Optical Engineering

OpticalEngineering.SPIEDigitalLibrary.org

Accelerating optics design optimizations with deep learning

Ravi S. Hegde

Accelerating optics design optimizations with deep learning

Ravi S. Hegde*
Indian Institute of Technology, Department of Electrical Engineering, Gandhinagar, Gujarat, India

Abstract. We show that design optimizations, an integral but time-consuming component of optical engineering, can be significantly sped-up when paired with deep neural networks (DNNs). By using the DNN indirectly for choosing initializations and candidate preselection, our approach obviates the need for large networks, big datasets, long training epochs, and excessive hyperparameter optimization. For a 16-layered thin-film design problem, our surrogate-assisted differential evolution (DE) algorithm is able to achieve similar optimal solutions as that of an unassisted DE using only 10% of the function evaluation budget. Our approach is a promising option for the optimal design of optical devices and systems. *© 2019 Society of Photo-Optical Instrumentation Engineers (SPIE)* [DOI: 10.1117/1.OE.58.6.065103]

Keywords: optical system design; optics and lens design; thin films; deep learning; artificial intelligence; evolutionary optimization.

Paper 190284 received Mar. 11, 2019; accepted for publication Jun. 5, 2019; published online Jun. 22, 2019.

1 Introduction

The search for optimal design parameters is a class of inverse problems that is assuming an increasing importance in optics and photonics as advancements in fabrication techniques and materials research allow us to explore complex structure and material combinations. While advances in computational power have made forward simulations faster, inverse problems, especially those involving large degrees of freedom, remain challenging due to the so-called "curse of dimensionality,"[1] a term that refers to the exponential explosion of the search space volume with a linear increase in dimensionality. Optimal photonics design is thus either restricted to limited searches in global space or to gradient-based searches that tend to get stuck at local optima.[2] The current resurgence of neural computing in the form of deep learning (DL)[3] has raised the intriguing possibility of artificial intelligence-based methods that could potentially overcome the "curse of dimensionality." The application of DL has shown early promise in the design of optical thin-films,[4] nanostructures,[5–8] metasurfaces,[9–11] and integrated photonics.[12,13]

Optics design differs from pattern recognition problems (a space where DL has achieved remarkable success) in many ways: (1) performance is often very sensitive to variations in design parameters; (2) large datasets are difficult to generate although labeling of data is automatic; (3) performance requirements are often stringent and, hence, uncertainties in the model are not acceptable; and (4) a given response can be realized through multiple designs, while a single design has a unique response (nonuniqueness).

The first challenge in applying DL to optics design is that deep neural networks (DNNs) can be easily trained to predict a response given a design (forward DNN) but not vice-versa due to the nonuniqueness problem. Peurifoy et al.[14] achieve this inversion by freezing the weights of the trained forward DNN and performing a gradient descent search over the input parameters. Subsequent workers have resorted to using two DNNs[7,9,12] to overcome the speed limitation of this approach. The tandem network concept (Liu et al.[4]) and the generative networks with critic moderation approach (Cai et al.[5]) provide speed improvements but require the training of two separate networks, inadvertently constrict the design space and are require careful hyperparameter tuning.

A second challenge in applying DL to optics design concerns the model uncertainties involved in DNNs and its implication on the correctness and optimality of designs obtained; however, this issue has remained unaddressed. It is well known that DNNs tend to perform well on problems, where most of the input information can be discarded as "noise" and perform somewhat poorly on problems, which are sensitive to input parameter values.[15] Furthermore, DNNs do not adequately model their own prediction uncertainty.[16] The implication is that a trained DNN may perform quite well on the training data as well as on test data but may model some regions of the design space poorly. Additionally, the bounds on design parameters imposed when generating training data may cause the DNN to miss the optima entirely. Larger datasets and bigger models may alleviate these concerns to some extent, but larger datasets are costly to generate and larger DNN models require extensive hyperparameter optimization.

The specific contributions of this paper are as follows: (1) An evolutionary search technique in model space as a solution to the inversion problem that obviates the need for a second network and avoids unnecessary constriction of the design space and (2) a hybrid approach using DNN as a surrogate model[17,18] as a solution to the optimality problem. We consider the problem of multilayered thin-film design by DNN surrogate model-assisted differential evolution (DE).[19] The problem of broadband antireflection coating (ARC) design has received extensive attention from researchers[20–24] and a broad range of theoretical and computational techniques (including DE)[24–28] have been reported. For our purpose, we note that this is a challenging multimodal optimization problems with regions of flat fitness[20,24,29] with strong mathematical and computational evidence

*Address all correspondence to Ravi S. Hegde, E-mail: hegder@iitgn.ac.in

0091-3286/2019/$25.00 © 2019 SPIE

regarding the existence of global optima.[21–23] After this introduction, a detailed description of assisted DE and a brief description of the development of DNN appear in Sec. 2; the training and testing performance of the DNN models and performance of assisted DE algorithm is described in Sec. 3 before concluding the paper.

2 Assisted Differential Evolution

Figure 1 shows a detailed pictorial comparison of a regular/plain DE optimization process (a) with a DNN-assisted DE (b). A DE revolves a set of designs (individuals) that form a population. A 16-element vector of layer thicknesses denotes an individual in this paper and a collection of such designs would be the population. An iteration essentially generates a newer population by stochastically mixing the current population. This translates to getting newer designs by mixing the existing designs stochastically. The fitness of a design is a number that measures how closely its spectrum matches the targeted spectrum. At any iteration, we need to evaluate the fitness values of all individual designs in our population to designs to rank them. The fitness value of a design determines how much its characteristics are passed on to subsequent designs. Mutation and crossover are operations performed on the existing designs to obtain newer designs. Mutation involves changing the thickness of layers while crossover involves interchanging layers between two designs.

If we denote the j'th individual at the beginning of the i'th iteration as p^i_j, then the set of designs p^0_j, $0 \le j \le P-1$ is the initial population. Before beginning the optimizations, a set F^0_j, $0 \le j \le P-1$ of real numbers called the fitness vector is generated by calling the merit function on this initial population. The fitness vector is updated at the end of every iteration and at the end of the i'th iteration denoted as F^i_j.

Each iteration begins with the mutation (M) phase followed by a crossover (C) phase, which results in another set of P individuals called mutants (j'th mutant during the i'th iteration is denoted by m^i_j). Another set of real numbers called the mutant fitness vector is calculated at every iteration by calling the merit function on the set of mutants, which is denoted as f^i_j, $0 \le j \le P-1$. The process of generating the j'th mutant involves an intermediate individual generated by the mutation phase, as described by $o^i_j = a_j + r_{\mathrm{mut}} * (b_j - c_j)$ and $a_j, b_j, c_j \in \{p^{i-1}_0, p^{i-1}_1, \cdots, p^{i-1}_{P-1}\} - \{p^{i-1}_j\}$, where the a, b, c individuals are randomly drawn for each individual and r_{mut} is a hyperparameter of the optimization called the mutation radius. Finally, the j'th mutant is obtained using p^{i-1}_j and o^i_j in the crossover phase as follows:

$$m^i_j[d] = \begin{cases} o^i_j[d], & \text{if } p >= p_{\mathrm{cross}} \\ p^{i-1}_j[d] & \text{otherwise} \end{cases}, \quad 0 \le d \le 15. \tag{1}$$

Crossover is an element-wise operation on the individual (which is a 16-dimensional vector) and involves drawing a random number p for each of the elements. This randomly drawn number is compared with p_{cross} (another hyperparameter of the optimization called the crossover probability) to generate the mutant. The iteration ends with a repopulation (R) phase that results in the next generation of individuals and can be summarized as follows:

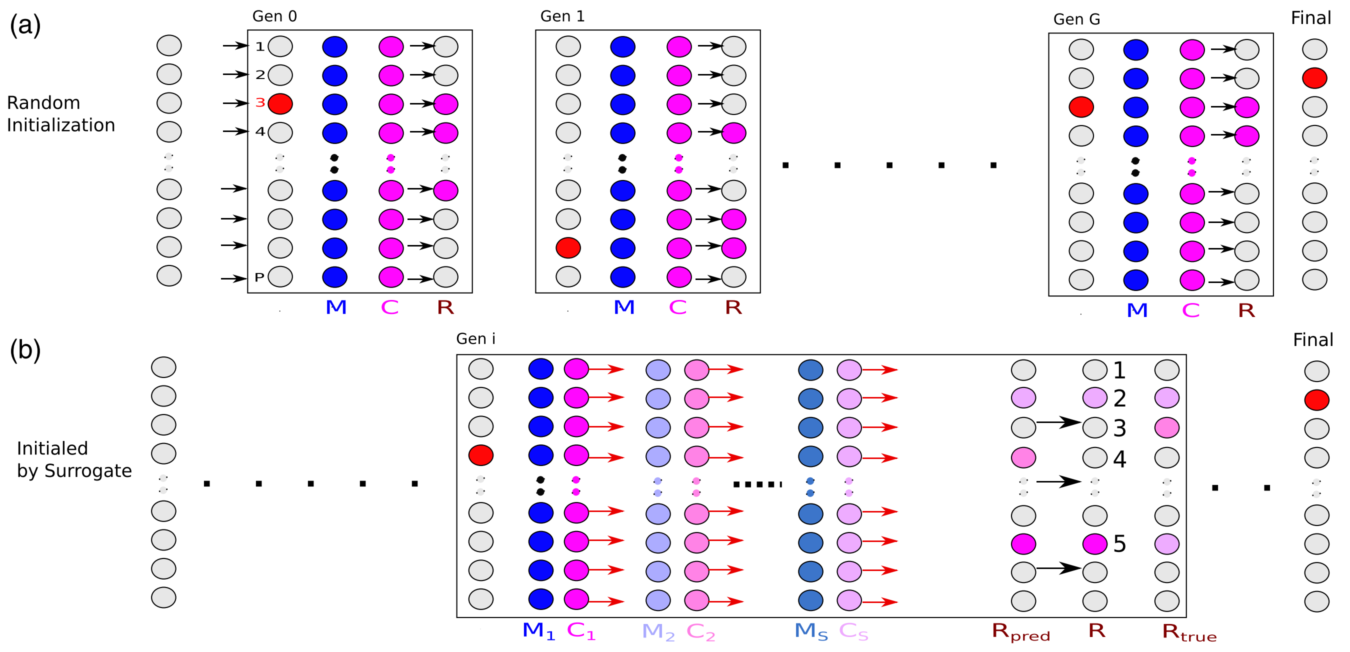


Fig. 1 Comparative overview of the DNN-assisted DE optimization (a) unassisted and (b) assisted. The individual, a multilayered thin-film design, is denoted by filled circles. In each generation, a population of individuals is created by mutation (M) and crossover (C) and repopulation (R) phases. The merit evaluations can be performed exactly (black arrows) or approximately via the DNN surrogate (red arrows). Red circles denote the best individual of the population at the start of the iteration. Blue and purple [and shades of these colors in (b)] represent mutated and crossover variants. In panel (b), the multiple rounds of mutation and crossover result in an intermediate population R_{pred}. Some exact function calls are used to verify and replace the population into R which could be different from the correct case R_{true}. Numbers 1 to 5 show the various possibilities resulting from approximating the repopulation phase.

$$\begin{cases} p_j^i = m_j^i, F_j^i = f_j^i & \text{if } f_j^i > F_j^{i-1} \\ p_j^i = p_j^{i-1}, F_j^i = F_j^{i-1} & \text{otherwise} \end{cases}, \quad 0 \le j \le P-1, \tag{2}$$

where the updating of the fitness vector is also shown. Note that it is not necessary to store the fitness vectors at each iteration. For N iterations, this results in $(N+1) * P$ function calls.

Next, consider the assisted version shown in Fig. 1(b). The first difference between assisted and unassisted versions is that the initial population is generated randomly for the unassisted version. The DNN surrogate is used to generate the initial population in the assisted version. using the graphical processing unit (GPU) version of the DE algorithm. DE is run simultaneously on multiple islands of randomly initialized populations, and the best individuals in all the islands are pooled and sent to the CPU, where the exact fitness of these individuals is calculated. If their fitness is below a certain threshold, they are kept aside or else they are discarded. The entire procedure is repeated until a predetermined number have been found.

The second major difference is in the repopulation phase, where the exact computation of the mutant fitness vector f_j^i for all the P members is replaced by exact computation only on a subset of the mutants. To determine the subset for which exact computation is necessary, two auxiliary fitness vectors g (analogous to f) and G (analogous to F) are calculated using the surrogate. The modified repopulation phase that results in the next generation of individuals can be summarized as follows:

$$\begin{cases} \begin{cases} p_j^i = m_j^i, F_j^i = f_j^i & \text{if } f_j^i > F_j^{i-1} \\ p_j^i = p_j^{i-1}, F_j^i = F_j^{i-1} & \text{otherwise} \end{cases} & \text{if } g_j^i > G_j^{i-1} \\ p_j^i = p_j^{i-1}, F_j^i = F_j^{i-1} & \text{otherwise} \end{cases}, \quad 0 \le j \le P-1, \tag{3}$$

where the updating of the fitness vector is also shown. The last difference is that several mutations and crossover phases can occur in parallel in the same iteration. The final mutant population in a particular iteration can then be selected by comparing the various intermediate fitness vectors.

There are several potential consequences of this approximation phase and all five of these possibilities are noted in Fig. 1(b). The first four can be summarized as: (1) $g_j^i < G_j^{i-1}$, $f_j^i < F_j^{i-1}$: one function call is correctly saved; (2) $g_j^i > G_j^{i-1}$, $f_j^i > F_j^{i-1}$: optimization has advanced correctly; (3) $g_j^i < G_j^{i-1}$, $f_j^i > F_j^{i-1}$: optimization has been impeded, no unnecessary function calls; and (4) $g_j^i > G_j^{i-1}$, $f_j^i < F_j^{i-1}$: optimization not impeded, but unnecessary function call made. The final possibility when one mutant is wrongly picked to be better than the one that is truly better only occurs when multiple rounds of mutation and crossover happen in a single iteration. A good surrogate can be identified as the one, where cases 1 and 2 occur more frequently than cases 3 and 4.

2.1 *Development of the DNN*

We have extensively experimented with the number of neurons in each layer, different activation functions, batch normalization and dropout layers, replacement of fully connected layers with convolutional layers, and various training methodologies [stochastic gradient descent (SGD), ADAM, ADAM method with Nesterov momentum (N-ADAM), etc].[30] Surprisingly, the number of trainable parameters was found to correlate most strongly with accuracy. and most other factors had only a minor influence. Rectified linear unit and Swish[31] activations performed nearly equally well and were substantially better than other activations. Batch normalization and dropout were not found to be beneficial. ADAM and N-ADAM training procedures were found to significantly outperform SGD. Preliminary experiments with convolutional layers showed some promise. We could reduce the number of trainable parameters with minor degradation in accuracy; however, convolutional layers took more time to train.

The final choice of the model was a feedforward neural network with fully connected layers, as shown in Fig. 2(a). All the hidden layers had the same number of neurons. Swish activation[31] was used on all but the last layer where sigmoid activation was used to bound the output between 0 and 1 [see Fig. 2(b)]. The structure is an air-clad 16-layered thin-film made of alternating layers of silica[32] and titania[33] on a semi-infinite substrate of refractive index 1.52. The maximum and minimum thickness of each layer are $d_{\min}$ and $d_{\max}$, respectively, and $d_{\min} = 0.1 d_{\max}$. Any possible geometry can then be denoted by the array of thickness values $X = [x_1, x_2, x_3, \cdots, x_{16}]$, which is the input layer. The reflectance spectrum at normal incidence of any design X at 64 points equally spaced in frequency between frequencies corresponding to a minimum and maximum wavelengths of 400 and 800 nm, respectively, is denoted by an array $Y = [y_1, y_2, y_3 \cdots y_{64}]$, which is the output. Figure 2(c) shows two possible ways we have used to generate datasets: (1) Latin hypercube (LH) sampling (this ensures maximum separation between the training data points) and (2) LH sampling on a larger hypercube followed by clipping.

3 Results and Discussion

The open-source python program transfer matrix method (TMM) provided by Byrnes[34] (with some modifications that allow it to calculate the spectrum in parallel) is used to calculate the exact spectrum. DNNs are implemented with the Keras library[30] running on MXnet backend.[35] The DE component was written by us from scratch with separate CPU and GPU versions and we ensured that unnecessary data transfers between the CPU and the GPU are avoided enabling very rapid evaluations of a large number of designs using the surrogate for any conceivable target spectrum. The source code for the implementation, datasets and saved models have been made available.[36] The following results are obtained by running our implementation on a workstation

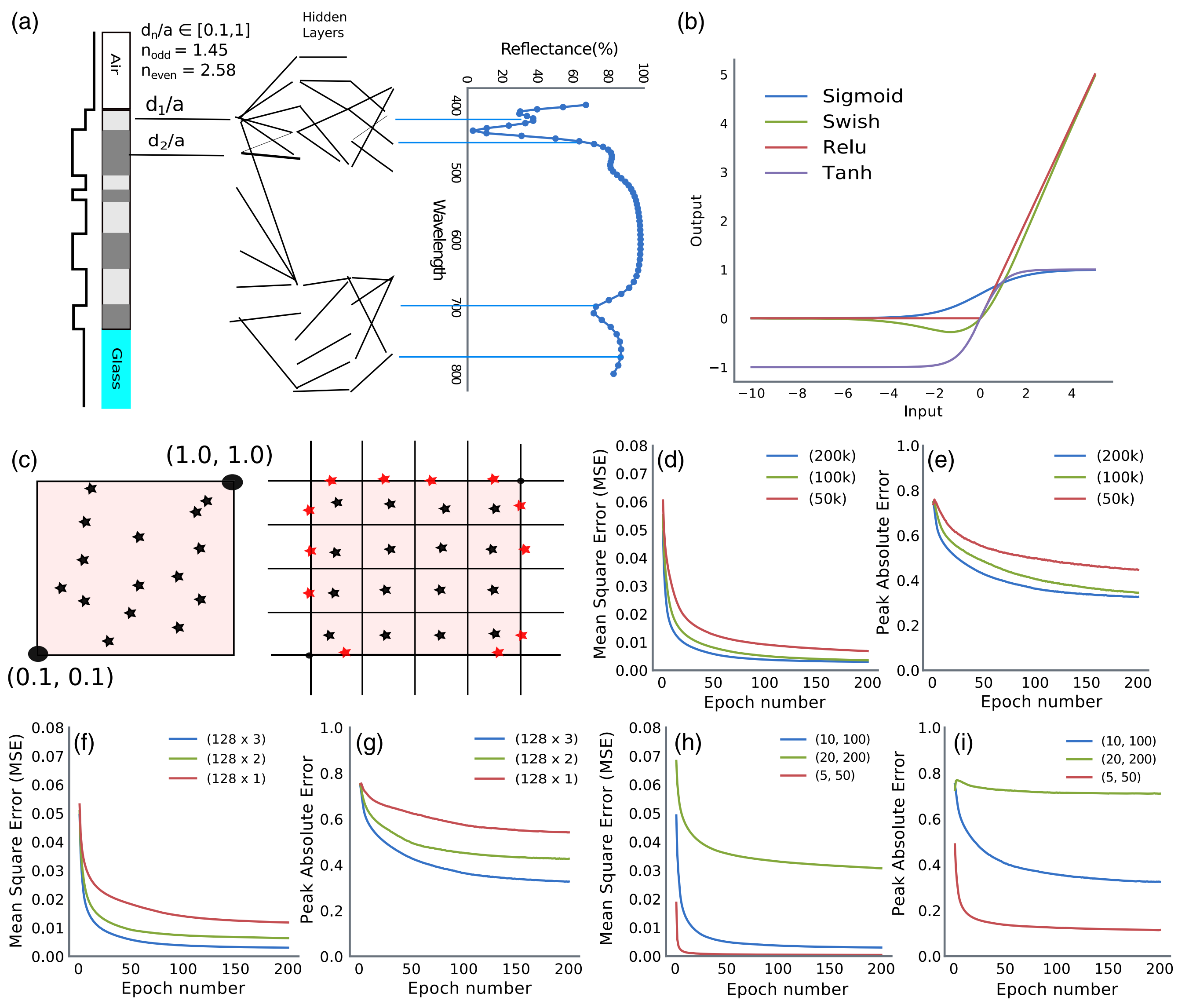


Fig. 2 DNN surrogate. (a) Overview of the DNN architecture. The neuron color is chosen to denote the activation function (black for linear, green for G "Swish" and blue for Sigmoid). (b) Various possible neuron activation functions. (c) Training data sampling technique illustrated on a 2-D hypercube. (d–i) denote the training error progression over 200 epochs [(d), (f), (h) show MSE while, e.g., (i) show the peak error] for various combinations of network architecture, training dataset size and layer size range. (d,e) Identical network (128 neurons in three hidden layers) and size range (10 nm, 100 nm), varying training dataset size 50k, 100k and 200k. (f, g) Varying network (128 neurons in one, two, and three hidden layers), identical size range (10 nm, 100 nm) and training dataset size of 200k. (h, i) Identical network (128 neurons in three hidden layers) and training dataset size of 200k, but varying size ranges (5 nm, 50 nm), (10 nm, 100 nm), and (20 nm, 200 nm).

with an Intel™ i9–7920X CPU with 3 NVIDIA™ GeForce GTX 1080 GPU cards with 12-GB memory.

3.1 *DNN Training and Testing*

This subsection examines the influence of system, model, and data on the accuracy of the trained DNN. and the main findings are summarized in Figs. 2(d)–2(f) and 3. Traditionally, the mean squared error (MSE)[4] and the mean relative error[14] have been used to quantify accuracy. We found that MSE is a good loss function to train the network but peak absolute error is a better indicator of the generalization ability of the trained DNN. Two observations are noted in Figs. 2(d)–2(f). First, reducing the MSE also reduced the peak error. Second, we notice two regimes in the error progression: a steep drop in error is followed by a gradual error loss. This is a universal feature that has been reported in several of the earlier papers.

For the experiment in Figs. 2(d) and 2(e), a network characterized by three hidden layers with 128 neurons (represented as 128×3) is trained on datasets with fixed minimum and maximum layer dimensions of 10 and 100 nm. Only the number of size of the dataset is varied (note 50k stands for 50,000 samples). It can be seen that 100k is about the optimal size and that further increase in size is only giving diminishing returns. For the experiment in Figs. 2(f) and 2(g), the dataset size is fixed at 200k, dimension range is fixed at (10, 100) and the number of hidden layers is varied

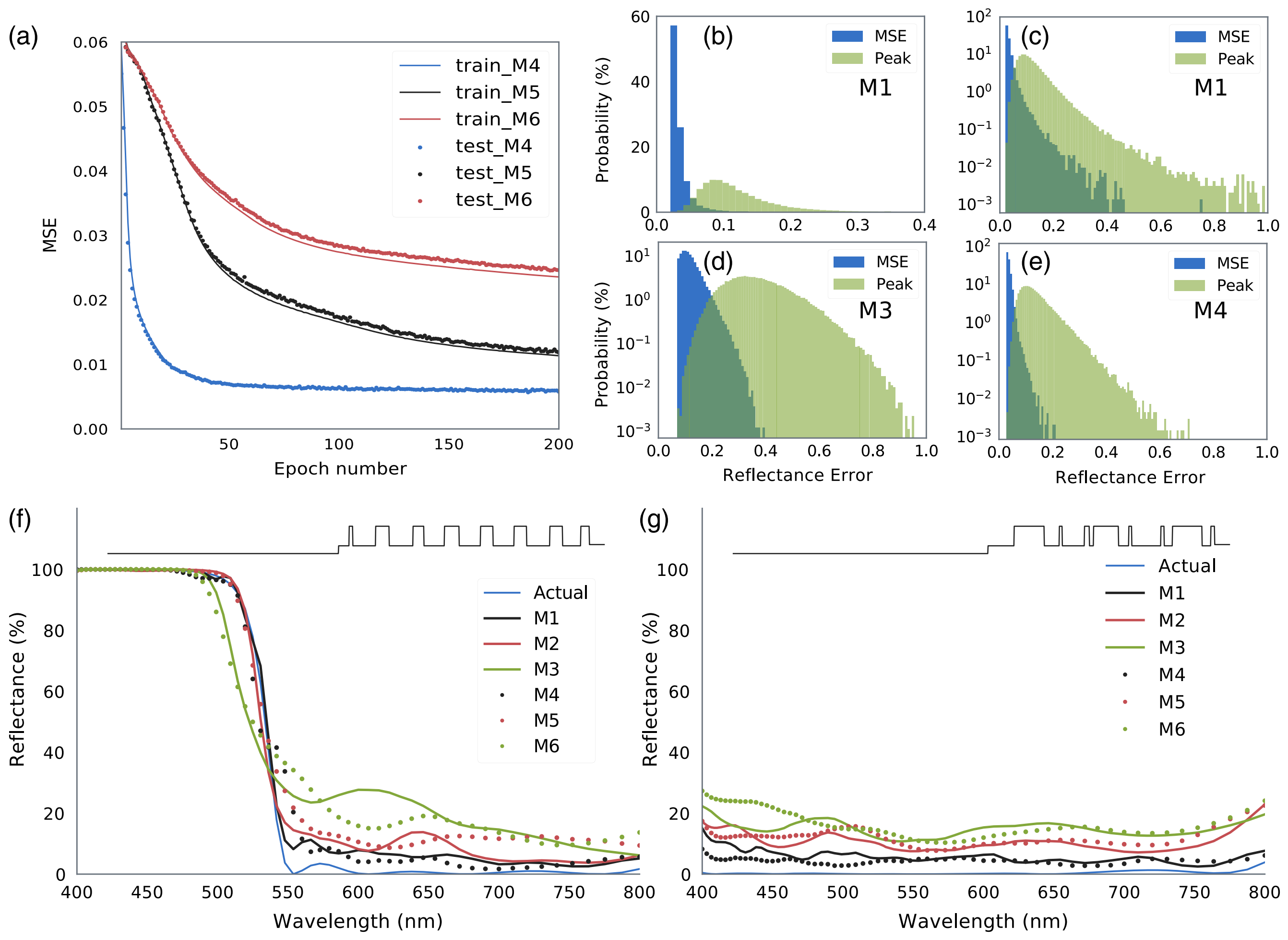


Fig. 3 DNN performance on the same unseen test data (size 100,000) for the six models shown in Table 1. (a) The progression with training epochs of the training and the test error for M4, M5, and M6. The curves for M1 were nearly identical to that of M4, M2 to that of M5, and M3 to that of M6. (b–e) Histograms of the mean and peak absolute errors for models M1, M1, M3, and M4, respectively. (c–e) The percentage probability on a logarithmic scale. (f) and (g) compare the spectrum predictions of all the models on the same geometry against the exact spectrum. (f) A longpass filter design and (g) an ARC design. The insets in (f, g) denote the refractive index profile of the designs.

in the DNN. Once again, it is seen that for a fixed dataset size, simply increasing the model complexity leads to diminishing returns. Finally, the experiment in Figs. 2(h) and 2(i) shows that expanding the constraints imposed on design parameters increases the Vapnik–Chervonenkis dimension of the problem and it becomes increasingly difficult to train a DNN. The summary observation is that extensive experimentation is needed to identify the correct model size and training dataset sizes.

For the remaining set of experiments, we consider six possible models listed in Table 1. Modern DL platforms can handle very large networks but larger dataset sizes continue to be

Table 1 Model summary. Train size large corresponds to 400k and small to 40k. Model size large corresponds to 128×3 and small to 128×1. LH + BW stands for Latin hypercube sampling with clipping [see Fig. 2(c)]. The training phased used three GPUs in parallel.

Name	Train size	Model size	Dataset bias	Train MSE	Peak error	Train time (s)
M1	Large	Large	Uniform	0.0012	0.23	28
M2	Small	Large	Uniform	0.0099	0.46	19
M3	Small	Small	Uniform	0.0183	0.63	15
M4	Large	Large	LH + BW	0.0013	0.25	28
M5	Small	Large	LH + BW	0.0131	0.50	19
M6	Small	Small	LH + BW	0.0240	0.66	15

harder to obtain for many problems. Thus, we do not consider the combinations with small models and large datasets. These possibilities coarsely represent the cost/benefit trade-off involved in DNN development. Note that the generation of 400k sized dataset takes nearly 2 h on our system in comparison to about 12 min for the 40k sized dataset. The models were all trained on datasets sampled in the (10 nm, 100 nm) range for each layer thickness.

The six models were tested on a separately generated dataset of size 100k and Fig. 3 summarizes their testing performance. The test and the train errors (MSE) are seen to track well in all cases, as shown in Fig. 3(a), with a hint of divergence seen for M5 and M6. This shows that a small amount of overfitting is expected when dataset sizes are smaller than ideal. With L2 regularization of about 1×10^{-5} added, we observed that this overfitting could be reduced slightly without unduly affecting convergence or training time. Figures 3(b)–3(e) are histogram plots showing the distribution of the mean and the peak absolute errors for each of the 100k test data points. It is seen that MSE is a more optimistic picture than the peak error; MSE is narrowly distributed while the peak error shows a flatter distribution with very large values possible. A surprising result is seen when we compare Figs. 3(c) and 3(d). Although the MSE for model M1 is far better than that of M3, both are seen to have nearly the same number of data points, where their prediction leads to a large peak error. Figure 3(e) shows that model M4 shows a much reduced rate of high peak errors in comparison to M1 which occurs because the training dataset and testing dataset were sampled on the same distribution unlike that of M1. Figure 3(f) shows the predicted spectrum of all six models on the same design for a longpass filter. This design is located in the interior and hence M1 is favored over M4. Figure 3(g) shows the predicted spectrum of all six models on the same design for a reflection cancellation filter. This design is located on the boundary and, hence, M4 is favored over M1.

3.2 *Performance of Assisted DE*

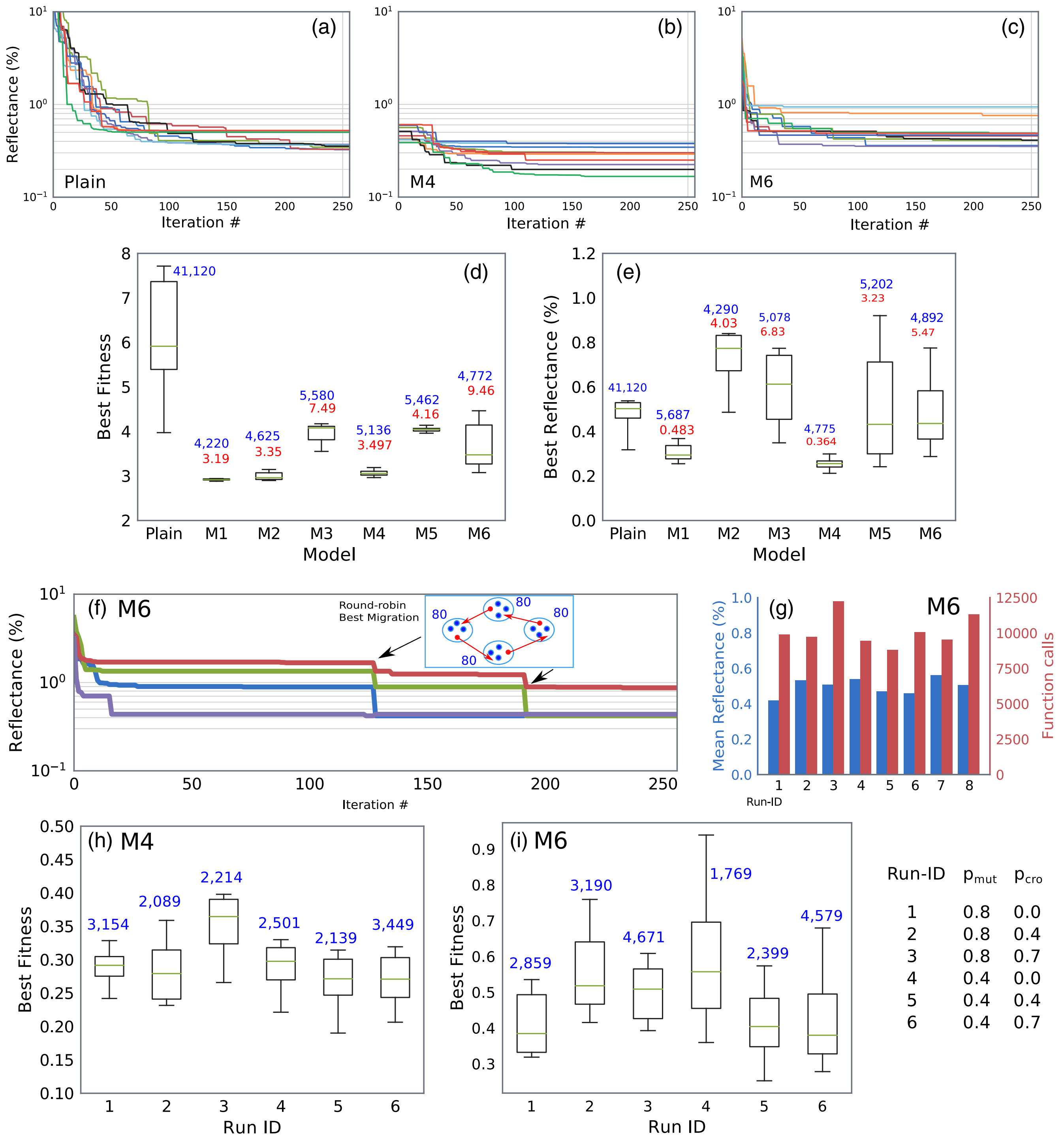
In this section, we compare the performance of assisted DE using the six different models described above with the plain DE version. The results are judged on three metrics: (1) how many exact function calls are used; (2) the best fitness achieved; and (3) the variance in the best fitness. DE is a stochastic algorithm and, hence, 10 runs are considered to show the range of possibilities. Figures 4(a)—4(c) and 4(f) show the general progression of the fitness with iteration number for DE (plain and assisted versions) and Figs. 4(d), 4(e), 4(h), and 4(i) use boxplots to show the best fitness value achieved at the end of 256 iterations.

We considered two different objectives: (1) an ARC with minimal mean reflectance over the entire band from 400 to 800 nm [see Fig. 3(g)] and (2) a longpass filter with the edge at 532 nm [see Fig. 3(f)]. Only Fig. 4(d) concerns the longpass design and the rest of the figures deal with the ARC design. The mean reflectance (in %) is the fitness measure for ARC design; for the longpass design, the mean of the absolute difference in the target and the actual spectrum (expressed in %) is the fitness. In comparing the achieved reflectances, we note that reports in the literature have established that for normal incidence, two material systems provide the best designs for ARC. The optimal reflectance achievable is known to depend on the highest and the lowest refractive indices available, the optical thickness as well as the number of layers.[20–23] For the silica/titania system, it has been reported that the best achievable mean reflectance may be expressed as $R_m = 0.1029 \times d_{\mathrm{opt}}^{-0.9758}$, where R_m is expressed in percentage and d_{opt} is the optical thickness expressed in microns. In our case, we consider designs, where the layer dimension is in the range (10 nm, 100 nm) and, thus, we can conclude that the exact optima must lie in the 0.1% to 0.2% range.

We note that for the evaluation of the spectrum of 100,000 geometries, the DNN (M1) took 479 ms. The exact calculation using the TMM python package with python-based multiprocessing library parallelism using 16 cores for 100,000 geometries took 6 min and 19 s and about 1 h 5 min for a single-threaded version. For generating the initializations, the GPU version of the DE algorithm employed 300 islands each with a population P of 40 and ran for 256 iterations with a mutation radius r_{mut} of 0.8 and crossover probability p_{cro} of 0.7. We have used only a single GPU for doing this procedure as it was found to be extremely fast to execute finishing in nearly a minute including the CPU verification phase. Red over text in Figs. 3(d) and 3(e) shows the fitness of the initial populations. It is seen that models M1 and M4 generate the best starting populations but still not close enough to the global optims. From previously published reports,[20–23] it is apparent that optimal ARC designs are lying at the boundary of the design space. As M4 is trained on a boundary weighted dataset, it performs best in initializations in comparison to M1. This trend is reversed in the case of a longpass design, where M1 performs better. This shows that DNNs are vulnerable to the dataset construction even for large-sized datasets and are not guaranteed to provide optimal designs.

Figures 4(a)–4(c) show that the assisted DE performs well in comparison to the plain DE. At first glance, this appears to be due to the initializations, as shown in Fig. 4(b), which uses the M4 model. But, even an inferior model, the M6, seems to do nearly as well as (a) with a larger variance. Figures 4(a)–4(e) also highlight the important observation that the assisted DE performs better than plain DE as well as DNN acting alone. This observation is true even for inferior models although they tend to have larger variance. The assisted DE is seen to result in a drastic reduction in the number of exact function calls, as shown in Figs. 4(d)–4(e), where the number of function calls is noted in blue over text. Nearly, all the versions resulted in only 10% exact function calls and ran five to six times faster. On our computer, the assisted versions ran in about 10 min.

In Figs. 4(f) and 4(g), we show a technique to deal with the large variance seen in assisted DE with coarser models. Unlike the previous runs, where the population count was 160, we reduced the population count to 80 and ran DE on four islands simultaneously. At the 128th and 192nd iteration mark, the best individuals in the islands were transmigrated in a round-robin fashion among these four islands. This procedure was repeated eight different times and the best fitness achieved each time and the number of exact function calls are noted in Fig. 4(g). Although using only 20% of the function calls of the plain version and an inferior DNN model, we see that we achieve comparable performance to it. Mutation radius and crossover probability are the hyperparameters of the assisted



Run-ID	p_{mut}	p_{cro}
1	0.8	0.0
2	0.8	0.4
3	0.8	0.7
4	0.4	0.0
5	0.4	0.4
6	0.4	0.7

Fig. 4 DNN-assisted DE performance. (a)–(c) The evolution of the best fitness as a function of DE iteration number for 10 separate runs. (a) The plain DE while (b) and (c) DE assisted by DNN models M3 and M6, respectively (see Table 1). (d) and (e) Boxplots of final best fitness values for 10 separate runs of DE (plain and assisted versions) after 256 iterations. In panels (d) and (e), blue text over boxes represent the number of function calls made and red text represents the best fitness of the initializing population. (f) Fitness evolution of a four population of 80 individuals with two instances of best individual round-robin migration using assisted DE with M6 (inset is a schematic representation of this operation). (g) Summary of 8 runs of the DE, as shown in (f). (h) and (i) A study using the M6 model with 80 members and with variation in the hyperparameters of the DE algorithm.

DE. In Figs. 4(h) and 4(i), we studied how the choice of these parameters influences the performance. It is seen that a mutation radius of 0.4 and a crossover probability of 0.4 work best for both models.

We finally evaluate the performance of our proposed method against the needle-point synthesis method provided by the freely available software OpenFilters.[37] The needle method implementation provided by the OpenFilters

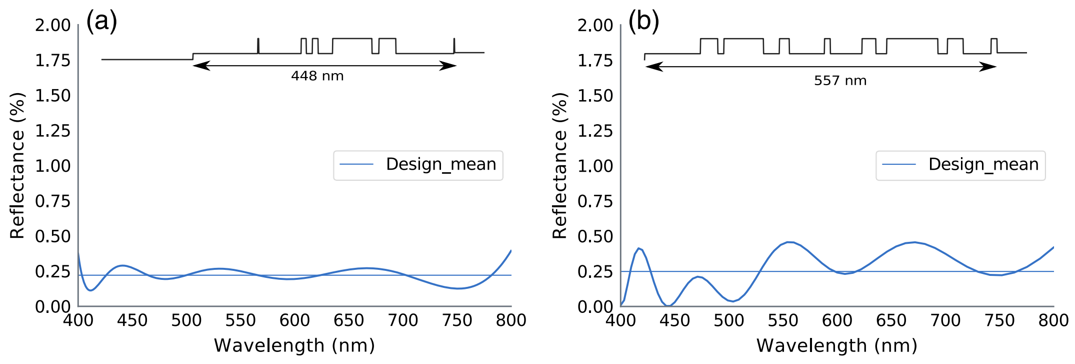


Fig. 5 The reflectance spectra and the index profiles of the best designs obtained using (a) OpenFilters needle point method and (b) our method.

program does not provide an option to limit the range of thickness values each layer can take but permits to eliminate layers below a specified thickness and research for optima. However, we could not find designs when the minimum thickness was restricted to 10 nm. Our method, however, had no problems in finding solutions, where the range of the thickness is from 10 to 100 nm. The best reflectance achieved by the methods along with the index profile of the designs is shown in Fig. 5. Our method was able to achieve $R_{\mathrm{avg}} \approx 0.0025$ and $R_{\mathrm{max}} = 0.0045$, which is comparable to that achievable by the needle method. The thickness of our design is 557 nm, which is slightly thicker than that obtained by the needle method. However, we note that the allowed thickness range is broader for the needle method and the obtained design has two very thin layers (≤ 2 nm thickness). We note that Yang and Kao[26] have previously reported the comparison between evolutionary algorithms and the needle method and found that although evolutionary approaches provide similar solutions like that of the needle method but were noticeably slower than these. Our algorithm is coded using the scripting language and, thus, it is unfair to compare its speed with that of a highly optimized code. Nonetheless, it took 6 min for our method as opposed nearly 2 min for the needle method (note that the needle method requires us to try out different initial conditions to succeed). It is not surprising that a hand-tuned method outperforms a generic black-box optimization method, but it is remarkable that a black-box method very closely approaches the performance of a hand-tuned method.

4 Conclusion

In this contribution, we explored the role of DL in accelerating photonics design optimizations. In our approach, training is done on one forward network (design → spectrum) and a fast global search of the DNN using a DE approach suffices to arrive at a working design given a target spectrum. It highlights the pitfalls in relying entirely on DNNs to arrive at designs as the optimality of these solutions is not certain even after meticulous trial and error to arrive at a good model. It proposes an alternate technique, where DNNs are only used for preselection to speed up conventional evolutionary optimization techniques. This alternate technique was found to result in significant reductions in computational burden even with inferior DNN models. We have explored low to medium dimensionality here and future work needs to consider whether an extension to 2-D and 3-D problems will be just as successful. Preliminary work has shown that further performance improvements are possible using surrogate-based mutation and crossover phases and by dynamically tuning the hyperparameters of the DE algorithm.

References

1. R. E. Bellman, *Dynamic Programming*, Dover Publications Inc., New York (2003).
2. J. Langhabel, J. Wolff, and R. Holca-Lamarre, "Learning to optimise: using Bayesian deep learning for transfer learning in optimisation," in *Workshop Bayesian Deep Learn., NIPS*, Barcelona, pp. 1–3 (2016).
3. Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature* **521**, 436–444 (2015).
4. D. Liu et al., "Training deep neural networks for the inverse design of nanophotonic structures," *ACS Photonics* **5**, 1365–1369 (2018).
5. Z. Liu et al., "Generative model for the inverse design of metasurfaces," *Nano Lett.* **18**, 6570–6576 (2018).
6. T. Zhang et al., "Spectrum prediction and inverse design for plasmonic waveguide system based on artificial neural networks," arXiv:1805, pp. 1–7 (2018).
7. I. Malkiel et al., "Plasmonic nanostructure design and characterization via deep learning," *Light Sci. Appl.* **7**, 60 (2018).
8. P. R. Wiecha, N. Mallet, and G. Larrieu, "Pushing the limits of optical information storage using deep learning," *Nat. Nanotechnol.* **14**, 237–244 (2019).
9. K. Yao, R. Unni, and Y. Zheng, "Intelligent nanophotonics: merging photonics and artificial intelligence at the nanoscale," *Nanophotonics* **8**(3), 339–366 (2019).
10. W. Ma, F. Cheng, and Y. Liu, "Deep-learning-enabled on-demand design of chiral metamaterials," *ACS Nano* **12**, 6326–6334 (2018).
11. S. Inampudi and H. Mosallaei, "Neural network based design of metagratings," *Appl. Phys. Lett.* **112**, 241102 (2018).
12. M. H. Tahersima et al., "Deep neural network inverse design of integrated nanophotonic devices," arXiv 1809.0355, pp. 1–8 (2018).
13. A. Gandhi and C. E. Png, "Modal classification in optical waveguides using deep learning," *J. Mod. Opt.* **66**(5), 557–561 (2019).
14. J. Peurifoy et al., "Nanophotonic particle simulation and inverse design using artificial neural networks," *Sci. Adv.* **4**(6), eaar4206 (2018).
15. R. Schwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," arXiv 1703.00810, pp. 1–19 (2017).
16. Y. Gal, "Uncertainty in deep learning," PhD Thesis, University of Cambridge (2016).
17. T. Peter, "Using deep learning as a surrogate model in multi-objective evolutionary algorithms," PhD Thesis, Otto-von-Guericke-Universität, Magdeburg (2018).
18. Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Comput.* **9**, 3–12 (2005).
19. R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.* **11**, 341–359 (1997).
20. U. B. Schallenberg, "Antireflection design concepts with equivalent layers," *Appl. Opt.* **45**(7), 1507–1514 (2006).
21. A. V. Tikhonravov and J. A. Dobrowolski, "Quasi-optimal synthesis for antireflection coatings: a new method," *Appl. Opt.* **32**(22), 4265–4275 (1993).
22. J. A. Dobrowolski et al., "Optimal single-band normal-incidence antireflection coatings," *Appl. Opt.* **35**(4), 644–658 (1996).
23. A. V. Tikhonravov, "Some theoretical aspects of thin-film optics and their applications," *Appl. Opt.* **32**(28), 5417–5426 (1993).
24. S. W. Anzengruber et al., "Numerical methods for the design of gradient-index optical coatings," *Appl. Opt.* **51**(34), 8277–8295 (2012).

25. Y. Zhao et al., "Optimal design of graded refractive index profile for broadband omnidirectional antireflection coatings using genetic programming," *Prog. Electromagn. Res.* **145**, 39–48 (2014).
26. J.-M. Yang and C.-Y. Kao, "Efficient evolutionary algorithm for the thin-film synthesis of inhomogeneous optical coatings," *Appl. Opt.* **40**(19), 3256–3267 (2001).
27. M. Ebrahimi and M. Ghasemi, "Design and optimization of thin film polarizer at the wavelength of 1540 nm using differential evolution algorithm," *Opt. Quantum Electron.* **50**, 192 (2018).
28. V. Janicki, J. Sancho-Parramon, and H. Zorc, "Refractive index profile modelling of dielectric inhomogeneous coatings using effective medium theories," *Thin Solid Films* **516**, 3368–3373 (2008).
29. H. Becker et al., "Design and realization of advanced multi-index systems," *Appl. Opt.* **53**(4), A88–A95 (2014).
30. F. Chollet, "Keras: the python deep learning library," *Github Repository*, https://github.com/fchollet/keras (2015).
31. P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," arXiv 1710.0594, pp. 1–13 (2017).
32. I. H. Malitson, "Interspecimen comparison of the refractive index of fused silica," *J. Opt. Soc. Am.* **55**(10), 1205–1209 (1965).
33. T. Siefke et al., "Materials pushing the application limits of wire grid polarizers further into the deep ultraviolet spectral range," *Adv. Opt. Mater.* **4**, 1780–1786 (2016).
34. S. J. Byrnes, "Multilayer optical calculations," arXiv 1603.02720, pp. 1–20 (2018).
35. T. Chen et al., "MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems," arXiv 1512.01274, pp. 1–6 (2015).
36. R. S. Hegde, "DeepMie—deep learning electromagnetic scattering," *Bitbucket Repository*, https://bitbucket.org/rshegde/deepmie-deep-learning-electromagnetic-scattering/overview (2018).
37. S. Larouche and L. Martinu, "OpenFilters: open-source software for the design, optimization, and synthesis of optical filters," *Appl. Opt.* **47**(13), C219–C230 (2008).

Ravi S. Hegde has been an assistant professor in the Department of Electrical Engineering at the Indian Institute of Technology, Gandhinagar, since 2015. He was a research scientist with the electronics and photonics division at the A*STAR Institute of high-performance computing in Singapore since 2009. He currently works on analytical and numerical modeling of nanoscale optical structures and devices and their application toward energy harvesting, sensing and, imaging. He was awarded the doctor of philosophy in electrical engineering by the University of Michigan at Ann Arbor, USA, and the master of science in electrical engineering (specialization in photonics technology) by the University of Southern California, USA. He earned the bachelor of engineering in electrical engineering from the National Institute of Technology, Karnataka India.