Working Title: Reverse Design of Meta-surface Stacks via Neural Network

Tim Turan

January 31, 2020

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

$$\hat{S} = \begin{pmatrix} \hat{T}^f & \hat{R}^b \\ \hat{R}^f & \hat{T}^b \end{pmatrix} \tag{1}$$

2.3 SASA and the Star Product

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.

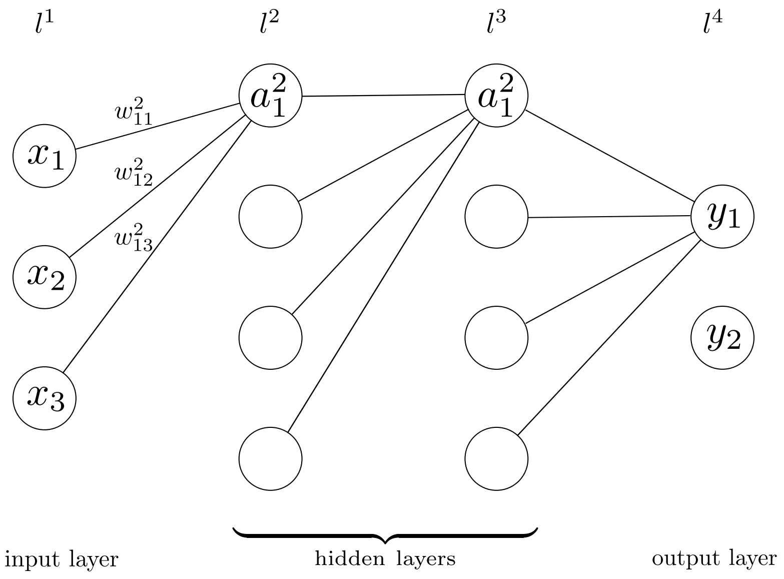


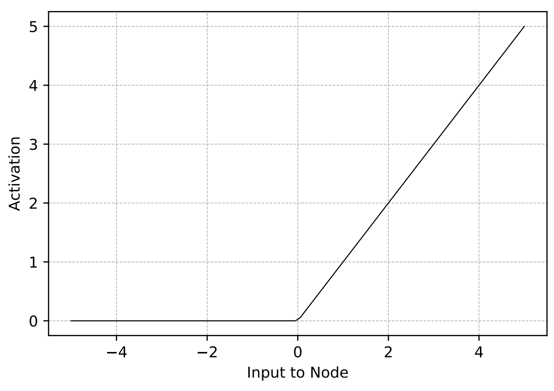
Figure 1: The most simple kind of NN is called densely connected or multilayer perceptron. For clarity only connections to the top most node of each layer are shown.

Densely Connected Neural Networks 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:

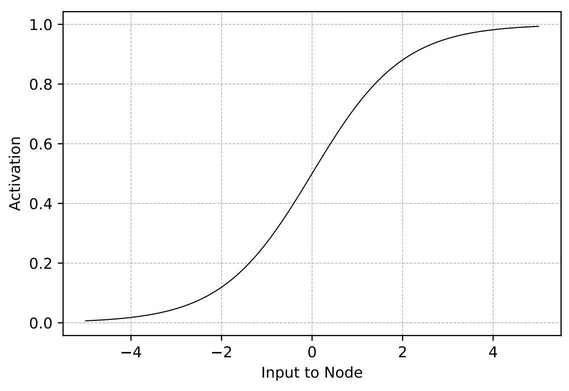
$$a^2_1 = \sigma\left(\sum_i w^2_{1i}\, x_i + b^2_1\right) \tag{2}$$

However it is more convenient to stop considering every node individually and to view the involved quantities as vectors and matrices. So that (2) can be written as:

$$\mathbf{a}^l = \sigma\Big(\hat{\mathbf{w}}^l \mathbf{a}^{l-1} + \mathbf{b}^l\Big) \tag{3}$$



(a) relu



(b) sigmoid

Figure 2: Two examples of activation functions σ.

Training During training the network output $\mathrm{NN}(\mathbf{x}) := \mathbf{y}'$ is calculated though repeated use of (3) 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 $\mathbf{y}'$:

$$C_{\mathrm{mse}}(\mathbf{y}, \mathbf{y}') = \sum_i \left(y_i - y_i'\right)^2 \tag{4}$$

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

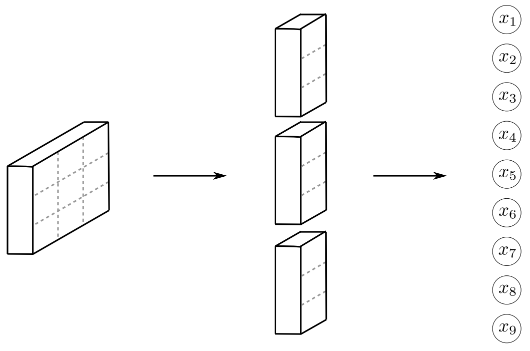


Figure 3: Flattening of a 3×3 matrix to fit the input of a multilayer perceptron.

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.

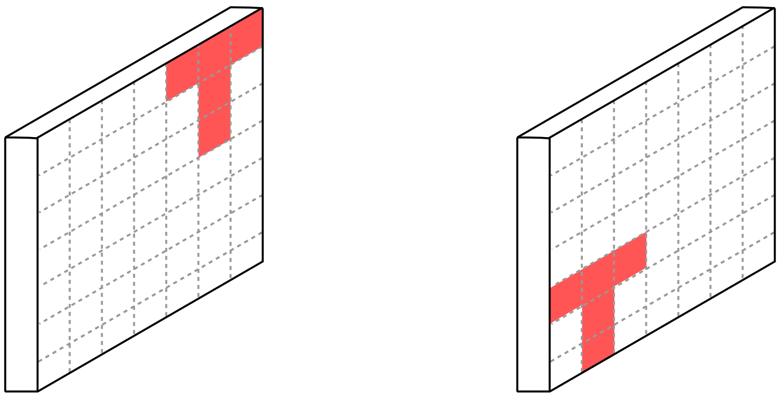


Figure 4

3 The Algorithm

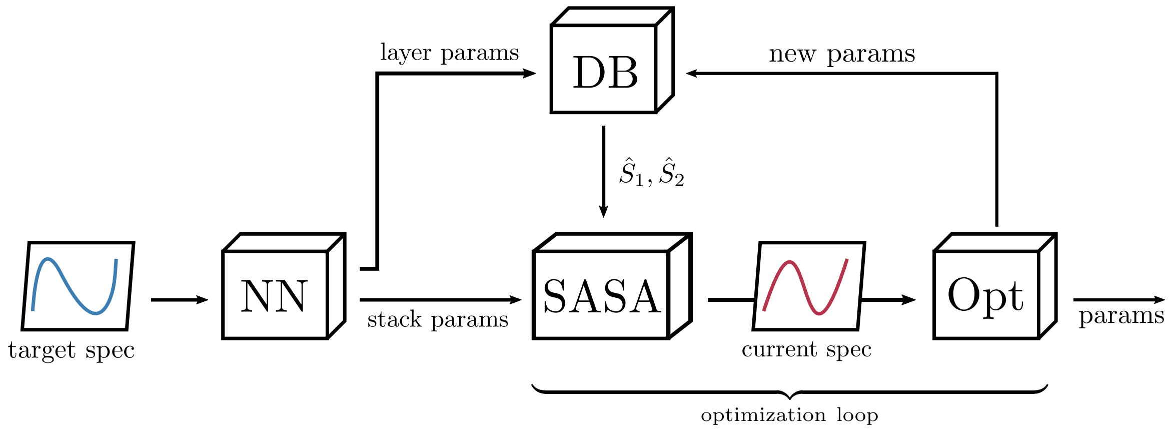


Figure 5: A Flowchart of the Algorithm

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

4 The Neural Network

5 The Optimizer

6 Results

7 Literaturverzeichnis

References

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

8 Anhang