Machine Learning Regression Approach to the Nanophotonic Waveguide Analyses

Sunny Chugh*, Souvik Ghosh, Aamir Gulistan and B. M. A. Rahman, *Life Fellow, IEEE*

***Abstract*—Machine learning is an application of artificial intelligence that focuses on the development of computer algorithms which learn automatically by extracting patterns from the data provided. Machine learning techniques can be efficiently used for a problem with a large number of parameters to be optimized and also where it is infeasible to develop an algorithm of specific instructions for performing the task. Here, we combine the finite element simulations and machine learning techniques for the prediction of mode effective indices, power confinement and coupling length of different integrated photonics devices. Initially, we prepare a dataset using COMSOL Multiphysics and then this data is used for training while optimizing various parameters of the machine learning model. Waveguide width, height, operating wavelength, and other device dimensions are varied to record different modal solution parameters. A detailed study has been carried out for a slot waveguide structure to evaluate different machine learning model parameters including number of layers, number of nodes, choice of activation functions, and others. After training, this model is used to predict the outputs for new input device specifications. This method predicts the output for different device parameters faster than direct numerical simulation techniques. Absolute percentage error of less than 5% in predicting an output has been obtained for slot, strip and directional waveguide coupler designs. This study pave the step towards using machine learning based optimization techniques for integrated silicon photonics devices.**

***Index Terms*—Machine learning, neural networks, regression, multilayer perceptron, silicon photonics.**

I. Introduction

MACHINE learning (ML) technology is being extensively used in many aspects of modern society: web searches, social networking, smartphones, bioinformatics, robotics, chatbots, and self-driving cars [1]. ML techniques are used to classify or detect objects in images, speech to text conversion, pattern recognition, natural language processing, sentiment analysis and recommendations of products/movies for users based on their search preferences. ML algorithms can be trained to perform exceptionally well when it is difficult to analyze the underlying physics and mathematics of the problem [2]. ML algorithms extract patterns from the raw data provided during the training without being explicitly programmed. The learned patterns can be used to make predictions on some other data of interest. ML systems can be trained more efficiently when massive amount of data is present [3], [4].

Recently, research on the application of ML techniques for optical communication systems and nanophotonic devices is gaining popularity. Several developments in ML over the past few years has motivated the researchers to explore its potential in the field of photonics, including multimode fibers [5], power splitter [6], plasmonics [7], grating coupler [8], photonic crystals [9], [10], metamaterials [11], photonic modes fields distribution [12], label-free cell classification [13], molecular biosensing [14], optical communications [15], [16] and networking [17], [18].

Complex nanophotonic structures are being designed and fabricated to enable novel applications in optics and integrated photonics. Such nanostructures comprise of a large number of parameters which needs to be optimized for efficient performance of the device and can be computationally expensive. For example, finite-difference time-domain (FDTD) method may require several minutes to hours to analyze the optical transmission response of a single photonic device depending on its design. ML approach offers a path for quick estimation of the optimized parameters for the design of complex nanostructures, which are critical for many sensing and integrated optics applications.

ML algorithm considers general function approximations to learn a complex mapping from the input to the output space. The most popular ML frameworks for building and training neural networks includes SciPy [19], Scikit-learn [20], Caffe [21], Keras [22], TensorFlow [23] and PyTorch [24]. PyTorch makes use of tensors for training neural networks along with strong GPU acceleration. It provides separate modules to build a neural network and automatically calculates gradients for backpropagation [25] that are required during the training of a neural network. PyTorch appears to be more flexible with Python and NumPy/SciPy stack compared to TensorFlow and other frameworks, which allows easy usage of popular libraries and packages to write neural network layers in Python. Scikit-learn is another simple and efficient ML library used for data mining and data analysis. In our implementation, we use PyTorch and Scikit-learn numerical computing environment to handle the front-end modeling and COMSOL Multiphysics for the back-end data acquisition.

Modeled waveguide designs considered in this paper are shown in Section II. Main concepts of ML related to integrated photonics applications are discussed in Section III. In Section IV, results from ML algorithms using PyTorch and Scikit-learn with FEM results for commonly used silicon photonic waveguides and devices are compared, and finally the paper is concluded in Section V.

II. Modeled Waveguides

In this work, we have considered three waveguide de-

*Corresponding author: sunny.chugh@city.ac.uk
Sunny Chugh, Souvik Ghosh, Aamir Gulistan and B. M. A. Rahman are with the School of Mathematics, Computer Science & Engineering, City, University of London, London, EC1V 0HB, U.K.

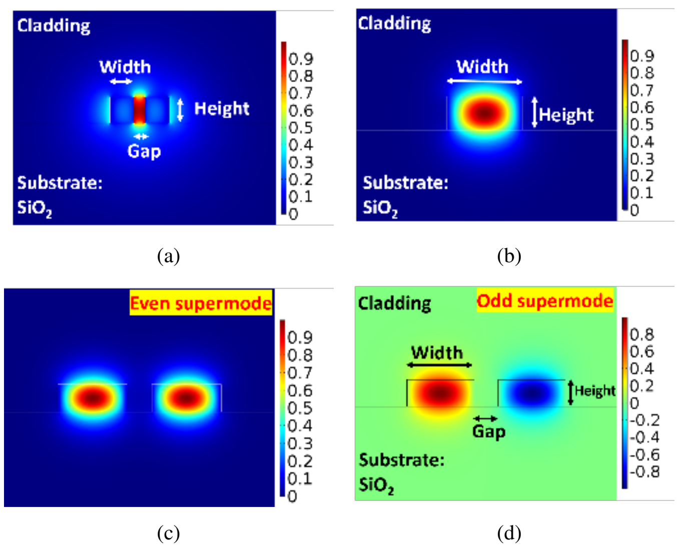


Fig. 1: An example of a (a) Slot waveguide showing E_x field profile, (b) Strip waveguide showing H_y field profile, and Directional coupler showing H_y field profile for (c) even supermode, and (d) odd supermode.

signs: slot waveguide, strip waveguide, and directional coupler. Cross-sectional view of these waveguides, along with their respective field profiles are shown in Fig. 1. A range of slot waveguides (Fig. 1a) and directional couplers (Figs. 1c and 1d) are simulated by changing the width, height, and gap between the silicon waveguides as the input parameters. Effective index and power confinement values are recorded for slot waveguides, while coupling length was obtained for directional coupler, corresponding to above mentioned input parameters. For strip waveguide (Fig. 1b), width, height of waveguides and wavelength are taken as input variables, while effective index is considered as the output variable. The commercial 2D FEM software such as COMSOL Multiphysics and Lumerical can provide the modal solution of any waveguide within a few minutes. However, a rigorous optimization of the waveguide design parameters through parameter sweep often becomes intensive for a modern workstation depending on the complexity of a design. In this case, we are proposing an in-house developed ML-algorithm as a stepping stone for the multi-parameter optimization process where only the algorithm training (one time process) requires a few minutes of computational time to learn the features of similar types of waveguides.

III. Neural Network Training

The most common form of machine learning is supervised learning in which the training dataset consists of pairs of inputs and desired outputs, which are analyzed by ML algorithm to produce an inferred function. It is then used to obtain output values corresponding to any unknown input data samples. Supervised learning can be further categorised into a classification or regression problem, depending on whether the output variables have discrete or continuous values, respectively. In this paper, we considered the output predictions of different integrated photonics structures as a regression problem.

A. *Artificial Neural Network (ANN)*

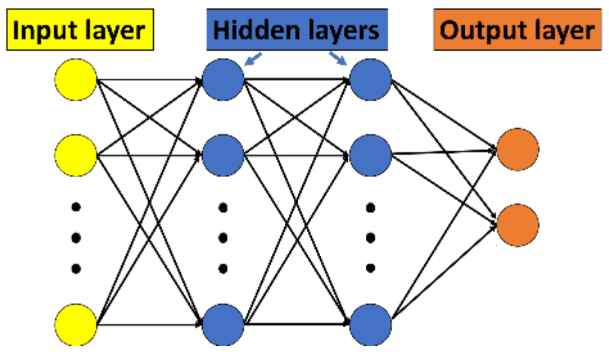


Fig. 2: General artificial neural network (ANN) representation, i.e. one input layer, two hidden layers, and one output layer.

ANN consists of a network of nodes, also called as neurons. ANN is a framework which is used to process complex data inputs and it learns from the specific input data without being programmed using any task-specific rules. One of the commonly used ANN is the multilayer perceptron (MLP). MLP consists of three or more number of layers. Here, in Fig. 2, we have shown a MLP with four layers of nodes: an input layer, two hidden layers and an output layer. These layers operate as fully connected layers, which means each node in one layer is connected to each node in the next layer. All the nodes have a variable weight assigned as an input which are linearly combined (or summed together) and passed through an activation function to obtain the output of that particular node.

B. *Algorithm of ANN*

The training procedure is illustrated in Fig. 3. Firstly, sufficient number of randomly generated data samples are collected from the simulations using COMSOL Multiphysics for slot, strip and directional coupler structures. Each case has an array of inputs, called features, and an array of numerically solved outputs, called labels. Waveguide width, height, material, gap between the waveguides, and operating wavelength values can be taken as the input variables which are assigned to

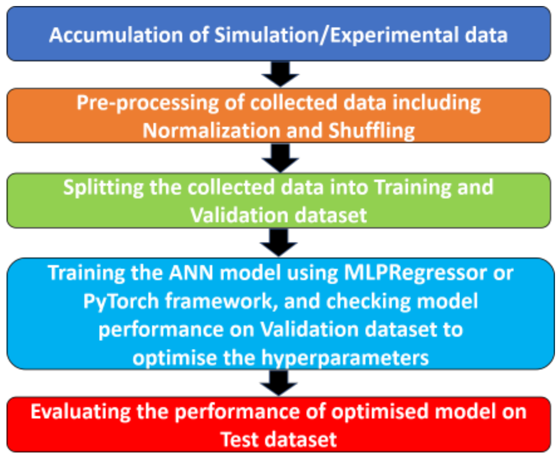

Accumulation of Simulation/Experimental data
Pre-processing of collected data including Normalization and Shuffling
Splitting the collected data into Training and Validation dataset
Training the ANN model using MLPRegressor or PyTorch framework, and checking model performance on Validation dataset to optimise the hyperparameters
Evaluating the performance of optimised model on Test dataset

Fig. 3: The flow chart of ANN implementation.

the nodes of the input layer. Effective index (n_{eff}), power confinement (P_{conf}), or coupling length (L_c) are taken as the output variables, which are assigned to nodes of the output layer depending on the specific design requirement. Next, preprocessing of the collected data is carried out by normalizing the input variables values between the range 0–1 to use a common scale. This is followed by shuffling of the normalized input data, otherwise the model can be biased towards particular input data values. Next step is to split the normalised input dataset into training and validation dataset. Validation dataset is used to provide an unbiased evaluation of a model fit on the training dataset while tuning various model parameters, also called as hyperparameters. 5–25% of data has been allocated for the validation dataset in this paper, while the rest was used for training the ANN model.

Neural networks have a tendency to closely or exactly fit a particular set of data during training, but may fail to predict the future observations reliably, which is known as overfitting. During overfitting, the model learns both the real and noisy data, which negatively impacts on new data. We can avoid overfitting through regularization such as dropout [26], while regularly monitoring the performance of the model during training on the generated validation dataset. Underfitting can be another cause for the poor performance of ANN model in which the trained model neither closely fits the initial data nor generalize to the new data. Hyperparameters needs to be tuned to reduce the mean squared error (mse) between the actual and predicted output values of the ANN model for a regression problem. During this optimization process, weights and biases of the model are repeatedly updated with each iteration or epoch using the backpropagation algorithm [25]. Various hyperparameters of choice includes activation functions, type of optimizer, number of hidden layers, number of nodes in each hidden layer, learning rate, number of epochs, and others.

1) Activation Functions: ANN connects inputs and outputs through a set of non-linear functions, which is approximated by using non-linear activation function. Sigmoid, Tanh (hyperbolic tangent), and ReLU (rectified linear unit) are few commonly used activation functions [2].

$$\text{Sigmoid} : \sigma(z) = \frac{1}{1+e^{-z}} \tag{1}$$

$$\text{Hyperbolic Tangent (Tanh)} : \sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2}$$

$$\text{Rectified Linear Unit (ReLU)} : \sigma(z) = max(0, z) \tag{3}$$

Among these, ReLU is used mostly as it trains the model several times faster in comparison to when using Tanh function, as discussed in [27].

2) Optimization Solvers: LBFGS, stochastic gradient descent (SGD), and Adam [28] solvers can be used to optimize the weights values during ML training process. Adam optimizer is a preferable choice as it works well on relatively large datasets.

3) Hidden Layers and Nodes: Number of layers or number of nodes in each layer of an ANN are decided by experimentation and from the prior experience of similar problems. There is no fixed rule to pre-decide their optimal values.

4) Learning Rate: Learning rate decides how much we are adjusting the weights of our network with each epoch or iteration. Choosing the lower value of learning rate means the model needs more epochs and longer time to converge. If the input dataset is big, it may take very long time to optimize the ANN model. On the other hand, if the learning rate has a large value, then the model might fail to converge at all with gradient descent [29], [30] overshooting the global minima. Learning rate can be chosen to have constant or adaptive value when using Scikit-learn MLPRegressor.

5) Epochs: Number of epochs to train a model should be decided by the user when mse value converges to acceptable lower limit. Depending on the dataset size, model training can be carried out using batches of inputs. In our case, when using MLPRegressor, we have used automatic batch size, while all the inputs are trained in one batch with PyTorch.

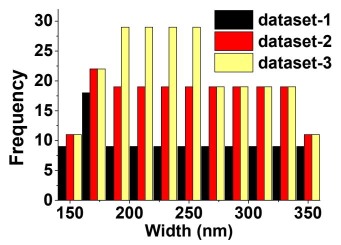
Once the optimal hyperparameters are obtained, the final step is to evaluate the performance of optimized trained model on the previously unseen test dataset (generated separately from the initially generated dataset) to observe the accuracy of the ANN model.

IV. Numerical Results and Discussion

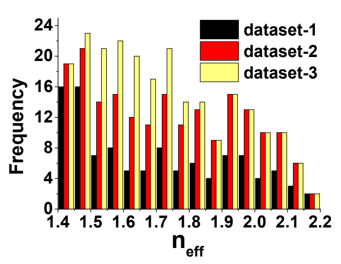
A. Slot Waveguide

Slot waveguide design structures are extensively being used for optical sensing applications [31], as the light is confined in low refractive index region, which allows strong interaction with the analyte leading to a large waveguide sensitivity. Here, we demonstrate the use of ML algorithms to predict the effective index (n_{eff}) and power confinement (P_{conf}) in a slot waveguide design [32], but first we optimize the various hyperparameters of the ANN model.

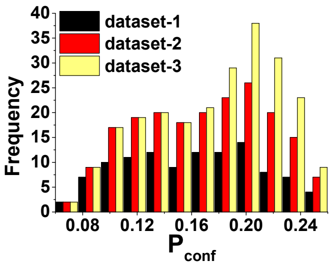
1) Histogram of Datasets: The training process requires a dataset of examples, which plays a crucial role in any ML algorithm. The accuracy of the trained model depends on the quality of the input data. A good training dataset which is well aligned with the problem to be solved is needed for ML to work properly. We have collected 3 different datasets to predict effective index (n_{eff}) and power confinement (P_{conf}) for a slot waveguide structure, shown in Fig. 4. Width, height, and gap between the waveguides in a slot waveguide design have been varied initially to record the n_{eff} and P_{conf} values for each dataset. n_{eff} and P_{conf} values recorded for a particular combination of width, height, and gap between the slot waveguides becomes one datapoint. Dataset-1 has 108 datapoints with nearly equal intervals between width, height, and gap. Dataset-2 also has points with nearly equal intervals but more values, 196 points. However, dataset-3 has higher frequency of data points for waveguide width in the range 200–250 nm, with 236 values in total. Figure 4a shows the histogram of different datasets plotted with respect to different widths of the waveguides. It should be noted here that these datasets are not explicit, which implies that dataset-1 is a subset of dataset-2 and similarly, dataset-2 is a subset of dataset-3. Figures 4b and 4c show the frequency of n_{eff} and P_{conf} for each of these



(a)



(b)



(c)

Fig. 4: Histogram of different datasets for slot waveguide with varying (a) width of waveguides, (b) n_{eff}, and (c) P_{conf}.

3 datasets. For our simulation setup, it took 2-3 minutes to record one datapoint, which means it took approximately 200, 400, 500 minutes to obtain dataset-1, dataset-2, and dataset-3, respectively. The time needed to collect one datapoint value may vary depending on the simulation/experimental setup.

2) Mean Squared Error: Mean squared error (mse) is considered as the loss function in a regression problem, which is defined as the average squared difference between the estimated and true values, given as:

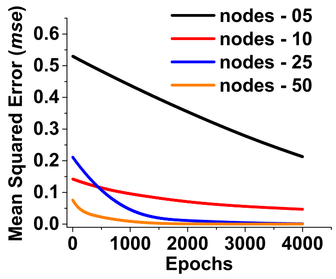
$$mse = \frac{1}{N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)^2 \tag{4}$$

where $\widehat{y_i}$ and y_i are the estimated and true data point values, respectively.

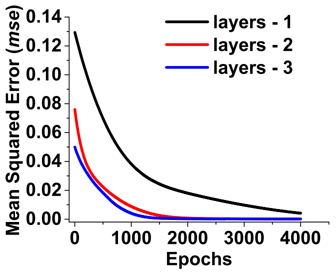
Smaller value of mse means the predicted regression values are closer to the original values and hence the model is well trained. Next, we compare the mse values to predict the n_{eff} for a slot waveguide design with different number of nodes or layers in an ANN model for dataset-3 using MLPRegressor from Scikit-learn. We have chosen dataset-3 as it has the maximum number of data points among the 3 datasets generated. Figure 5a shows that mse decreases faster to a stable value when number of nodes is larger. mse for nodes = 50 quickly reaches a stable low value of 0.0025 at epochs = 1500, shown by a orange line in comparison to 0.0192, 0.0820, and 0.3954 when nodes are taken as 25, 10 and 5, respectively. Random weights are assigned at the start of the algorithm, hence mse for more number of nodes at first epoch can be larger than that for less number of nodes, as can be seen from blue and red lines having values of 0.2112 and 0.1423 at first epoch, respectively. It can also be observed from red and blue lines that model with more number of nodes attains optimal updated weights quickly than those with lower number of nodes, as mse for nodes = 25 (blue line) decreases quickly. We run the simulations till 4000 epochs so as to be sure that mse decreases to a lower value. At epochs = 4000, mse values are 0.21279, 0.04685, 0.00109, and 0.00018 when number of nodes are 5, 10, 25, and 50, respectively. This shows that more neurons/nodes helps is achieving better accuracy for the ANN model by quickly decreasing the mse value to the minimum, but the computational loading also increases.

Next, we consider the mse variations when number of layers are varied in an ANN model having 50 nodes in each layer, as shown in Fig. 5b. The mse values of 0.0025 and 0.0006 are obtained for models with 2 and 3 number of layers, respectively at epochs = 1500 in comparison to 0.0243 when number of layers is equal to 1. Lower stable mse values at epochs = 4000 are 0.00412, 0.00018, and 0.00009 when number of layers are 1, 2 and 3, respectively, with each layer having 50 nodes. Following this study, we have considered the number of layers as 2 with 50 nodes in each layer for future optimization, to avoid more computational loading compared to if number of layers were chosen as 3.

3) Activation Functions: Sigmoid, Tanh and ReLU activation functions are tested to predict the n_{eff} and P_{conf} using MLPRegressor trained model with 2 layers having 50 nodes in each layer. Dataset-3 is used during the training process. It can be seen from Fig. 6a that Tanh and ReLU are closely predicting the n_{eff} values to the true values at waveguide height = 225 nm of the slot design. Data corresponding to waveguide height = 225 nm was never recorded or provided during the training of the model. However, data for other waveguide heights were used for the training. On the other hand, Sigmoid function is predicting almost a horizontal line

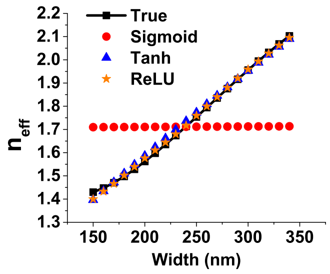


(a)

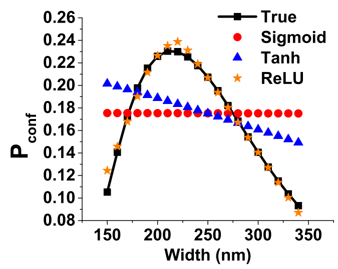


(b)

Fig. 5: Mean squared error (mse) using training dataset-3 for (a) different number of nodes with 2 hidden layers, (b) different number of hidden layers with 50 nodes in each hidden layer.



(a)

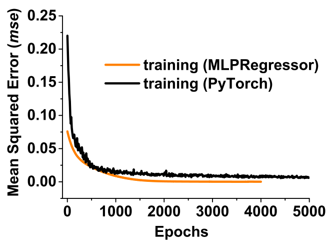


(b)

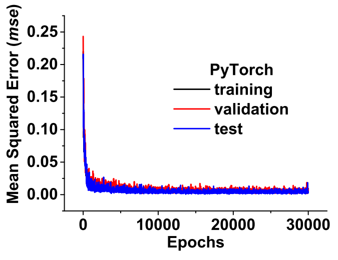
Fig. 6: Variation of (a) n_{eff} and (b) P_{conf} with waveguide width for different activation functions at waveguide height = 225 nm using training dataset-3.

as shown by red circle symbols. Sigmoid function failed to predict the n_{eff} values accurately, as it might converges well for a classification problem. When we tested to predict the P_{conf} of slot design, only ReLU activation function is able to predict the pattern much better, shown by orange star symbols and black rectangle symbols solid line in Fig. 6b, hence seems to be a better choice.

4) Comparing PyTorch Framework and MLPRegressor models: We have developed in-house codes using PyTorch framework and MLPRegressor from Scikit-learn. We have used 2 fully connected hidden layers each with 50 nodes. ReLU activation function and Adam optimizer are employed for both of the generated codes. Learning rate is chosen as 0.0001 or less. Dropout fraction of 0.5 is used for regularization to prevent over-fitting when using PyTorch framework, while dropout regularization is not available in the MLPRegressor. Number of epochs are decided based on when mse is reduced to a stable value for the considered photonics design structure.



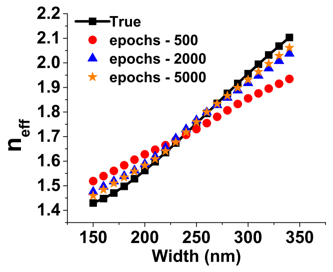
(a)



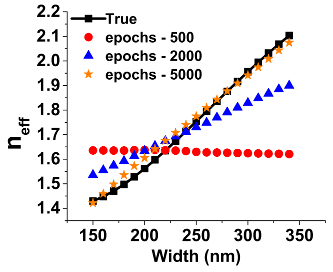
(b)

Fig. 7: Mean squared error (mse) using (a) training dataset-3 for MLPRegressor and PyTorch (b) training, validation, and test dataset-3 for PyTorch, having 2 hidden layers with 50 nodes in each layer.

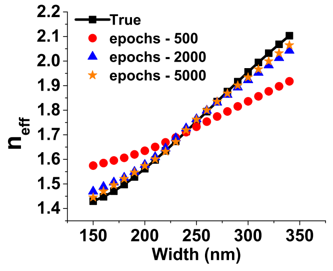
In Fig. 7, we have compared mse using MLPRegressor and PyTorch for training, validation and test datasets. It should be noted here that mse or loss function for validation and test datasets are not readily available in MLPRegressor. The loss curve in MLPRegressor also depends on the initially defined tolerance for the optimization (1e^{-8}, used in our case). MLPRegressor training automatically stops when the loss or score is not improving by at least tolerance in the consecutive iterations, which is shown by a orange solid line in Fig. 7a, where the mse curve stops by itself at around 4000 epochs. PyTorch can be used to visualise mse at very large epochs (>>4000) for training, validation, and test datasets, as shown in Fig. 7b. The fluctuations in the mse curves with PyTorch are due to the use of dropout fraction. mse for training, validation, and test datasets with PyTorch follow similar trend, and achieves a stable value at around 4000 epochs. mse for training dataset at epochs = 4000, 10000, and 30000 are 0.00808, 0.00451, and 0.00300, respectively. It can be observed here, that mse is almost similar and decreases slowly when epochs is greater than 4000. Hence, it is good to fix the epochs when mse achieves a stable low value, rather than allowing the algorithm to run for very large epochs. MLPRegressor is showing lower mse for training dataset in comparison to PyTorch, as shown in Fig. 7a. Using above mentioned parameters, we now predict n_{eff} and P_{conf} for a slot waveguide structure using PyTorch framework and MLPRegressor library for different datasets.



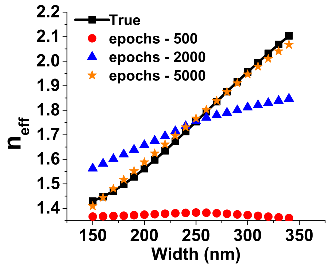
(a)



(b)

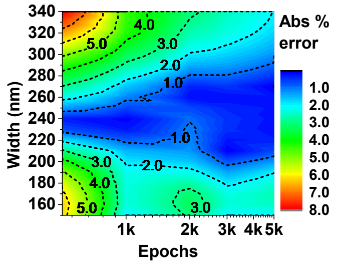


(c)

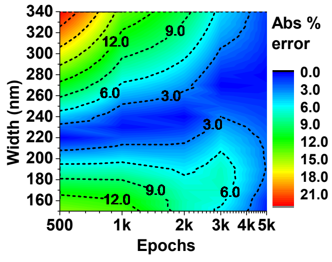


(d)

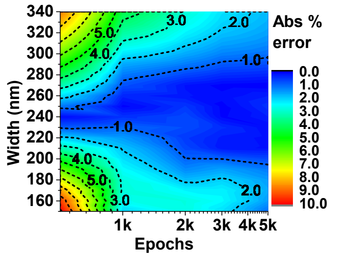
Fig. 8: Slot waveguide design predicting n_{eff} at waveguide height = 225 nm with (a) PyTorch using dataset-1, (b) MLPRegressor using dataset-1, (c) PyTorch using dataset-2, and (d) MLPRegressor using dataset-2.



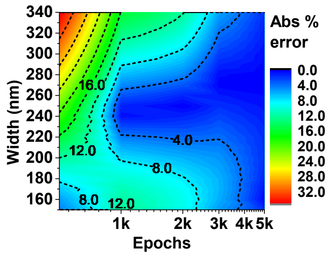
(a)



(b)



(c)



(d)

Fig. 9: Slot waveguide design showing contour of absolute percentage error for predicting n_{eff} at waveguide height = 225 nm with (a) PyTorch using dataset-1, (b) MLPRegressor using dataset-1, (c) PyTorch using dataset-2, and (d) MLPRegressor using dataset-2.

a) Effective Index (n_{eff}): Figure 8 shows the prediction of n_{eff} at waveguide height = 225 nm for a slot waveguide design. It should be noted that the datasets did not have any value corresponding to waveguide height = 225 nm. True n_{eff} values of test dataset are compared with predicted values for PyTorch and MLPRegressor models at 500, 2000 and 5000 epochs using dataset-1 and dataset-2 during training. It can be seen from subfigures (Figs. 8a–8d) that epochs = 5000 are sufficient to efficiently predict the n_{eff} values of test dataset for both PyTorch and MLPRegressor models using either dataset-1 or dataset-2. Predicted and true values are almost overlapping in all of the four cases, shown by orange star symbols and black rectangle symbols solid line, respectively.

Next, in Fig. 9, we have shown the percentage error in predicting the n_{eff} values for both PyTorch and MLPRegressor using dataset-1 and dataset-2. Here, percentage of error has been calculated by comparing the predicted solution with the simulated results using COMSOL Multiphysics. The absolute percentage error at epochs = 500 for both datasets varies between 6–10% (Figs. 9a and 9c) and 15-32% (Figs. 9b and 9d) for PyTorch and MLPregressor models, respectively. As the epochs are increased to 5000, absolute percentage error reduces to only 1-2% for all the cases which may be acceptable in predicting n_{eff} values for a slot design. It can be seen that both the models perform almost similar in predicting the true n_{eff} values.

b) Power Confinement (P_{conf}): Here, we train the model using PyTorch and MLPRegressor for different datasets to predict the P_{conf}. Figures 10a and 10b show the training prediction at waveguide height = 225 nm for PyTorch and MLPRegressor, respectively using dataset-1. It can be seen that even at epochs = 10000 (shown by orange star symbols), predicted value of P_{conf} is very much different from the true values especially when width is between 200–250 nm for both the algorithms. This error comes from the neural network modeling itself due to insufficient data points in the parameter space of width between 200–250 nm in dataset-1, which lead to underfitting of the trained model. Next, we train the neural network using dataset-2, which contain more data points compared to dataset-1, which was shown in Fig. 4a. Figures 10c and 10d show the predictions of trained model using dataset-2 for PyTorch and MLPRegressor, respectively. It is observed that trained model is performing better than when dataset-1 was considered, but still not good enough for widths in the range 200–250 nm. This error can be further reduced by collecting more data points in the widths ranging from 200-250 nm during training. Figures 10e and 10f show the trained model performance using dataset-3, which has more data values in the range 200–250 nm, as shown in Fig. 4a. It can be observed that PyTorch trained model (Fig. 10e) is still not performing efficiently at epochs = 10000 and predicting almost constant values in the waveguides width ranging between 200–250 nm. This error can be minimised if the dropout factor is reduced or taken as zero, but this may lead to over-fitting of the model. On the other hand, MLPRegressor (Fig. 10f) predicts the P_{conf} curve better at epochs = 10000, shown by orange star symbols, which is similar to the true shape (black square symbols solid line). Although, predicted P_{conf} values at waveguide width = 150 nm deviates significantly, but this can be further improved by collecting more data points in this range of waveguides width. For respective PyTorch and MLPRegressor models, it can be observed that prediction of P_{conf} using dataset-3 is better than dataset-2, which in turn is better than dataset-1 at epochs = 10000, shown by orange star symbols in Figs. 10a–10f. This shows that quality of dataset plays an important role along with the choice of algorithm and optimized values of hyperparameters.

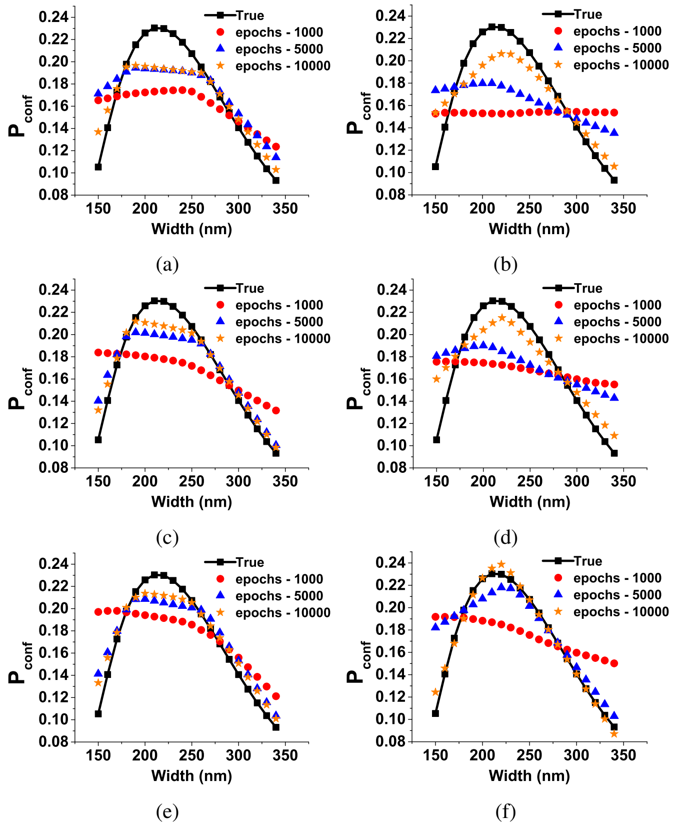


Fig. 10: Slot waveguide design predicting P_{conf} at waveguide height = 225 nm with (a) PyTorch using dataset-1, (b) MLPRegressor using dataset-1, (c) PyTorch using dataset-2, (d) MLPRegressor using dataset-2, (e) PyTorch using dataset-3, and (f) MLPRegressor using dataset-3.

Next, we compare the absolute percentage error for predicting P_{conf} using dataset-1, dataset-2, and dataset-3. When using dataset-1, both PyTorch and MLPRegressor have error ranging between 10–40% when epochs varies from 1000 to 10000, as shown in Figs. 11a and 11b, respectively. Figures 11c and 11d show that by taking more data values as in dataset-2 this error reduced between 7–30% with epochs. Dataset-3 shows better performance with trained model using both PyTorch and MLPRegressor, as shown in Figs. 11e and 11f, respectively. Absolute percentage error ranges between approximately 7–10% and 1–4% for PyTorch and MLPRegressor at epochs = 10000, respectively. This shows that MLPRegressor is performing better than PyTorch for this particular design specifications of slot waveguide with dataset-3. This performance difference between MLPRegressor and PyTorch is due to the different functionalities available in the algorithms, for example, dropout can only be implemented with PyTorch, while MLPRegressor uses exponential decay rates and numerical stability functions with Adam optimizer. Furthermore, the advantage of ANN can be more pronounced if the sample space is also large.

5) Training Dataset Sizes: In the proposed model algorithms, we split the initial collected data into training and validation dataset depending on the percentage parameter. Figure 12 compares the true P_{conf} values with predicted values at waveguide height = 225 nm using MLPRegressor

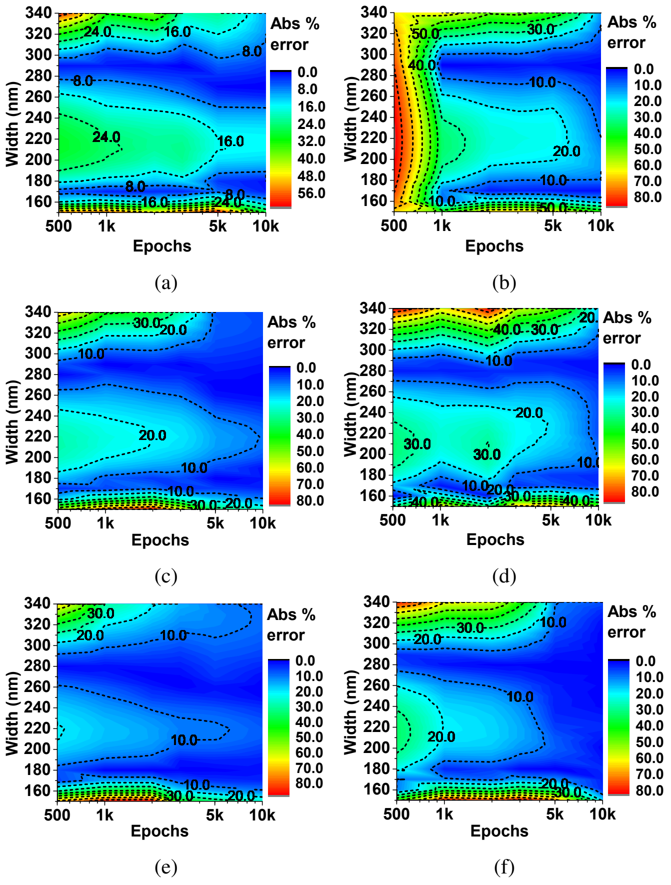


Fig. 11: Slot waveguide design showing contour of absolute percentage error for predicting P_{conf} at waveguide height = 225 nm with (a) PyTorch using dataset-1, (b) MLPRegressor using dataset-1, (c) PyTorch using dataset-2, (d) MLPRegressor using dataset-2, (e) PyTorch using dataset-3, and (f) MLPRegressor using dataset-3.

trained model with 25%, 50%, 75%, or 95% of initial collected data as training dataset. The data points are randomly selected for each case so that the final trained model is not biased towards any particular data points.

It can be seen that when 95% of data is used for training, the trained model is predicting the true values more accurately, as shown by orange star symbols. When 75% of data is taken as input (shown by green diamond symbols), we can observe that data points in the width ranging from 150–175 nm are not well predicted. Similarly, when only 50% or 25% of data is considered, algorithm is showing more errors in predicting the values especially in the waveguide width range of 150–225 nm, shown by blue triangle and red circle symbols, respectively. This is understandable as ANN prediction error after training can be large if the sample space has limited dataset points. So, if the dataset is overly reduced, error values of prediction increases significantly, as in the case of 25% in Fig. 12. It should be noted here that predicting an output only takes few milliseconds, once the model is trained by either MLPRegressor or PyTorch. On the other hand, it takes few minutes to get an output for particular waveguide dimensions with direct numerical simulation, which also depends on the density of the considered mesh.

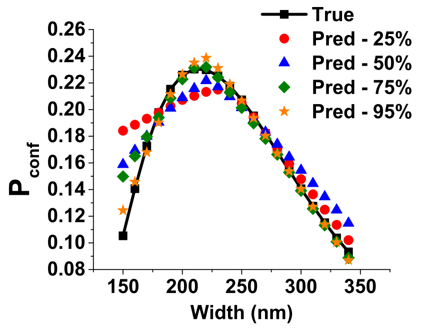


Fig. 12: Variation of P_{conf} with width at waveguide height = 225 nm for different data sizes of training dataset-3 using MLPRegressor.

B. Strip Waveguide

We now train an ANN model to obtain optimized hyperparameters for predicting n_{eff} values of a strip waveguide. MLPRegressor model is considered to train the initially recorded 225 data points with varying operating wavelength, height and width of the strip waveguide. Hyperparameters are optimized while training the model to obtain low and stable mse. The optimal weights of the model are then used or saved (to be used later) to predict the n_{eff} values on unseen test dataset.

Figures 13a, 13c, and 13e show the true and predicted n_{eff} values for different epochs with waveguide width, height and wavelength, respectively. Figures 13b, 13d, and 13f show the contour of absolute percentage error between true and predicted n_{eff} values with epochs (on logarithmic scale). It can be seen that percentage error is approximately 16–20% when epochs = 250, and decreases to approximately 2% when epochs = 2000 for all the above mentioned cases. When epochs = 4000, percentage error is further reduced to less than 1%. Hence, the trained model is performing good at epochs = 4000 and we can now save the trained model weights at epochs = 4000 for future testing.

Next, we check the performance of our saved model for some random strip waveguide design parameters which were not available in the training dataset. True and predicted n_{eff} are compared in Table I when all the input parameters (operating wavelength, waveguides height and width) are unknown to the trained model, which implies that outputs corresponding to these parameters were never recorded during the initial data collection. It can be observed that when epochs = 4000, absolute value of percentage error to predict n_{eff} is less than

TABLE I: Comparing predicted with true n_{eff} values and corresponding absolute percentage error for random wavelength, height, and width of strip waveguide design.

Wavelength (μm)	Height (nm)	Width (nm)	Predicted - n_{eff}		True - n_{eff}	Absolute % error - n_{eff}	
			epochs - 1000	epochs - 4000		epochs - 1000	epochs - 4000
1.52	210	490	2.323764	2.418138	2.424294	4.14	0.25
1.52	230	510	2.333022	2.541238	2.528034	7.71	0.52
1.54	210	490	2.32193	2.396881	2.402842	3.36	0.24
1.54	230	510	2.342372	2.520985	2.50819	6.61	0.51
1.56	210	490	2.328336	2.370507	2.38129	2.22	0.45
1.56	230	510	2.354483	2.489118	2.488223	5.37	0.03
1.58	210	490	2.330601	2.34713	2.359644	1.23	0.53
1.58	230	510	2.35926	2.45187	2.468138	4.41	0.65

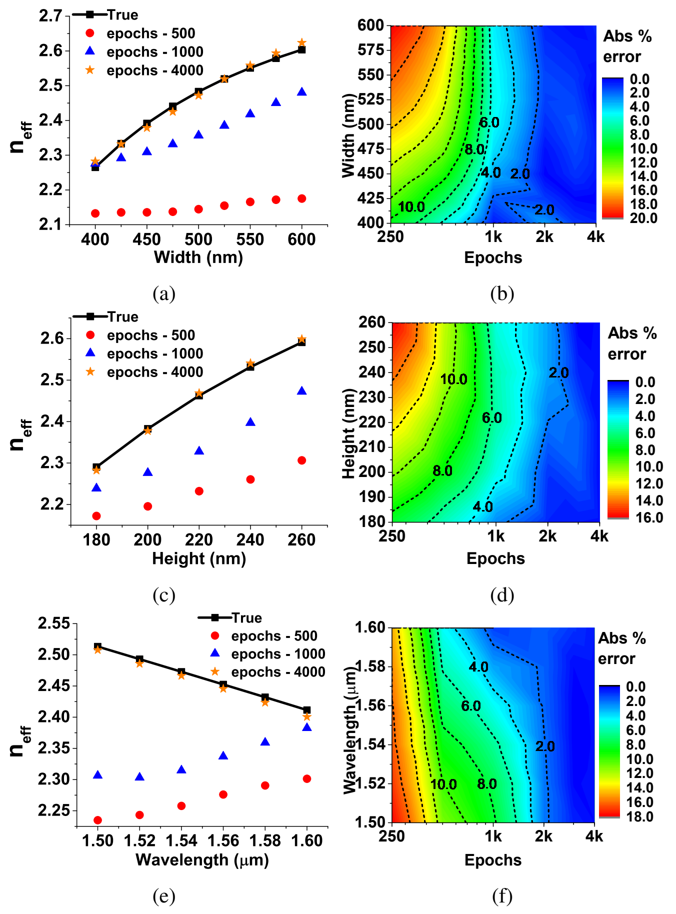


Fig. 13: Strip waveguide design (a) predicting n_{eff} at waveguide height = 230 nm, (b) showing contour of absolute percentage error for predicting n_{eff} at waveguide height = 230 nm, (c) predicting n_{eff} at waveguide width = 510 nm, (d) showing contour of absolute percentage error for predicting n_{eff} at waveguide width = 510 nm, (e) predicting n_{eff} with change in wavelength at waveguide width = 510 nm, and (f) showing contour of absolute percentage error for predicting n_{eff} at waveguide width = 510 nm.

1% for randomly chosen different input design parameters. This demonstrate that the model is performing well to predict n_{eff} for a strip waveguide design.

C. Directional Coupler

Directional coupler has been the main component in many photonic devices including spot-size converter [33], mode demultiplexer [34], polarization rotator [35], polarization splitter [36], etc.

Here, we present the application of ML algorithm using MLPRegressor to predict the coupling length (L_c) of a directional coupler. Again, we first optimize the model parameters to obtain minimum stable mse value using training dataset for a directional coupler design. Different height, width and gap between the waveguides are considered as the input parameters, and L_c is taken as output parameter during the ANN training. Figure 14a shows the predicted L_c when epochs are taken as 500, 1000, 5000, and 10000. It can be observed that predicted L_c values are closer to the true values when epochs are 5000 or 10000 at waveguide height = 230 nm. It should be noted that the training dataset did not have L_c values when waveguide height = 230 nm. Figure 14b shows that there is approximately 6–10% of absolute percentage error in L_c value at different widths when epochs = 5000. This error is reduced to 1–4% for different widths when model is trained to 10000 epochs. Increasing the initially recorded data points to train the model can help in further reducing this absolute percentage error between predicted and true L_c values of a directional coupler. This model can be trained to calculate L_c for any given height, width, separation and operating wavelength of a directional coupler.

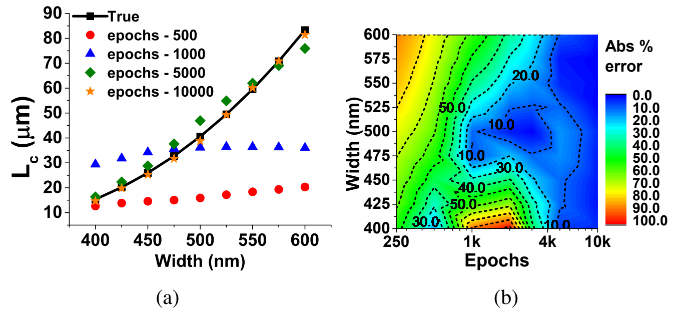


Fig. 14: Directional coupler design (a) predicting L_c at waveguide height = 230 nm and (b) showing contour of absolute percentage error for predicting L_c at waveguide height = 230 nm.

V. Conclusion

In summary, a machine learning model for predicting the effective index, power confinement and coupling length in a slot waveguide, strip waveguide and directional coupler design structure has been developed. Dataset-3 was the better choice in comparison to other datasets for the considered slot waveguide design, as it contain more number of input data points which helps the machine learning model to be trained better. Absolute percentage error in predicting effective index for a slot waveguide design was lower than 2% for both PyTorch or MLPRegressor. PyTorch and MLPRegressor models gives absolute percentage error of approximately 7–10% and 1–4%, respectively for predicting the power confinement, which shows that MLPRegressor model performs better. ReLU activation function is preferred as it predicts the n_{eff} and P_{conf} curve better in comparison to when using Tanh or Sigmoid function. MLPRegressor model has also been used to predict the effective index in a strip waveguide design with 99% accuracy. Similarly, hyperparameters have been optimized for a separate MLPRegressor model to predict the coupling length for a directional coupler design, giving only 1–4% of absolute percentage error. To the best of our knowledge, this is the first time machine learning is used in conjugation with rigorous finite element method for various nanophotonic waveguide analyses. Our approach can accurately predict the waveguide parameters without extensive use of the computationally expensive time- and frequency-domain numerical methods. Therefore, we believe machine learning combined with numerical approaches has an immense potential to enrich the design and fabrication of exotic nanophotonic waveguides and resonators.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] F. N. Khan, Q. Fan, C. Lu, and A. P. T. Lau, "An optical communication's perspective on machine learning and its applications," *Journal of Lightwave Technology*, vol. 37, no. 2, pp. 493–516, 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] D. Cote, "Using machine learning in communication networks," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D100–D109, 2018.

[5] N. Borhani, E. Kakkava, C. Moser, and D. Psaltis, "Learning to see through multimode fibers," *Optica*, vol. 5, no. 8, pp. 960–966, 2018.

[6] M. H. Tahersima, K. Kojima, T. Koike-Akino, D. Jha, B. Wang, C. Lin, and K. Parsons, "Deep neural network inverse design of integrated photonic power splitters," *Scientific Reports*, vol. 9, no. 1, p. 1368, 2019.

[7] J. Baxter, A. C. Lesina, J.-M. Guay, A. Weck, P. Berini, and L. Ramunno, "Plasmonic colours predicted by deep learning," *Scientific Reports*, vol. 9, no. 1, p. 8074, 2019.

[8] D. Gostimirovic and W. N. Ye, "Automating photonic design with machine learning," in *2018 IEEE 15th International Conference on Group IV Photonics (GFP)*. IEEE, 2018, pp. 1–2.

[9] T. Asano and S. Noda, "Optimization of photonic crystal nanocavities based on deep learning," *Optics Express*, vol. 26, no. 25, pp. 32 704–32 717, 2018.

[10] A. da Silva Ferreira, G. N. Malheiros-Silveira, and H. E. Hernández-Figueroa, "Computing optical properties of photonic crystals by using multilayer perceptron and extreme learning machine," *Journal of Lightwave Technology*, vol. 36, no. 18, pp. 4066–4073, 2018.

[11] W. Ma, F. Cheng, and Y. Liu, "Deep-learning-enabled on-demand design of chiral metamaterials," *ACS Nano*, vol. 12, no. 6, pp. 6326–6334, 2018.

[12] C. Barth and C. Becker, "Machine learning classification for field distributions of photonic modes," *Communications Physics*, vol. 1, no. 1, p. 58, 2018.

[13] C. L. Chen, A. Mahjoubfar, L.-C. Tai, I. K. Blaby, A. Huang, K. R. Niazi, and B. Jalali, "Deep learning in label-free cell classification," *Scientific Reports*, vol. 6, p. 21471, 2016.

[14] A. Tittl, A. John-Herpin, A. Leitis, E. R. Arvelo, and H. Altug, "Metasurface-based molecular biosensing aided by artificial intelligence," *Angewandte Chemie International Edition*, 2019.

[15] B. Karanov, M. Chagnon, F. Thouin, T. A. Eriksson, H. Bülow, D. Lavery, P. Bayvel, and L. Schmalen, "End-to-end deep learning of optical fiber communications," *Journal of Lightwave Technology*, vol. 36, no. 20, pp. 4843–4855, 2018.

[16] T. A. Eriksson, H. Bülow, and A. Leven, "Applying neural networks in optical communication systems: possible pitfalls," *IEEE Photonics Technology Letters*, vol. 29, no. 23, pp. 2091–2094, 2017.

[17] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "An overview on application of machine learning techniques in optical networks," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1383–1408, 2018.

[18] R. M. Morais and J. Pedro, "Machine learning models for estimating quality of transmission in DWDM networks," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D84–D99, 2018.

[19] E. Jones, T. Oliphant, and P. Peterson, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[22] F. Chollet, "Keras," https://keras.io, 2015.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[25] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School*, vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, pp. 21–28.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[30] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[31] C. A. Barrios, "Optical slot-waveguide based biochemical sensors," *sensors*, vol. 9, no. 6, pp. 4751–4765, 2009.

[32] F. Dell'Olio and V. M. N. Passaro, "Optical sensing by optimized silicon slot waveguides," *Optics Express*, vol. 15, no. 8, pp. 4977–4993, 2007.

[33] W. Jiang, N. Kohli, X. Sun, and B. A. Rahman, "Multi-poly-silicon-layer-based spot-size converter for efficient coupling between silicon waveguide and standard single-mode fiber," *IEEE Photonics Journal*, vol. 8, no. 3, pp. 1–12, 2016.

[34] C. Pan and B. M. A. Rahman, "Accurate analysis of the mode (de) multiplexer using asymmetric directional coupler," *Journal of Lightwave Technology*, vol. 34, no. 9, pp. 2288–2296, 2016.

[35] A. Barh, B. M. A. Rahman, R. K. Varshney, and B. P. Pal, "Design and performance study of a compact SOI polarization rotator at 1.55 μm," *Journal of Lightwave Technology*, vol. 31, no. 23, pp. 3687–3693, 2013.

[36] W. Jiang, X. Sun, and B. M. A. Rahman, "Compact and fabrication-tolerant polarization splitter based on horizontal triple-slot waveguide," *Applied Optics*, vol. 56, no. 8, pp. 2119–2126, 2017.