The importance of nonlinearity for physical neural networks

Bram Verreussel

[datum]

Contents

Abstract

This thesis investigates the effects of nonlinearity on physical neural networks as part of the Vaporware network project. The Vaporware network, which has not been implemented yet, will do computations which are done by neural networks using the propagation of laser light. For large enough networks this will speed up the evaluation of these neural networks. In this network the role of a nonlinear activation function is taken on by propagating the light through dense atomic vapour. To get nonlinear effects the frequency of the light has to be near an absorption line and the intensity high enough that the vapour is partially saturated. We simulated the Vaporware model numerically using Keras to see how different values of the susceptibility affected the accuracy of the network. A susceptibility of zero corresponds to propagation in a vacuum and this corresponds to having no activation function. As expected this showed the lowest accuracy for the MNIST dataset. We found a maximum accuracy of 96%. We found that the sign of the susceptibility only matters to a small extent for the accuracy.

1 Introduction

Neural networks, a form of machine learning, have become quite popular over the years and has use in areas ranging from medical imaging to the newest snapchat filter. Neural networks have grown more complex over the years and as such the larger networks take longer to execute and consume more power. Physical networks try to solve this. If the execution of these networks can be performed using an optical setup, then the execution will happen at the speed of light and we are only limited in the readout time. The size of the network will not matter anymore. This would be useful for self-driving cars which rely on huge neural networks and they require fast execution times and low power usage. The Vaporware network is a proposed network which makes use of a laser and multiple spatial light modulators. The pixels of the spatial light modulator are like the weights in neural networks; the pixels are tunable parameters which are continuously changed during the training phase. The training phase will take longer than conventional neural networks but once it is trained the network becomes static and will provide computation essentially as fast as you can read out the the signal. Neural networks consist of layers which usually have linear parts which then feed in nonlinear functions called activation functions. Without these nonlinear functions the network is limited in its capabilities because stacking on linear layers has no effect. These nonlinear functions will be modeled by vapour with nonlinear optical properties. In this thesis we will study the effect of this nonlinear vapour.

2 Theory

2.1 Machine Learning

Machine learning is a set of algorithms that aim to perform tasks with as little human intervention as possible. An example would be image recognition; suppose you have a collection of images and you want to locate any faces that are on them. Before machine learning the usual thing to do would be to hand craft an algorithm that would first detect features like eyes, a mouth, nostrils etc. and would then decide, based on the relative locations of these features, if it is seeing a face. The machine learning approach would be to provide a large data set of faces to the computer and have it 'train' on the data set. This training is done by incrementally adjusting various parameters of the model , which will be explained in more detail in the theory section. The machine learning approach has proven itself time and time again and machine learning has started appearing in numerous disciplines (too many to list).

A notable example of the strength of machine learning is the Alphazero software by google. The goal of Alphazero was to create a general purpose algorithm for playing games. It competed in three games: chess, go and shogi. For each of these games there exists well optimized software that is designed to play the games as well as possible. Basically a more sophisticated version of what you play against when you play 'against the computer'. Especially the chess engine (Stockfish) was optimized very well and has years of research and improvements to precede it. Using only self-play as training it was to able to beat all three engines within a total of 24 hours. This shows the potential of raw computing power in the right setting.

2.1.1 The Perceptron network

Machine learning really started to take of when neural networks were developed. Neural networks are a particular structure that is inspired by how neurons are connected in the brain. Even though the resemblance to actual brains is far sought, neural networks have proven themselves to be useful. The first neural networks used perceptrons which are a very basic model of how neurons fire. A perceptron has multiple inputs and performs a weighted sum on them to determine the output. They are usually depicted as a little circle with every input and output represented by a line, see Fig. 1. Every input is associated with a weight so in drawings of networks every line also represents a weight. By chaining many perceptrons together it is possible to make complex decisions. Suppose one such perceptron has to model whether you want to buy a bar of chocolate or not. In that case it has three inputs which are all numbers: how hungry you are, how much money you have and much you need to lose weight. The input could then look like this (0.9, 0.25, 0.5) where the inputs are scaled so they are between 0 and 1. Not every input is equally important; if you think hunger is very important, give less importance to money and your health, the weights could for instance become (2, -0.5, -0.5). The last two weights are negative because when the inputs are high you are *less* likely to buy the bar. To compute the final outcome you would perform a weighted sum of the inputs and weights: $0.9 \cdot 2 + 0.25 \cdot (-0.5) + 0.5 \cdot (-0.5) = 1.425$. If this sum is bigger than some threshold the output is 1 (buy it) and if it's less than the threshold it's 0 (don't buy it).

We can write this last step as

$$a = \sigma(\mathbf{w} \cdot \mathbf{x} + b), \tag{1}$$

where a is called the activation (a term borrowed from neuroscience) and the weighted sum is written as the dot product a weight vector. The function σ is called an activation function and in this case it is a simple step function:

$$\sigma(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The threshold is then equal to $-b$ since $\mathbf{w} \cdot \mathbf{x} + b > 0$ is equivalent to saying $\mathbf{w} \cdot \mathbf{x} > \textit{threshold}$.

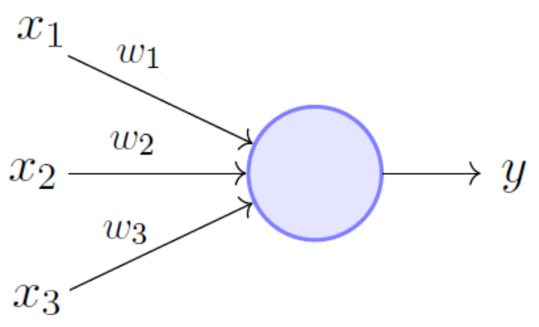


Figure 1: Depiction of a perceptron with 3 inputs. Source: towardsdata-science.com

2.1.2 Layered networks

As mentioned in the previous section, chaining multiple perceptrons can result in networks that make complex calculations. It has been proven even that, under some assumptions, neural networks can approximate arbitrary functions. The most common method to do this is by dividing the networks in layers. Each layer is connected only to the layer before and after it. If every neuron in a layer is connected to every input neuron that layer is called a dense layer. Dense layers have long been the standard and even though there are now alternatives like convolutional layers dense layers are still ubiqitous. An example of a dense network can be seen in Fig. 2. Dense networks are particularly suitable for parallelization. Equation (1) can be extended easily for multiple inputs and multiple outputs. The weight vector is replaced by a matrix and the bias is replaced by a vector, yielding

$$a_i = \sigma(w_{ij}x_j + b_i), \tag{2}$$
$$\mathbf{a} = \sigma(w\mathbf{x} + \mathbf{b}). \tag{3}$$

Using this notation you can write the entire network as function composition. Let a^n denote the *nth* layer of the network. Then the *nth* layer of a network can be writtes as

$$\mathbf{a}^n = \sigma(\mathbf{b}^{n-1} + w\mathbf{a}^{n-1}).$$

Every layer that is not the input or the output is called a 'hidden layer' because the outputs of these layers has no apparent meaning. That is why neural networks are usually treated as a black box. The hidden layers are usually not looked at, although some work has been done at better understanding their role (for example Google Deep Dream). The number of hidden layers can vary greatly. For simple datasets such as MNIST (which will be discussed later) a single hidden layer is sufficient while state of the art networks can have up to a 100 layers.

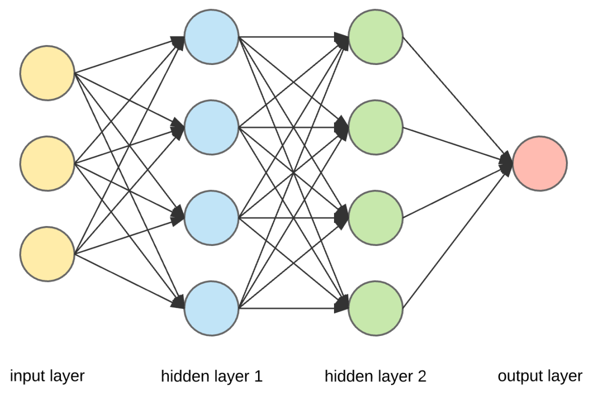


Figure 2: A dense network with two hidden layers. sourec: `towardsdatascience.com`

2.1.3 Backpropagation

Now we know how calculations are done in neural networks there is one thing missing: determining the weights. Determining what weights to use is central to machine learning. This is an optimization problem so this problem is usually casted in a form where gradient descent can be used. To do this we need a 'loss function' that tells us how good the network is performing with its current weights. Let the output of the network be denoted by $\mathbf{y}$. For some input of the network we know how the output should like, let's call this target output y_t. A possible loss function would then be the mean squared error:

$$L = \sum_i \tfrac{1}{2}(y_{t,i} - y_i)^2 \tag{4}$$

If $\mathbf{y}$ is very close to $\mathbf{y}_t$ then the loss is small and vice versa. One can view the loss as a single function of all the inputs and weights:

$$L = f(\mathbf{x}, \{w_{ij}^n\})$$

If the network is using only continuous and differentiable activation functions then it is possible to calculate the gradient with respect to all the weights;

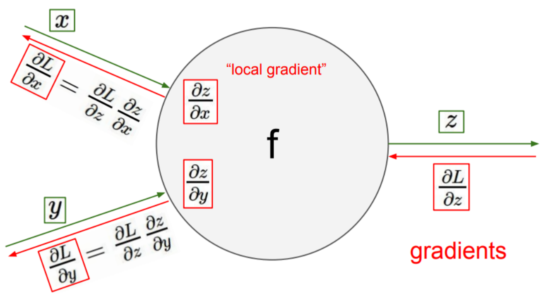
$$\nabla_w L \equiv \frac{\partial L}{\partial w_{ij}^n}.$$

Calculating it is just a matter of applying the chain rule iteratively. Minimizing L is then as simple as moving all the weights against the gradient. The weight update then looks like

$$w_{ij}^n \longrightarrow w_{ij}^n - \eta \frac{\partial L}{\partial w_{ij}^n}.$$

where η is an important parameter called the 'learning rate' that determines how large the stepsize should be. Naively computing the gradient for every weight is very slow. Here is where backpropagation shines. Backpropagation makes intelligent use of the layered structure of neural networks. If you were to calculate the partial derivatives by hand you should notice at some point that you are just multiplying a lot of Jacobians. A neuron gives the same Jacobian to every one of its inputs so for this neuron it has to be calculated only once. A second insight is that a layer depends only on the layer before it, neurons have no dependencies in the layer they are in. So to perform backpropagation you start by calculating the Jacobians in the last layer and them pass them backwards to they can be multiplied with the Jacobians of that previous layer. This continues until every layer is reached. This way each Jacobian is calculated the minimum number of times. A Jacobian is always evaluated at some point, so before calculating the gradient you first have to evaluate the input which is propagated like normal, but intermediate values are saved. This called the forward pass. After the forward pass the gradient is propagated backwards using the precomputed values from the forward pass.The rough idea of backpropagation is shown in Fig. 3.

The update as I mentioned is actually a bit of a simplification. In practice there are many ways to update the weights. The learning rate can change, the magnitude of the gradient vector can be rescaled and momentum can be used, which means the gradient doesn't directly change the weight but instead accelerates the weights. The algorithm that performs the update is called an optimizer and finding good optimizers is still an active area of research.



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure 3: Rough idea of backpropagation for a single neuron. source: `https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-`

2.1.4 Activation Functions

As mentioned before, For backpropagation to work the activation functions should be continuous and differentiable. A small change in the output should mean a small change in the output. This is clearly not the case for the step function. This is why the sigmoid function (also known as logistic function) quickly emerged as a better alternative for the step function.

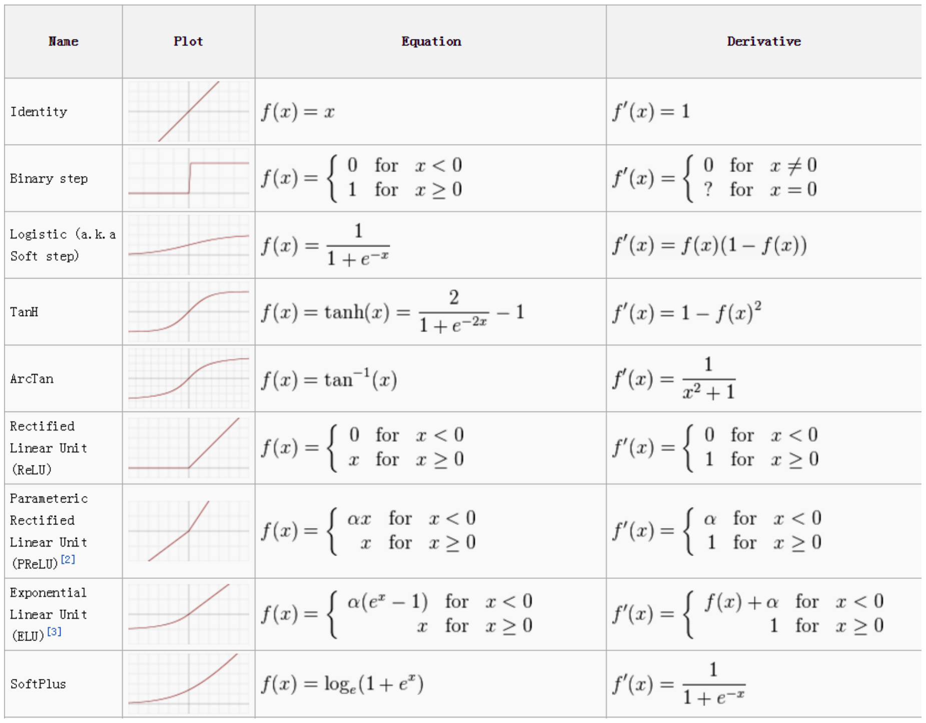
See Fig. 4 for the definition of these functions.

The sigmoid function is basically a smoothed version of the step function. For large negative values it approaches zero. For large positive values it approaches one. A similar function is Tanh, which is a rescaled version of the sigmoid. The range is $[-1, 1]$ instead of $[0, 1]$. The range being centered around zero works slightly better for learning. The sigmoid (together with tanh) was the standard for a long time but was slowly replaced by ReLU because it suffers from the vanishing gradient problem: when the input weights become very large the sigmoid can get stuck in the flat portion of the function. The gradient of the sigmoid becomes extremely small so any further updates also become really small and the neuron becomes 'stuck' at whatever value it is at the time.

The Rectified Linear Unit (ReLU) solves this problem and is therfore very popular in modern networks. The positive part doesn't suffer from the vanishing gradient since the gradient is always constant. Its biggest advantage is its speed. Evaluating the ReLU and its derivative are both very cheap, allowing for more training. The fact that the output is zero for some inputs also means it interferes less with the other neurons (sparse activation), which may be desirable. The gradient for the negative part is zero so there it suffers from a vanishing gradient (for ReLU's this is usually called the dying ReLU problem) but this is often not a big problem if the learning rate is not too high.

If the dying ReLU problem does become severe, it is often battled with a Leaky ReLU, which has a small slope αx for $x < 0$ instead of being zero. Despite being slightly better ReLU's are often still preferred for their simplicity and speed (α is another parameter you have to get right). Parametric ReLU's are Leaky ReLU's with α being learnable. These can perform better but are also harder to train.

There is a myriad of other activation functions which each have their advantages and disadvantages. See for example Fig. 4. In this thesis, we will introduce a new activation function with the main benefit that it can be calculated using a physical setup.



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 4: Common activation functions and their derivatives. Source: `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`

2.1.5 Convolution Layers

In visual neural networks there has been a breakthrough called 'convolutional neural networks'. They make use of the fact that images are spatially correlated, a pixel is often connected to its neighbours but not necessarily to pixels further away. So instead of connecting every neuron to every other neuron each neuron only has as input only a small neighbourhood. Subsequent layers then determine increasingly features. This was inspired by the way neurons are connected in the visual cortex. This small neighbourhood is realized using convolutions. To perform convolution on an image you first select a pixel and then calculate the value of the output pixel with a weighted sum of the neighbouring pixels. The weights are constant for the image and is called a 'kernel'. The kernel is always a rectangle and is often a square with odd dimensions like 3x3 or 5x5 so the kernel can be centered around the input pixel. The convolution operation is shown in Fig. 5.

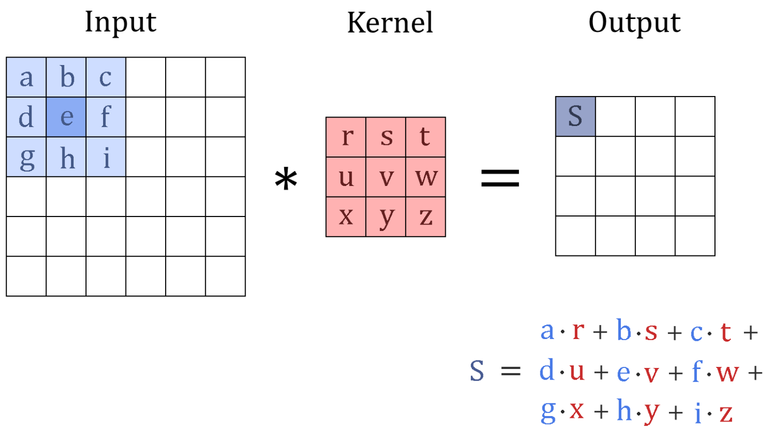


Figure 5: The convolution operation.

2.2 Linear optics

In this thesis the Vaporware network is modeled numerically to look at its behaviour. Since the propagation is done by light some knowledge on nonlinear optics is needed. This will be explained in the next sections.

2.2.1 The Wave equation

To derive the wave equation we start with the following form of Maxwell's equations [4]

$$\nabla \cdot D = 0 \tag{5}$$

$$\nabla \cdot B = 0 \tag{6}$$

$$\nabla \times E = -\frac{\partial B}{\partial t} \tag{7}$$

$$\nabla \times H = \frac{\partial D}{\partial t} \tag{8}$$

where D and B are defined by

$$D \equiv \epsilon_0 E + P \tag{9}$$

$$B \equiv \mu_0 H + M. \tag{10}$$

Here E and H are the electric and magnetic field, ϵ_0 and μ_0 are the vacuum permittivity and permeability and D and B are the electric displacement and magnetic flux. We have assumed here are no free charges or currents in the material (atomic vapour). We also put the magnetization M to zero since its effect will be negligible. If we assume the medium is linearly polarizable we get

$$P = \epsilon_0 \chi E \tag{11}$$

where χ is the electric susceptibility of the medium. This implies

$$D = \epsilon_0(1+\chi)E \equiv \epsilon E \tag{12}$$

By taking the curl of (7) you get

$$\nabla \times \nabla \times E = -\frac{\partial}{\partial t}\nabla \times B. \tag{13}$$

Plugging in (8) gives

$$\nabla \times \nabla \times E = -\mu_0 \epsilon \frac{\partial^2 E}{\partial t^2}. \tag{14}$$

We can then use $\nabla \times \nabla \times E = \nabla(\nabla \cdot E) - \nabla^2 E$ which can be simplified to $-\nabla^2 E$ since the divergence of E is zero. For a derivation see appendix A. This gives

$$\nabla^2 E = \mu_0 \epsilon \frac{\partial^2 E}{\partial t^2}. \tag{15}$$

This has the form of a wave equation so we have proven that, under the assumptions given above, the electric field obeys the wave equation. We can read off the phase velocity and relate it to the vacuum speed of light with (12)

$$\begin{aligned} v &= \frac{1}{\sqrt{\mu_0 \epsilon}} && (16)\\ &= \frac{1}{\sqrt{\mu_0 \epsilon_0 (1+\chi)}} && (17)\\ &= \frac{c}{\sqrt{1+\chi}} && (18) \end{aligned}$$

With v defined like this the wave equation becomes

$$\left(\nabla^2 - \frac{1}{v^2}\frac{\partial^2}{\partial t^2}\right) E(\mathbf{x}, t) = 0 \tag{19}$$

The refractive index is then

$$n \equiv \frac{c}{v} = \sqrt{1+\chi}. \tag{20}$$

With the refractive index we know how fast the light travels at every location and in principle we can propagate from any initial configuration.

2.2.2 Fourier optics

We are now going to look at optical propagation; given the field in an initial plane we want to know the field in some other plane. This plane-to-plane propagation is typical for setups with lasers where the light travels a large distance in one direction (the optical axis) but not so much in the other directions.

The fact that the propagation is in one direction means that we can use the paraxial approximation. The paraxial approximation assumes that the angle between the wave propagation and the optical axis is really small. This has many implications among the fact that the polarisation direction $\hat{e}$, defined by $\mathbf{E}(\mathbf{x}, t) = \hat{e}E(\mathbf{x}, t)$, can be taken constant to a good approximation. This reduces the wave equation to the scalar wave equation given by

$$\left(\nabla^2 - \frac{n^2}{c^2}\frac{\partial^2}{\partial t^2}\right)\mathbf{E}(\mathbf{x}, t) = 0.$$

Using separation of variables it is possible to split off the time dependence.

$$E(\mathbf{x}, t) = \psi(\mathbf{x})T(t) \tag{21}$$

$$\implies \frac{1}{\psi}\nabla^2\psi = \frac{1}{v^2 T}\frac{\mathrm{d}^2 T}{\mathrm{d}t^2} \equiv -k^2 \tag{22}$$

Both sides of equation 22 must necessarily be constant so we define it as $-k^2$. The time dependence is just a second order differential equation so the solution is given by

$$\frac{\mathrm{d}^2 T}{\mathrm{d}t^2} = -(kv)^2 T \tag{23}$$

$$T(t) = e^{\pm ikvt} \tag{24}$$

$$T(t) = e^{i\omega t} \tag{25}$$

where the angular frequency is given by $\omega = vk$. Without loss of generality I picked the positive solution. This solution describes light oscillating at a single frequency $f = \omega/(2\pi)$, this is because separation of variables only works for monochromatic light. Our setup uses a narrow band laser so monochromatic light is a good description. Now we know that the time dependence is a simple phase factor we can ignore it and move on to the spatial dependence. Once we know the spatial solution finding the full solution is trivial. Looking at the spatial part of equation (22) we get the equation known as the Helmholtz equation,

$$(\nabla^2 + k^2)\psi(\mathbf{x}) = 0. \tag{26}$$

Using separation of variables we get analogously

$$\psi(\mathbf{x}) = X(x)Y(y)Z(x) \tag{27}$$

$$\implies \begin{cases} X(x) = e^{ik_x x} \\ Y(y) = e^{ik_y y} \\ Z(z) = e^{ik_z z} \end{cases} \tag{28}$$

$$\implies \psi(\mathbf{x}) = e^{i(k_x x + k_y y + k_z z)}. \tag{29}$$

After plugging this solution into the Helmholtz equation we get the following equation for the wave vector components

$$k_x^2 + k_y^2 + k_z^2 = k^2. \tag{30}$$

This last equation is important because it allows us to eliminate one of the wave vector components. With this restriction it is possible to propagate ψ knowing only a plane as boundary condition, as we will see shortly. By eliminating k_z for example you get

$$\psi(\mathbf{x}) = e^{i(k_x x + k_y y)} e^{\pm iz\sqrt{k^2 - k_x^2 - k_y^2}} \tag{31}$$

The general solution of the Helmholtz equation is formed by taking a superposition of (31) with different values of the wave vector $\mathbf{k} = (k_x, k_y, k_z)$:

$$\psi(x, y, z) = \int d\mathbf{k}\ \Psi(k_x, k_y) e^{i(k_x x + k_y y)} e^{\pm iz\sqrt{k^2 - k_x^2 - k_y^2}} \tag{32}$$

Integrals are assumed to run from $-\infty$ to $+\infty$. This equation almost has the form of a Fourier transform. Let's say we want to propagate along z and we know the solution at a plane located at $z = 0$:

$$\psi_0(x, y) = \psi(x, y, 0) \tag{33}$$

At $z = 0$ the solution is exactly a Fourier transform. Here it becomes

$$\psi_0(x, y) = \int d\mathbf{k}\ \Psi(k_x, k_y) e^{i(k_x x + k_y y)} = \mathcal{F}^{-1}\{\Psi\} \tag{34}$$

where I use the following definition of the Fourier transform:

$$\mathcal{F}\{f\}(\mathbf{k}) = \hat{f}(\mathbf{k}) = \frac{1}{(2\pi)^n} \int d\mathbf{x}^n\ f(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}} \tag{35}$$

$$\mathcal{F}^{-1}\{\hat{f}\}(\mathbf{x}) = f(\mathbf{x}) = \int d\mathbf{k}^n\ \hat{f}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}} \tag{36}$$

Now the power of Fourier optics emerges. Since ψ_0 is the inverse Fourier transform of Ψ we can calculate Ψ using the Fourier transform. This allows us to write the propagation of light as follows:

$$\psi(x, y, z) = \mathcal{F}^{-1}\left\{h(k_x, k_y) \cdot \Psi(k_x, k_y)\right\} \tag{37}$$

$$= \mathcal{F}^{-1}\left\{h(k_x, k_y) \cdot \mathcal{F}\left\{\psi_0(x, y)\right\}\right\} \tag{38}$$

Or put in words: to calculate ψ at some z, first calculate the Fourier transform of ψ at $z = 0$, then multiply by a kernel

$$h(k_x, k_y) \equiv e^{iz\sqrt{k^2 - k_x^2 - k_y^2}}$$

and finally transform back to real space using an inverse Fourier transform. Equation (38) almost has the form of the convolution theorem:

$$f(x) * g(x) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{f\right\} \cdot \mathcal{F}\left\{g\right\}\right\}. \tag{39}$$

Rewriting 38 as

$$\psi(x, y, z) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{\mathcal{F}^{-1}\left\{h(k_x, k_y)\right\}\right\} \cdot \mathcal{F}\left\{\psi_0(x, y)\right\}\right\} \tag{40}$$

makes it an exact convolution

$$\psi(x, y, z) = \mathcal{F}^{-1}\left\{h(k_x, k_y)\right\} * \psi_0(x, y) \tag{41}$$

This has two implications; it means convolution, a useful mathematical operation, can be performed by light propagation. Although only for this specific kernel. This is one of the driving reasons for the Vaporware network. It is also means it is possible to simulate light propagating through our setup using convolutions. Convolutions are often well optimized, especially in software for training neural networks. So this is good for execution speed.

It is convenient to be able to shift the kernel. The numerical implementation of the 2D fourier transform for is often has the origin in the top left corner which is not desirable. For a convolution the origin of the kernel needs to be in the center. The sifting property of convolution,

$$f(x - x_0) = f(x) * \delta(x - x_0), \tag{42}$$

can be used to shift $\mathcal{F}^{-1}\left\{h(k_x, k_y)\right\}$, which has coordinates in real space, to arbitrary locations. After applying the convolution theorem once again this results in multiplying h by the fourier transform of $\delta^2_{x_0,y_0} = \delta(x - x_0)\delta(y - y_0)$:

$$\psi(x, y, z) = \mathcal{F}^{-1}\left\{h(k_x, k_y) \cdot \mathcal{F}\left\{\delta^2_{x_0,y_0}\right\}\right\} * \psi_0(x, y). \tag{43}$$

The discrete version of this equation is how (linear) wave propagation is implemented in the Vaporware network. The expression that is convolved with ψ_0 is also the impulse response; the response of the system to a delta function. As a nice bonus this shows the connection between fourier optics and Linear Time Invariant theory, which also uses impulse responses.

—–

Maxwells equations -¿ dependence of n on chi propagation of fields (fourier optics)

nonlinear optics: nonlinear dependence of P on E propagation using nonlinear n helmholtz equation spectral method-¿ kernel

self focusing?

2.3 Nonlinear Optics

The optics described in the previous sections are linear; the electric field obeys the wave equation which is a linear equation. Consequently the fields obey the superposition principle, which means a linear combination of two solutions to the wave equation is also a solution. Linear optics is applicable too many situations in the everyday life, but when the intensity of the light is too high like for high-intensity lasers this breaks down. The superposition principle is no longer obeyed and this introduces headaches but also a ton of interesting phenomena. Neural networks rely on nonlinear functions to work. That is why this nonlinear regime has to be explored for the Vaporware network. Fundamental to this discussion is the equation for the polarization

$$P(\omega) = \epsilon_0 \chi(\omega) E(\omega). \tag{44}$$

Where χ generally depends on the frequency. This equation tells how the atoms and the field interact. The field induces dipoles by moving apart the charges in the atoms. These dipoles in turn modify the field. The susceptibility is generally a complex number:

$$\chi = \chi' + i\chi'' \tag{45}$$

The imaginary part is associated with damping or amplification of the wave. You can see this by expanding for small χ

$$\begin{aligned} k &= \frac{\omega}{c} n && (46)\\ &= \frac{\omega}{c}\sqrt{1+\chi} && (47)\\ &\approx \frac{\omega}{c}(1 + \tfrac{1}{2}\chi) && (48) \end{aligned}$$

and noticing that plugging in this n in the plane wave solution $e^{i(\omega t - kz)}$ leads to an extra factor $\propto e^{zn''}$ Classically these dipoles can be imagined as charges attached on springs, which would allow us to calculate the positions of the charges and thus the dipole moment $\mathbf{p} = q\mathbf{d}$ with $\mathbf{d}$ the distance vector between the charges. The dipole moment is related to the polarization density simply by $P = Np$ with N the amount of dipoles per volume.

For a more accurate description some quantum mechanics is necessary. The dipole moment is replaced by the dipole operator

$$p = \langle q\mathbf{x} \rangle = q\langle \psi | \mathbf{x} | \psi \rangle. \tag{49}$$

It is important to note that for the regular atomic states this always zero due to symmetry of the wavefunctions. Only when the atom is in a superposition of states can it have a dipole moment. We'll study a two level system, so we consider only the ground state and first excited state of the atom. This captures a lot of the dynamics without too much theory. The dipole moment occurs only with an oscillating electric field since otherwise the atom would decay quickly to the ground state. If we consider the nonlinear response of this two level system to a sinusoidal driving frequency it can be shown that

$$\chi'(\omega) \propto \frac{f(\omega)}{1+I/I_s} \cdot \delta \tag{50}$$

$$\chi''(\omega) \propto \frac{f(\omega)}{1+I/I_s} \tag{51}$$

where δ is the difference between the frequency ω and the resonant frequency (δ is also called detuning), I is the intensity of the light and I_s is a constant called the saturation intensity. The derivation of this is beyond the scope of this thesis so we will refer to [2] or [5]. For high intensities the medium becomes saturated; at this point the number of atoms in the excited state is equal to the atoms in the ground state. Additional light cannot be absorbed anymore and the medium will appear transparent. The dependence of I is the reason for the nonlinear propagation. The intensity depends on the electric field by $I = |E|^2$ and since the propagation of $\mathbf{E}$ now depends on itself the linearity no longer holds. For the rest of this thesis we will focus on the real part of the susceptibility for reasons that will become apparent. The total effect is that for high intensities the extra phase factor, which is picked during propagation when $n \neq 1$, is diminished and the medium acts more like a vacuum.

3 Simulation

3.1 The MNIST dataset

The MNIST dataset is a large, labeled collection of images of handwritten digits. It is a modification of the original NIST dataset that extended and improved it. The images are 28x28 grayscale images. The MNIST dataset is relatively easy to train because it is so well behaved. For that reason it is often called the 'hello world of machine learning' since it is usually the first dataset you encounter when learning machine learning. In this thesis we use the MNIST dataset because its simplicity gives us a clearer picture of the performance of the network.

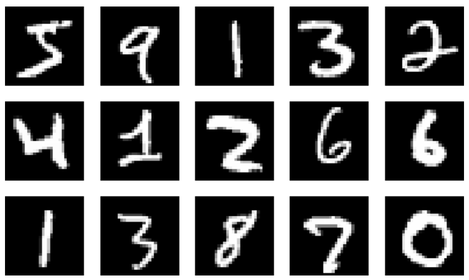


Figure 6: Small subset of the MNIST dataset.

The dataset contains 60 000 training images and 10 000 validation images. During training the validation images are never shown to the network and this is necessary to detect 'overfitting'. Neural networks often suffer from overfitting because they have so many parameters. It occurs when the network learns specific features of a single image to classify it correctly instead of learning more general features that are applicable to other digits as well. The equivalent would be a student learning all the answers to a multiple choice by heart without actually understanding the questions. Because this is so common it is important to have validation data to detect when it happens. Validation accuracy measures how accurate a neural network is at predicting these validation images and this is the most important metric when discussing performance.

3.2 Keras

Keras is a high level machine learning API that aims to make coding neural networks easy and concise while still allowing for flexibility. This flexibility is important for us because we want to define custom layers that can model the physical processes. Keras is built on top of three, more low-level API's: either TensorFlow, CNTK, or Theano. We chose Tensorflow for this project since it is the most standard. Keras adds functionality to these low-level API's that makes it more abstract and less cumbersome to use.

Creating a model in Keras consists of three steps: creating the network (also called 'model' in Keras), compiling the model and finally performing the training. Creating the model looks as simple as

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dense(10,  activation='softmax' ),
])
```

Here sequential means the model contains only stacked layers, each layer only feeds into the layer after it. There is one hidden layer which has 28x28=784 inputs since each image in the MNIST dataset has 28x28 pixels. The input layer is not explicitly defined in the model. Finally there is an output layer with ten units for each of the classes. The output is in this case 'one-hot encoded'. The output 'three' is represented by a list of ten zeros where the third zero is replace by a one. The Softmax activation ensures that the output is between zero and one. Compiling the model looks like

```
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

This call does a couple of things. Firsly it specifies the loss, which tells you how large the error is. The 'categorical crossentropy' is a loss function specifically designed for one-hot encoded outputs. Secondly it specifies the optimizer. Adam is an optimizer with variable learning rate that generally performs well. The metrics parameter defines which metrics are saved during training but has no effect on the performance.

Now the model is ready to train on the data. The standard MNIST dataset is used for training. The images are normalized such that a pixel value of zero corresponds to black and one corresponds to white. The class labels are one-hot encoded. To train the network you would use

```
model.fit(data, labels,
          epochs=10,
          batch_size=128,
          validation_data=(testdata, testlabels)
)
```

The model will train for 10 epochs and one epoch is equal to a full iteration over the dataset. The validation data is never used for training so the model can be benchmarked against data the network has never seen. The batch size determines how many images get taken into account in a single gradient update. Larger batch sizes train quicker because Keras can more efficiently make use of the GPU, but smaller batch sizes tend to improve more per epoch and generalize better.

3.3 Simple network

3.3.1 Importance of nonlinearity

When training neural networks one should take care in picking the right activation functions. One of the important properties of an activation function is that it is nonlinear; if the activation function is linear the entire network will be essentially a single matrix multiplication. When you compose multiple linear operators the result is a single linear operator. If we take the identity operator as the activation function our network will look like this:

$$\mathbf{y} = \mathbf{b}^N + w^N(\mathbf{b}^{N-1} + w^{N-1}(\dots(\mathbf{b}^0 + w^0\mathbf{x}))) = \tilde{\mathbf{b}} + \tilde{w}\mathbf{x}$$

where $\tilde{b}$ and $\tilde{w}$ are the result of multiplying everything out. This means that our network which has 784 inputs and 10 outputs can never perform better than a 784x10 matrix with bias vector[1], regardless of the size of the hidden layers.

Linear networks do show some modest results, Lecun [1] reported a validation accuracy of 88% for the MNIST data set. Nonlinear networks are far superior though and can easily get accuracies of over 98%. Because wave propagation is linear we hypothesize that we need to add a nonlinear medium in our setup to get decent results.

To inverstigate the effect of nonlinearity on the accuracy we will first look at a simple dense network. To be able to smoothly transition between a linear

[1]You can write an N-dimensional translation (bias) as an (N+1)x(N+1) matrix. So you could also say that the network performs as good as a 785x11 matrix.

and non-linear activation I will use a leaky ReLU with a constant parameter:

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & \text{otherwise} \end{cases}$$

When $\alpha = 0$ this becomes a regular ReLU and when $\alpha = 1$ this becomes the identity function (linear activation). An important question comes to mind. Is the accuracy a continuous function of α? Do we get a big jump in performance when α goes from 1 to 0.99 or does it transition smoothly? With $\alpha = 0.99$ the network is technically non-linear but it is not obvious if the strength of the nonlinearity matters.

3.4 Simple model with Softmax

3.4.1 Setup

To answer the question mentioned in the previous section we used the following model:

```
model = Sequential([
    Dense(128, activation=LeakyReLU(alpha=slope),
        input_shape=(784,)),
    Dense(10,  activation='softmax' ),
])

model.compile(loss='logcosh',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Here 'slope' is a variable that is defined outside this code block to change the behaviour of the network. It is the same parameter as α in the previous section. Instead of the usual loss and optimizer we have chosen for 'logcosh' as loss and 'rmsprop' for optimizer. Because the network is linear for $\alpha = 1$ it was harder to train than your standard network and preliminary testing showed this loss and optimizer performed the best.

3.4.2 Results

The experiment was performed by changing α and by completely rebuilding the model each time to ensure everything was reset between runs. The accuracy as a function of α is shown in 7.

This plot answers the question asked in the previous section. The graph is continuous. This means that the network is not only sensitive to the presence of a nonlinearity but also to the strength of the nonlinearity. We can't just pick any nonlinearity and call it a day. We have to also pick the right scale for the model to train well. I suspect this will be the same for the Vaporware model. When we look at the shape of this curve it looks like an asymmetric

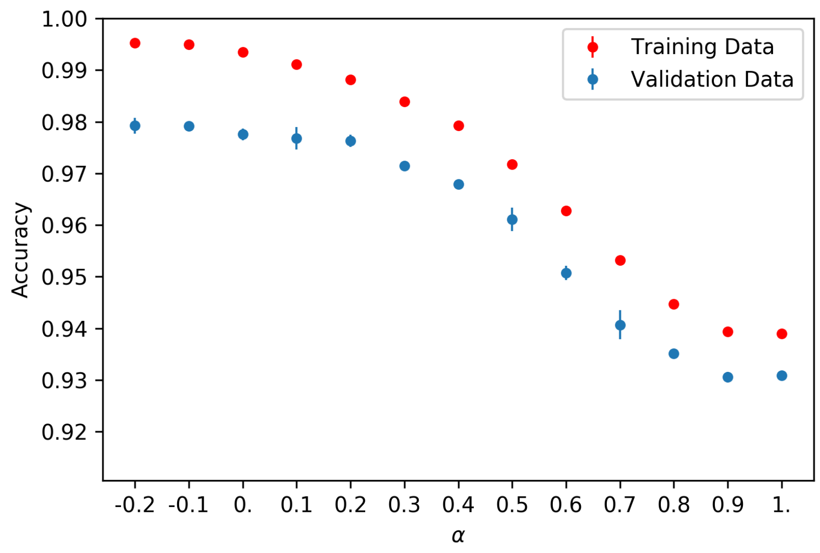


Figure 7: Performance of a dense 784x128x10 network with Leaky ReLU activation in the hidden layers and Softmax activation in the final layer. The accuracy is plotted against α, the Leaky ReLU parameter. The value $\alpha = 1$ corresponds to a linear activation and $\alpha = 0$ to a nonlinear activation (standard ReLU).

sigmoid (steeper towards $\alpha = 1$). We can also see the network shows significant overfitting; the training data performs a lot better than the validation data. The number of neurons in the hidden layer (128) is quite high for the MNIST data set. We chose this number because this gave clean results. Our focus is on investigating the effect of nonlinearity.

The performance of this network is actually too high in the linear regime when compared to Lecun's cited values. Two linear networks should produce the same results. We suspected this was due to the presence of the Softmax activation in the last layer so we will investigate this further in the next section.

3.5 Simple model without Softmax

3.5.1 Setup

In the previous section we trained a model that could interpolate between linear and nonlinear. The last layer has a Softmax activation. The reasoning behind this activation was as follows: the Softmax doesn't change the outcome (the largest value stays the largest value) so it shouldn't have impact on the accuracy. The network should still have the same accuracy as a purely linear network. To test this assumption we modified the last layer to have the same activation function. So the last layer of the network is now `Dense(10, activation='LeakyReLU(alpha=slope)')` instead of `Dense(10, activation='softmax')`. Now the network can become genuinely linear.

3.5.2 Results

The same experiment was repeated with the new network. The result is shown in Fig. 8. In this graph you see behaviour similar to the previous experiment, but this time the accuracy for $\alpha = 1$ is much lower. This accuracy is in line with the value reported by Lecun. So we can conclude that adding a Softmax activation can improve the performance of the network. This is important because the Vaporware network also has a Softmax as final activation. The inset of Fig. 8 shows the same plot but with a slightly bigger range of α. For $\alpha = 0$ the performance is significantly worse than the other values. We suspect this is a combination of the lack of normalization in the last layer and the dying ReLU problem. Normally the Softmax would rescale the values to be between zero and one, but here the weights in the last layer can grow unchecked as long as they provide the right answer. Large weights backpropagate large gradients and the dying ReLU problem is sensitive to large gradients. This is just speculation and a definitive answer would require a more thorough investigation. Since this behaviour is relevant for the ReLU but not for our narrative we will focus on $\alpha > 0$.

We also fitted a power law on the fully linear network to get a rough idea of how the accuracy behaves. This fit is shown in Fig. 9

3.6 Vaporware Network

3.6.1 Physical Setup

The setup will consist of a laser source, three spatial light modulators (SLM's), two vapour cells and a camera to read out the signal. To send input to this network one should imprint the image in the laser light. The light then bounces of the SLM's, which allow control over the light, and passes through the atomic vapour. Finally the light is focused in ten spots. Whatever spot lights up is the prediction of the model. If the third spot lights up the network predicts the input was an image of a 3. To get an idea on how to imprint the light you could cut out the digit from a piece of cardboard and have the laser pass through it. A more precise way would be to imprint it via intensity modulation either on the first SLM or on an additional SLM between the laser and first SLM. An SLM is like a mirror that is divided into pixels. Each pixel provides control over either the phase or the intensity of the light. Control over both phase and intensity is also possible although for current SLM's this restricts the attainable range of values [3]. The vapour cells are filled with rubidium vapour and will provide the nonlinear behaviour of the light. Rubidium is chosen because lasers with a narrow linewidth are readily available for the absorption lines. It is important to

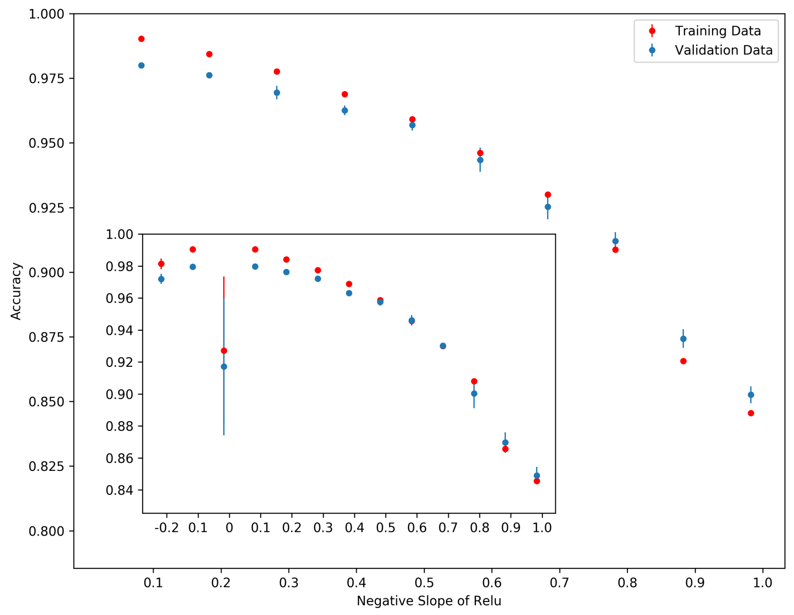


Figure 8: Performance of a dense 784x128x10 network with Leaky ReLU activation in the hidden layers. The accuracy is plotted against α, the Leaky ReLU parameter. The value $\alpha = 1$ corresponds to a linear activation and $\alpha = 0$ to a nonlinear activation (standard ReLU). The inset is the same as the big plot but shows a slightly bigger range of α

keep the absorption as small as possible, but any nonlinear behaviour necessarily has to be near an absorption line (small detuning). Finally the camera will read out the intensity in the output plane and location of the brightest spot will be the output of the network. A schematic of the setup can be found in Fig. 10

3.6.2 Model

We will now elaborate how this physical model is implemented in Keras. The components are modeled as layers and are then stacked using `Sequential`. The output of each layer is plotted in Fig. 11 to get a better understanding of the model. The following layers were used.

SLM layer The first two SLM layers perform a pure phase rotation. This rotation can range from 0 to 2π. The rotation of each pixel is a learnable parameter. The last SLM is presented differently, it is called matrix multiply. In the setup there is a large distance between the SLM and the camera and the

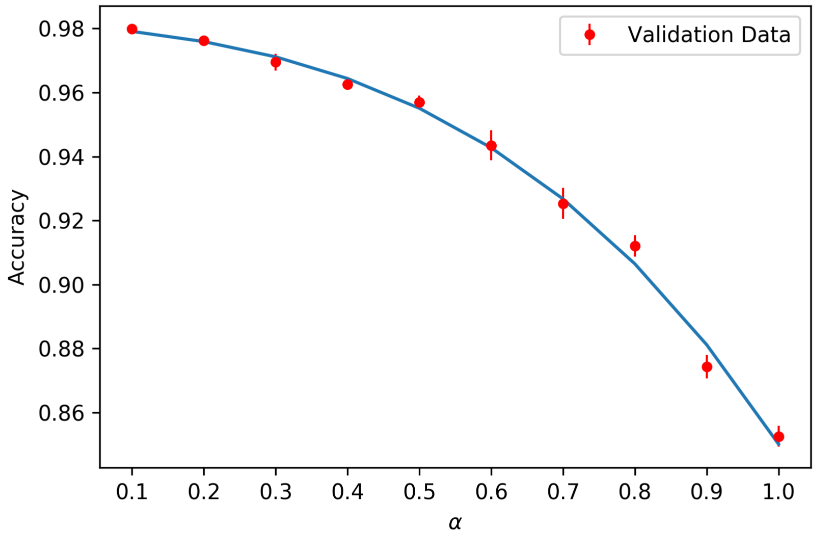


Figure 9: Power law fitted to the performance curve for the simple network. The accuracy is plotted against α, the Leaky ReLU parameter. This curve follows $y(x) = 0.98 - 0.025(x + 0.52)^{3.92}$
. More precise values can be found in Appendix B.

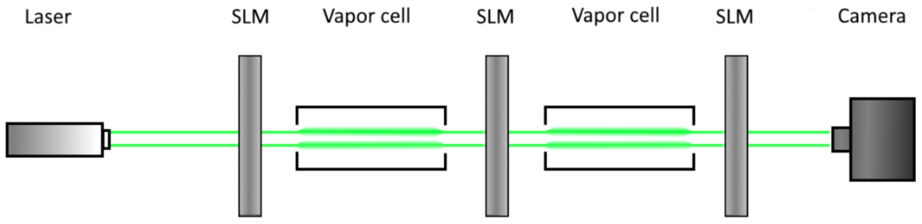


Figure 10: Setup of the Vaporware network. Image by Sybren Huitink.

light is focused in a relatively a small area. The intensity of the light at the a single spot on the camera is a superposition of all the lightrays that come from the SLM. We assume that the SLM has enough resolution that we have full control over this superposition (this time we control intensity as well) and as such we can write it as a matrix multiplication. This is a crude approximation of the SLM but it keeps the model simple. It is possible for an SLM to shape an arbitrary wavefront to a point (see wavefront shaping) so this approximate model has some credibility.

Propagation layers The propagation layer propagates the wave through space. The saturation of the vapour is approximated using a split-step method.

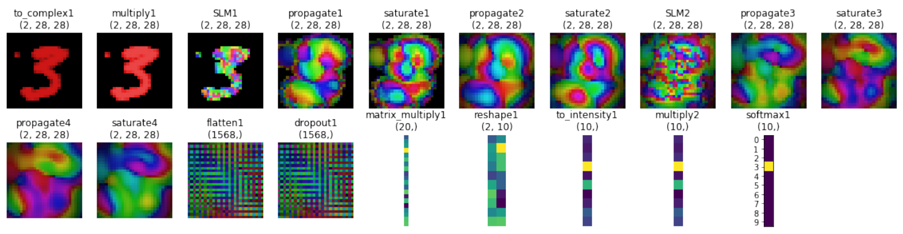


Figure 11: Output of all the different layers of the Vaporware network. The output consists of complex numbers except for the last 5 layers which are just real numbers. The brightness indicates the absolute value of the complex numbers and the hue indicates the phase. Under the layer names you can see the dimensions of each layer.

Instead of modeling the propagation through saturable vapour exactly, the light is first propagated through free space and then the saturation effects are applied. By alternating these two, the nonlinear propagation can be well approximated if the steps are small enough. The number of propagation layers is relatively small, only two free propagation and two saturation layers between the SLM's. Training slowed by a lot when more layers were added. This could perhaps be improved later for better accuracy.

Intensity layer The output of the last SLM is a complex number representing the phase and magnitude of the light. The `to_intensity` layer takes the square of the magnitude to compute the intensity, which is ultimately what is measured by the photodiodes.

Multiplication layers These layers just multiply every pixel with a scalar. As mentioned in the simple network section, the scale of the data is important for learning. In our network it modifies the strength of the nonlinearity. It also modifies the behaviour of the softmax. The first multiply layer multiplies by 7 and the second layer by 0.4. These values were found by first making these values trainable and then looking whether they had changed during training. If they had changed by much the model was trained again with a large step in the direction of these new values, because the parameters can only change so much during training. The input data was normalised to be scaled between 0 and 1.

Dropout A dropout layer randomly disables neurons during training. This does not have a physical representation but it speeds up training and helps the model generalize better.

Softmax layer The softmax layer is a postprocessing layer that helps learning. For a definition we will refer back to Fig. 4 The softmax is sensitive to scale: in the limit of large inputs it behaves as a max function (the output is 1 for the largest value, 0 for the other values) and in the limit of small values every output is the same (every output is $\frac{1}{10}$). If the wrong scale is chosen the gradients will vanish, because the max function and the constant function both have zero gradients. That is why there is multiply layer just before the softmax.

3.6.3 The saturation layer

The saturation is one of the two propagation layers. Since this layer is our topic of interest we will elaborate further upon it. This layer represents an additional phase factor that is picked up during propagation:

$$\psi(x, y, z + \Delta z) \approx \psi(x, y, z) \exp\left(i\Delta z \frac{\sigma}{1 + I/I_s}\right)$$

Here σ is a loosely defined parameter that defines the strength of the saturation behaviour. We will call this parameter the susciptibility even though it is not exactly equal. In the simulation we define the intensity in units of I_s so I_s can be taken as 1.

In general there will also be extinction; loss of intensity due to interactions with the material. Here we have taken the extinction as zero because pure phase rotation is possible experimentally at least to a reasonable approximation. If we allow extinction then the intensity of subsequent layers will decrease which is detrimental to the training process: layers with small intensity also pass on small gradients while the first few layers get disproportionately large gradients.

If we let the σ go to zero then the phase factor becomes just one. In that case no saturation takes place. The entire propagation is then linear. The propagation in free space is written as a convolution with a kernel matrix and since convolutions are linear, so is the propagation. The SLM-layers are also linear because they multiply each pixel with some phase factor, which is the same as multiplying by a constant. Therefor, σ is the parameter that determines the nonlinearity of the propagation. The layer that converts to intensity is also nonlinear together with the Softmax layer. The nonlinearity of the propagation is important because a great part of the computation occurs during propagation. If the propagation were linear it would greatly reduce the attainable complexity of the network.

3.6.4 Results

We have determined the dependency of the accuracy on σ. Doing this was quite straightforward: we let σ take on multiple values and for each value determined the accuracy of the model after a set number of epochs. The same network as in Fig. 11 was used. The result of this experiment is shown in Fig. 12.

A few interesting things can be seen from this figure. Firstly we see a sharp dip at zero, this is luckily what we expected. Adding nonlinearity makes the network better. Secondly the graph is quite symmetric around zero. The accuracy improves slightly faster when going towards negative susceptibility, but this effect is much smaller than we anticipated. A negative susceptibility corresponds to self focusing of the light in the medium and it was expected that information propagated better through a negative susceptibility. A positive susceptibility tends to defocus the light which might result in information loss. So to our surprise we see that this model predicts comparable outcomes for both negative and positive susceptibilities. Finally we see that there is a maximum

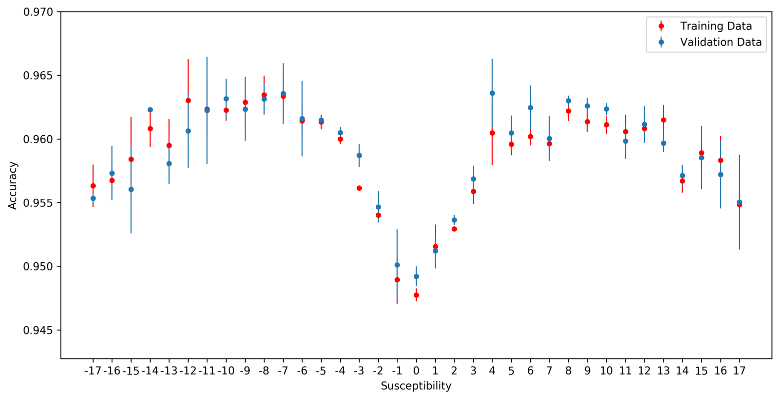
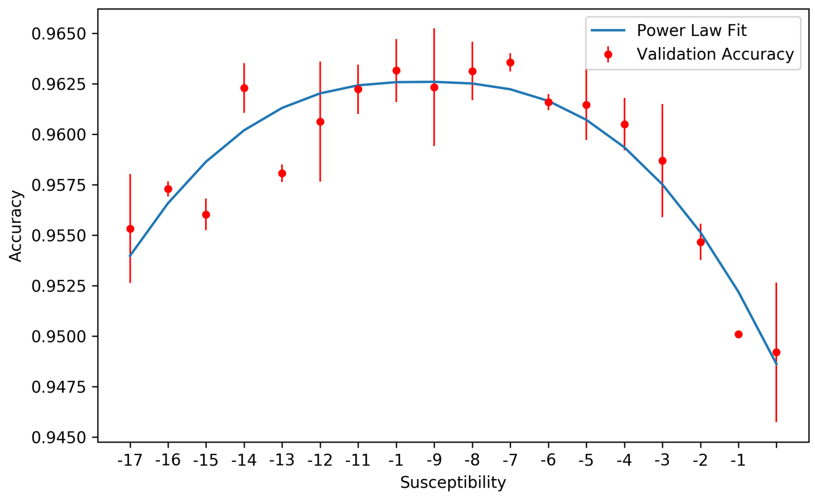


Figure 12: Performance of the Vaporware network for different susceptibilities. Every measurement was repeated three times; the errorbars indicate the standard deviation of those three measurements. A susceptibility of zero corresponds to a completely linear propagation.

at $\sigma \approx \pm 9$. A susceptibility bigger than this will cause very large phase rotations that are hard to control and cause big gradients. This value can't be translated yet to a physical value. In this simulation not enough care was taken of the units to assign to this value a physical susceptibility. Either more units have to be taken into account or the optimal value has to be found through experimentation. Note that we are free to rescale the susceptibility as we like. In the simulation σ only occurs next to Δz and we can rescale both such that their product remains constant.

A power law was also fitted to the Vaporware network. Two fits were used to account for the positive and negative portions of the susceptibility. The positive and negative part both follow a different power law, indicating there is at least some difference between positive and negative susceptibility. The fits are shown in Fig. 13.

These two fits tell us more about the accuracy dependency. The value for n is slightly lower than for the simple network: 2.5 and 2.9 compared to 3.92. This means that near the maximum the drop-off is quicker, so the optimum is harder to find. The optimum is still broad enough though that this shouldn't be a problem. Finally we see that the maximal accuracy (according to the fit) occurs at 96%. This value is not particularly impressive compared to conventional networks, but our aim is not to improve upon conventional networks. Our aim is to achieve good rates that are realisable on physical systems. This method is still fairly new and we expect much better performance in the future.



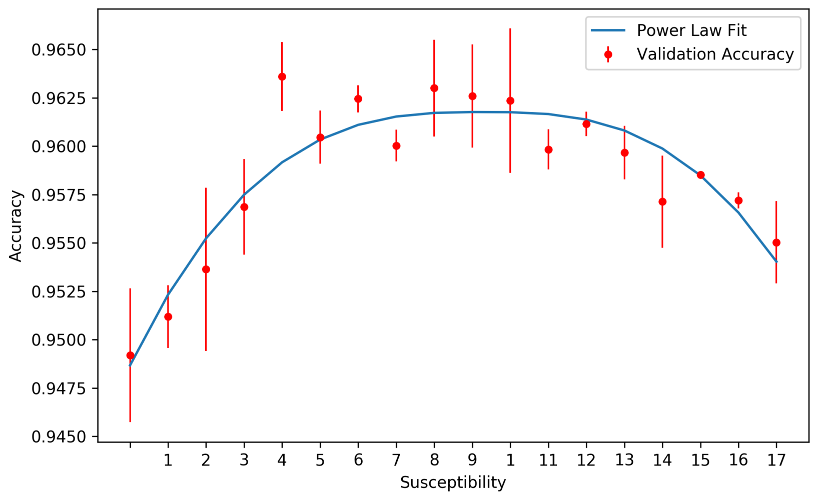


Figure 13: A power law fit of the form $y(x) = c - |x - x_0|^n \cdot a$ for the Vaporware network. Two separate fits were made for the positive and negative parts of the susceptibility. The fits follow the formulae $y_-(x) = 0.96 - |x + 9.3|^{2.6} \cdot 4.3 \cdot 10^{-5}$ and $y_+(x) = 0.96 - |x - 9.3|^{2.9} \cdot 2.2 \cdot 10^{-5}$. The precise values can be found in the appendix Tab. 2 and 3.

4 Discussion and Outlook

We will briefly discuss all the results that are presented in previous sections.

The Vaporware network is a neural network, which has not been implemented yet, that aims to perform most of its calculations by the propagation of light. To control this propagation we make use of a spatial light modulator. We intent to find out what properties the medium of propagation should have for optimal learning and in this paper we focus on the susceptibility. The role of the nonlinear activation functions is replaced by the optical response of dense atomic vapour.

In preparation we looked at the importance of linearity for a simple dense network to better isolate the behavior. We found that nonlinearity *is* important for performance and that purely linear networks have an accuracy at around 86% for the MNIST dataset. With nonlinearity these simple networks can achieve over 97%. We also found that the mere addition of a Softmax activation in the final layer can improve the accuracy of a linear network to about 93%.

Then we looked at the Vaporware network. As no physical implementation exists yet, we simulated the behaviour. The nonlinearity of the Vaporware network is controlled by the susceptibility of the medium. A positive susceptibility corresponds to a defocusing nonlinearity. A negative susceptibility causes self focusing in light. We expected the negative susceptibility to perform better since defocusing light loses more information while propagating. A susceptibility of zero is the same as propagation through empty space. The results showed the accuracy made a sharp dip around zero susceptibility and had two fairly symmetric peaks for both the negative and positive susceptibility. The dip was expected since the propagation is linear when the susceptibility is zero. The accuracy improves slightly faster for the negative susceptibility near zero but further from zero the accuracy is about symmetrical. This was against expectations and apparently for large susceptibilities the sign doesn't matter for the accuracy.

We can conclude that there exists a maxium performance for a certain susceptibility. If it is not possible to tune the susceptibility it might be possible to change how far the light propagates through the vapour. According to our results it will not matter much if the susceptibility is positive or negative, so either positive or negative detuning can be chosen.

For future research one could aim for a higher accuracy of the network. The network is still in its proof-of-concept phase so we expect much improvement performance wise. Its accuracy is still not competitive to conventional networks. One could also look at better models for the light propagation. We looked at a two level system, the simplest case. We know the ground and first excited state of rubidium split into seven states due to hyperfine splitting. A better approximation would be to consider two ground states and one excited state (lambda system) since the second ground state can heavily influence the populations. Atoms can easily enter this second ground state but once there they cannot leave so after a while the medium becomes depleted of available atoms. This would make the program slightly more complex but would provide more accurate results.

5 References

References

[1] Lecun, Y. (1998). Gradient based learning applied to document recognition. [online] Yann.lecun.com. Available at: `http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf`

[2] Kärtner, F. (2019). Ultrafast Optics Chapter 2. [online] Ocw.mit.edu. Available at: `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-977-ultrafast-optics-spring-2005/lecture-notes/chapter2.pdf`

[3] Monaghan, D., Hennelly, B., Kelly, D. and Naughton, T. (2019). [online] Applications of Spatial Light Modulators in Optical Signal Processing Systems. Available at: `https://www.researchgate.net/profile/John_Sheridan2/publication/228673618_Applications_of_spatial_light_modulators_in_optical_signal_processing_systems/links/0912f50a276cd8f341000000/Applications-of-spatial-light-modulators-in-optical-signal-processing-systems.pdf` [Accessed 12 Jun. 2019].

[4] Maier, S. (2019). PLASMONICS: FUNDAMENTALS AND APPLICATIONS. [online] Available at: `https://amolf.nl/wp-content/uploads/2016/04/Maier_PLASMONICS.pdf` [Accessed 12 Jun. 2019].

[5] Harold J. Metcalf, Peter van der Straten *Laser Cooling and Trapping* Springer 1999

6 Appendix

6.1 Appendix A

$$\begin{aligned}
\nabla \times \nabla \times E &= \\
\epsilon_{ijk} \frac{\partial}{\partial x_j} \left(\epsilon_{klm} \frac{\partial}{\partial x_l} E_m \right) &= \\
\epsilon_{kij} \epsilon_{klm} \frac{\partial^2 E_m}{\partial x_j \partial x_m} &= \\
(\delta_{il} \delta_{jm} - \delta_{im} \delta_{jl}) \frac{\partial^2 E_m}{\partial x_j \partial x_l} &= \\
\frac{\partial^2 E_j}{\partial x_j \partial x_i} - \frac{\partial^2 E_i}{\partial x_j \partial x_j} &= \\
\frac{\partial}{\partial x_i} \frac{\partial E_j}{\partial x_j} - \frac{\partial^2}{\partial x_j^2} E_i &= \\
\nabla(\nabla \cdot E) - \nabla^2 E &=
\end{aligned}$$

6.2 Appendix B

Power law fit of the simple network

Parameter	Value
n	3.920 789 019 397 453
a	$-0.025\,324\,627\,798\,680\,03$
x_0	$-0.526\,398\,924\,637\,799\,6$
c	0.983 129 397 248 702 5

Table 1: Precise values of the power law fit $y(x) = c + a|x - x_0|^n$ of the simple network. Here x is the negative slope of the LeakyReLU activation (α) and y is the accuracy of the model.

Power law fit of the Vaporware network (positive susceptibility)

Parameter	Value
n	2.867 677 678 148 010 2
a	$-2.201\,163\,316\,101\,709\,5 \cdot 10^{-5}$
x_0	9.278 487 280 126 003 6
c	0.961 765 966 776 804 3

Table 2: Precise values of the power law fit $y(x) = c+a|x-x_0|^n$ of the Vaporware network

Power law fit of the Vaporware network (negative susceptibility)

Parameter	Value
n	2.593 957 456 691 524
a	$-4.311\,672\,648\,172\,587 \cdot 10^{-5}$
x_0	$-9.290\,051\,096\,155\,036$
c	0.962 602 704 674 515 6

Table 3: Precise values of the power law fit $y(x) = c+a|x-x_0|^n$ of the Vaporware network