Physical Communication 41 (2020) 101057

Contents lists available at ScienceDirect

Physical Communication

journal homepage: www.elsevier.com/locate/phycom

Full length article

Novel suboptimal approaches for hyperparameter tuning of deep neural network [under the shelf of optical communication]

M.A. Amirabadi *, M.H. Kahaei, S.A. Nezamalhosseini

School of Electrical Engineering, Iran University of Science and Technology (IUST), Tehran, 1684613114, Iran

ARTICLE INFO

Article history:
Received 6 September 2019
Received in revised form 13 January 2020
Accepted 21 February 2020
Available online 27 April 2020

Keywords:
Hyperparameter tuning
Grid search
Deep neural network
Free space optical communication
Fiber optical communication

ABSTRACT

Grid search is the most effective method for tuning hyperparameters in machine learning (ML). However, it has high computational complexity, and is not appropriate when here are many hyperparameters to be tuned. In this paper, two novel suboptimal grid search methods are presented, which search the grid marginally and alternatively. In order to show the efficiency of hyperparameter tuning by the proposed methods four datasets are used. Two datasets were collected by simulating FSO and fiber OC links by MATLAB software, and two other datasets were collected by experimental setups for FSO and fiber OC links in Optisystem software. Results indicate that despite greatly reducing computational complexity, the proposed methods achieve a favorable performance. The proposed structures are compared with some of the recently published most relevant works, and the efficiency of the proposed methods is proved.

1. Introduction

During the last decades, optical communication (OC) systems have attracted many considerations towards themselves. This is mainly because of progressions in optoelectronic devices and reduction in their costs. These systems have a large un-licensed secure bandwidth, and simple implementation. They are suitable for the last-mile backup/ bottleneck applications for next generation communication systems [1,2]. Despite these advantages, they have some critical barriers, which limits their practical applications. These barriers are created by the propagating media; e.g., free space optical (FSO) communication is highly sensitive to atmospheric turbulence, which is caused by random fluctuations of the received signal intensity [3,4], or fiber OC is highly sensitive to fiber linear/ nonlinear effects.

Recently, machine learning (ML) has been extensively used in various OC applications including mitigation the mentioned barriers and achieved significant results. ML is a branch of artificial intelligence that makes a machine (an OC system) to learn a task by training based on an input dataset [5]. Despite flexibility and performance of ML algorithms, they could not learn complex tasks. Deep learning (DL) is a branch of ML that could learn deeper, and is appropriate for complex tasks [6,7]. DL or deep neural network (DNN) learning has been widely used for complex OC based tasks, such as fiber effects mitigation [8], performance monitoring [9], and modulation format identification [10,11].

One of the main ambiguities and difficulties in working with DNN is hyperparameter tuning. Hyperparameters are the design parameters, and could affect the training qualification. Hyperparameter tuning is very important [12], because the difference between a tuned DNN, and a non-tuned DNN is the difference between a machine that learns complex task perfectly and an algorithm that could not learn anything.

1.1. Related works

Recently, the hyperparameter tuning, due to its importance, has changed to a new interesting topic in ML community. However, there is lack of investigation about hyperparameter tuning in ML for OC community. The most related works are the hyperparameter tuning in pure ML literatures; so, this section focuses on these works. Some works do hyperparameter tuning empirically or based on prior knowledge from previous literatures; so, their solutions are mostly based on trial-and-error and might not be trustable [13]. The hyperparameter tuning algorithms are either model free or model based. Model free algorithms do not use the knowledge about the solution space extracted during the optimization; this category includes manual search [14,15], grid search [16], and random search (Fig. 1) [17–19]. Model based algorithms, build a surrogate model of the hyperparameter space via its precise exploration and exploitation; this category includes Bayesian search [20,21].

* Corresponding author.
E-mail addresses: m_amirabadi@elec.iust.ac.ir (M.A. Amirabadi), kahaei@iust.ac.ir (M.H. Kahaei), nezam@iust.ac.ir (S.A. Nezamalhosseini).

https://doi.org/10.1016/j.phycom.2020.101057
1874-4907/

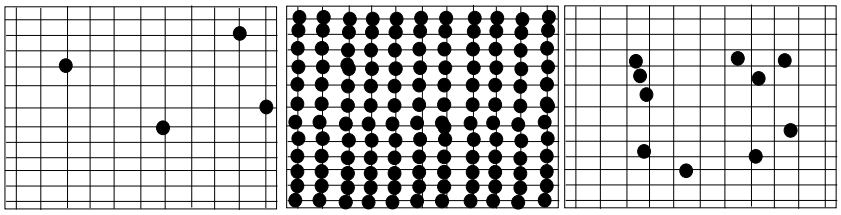


Fig. 1. Manual search (left), grid search (center), and random search (right).

Considering the model free algorithms, in manual search, the user chooses some sets of hyperparameters (based on previous knowledge), trains the machine by them, calculates the performance, and selects the set of hyperparameters that has the best performance. However, this technique is dependent on the previous knowledge and its reliability is dependent on the correctness of the previous knowledge. In this situation, random search [17] would have better results (with the same computational complexity). The complexity of the hyperparameter tuning problem depends on the complexity of the application, for which the ML is used. The more complicated is the application, the more precise tuning is required; but manual and random search methods do not provide an accurate tuning, because they investigate few sets of hyperparameters [22]. In this situation, grid search method would have better results (with high computational complexity). This method is the best among model based and model free methods, because it applies a heuristic search and considers all sets of hyperparameters. In grid search method, first the user collects all possible sets of hyperparameters, then trains the machine based on them. The set with the best performance is chosen as the tuning output. The main drawback of grid search method is its high complexity. Accordingly, it is commonly used when there are few hyperparameters to be tuned; in other words, grid search is performed when the best combinations are known [17]. According that the grid search method has a better performance, many ML literatures are based on this algorithm. However, they used small grid size and small range for reducing complexity, so, their results may not be trustable [23,24]. As another solution for the complexity problem of grid search and having reliability, in some other ML literature some hyperparameters are adjusted by grid search, some by random search and some by manual search [25]. However, this method does not have the generality of grid search method.

Considering the model based algorithms, most of them are based on Bayesian optimization [26], which uses a set of previously observed parameters and resulting accuracies, and makes an assumption about unobserved parameters. Acquisition function using this information suggests the next set of parameters. Bayesian optimization adopts probabilistic surrogate models like Gaussian processes to approximate and minimize the validation error function of hyperparameter values [27]. However, probabilistic surrogates require accurate estimates of sufficient statistics of error function distribution (e.g. covariance). So, it requires some function evaluations with a sizeable number of hyperparameters, and is not appropriate when number of hyperparameters is large. For solving this problem non-probabilistic hyperparameter tuning methods including radial basis function error surrogates are suggested [28,29]. Another solution is encompassing the covariance matrix adaptation evolution [30]. In [31], very initial results of evolving DNNs using genetic algorithms were reported. In some ML literatures, Bayesian search is used for joint finding the searching structure and tuning hyperparameter [12, 32]; in this situation, tree-based Bayesian search are better to be used [33,34].

1.2. Novelties and contributions

As mentioned above, each hyperparameter tuning method has its own advantages and disadvantages, and according to the application and situation, one of them could be used. However, in general, grid search method has the best performance, but it is required of running multiple full training rounds and has high computational complexity, which increases exponentially while adding tunable hyperparameters. This method is preferable for simple ML algorithms, which have few tunable hyperparameters; but, it is not in complex ML algorithms such as DNN, which has many tunable hyperparameters [17].

A suboptimal search method or a method that could automatically tune hyperparameters in one training round would be more practical, especially when the user do not have a strong intuition regarding appropriate sets of hyperparameters [35]. Accordingly and considering the tradeoff between performance and complexity, this paper presents two novel suboptimal grid search methods for hyperparameter tuning of a DNN. In the first method, marginal search is developed over the grid points, in the sense that at first a grid with large size and long range is constructed, an initial point is selected, and one of the hyperparameters is tuned over the whole range. Then considering the initial point, another hyperparameter is tuned, and the same procedure will be continued until all of hyperparameters would be tuned. In this method, at each round, all of the hyperparameters are fixed and only one of them will be tuned. In the second method, alternating optimization is used for tuning hyperparameters, in the sense that at first a grid with large size and long range is constructed, an initial point is selected, and one of the hyperparameters is tuned over the whole range. This point is chosen as the initial point for the next round, and the same procedure will be continued until all of hyperparameters would be tuned. In order to examine the accuracy and universality of the proposed methods, four different OC based datasets are used. Two datasets are collated from MATLAB software simulations for FSO and fiber OC systems, and two other datasets are collected from Optisystem software for FSO and fiber OC. Optisystem is a software that considers many real scenarios and its results are close to experimental results. In these structures, a DNN is applied at the receiver for joint equalizer, detector, and demodulator. The novelties and contributions of this paper include:

(1) Presenting a step by step explanation about hyperparameter tuning of a DNN (see Section 4).

(2) Presenting and solving the problem of hyperparameter tuning in ML for OC applications.

(3) Presenting two novel suboptimal grid search hyperparameter tuning methods.

(4) Presenting an ML technique applicable for both FSO and fiber OC systems.

(5) Deploying a DNN for joint equalization, detection, and demodulation in FSO and fiber OC systems.

The rest of this work is organized as follows; Section 2, and 3 present the channel model and DNN based OC system model, respectively. Section 4 presents a step by step explanation on hyperparameter tuning of a DNN. Section 5 presents the proposed methods. Section 6 is the results and discussions, and Section 7 is the conclusion of this paper.

2. Channel model

In order to show the universality of the proposed methods for OC tasks, two completely different OC systems including fiber OC and FSO systems are considered. This section presents their channel models.

2.1. FSO Channel model

Various statistical distributions have been proposed to model atmospheric turbulence of FSO channel, e.g., Exponential–Weibull [36], Generalized Malaga [37], Lognormal [38], Gamma–Gamma [39], and Negative Exponential [40]. Among them, Gamma–Gamma is commonly used and is highly accompanied with the actual results. The probability distribution function of Gamma–Gamma atmospheric turbulence is as follows:

$$f(I) = \frac{2(\alpha\beta)^{\frac{\alpha+\beta}{2}}}{\Gamma(\alpha)\Gamma(\beta)} I^{\frac{\alpha+\beta}{2}-1} K_{\alpha-\beta}\left(2\sqrt{\alpha\beta I}\right); I > 0, \tag{1}$$

where I is the atmospheric turbulence intensity, $\Gamma(.)$ is the well-known gamma function, $K_{.}(.)$ is modified Bessel function of the second kind, $\alpha = \left[\exp\left(0.49\sigma_R^2/\left(1+1.11\sigma_R^{12/5}\right)^{7/6}\right)-1\right]^{-1}$ is the shaping parameter, and $\beta = \left[\exp\left(0.51\sigma_R^2/\left(1+0.69\sigma_R^{12/5}\right)^{5/6}\right)-1\right]^{-1}$ is the scaling parameter that characterize the irradiance fluctuations in Gamma–Gamma model, where $\sigma_R^2 = 1.23c_n^2k^{7/6}l^{11/6}$, where $k = 2\pi/\lambda$ is the wave number, and l is FSO link distance [41].

2.2. Fiber optic channel model

Modeling of fiber optic could help better and simpler understanding of the fiber effects. The model expressed for fiber OC is the Gaussian noise model [42]. In this model, the fiber effects are modeled as an additive Gaussian noise with a variance dependent on fiber effect parameters. In this model the input signal to the fiber is assumed to be Gaussian, and the modulation format has no effect on the model; so [43] extended this model and applied it for all signals considering modulation format. According to [43] model, and considering x as the input signal to a fiber, the output signal would be as follows:

$$y = x + n_{ASE} + n_{NL}, \tag{2}$$

where n_{ASE} is the amplifier spontaneous emission noise, and n_{NL} is nonlinear interference noise [42], which its variance is a function of the optical launch power and moments of the constellation. The Matlab and Python codes for obtaining this variance could be found in [44] and [45], respectively.

3. DNN Based OC system model

The DNN based OC system model is shown in Fig. 2. The generated M-ary symbols are first converted to one-hot vectors (because at the end, the output of the DNN would be an M-ary vector, which should be compared with the transmitted vector). The produced one-hot vectors are then mapped to an MQAM constellation, transmitted through FSO/ fiber channel and added by the receiver input additive white Gaussian noise (AWGN) with zero mean and unit variance. The receiver is assumed to implement equalization, detection, and demodulation jointly, by use of a DNN. The DNN has 2 inputs (because the input is complex, but DNN only takes real values), and M outputs, N_{hid} hidden layers, and N_{neu} hidden neurons. The target is to adjust DNN parameters (weights W and biases b) such that, the M-ary output vector of the DNN ($\hat{\boldsymbol{s}}$) be the same as the transmitted M-ary one-hot vector ($\boldsymbol{s}$). In another words, the following loss function should be minimized:

$$L(\boldsymbol{\theta}) = \frac{1}{K}\sum_{k=1}^{K}\left[l^{(k)}\left(\boldsymbol{s},\hat{\boldsymbol{s}}\right)\right], \tag{3}$$

where $\boldsymbol{\theta}$ is the vector containing DNN parameters, K is the batch size, $l(.,.)$ is loss function. This loss function could be minimized by calculating the gradients of all layers and updating the DNN parameters by back propagating the error using the following formula:

$$\theta^{(j+1)} = \theta^{(j)} - \eta\nabla_\theta\tilde{L}\left(\theta^{(j)}\right) \tag{4}$$

, where η is learning rate, j is the training step iteration, and $\nabla_\theta\tilde{L}(.)$ is the estimate of the gradient. This problem could be solved using the well-known Stochastic Gradient Descent methods [46]. However, in order to do that, first should tune the hyperparameters.

4. Hyperparameter tuning of a DNN

Among ML algorithms, DNN has the most number of hyperparameters to be tuned. So, there would be many difficulties and ambiguities while tuning them. In this section, a brief but sufficient explanation is presented about a standard hyperparameters, and their tuning. This section shows that at each step which hyperparameter should be tuned and how to tune it. The steps of hyperparameter tuning of an standard DNN (in a row) are selecting number of epoch and batch size, normalizing input data, selecting the layer type, selecting number of neurons and hidden layers, selecting the activation functions, selecting the loss function, selecting the optimizer, selecting the learning rate and number of iterations.

4.1. Selecting number of epochs and batch size

The first step for hyperparameter tuning is simply preparing the input dataset (features). Before designing the structure of a DNN for a specific task, the collected dataset should be preprocessed. Terminologies like number of epochs and batch size appear while the entering data is large and cannot be passed through DNN at once. One epoch is when the entire dataset is passed through the DNN. Since the entering dataset is large, it should be divided into several smaller batches. Updating DNN parameters in one epoch is not a good idea, and leads to underfitting; accordingly, it should be fed in multiple epochs. As the number of epochs increases, the curve goes from under fitting to good fitting and to overfitting. The number of epochs is different for different datasets, but it is related to the diversity of the dataset. Batch size is the total number of training samples presented in a single batch.

4.2. Normalizing input dataset

The second step in hyperparameter tuning is reducing the variance of input dataset by normalization (scaling). The DNN could better track the changes and derive the relationships when the variance of the input dataset is limited. So, normalization would accelerate training. The input dataset could be either discrete or continuous. The continuous dataset could be in the range of $[-1, 1]$, or $[0, 1]$, or distributed normally with zero mean and unit variance. The discrete dataset could be presented by one-hot vector representation. In should be noted that the same normalization method should be used for both training and testing data. Also, it is important that if train and test dataset do not have the same distribution, validation and test datasets should have the same distribution. There are many ways for normalization available in literature [47], however, the input dataset of this paper is normalized by itself (one-hot vectors).

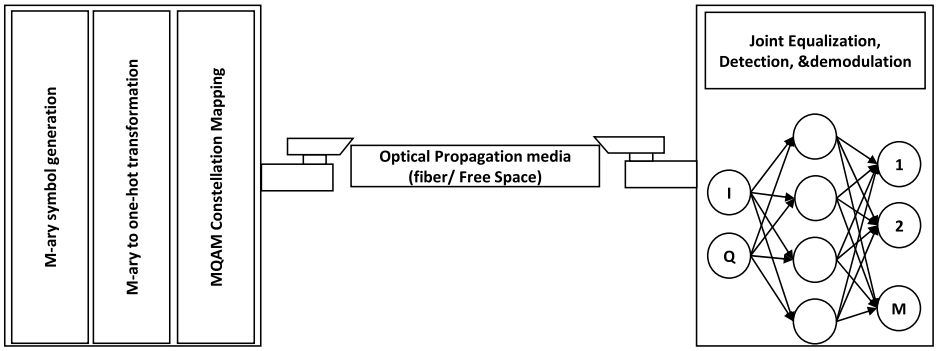


Fig. 2. The DNN based OC system model.

4.3. Selecting the layer type

The third step in tuning hyperparameters of a DNN is selecting the layer type. There are many layer types available for DNN, which each of them is proper for some specific tasks, and could not be used everywhere. For example, the most popular layer types are Feedforward, Radial Basis Function (for linear data), Multilayer Perceptron (for non-linear data), Convolutional (for imagery data), and Recurrent (for data with memory). If there would be no knowledge about the input data, then the best solution would be trial and error. In this paper layer type selection is not investigated, because it has been already selected (DNN, which is an extension of Multilayer Perceptron). However, because there is no note in ML for OC over this subject in this part the famous layer types are reviewed briefly and sufficiently.

Feedforward [48] is one of the simplest types, in which the input dataset propagates in only one direction through one or more layers. It is used for dealing with noisy datasets, e.g. in face recognition and computer vision. Radial Basis Function [49] considers the distance of any point relative to the center. It has two layers, in the inner layer, the features are combined with the radial basis function. It is used in power restoration systems. Multilayer Perceptron [50] has three or more fully connected layers. It uses a nonlinear activation function (mainly hyperbolic tangent or logistic). It is used in speech recognition and machine translation. Convolutional neural network (NN) [51] contains one or more interconnected or pooled convolutional layers. Before passing to the next layer, it applies a convolutional operation on the input. Accordingly, the network can be much deeper but with much fewer parameters. It is used in image and video recognition, natural language processing, recommender systems, semantic parsing, and paraphrase detection. In Recurrent NN [52], except the first layer, other layers have feedback. From each time-step to the next, each node acts as a memory cell and remembers its previous information. It is used in text-to-speech conversion.

4.4. Selecting number of the neurons and layers

After selecting the layer type, it is the turn to choose number of layers as well as number of neurons, which are completely dependent on the input data type. There is no specific formulation for tuning the number of hidden neurons as well as layers. However, there are some empirical rules, e.g. the optimal size of the hidden layer is usually between the size of the input and output layers. In linear data, there is no need for hidden layer. So, usually one hidden layer is sufficient and situations, in which the addition of a second (or third, etc.) hidden layer improves performance are very few. NN presents a nonlinear function, accordingly, in binary classification task one layer would be sufficient for this purpose. In these situations, the number of neurons is the geometric mean of the neurons in the input and output layers [53].

Table 1
Tensorflow activation functions.

Activation function	Equation
Linear	$f(x) = cx$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Relu	$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$
Elu	$f(x) = \begin{cases} x & x > 0 \\ \alpha\,(e^{x}-1) & x \leq 0 \end{cases}$
Selu	$f(x) = \lambda \begin{cases} x & x > 0 \\ \alpha\,(e^{x}-1) & x \leq 0 \end{cases}$
Relu6	$f(x) = \begin{cases} 6 & 6 < x \\ x & 0 < x \leq 6 \\ 0 & x \leq 0 \end{cases}$
Crelu	$f(x) = \begin{cases} max(0,x) & x > 0 \\ max(0,-x) & x \leq 0 \end{cases}$
Softmax	$f(x_i) = \frac{e^{-x_i}}{\sum_j e^{-x_i}}$
Softsign	$f(x) = \frac{x}{1+\lvert x\rvert}$
Softplus	$f(x) = \ln(1+e^{x})$

4.5. Selecting the activation function

The next step in hyperparameter tuning is specifying the activation function type, which is somehow related to the task and input data; e.g., Sigmoid and Softmax functions are well for binary and M-ary classification, respectively, and use of them will lead to faster convergence. If there is no knowledge about the task or data, then Rectifier Linear Unit (Relu) is a good choice. The activation function is either linear or nonlinear; nonlinear activation function produces output in range (0, 1) or (−1, 1). Therefore, it could be used for classification task. Linear activation function produces any output values. The results of this paper are obtained by deploying a DNN in Tensorflow environment, the available activation function in this environment are collected in Table 1. The history behind their generation is quoted in the following.

The step function is threshold based, and activates only when the input is above a certain level. In binary classification it works, but its Achilles Hell is M-ary classification problem, in which multiple neurons are connected. Linear function appeared to solve this problem (in situations with more than one firing neuron, max or softmax could be used to solve the problem, but it has a fixed

Table 2
Tensorflow loss functions.

Softmax cross entropy with logits
Sigmoid cross entropy with logits
Softmax cross entropy with logits v2
Weighted cross entropy with logits

Table 3
Tensorflow optimizers.

Gradient Descent Optimizer
Proximal Gradient Descent Optimizer
Momentum Optimizer
Adagrad Optimizer
Proximal Adagrad Optimizer
Adadelta Optimizer
Adam Optimizer
RMS Prop Optimizer
Ftrl Optimizer

gradient. In addition, considering a DNN with linear activation function, then output of each layer would be the input of the next layer, each firing is based on another linear firing. So, it could be assumed that only one linear firing exists in the whole NN. Sigmoid function looks like a smooth and step function, and solved problems of both of previous functions. It has limited analog output, smooth gradient, and nonlinear combinations. However, the input in the range $(-2, 2)$ causes observable change in the output, and the gradient would be small out of this range. Tanh function is a scaled sigmoid function with stronger gradients than sigmoid, and solved the problem somehow.

The Relu is a nonlinear function in range $[0, \infty)$ that can be approximated with its combinations. It is a sparse activation function, which only fires a few neurons (almost 50%); it solved the problem of computational complexity of previous functions. However, it has zero gradient for negative inputs, which causes the so-called dying Relu problem, in which the neurons cannot response to the changes. Exponential Linear Unit (Elu) solved this problem by adding a small slope for negative inputs (the slope is defined by a positive constant); furthered, Selu function extended Elu by replacing the linear slope by twisted slope (and an additional constant). These constants are related to the input, e.g., for standard scaled inputs (mean 0, var. 1), the values are $\alpha = 1.6732, \lambda = 1.0507$. Concatenated Relu (Crelu) extended Relu by doing the same in the negative direction.

Softmax function calculates probability distribution function of each target class over all possible target classes. Softsign function is an alternative to hyperbolic tangent. Even though tanh and softsign functions are closely related, tanh converges exponentially whereas softsign converges polynomially. Softplus is an alternative of traditional functions, because it is differentiable and its derivative is easy to demonstrate. Sigmoid and tanh outputs have upper and lower limits whereas softplus outputs are in range $(0, \infty)$.

4.6. Selecting loss function

After designing the DNN structure and before running it, an appropriate loss function should be found. Table 2 shows the Tensorflow loss functions. The same as before, this task is empirical and depends on the input data. For example in M-ary classification, softmax cross entropy, and in binary classification, sigmoid cross entropy are good choices. If the input structure is sparse, then sparse softmax cross entropy would be preferable, and in situations which one of the classes has a higher weight, weighed cross entropy, which is an extension of sigmoid cross entropy is preferred. MMSE, and cross entropy, are both the famous loss functions; however, Cross entropy is mostly used in literature. It measures the distance between actual class and predicted value, which is usually a real number between 0 and 1.

Sigmoid cross entropy loss function is very similar to cross entropy loss function, except that it uses a sigmoid activation function at the last layer. Weighted cross entropy loss function is a weighted version of the sigmoid cross entropy loss function, which provides a weight on the positive target. Softmax cross entropy loss function measures the probability distribution functions by applying a softmax activation function at the last layer. Sparse softmax cross entropy loss function is the same as softmax cross entropy loss function, except that instead of being the target a probability distribution, it is an index of which category is true. Instead of a sparse all-zero target vector with one value of one, it just passes in the index of which category is the true value.

4.7. Selecting optimizer

In order to find the best DNN hyperparameters, the selected loss function should be minimized; this is done by iterative optimization algorithms. Optimization is a tricky subject, which again depends on the input quality and quantity, model size, and the contents of the weight matrices; again trial and error is the way to determine the best optimizer. Table 3 shows the Tensorflow optimizers. The iterative Gradient Descent based formulation for updating DNN parameters is $\theta = \theta - \eta \nabla J(\theta)$, where η is the learning rate, $J(\theta)$ is loss function, $\nabla J(\theta)$ is the gradient of loss function w.r.t parameters θ. The most popular algorithm for optimizing DNN is Stochastic Gradient Descent (SGD) [54]. However, SGD has high variance oscillations and could not converge properly. This problem could be solved by addition of a momentum term [55], which navigates SGD along the relevant direction and softens the oscillations in irrelevant directions. In the momentum, the updating function changes to $V(t) = \gamma V(t-1) + \eta \nabla J(\theta)$, and $\theta = \theta - V(t)$, where γ is the momentum term (usually set to 0.9). The momentum term γ increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.

The momentum is high while reaching the minima (it does not slow down at that point), so, it passes the minima. Nesterov accelerated gradient could solve this problem and prevents going too fast and missing the minima. It takes a big jump according to the previous momentum, then calculates the gradient, makes a correction, and finally updates the parameters. Computing $\theta - \gamma V(t-1)$ gives an approximation of the next position of the parameters. Calculating the gradient w.r.t. the approximate future position of parameter, i.e., $V(t) = \gamma V(t-1) + \eta \nabla J(\theta - \gamma V(t-1))$ gives a look ahead, and finally could update the parameters using $\theta = \theta - V(t)$.

Although this method speeded up the updating, it would be better to apply larger or smaller updates for each individual parameter based on its importance. Adagrad solved this problem by making big and short updates for infrequent and frequent parameters, respectively. Considering $g(t, i)$ to be the loss function gradient w.r.t. to the parameter $\theta(i)$ at time step t, the updating formula becomes $\theta(t+1, i) = \theta(t, i) - \eta g(t, i)/\sqrt{G(t, ii) + \epsilon}$. Actually, it modifies the learning rate at each time step t for every parameter $\theta(i)$ based on the past gradients computed for $\theta(i)$. Adagrad does not require to know learning rate (a default value of 0.01 would be sufficient); however, its learning rate is always decreasing and decaying. Adadelta [54] solved this problem by calculating the momentum; it limits the accumulated past gradients to a window with size w. The running average $E[g^2](t)$ at time step t then depends only on the previous average and the current gradient. So, the updating formula changes to $E[g^2](t) = \gamma E[g^2](t-1) + (1-\gamma) g^2(t)$, $\theta(t+1) = \theta(t) - \eta g(t, i)$.

Adam [55] extended the Adadelta by calculating momentums for each parameter. In addition to storing an exponentially decaying average of the past squared gradients like Adadelta, Adam also keeps an exponentially decaying average of past gradients$M(t)$, similar to momentum. The formulas for the first moment (mean) and the second moment (the variance) of the Gradients are $\hat{m}(t) = m(t)/(-\beta_1(t))$, and $\hat{v}(t) = v(t)/(-\beta_2(t))$, where $m(t)$ and $v(t)$ are values of the first and second moment, respectively. The updating formula changes to $\theta(t+1) = \theta(t) - \eta/(\sqrt{\hat{v}(t)} + \epsilon) \times \hat{m}(t)$. RMS Prop is similar to Adam it just uses different moving averages but has the same goals. Ftrl Proximal was developed for ad-click prediction where they had billions of dimensions and hence huge matrices of weights that were very sparse. The main feature here is to keep near zero weights at zero, so calculations can be skipped and optimized.

4.8. Selecting learning rate, and number of iterations

The last step in tuning hyperparameters is selecting the learning rate, as well as number of iterations, which is very important. If the loss is oscillating around a point at the start of training, the learning rate is chosen high. If the loss is decreasing consistently but very slowly, increasing the learning rate is a good idea. Low learning rates not only slow down training but also can degrade the performance of the trained model. High learning rates increase generalization ability, as well as the noise on the stochastic gradient, which acts as an implicit regularizer. Learning rates can take a wide range of values, so gradually adjusting is time-consuming. In addition, results of using learning rates of 0.001 and 0.0011 are not very different. So, different learning rates should be used to determine the exploring range of learning rates. After finding the optimal range of learning rates, search in smaller grids around the optimal range. Before determining number of iterations, it is required to specify the acceptable error tolerance of the trained model. The iterations could be done as much as either reaching a threshold, or failing to make additional progress. In the latter case, ought to adjust hidden layers, consider alternative algorithms, treat data beforehand, or use DL methods.

5. Proposed hyperparameter tuning methods

Among hyperparameter tuning methods, grid search is the most popular, because its results are more accurate and trustable. However, it is highly dependent on the grid size and grid range. DNN has much more hyperparameters than the other ML algorithms; so, the use of grid search for DNN hyperparameter tuning would take a long time. DNN has at least 9 hyperparameters (see Section 4), which each of them should get at least 9 points in the grid (because a grid should be wide enough). So, in order to tune hyperparameters of a DNN by grid search method it is required to run $9! = 362880$ times the cross validation and see the results, and then decide. Marginally and alternatively searching the previous scenario would require $9 \times 9 = 81$ computations. Simply developing the grid search marginally and alternatively rather than jointly might better deserve the tradeoff between performance and complexity. How much would be degradation of so much computation reduction? The answer to this question is completely data dependent, however, the results of paper prove that in OC applications, there is slight difference between performances of different hyperparameter sets, and there is no need to develop such a huge investigations. In this section, two novel suboptimal (marginally and alternatively) grid search algorithms are presented.

Table 4
The initial hyperparameter set of both methods for both OC systems.

Hyperparameter	Value
Modulation order	16
Number of layers	2
Number of hidden neurons	32
Activation function	Selu
Sample Size/ Batch Size	8
Batch Size	2^{17}
Learning Rate	0.001
Iterations	250
Loss Function	Softmax cross entropy
Optimizer	Adam

Table 5
FSO channel parameters.

α	4.2
β	1.4
$Es/N0[dB]$	0

Table 6
Fiber OC channel.

C [m/s]	299792458
h	$6.6261e{-}34$
D [ps/nm/km]	16.4640
β_2 [ps2/km]	21
f_c [Hz]	$1.9341e{+}14$
γ [1/W/km]	1.3
α [dB/km]	0.2
Number of spans	20
Span length [km]	100
Pre-dispersion [ps^2]	0
P_0 [dBm]	2
Baud-rate [GHz]	32
Channel spacing [GHz]	50
Second order modulation factor	1.32
Third order modulation factor	1.96
EDFA Noise figure [dB]	5

5.1. First method (marginal grid search)

In the first method, first the size and the range of the grid (the hyperparameters and their values) should be defined. Then, (based on previous knowledge from literature) an initial hyperparameter set should be selected for the starting point. Considering this initial set, one of the hyperparameters would be tuned over its defined grid range. Again, considering the initial set, another hyperparameter would be tuned over its defined grid range. This procedure will continue until tuning all of the hyperparameters. This would be the first round, the initial point of the second round is the hyperparameter set of the first round that has the least Symbol Error Rate (SER) performance. The same procedure would be continued in at all of the following rounds until reaching a desired convergence.

Algorithm 1: First method

1: Input data {the transmitted and received vectors}
2: Define grid search range and size {Table 7}
3: Choose an initial hyperparameter set
4: **While** reaching a convergence **do**
5: **For** each of the grid points **do**
6: Calculate SER
7: Choose the grid point with the lowest SER
8: update initial hyperparameter set

Table 7
Results of using proposed methods in FSO and fiber OC systems.

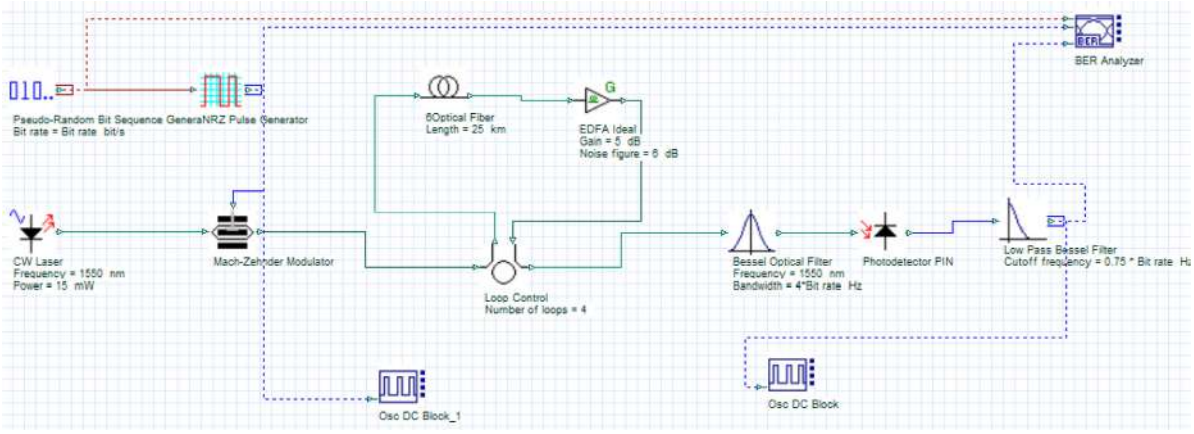
Learning rate	**0.00005**	**0.0001**	**0.0005**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**	**0.5**
method 1-FSO SER	0.785	0.773	0.770	0.768	0.768	0.769	0.799	0.774	0.9369
method 2-FSO SER	0.785	0.773	0.770	0.768	0.768	0.769	0.769	0.774	0.9369
method 1-Fiber SER	0.634	0.137	0.0346	0.0325	0.029	0.0288	0.0295	0.0302	0.0356
method 2-Fiber SER	0.191	0.0460	0.0302	0.0288	0.0294	0.0394	0.0300	0.0292	0.0303
# of iteration	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**
method 1-FSO SER	0.766	0.767	0.767	0.762	0.767	0.760	0.764	0.763	0.769
method 2-FSO SER	0.765	0.767	0.767	0.761	0.767	0.760	0.764	0.762	0.769
method 1-Fiber SER	0.0339	0.0323	0.0317	0.0293	0.0292	0.0265	0.0263	0.0279	0.0309
method 2-Fiber SER	0.0300	0.0289	0.0291	0.0291	0.0350	0.0269	0.0283	0.0292	0.0311
# of Layer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
method 1-FSO SER	0.768	0.768	0.769	0.768	0.799	0.768	0.766	0.767	0.766
method 2-FSO SER	0.762	0.760	0.759	0.760	0.761	0.758	0.759	0.761	0.759
method 1-Fiber SER	0.0373	0.0325	0.0294	0.029	0.029	0.029	0.0290	0.0293	0.0294
method 2-Fiber SER	0.0260	0.0269	0.0275	0.0295	0.0279	0.679	0.809	0.932	0.939
# of Neuron	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
method 1-FSO SER	0.770	0.767	0.768	0.768	0.767	0.769	0.767	0.768	0.767
method 2-FSO SER	0.759	0.761	0.759	0.760	0.761	0.761	0.759	0.761	0.761
method 1-Fiber SER	0.0348	0.0351	0.0328	0.0322	0.0310	0.0305	0.0302	0.0302	0.0303
method 2-Fiber SER	0.0261	0.0257	0.0352	0.0265	0.0263	0.0272	0.0268	0.0267	0.0269
Activation function	**Relu**	**Crelu**	**Elu**	**Selu**	**Relu6**	**Tanh**	**Softmax**	**Softsign**	**Softplus**
method 1-FSO SER	0.769	0.767	0.769	0.768	0.769	0.790	0.782	0.769	0.768
method 2-FSO SER	0.759	0.799	0.759	0.759	0.759	0.759	0.966	0.758	0.760
method 1-Fiber SER	0.0394	0.0306	0.0333	0.0325	0.0319	0.0327	0.848	0.0359	0.0330
method 2-Fiber SER	0.0268	0.0272	0.0291	0.0261	0.0263	0.0265	0.0272	0.0261	0.0261
Optimizer	**Adam**	**Adadelta**	**Adagrad**	**Ftrl**	**Gradient Descent**	**Proximal Adagrad**	**Proximal Gradient Descent**	**RMS Prop**	**Momentum**
method 1-FSO SER	0.768	0.922	0.807	0.936	0.818	0.807	0.818	0.769	0.777
method 2-FSO SER	0.759	0.877	0.765	0.936	0.775	0.765	0.775	0.765	0.763
method 1-Fiber SER	0.0325	0.906	0.874	0.938	0.899	0.874	0.899	0.0297	0.629
method 2-Fiber SER	0.0261	0.0678	0.0261	0.0264	0.0267	0.0261	0.0267	0.0312	0.0263
Sample size/ Batch size	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
method 1-FSO SER	0.775	0.764	0.765	0.765	0.799	0.769	0.769	0.768	0.763
method 2-FSO SER	0.759	0.767	0.770	0.766	0.765	0.766	0.766	0.759	0.766
method 1-Fiber SER	0.0719	0.0454	0.0354	0.0335	0.0353	0.0368	0.0347	0.035	0.0349
method 2-Fiber SER	0.0282	0.0298	0.0297	0.0278	0.0287	0.0288	0.0295	0.0261	0.0285
Batch size	**4*16**	**8*16**	**16*16**	**32*16**	**64*16**	**128*16**	**256*16**	**512*16**	**1024*16**
method 1-FSO SER	0.781	0.757	0.723	0.759	0.786	0.775	0.771	0.763	0.768
method 2-FSO SER	0.774	0.789	0.773	0.771	0.769	0.780	0.778	0.758	0.759
method 1-Fiber SER	0.0312	0.0234	0.0429	0.0410	0.0312	0.0327	0.0419	0.0363	0.0351
method 2-Fiber SER	0.0312	0.0264	0.0507	0.0312	0.0312	0.0410	0.0288	0.0313	**0.0261**
Loss function	**Softmax cross entropy**	**Softmax cross entropy v2**	**Sigmoid cross entropy**	**Weighted cross entropy**					
method 1-FSO SER	0.768	0.768	0.769	0.770					
method 2-FSO SER	0.773	0.773	0.769	0.781					
method 1-Fiber SER	0.0288	0.0288	0.0355	0.0351					
method 2-Fiber SER	0.0261	0.0261	0.0496	0.0364					

5.2. Second method (alternative grid search)

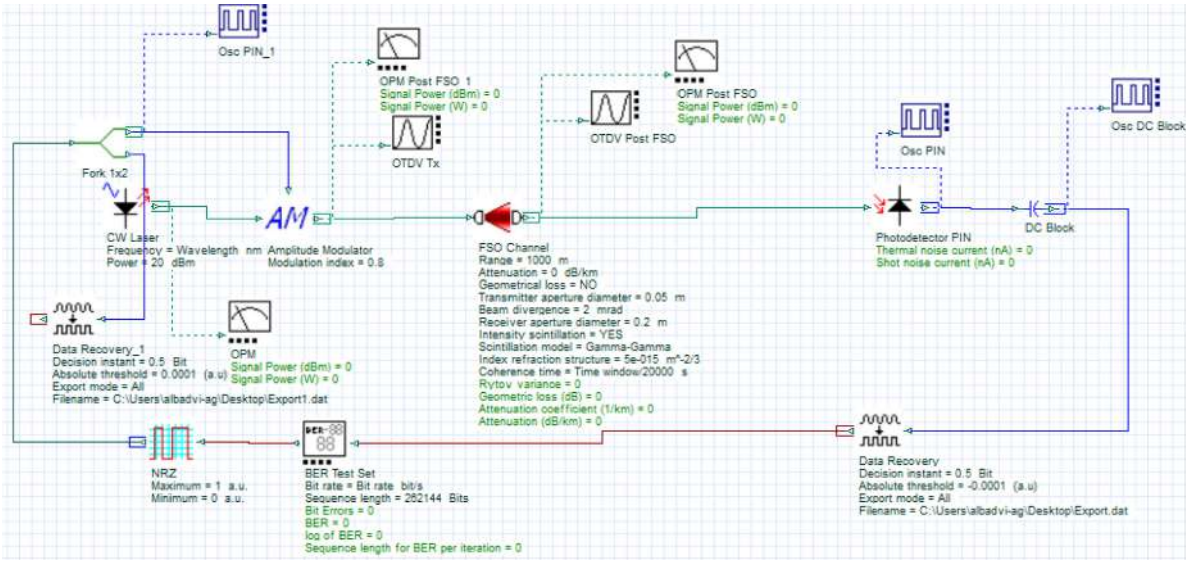
In the second method, the idea of alternating optimization (which is conducted for multivariate iterative optimization properly) is used in hyperparameter tuning. First the size and range of the grid (hyperparameters and their values) should be defined. Then, (based on previous knowledge from literature) an initial hyperparameter set should be defined. Considering this initial point, one of the hyperparameters would be tuned over its defined grid range. Then, the tuned point would be replaced (updated) in the initial set. Then, the next hyperparameter would be tuned over its grid range and would be replaced (updated) in the initial set. This process continues until all of the hyperparameters would be tuned and would be replaced (updated) in the initial set. This would be the first round, the initial point of the second round is the updated hyperparameter set. The same procedure would be continued in at all of the following rounds until reaching a desired convergence.

Algorithm 2: Second method

```
1: Input data {the transmitted and received vectors}
2: Define hyperparameters and grid search range {Table 7}
3: Choose an initial hyperparameter set
4: While reaching a convergence do
5:     For each hyperparameter do
6:         For each range point do
7:             Calculate SER
8:         Choose the grid point with the lowest SER
9:         update the initial hyperparameter set
```



(a)



(b)

Fig. 3. Optisystem setups for a. FSO and b. Fiber OC systems.

Table 8
FSO channel parameters.

Link range [km]	1
Attenuation [dB/km]	0
Transmitter aperture diameter [m]	0.05
Beam divergence [mrad]	2
Receiver aperture diameter [m]	0.2
Scintillation model	Gamma–Gamma
Wavelength [nm]	1550
Index refraction structure [$m^{-\frac{2}{3}}$]	$5etmin15$
Transmitter power [dBm]	20
Bit rate [Gbps]	10

Table 9
Fiber OC channel parameters.

C[m/s]	299792458
h	$6.6261e{-}34$
β_2[ps^2/km]	-20
$f_c[Hz]$	$1.9341e{+}14$
γ [1/W/km]	1.3
α [dB/km]	0.2
Length [km]	25
P_0 [mW]	15
Bit rate [Gbps]	10
EDFA gain [dB]	5
EDFA Noise figure [dB]	6
PMD coefficient [ps/km$^{1/2}$]	0.5

6. Results and discussions

In this section, the results of applying the proposed methods for hyperparameter tuning of a DNN are discussed. The DNN based simulations are developed in Tensorflow environment, because it is very helpful for DNN. Table 4 shows the hyperparameter set, which is selected based on previous knowledge from literature. Four datasets are collected as the DNN input; two datasets collected by simulating the FSO and fiber OC channels in MATLAB software. However, in order to get closer to the actual results, Optisystem software is also used for collecting two other datasets on FSO and fiber OC channels. Because Optisystem considers more parameters and is closer to the experimental results (comparing with simple MATLAB simulations). The first

Table 10
Results of using proposed methods in FSO and fiber optic communication systems on the experimental setups.

Learning rate	**0.00005**	**0.0001**	**0.0005**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**	**0.5**
method 1-FSO SER	0.495	0.491	0.483	0.482	0.492	0.483	**0.479**	0.482	0.495
method 2-FSO SER	0.482	0.481	0.483	0.482	0.482	0.483	**0.479**	0.482	0.495
method 1-Fiber SER	0.500	0.499	0.495	0.495	0.497	0.497	0.503	0.497	0.502
method 2-Fiber SER	0.500	0.499	0.495	0.495	0.497	0.497	0.503	0.497	0.502
textbf# of iteration	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**
method 1-FSO SER	0.495	0.480	0.485	0.491	0.481	0.499	0.483	0.482	0.480
method 2-FSO SER	0.498	0.482	0.482	0.480	0.484	0.480	0.484	0.483	0.489
method 1-Fiber SER	0.512	0.496	0.501	0.497	0.495	0.494	0.495	0.520	0.498
method 2-Fiber SER	0.514	0.497	0.499	0.500	0.495	0.496	0.496	0.496	0.500
# of Layer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
method 1-FSO SER	0.489	0.492	0.482	0.483	0.499	0.483	0.492	0.482	0.481
method 2-FSO SER	0.480	0.480	0.499	0.484	0.493	0.487	0.492	0.480	0.495
method 1-Fiber SER	0.520	0.495	0.500	0.497	0.495	0.498	0.530	0.497	0.495
method 2-Fiber SER	0.530	0.495	0.521	0.495	0.493	0.500	0.492	0.499	0.495
# of Neuron	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
method 1-FSO SER	0.489	0.486	0.498	0.482	0.481	0.485	0.483	0.483	0.483
method 2-FSO SER	0.481	0.481	0.481	0.497	0.481	0.481	0.481	0.481	0.481
method 1-Fiber SER	0.495	0.495	0.496	0.494	0.495	0.495	0.500	0.521	0.494
method 2-Fiber SER	0.496	0.493	0.492	0.498	0.513	0.521	0.530	0.495	0.497
Activation function	**Relu**	**Crelu**	**Elu**	**Selu**	**Relu6**	**Tanh**	**Softmax**	**Softsign**	**Softplus**
method 1-FSO SER	0.493	0.482	0.499	0.482	0.483	0.482	0.482	0.489	0.482
method 2-FSO SER	0.480	0.481	0.481	0.481	0.490	0.481	0.498	0.481	0.489
method 1-Fiber SER	0.496	0.496	0.494	0.495	0.521	0.495	0.513	0.494	0.5023
method 2-Fiber SER	0.497	0.497	0.495	0.492	0.502	0.495	0.497	0.496	0.520
Optimizer	**Adam**	**Adadelta**	**Adagrad**	**Ftrl**	**Gradient Descent**	**Proximal Adagrad**	**Proximal Gradient Descent**	**RMS Prop**	**Momentum**
method 1-FSO SER	0.482	0.517	0.482	0.495	0.482	0.482	0.492	0.500	0.482
method 2-FSO SER	0.480	0.480	0.480	0.500	0.481	0.496	0.479	0.478	0.480
method 1-Fiber SER	0.495	0.498	0.522	0.497	0.503	0.503	0.523	0.497	0.497
method 2-Fiber SER	0.492	0.516	0.497	0.497	0.504	0.516	0.497	0.492	0.497
Sample size/Batch size	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
method 1-FSO SER	0.502	0.496	0.486	0.484	0.497	0.481	0.485	0.482	0.486
method 2-FSO SER	0.481	0.482	0.491	0.487	0.500	0.485	0.492	0.478	0.496
method 1-Fiber SER	0.495	0.501	0.500	0.536	0.494	0.501	0.495	0.509	0.497
method 2-Fiber SER	0.495	0.532	0.499	0.492	0.500	0.496	0.492	0.495	0.498
Batch size	**4*16**	**8*16**	**16*16**	**32*16**	**64*16**	**128*16**	**256*16**	**512*16**	**1024*16**
method 1-FSO SER	0.480	0.500	0.535	0.494	0.483	0.486	0.475	0.485	0.482
method 2-FSO SER	0.486	0.523	**0**.476	0.505	0.473	0.473	0.492	0.480	0.480
method 1-Fiber SER	0.491	0.494	0.493	0.496	0.483	0.500	0.512	0.496	0.495
method 2-Fiber SER	0.499	0.478	0.488	0.469	0.496	0.461	0.465	0.482	0.443
Loss function	**Softmax cross entropy**	**Softmax cross entropy v2**	**Sigmoid cross entropy**	**Weighted cross entropy**					
method 1-FSO SER	0.482	0.482	0.483	0.482					
method 2-FSO SER	0.473	0.486	0.490	0.476					
method 1-Fiber SER	0.495	0.495	0.494	0.494					
method 2-Fiber SER	0.443	0.443	0.443	0.443					

subsection discusses results of the first two datasets, the second subsection discusses results of the other datasets. The third subsection is dedicated to comparison between the results of the proposed methods, [24], and [15] (based on the first two datasets).

6.1. MATLAB Simulations

Table 5, and Table 6 show FSO and fiber OC channel parameters, respectively. These parameters were considered for simulating the OC systems using MALAB software. Table 7 shows obtained results for hyperparameter tuning of the proposed methods for both of FSO and fiber OC systems. FSO link is assumed in strong regime of Gamma–Gamma atmospheric turbulence ($\alpha = 4.2, \beta = 1.4$), and $Es/N0 = 0$ dB. The fiber link is assumed to have dispersion, path loss, and nonlinearity. Results indicate that proposed methods are dependent on the input data, and have different results for different datasets. This shows the importance of the prior knowledge from previous literature. For the same application (detection), different datasets (fiber and FSO) have different DNN structures. In Table 7, appropriate range for the tuned hyperparameters are bolded and underlined. In the first method, the hyperparameter set with the least SER would be selected; in the second method, the hyperparameter set (of the last row) with the least SER would be selected. As seen, for FSO, the first method results in SER of 0.723, while the second method achieves 0.760. For fiber OC, the first method achieves SER of 0.0234, while the second method achieves 0.0261. For simplicity and without loss of generality, only one round results are presented, (because for showing the results of each iteration, a one page length table should be added, and the aim of this paper is not to find the optimum point, it aims to compare the results of the proposed methods, and their difference is obvious even at the first step).

Table 11
Results of using grid search of [24] in FSO and fiber OC systems.

Learning rate	**0.00005**	**0.0005**	**0.005**	**0.05**	**0.5**
FSO SER	0.785	0.770	0.768	0.769	0.937
Fiber SER	0.634	0.035	0.029	0.030	0.039
# of iteration	**100**	**300**	**500**	**700**	**900**
FSO SER	0.776	0.777	0.787	0.774	0.769
Fiber SER	0.034	0.039	0.029	0.029	0.041
# of Layer	**1**	**3**	**5**	**7**	**8**
FSO SER	0.770	0.769	0.769	0.776	0.777
Fiber SER	0.037	0.029	0.029	0.029	0.032
# of Neuron	**10**	**30**	**50**	**70**	**90**
FSO SER	0.770	0.768	0.767	0.797	0.767
Fiber SER	0.035	0.033	0.041	0.030	0.030
Sample size/ Batch size	**1**	**3**	**5**	**7**	**9**
FSO SER	0.775	0.795	0.769	0.769	0.773
Fiber SER	0.072	0.036	0.045	0.035	0.035
Batch size	**4*16**	**64*16**	**128*16**	**256*16**	**1024*16**
method 1-FSO SER	0.781	0.796	0.775	0.771	0.768
method 1-Fiber SER	0.031	0.032	0.033	0.042	0.035

Table 12
Results of using manual search of [15] in FSO and fiber OC systems.

Learning rate	**0.00005**	**0.0005**	**0.005**	**0.05**	**0.1**
FSO SER	0.785	0.770	0.768	0.769	0.774
Fiber SER	0.634	0.035	0.029	0.040	0.030
# of iteration	**200**	**400**	**500**	**700**	**900**
FSO SER	0.767	0.768	0.787	0.774	0.769
Fiber SER	0.032	0.039	0.029	0.029	0.031
# of Layer	**1**	**5**	**6**	**7**	**8**
FSO SER	0.768	0.779	0.768	0.776	0.767
Fiber SER	0.037	0.039	0.029	0.029	0.029
# of Neuron	**10**	**20**	**50**	**60**	**90**
FSO SER	0.770	0.767	0.767	0.789	0.767
Fiber SER	0.035	0.045	0.031	0.030	0.030
Sample size/Batch size	**1**	**4**	**6**	**7**	**8**
FSO SER	0.775	0.765	0.799	0.789	0.768
Fiber SER	0.072	0.034	0.037	0.045	0.035
Batch size	**4*16**	**32*16**	**64*16**	**128*16**	**256*16**
FSO SER	0.781	0.769	0.796	0.775	0.771
Fiber SER	0.031	0.041	0.031	0.033	0.042

6.2. Optisystem simulations

In this section, Optisystem software is used for collecting datasets for FSO and fiber OC systems. The reason is that Optisystem considers more things than MATLAB simulations and its results are closer to the experimental results. The implemented setups for FSO and fiber OC systems are shown in Fig. 3. a and b, respectively. The details of their channel models are shown in Tables 8 and 9. In order to show the generality of the proposed methods, a bit different channel parameters are considered in experimental setups, but the system models are the same. Table 10 shows the results of implementation of the proposed methods on the defined setups. As seen, for FSO, the first method results in SER of 0.479, while the second method achieves 0.473. For fiber OC, the first method achieves SER of 0.494, while the second method achieves 0.443. As seen, in the first two datasets, first method performs better than the second method, however, for the two second datasets this is inverse. This is because both of the proposed methods are suboptimal, and they are dependent on the initial point. However, both methods have close performance and are efficient for different OC applications.

6.3. Comparison

Tables 11 and 12 are added for better comparison between the proposed methods and grid search of [24] as well as manual search of [15] on the first two datasets. As mentioned in Section 1.1, grid search method used in [24] and manual search used in [15] had small size and small range. As seen, for FSO, SER of the proposed methods, [24], and [15] are 0.723, 0.760 0.768, 0.767, respectively. For fiber OC, SER of the proposed methods, [24], and [15] are 0.0234, 0.0261, 0.029, 0.029, respectively. As seen, the proposed methods perform better, because they considered larger size and range in the grid search.

7. Conclusion

The first important step towards deploying a ML algorithm is tuning its hyperparameters. There are many ways to this end, among them grid search is the most popular method. However, it has high computational complexity, and is not appropriate for ML algorithms with many hyperparameters. Accordingly, this paper presented two novel suboptimal grid search methods, which search the grid marginally and alternatively. These methods are

investigated on a DNN under four datasets. Two datasets were collected by simulating FSO and fiber OC links by MATLAB software, and two other datasets were collected by experimental setups for FSO and fiber OC links in Optisystem software. Results indicated that despite greatly reducing computation load, favorable performance could be achieved by the proposed methods. The proposed structures were compared with some of the recently published most relevant works, and the efficiency of the proposed methods was indicated.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

M.A. Amirabadi: Conceptualization, Writing - original draft. **M.H. Kahaei:** Supervision. **S.A. Nezamalhosseini:** Supervision.

References

[1] M.A. Khalighi, M. Uysal, Survey on free space optical communication: A communication theory perspective, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2231–2258.

[2] X. Zhou, X. Zheng, R. Zhang, L. Hanzo, Chip-interleaved optical code division multiple access relying on a photon-counting iterative successive interference canceller for free-space optical channels, Opt. Exp. 21 (13) (2013) 15926–15937.

[3] A.K. Majumdar, Advanced Free Space Optics (FSO): A Systems Approach, Vol. 186, Springer, 2014.

[4] M. Uysal, C. Capsoni, Z. Ghassemlooy, A. Boucouvalas, E. Udvary (Eds.), Optical Wireless Communications: An Emerging Technology, Springer, 2016.

[5] C.M. Bishop, Pattern Recognition and Machine Learning, springer, 2006.

[6] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, 2016.

[7] M.T. Hagan, H.B. Demuth, M.H. Beale, O. De Jesús, Neural Network Design, Vol. 20, Pws Pub., Boston, 1996.

[8] R.T. Jones, T.A. Eriksson, M.P. Yankov, D. Zibar, Deep learning of geometric constellation shaping including fiber nonlinearities, in: 2018 European Conference on Optical Communication (ECOC), IEEE, 2018, pp. 1–3.

[9] C. Wang, S. Fu, Z. Xiao, M. Tang, D. Liu, Long short-term memory neural network (LSTM-NN) enabled accurate optical signal-to-noise ratio (OSNR) monitoring, J. Lightwave Technol. (00) (2019).

[10] F.N. Khan, K. Zhong, W.H. Al-Arashi, C. Yu, C. Lu, A.P.T. Lau, Modulation format identification in coherent receivers using deep machine learning, IEEE Photonics Technol. Lett. 28 (17) (2016) 1886–1889.

[11] F.N. Khan, K. Zhong, X. Zhou, W.H. Al-Arashi, C. Yu, C. Lu, A.P.T. Lau, Joint OSNR monitoring and modulation format identification in digital coherent receivers using deep neural networks, Opt. Exp. 25 (15) (2017) 17767–17776.

[12] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.

[13] N. Schilling, M. Wistuba, L. Drumond, L. Schmidt-Thieme, Hyperparameter optimization with factorized multilayer perceptrons, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, Cham, 2015, pp. 87–103.

[14] F. Hutter, J. Lücke, L. Schmidt-Thieme, Beyond manual tuning of hyperparameters, DISKI 29 (4) (2015) 329–337.

[15] M.W.S. Chang, PianoNet: A Hyperparameter Study of a Deep Neural Network for Automatic Music Transcription.

[16] F. Friedrichs, C. Igel, Evolutionary tuning of multiple SVM parameters, Neurocomputing 64 (2005) 107–117.

[17] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research, 13(Feb) 28 (2012) 1–305.

[18] R.G. Mantovani, A.L. Rossi, J. Vanschoren, B. Bischl, A.C. De Carvalho, Effectiveness of random search in SVM hyper-parameter tuning, in: 2015 International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1–8.

[19] L. Li, A. Talwalkar, Random search and reproducibility for neural architecture search, 2019, arXiv preprint arXiv:1902.07638.

[20] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in Neural Information Processing Systems, 2012, pp. 2951–2959.

[21] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In NIPS workshop on Bayesian Optimization in Theory and Practice (Vol. 10, 3), 2013.

[22] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, in: Proceedings of the 24th International Conference on Machine Learning, ACM, 2007, pp. 473–480.

[23] D. Choi, H. Cho, W. Rhee, On the difficulty of dnn hyperparameter optimization using learning curve prediction, in: TENCON 2018-2018 IEEE Region 10 Conference, IEEE, 2018, pp. 0651–0656.

[24] Y. Zhou, S. Cahya, S.A. Combs, C.A. Nicolaou, J. Wang, P.V. Desai, J. Shen, Exploring tunable hyperparameters for deep neural networks with industrial ADME data sets, J. Chem. Inf. Model. 59 (3) (2018) 1005–1016.

[25] K. Wang, C. Shang, F. Yang, Y. Jiang, D. Huang, Automatic hyper-parameter tuning for soft sensor modeling based on dynamic deep neural network, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2017, pp. 989–994.

[26] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, et al., Scalable bayesian optimization using deep neural networks, in: International conference on machine learning, 2015, pp. 2171-2180.

[27] C. Rasmussen, C. Williams, Gaussian Processes for Machine Learning, The MIT Press, 2006.

[28] I. Ilievski, T. Akhtar, J. Feng, C.A. Shoemaker, Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[29] I. Ilievski, T. Akhtar, J. Feng, C.A. Shoemaker, Hyperparameter optimization of deep neural networks using non-probabilistic RBF surrogate model, 2016, arXiv preprint arXiv:1607.08316.

[30] I. Loshchilov, F. Hutter, CMA-ES For hyperparameter optimization of deep neural networks, 2016, arXiv preprint arXiv:1604.07269.

[31] O.E. David, I. Greental, Genetic algorithms for evolving deep neural networks, in: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 1451–1452.

[32] F. Hutter, H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proc. of LION'11, 2011, pp. 507–523.

[33] J. Bergstra, D. Yamins, D.D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, 2013.

[34] T. Domhan, J.T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[35] J. Luketina, M. Berglund, K. Greff, T. Raiko, Scalable gradient-based tuning of continuous regularization hyperparameters. In International Conference on Machine Learning, 2016, pp. 2952-2960.

[36] P.K. Sharma, A. Bansal, P. Garg, T. Tsiftsis, R. Barrios, Relayed FSO communication with aperture averaging receivers and misalignment errors, IET Commun. 11 (1) (2017) 45–52.

[37] P.V. Trinh, T.C. Thang, A.T. Pham, Mixed mmWave RF/FSO relaying systems over generalized fading channels with pointing errors, IEEE Photonics J. 9 (1) (2017) 1–14.

[38] F.J. Lopez-Martinez, G. Gomez, J.M. Garrido-Balsells, Physical-layer security in free-space optical communications, IEEE Photonics J. 7 (2) (2015) 1–14.

[39] M.L.B. Riediger, R. Schober, L. Lampe, Blind detection of on-off keying for free-space optical communications, in: 2008 Canadian Conference on Electrical and Computer Engineering, IEEE, 2008, pp. 001361–001364.

[40] Y.J. Zhu, Z.G. Sun, J.K. Zhang, Y.Y. Zhang, A fast blind detection algorithm for outdoor visible light communications, IEEE Photonics J. 7 (6) (2015) 1–8.

[41] M.R. Bhatnagar, Z. Ghassemlooy, Performance analysis of gamma–gamma fading FSO MIMO links with pointing errors, J. Lightwave Technol. 34 (9) (2016) 2158–2169.

[42] P. Poggiolini, The GN model of non-linear propagation in uncompensated coherent optical systems, J. Lightwave Technol. 30 (24) (2012) 3857–3879.

[43] R. Dar, M. Feder, A. Mecozzi, M. Shtaif, Properties of nonlinear noise in long, dispersion-uncompensated fiber links, Opt. Express 21 (22) (2013) 25685–25699.

[44] R. Dar, M. Feder, A. Mecozzi, M. Shtaif, Accumulation of nonlinear interference noise in fiber-optic systems, Opt. Exp. 22 (12) (2014) 14199–14211.

[45] R. Jones, Source code, 2018, https://github.com/rassibassi/claude.

[46] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Physica-Verlag HD, 2010, pp. 177–186.

[47] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning, PMLR, vol. 37, no. 2015, 2015, pp. 448-456.

[48] H.C. Leung, C.S. Leung, E.W. Wong, S. Li, Extreme learning machine for estimating blocking probability of bufferless OBS/OPS networks, IEEE/OSA J. Opt. Commun. Networking 9 (8) (2017) 682–692.
[49] S.T. Ahmad, K.P. Kumar, Radial basis function neural network nonlinear equalizer for 16-QAM coherent optical OFDM, IEEE Photonics Technol. Lett. 28 (22) (2016) 2507–2510.
[50] T.F. de Sousa, M.A. Fernandes, Multilayer perceptron equalizer for optical communication systems, in: 2013 SBMO/IEEE MTT-S International Microwave & Optoelectronics Conference (IMOC), IEEE, 2013, pp. 1–5.
[51] D. Wang, M. Zhang, J. Li, Z. Li, J. Li, C. Song, X. Chen, Intelligent constellation diagram analyzer using convolutional neural network-based deep learning, Opt. Exp. 25 (15) (2017) 17150–17166.
[52] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, S. Khudanpur, Recurrent neural network based language model, in: Eleventh annual conference of the international speech communication association, 2010.
[53] T. Masters, Practical Neural Network Recipes in C++, Morgan Kaufmann, 1993.
[54] M.D. Zeiler, ADADELTA: AN adaptive learning rate method, 2012, arXiv preprint arXiv:1212.5701.
[55] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

Mohammad Ali Amirabadi was born in Zahedan, Iran, in 1993. He received the B.Sc. degree in Optics & Laser Engineering from Malek-e-Ashtar University of Technology, Isfahan, Iran, in 2015, and the M.Sc. degree in Communication Engineering from Iran University of Science and Technology, Tehran, Iran in 2017. Now he is studying Ph.D. in Communication Engineering in Iran University of Science and Technology, Tehran, Iran. His research interests include Multimode Fiber Optic Communication, Free Space Optical Communication, and Deep Learning.

Mohammad Hossein Kahaei received the B.Sc. degree from Isfahan University of Technology, Isfahan, Iran, in 1986, the MSc degree from the University of the Ryukyus, Okinawa, Japan, in 1994, and the Ph.D. degree in signal processing from the School of Electrical and Electronic Systems Engineering, Queensland University of Technology, Brisbane, Australia, in 1998. Since 1999, he has been with the School of Electrical Engineering, Iran University of Science and Technology, Tehran, Iran, where he is currently an Associate Professor and the head of Signal and System Modeling laboratory. His research interests include array signal processing with primary emphasis on compressed sensing, blind source separation, localization, tracking, and DOA estimation, and wireless sensor networks.

S. Alireza Nezamalhosseini received the B.Sc. degree in electrical engineering from Amirkabir University of Technology, Tehran, Iran, in 2006, and the M.Sc. and Ph.D. degrees in electrical engineering from Sharif University of Technology (SUT), Tehran, Iran, in 2008 and 2013, respectively. He is currently an assistant professor at Iran University of Science and Technology (IUST), Tehran. His research interests include underwater wireless optical communications, mode-division multiplexing in optical fibers, and visible light communications.