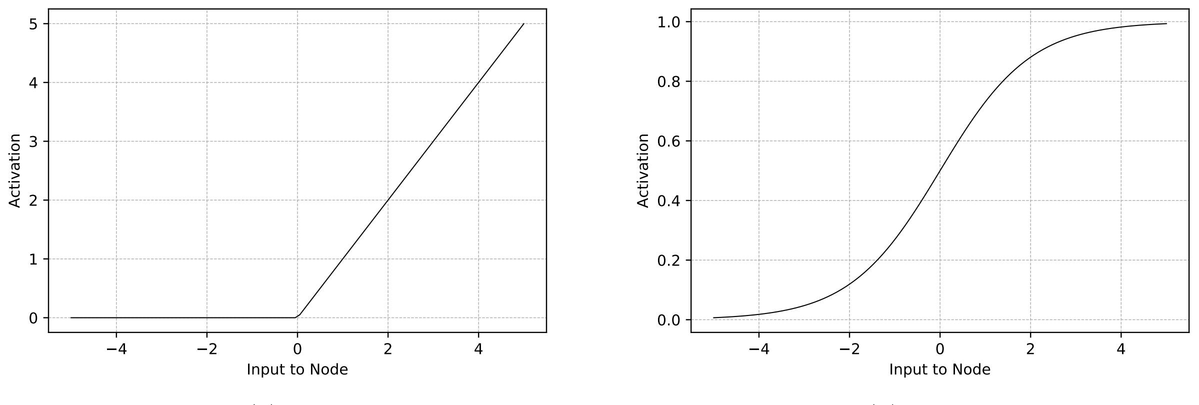
Figure 1: The most simple kind of NN is called densely connected or multilayer perceptron. For clarity only connections to the top most node of each layer are shown.

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 1 that means:

$$a^2_1 = \sigma\left(\sum_i w^2_{1i}\, x_i + b^2_1\right) \tag{6}$$

However it is more convenient to stop considering every node individually and to view the involved quantities as vectors and matrices. So that (6) can be written as:

$$\mathbf{a}^l = \sigma\left(\hat{\mathbf{w}}^l \mathbf{a}^{l-1} + \mathbf{b}^l\right) \tag{7}$$



(a) relu (b) sigmoid

Figure 2: Two examples of activation functions σ. Especially the relu function has a destinct on and off state similar to biological neurons.

Training During training the network output $\mathrm{NN}(\mathbf{x}) := \mathbf{y}'$ is calculated though repeated use of (7) 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 $\mathbf{y}'$:

$$C_{\text{mse}}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{8}$$

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 $\nabla C_{\mathbf{b}^l}$ and $\nabla C_{\hat{\mathbf{w}}^l}$. 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: [?].

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 3). But now the number of weights to the next layer $\hat{\mathbf{w}}^2$ are in the order of $\mathcal{O}\left(n^3\right)$ (<- guess) which soon becomes unfeasible.

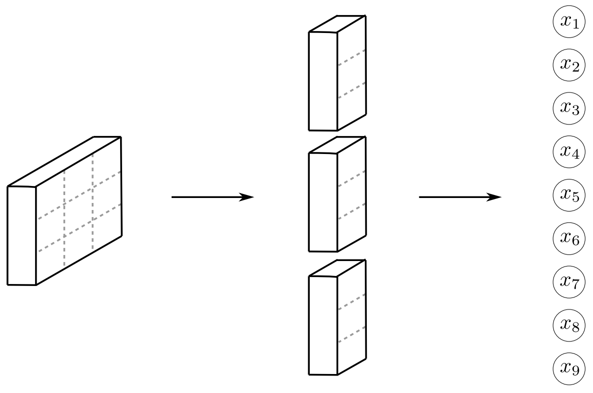


Figure 3: Flattening of a 3×3 matrix to fit the input of a multilayer perceptron.

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.

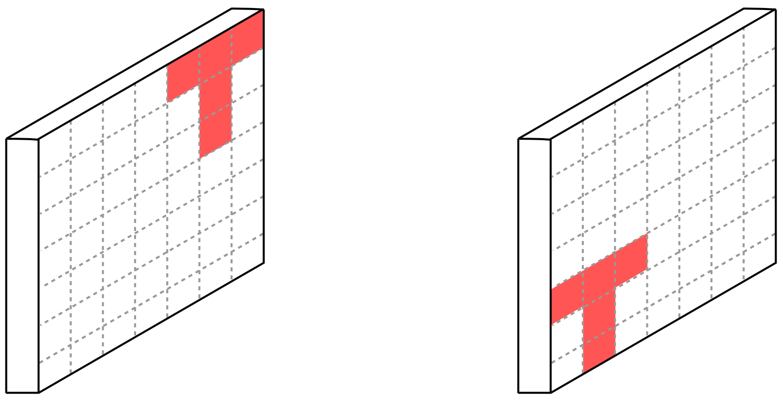


Figure 4: Two pictures of a T at different positions where the red color signifies a high value in the grayscale image. After the flatten operation seen in figure 3 very different nodes are active.

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 5). 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.

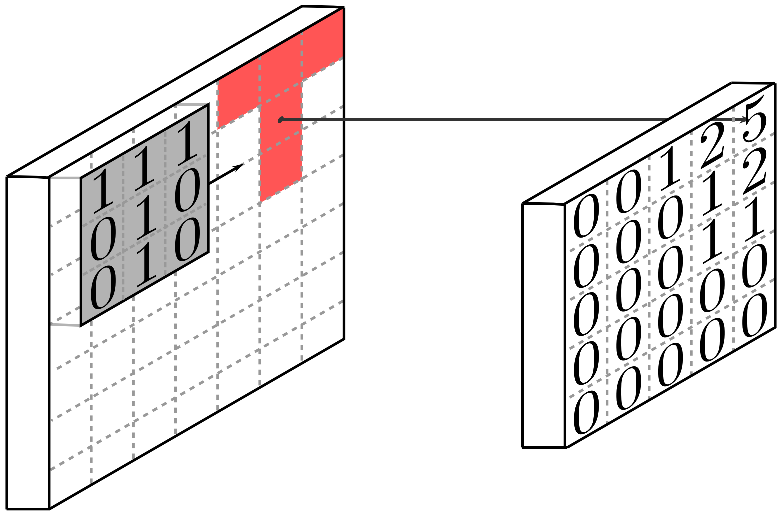


Figure 5: Example of a convolution where white pixel are 0 and red pixel are 1. The "T kernels" feature map is greatest at the position of the original letter.

Pooling Layers For a big image and 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 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.

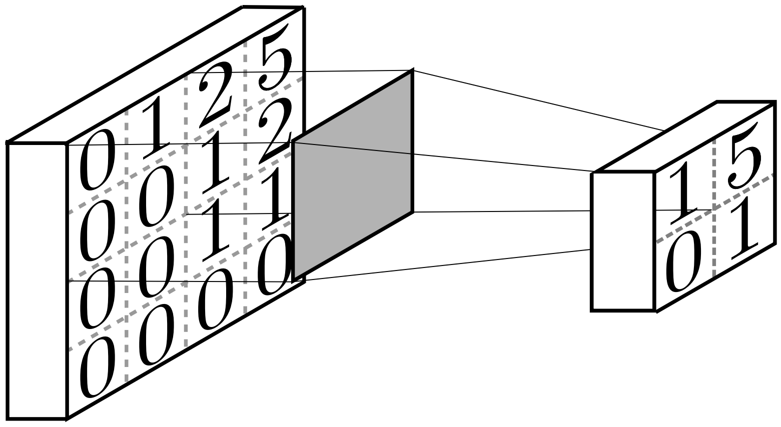


Figure 6: Example of a Max Pooling Layer. For every 2 by 2 field the maximum is calculated. After applying first the convolution and then the pooling layer the information "T in the top right corner" is still there and size of the resulting matrix is very manageable

Example Network Architecture Now all the building blocks for a complete ConvNet are available. Repeatedly alternating convolution and pooling 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.

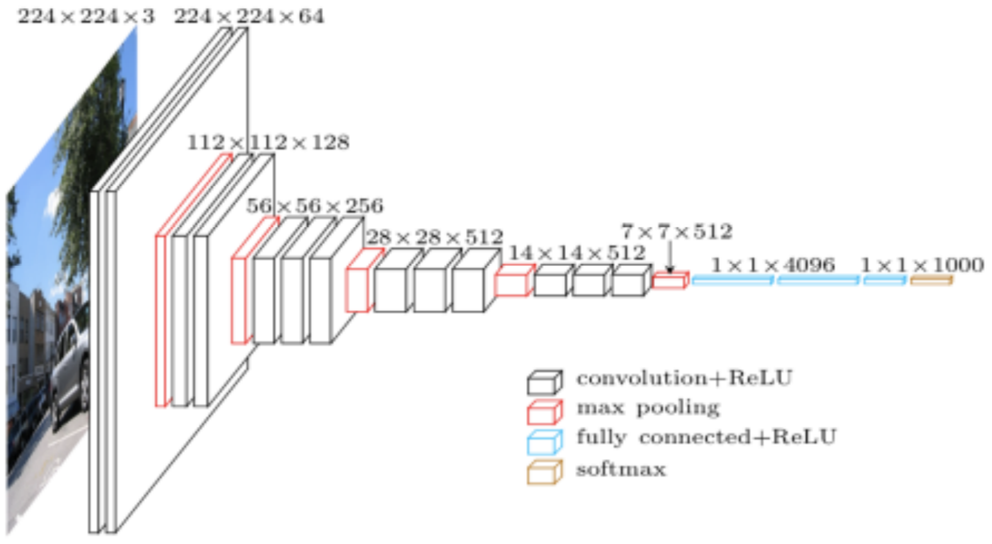


Figure 7: Example for a complete ConvNet. Note the input is in this case an RGB image so there are three layers in the z dimension corresponding to the different colors. In this case the the first layer has 64 kernels of size $3 \times 3 \times 3$. The next convolution then has 128 kernels of size $3 \times 3 \times 64$. [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.

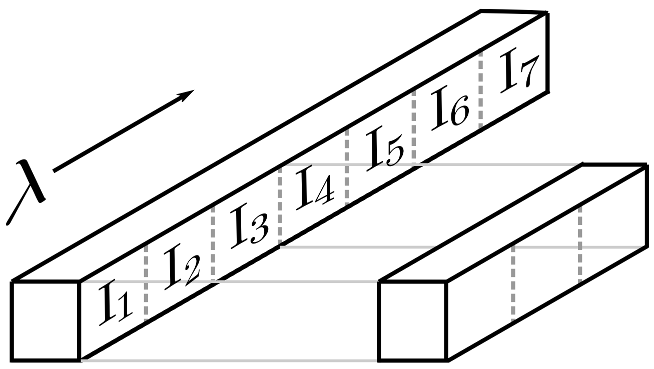


Figure 8: Example of a 1D convolution. A 1×3 kernel is shifted over a spectrum I discretized at 9 wavelengths

3 The Algorithm

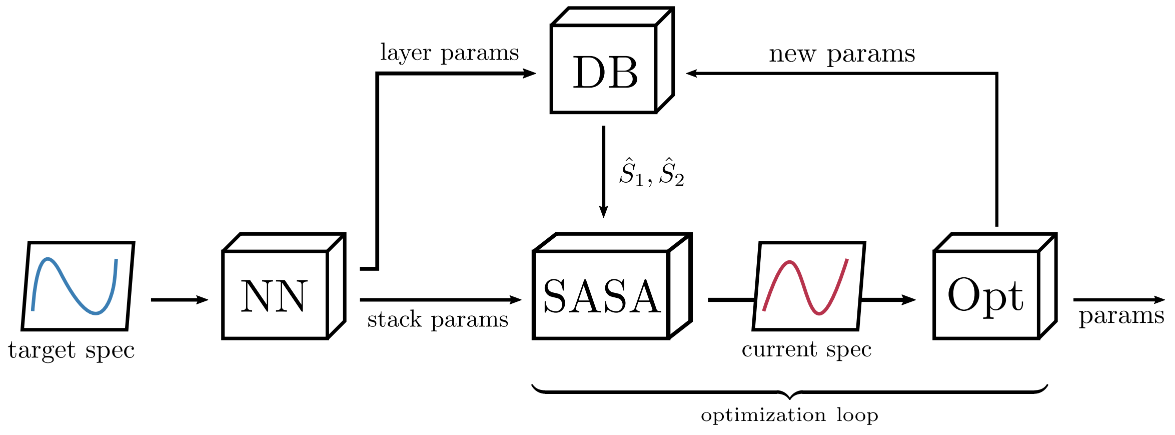


Figure 9: A Flowchart of the Algorithm

NN	convolutional neural ntwork trained to map spectra to stack and layer parameters
DB	database of FMM simulated single layers
SASA	algorithm calculating $\hat{S}_{\text{stack}} = \hat{S}_{\text{stack}}(\hat{S}_1,\, \hat{S}_2,\, ...)$
Opt	optimizer changing parameters to minimize the difference between the current and target spectrum
$\boldsymbol{\hat{S}_1},\ \boldsymbol{\hat{S}_2}$	S-matrices of the top and bottom layer
layer params	these include the geometry of the periodic meta surface cell and the kind of material used
stack params	the rotation angle of the layers to one another and the distance between
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

3.1 Network

3.2 Database

3.3 Optimizer

4 Sources