Review

Abdourahman Khaireh-Walieh, Denis Langevin, Pauline Bennet, Olivier Teytaud, Antoine Moreau* and Peter R. Wiecha*

A newcomer's guide to deep learning for inverse design in nano-photonics

Abstract: Nanophotonic devices manipulate light at sub-wavelength scales, enabling tasks such as light concentration, routing, and filtering. Designing these devices to achieve precise light–matter interactions using structural parameters and materials is a challenging task. Traditionally, solving this problem has relied on computationally expensive, iterative methods. In recent years, deep learning techniques have emerged as promising tools for tackling the inverse design of nanophotonic devices. While several review articles have provided an overview of the progress in this rapidly evolving field, there is a need for a comprehensive tutorial that specifically targets newcomers without prior experience in deep learning. Our goal is to address this gap and provide practical guidance for applying deep learning to individual scientific problems. We introduce the fundamental concepts of deep learning and critically discuss the potential benefits it offers for various inverse design problems in nanophotonics. We present a suggested workflow and detailed, practical design guidelines to help newcomers navigate the challenges they may encounter. By following our guide, newcomers can avoid frustrating roadblocks commonly experienced when venturing into deep learning for the first time. In a second part, we explore different iterative and direct deep learning-based techniques for inverse design, and evaluate their respective advantages and limitations. To enhance understanding and

*Corresponding authors: Antoine Moreau, Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Institut Pascal, F-63000 Clermont-Ferrand, France, E-mail: antoine.moreau@uca.fr; and
Peter R. Wiecha, LAAS, Université de Toulouse, CNRS, Toulouse, France, E-mail: pwiecha@laas.fr. https://orcid.org/0000-0002-4571-0116 (P.R. Wiecha)
Abdourahman Khaireh-Walieh, LAAS, Université de Toulouse, CNRS, Toulouse, France
Denis Langevin and Pauline Bennet, Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Institut Pascal, F-63000 Clermont-Ferrand, France
Olivier Teytaud, Meta AI Research Paris, Paris, France

facilitate implementation, we supplement the manuscript with detailed Python notebook examples, illustrating each step of the discussed processes. While our tutorial primarily focuses on researchers in (nano-)photonics, it is also relevant for those working with deep learning in other research domains. We aim at providing a solid starting point to empower researchers to leverage the potential of deep learning in their scientific pursuits.

Keywords: nano-optics; inverse design; deep learning; inverse problems; tutorial

1 Introductions

The broad field of (nano-)photonics deals with the interaction of light with matter and with applications that arise from structuring materials at sub-wavelength scales in order to guide or concentrate light in a pre-defined manner [1–5]. Astonishing effects can be obtained in this way, such as unidirectional scattering, negative refraction, enhanced nonlinear optical effects, amplified quantum emitter yields, magnetic optical effects at visible frequencies [6–12]. Tailoring of such effects via the rational design of nanodevices is typically termed "inverse design". Unfortunately, like most inverse problems, nanophotonics inverse design is in general an ill-posed problem and cannot be solved directly [13]. Usually, iterative approaches like global optimization algorithms or high-dimensional gradient based adjoint methods are used, which however are computationally expensive and slow, especially if applied to repetitive design tasks [14, 15].

In the recent past it has been shown that deep learning models can be efficiently trained on predicting (nano-)optical phenomena [16–20]. This rapidly growing research interest stems from remarkable achievements that deep learning accomplished in computer science since around 2010, especially in the fields of computer vision [21–24] and natural language processing [25–27]. The main underlying assumption is that neural networks are universal function

approximators [28]. It has been shown that deep learning is capable to solve various inverse design problems in nano-photonics. A non-exhaustive list of examples includes single nano-scatterers [29, 30], gratings [31, 32], Bragg mirrors [33–35], photonic crystals [36], waveguides [37], or sophisticated light routers [38–40]. For an extensive overview of the current state of research we refer the interested reader to recent review articles on the topic [41–48], or to more methodological reviews and comparative benchmarks [15, 49–51].

This work aims at providing a comprehensive tutorial for deep learning techniques in nano-photonics inverse design. Rather than assembling a complete review of the literature, we try to develop a pedagogical guide through the typical workflow, and focus in particular on the discussion of practical guidelines and good habits, that will hopefully help with the creation of robust models, avoiding frustration through typical pitfalls. We start with a concise introduction to the basic ideas behind deep learning and discuss their practical implications. We then study the question about which types of problem may benefit from solving them with deep learning, and which problems are probably better solved with other approaches. Subsequently we provide considerations on the typical deep learning workflow, including detailed advice for best practices. This ranges from the choice of the model architecture, data generation, parameterization, and normalization, to the setup and running of the training procedure and tuning of the associated hyperparameters.

In the second part, we introduce a selection of methods from the two most popular categories of inverse design using deep learning models. The first group of methods is based on iterative optimization, using deep learning models as ultra-fast and differentiable surrogates for slow numerical simulations. The second approach aims at developing end-to-end network training for solving the inverse problem. The latter, so-called "one-shot" solvers, can be implemented in different ways, we specifically discuss the tandem network, as well as conditional variational autoencoders and conditional generative adversarial networks (cVAE, cGAN). Finally we provide a short overview of further techniques. The paper is accompanied by a set of extensively commented Python notebook tutorials [52], that demonstrate the practical details of the presented techniques on two specific examples from (nano-)photonics: The design of reflective multi-layer systems and tailoring of the scattering response of individual nano-scatterers.

2 Introduction to deep learning and the typical workflow

Before diving into the technical details of different approaches to use deep learning (DL) for inverse design, it is crucial to become familiar with some basic concepts and good practices. In the following section we therefore provide a very concise introduction to artificial neural networks, we discuss how to assess whether it could make sense to apply deep learning to a problem and in which cases one should rather stick to conventional methods.

2.1 Short introduction to artificial neural networks

2.1.1 Artificial neurons and neural networks

The basic building block of an artificial neural network (ANN) is the artificial neuron. An artificial neuron is simply a mathematical function taking several input values. A weight parameter is associated with each of the inputs. As depicted in Figure 1a, first the sum of the weighted input values is calculated, and then the value of an additional bias parameter is added. Subsequently a so-called activation function f is applied to the resulting number. The result of the activation function is the neuron's output value, also simply called its activation. A neural network is nothing else than several of these artificial neurons connected to each other in some way, for instance by feeding the output of one neuron into other neurons' input layers (c.f. Figure 1b). Please note that some network architectures also implement other types of connections, for example of neurons within a layer or to preceding layers, etc.

2.1.2 Deep networks with nonlinear activations

A key hypothesis underlying deep learning ("deep" means that a network has many layers) is that the ANN is learning a hierarchy of features from the input data, where each layer is extracting a deeper level of characteristics. In an image for instance, the first layer could be recognizing lines and edges, the second layer may "understand" how they form specific shapes like eyes or ears and a subsequent layer may then analyze the relative positions, orientations and sizes in the ensemble of these features, to identify complex objects like animals or human faces. In consequence, using many layers
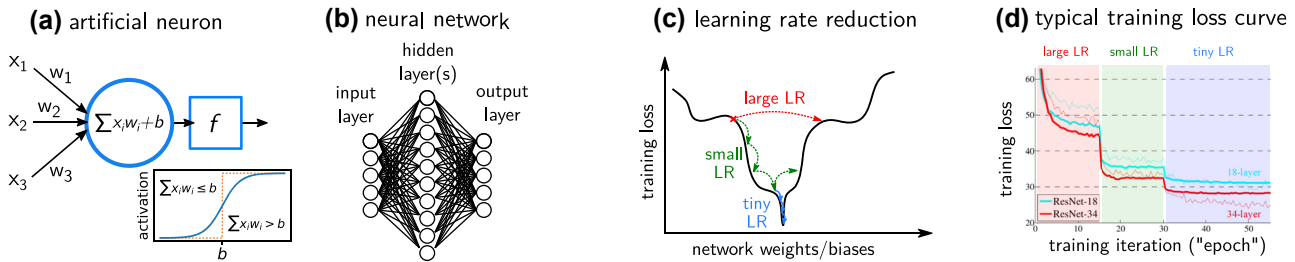
Figure 1: Basic concepts of artificial neural networks and their training. (a) A single neuron where *x* is the input vector, *w* and *b* are, respectively, the weights and biases and *f* is the activation function. The inset shows the output of the sigmoid activation function. (b) A set of neurons arranged in several, consecutively connected layers, forming a neural network. (c) Learning rate size effect on the loss function optimization. (d) A typical training loss or validation error plot. When the loss decrease starts to stagnates, reducing the learning rate leads to further convergence. Adapted from Ref. [22].

is essential to ensure that a network has a high abstraction capacity.

Now, it is technically possible to use linear activation functions throughout a network. However, it is trivial to show, that any neural network with multiple layers of linear activation can be identically represented by a single linear layer, as the chain of two linear functions is still linear. Hence, using nonlinear activation functions is crucial in any deep ANN in order to perform hierarchical feature extraction. Various activation functions can be used. The activation which is closest to a biological neuron's response is probably the sigmoid neuron, which implements a logistic activation function, depicted in the inset of Figure 1a. The sigmoid is however suffering from the fact that for inputs far away from the bias, the gradient is very small. For such inputs learning is very slow. Therefore, other activations such as rectified linear units (ReLU) expressed as $\max(0, x)$ often preferred in the hidden layers of an ANN, because of its constant gradient and cheap computation cost. In the "body" of the network, ReLU is usually a very good first choice and leads to robust networks.

The activation of the very last layer of a network, its *output layer*, needs to be chosen depending on the task to solve, as well as on the numerical range of the output data values. A softmax activation layer is a form of a generalized logistic function, where the sum of the neurons' activations of the entire layer is normalized. Hence it is adequate for outputs that correspond to a probability distribution like in classification tasks. In regression tasks – inverse design typically falls in this category – a linear output activation function is the simplest first choice, since it can take arbitrary values. Be aware that in case of a linear output activation the ANN needs to learn the data range along with the actual problem to solve. Therefore one should consider normalizing also the output part of the dataset (see also below). In this case Sigmoid (data range $[0, 1]$) or tanh activation (data

range $[-1, 1]$) can help the network to focus on the essential learning task (c.f. "inductive bias").

2.1.3 Training

Deep learning is a statistical method; the goal is to adapt the network parameters (weights and biases of the artificial neurons) such that the ANN learns to solve a task that is implicitly defined by a large dataset. The trick to learn from such data is to define a loss function that quantifies the network's prediction error. More precisely, the loss expresses how much the neural network's predictions of a set of samples are different from the expected network outputs. The notion of "set" is important, because the loss is defined in a stochastic approximation for *batches* of samples [53]. The expected outputs obviously need to be known and thus are also part of the training data.

Using an optimization algorithm, the loss function is minimized by modifying the weight and bias parameters of the network. During this optimization, the model repeatedly computes a batch of training samples in forward propagation to calculate the loss. Subsequently the gradients of the loss function with respect to all network weights are calculated in a backpropagation step, using automatic differentiation (autodiff) [54]. The network parameters are finally adapted towards the negative gradient direction, in order to minimize the loss function. Autodiff is the core of deep learning and hence all DL libraries are essentially autodiff libraries with tools for neural network optimization.

In practice, the model parameters are randomly initialized, hence DL is typically not deterministic. Restarting a training process will not give the exact same network. During training, the size of the parameter modification steps is a crucial parameter. It is typically called "learning rate" and takes generally values smaller than 1 (10^{-3} or 10^{-4} are common start values). For a given network parameter w_i,

the update is expressed as $w_i = w_i - LR \times \partial\text{loss}/\partial w_i$. If the learning rate (*LR*) is very small, the algorithm requires many iterations to reach the loss function minimum, and if it is too large, it can miss the optimal solution which often is a steep minimum. Thus it is important to reduce the learning rate during training, as depicted in Figure 1c and d.

2.1.4 Which optimizer

The most popular optimization algorithm in deep learning is stochastic gradient descent (SGD) [53, 55], which performs the weight updates as described above. A popular alternative, that often offers a faster convergence are the SGD variants "adam" [56] or "adamW" [57]. It is also more robust with respect to its hyperparameter configuration, and therefore an excellent first choice.

2.1.5 Practical implementation of neural networks

The *de facto* standard programming language in the deep learning community is Python, but frameworks exist for virtually all programming languages. The most popular libraries to build and train neural networks are "PyTorch" [58], "TensorFlow" with its high-level API "Keras" [59, 60], "Flax"/"JAX" [61], or "MXNet" [62], among others.

2.2 When is deep learning useful?

With the rapidly growing research interest around deep learning in the last years, a newcomer can easily get the impression that DL is the perfect solution to basically any problem. This is a dangerous fallacy. Assuming limitation to reasonable computational invests, deep learning will in many situations actually lead to inferior results compared to conventional methods, and due to data generation and network training, it will often come with a higher total computational cost. We therefore want to start with a survey of inverse design scenarios, and discuss situations in which deep learning may, or rather may not, be an adequate method.

The question of whether deep learning may or may rather not be an interesting option stands and falls with the quantity of available data. If huge amounts of data are available, with an appropriate model layout, deep learning likely works well on essentially any problem [26, 63, 64]. Unfortunately, data is often expensive. In photonics for instance, many simulation methods are relatively slow. In our considerations below, we will therefore assume the case of datasets with high computational cost and limited size, in the order of thousands to tens of thousands of samples.

2.2.1 When to use deep learning

2.2.1.1 Intuition

Deep learning is a data-driven approach: during training, the neural network is trying to figure out correlations in the dataset that allow to link the input to the output values, eventually converging to an empirical model describing the implicit rules behind the dataset (in our case the implicit physics). In that sense, neural network training can be somehow compared to human learning: By mere observations, humans figure out causal correlations in nature. In nanophotonics for instance, a person who is studying plasmonic nanostructures will sooner or later develop an intuition for the expected red-shift of a particle's localized surface plasmon resonance with increasing particle size.

Therefore, a good question to ask is: With the given dataset, does it seem easily possible to develop an *intuition* of the physical response? If this is the case, training a deep learning model on a large enough dataset promises to be successful. On the other hand, if a problem or its description is highly entangled and we can only hardly imagine that an intuitive understanding can be learned without further guidance, then an artificial neural network is likely going to have a hard time understanding the correlations in the dataset.

An example is shown in Figure 2. A Bragg mirror design problem is parameterized by its geometry of *N* dielectric layers with arbitrary material and thickness, and the target physical observable is the reflectivity at a fixed wavelength. The learning problem is a mapping of 2*N* values to a single reflectivity value. Providing a human with a large set of such data samples would most likely result in confusion rather than in an intuitive understanding of the correlation between geometry and physical property. The problem becomes easier to grasp intuitively if the layer-order is given to the network so it does not need to learn that the order is important. In deep learning this can be done using a convolutional layer, which keeps the input structure and searches correlations only between "neighbor" values. We exploit the convolution's *inductive bias* (see also below). We know that periods of the same pair of layers form the ideal Bragg mirror. With that prior knowledge we could further simplify the intuitive accessibility and parameterize the geometry with only two thicknesses and two materials values, this layer-pair being repeated for *N*/2 times. Finally, instead of predicting only a single reflectivity at one wavelength, we could train the network on predicting reflection spectra in a large spectral window. Now, the geometry can be understood easier and the impact of changes in the layer pair geometry is easier to interpret. A wavelength shift of
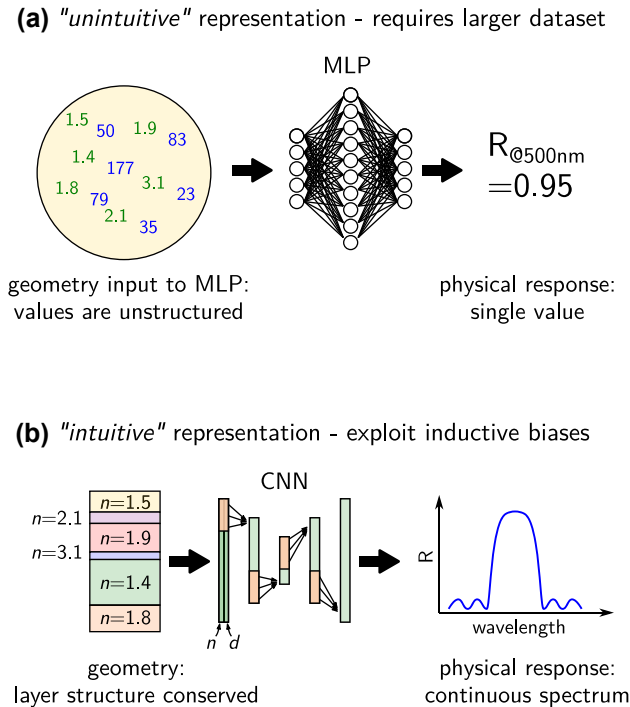
(a) *"unintuitive"* representation - requires larger dataset

MLP

$R_{@500nm} = 0.95$

geometry input to MLP: values are unstructured

physical response: single value

(b) *"intuitive"* representation - exploit inductive biases

CNN

$n=2.1$ $n=1.5$
$n=3.1$ $n=1.9$
   $n=1.4$
   $n=1.8$

n d

geometry: layer structure conserved

physical response: continuous spectrum

wavelength

Figure 2: Comparison of two different parameterizations of the same problem. (a) Using a dense neural network, the order of the thicknesses (blue values) and refractive indices (green values) is lost, because every input value is fed into every neuron in the first layer. The network needs to learn during training that the order matters. Furthermore, the correlation between the many input values and the single output reflectivity are difficult to understand. (b) Using a CNN, the layer order can be conserved and using two input channels, even the association of thickness and refractive index of a single layer can be directly passed to the network, hence during training these correlations do not need to be learned. Furthermore, predicting a whole reflectivity spectrum facilitates to identify correlations between changes of the geometry and for example resonance peak shifts. Returning this again via a convolutional layer conserves the order of the spectrum. On small to medium size datasets, exploiting these inductive biases of a CNN can significantly improve performance.

the stop-band for instance can be easily quantified if a spectrum is given instead of just one reflectivity value. In the same way as the latter representation is easier to interpret for a human, a deep learning model will be capable to develop an empirical model much easier. Other examples where such additional tasks, also called auxiliary tasks, are useful for improving the performance on the machine learning principle task, have been investigated in reference [65], and more recently in the case of the game "Go" [66]. In conclusion, richer information about the problem is often useful for making training faster and less data intensive.

2.2.1.2 Repetitive and speed-critical design tasks

In case it appears reasonable to assume that an intuitive comprehension of the problem can be built, one should also think about the motivation of using deep learning. On often highlighted advantage of DL models is their high evaluation speed (leaving aside the training phase). Using deep learning with the goal to speed up inverse design therefore seems reasonable. The training phase however requires significant computational work for network training and often also for data generation. Hence deep learning makes most sense in cases of highly repetitive design tasks like meta-surface meta-atom creation, or in speed-critical scenarios like real-time applications, such as spatial light modulator control.

2.2.1.3 Differentiable surrogate models

Another tremendous strength of deep learning models is the fact that they are differentiable. While gradients in numerical simulations may be obtained with adjoint methods [14], deep learning models can learn differentiable models even from empirical data, for example from experimental measurements.

2.2.1.4 Latent descriptions of high-dimensional data

A key capability of deep learning is the possibility to learn latent representations of complex, high-dimensional data. The latent space is not only a crucial concept in deep generative models [67, 68], it can also be used to compress bulky data [69, 70], or to gain insight into hidden correlations [71–73].

2.2.1.5 Empirical models from experimental data

On data that is very complex and/or high-dimensional, it can be difficult to fit a conventional physical model. A deep learning neural network may be a promising alternative to obtain a differentiable description for the physics, based on experimental data. A specific use-case application may be when a theoretical model fails to reproduce experimental observations. A so-called "multi-modal" [74] model could learn in parallel from experimental data and the simulated data. Both inputs are separately projected in a shared latent space. After successful training, the latent description creates a learned link between experiment and simulated data.

2.2.2 When to NOT use deep learning

Writing a good data generation routine to create a useful training set is at least as challenging, but often even more challenging than writing a good fitness function for a conventional optimization technique. Additionally, it has been demonstrated on several occasions, that simple conventional methods often outperform heavy GPU-based

black-box optimization methods [75–78]. Before rushing into data generation, we therefore urge the reader to consider whether conventional techniques may not be a sufficient alternative to deep learning on their specific problem.

2.2.2.1 "Unintuitive" problems or parameterizations

Deep learning may perform badly if the problem or its parameterization is not intuitive, hence where the correlations between input and output are highly abstract. This is the case in the above example of a sequence of N random dielectric layers, where the goal is to map the $2N$ values of layer thicknesses and materials to a single output value (the reflectivity). While the problem is very easy to solve with the right physical model at hand (e.g. with transfer matrix or s-matrix method), a human presented with examples of such $2N + 1$-value samples will have a very hard time to understand the correlation. This can typically be solved by proper pre-processing and network architecture design, but requires supplementary effort.

2.2.2.2 Single design tasks and simple problems

We argue that the main argument against deep learning is that it is inappropriately expensive in many situations. If the geometric model used in the design problem is described by only a few free parameters (like 3 or 4), a systematic analysis is probably very promising and maybe even cheaper than generation of a large dataset. In many cases this may furthermore be done around an intuitive, first guess. Similarly, if the geometry is more or less known and only small variations are to be optimized, conventional optimization or again a systematic exploration of all parameters are more appropriate. Such situations are depicted in Figure 3. Even if a systematic analysis cannot be performed with sufficiently dense parameter steps, it may be worth considering conventional interpolation approaches such as Chebyshev expansion or also Bayesian optimization [79–82]. Finally, if

the problem consists in solving a single design target, even for complex scenarios a conventional global optimization run is probably the more adequate approach [15].

In conclusion, finding problems that really benefit from deep learning based inverse design is not as obvious as it is often suggested in literature. As an example to support this claim we recall that to date all production-scale metasurface design is being done with conventional lookup tables, while deep learning is still only used on toy-problems for testing.

2.3 General workflow and good habits

Let's suppose we have decided to go for deep learning as method of choice. Before diving into the details of methods for inverse design, we want to provide a loose guide of the typical workflow with suggestions for good habits that should be useful when applying deep learning to any problem, not limited to inverse design. In the following we aim at providing guidelines that implement findings of modern deep learning and that will be useful to get good results with little trial and error. The following section is structured in the way that we believe the workflow should be organized. A schematic overview is depicted in Figure 4.

2.3.1 Overview of the major model architectures

Neurons can be connected in different ways to form an ANN. The most adequate neural network type will also depend on the dataset to be processed. Multilayer perceptrons (MLP), convolutional neural networks (CNN) and recurrent neural networks (RNNs) are the most widely used general architectures. Recently so-called "attention"-layers have gained significant popularity as well. In the following we will discuss the advantages and drawbacks of the different network architectures and give suggestions concerning their applications.

2.3.1.1 Multilayer perceptron (MLP)

The MLP is also called fully connected or dense network, and consists of layers of unstructured neurons, where each neuron of a layer is connected to every neuron of the preceding and of the following layer [83]. Because they scale badly with dimension, MPLs should not be used on raw data, but on *features* of the data. Exceptions could be natively relational data, or if the dataset consists already of high-level features. In geometric inverse design, raw data might be images of the structure, whereas size and shape parameters would be the features. In consequence, MLPs are typically part of a larger neural network, where features are extracted in a first stage.

(a) small problem size
e.g. plasmonic bow-tie
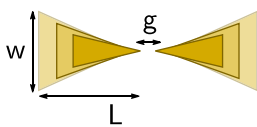
(b) single design target
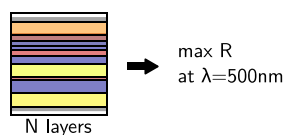e.g. thin film stack

max R
at λ=500nm

N layers

Figure 3: Examples of problem configurations, for which deep learning is probably ***not*** adequate. (a) Problems with few parameters like the here depicted plasmonic bow-tie antenna design (3 free parameters) can easier be solved by conventional approaches, intuition or may even be systematically explored. (b) In problems with a single design target, the computational overhead of deep learning is not paying back. Conventional global optimization is the method of choice.
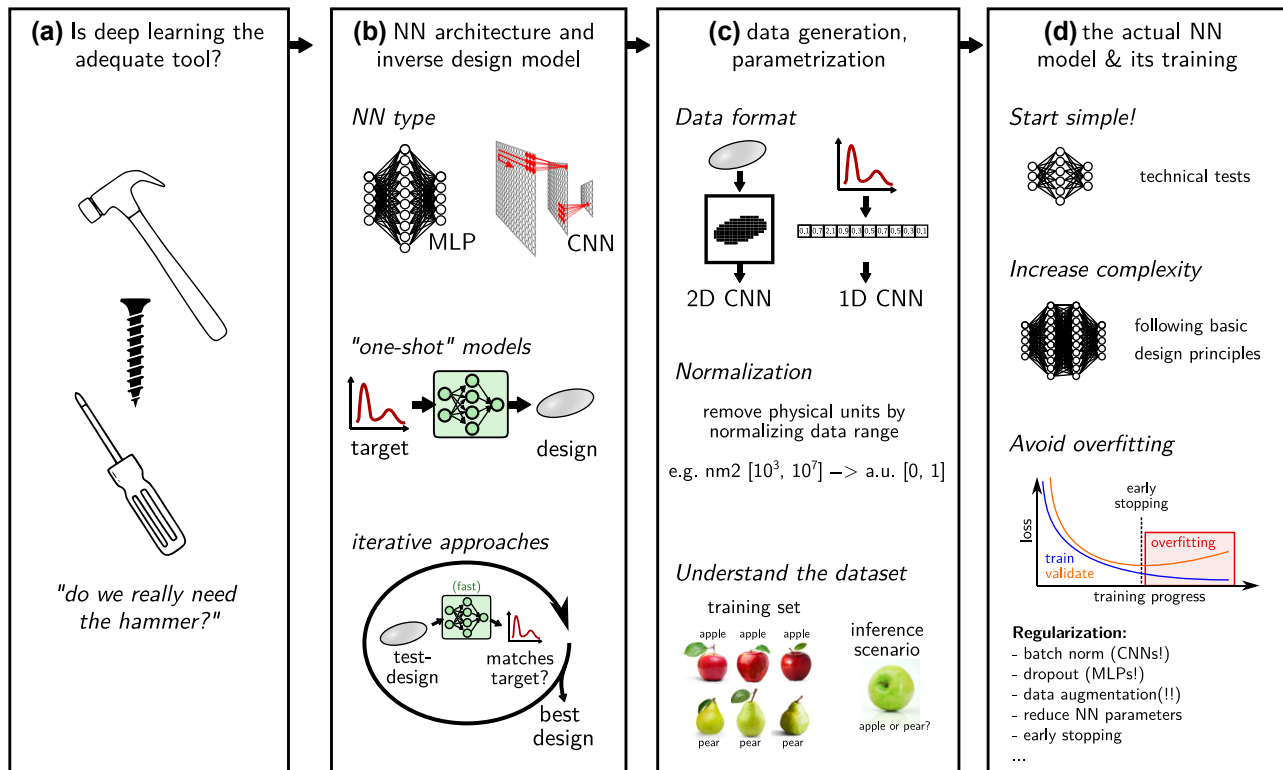
Figure 4: Workflow for the application of deep learning in inverse design. (a) The workflow should start with an unbiased assessment, whether other methods might not be more adequate than deep learning ("do we need the hammer, or is it rather a screwdriver?"). (b) Think about how the problem can be parameterized. Which network type will be necessary? Is the goal ultimate acceleration (use one-shot inverse design), or a less time critical, best possible optimization (use iterative inverse design). (c) Prepare and understand the dataset. (d) Implement the actual mode. Start simple, increase complexity, avoid overfitting (regularization!). Always keep in mind literature design guidelines.

Those features are then passed into the MLP, for example in various convolutional neural network architectures, or similarly also in the attention heads of transformers (see below) [84, 85]. With large layers the number of free parameters in fully connected networks can quickly diverge, which makes this architecture particularly prone to overfitting.

We argue that MLPs can be a valid option for inverse design, for example when a model is parameterized with only a low number of parameters and/or when the dataset comprises relatively high-level features of low dimensionality (e.g. a low number of size and/or position parameters). However, for more general problems and parameterizations (e.g. parameterization at the pixel/voxel level), we recommend CNNs (see below).

2.3.1.2 Convolutional neural networks (CNN)

CNNs are used since the 1980ies [86–89], and were inspired by how receptive fields in the visual cortex process signals in a hierarchical manner [90]. They work similarly to popular computer vision algorithms of the 1990s that processed images via feature detection kernels [91], with the difference that the CNN kernels are composed of artificial neurons and

automatically learn from data, instead of being manually defined. Feature detection itself is performed by convolutions of the kernels with the structured input data (e.g. images) [55]. In the early 2010s, GPUs rendered calculation of discrete convolutions computationally cheap. This allowed to scale-up CNNs, which lead to a breakthrough in computer vision performance. Ever since, CNNs are amongst the most popular artificial neural network architectures that scale well from small up to gigantic data set and problem sizes [21, 92]. In particular the idea of residual blocks with identity connections allowed to scale the depth of CNNs by orders of magnitude (thousand layers and more), while maintaining efficient training performance [22, 93, 94].

The popularity of CNN is a result of many favorable properties. For instance they offer inductive biases (see also below), that are useful for many applications like locality (assuming that neighbor input values are closely related e.g. in an image or a spectrum), or translation invariance of the feature detection (a feature will be detected regardless of its position in the input data). CNNs are easy to configure and very robust in training. They are applicable and perform very well over a large range of input dimensions, e.g.

from tiny input arrays to megapixel images, and they scale well to large dataset sizes [67]. They are computationally friendly since the convolutions can be very efficiently calculated on modern GPU accelerators. Please note, that an often forgotten advantage of CNNs is also their ability to work on variable input sizes [95]. And finally, due to their tremendous success in the past 10 years, an abundance of recipes and optimization guides are available in the literature, which lowers the entry barrier for newcomers to an absolute minimum.

2.3.1.3 Recurrent neural networks (RNN)

Recurrent neural networks implement memory mechanisms in networks which process data sequentially. Thanks to this memory, long-term correlations in data sequences can be processed [96, 97]. RNNs have been the state of the art in natural language processing for some time, and due to their success in this area, researchers have applied RNN concepts to various other fields as well [98]. However, RNNs suffer from a main drawback, their training is largely sequential and cannot easily be parallelized, hence training scales badly with increasing amounts of data [85]. With the recent exploding availability of data and highly parallelized accelerator clusters, RNNs were losing foot in terms of performance and have today been mostly replaced by other architectures like CNNs or attention based models like transformers [99, 100].

We argue that RNNs should not be the first choice for typical inverse design tasks, because of their drawbacks such as non-parallel training and more complex hyperparameter tuning compared to CNNs.

2.3.1.4 Graph neural networks (GNN)

Graph neural networks (GNNs) are an emerging class of network architectures that process data represented as graphs. Graphs are natural representations for various data structures, for example social networks or molecules in chemistry [101, 102]. Several variants of GNNs exist, such as convolutional GNNs [103], Graph-attention networks [104], or recurrent GNNs [105]. Please note that also CNNs are strictly speaking a specific type of GNN, processing image-"graphs" where connections exist only between neighbor pixels. One of their advantages is that they can operate on data of variable input size and are very flexible regarding the data format. In physics GNNs have been proposed for example to learn dynamic mesh representations [106]. In nano-photonics, GNNs have started to be used only recently. GNNs have also been used for the decription of optically coupled systems like metasurfaces, including non-local effects [107]. It has been also demonstrated recently that GNNs can learn domain-size agnostic computation schemes. A GNN learned the finite difference time domain (FDTD) time-step

update scheme to calculate light propagation through complex environments [108].

In our opinion, GNNs can be a very interesting approach, i.e. for difficult parametrizations. But we argue that they should not be the first choice for a newcomer, because of the scarce literature in nano-photonics and challenges in its proper configuration.

2.3.1.5 Transformer

The transformer is a recent, very successful, attention-based model class [85]. The underlying attention mechanism [109] mimics cognitive focusing on important stimuli while omitting insignificant information. The attention module gives the network the capacity to learn a hierarchy of correlations within an input sequence. In natural language processing (NLP), transformers largely outperform formerly used recurrent neural network architectures, which they entirely replaced in all NLP applications [26, 99]. In 2020 the concept was adopted to computer vision with so-called vision transformers (ViT), followed by important research efforts with some remarkable results [110–113]. However, tuning the hyperparameters of a transformer is significantly more difficult than conceiving a good CNN design, therefore also approaches that combine CNNs with ViTs were proposed [114, 115]. Furthermore, while the transformer's main advantage is its excellent scaling behavior to huge datasets and model sizes, the need for gigantic datasets is probably also their biggest shortcoming. When the size of the datasets decrease or for smaller model sizes, their advantages diminish [116]. In fact, "small" datasets in the world of transformers still range in the order of hundreds of thousands samples [117].

We argue that transformers are in most cases not the most adequate architecture for inverse design, since their advantages unleash only for datasets of gigantic sizes, in the order of millions or even billions of samples [20, 116].

2.3.2 Which network type to use

Comparing the arguments in favor and against the different network architecture that are presented in the preceding section, we come to the conclusion that, whenever applicable, CNNs are generally the best first choice for approaching an inverse design (or other) problem with deep learning. They are simple to design, robust in training and have excellent scaling behavior. If the data is not in an adequate format, often it is even advantageous to reformat the data in order to render it compatible with a CNN. There is no "one" recipe to do so, but if a meaningful structure exists in the data values, it should be conserved. In case of a sequence of thin film layers for instance, it makes sense to concatenate

layer thicknesses and layer materials in two lists of values that can be fed to a one-dimensional CNN (c.f. Figure 2b).

Besides the first technical tests, in general one should not use simple sequences of convolutions, typically called "visual geometry groups" (VGG). It is long known that such VGG-type CNNs have severe limitations and do not scale well with model size [22]. Instead, convolutional layers should be organized in residual blocks [93] or, even better and only slightly more complex, in residual "ResNeXt" blocks with inverted bottleneck layers and grouped convolutions [116, 118]. We refer the reader to the supplementary Python notebook tutorials for a detailed technical description and example implementation [52].

2.3.3 Choosing the inverse design method

2.3.3.1 Iterative DL based approaches
Following conventional inverse design by global optimization, a deep learning surrogate forward model can be used to accelerate iterative optimization for inverse design [51, 119].

Drawbacks of iterative methods are: They are slower than one-shot approaches (due to multiple evaluations through the iterations) and coupled with deep learning, they may bear the risk of convergence to network singularities, since they actively search for extrema in the parameter space [120]. Also convergence to parameters in the extrapolation zone needs to be avoided, since often the physical model collapses in the extrapolation regime, adding an additional technical challenge since the interactive solver needs proper regularization [18, 121]. Such dangerous extrapolations can also be mitigated by increasing ground truth data: a real physical simulator is sometimes used (typically at designs proposed by the surrogate model), and the learned model is updated to take into account these new data. This is a classical procedure in machine-learning enhanced global optimization [122–126].

2.3.3.2 Direct inverse design ("one-shot")
An alternative class of inverse design methods is one-shot methods. The goal is to create a neural network, which takes the design target as input and immediately returns a geometry candidate that implements the desired functionality. The main advantage of such methods is the ultimate speed. A single network call gives the response to the inverse problem. However, this also means that no optimization is performed in this case, and the solution is likely not the optimum. To obtain close-to-optimum solutions from one-shot approaches, considerable effort needs to be invested in dataset generation, network design, optimum training and proper testing [127]. Alternatively, a one-shot inverse model

can be very useful to provide high-quality initial guesses for a subsequent (gradient based) optimization.

2.3.4 The dataset – questions to ask about the data

The data are the most important resource in a deep learning model. It is therefore essential to meticulously carry out data generation or, if data already exists, to understand the dataset. In both cases it is often helpful to perform appropriate preprocessing. As a rough guide, we provide a few questions that one should ask about the data:

– *How much data is necessary?*

Most deep learning architectures require at the very least a few thousand samples, likely more. There are cases where less data can suffice, for instance in transfer learning/fine-tuning of an existing model. In some cases it's possible to learn on slices of large but few samples, for instance in segmentation tasks [128, 129].

– *Is random data generation possible?*

The design targets may be sparse in the parameter space, in which case training using a random dataset may not work. This is often the case in free-form geometries, where the number of design parameters is high. Using a weak optimization method for data generation can help in such cases, but special care must be taken to avoid bias of the dataset towards a specific type of pre-optimized solution, so enough randomness is important [40]. Also systematic data sampling approaches (e.g. Sobol sampling) may be useful [130–132]. The potential difference between random and optimized data is illustrated in Figure 5.
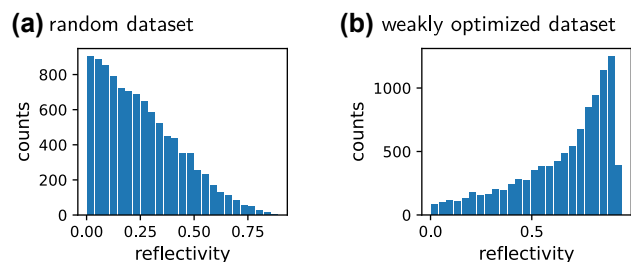

(a) random dataset   **(b)** weakly optimized dataset

Figure 5: Histograms of two illustrative datasets of the reflectivity (at $\lambda = 500$ nm) of 10,000 dielectric thin-film sequences. (a) Randomly generated thin film sequences. Their reflectivity is in general low. A network for the design of high reflectivity solutions, that is trained on this dataset, will very likely fail. (b) Weakly optimized data generation, starting from random samples. The dataset offers a large portion of medium to high reflectivity samples. Care should be taken that still enough "randomness' is present to avoid biases towards high R solutions. A network trained on the optimized set will perform better at the task of reflectivity maximization.

– *Are there obvious biases in the data?*

Biased data is one of the most important challenges in deep learning. In short: "what you put in is what you get out". For instance if a dataset is built with resonant nanostructures, a network trained on this data may not correctly identify non-resonant cases.

– *Is the data meaningful?*

Samples carrying the essential information may be sparse in the dataset or the important features may be hidden in a large number of irrelevant attributes. A careful pre-processing of the dataset may help in such cases.

– *Can I shuffle the data?*

Are subsets of the data partly redundant (e.g. after sequential data generation)? If yes, a splitting method is required that guarantees that training, validation, and test subsets are independent.

– *Are the subsets (training data, validation data) representative for the problem?*

In order for the training loss and benchmark metrics to be meaningful, each subset needs to fully cover the entire problem to be solved. If we apply the network on significantly different samples than it was trained on, we probe the extrapolation regime, where performance is generally weak. A non-representative validation or test dataset will therefore provide a bad benchmark [18]. An illustration depicting non-representative subsets is shown in Figure 4c.

– *How frequent are outlier*

If necessary, can the impact of exceptions or anomalies in the data be reduced, e.g. by selective removal or by pre-processing with a non-linear scaling function? [133, 134].

– *How should the data be normalized?*

Input values should in general be standardized: Subtract the mean and divide by the standard deviation. Figure 6 depicts why this is essential. The choice of the normalization for the network output is closely linked to the choice of the loss function, but also depends on the data representation and on the significance of outliers, among other factors. If the data has multiple output values or channels, are those of similar numerical magnitude? If not, should they be normalized with the same scaling process or each channel individually? Appropriate normalization is highly important for an accurate model.

– *Can we exploit inductive biases?*

The term *inductive bias* denotes any assumption about how to solve a task that is implicitly included in the model. In a classification task for instance, the
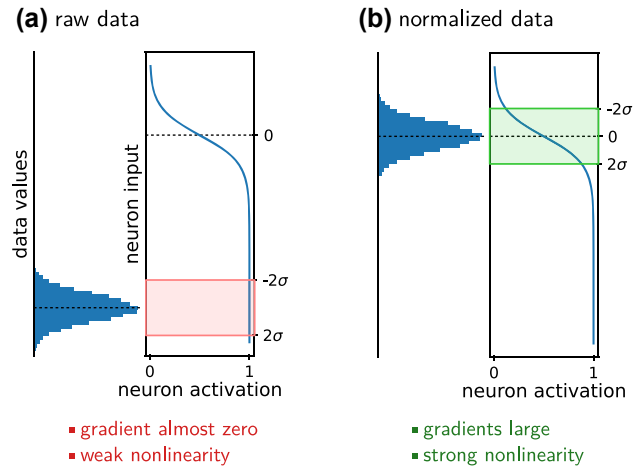

Figure 6: Normalizing the data is important. (a) Large numerical values do not exploit the non-linearity of typical neuron activation functions. In case of specific activations like Sigmoid or tanh, furthermore the gradients of the neuron output are very small. Both situations are unfavorable for network training. (b) The numerical values of normalized data cover the full range of an artificial neuron's non-linearity. Also the neuron gradients are large for all typically used activation functions. This significantly helps the learning process. On non-normalized data, during training first the neuron biases need to be optimized which consumes unnecessary computation. The same argument explains why batch normalization is so effective.

output layer is chosen as a probability distribution. In a regression task, the output layer is a continuous activation function. Such inductive biases improve the network training. In their absence, a network needs to learn them from the data in addition to the actual task. For example a classifier would need to learn that the output is a binary yes/no decision, and parts of the network would be used for this aspect. Often it is possible to include some properties of the problem implicitly in the network architecture. CNNs typically imply translation invariance. Likewise, symmetries may be included, or the valid data range, by choice of an adequate output activation. In physics, a possibility to exploit inductive bias is causality, which may be enforced through Kramers–Kronig relations or, more simply, using a Lorentzian output layer [18, 135].

2.3.5 Practical network implementation and training

Once the data is prepared, the full set needs to be split into training, validation and test subsets. With these subsets we are ready for the actual training of a network. The next step is to code and train the model. This section is a little technical, we ask the reader to look at the Python notebook tutorials that demonstrate explicit implementations of networks following the below guidelines [52].

Since a sophisticated model may train very slowly, it is important to do a first technical test with a very simple network model. Even though you shouldn't use it as a final layout, the initial test network can be a sequence of a few stacked convolutional layers ("Visual Geometry Group", VGG). With this we test the implementation of the input and output layer dimensions, if the data scales are correct, and if the network output format is adequate. Finally we get a first idea about the typical learning loss. As for the specific configuration of the convolutions, small 3×3 kernels, strides 1, and "ReLU" activations can be used as a relatively foolproof rule of thumb [118]. Kernels with larger receptive field are occasionally used in literature [116], but increasing the convolutional kernel size increases quadratically (in 2D) the computational cost as well as the number of network parameters. It can be easily shown that stacking N 3×3 convolutions covers a receptive field equivalent to a single layer with kernel size $(2N + 1) \times (2N + 1)$, but with significantly less trainable parameters, reduced computational effort and potentially the additional benefit of hierarchical feature extraction capabilities [136].

Once this technical test is passed, it is time to increase the model complexity. Simple convolutions are converted into residual blocks [93] or, even better, "ResNeXt" blocks [116, 118], and more and more layers are added to the network, while paying attention to overfitting. Once overfitting occurs, the network complexity needs to be reduced again, or if the accuracy is not yet sufficient, other regularization strategies can be considered, such as data augmentation (for example, shifting or rotating the input data, if possible). Following dense layers, dropout can be applied, which randomly deactivates some of the neurons during training, leading to a model less prone to overfitting [137].

2.3.5.1 A note on overfitting

Recent research showed that overfitting occurs mainly in medium size networks, at the so-called interpolation threshold, where the number of model parameters is comparable to the number of degrees of freedom of the dataset. It appears that overfitting can not only be avoided by reducing the model size (or increasing the amount of data), but it can in fact be circumvented also by increasing the model size far into the overdetermined region [138–141]. These findings are in direct link with the modern concept of so-called *foundation models* like the large language models GPT3/GPT4 [26, 142]. Such foundation models are extremely large network models that are pre-trained on a gigantic corpus of generic data, and in a second step are fine-tuned on specific downstream tasks, which is usually successful even on very small datasets [20, 143].

In a typical scenario of deep learning for nanophotonics inverse design however, the model and dataset sizes are likely in a regime where the conventional considerations regarding overfitting hold and corresponding parameter tuning will be beneficial.

2.3.5.2 Dropout and batch normalization

We feel it is important to dedicate a paragraph on dropout and batch normalization. Both methods are applied on training time. Dropout deactivates random subsets of neurons with the goal to avoid overfitting because the model cannot rely on exactly memorized information [137]. Batch normalization normalizes the samples that it receives according to the statistics of the current batch of samples, with the goal to ideally exploit gradients and non-linearities of the activation functions (see also Figure 6) [144].

Even though dropout is often found in internet tutorials, using it in convolutions should be avoided. Its effect in CNNs is entirely different than in dense layers, for which it was originally proposed. If it does help regularizing a CNN, this will be rather accidental, than an effect that can in general be expected [145].

Instead of dropout, batch normalization (BN) should be used in deep CNNs (in particular in deep res-nets), after the convolution and before the nonlinearity [144]. As a rule of thumb, BN and dropout should not be acting on the same layer of neurons [146]. BN is indispensable in very deep architectures as it counteracts internal covariate shift. Furthermore, BN typically stabilizes training in the sense that it allows to use larger learning rates.

However, in smaller networks (in the order of a few tens of convolutions) and in particular with normalized data, BN is often not very useful, since the data normalization is more or less conserved through late layers in smaller networks. On the other hand, BN slows down data processing and training due to its computational overhead [147]. Even worse though, in physics regression tasks, using batch normalization (and also dropout) can be an unexpected pitfall, often difficult to pinpoint. This is because BN and dropout are both dynamic regularization techniques that act differently on every training batch. This can in fact be problematic when the statistics of individual batches fluctuate significantly, which is the case if the training data has a large variance. The network then re-normalizes each data batch very differently, introducing a new statistical error. In many applications, such as in segmentation tasks or in classification problems, the network output goes through a final normalization layer (typically softmax), then this problem is usually negligible [148, 149]. For regression tasks on the other hand, this can be a source of a considerable error. Large batch-sizes in the late training can counteract

the problem to a certain extent. Still, in many regression problems, in particular with small to medium size network models, the best solution is often to avoid using batch normalization (or dropout) [150]. When large models are required, BN should be used with an adequate batch-size increase schedule.

To conclude, we want to remind again, that normalization of the input data using the whole data-set statistics is a small, yet crucial step in the pre-processing workflow (see Figure 6). This by itself speeds up training due to the same mechanisms as used in BN, while avoiding some of the problems mentioned here [151].

2.3.6 Training loop good habits

A lot of frustration can be avoided if some good habits are respected concerning the training loop itself.

– *Learning rate:* The learning rate is an important training hyperparameter. If it is too small, training gets stuck in a local minimum. If it is too large, the network parameters may "overshoot", resulting in a diverging training. The choice of the initial learning rate (LR) is strongly dependent on the problem and on the network architecture. Typically it needs to be tested which LR works well. Once a good starting value is found, the LR should not be left constant through the entire training loop. Training rate schedules of varying complexity exist, but a good starting procedure is usually to gradually reduce the LR (see also Figure 1c and d). A simple learning rate decay schedule such as dividing the LR by a factor 10 every time the validation loss stagnates for a few epochs, is often already very efficient.

– *Batch size:* Evaluating the training loss on small batches of random samples is one of the key mechanisms in deep learning. It is the pivotal procedure that impedes the optimizer to get stuck in local minima. As a result the batch size (BS) is also a crucial hyperparameter that has a tremendous impact on the training convergence and model performance. While it depends also on each specific problem, in general the BS should be small in early training. 16 or 32 are good initial values. Too large starting batch sizes usually converge to local minima and thus lead to less accurate models [152, 153]. However, once the loss stagnates, it is good practice to increase the batch size since it renders the gradients smoother, and helps further decreasing the loss function, similar to the reduction of the learning rate [154]. Please note that, as in any global optimization scheme, it is impossible to evaluate whether the actual global minimum was reached, because in general training will always end in a local minimum.

A few additional, rather technical tips are given in a dedicated Section 4.
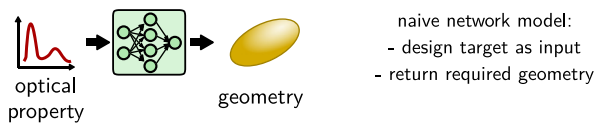
2.4 Alternative data-based approaches

Developing, optimizing and training a deep learning model can require a considerable time investment and can be very resource consuming. Having a look at other data based approaches can be worth a try. Methods such as k-means or principle component analysis [155], clustering algorithms like DBscan [156] or t-SNE [157], and machine learning methods like support vector machines [158] or random forest [159] are typically straightforward in their application and often computationally more efficient than large deep learning models. Since this is not the scope of this work, we refer the reader to the above cited literature. The python package scikit-learn is also a very accessible opensource collection of machine learning algorithms and tools with an extensive and well-written online documentation [160].

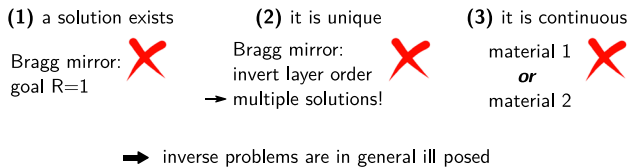# 3 Deep learning based inverse design

After having discussed the general approach and workflow for deep learning, we will now explain different possible approaches to solve inverse design problems via deep artificial neural networks.

The naive approach to solve inverse design with deep learning would be to use a feed-forward neural network that takes the optical property as input and returns the geometry parameters as output. This would be trained on a large dataset. Unfortunately such approach, as depicted in Figure 7a, does not work.

The main challenge in nanophotonics inverse design is that the problem is in general ill posed, in consequence it is impossible to solve the problem directly. J. Hadamard described a so-called "well posed problem" as one for which a solution does exist, this solution is unique and continuously dependent on the parameterization (c.f. Figure 7b) [13]. The typical inverse design problem however has in general non-unique solutions (multiple geometries yield the same or very similar property). Often design targets exist that cannot be optimally implemented, hence no exact solution exists (e.g. a mirror with unitary reflectivity). And finally, in many cases the physical property of a device is not continuously dependent on the geometry, but the parameter space is at least partially discrete (e.g. if a choice from a finite number of materials has to be made). Training of a naive network on a problem with multiple possible solutions will

(a) naive inverse network (**this generally fails!**)

optical property → geometry

naive network model:
- design target as input
- return required geometry

(b) well posed problems: Hadamard conditions

(1) a solution exists

Bragg mirror: goal R=1 ❌

(2) it is unique

Bragg mirror: invert layer order → multiple solutions! ❌

(3) it is continuous

material 1 *or* material 2 ❌

➡ inverse problems are in general ill posed

(c) train naive network on data with non-unique solutions

inference ➡

Au Ag Au Ag Au

unstable training, non-physical "average" output
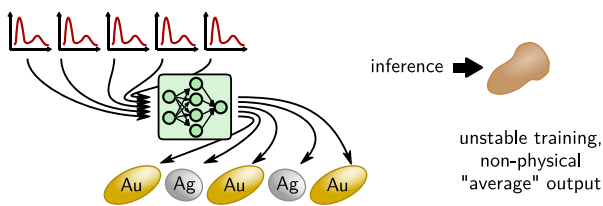
Figure 7: The crux of the ill posed problem. (a) A naive, not working implementation of a simple feed-forward inverse network would take as input the design target (e.g. an optical property) and returns the design that is required to obtain it. (b) Only well posed physical problems can be solved this way. Such problem obeys the three Hadamard conditions. However, neither of these conditions is in general fulfilled in photonics inverse design, as illustrated by a selected example under each condition. (c) In case of multiple solutions, the training process would iterate of these several times, every time adapting the network parameters to return a different design. Training is unstable and eventually the network will learn some non-physical mix of the multiple solutions. If a non-continuous parameterization is used (here: two distinct materials), the naive network may also return non-allowed mixtures of those.

oscillate between the different possible outputs and finally learn some non-physical average between those different solutions, as illustrated in Figure 7c [45].

Fortunately, methods exist to solve ill posed inverse problems with deep learning. We discuss in the following two popular groups of approaches. The first contains iterative methods that use optimization algorithms to discover the best possible solution(s). The second group consists of direct ("one-shot") inverse design methods.

3.1 Iterative approaches

Conventional inverse design generally uses iterative methods to search for an optimum solution to a given design problem. One way to speed up inverse design with deep learning is hence to accelerate the iterative process with

a deep learning surrogate model. A surrogate model is a forward deep learning network that is trained on predicting the physical property of a structure, hence to solve the "direct" problem [17, 50, 119, 161, 162]. This is usually straightforward, and consists in a feed-forward neural network that takes the design parameters as input (e.g. geometry, materials, …) and returns the physical property of interest via its output layer (e.g. reflection or absorption spectrum, …). Provided a large enough dataset is available, designing and training of such forward model is generally not difficult. We advise to follow the design guidelines from Section 2 and to consult the Python tutorial notebooks.

An accurate forward model can be used in various ways for rapid iterative inverse design.

3.1.1 Avoiding the extrapolation zone

The most imminent danger when using iterative optimization with deep learning surrogates is that the optimizer pushes the design parameters outside of the model's validity range. Most neural networks are strong at interpolation but bad at extrapolation tasks. This is illustrated in Figure 8, where a neural network has been trained on predicting the reflectivity of a dielectric thin film layer stack. This network totally fails in doing so outside of the range of the training data. For input values that are 10 % larger than the largest training data, the average error is already in the order of 20 % (Figure 8b). This illustrates that it is important to constrain the allowed designs to the parameter range that has been used for training, hence to the interpolation regime of the forward model. Please note that even if the model is efficiently constrained, a non-negligible risk of converging to network singularities remains in any case, since optimization algorithms seek extrema of the target function [83, 120]. We insist therefore that careful verification of results obtained from deep learning is crucial in any case. A classical approach is also to retrain the surrogate model with the new data, after output data has been verified by a classical physical model.

In order to constrain the designs, in specific cases dependent on the design parameterization, it is possible to formulate a penalty loss. For instance when the design is described by size values for which the limits are known [121]. The penalty loss is added to the optimization fitness function, which then increases, if the optimizer leaves the allowed parameter range. A drawback of this method can be the increasing complexity of the fitness function, and the requirement of problem-specific weight tuning to balance the contributions to the total fitness function.

In many situations this tuning can turn out to be complicated or the parameterization of the design is difficult to
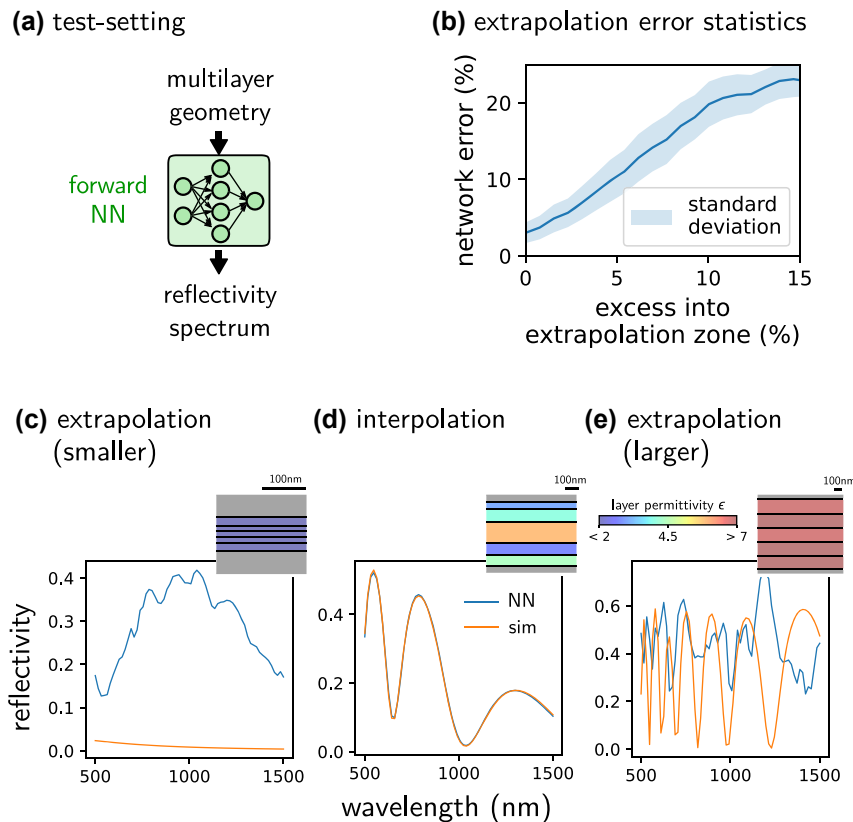
Figure 8: Illustration of failed extrapolation. (a) A neural network was trained on predicting the reflectivity spectrum of a thin film layer stack made of ideal dielectrics with constant permittivity ε. (b) Average forward network error as function of the excess of the input parameters outside of the training data range. (c) Failed prediction example outside of the training data range (smaller permittivities and thinner layers). (d) Example inside the range of the training-set design parameters (interpolation). The network predictions are accurate. (e) Same as (b) but for larger permittivities and thicker layers than used in training. Orange lines: PyMoosh simulated reflectivity spectra. Blue lines: neural network predicted spectra. Insets represent the used layer stack. Bar heights correspond to the layer thickness according to the scale bar, the color code indicates the layer dielectric's permittivity.

constrain, for example in free-form optimization, where the design may be given as a 2D image. In such cases, a separate deep learning model can provide an elegant solution. By training in a first step a generative model on the designs alone (without any physics knowledge), a design parameterization can be *learned* from the dataset [71, 73, 163]. Adequate models are for instance variational autoencoders (VAEs) or generative adversarial networks (GANs). These two approaches are depicted in Figure 9a, respectively 9b. For technical details and an example implementation, we refer the reader to the accompanying tutorial notebooks [52].

The latent space of such a VAE or GAN represents a learned parameterization of the designs. Thanks to the properties of the regularized latent space of VAEs or GANs, the parameterization is compact and continuous, even if the original design description was not (e.g. if a list of discrete materials is used). Please note that some techniques like variational regularization are not differentiable. Since

backpropagation necessarily requires deterministic gradients, reparametrization tricks need to be applied to the backward path in such cases [164]. Provided the VAE or GAN training did converge, this means that every point in the latent space yields a physically meaningful solution. Most importantly, the latent space of a well-trained VAE or GAN is regularized to a normal distribution with unitary standard deviation.

This means that in an iterative optimization loop, we can now constrain the latent variable that describes the designs to the numerical range of a normal distribution (e.g. to a 2σ confidence interval). This is then equivalent to constraining the entire problem to the interpolation regime of the dataset. Practically we replace the original design parameters by the latent input variable of the trained geometry generator, as depicted in Figure 10. In the subsequent optimization, the latent input can be conveniently constrained to a normal distribution e.g. by simply penalizing large values as in inspirational
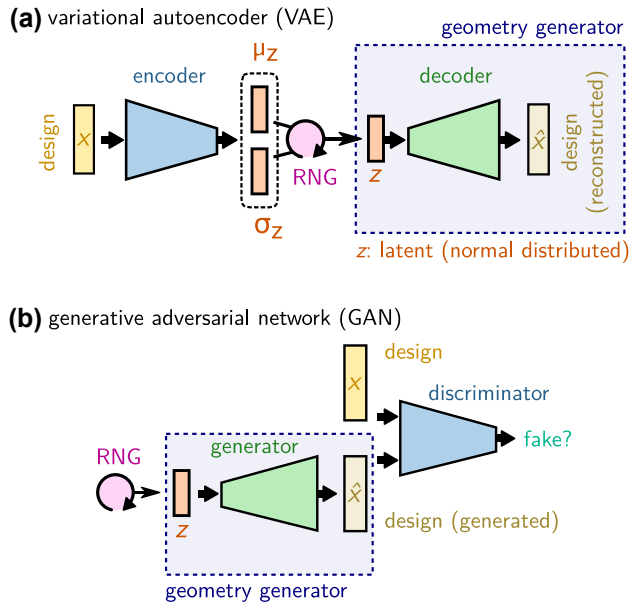
(a) variational autoencoder (VAE)

(b) generative adversarial network (GAN)

Figure 9: Avoid the extrapolation regime of the forward model with learned design parameterization. (a) Sketch of a variational autoencoder (VAE) trained to reconstruct the design. In a VAE, the encoder is trained to return the mean value (μ_z) and standard deviation (σ_z) of the latent variable z. A randomized, normal distributed latent vector is passed to the decoder for the reconstruction task (via random number generator "RNG"). By further constraining σ_z with a KL loss (c.f. text), one obtains a compact and smooth latent space that is normally distributed. (b) Sketch of a generative adversarial network (GAN). As in the VAE, by using a normal distributed random number generator during training for the latent space input, the generator develops a smooth and compact latent space, essentially representing the interpolation regime of the dataset.

generation [165], or by using a Kullback–Leibler divergence loss (KL loss), as used for variational autoencoders [69, 166].
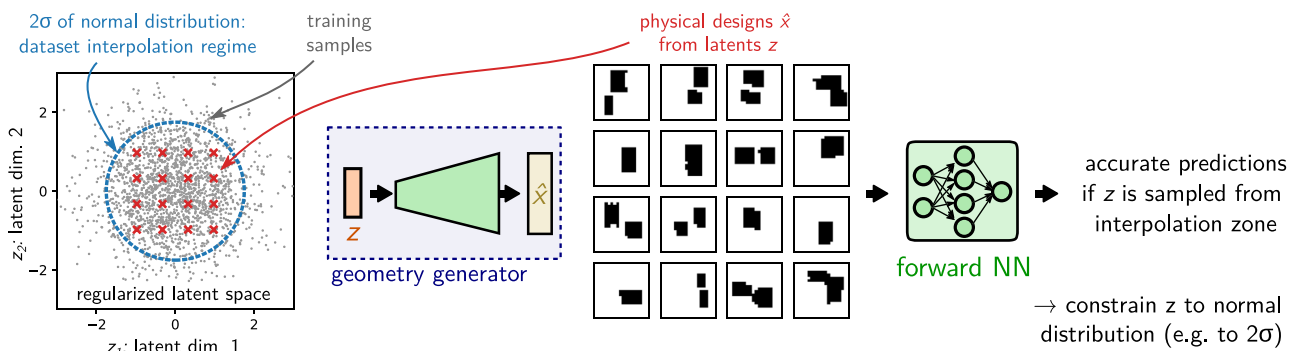
3.1.1.1 Technical hint: GAN normalization

To conclude, we like to put emphasis on an important technical detail in the generative adversarial network layout, especially in deep GAN architectures. In fact, the activation function of the generator output and the normalization of the associated data is important for a robust model design. As discussed above (c.f. sections on data normalization and on batch normalization), the statistical assumption behind deep learning is that the data follows a normal distribution with mean value of zero and unity variance (c.f. also Figure 6). Therefore, the design parameters (x in Figure 9b) should follow this assumption, since the output of the GAN generator is being fed back into the discriminator network. The most simple mean to accomplish this is to normalize the data (e.g. the design images) between $[-1, 1]$, and use a tanh activation function at the generator output.

3.1.2 Heuristics: forward network with global optimization

A robust way to overcome local extrema and assure convergence to the global optimum is using gradient-free heuristics such as evolutionary optimization or genetic algorithms [15, 167–171].

Accelerating global optimization based inverse design with deep learning is in principle straightforward. Instead of using numerical simulations, the evaluation step in the loop of the optimizer (e.g. evolutionary optimization, particle swarm, genetic algorithm …) is done with a deep learning surrogate model. This is depicted in Figure 11a. As discussed in the previous section, it is recommended to constrain the designs to the interpolation regime, for example using a generative model that precedes the physics
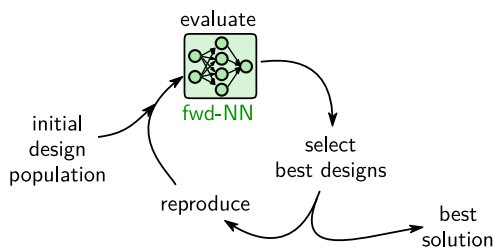

Figure 10: Re-parameterized the forward model using a learned latent representation as design input: The trained geometry generator (e.g. from a VAE or a GAN) is simply plugged before the input of the forward network. It converts a latent vector z to a physical design $\hat{x}$. If the latent space was properly regularized, sampling from within the range of a normal distribution with unitary variance will generate geometries in the interpolation regime of the training data, where the forward network works accurately. Instead of optimizing the physical design parameters, we can now run the optimization on the latent variable of the geometry generator. Constraining the numerical range of the optimization parameters accordingly, renders iterative optimization robust.

(a) forward network (physics predictor) for global optimization

(b) neural adjoint: gradient based, local optimization
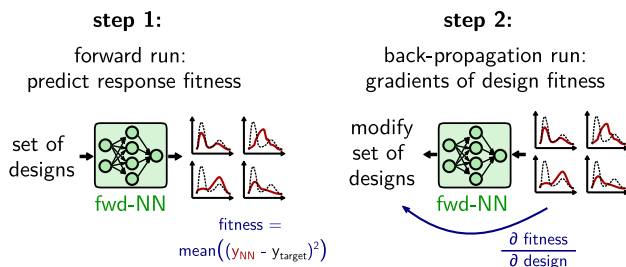
Figure 11: Inverse design via iterative optimization using a forward model as fast physics solver surrogate. (a) Use forward model to accelerate a global optimization loop. (b) Neural adjoint method: neural networks are differentiable and allow gradient based optimization. Reduce risk of local minima by operating on a large set of random initial designs. The optimization tries to minimize the error between the predicted optical property (solid red line) and the design target (dashed black line).

predictor, as illustrated in Figure 10. The actual optimization can be done with any algorithm, since the operation of the deep learning model is restricted to the calculation of the fitness function.

One of the major advantages of deep learning surrogates is their differentiability. We therefore encourage to not use gradient-free global optimization alone, but combine it with gradient-based optimization for faster convergence. This will be discussed in the following.

3.1.3 Gradient descent – neural adjoint method

While global optimization is robust and generally converges well towards the overall optimum, such methods are also inherently slow since they do not take advantage of gradients. This is unfortunate because gradients are available "for free" in deep learning surrogate models. On the other hand, gradient based approaches tend to get stuck in local minima. This can be accounted for to a certain extent, but it usually depends strongly on the individual design problem if a gradient based method will work.

The idea of gradient based optimization is the same as in the Newton–Raphson method. A fitness function is

defined such that it is a measure of the error of a solution compared to the design target. Then, the derivatives of the fitness function with respect to the design parameters of a test solution are calculated and used to modify the test-design towards the negative gradient. By minimizing the fitness function in this way, the solution iteratively gets closer and closer to the ideal design target until a minimum is reached.

Typical numerical simulation methods are not differentiable, and hence gradient based methods cannot be applied directly. While gradients can be calculated using adjoint methods [14], these still require multiple calls of the, generally, slow simulation, and hence are usually computationally expensive. Both problems can be solved to some extent by forward neural network models. A key advantage is, besides the evaluation speed, that gradients can be calculated "for free", because the network is an analytical mathematical function. For the same reason, the gradients of the surrogate model are also continuous, since this is a key requirement for the network training algorithms. As stated in the beginning, the training procedure of a neural network is in fact a gradient based optimization by itself, therefore the main functionality of all deep learning toolkits is automatic differentiation. A forward neural network model can thus always be used for gradient based inverse design, which consists of two steps that are illustrated in Figure 11b. In a first step, a set of test-designs (typically random initial values) is evaluated with the forward model. Their predicted physical behavior is compared to the design target, for which a fitness function evaluates the error between target and prediction. Now the deep learning toolkit is used to calculate the gradients of this fitness with respect to the input design parameters via backpropagation and the chain rule. Finally, the designs are modified by a small step towards the negative gradients. Repeating this procedure minimizes the fitness [29, 36, 51, 121, 172, 173].

Note that, if the gradients of the underlying physics data source are known, they can be added in the training step. This often leads to better convergence of the model, since the training can use deeper correlations to build its model [174].

As mentioned before, the main difficulty in this approach is to avoid getting stuck in local minima of the fitness function. To a certain extent this can be accounted for by optimizing a large number of random designs in parallel (see Figure 12a). While such strategy would be prohibitively expensive using numerical simulations, with a machine learning surrogate model it is typically possible to optimize several hundreds or even thousands of designs in parallel. However, depending on the problem, the number of local
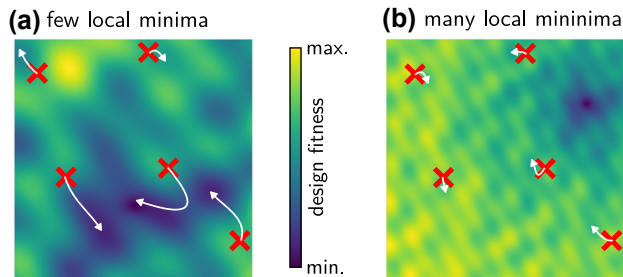
(a) few local minima   **(b)** many local mininima

Figure 12: Schematic fitness landscapes of (a) a friendly problem with relatively few local extrema. (b) A complicate problem with many local fitness minima. Using gradient based methods with a large number of initial test-sets, problem (a) will likely converge to the global optimum. In problem (b) on the contrary, the chance is high that none of the initial designs is close enough to the global optimum, and optimization will converge to a local solution. The paths taken by a gradient-based method path are indicated by white arrows.

extrema may be too large for successful convergence (see Figure 12b). This can be tested by running the optimization several times. If multiple runs do not converge to a similar solution, the parameter landscape of the problem is probably too "bumpy" for gradient based inverse design.

As explained above, it is crucial also in gradient based optimization to remain in the forward model's interpolation regime since extrapolation bears a high risk of converging towards non-physical minima of the deep learning model [121]. Also, if the dimensionality of a problem is high, the risk of strongly varying gradients further increases and optimization may always converge to unsatisfying local minima. As discussed above, in such cases it is helpful to train a separate generator network that maps the design parameters onto a regularized latent space (e.g. VAEs or GANs, c.f. also Figure 9). Instead of optimizing the physical design parameters, the optimizer then acts on this design latent space. Because the latent space is regularized, it is possible to constrain the designs to the neural network's interpolation regime e.g. by using a KL loss term in the fitness function.

Finally we want to recall once again, that unlike conventional numerical simulation methods, deep learning surrogates can possess singularities, also called failure modes or adversarial examples [120]. Gradient based optimization, especially, comes with the risk of converging to such singularities of the surrogate network. In fact, the neural adjoint method is very similar to the "fast gradient signed method" that is specifically used to find network failure modes [83]. To avoid convergence to a network singularity, it has been proposed to alternate the evaluation in the optimizer loop between the surrogate network model and exact numerical simulations. Such occasional verification of the optimized solutions effectively eliminates non-physical designs [119].

3.1.4 Hybrid approach: global optimization followed by neural adjoint

As mentioned above, inverse design tasks often possess of a large number of local extrema. As illustrated in Figure 12b, in such case gradient based algorithms may get stuck in those local extrema, even if a large number of designs is optimized in parallel. A possible solution can be a combination of iterative global optimization and local gradient based neural adjoint. In such a scenario, a global optimizer runs for a few iterations with a rather large population of solutions. During the first generations global solvers usually converge the most rapidly towards the global optimum. However, they can be expensive in the final convergence towards the exact extremum. Using the population obtained from a few iterations of a global optimization run can be very helpful as the initial set of designs for the neural adjoint method. Those designs are then relatively close to the global optimum and the chance that at least a few manage to avoid local minima is considerably increased. The approach is depicted schematically in Figure 13c.

3.2 Direct inverse design networks

In the above discussed iterative approaches, the optimality of the solution is the highest priority. Deep learning methods for direct inverse design on the other hand, put the design speed over all other criteria. The goal is to solve the inverse problem with a single network call. These approaches typically yield the ultimate acceleration, but results are generally not optimum, since no optimization algorithm pushes the solution to the extremum. These "one-shot" techniques
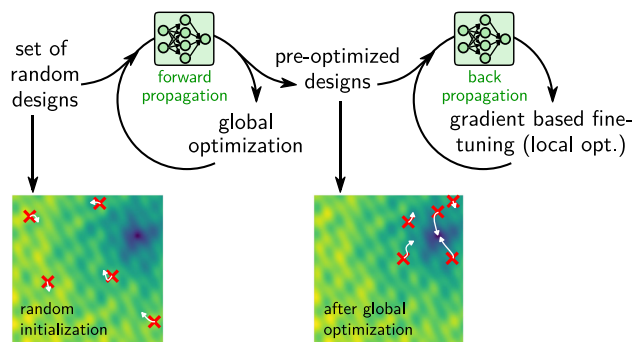

Figure 13: Global pre-optimization. If too many local minima exist, a promising solution is to start by pre-optimizing a set of random designs with global optimization. The positions of the random initial samples in an illustrative fitness landscape are depicted in the bottom left. After a few iterations of a global optimizer, the solutions are closer to the global optimum, as illustrated in the bottom right. Using this set as initial population for gradient based neural adjoint is likely to converge.

rather perform a kind of similarity matching and typically yield solutions that resemble the design target.

For the sake of accessibility we focus first on two popular variants, the tandem network and conditional variational autoencoders, for which also detailed jupyter notebook tutorials are provided as supplemental documents [52]. Subsequently we give a brief overview of other direct inverse design techniques.

3.2.1 Tandem network

One of the most simple configurations for a one-shot inverse design network is the so-called "tandem network" [33]. The tandem network is a variation on autoencoder acting on the physics domain. It takes as input the desired physical property and returns a reconstruction of these physics (for instance a target reflectivity spectrum and its reconstruction). The difference to a conventional autoencoder is that the decoder is trained in a first step on predicting the physical properties using the design parameters as input. This means, the decoder is simply a "forward" physics predictor, solving the direct problem ("fwd" in Figure 14a). Subsequently, a second training step is performed, in which the forward model weights are fixed, and the encoder, which is actually trained on generating the designs, is added to the model (generator "G" in Figure 14a). In this second step, the full model is trained, but now only the physical responses from the training set are used. The physical property (e.g. reflectivity spectrum, etc …) is fed into the encoder, which predicts a design. However, instead of comparing this design to the known one from the dataset, the generated design is fed into the forward model, that predicts the physical property of the suggestion. This predicted response is finally compared with the input response, the error between both being minimized as training loss. This means, that even if multiple possible design solutions exist, the training remains unambiguous since only the physical response of the design is evaluated, regardless of how it is achieved. The full model is then essentially an autoencoder of which the latent space is being forced to correspond to the design parameters by using the fixed, pre-trained forward network as decoder.

A practical advantage of the tandem is that the inverse problem is split in two sub-problems that are individually easier to fit, compared to end-to-end training of the full inverse problem. In a first step the forward problem is learned, which is usually a relatively straightforward task. This physics knowledge is then used in the second step to guide training of the generator network.
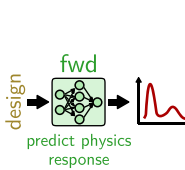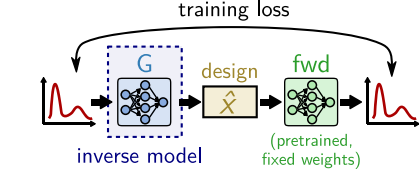
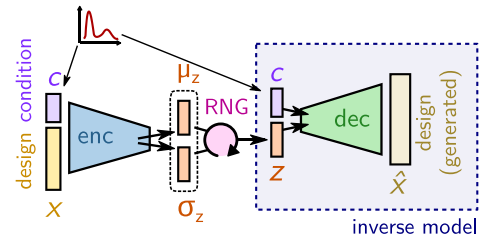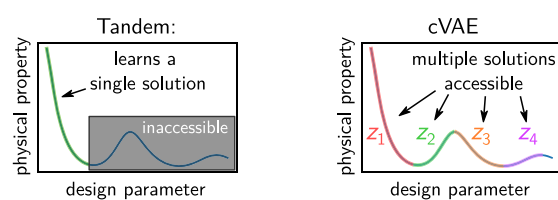**Figure 14:** Direct inverse design models. (a) Tandem model. The training is divided in two steps. At first a forward predictor is trained on the direct problem. Subsequently, the forward network is fixed and used to train the generator, (b) the conditional variational autoencoder (cVAE) is trained end-to-end in a single run. A latent space z is used to provide additional degrees of freedom to handle ambiguities in the design problem. (c) Inverse problems typically can be solved by multiple solutions. A tandem model will learn only one of possibly multiple solutions, the other remain inaccessible. The cVAE on the other hand typically learns the set of possible solutions which can be retrieved via the latent vector z.

3.2.2 Conditional variational autoencoder (cVAE)

A drawback of the tandem network is that only a single solution is learned, even if multiple designs are possible to reach the design target. Several network architectures have been developed to learn mappings to the set of multiple solutions in ambiguous inverse problems. We discuss in the following a very efficient and robust model, the conditional variational autoencoder (cVAE).

The reason why a "vanilla" variational autoencoder (VAE) cannot be directly trained on an inverse design task is the correlation between geometry and physics domain. The latent space of the autoencoder forms during training

and represents the most efficient and compact, reduced representation of the inputs. This is generally *not* the design parametrization. In consequence it is necessary to force the latent space to correspond to the design space, which is achieved in the "tandem" architecture via a two-step training procedure (see above). The tandem is hence an autoencoder with design-regularized latent space.

By *conditioning* the designs on their physical properties (here: optical), a modified variant of a VAE, a so-called *conditional* variational autoencoder (cVAE), can however be trained as an inverse design network. To this end, the classical VAE, that reproduces a design through encoder–decoder architecture, is extended by an additional input, the design condition. Here this is a physical property (e.g. a reflectivity spectrum) – the design target. As depicted in Figure 14b, this additional condition is added as input to both, the encoder (blue) as well as the decoder (green). During training, multiple possible solutions are associated with different values of the latent vector z, i.e. can be treated without training ambiguities, as illustrated in Figure 14c. After successful network training, only the decoder is used for the inverse design. The possibility to identify multiple possible solutions with a cVAE is illustrated in Figure 15 by the example of multi-layer designs for a fixed reflectivity target.

An advantage of the cVAE is its generally robust training. It often also works well with low-dimensional latent spaces, so that the latent space can be explored systematically, to identify different possible solutions [35]. A recent comparison indicates, that cVAEs are among the most effective methods for direct inverse design tasks [175].

3.2.2.1 Technical hint: regularization

As mentioned before, (c) VAEs require a latent regularization scheme in their training. The goal is for the latent space to become continuous and smooth (to allow meaningful interpolation). This is achieved using perturbative random latent sampling in the forward path, so the network learns that similar latent values correspond to similar solutions. In order to additionally achieve compactness (no blank regions in latent space), a weighted ("β"-coefficient) KL-loss is added to the training, which pushes the latent variables to a normal distribution around zero with unitary variance. If the KL loss weight is too large, the latent space will be normally distributed around zero, but reconstruction will fail. If the KL loss weight is too small, it has no effect. Then reconstruction will be good, but blank spaces in latent space may occur that do not carry useful information and impede to perform meaningful interpolation between solutions. In consequence, the weight of the KL loss with respect to the reconstruction loss needs to be carefully chosen ("β-VAE" [176]).

Unfortunately this value needs to be adapted for each problem/network model, so some trial and error is required to find the adequate value. Good starting values are typically $\beta = 0.01$ or $\beta = 0.001$. Blank latent spaces may be difficult to spot in training, so the easier approach to find a good weight is to increase the β value until the reconstruction loss starts suffering notably.

3.2.2.2 Technical hint: condition

The condition is the design target and may be a high-dimensional construct such as a reflectivity spectrum. It is possible to process the condition with a sub-network (e.g. a 1D-CNN), that can be a common, shared network, before the two input branches (encoder and decoder).

3.3 Further deep inverse design approaches

The scope of the present work is to provide an entry-level guide to deep learning for inverse design, specifically targeting a newcomer audience. We therefore focused on a few popular examples and provided a detailed discussion of
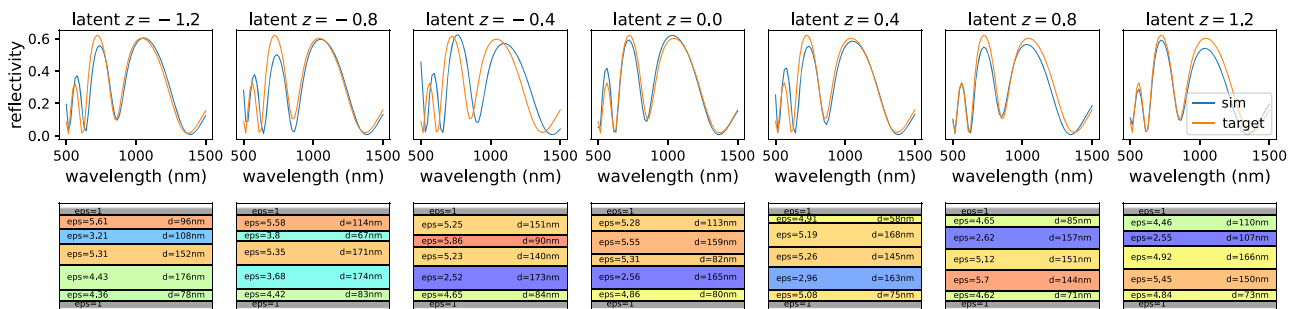

Figure 15: Dielectric layer stacks implementing an arbitrary reflectivity design spectrum. Inverse designed by a cVAE. By sweeping through the latent space of the cVAE generator with fixed target spectrum, multiple possible design solutions can be identified. Note that the cVAE discovered that mirrored structures yield the same reflectivity spectra (c.f. for example latents $z = 0$ and $z = 0.8$). A systematic latent inspection can also be done for further optimizing the solution, for example by a search for the best possible spectral match, or by identification of the most robust design, etc. The colors in the layerstack illustrations (bottom) correspond to the permittivity (c.f. color legend in Figure 8).

practical challenges. As an outlook, here we want to briefly summarize a few further deep-learning based methods that can be used for inverse design.

3.3.1 Global optimization by deep reinforcement learning

Some key successes of recent deep learning are based on combining forward modeling and direct policy learning (such as Alpha-Zero [177]). In this regard, conventional global optimization can be replaced by deep reinforcement learning (RL) [178], or [39, 179] a RL policy can be applied directly for choosing the design parameters. However, comparison with classical evolutionary methods is in most cases not clearly in favor of deep reinforcement learning, so we believe it is questionable whether the extra effort of using a less studied method is advisable at the moment [180, 181]. Please note that there are debates whether RL can indeed outperform conventional global optimization in a general picture [182].

3.3.2 Conditional generative adversarial network (cGAN)

Generative adversarial networks (GANs) can be seen as a variation of VAEs. The position of the decoder and the encoder are interchanged, and rather than compressing information, the encoder acts as a dynamically trained loss-function, which has as objective to distinguish generated samples from real data. Analogously to the case of conditional VAEs, in order to solve an ill posed inverse design problem, the GAN inputs need to be conditioned on the physics design. The GAN becomes a *conditional* GAN (cGAN).

Note that, the original GAN uses a min-max loss function between generator and discriminator, which is in practice very difficult to handle and often suffers from severe convergence problems [183]. The convergence of the shortly later proposed Wasserstein-distance loss for GAN training (WGAN) is significantly more robust [184]. Typically it is combined with gradient penalty regularization (WGAN-GP), which leads to even better robustness, and is today used in the majority of large GAN models [185].

Note also that very well performing inverse design GANs have been proposed that take inspiration from NVIDIA's "style GAN". By progressively growing the network model as well as the design resolution during training, very good design results and high accuracy can be obtained, however at the cost of a highly increased computational training budget [67, 127, 186].

3.3.3 Diffusion models

Recently, stochastic generative modeling was shown to be capable to solve inverse problems. These so-called "diffusion models" were described by Y. Song et al. by: "Creating noise from data is easy; creating data from noise is generative modeling" [187]. The key idea is inspired by thermodynamic processes and is based on performing a sequence of denoising steps by a deep learning model. The model is trained on removing a small amount of noise from input data, which was perturbed by different amounts of noise [188]. After several denoising steps, a large amount of initial noise can be entirely removed. Starting this iterative process on random noise, and combining it with some guidance using latent information about the original content, a denoising diffusion model can generate samples that match the target latent description. Such networks are very popular in computer vision and image generation ("text to image") [68, 189, 190]. Very recently, a first application of a diffusion model for metasurface inverse design has been demonstrated [191].

3.3.4 Invertible neural networks

A possibility to solve the inverse problem in a one-shot manner is invertible neural networks. Such networks are constructed using exclusively mathematical operations that guarantee that the full network remains an invertible mathematical expression, so that it is bijective. Every point in the target space matches exactly one point in the input space [192]. Obviously, this approach runs into troubles with the ill-posed character of typical inverse design problems that prohibits finding a bijective projection between physics and design space. To allow the network to learn such a bijective projection, latent dimensions need to be added to the design space, which during training are fitted to distinguish between multiple solutions or to identify cases with no solution at all [193].

3.3.5 Physics informed neural networks

As discussed above, a very interesting aspect of deep learning models is that they are differentiable. Furthermore, the basis of every deep learning application is the universal approximator theorem, stating that *any* function can be approximated with arbitrary accuracy with a sufficiently large neural network [28]. The idea behind physics informed neural networks (PINNs) [194, 195] is to learn an approximation to a solution for a partial differential equation (PDE),

by testing arbitrary values and minimizing a physics-based loss that tests the validity of the predicted solution in the PDE. PINNs are known for their extremely high accuracy and have been used for inverse design as well, where design parameters are typically included in the model and iteratively fitted during the physics-based training [196–198].

In the context of PINNs, we want to note that periodic activation functions can be very powerful in the context of differential equation solving, since their derivatives naturally represent many typical solutions [174, 199].

3.4 Improving inverse design performance

There are several ways to improve the inverse design performance. This includes neural network model optimization, tuning of the training hyper-parameters, application of regularization techniques or multi-step training methods.

Eventually, the accuracy of a deep learning model stands and falls with the quality of the data (and its quantity). As discussed before, it is crucial that the dataset is representative for the problem and that it includes as little bias as possible. However, merely generating more and more random samples has often only a very limited effect on the quality of the design predictions. A very efficient way of improving the dataset in a more purposeful manner is iterative or interactive data generation, also called active learning. The idea behind active learning is to reduce the effort of data generation by letting a neural network "learn from its own mistakes". To this end, first a network is trained on an imperfect, but cheap dataset. Subsequently, the model predictions are evaluated and data is added to the training set with a particular focus on cases where the network model performs weakly. This evaluation can be done by ensemble statistics, in which evaluations of multiple neural networks are gathered and the statistics of the predictions are used to assess the quality of the results [200]. In inverse design it is even easier, since the generated designs can be simply simulated and the results appended to the dataset [40, 127, 201].

4 Additional technical tips

4.1 Mixed precision training

If supported by the accelerator (GPU, TPU), mixed precision (bf16) training should be used. With this option, the computation device uses half precision (16 bit per number) for most calculations, which requires half the memory and runs very fast on modern hardware. On large models a factor 2 can be easily achieved both in runtime acceleration and memory

reduction, allowing larger models to fit in the GPU RAM. It also allows for larger batch sizes in the late training, which typically comes with additional acceleration.

4.2 Checkpoint saving

A callback to automatically save the best validation model can be very useful, in this way the best non-overfitted model is saved automatically even if severe overfitting occurs later in the training ("early stopping") [202].

4.3 Ensemble averaging

A typical procedure is to train a network several times, saving the best model of every run. In inference, all of these models are then used and the average solution is taken. The variance of the prediction statistics provides further information about the prediction certainty [45, 203].

4.4 Mixture of experts

For very large models, it is common practice to essentially split the model into smaller sub-networks. The details are beyond the scope of this newcomer's guide, but the technique can significantly reduce computational cost for very large architectures [204, 205].

4.5 Make the problem more specific

Often it is possible to exploit the fact that deep learning models usually perform better, the more specifically a problem is defined. It may be possible to re-formulate a problem, or to use multiple networks that predict different sub-problems, which may then be combined by a further neural network. In nano-photonics, for example in a metasurface problem, at the network prediction stage one may separate local light–matter interaction and far-field propagation, instead of directly predicting the far-field response [17, 19].

5 Tutorial notebooks

The typical workflow described above is demonstrated in a series of python notebooks as supplemental material, accessible online [52]. We demonstrate the full workflow from data generation, and data-processing, over network architecture design and hyperparameter tuning, to an implementation of the above discussed different inverse design approaches. We use two specific problems for the tutorial notebooks:

5.1 Problem 1: reflectivity of a layer stack with *PyMoosh*

The first problem used to demonstrate typical deep learning workflow is a dielectric multi-layer stack with the goal to tailor the reflectivity spectrum. For the physics calculations these tutorials use *PyMoosh* [206], the python version of *Moosh*, an s-matrix based solver for multilayer optics problems [207]. As deep learning framework, we use *Keras* [59], a *PyTorch* implementation of the notebooks is planned for the near future. For global optimization we use the package *Nevergrad* [208]. This covers following tutorials:

- Data generation: Fully random designs. link
- Data generation: Weakly optimized designs. link
- Forward network training of increasingly complex models. link
    - Forward problem: Using batch normalization for regression tasks. link
    - Forward problem: Danger of extrapolation. link
- Direct inverse design: The tandem model. link
- Direct inverse design: The conditional variational autoencoder. link
- Iterative optimization: Gradient based (neural adjoint). link
- Iterative optimization: Global pre-optimizaiton, then gradient descent. link
- Iterative/active learning: Iterative fine-tuning of the inverse design model on the actual downstream problem. link

5.2 Problem 2: scattering of dielectric nanostructures

A second problem is used to illustrate the case of structure parameterization via images (here of the geometry top-view). This is a typical scenario for many top-down fabricated nano-photonic devices like metasurfaces. We demonstrate how a Wasserstein GAN with gradient penalty [185] can be trained on learning a regularized latent description for 2D geometry top-view images. Using this in combination with a forward predictor model is then demonstrated using global and gradient based optimization of scattering spectrum inverse design, simultaneously for two incident polarizations. The nano-scattering dataset is created using simulations with the *pyGDM* toolkit [209, 210]. This problem contains following tutorials:

- Data generation: Random nano-scatterer 2D geometries. link
- Learned design parameterization: Train a WGAN on the geometries. link

- Forward Model: ResNet for nano-scattering prediction. link
- Constrained inverse design: Gradient based iterative optimization of nano-scatterer geometries in the WGAN latent space. link

6 Conclusions and perspectives

In conclusion, we provided a practical newcomer's guide to approach inverse design problems with deep learning. We gave an introduction to the key concepts in deep learning, and critically discussed how should be assessed whether deep learning is a promising strategy for solving a specific problem. We discussed guidelines for the evaluation of a dataset in a first step and for the subsequent, practical implementation and training of deep neural network architecture. We then discussed deep learning based inverse design approaches, which fall into in two main groups: iterative techniques and direct (one-shot) inverse networks. After specifically discussing the possibility to train an auxiliary network on learning a new, regularized design parameterization, we concluded with referring to a set of python tutorial notebooks, which demonstrates practically all above discussed steps and methods.

Specialists working on topics around nano-photonics, who want to apply statistical deep learning techniques to their problems, often do not have the time to learn all the subtleties of deep learning model implementation the hard way. We believe that our tutorial will be particularly useful for such deep learning newcomers, since we try to discuss various possible pitfalls and provide hints and guidelines for robust architectures that may "just work" without too much painful parameter tuning.

To conclude, we want to recall once again, that deep learning is not the solution to all problems [211]. For many applications of inverse design there exist highly optimized, specific algorithms and other solutions, which will often outperform deep learning. Considering whether a deep neural network is indeed the way to go, is an extremely important first question that one should always ask before starting. However there are many situations where deep learning can offer unique assets in inverse design. For instance if ultimate speed is required, or when a large number of design tasks need to be solved for the same problem, deep learning can be a literal game changer. If, for instance, a large dataset for a photonic platform exists and struggles understanding the underlying physical mechanisms, deep learning offers an ideal platform with latent space methods. Finally, a key property of deep learning models is automatic

differentiation, which allows to build analytical, differentiable models from empirical data, for instance from experimental measurements. We believe that, used with the necessary caution and upon proper verification of the predictions, deep learning offers a very powerful platform not only for inverse design problems but far beyond.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Conflict of interest: Authors state no conflicts of interest.

Informed consent: Informed consent was obtained from all individuals included in this study.

Ethical approval: The conducted research is not related to either human or animals use.

Data availability: All data and python scripts to reproduce the results are published by the authors as open source and are available online (https://gitlab.com/wiechapeter/newcomer_guide_dl_inversedesign).

References

[1] P. Mühlschlegel, H.-J. Eisler, O. J. F. Martin, B. Hecht, and D. W. Pohl, "Resonant optical antennas," *Science*, vol. 308, p. 1607, 2005.

[2] C. Girard, "Near fields in nanostructures," *Rep. Prog. Phys.*, vol. 68, p. 1883, 2005.

[3] L. Novotny and B. Hecht, *Principles of Nano-Optics*, Cambridge, New York, Cambridge University Press, 2006.

[4] A. I. Kuznetsov, A. E. Miroshnichenko, M. L. Brongersma, Y. S. Kivshar, and B. Luk'yanchuk, "Optically resonant dielectric nanostructures," *Science*, vol. 354, p. aag2472, 2016.

[5] C. Girard and E. Dujardin, "Near-field optical properties of *top-down* and *bottom-up* nanostructures," *J. Opt. A: Pure Appl. Opt.*, vol. 8, p. S73, 2006.

[6] J. B. Pendry, "Negative refraction makes a perfect lens," *Phys. Rev. Lett.*, vol. 85, p. 3966, 2000.

[7] P. R. Wiecha, A. Cuche, A. Arbouet, et al., "Strongly directional scattering from dielectric nanowires," *ACS Photonics*, vol. 4, p. 2036, 2017.

[8] M. Kauranen and A. V. Zayats, "Nonlinear plasmonics," *Nat. Photonics*, vol. 6, p. 737, 2012.

[9] P. Genevet, F. Capasso, F. Aieta, M. Khorasaninejad, and R. Devlin, "Recent advances in planar optics: from plasmonic to dielectric metasurfaces," *Optica*, vol. 4, p. 139, 2017.

[10] G. Colas des Francs, J. Barthes, A. Bouhelier, et al., "Plasmonic Purcell factor and coupling efficiency to surface plasmons. Implications for addressing and controlling optical nanosources," *J. Opt.*, vol. 18, p. 094005, 2016.

[11] J. Wang, F. Sciarrino, A. Laing, and M. G. Thompson, "Integrated photonic quantum technologies," *Nat. Photonics*, vol. 14, p. 273, 2020.

[12] P. R. Wiecha, C. Majorel, C. Girard, et al., "Enhancement of electric and magnetic dipole transition of rare-earth-doped thin films tailored by high-index dielectric nanostructures," *Appl. Opt.*, vol. 58, p. 1682, 2019.

[13] J. Hadamard, "Sur les problèmes aux dérivés partielles et leur signification physique," *Princet. Univ. Bullet.*, vol. 13, p. 49, 1902.

[14] J. S. Jensen and O. Sigmund, "Topology optimization for nano-photonics," *Laser Photonics Rev.*, vol. 5, p. 308, 2011.

[15] M. M. R. Elsawy, S. Lanteri, R. Duvigneau, J. A. Fan, and P. Genevet, "Numerical optimization methods for metasurfaces," *Laser Photonics Rev.*, vol. 14, p. 1900445, 2020.

[16] I. Malkiel, M. Mrejen, A. Nagler, U. Arieli, L. Wolf, and H. Suchowski, "Plasmonic nanostructure design and characterization via Deep Learning," *Light: Sci. Appl.*, vol. 7, p. 60, 2018.

[17] P. R. Wiecha and O. L. Muskens, "Deep learning meets nanophotonics: a generalized accurate predictor for near fields and far fields of arbitrary 3D nanostructures," *Nano Lett.*, vol. 20, p. 329, 2020.

[18] A.-P. Blanchard-Dionne and O. J. F. Martin, "Teaching optics to a machine learning network," *Opt. Lett.*, vol. 45, p. 2922, 2020.

[19] M. Chen, R. Lupoiu, C. Mao, et al., "High speed simulation and freeform optimization of nanophotonic devices with physics-augmented deep learning," *ACS Photonics*, vol. 9, p. 3110, 2022.

[20] T. Ma, H. Wang, and L. J. Guo, "OptoGPT: a foundation model for inverse design in optical multilayer thin film structures," 2023, arxiv:2304.10294 [physics].

[21] A. Krizhevsky, I. Sutskever, G. E. Hinton, et al., "ImageNet classification with deep convolutional neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 25, p. 1097, 2012.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for Image recognition", 2015, arxiv:1512.03385 [cs].

[23] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: a review," *Neurocomputing*, vol. 187, p. 27, 2016.

[24] A. Kirillov, E. Mintun, N. Ravi, et al., "Segment anything," 2023, arxiv:2304.02643 [cs].

[25] M. Sundermeyer, R. Schlüter, and H. Ney, *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[26] T. B. Brown, B. Mann, N. Ryder, et al., "Language models are few-shot learners," *Adv. Neural Inf. Process. Syst.*, vol. 300, pp. 1877−1901, 2020.

[27] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, p. 604, 2021.

[28] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, p. 359, 1989.

[29] J. Peurifoy, Y. Shen, L. Jing, et al., "Nanophotonic particle simulation and inverse design using artificial neural networks," *Sci. Adv.*, vol. 4, p. eaar4206, 2018.

[30] A. Estrada-Real, A. Khaireh-Walieh, B. Urbaszek, and P. R. Wiecha, "Inverse design with flexible design targets via deep learning: tailoring of electric and magnetic multipole scattering from nano-spheres," *Photonics Nanostructures — Fundam. Appl.*, vol. 52, p. 101066, 2022.

[31] J. Jiang and J. A. Fan, "Global optimization of dielectric metasurfaces using a physics-driven neural network," *Nano Lett.*, vol. 19, p. 5366, 2019.

[32] J. Jiang and J. A. Fan, "Simulator-based training of generative neural networks for the inverse design of metasurfaces," *Nanophotonics*, vol. 9, p. 1059, 2019.

[33] D. Liu, Y. Tan, E. Khoram, and Z. Yu, "Training deep neural networks for the inverse design of nanophotonic structures," *ACS Photonics*, vol. 5, p. 1365, 2018.

[34] R. Unni, K. Yao, and Y. Zheng, "Deep convolutional mixture density network for inverse design of layered photonic structures," *ACS Photonics*, vol. 7, no. 10, pp. 2703−2712, 2020.

[35] P. Dai, K. Sun, X. Yan, et al., "Inverse design of structural color: finding multiple solutions *via* conditional generative adversarial networks," *Nanophotonics*, vol. 11, p. 3057, 2022.

[36] T. Asano and S. Noda, "Iterative optimization of photonic crystal nanocavity designs by using deep neural networks," *Nanophotonics*, vol. 8, p. 2243, 2019.

[37] T. Zhang, J. Wang, Q. Liu, et al., "Efficient spectrum prediction and inverse design for plasmonic waveguide systems based on artificial neural networks," *Photonics Res.*, vol. 7, p. 368, 2019.

[38] M. H. Tahersima, K. Kojima, T. Koike-Akino, et al., "Deep neural network inverse design of integrated photonic power splitters," *Sci. Rep.*, vol. 9, p. 1368, 2019.

[39] S. Banerji, A. Majumder, A. Hamrick, R. Menon, and B. Sensale-Rodriguez, "Machine learning enables design of on-chip integrated silicon T-junctions with footprint of 1.2 μm×1.2 μm," *Nano Commun. Netw.*, vol. 25, p. 100312, 2020.

[40] N. J. Dinsdale, P. R. Wiecha, M. Delaney, et al., "Deep learning enabled design of complex transmission matrices for universal optical components," *ACS Photonics*, vol. 8, p. 283, 2021.

[41] J. Zhou, B. Huang, Z. Yan, and J.-C. G. Bünzli, "Emerging role of machine learning in light-matter interaction," *Light: Sci. Appl.*, vol. 8, p. 1, 2019.

[42] S. So, T. Badloe, J. Noh, J. Bravo-Abad, and J. Rho, "Deep learning enabled inverse design in nanophotonics," *Nanophotonics*, vol. 9, p. 1041, 2020.

[43] J. Jiang, M. Chen, and J. A. Fan, "Deep neural networks for the evaluation and design of photonic devices," *Nat. Rev. Mater.*, vol. 6, p. 679, 2021.

[44] Z. Liu, D. Zhu, L. Raju, and W. Cai, "Tackling photonic inverse design with machine learning," *Adv. Sci.*, vol. 8, p. 2002923, 2021.

[45] P. R. Wiecha, A. Arbouet, C. Girard, and O. L. Muskens, "Deep learning in nano-photonics: inverse design and beyond," *Photonics Res.*, vol. 9, p. B182, 2021.

[46] Y. Deng, S. Ren, J. Malof, and W. J. Padilla, "Deep inverse photonic design: a tutorial," *Photonics Nanostructures — Fundam. Appl.*, vol. 52, p. 101070, 2022.

[47] K. Yao and Y. Zheng, *Nanophotonics and Machine Learning — Concepts, Fundamentals, and Applications, Springer Series in Optical Sciences*, Cham, Switzerland, Springer, 2023.

[48] W. Ji, J. Chang, H.-X. Xu, et al., "Recent advances in metasurface design and quantum optics applications with machine learning, physics-informed neural networks, and topology optimization methods," *Light: Sci. Appl.*, vol. 12, p. 169, 2023.

[49] P.-I. Schneider, X. Garcia Santiago, V. Soltwisch, M. Hammerschmidt, S. Burger, and C. Rockstuhl, "Benchmarking five global optimization approaches for nano-optical shape optimization and parameter reconstruction," *ACS Photonics*, vol. 6, p. 2726, 2019.

[50] R. S. Hegde, "Deep learning: a new tool for photonic nanostructure design," *Nanoscale Adv.*, vol. 2, p. 1007, 2020.

[51] S. Ren, A. Mahendra, O. Khatib, Y. Deng, W. J. Padilla, and J. M. Malof, "Inverse deep learning methods and benchmarks for artificial electromagnetic material design," *Nanoscale*, vol. 14, p. 3958, 2022.

[52] P. R. Wiecha, "A newcomer's guide to deep learning for inverse design in nano-photonics," 2023. Available at: https://gitlab.com/wiechapeter/newcomer_guide_dl_inversedesign.

[53] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, p. 400, 1951.

[54] L. Heinrich, *PyHEP 2020 Autodiff Tutorial*, 2020. Available at: https://github.com/lukasheinrich/pyhep2020-autodiff-tutorial.

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[56] D. P. Kingma and J. Ba, 2014, arXiv:1412.6980 [cs].

[57] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019, arxiv:1711.05101 [cs, math].

[58] A. Paszke, S. Gross, F. Massa, et al., "PyTorch: an imperative style, high-performance deep learning library," 2019, arxiv:1912.01703 [cs, stat].

[59] M. Abadi, A. Agarwal, P. Barham, et al., 2015. Available at: https://www.tensorflow.org/.

[60] F. Chollet, *Deep Learning with Python*, Manning Publications Company, 2017.

[61] J. Heek, A. Levskaya, A. Oliver, et al., "Flax: a neural network library and ecosystem for JAX," 2023. Available at: http://github.com/google/flax.

[62] T. Chen, M. Li, Y. Li, et al., "MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems," 2015, arxiv:1512.01274 [cs].

[63] J. Kaplan, S. McCandlish, T. Henighan, et al., "Scaling laws for neural language models," 2020, arxiv:2001.08361 [cs, stat].

[64] J. Yu, Y. Xu, J. Y. Koh, et al., "Scaling autoregressive models for content-rich text-to-image generation," 2022, arxiv:2206.10789 [cs].

[65] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, p. 41, 1997.

[66] D. J. Wu, "Accelerating self-play learning in go," 2020, arxiv:1902.10565 [cs, stat].

[67] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of StyleGAN," 2020, arxiv:1912.04958 [cs, eess, stat].

[68] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022, arxiv:2112.10752 [cs].

[69] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Found. Trends Mach. Learn.*, vol. 12, p. 307, 2019.

[70] A. Khaireh-Walieh, A. Arnoult, S. Plissard, and P. R. Wiecha, "Monitoring MBE substrate deoxidation via RHEED image-sequence analysis by deep learning," *Cryst. Growth Des.*, vol. 23, p. 892, 2023.

[71] D. Melati, Y. Grinberg, M. Kamandar Dezfouli, et al., "Mapping the global design space of nanophotonic components using machine learning pattern recognition," *Nat. Commun.*, vol. 10, p. 4775, 2019.

[72] Y. Kiarashinejad, M. Zandehshahvar, S. Abdollahramezani, O. Hemmatyar, R. Pourabolghasem, and A. Adibi, "Knowledge discovery in nanophotonics using geometric deep learning," *Adv. Intell. Syst.*, vol. 2, p. 1900132, 2020.

[73] M. Zandehshahvar, Y. Kiarashinejad, M. Zhu, H. Maleki, T. Brown, and A. Adibi, "Manifold learning for knowledge discovery and intelligent inverse design of photonic nanostructures: breaking the geometric complexity," *ACS Photonics*, vol. 9, p. 714, 2022.

[74] R. Bachmann, D. Mizrahi, A. Atanov, and A. Zamir, *Computer Vision — ECCV 2022, Lecture Notes in Computer Science*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., Cham, Springer Nature Switzerland, 2022, pp. 348−367.

[75] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, p. 550, 2023.

[76] L. Li and A. Talwalkar, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, PMLR, 2020, pp. 367−377.

[77] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 4095−4104.

[78] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, Honolulu, Hawaii, USA, AAAI'19/IAAI'19/EAAI'19 AAAI Press, 2019, pp. 4780−4789.

[79] M. Hammerschmidt, P.-I. Schneider, X. G. Santiago, L. Zschiedrich, M. Weiser, and S. Burger, *Computational Optics II*, vol. 10694, Frankfurt, Germany, SPIE, 2018, pp. 38−45.

[80] X. Garcia-Santiago, S. Burger, C. Rockstuhl, and P.-I. Schneider, "Bayesian optimization with improved scalability and derivative information for efficient design of nanophotonic structures," *J. Lightwave Technol.*, vol. 39, p. 167, 2021.

[81] T. Wu, D. Arrivault, M. Duruflé, et al., "Efficient hybrid method for the modal analysis of optical microcavities and nanoresonators," *JOSA A*, vol. 38, p. 1224, 2021.

[82] M. M. R. Elsawy, A. Gourdin, M. Binois, et al., "Multiobjective statistical learning optimization of RGB metalens," *ACS Photonics*, vol. 8, p. 2498, 2021.

[83] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015, arxiv:1412.6572 [cs, stat].

[84] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, arxiv:1409.1556 [cs].

[85] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is all you need," 2017, arxiv:1706.03762 [cs].

[86] K. Fukushima, "Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, p. 193, 1980.

[87] L. Atlas, T. Homma, and R. Marks, *Neural Information Processing Systems*, Denver, United States, American Institute of Physics, 1987.

[88] Y. LeCun, B. Boser, J. S. Denker, et al., "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, p. 541, 1989.

[89] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, "Parallel distributed processing model with local space-invariant interconnections and its optical architecture," *Appl. Opt.*, vol. 29, p. 4790, 1990.

[90] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *J. Physiol.*, vol. 148, p. 574, 1959.

[91] D. Lowe, *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150−1157.

[92] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, p. 6999, 2022.

[93] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *ECCV*, pp. 630−645, 2016.

[94] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," 2016, arxiv:1603.09382 [cs].

[95] M. Lin, Q. Chen, and S. Yan, "Network in network," 2014, arxiv:1312.4400 [cs].

[96] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735, 1997.

[97] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer, and J. F. Kolen, Eds., IEEE Press, 2001.

[98] I. Sutskever, O. Vinyals, and Q. V. Le, *Advances in Neural Information Processing Systems*, vol. 27, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., Curran Associates, Inc., 2014.

[99] S. M. Lakew, M. Cettolo, and M. Federico, "A comparison of transformer and recurrent neural networks on multilingual neural machine translation," 2018, arxiv:1806.06957 [cs].

[100] T. Wolf, L. Debut, V. Sanh, et al., *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Association for Computational Linguistics, 2020, pp. 38−45.

[101] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, p. 61, 2009.

[102] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: grids, groups, graphs, geodesics, and gauges," 2021, arxiv:2104.13478 [cs, stat].

[103] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017, arxiv:1609.02907 [cs, stat].

[104] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018, arxiv:1710.10903 [cs, stat].

[105] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017, arxiv:1511.05493 [cs, stat].

[106] S. Deshpande, S. P. A. Bordas, and J. Lengiewicz, "MAgNET: a graph U-net architecture for mesh-based simulations," 2023, arxiv:2211.00713 [cs].

[107] E. Khoram, Z. Wu, Y. Qu, M. Zhou, and Z. Yu, "Graph neural networks for metasurface modeling," *ACS Photonics*, vol. 10, p. 892, 2023.

[108] L. Kuhn, T. Repän, and C. Rockstuhl, "Exploiting graph neural networks to perform finite-difference time-domain based optical simulations," *APL Photonics*, vol. 8, p. 036109, 2023.

[109] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016, arxiv:1409.0473 [cs, stat].

[110] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," 2020, arxiv:1911.03584 [cs, stat].

[111] A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al., "An image is worth 16x16 words: transformers for image recognition at scale," 2021, arxiv:2010.11929 [cs].

[112] Z. Liu, Y. Lin, Y. Cao, et al., "Swin transformer: hierarchical vision transformer using shifted windows," 2021, arxiv:2103.14030 [cs].

[113] M. Naseer, K. Ranasinghe, S. Khan, M. Hayat, F. S. Khan, and M.-H. Yang, "Intriguing properties of vision transformers," 2021, arxiv:2105.10497 [cs].

[114] T. Xiao, M. Singh, E. Mintun, T. Darrell, P. Dollár, and R. Girshick, "Early convolutions help transformers see better," 2021, arxiv:2106.14881 [cs].

[115] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "CoAtNet: marrying convolution and attention for all data sizes," 2021, arxiv:2106.04803 [cs].

[116] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," 2022, arxiv:2201.03545 [cs].

[117] S. H. Lee, S. Lee, and B. C. Song, "Vision transformer for small-size datasets," 2021, arxiv:2112.13492.

[118] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," 2017, arxiv:1611.05431 [cs].

[119] R. S. Hegde, "Photonics inverse design: pairing deep neural networks with evolutionary algorithms," *IEEE J. Sel. Top. Quantum Electron.*, vol. 26, pp. 1−8, 2020.

[120] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," 2017, arxiv:1702.02284 [cs, stat].

[121] Y. Deng, S. Ren, K. Fan, J. M. Malof, and W. J. Padilla, "Neural-adjoint method for the inverse design of all-dielectric metasurfaces," *Opt. Express*, vol. 29, p. 7526, 2021.

[122] Ž. Lukšič, J. Tanevski, S. Džeroski, and L. Todorovski, "Meta-model framework for surrogate-based parameter estimation in dynamical systems," *IEEE Access*, vol. 7, p. 181829, 2019.

[123] K. Khowaja, M. Shcherbatyy, and W. K. Härdle, "Surrogate models for optimization of dynamical systems," 2021, arxiv:2101.10189 [math, stat].

[124] L. Hu, J. Chen, V. N. Nair, and A. Sudjianto, "Surrogate locally-interpretable models with supervised machine learning algorithms," 2020, arxiv:2007.14528 [cs, stat].

[125] A. A. Popov and A. Sandu, "Multifidelity ensemble kalman filtering using surrogate models defined by physics-informed autoencoders," 2021, arxiv:2102.13025 [cs, math].

[126] A. J. Dave, J. Wilson, and K. Sun, "Deep surrogate models for multi-dimensional regression of reactor power," 2020, arxiv:2007.05435 [physics].

[127] F. Wen, J. Jiang, and J. A. Fan, "Robust freeform metasurface design based on progressively growing generative networks," *ACS Photonics*, vol. 7, p. 2098, 2020.

[128] M. D. Zeiler and R. Fergus, *Computer Vision — ECCV 2014, Lecture Notes in Computer Science*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham, Springer International Publishing, 2014, pp. 818−833.

[129] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: convolutional networks for biomedical image segmentation", 2015, arxiv:1505.04597 [cs].

[130] F. Provost, D. Jensen, and T. Oates, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, KDD '99 Association for Computing Machinery, 1999, pp. 23−32.

[131] J. Bierkens, P. Fearnhead, and G. Roberts, "The Zig-Zag process and super-efficient sampling for Bayesian analysis of big data," *Ann. Stat.*, vol. 47, p. 1288, 2019.

[132] M. Renardy, L. R. Joslyn, J. A. Millar, and D. E. Kirschner, "To Sobol or not to Sobol? The effects of sampling schemes in systems biology applications," *Math. Biosci.*, vol. 337, p. 108593, 2021.

[133] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, p. 954, 2000.

[134] J. Karvanen, "Estimation of quantile mixtures via L-moments and trimmed L-moments," *Comput. Stat. Data Anal.*, vol. 51, p. 947, 2006.

[135] O. Khatib, S. Ren, J. Malof, and W. J. Padilla, "Learning the physics of all-dielectric metamaterials with deep lorentz neural networks," *Adv. Opt. Mater.*, vol. 10, p. 2200097, 2022.

[136] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," 2017, arxiv:1701.04128 [cs].

[137] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929, 2014.

[138] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias−variance trade-off," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 116, p. 15849, 2019.

[139] M. Loog, T. Viering, A. Mey, J. H. Krijthe, and D. M. J. Tax, "A brief prehistory of double descent," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 117, p. 10625, 2020.

[140] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: where bigger models and more data hurt*," *J. Stat. Mech. Theory Exp.*, vol. 2021, p. 124003, 2021.

[141] R. Schaeffer, M. Khona, Z. Robertson, et al., "Double descent demystified: identifying, interpreting & ablating the sources of a deep learning puzzle," 2023, arxiv:2303.14151 [cs, stat].

[142] S. Bubeck, V. Chandrasekaran, R. Eldan, et al., "Sparks of artificial general intelligence: early experiments with GPT-4," 2023, arxiv:2303.12712 [cs].

[143] R. Bommasani, D. A. Hudson, E. Adeli, et al., "On the opportunities and risks of foundation models," 2022, arxiv:2108.07258 [cs].

[144] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", 2015, arxiv:1502.03167 [cs].

[145] P. Mianjy, R. Arora, and R. Vidal, "On the implicit bias of dropout," 2018, arxiv:1806.09777 [cs, stat].

[146] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," 2018, arxiv:1801.05134 [cs, stat].

[147] A. Brock, S. De, and S. L. Smith, "Characterizing signal propagation to close the performance gap in unnormalized ResNets," 2021, arxiv:2101.08692 [cs, stat].

[148] X. Lian and J. Liu, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 3254−3263.

[149] A. Özgür and F. Nar, *2020 28th Signal Processing and Communications Applications Conference (SIU)*, 2020, pp. 1−4.

[150] Y. Wu and J. Johnson, "Rethinking "batch" in BatchNorm," 2021, arxiv:2105.07576 [cs].

[151] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science*, G. B. Orr, and K.-R. Müller, Eds., Berlin, Heidelberg, Springer, 1998, pp. 9−50.

[152] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: generalization gap and sharp minima," 2017, arxiv:1609.04836 [cs, math].

[153] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018, arxiv:1804.07612 [cs, stat].

[154] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size", 2018, arxiv:1711.00489 [cs, stat].

[155] Q. Fournier and D. Aloise, *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2019, pp. 211−214.

[156] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN," *ACM Trans. Database Syst.*, vol. 42, no. 19, p. 1, 2017.

[157] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579−2605, 2008.

[158] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273−297, 1995.

[159] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5−32, 2001.

[160] F. Pedregosa, G. Varoquaux, A. Gramfort, et al., "Scikit-learn: machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825−2830, 2011.

[161] R. Hegde, "Sample-efficient deep learning for accelerating photonic inverse design," *OSA Continuum*, vol. 4, p. 1019, 2021.

[162] C. Majorel, C. Girard, A. Arbouet, O. L. Muskens, and P. R. Wiecha, "Deep learning enabled strategies for modeling of complex aperiodic plasmonic metasurfaces of arbitrary size," *ACS Photonics*, vol. 9, p. 575, 2022.

[163] Z. Liu, L. Raju, D. Zhu, and W. Cai, "A hybrid strategy for the discovery and design of photonic structures," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, p. 126, 2020.

[164] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022, arxiv:1312.6114 [cs, stat].

[165] B. Rozière, M. Riviere, O. Teytaud, J. Rapin, Y. LeCun, and C. Couprie, "Inspirational adversarial image generation," 2021, arxiv:1906.11661 [cs, stat].

[166] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, p. 79, 1951.

[167] P. R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, and V. Paillard, "Evolutionary multi-objective optimization of colour pixels based on dielectric nanoantennas," *Nat. Nanotechnol.*, vol. 12, p. 163, 2017.

[168] P. R. Wiecha, C. Majorel, C. Girard, et al., "Design of plasmonic directional antennas via evolutionary optimization," *Opt. Express*, vol. 27, p. 29069, 2019.

[169] J. Liu, A. Moreau, M. Preuss, et al., *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, New York, NY, USA,

GECCO '20 Association for Computing Machinery, 2020, pp. 620−628.

[170] M. A. Barry, V. Berthier, B. D. Wilts, et al., "Evolutionary algorithms converge towards evolved biological photonic structures," *Sci. Rep.*, vol. 10, p. 12024, 2020.

[171] Y. Brûlé, P. Wiecha, A. Cuche, V. Paillard, and G. C. Des Francs, "Magnetic and electric Purcell factor control through geometry optimization of high index dielectric nanostructures," *Opt. Express*, vol. 30, p. 20360, 2022.

[172] Y. Jing, H. Chu, B. Huang, J. Luo, W. Wang, and Y. Lai, "A deep neural network for general scattering matrix," *Nanophotonics*, vol. 12, no. 13, pp. 2583−2591, 2023.

[173] Y. Augenstein, T. Repän, and C. Rockstuhl, "Neural operator-based surrogate solver for free-form electromagnetic inverse design," *ACS Photonics*, vol. 10, p. 1547, 2023.

[174] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 7462−7473.

[175] T. Ma, M. Tobah, H. Wang, and L. J. Guo, "Benchmarking deep learning-based models on nanophotonic inverse design problems," *Opto-Electron. Sci.*, vol. 1, p. 210012, 2022.

[176] I. Higgins, L. Matthey, A. Pal, et al., *ICLR Conference*, 2017.

[177] D. Silver, T. Hubert, J. Schrittwieser, et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017, arxiv:1712.01815 [cs].

[178] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017, arxiv:1611.01578 [cs].

[179] H. Wang, Z. Zheng, C. Ji, and L. J. Guo, "Automated multi-layer optical design via deep reinforcement learning," *Mach. Learn.: Sci. Technol.*, vol. 2, p. 025013, 2020.

[180] E. Real, S. Moore, A. Selle, et al., "Large-scale evolution of image classifiers," 2017, arxiv:1703.01041 [cs].

[181] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, *Proceedings of the 2023 International Symposium on Physical Design*, New York, NY, USA, ISPD '23 Association for Computing Machinery, 2023, pp. 158−166.

[182] I. L. Markov, "The false dawn: reevaluating google's reinforcement learning for chip macro placement," 2023, arxiv:2306.09633 [cs].

[183] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, et al., 2014, arxiv:1406.2661 [cs, stat].

[184] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, arxiv:1701.07875 [cs, stat].

[185] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," 2017, arxiv:1704.00028 [cs, stat].

[186] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019, arxiv:1812.04948 [cs, stat].

[187] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," 2021, arxiv:2011.13456 [cs, stat].

[188] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, 2015, pp. 2256−2265.

[189] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: a survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, p. 10850, 2023.

[190] Z. Chang, G. A. Koulieris, and H. P. H. Shum, "On the design fundamentals of diffusion models: a survey," 2023, arxiv:2306.04542 [cs].

[191] Z. Zhang, C. Yang, Y. Qin, H. Feng, J. Feng, and H. Li, "Diffusion probabilistic model based accurate and high-degree-of-freedom metasurface inverse design," *Nanophotonics*, vol. 12, no. 20, pp. 3871−3881, 2023.

[192] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," 2019, arxiv:1811.00995 [cs, stat].

[193] L. Ardizzone, J. Kruse, S. Wirkert, et al., 2018, arxiv:1808.04730 [cs, stat].

[194] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, p. 686, 2019.

[195] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" 2023, arxiv:2302.04107 [cs, math].

[196] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro, "Physics-informed neural networks for inverse problems in nano-optics and metamaterials," *Opt. Express*, vol. 28, p. 11618, 2020.

[197] Z. Fang and J. Zhan, "Deep physical informed neural networks for metamaterial design," *IEEE Access*, vol. 8, p. 24506, 2020.

[198] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, "Physics-informed neural networks with hard constraints for inverse design," *SIAM J. Sci. Comput.*, vol. 43, p. B1105, 2021.

[199] S. Klocek, Ł. Maziarka, M. Wołczyk, J. Tabor, J. Nowak, and M. Śmieja, *Artificial Neural Networks and Machine Learning − ICANN 2019: Workshop and Special Sessions, Lecture Notes in Computer Science*, I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis, Eds., Cham, Springer International Publishing, 2019, pp. 496−510.

[200] R. Pestourie, Y. Mroueh, T. V. Nguyen, P. Das, and S. G. Johnson, "Active learning of deep surrogates for PDEs: application to metasurface design", 2020, arxiv:2008.12649 [physics].

[201] A.-P. Blanchard-Dionne and O. J. F. Martin, "Successive training of a generative adversarial network for the design of an optical cloak," *OSA Continuum*, vol. 4, p. 87, 2021.

[202] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constr. Approx.*, vol. 26, p. 289, 2007.

[203] S. Wang, K. Fan, N. Luo, et al., "Massive computational acceleration by using neural networks to emulate mechanism-based biological models," *Nat. Commun.*, vol. 10, p. 4354, 2019.

[204] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," 2014, arxiv:1312.4314 [cs].

[205] N. Shazeer, A. Mirhoseini, K. Maziarz, et al., "Outrageously large neural networks: the sparsely-gated mixture-of-experts layer," 2017, arxiv:1701.06538 [cs, stat].

[206] A. Moreau, "PyMoosh," 2023. Available at: https://github.com/AnMoreau/PyMoosh.

[207] J. Defrance, C. Lemaître, R. Ajib, et al., "Moosh: a numerical Swiss army knife for the optics of multilayers in octave/matlab," *J. Open Res. Softw.*, vol. 4, p. e13, 2016.

[208] P. Bennet, C. Doerr, A. Moreau, J. Rapin, F. Teytaud, and O. Teytaud, "Nevergrad: black-box optimization platform," *ACM SIGEVOlution*, vol. 14, p. 8, 2021.

[209] P. R. Wiecha, "pyGDM—a python toolkit for full-field electro-dynamical simulations and evolutionary optimization of nanostructures," *Comput. Phys. Commun.*, vol. 233, p. 167, 2018.

[210] P. R. Wiecha, C. Majorel, A. Arbouet, et al., ""pyGDM" − new functionalities and major improvements to the python toolkit for nano-optics full-field simulations," *Comput. Phys. Commun.*, vol. 270, p. 108142, 2022.

[211] P. R. Wiecha, "Deep learning for nano-photonic materials − the solution to everything!?" 2023, arxiv:2310.08618 [physics].