Layered-Parameter Perturbation for Zeroth-Order Optimization of Optical Neural Networks

Hiroshi Sawada[1], **Kazuo Aoyama**[1], **Masaya Notomi**[2]
[1]Communication Science Laboratories, NTT Corporation, Japan
[2]Basic Research Laboratories, NTT Corporation, Japan
hrsh.sawada@ntt.com, kazuo.aoyama@ntt.com, masaya.notomi@ntt.com

Abstract

Optical neural networks (ONNs) have attracted great attention due to their low power consumption and high-speed processing. When training an ONN implemented on a chip with possible fabrication variations, the well-known backpropagation algorithm cannot be executed accurately because the perfect information inside the chip cannot be observed. Instead, we employ a black-box optimization method such as zeroth-order (ZO) optimization. In this paper, we first discuss how ONN parameters should be perturbed to search for better values in a black-box manner. Conventionally, parameter perturbations are sampled from a normal distribution with an identity covariance matrix. This is plausible if the parameters are not interrelated in a module, like a linear module of an ordinary neural network. However, this is not the best way for ONN modules with layered parameters, which are interrelated by optical paths. We then propose to perturb the parameters by a normal distribution with a special covariance matrix computed by our novel method. The covariance matrix is designed so that the perturbations appearing at the module output caused by the parameter perturbations become as isotropic as possible to uniformly search for better values. Experimental results show that the proposed method using the special covariance matrix significantly outperformed conventional methods.

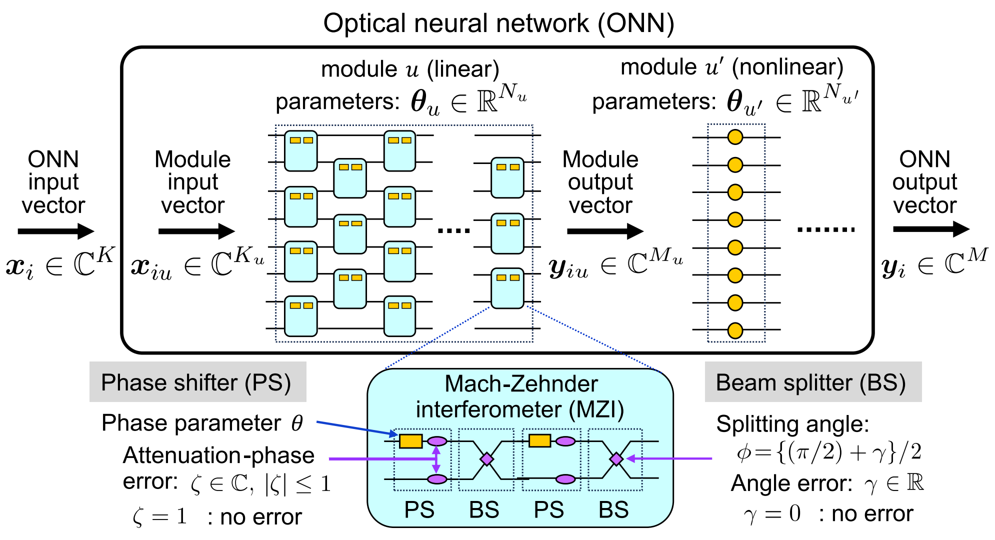
1 Introduction

Neural networks (NNs) have demonstrated remarkable performance in a wide variety of tasks at the ever-increasing cost of their size and power consumption. In this regard, optical neural networks (ONNs) are expected to be one of the promising systems due to their intrinsic characteristics of low power consumption and high-speed processing (Shen et al. 2017; Fang et al. 2019; Tang et al. 2021; Ashtiani, Geers, and Aflatouni 2022). An ONN is implemented on a programmable optical circuit based on silicon photonics. It processes complex-valued analog signals with coherent light (Carolan et al. 2015; Bogaerts et al. 2020). Figure 1 shows the structure of an ONN, which consists of linear and nonlinear modules as well as an ordinary NN. A linear module in an ONN typically comprises an array of Mach-Zehnder interferometers (MZIs) connected by waveguides (Reck and Zeilinger 1994; Clements et al. 2016). The MZI consists of two pairs of a phase shifter (PS) with a phase parameter θ and a beam splitter (BS). A nonlinear module may also have parameters. The programmability (or trainability) of ONNs is achieved by adjusting such parameters.

A major approach to training NNs in general is to compute the gradient of a loss function with respect to parameters. And the backpropagation algorithm (Bishop 2006; LeCun, Bengio, and Hinton 2015) is very common to compute the gradient. However, for training ONNs, we cannot accurately compute the gradient by backpropagation. This is because each ONN circuit implemented on a chip has its own fabrication variations (Fang et al. 2019; Banerjee, Nikdast, and Chakrabarty 2023). As shown in the bottom area of Fig. 1, let us model such variations by errors from ideal component characteristics, such as splitting angle errors $\gamma \in \mathbb{R}$ (Mikkelsen, Sacher, and Poon 2014; Vadlamani, Englund, and Hamerly 2023) and attenuation-phase errors $\zeta \in \mathbb{C}$. We may be able to use backpropagation with an ONN model simulated on an ordinary computer, with (Zheng et al. 2023; Sawada, Aoyama, and Ikeda 2024) or without modeling these errors, for an early stage of training. However, it is not a good idea to rely only on such imperfect gradients obtained from the backpropagation until the final stage of training, where accurate control of the parameters is of great importance.

As the perfect information inside an ONN chip is not precisely observable, black-box optimization methods have been applied to ONN training, including particle swarm optimization (Zhang et al. 2019), covariance matrix adaptation evolution strategy (CMA-ES) (Chen et al. 2022; Lupo et al. 2023), and zeroth-order (ZO) optimization (Shen et al. 2017; Zhou et al. 2020; Gu et al. 2020, 2021; Bandyopadhyay et al. 2022). Among these methods, ZO optimization (Liu et al. 2020) has become a major approach as (Gu et al. 2020) has experimentally shown that ZO optimization was superior to particle swarm optimization. We continue to focus on ZO optimization not only because of the shown superiority but also because it computes the (approximate) gradient of a loss function like backpropagation and is expected to scale to an ONN with a large number of parameters.

ZO optimization randomly perturbs the parameters to search for better values. The random parameter perturbations are generally sampled from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ with an identity covariance matrix $\mathbf{I}$ (Liu et al. 2020;



Optical neural network (ONN)
module u (linear)
parameters: $\boldsymbol{\theta}_u \in \mathbb{R}^{N_u}$
module u' (nonlinear)
parameters: $\boldsymbol{\theta}_{u'} \in \mathbb{R}^{N_{u'}}$
ONN input vector
$\boldsymbol{x}_i \in \mathbb{C}^K$
Module input vector
$\boldsymbol{x}_{iu} \in \mathbb{C}^{K_u}$
Module output vector
$\boldsymbol{y}_{iu} \in \mathbb{C}^{M_u}$
ONN output vector
$\boldsymbol{y}_i \in \mathbb{C}^M$
Phase shifter (PS)
Phase parameter θ
Attenuation-phase error: $\zeta \in \mathbb{C}$, $|\zeta| \le 1$
$\zeta = 1$: no error
Mach-Zehnder interferometer (MZI)
PS BS PS BS
Beam splitter (BS)
Splitting angle:
$\phi = \{(\pi/2) + \gamma\}/2$
Angle error: $\gamma \in \mathbb{R}$
$\gamma = 0$: no error

Figure 1: Optical neural network (ONN) based on Mach-Zehnder interferometer (MZI)

Gu et al. 2020), or similarly from independent Bernoulli distributions (Bandyopadhyay et al. 2022). This means that the parameters are perturbed independently in a module as well as in a whole NN. This is plausible for a linear module of an ordinary NN, where each parameter as an element of a matrix is independently responsible for each input-output pair. However, the parameters of MZI-based ONN linear modules are not independent, but interrelated to each other, as represented by their layered structures shown in the upper part of Fig. 2. By interrelation, we mean that a change in one parameter can be approximated by changes in its upstream/downstream parameters. This fact suggests us to consider another way of generating random perturbations than the one from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. One way is to use coordinate-wise perturbations (Liu et al. 2020; Gu et al. 2020, 2021). It avoids the effect of interrelation because only one parameter has a non-zero element in its perturbation vector. However, it is not very efficient for a need to change many parameters.

This paper proposes a new notion we call *layered-parameter perturbation*, which fits well to ONN parameters. Specifically, we propose to perturb the parameters of a linear module u with a layered structure by a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_u)$ with a covariance matrix $\mathbf{\Sigma}_u$ computed by our novel method. The covariance matrix is designed so that the perturbations of the output vector $\boldsymbol{y}_{iu}$ (Fig. 1), caused by parameter perturbations, become as isotropic as possible (Fig. 3). As a consequence, the whole ONN can be better trained in a non-informative black-box setting by uniformly searching for better values at the output of linear modules. Note that our special covariance matrices designed as above are fundamentally different from those in CMA-ES (Hansen 2016), which are computed directly from selected solutions of the population.

The notion of *layered-parameter perturbation* is considered for each linear module u in this paper, although it can be extended to the whole ONN. The reasons are twofold. The first is to keep our discussion and proposal simple. The second is to make the complexity of computing the covariance matrix $\mathbf{\Sigma}_u$ feasible. If it were of the whole ONN, the number of matrix elements would become the square of the total parameters. This would be infeasible for a large ONN with many parameters.

The rest of this paper is organized as follows. Section 2 provides the prerequisite knowledge of ONNs, training NNs, ZO optimization, and Jacobian matrix. After we show motivating examples in Sec. 3 regarding how parameters are interrelated, we propose a novel method for calculating the covariance matrix $\mathbf{\Sigma}_u$ in Sec. 4. Section 5 shows the experimental results. Section 6 concludes the paper. Table 1 lists the notations used in the paper.

2 Preliminaries

Optical Neural Network (ONN)

As shown in Fig. 1, an ONN consists of linear and nonlinear modules. Let a module u have N_u real-valued parameters $\boldsymbol{\theta}_u$. As a total, the ONN has $N = \sum_u N_u$ parameters $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_N] \in \mathbb{R}^N$. The ONN implements a function $\boldsymbol{f}$ that produces a complex-valued output vector $\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}) \in \mathbb{C}^M$ for an input vector $\boldsymbol{x}_i \in \mathbb{C}^K$. If we look at a module u, it implements a function $\boldsymbol{f}_u$ that produces an output vector $\boldsymbol{y}_{iu} = \boldsymbol{f}_u(\boldsymbol{x}_{iu}, \theta_u) \in \mathbb{C}^{M_u}$ for an input vector $\boldsymbol{x}_{iu} \in \mathbb{C}^{K_u}$. Outside the ONN, a loss function $\ell(\boldsymbol{y}_i, \boldsymbol{t}_i) \in \mathbb{R}$ with a target vector $\boldsymbol{t}_i$ is defined to train the ONN.

A linear module is typically an array of MZIs. An MZI consists of two PS-BS pairs. One PS-BS pair serves as the transfer matrix

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & j \\ j & 1 \end{pmatrix} \begin{pmatrix} e^{j\theta} & 0 \\ 0 & 1 \end{pmatrix} \tag{1}$$

programmable by a phase parameter θ. The imaginary unit j satisfies $j^2 = -1$. Here, we assume an ideal situation without error, i.e., $\zeta = 1$ and $\gamma = 0$. Then, an MZI represents a 2×2 unitary matrix $\mathbf{U}$, which satisfies $\mathbf{U}\mathbf{U}^{\mathsf{H}} = \mathbf{I}$ with $^{\mathsf{H}}$ being a conjugate transpose operator. For a nonlinear

$\boldsymbol{x}_i \in \mathbb{C}^K$	Input vector to ONN
$\boldsymbol{y}_i \in \mathbb{C}^M$	Output vector of ONN
$\boldsymbol{\theta} \in \mathbb{R}^N$	ONN parameters
$\gamma \in \mathbb{R}$	splitting angle error
$\zeta \in \mathbb{C}$	attenuation-phase error
$\boldsymbol{x}_{iu} \in \mathbb{C}^{K_u}$	Input vector to module u
$\boldsymbol{y}_{iu} \in \mathbb{C}^{M_u}$	Output vector of module u
$\boldsymbol{\theta}_u \in \mathbb{R}^{N_u}$	Module parameters
$\mathbf{J}_{iu} \in \mathbb{C}^{M_u \times N_u}$	Jacobian matrix from $\boldsymbol{\theta}_u$ to $\boldsymbol{y}_{iu}$
$\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^D$	Training dataset
$\mathcal{B} \subset \mathcal{D}$	Mini-batch dataset
$\ell \in \mathbb{R}$	Loss function
$\delta\boldsymbol{\theta}_q \in \mathbb{C}^N$	Perturbation vector for ZO optimization
$\delta\boldsymbol{\theta}_{uq} \in \mathbb{C}^{N_u}$	Perturbation for module u
$\boldsymbol{\Sigma}_u \in \mathbb{C}^{N_u \times N_u}$	Covariance matrix for sampling $\delta\boldsymbol{\theta}_{uq}$
$\mathbf{F}_u^{(i)} \in \mathbb{C}^{N_u \times N_u}$	Fisher information matrix
$\mathbf{C}_{\boldsymbol{y}_{iu}} \in \mathbb{C}^{M_u \times M_u}$	Output covariance matrix

Table 1: Notations

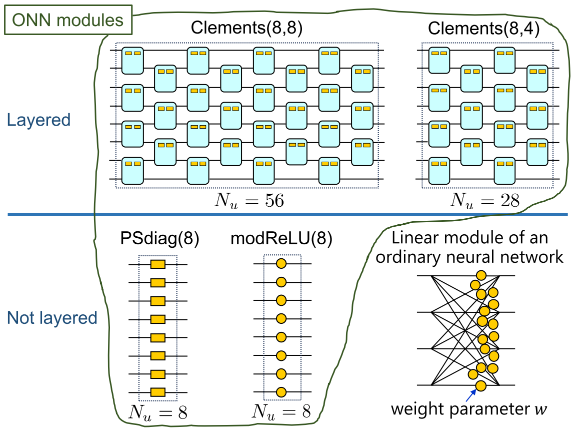


Figure 2: Module examples used in neural networks. The parameters of the upper modules are arranged in a layered structure in depth and therefore interrelated, while those of the lower modules are not.

module, we employ modReLU (Arjovsky, Shah, and Bengio 2016; Maduranga, Helfrich, and Ye 2019; Williamson et al. 2020) with a bias parameter θ whose element-wise function is given by

$$\text{modReLU}(y) = \begin{cases} y(|y| + \theta)/|y| & \text{if } |y| + \theta \geq 0 \\ 0 & \text{otherwise}. \end{cases} \tag{2}$$

Figure 2 shows module examples used in NNs. The Clements mesh (Clements et al. 2016) is the most popular ONN module, whose $K_u = 8$ case is illustrated as Clements(8,8). It realizes an arbitrary unitary matrix together with a diagonal matrix, illustrated as PSdiag(8), whose diagonal elements are of the form $e^{j\theta}$. In this case, the connected modules Clements(8,8)+PSdiag(8) have $56 + 8 = 64$ parameters in total, which corresponds to the required and sufficient number of parameters for an arbitrary 8-dimensional unitary matrix. One can reduce the circuit size by employing a truncated version (Fang et al. 2019) of the Clements mesh, whose 4-layer case is illustrated as Clements(8,4). In this case, the number of parameters reduces to 28, while the degree of freedom for realizing a unitary matrix is also decreased. The Clements mesh and its truncated version are layered arrays of MZIs. Thus, the phase parameters in the module are interrelated. Examples will be shown in Sec. 3. On the other hand, the parameters of PSdiag and modReLU are not interrelated when viewed as a single module. A linear module of an ordinary NN has weight parameters $\boldsymbol{w}$ that are also not interrelated.

Training NN by Zeroth-Order Optimization

Given a training dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^D$ of size D, training an NN is to optimize the parameters $\boldsymbol{\theta}$ so as to minimize the loss function

$$\ell_{\mathcal{D}}(\boldsymbol{\theta}) = \frac{1}{D}\sum_{i=1}^{D} \ell(\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}), \boldsymbol{t}_i)\,. \tag{3}$$

A standard way to optimize the parameters $\boldsymbol{\theta}$ is by mini-batch stochastic gradient descent (SGD) (Bottou 2012; Goodfellow, Bengio, and Courville 2016) in which we iterate the parameter updates

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta})\,, \tag{4}$$

$$\ell_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B} \ell(\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}), \boldsymbol{t}_i)\,, \tag{5}$$

where $\mathcal{B} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^B \subset \mathcal{D}$ is a mini-batch dataset drawn from $\mathcal{D}$, $\nabla_{\boldsymbol{\theta}}$ is the gradient with respect to $\boldsymbol{\theta}$ typically calculated by backpropagation (Bishop 2006; LeCun, Bengio, and Hinton 2015), and $\eta > 0$ is a learning rate.

For training ONNs, we cannot accurately compute the gradient by backpropagation because of fabrication variations. Therefore, we use ZO optimization (Liu et al. 2020) as a black-box optimization method to approximately compute the gradient:

$$\nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta}) \approx \hat{\nabla}_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta}) = \frac{\lambda}{Q}\sum_{q=1}^{Q} \delta\ell_q \cdot \delta\boldsymbol{\theta}_q\,, \tag{6}$$

where $\lambda > 0$ is a scale hyperparameter, $\delta\boldsymbol{\theta}_q$, $q = 1, \ldots, Q$ are random perturbation vectors, and

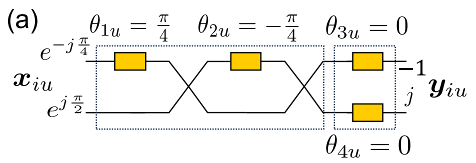
$$\delta\ell_q = \frac{\ell_{\mathcal{B}}(\boldsymbol{\theta} + \mu \cdot \delta\boldsymbol{\theta}_q) - \ell_{\mathcal{B}}(\boldsymbol{\theta})}{\mu}\,, \quad q = 1, \ldots, Q \tag{7}$$

are difference quotients by the finite difference method with a smoothing hyperparameter $\mu > 0$.

Typically, the random perturbation vectors $\delta\boldsymbol{\theta}_q$ are sampled from a zero-mean normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ with the covariance matrix being an identity matrix of size N, i.e., $\delta\boldsymbol{\theta}_q \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$. This means that the parameters $\boldsymbol{\theta}_u$ of a module u are perturbed by

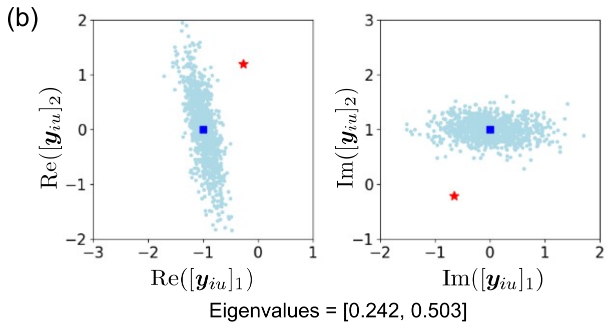
$$\delta\boldsymbol{\theta}_{uq} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{N_u})\,. \tag{8}$$

In Sec. 3, we demonstrate by examples that an identity matrix $\mathbf{I}_{N_u}$ is not the best covariance matrix for a module u



(c)
$$\frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\mathbf{F}_u^{(i)}=\begin{pmatrix}2.041 & 0.993 & 0.282 & 1.760\\ 0.993 & 1.926 & 0.962 & 0.997\\ 0.282 & 0.962 & 1.972 & 0.002\\ 1.760 & 0.997 & 0.002 & 2.073\end{pmatrix}$$

(d)
$$\mathbf{\Sigma}_u=\begin{pmatrix}1.973 & -0.020 & -0.270 & -1.665\\ -0.020 & 1.024 & -0.496 & -0.475\\ -0.270 & -0.496 & 0.787 & 0.467\\ -1.665 & -0.475 & 0.467 & 2.125\end{pmatrix}$$



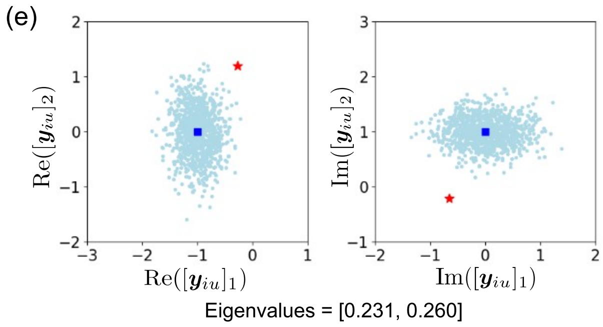


Figure 3: (a) Linear module u that represents a 2-dimensional unitary matrix with 4 phase parameters. (b) Output perturbations $\delta\boldsymbol{y}_{iuq}$ (light blue dots) occurred by perturbing the parameters with $\delta\boldsymbol{\theta}_{uq}\sim\mathcal{N}(\mathbf{0},\mathbf{I}_4)$. (c) Averaged Fisher information matrix that represents how parameters are interrelated. (d) Covariance matrix $\mathbf{\Sigma}_u$ calculated by the proposed method. (e) Output perturbations $\delta\boldsymbol{y}_{iuq}$ (light blue dots) by perturbing the parameters with $\delta\boldsymbol{\theta}_{uq}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma}_u)$.

whose parameters $\boldsymbol{\theta}_u$ are interrelated due to its layered structure. We therefore instead perturb such parameters

$$\delta\boldsymbol{\theta}_{uq}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma}_u) \tag{9}$$

with a covariance matrix $\mathbf{\Sigma}_u$.

Let us explain (Rasmussen and Williams 2006) how to generate such samples $\delta\boldsymbol{\theta}_{uq}$ from $\mathbf{\Sigma}_u$ when we have solely a subroutine generating a random vector $\boldsymbol{r}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$ with an identity covariance matrix, i.e., $\mathbb{E}\left[\boldsymbol{r}\boldsymbol{r}^\top\right]=\mathbf{I}$ with $^\top$ being a transpose operator. We first compute the Cholesky decomposition $\mathbf{\Sigma}_u=\mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix. We then compute what we need as $\delta\boldsymbol{\theta}_{uq}=\mathbf{L}\,\boldsymbol{r}$, which conforms to $\mathbb{E}_q\left[\delta\boldsymbol{\theta}_{uq}\delta\boldsymbol{\theta}_{uq}^\top\right]=\mathbf{L}\,\mathbb{E}\left[\boldsymbol{r}\boldsymbol{r}^\top\right]\mathbf{L}^\top=\mathbf{L}\mathbf{L}^\top=\mathbf{\Sigma}_u$.

Jacobian Matrix

To model how perturbations propagate from the parameters $\boldsymbol{\theta}_u$ to the output vector $\boldsymbol{y}_{iu}$ on a module u for a specific input vector $\boldsymbol{x}_{iu}$, let us introduce a Jacobian matrix as

$$\mathbf{J}_{iu}=\frac{\partial\boldsymbol{y}_{iu}}{\partial\boldsymbol{\theta}_u}=\begin{pmatrix}\frac{\partial[\boldsymbol{y}_{iu}]_1}{\partial[\boldsymbol{\theta}_u]_1} & \cdots & \frac{\partial[\boldsymbol{y}_{iu}]_1}{\partial[\boldsymbol{\theta}_u]_{N_u}}\\ \vdots & \ddots & \vdots\\ \frac{\partial[\boldsymbol{y}_{iu}]_{M_u}}{\partial[\boldsymbol{\theta}_u]_1} & \cdots & \frac{\partial[\boldsymbol{y}_{iu}]_{M_u}}{\partial[\boldsymbol{\theta}_u]_{N_u}}\end{pmatrix}\in\mathbb{C}^{M_u\times N_u}\,. \tag{10}$$

Then, the output perturbation $\delta\boldsymbol{y}_{iuq}$ caused by a parameter perturbation $\delta\boldsymbol{\theta}_{uq}$ can be expressed as a Jacobian-vector product

$$\delta\boldsymbol{y}_{iuq}=\mathbf{J}_{iu}\,\delta\boldsymbol{\theta}_{uq}\,, \tag{11}$$

which can be computed by forward mode automatic differentiation (AD) (Baydin et al. 2018). Conversely, the parameter perturbation $\delta\boldsymbol{\theta}_{up}^{(i)}$ caused by an output perturbation $\delta\boldsymbol{y}_{iup}$ (here we use index p to distinguish it from the above using q) can be expressed as a vector-Jacobian product (the following form is transposed $^\top$ for column vectors)

$$\delta\boldsymbol{\theta}_{up}^{(i)}=\mathbf{J}_{iu}^\top\,\delta\boldsymbol{y}_{iup}\,, \tag{12}$$

which can be computed by backpropagation, or reverse mode AD in this context. When we perform forward and reverse mode ADs, we have no choice but to assume an ideal situation without errors, i.e., $\zeta=1$ and $\gamma=0$, because both need the precise information inside the module u.

3 Motivating Examples

This section shows some examples on what happens if we use (8) for parameter perturbations, and how the situation improves by using *layered-parameter perturbations* (9).

2-Dimensional Case

Figure 3 shows a simple 2-dimensional case where we can inspect the situation in details. Panel (a) depicts a linear module u that represents a 2-dimensional unitary matrix with 4 phase parameters $\boldsymbol{\theta}_u=[\frac{\pi}{4},-\frac{\pi}{4},0,0]^\top$. For an input vector $\boldsymbol{x}_{iu}=[e^{-j\frac{\pi}{4}},e^{j\frac{\pi}{2}}]^\top$, the module produces $\boldsymbol{y}_{iu}=[-1,j]^\top$ at the output. Panels (b) and (e) represent the output's 2-dimensional complex spaces by the plots of the real parts $\mathrm{Re}([\boldsymbol{y}_{iu}]_1),\mathrm{Re}([\boldsymbol{y}_{iu}]_2)$ and the imaginary parts $\mathrm{Im}([\boldsymbol{y}_{iu}]_1),\mathrm{Im}([\boldsymbol{y}_{iu}]_2)$. In both panels, the blue squares represent the output values $[-1,j]^\top$ with no perturbation, i.e., with the current parameters $\boldsymbol{\theta}_u$, and the red stars represent the output values with the optimal parameters that we aim at. The light blue dots in panel (b) show the output perturbations (11) caused by typical parameter perturbations $\delta\boldsymbol{\theta}_{uq}$, $q=1,\ldots,Q$ with $Q=1000$, sampled from a distribution $\mathcal{N}(\mathbf{0},\mathbf{I}_4)$ with a 4-dimensional identity matrix. We observe that the output perturbations $\delta\boldsymbol{y}_{iuq}$ are not isotropic, i.e., strong in some specific directions and weak in others.

The reason comes from the fact that the parameters are interrelated by the layered structure shown in panel (a). One way to measure the degree of interrelation is to calculate the Fisher information matrix (Martens 2020) for the input $\boldsymbol{x}_{iu}$ with respect to the parameters $\boldsymbol{\theta}_u$. In our context, it is computed as the covariance matrix

$$\mathbf{F}_u^{(i)} = \mathbb{E}_p \left[\delta\boldsymbol{\theta}_{up}^{(i)} \delta\boldsymbol{\theta}_{up}^{(i)\mathsf{T}} \right] \approx \frac{1}{R_{\text{out}}} \sum_{p=1}^{R_{\text{out}}} \delta\boldsymbol{\theta}_{up}^{(i)} \delta\boldsymbol{\theta}_{up}^{(i)\mathsf{T}} \quad (13)$$

caused by newly generated isotropic perturbations

$$\delta\boldsymbol{y}_{iup} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{M_u}), \quad p = 1, \ldots, R_{\text{out}} \quad (14)$$

at the output $\boldsymbol{y}_{iu}$. Here, $\delta\boldsymbol{\theta}_{up}^{(i)}$ are the caused perturbations at the parameters $\boldsymbol{\theta}_u$, and can be computed by backpropagating (12) the generated output perturbation vectors $\delta\boldsymbol{y}_{iup}$. Panel (c) shows the average of such matrices $\frac{1}{R_{\text{in}}} \sum_{i=1}^{R_{\text{in}}} \mathbf{F}_u^{(i)}$ over $R_{\text{in}} = 1000$ random input vectors $\boldsymbol{x}_{iu}$ and $R_{\text{out}} = 10$ output perturbations. It represents that all parameter pairs except $(\theta_{3u}, \theta_{4u})$, which are not connected by an optical path as shown in panel (a), are interrelated by non-negligible values.

This paper proposes to use parameter perturbations sampled from a distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_u)$ with a covariance matrix $\boldsymbol{\Sigma}_u$ computed by the method explained in Sec. 4. In this specific case, $\boldsymbol{\Sigma}_u$ is given by panel (d), which apparently differs from $\mathbf{I}_4$. The light blue dots in panel (e) show the output perturbations $\delta\boldsymbol{y}_{iuq}$, $q = 1, \ldots, Q$ caused by the parameter perturbations $\delta\boldsymbol{\theta}_{uq}$ sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_u)$. We observe that these in panel (e) are more isotropic than those in panel (b). Since we aim at the optimal parameters that produce the red stars at the output by ZO optimization, the output perturbations in panel (e) are better at easily approaching the red stars than those in panel (b). Note that complete isotropy for each i-th output $\boldsymbol{y}_{iu}$ is difficult because the covariance matrix $\boldsymbol{\Sigma}_u$ is computed from the average of $\mathbf{F}_u^{(i)}$. One way to grasp how good or bad the output perturbations are is by inspecting the output covariance matrix

$$\mathbf{C}_{\boldsymbol{y}_{iu}} = \mathbb{E}_q \left[\delta\boldsymbol{y}_{iuq} \delta\boldsymbol{y}_{iuq}^{\mathsf{H}} \right] \approx \frac{1}{Q} \sum_{q=1}^{Q} \delta\boldsymbol{y}_{iuq} \delta\boldsymbol{y}_{iuq}^{\mathsf{H}}, \quad (15)$$

whose eigenvalues especially tell us the degree of isotropic. The more similar the eigenvalues are, the more isotropic the output perturbations are. The eigenvalues in this example are shown at the bottom of panels (b) and (e).

Clements Mesh and Its Truncated Version

Here, we examine the situations of Clements(8,8) and Clements(8,4), more practical modules than that of the previous subsection. In the left part of Fig. 4, the first column shows the averaged Fisher information matrices $\frac{1}{R_{\text{in}}} \sum_{i=1}^{R_{\text{in}}} \mathbf{F}_u^{(i)}$ over $R_{\text{in}} = 100$ random input vectors $\boldsymbol{x}_{iu}$ and $R_{\text{out}} = 10$ output perturbations. These two matrices correspond to panel (c) of Fig. 3. We observe that the parameters are interrelated as these matrices contain many off-diagonal non-negligible values. They are caused by the Clements(8,8) and Clements(8,4) structures depicted in Fig. 2. The second column shows the computed covariance matrices $\boldsymbol{\Sigma}_u$ for parameter perturbations, which correspond to panel (d) of Fig. 3. We observe that these matrices apparently differ from identity matrices $\mathbf{I}_{56}$ and $\mathbf{I}_{28}$.

The right hand side of Fig. 4 shows the eigenvalue distributions of the output covariance matrices $\mathbf{C}_{\boldsymbol{y}_{iu}}, i = 1, \ldots, R_{\text{in}}$ with $Q = 100$ parameter perturbations. The vertical bars represent one standard deviation over the $R_{\text{in}} = 100$ different input vectors. We observe that there are many large eigenvalues when we use the identity matrices $\mathbf{I}_{56}$ and $\mathbf{I}_{28}$ for perturbing the parameters. This means that the output perturbations are directional like panel (b) of Fig. 3. Contrary, if we use the covariance matrices $\boldsymbol{\Sigma}_u$ for perturbing the parameters, the eigenvalues become similar, meaning that the output perturbations are close to isotropic like panel (e) of Fig. 3.

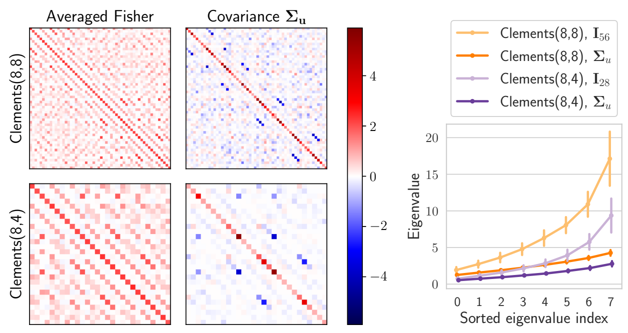


Figure 4: Clements(8,8) and Clements(8,4) examples. Left: averaged Fisher information matrices $\frac{1}{R_{\text{in}}} \sum_{i=1}^{R_{\text{in}}} \mathbf{F}_u^{(i)}$ and covariance matrices $\boldsymbol{\Sigma}_u$. Right: eigenvalue distributions of output covariance matrices $\mathbf{C}_{\boldsymbol{y}_{iu}}, i = 1, \ldots, R_{\text{in}}$ to show how isotropic or not the output perturbations are.

4 Proposed Method

Our proposed method improves ZO optimization for ONN training. The core of the improvement is to perturb module parameters $\boldsymbol{\theta}_u$ as $\boldsymbol{\theta}_u + \mu \cdot \delta\boldsymbol{\theta}_{uq}$ by (9) with $\boldsymbol{\Sigma}_u$ instead of (8) with $\mathbf{I}_{N_u}$ for modules u with layered parameters, such as those in the upper part of Fig. 2.

Covariance Matrix $\boldsymbol{\Sigma}_u$ for Perturbations

We design the covariance matrix $\boldsymbol{\Sigma}_u$ in (9) so that it minimize a cost function

$$\mathcal{C}(\boldsymbol{\Sigma}_u) = \frac{1}{R_{\text{in}}} \sum_{i=1}^{R_{\text{in}}} \mathcal{L}_{M_u}(\mathbf{C}_{\boldsymbol{y}_{iu}}, \mathbf{I}_{M_u}) + \rho \cdot \mathcal{L}_{N_u}(\boldsymbol{\Sigma}_u, \mathbf{I}_{N_u}), \quad (16)$$

where $\mathcal{L}_{M(\leftarrow M_u, N_u)}$ is the LogDet (Davis et al. 2007) (or Burg matrix (Davis and Dhillon 2006) or multichannel Itakura-Saito (Sawada et al. 2013)) divergence for matrices of size $M \times M$:

$$\mathcal{L}_M(\mathbf{A}, \mathbf{B}) = \text{trace}(\mathbf{A}\mathbf{B}^{-1}) - \log\det(\mathbf{A}\mathbf{B}^{-1}) - M. \quad (17)$$

The LogDet divergence measures how much two covariance matrices differ, and becomes zero if they are the same.

The first term of (16) is motivated by the examples and discussions in Sec. 3. They suggest to make the output covariance matrices $\mathbf{C}_{\boldsymbol{y}_{iu}}$ close to an identity matrix, whose eigenvalues are the same, for various input vectors $\boldsymbol{x}_{iu}$, $i = 1, \ldots, R_{\text{in}}$. Consequently, the output perturbations become as isotropic as possible, as shown in panel (e) of Fig. 3.

The second term with a hyperparameter $\rho > 0$ regularizes to prevent $\mathbf{\Sigma}_u$ from being singular by not making it far away from an identity matrix $\mathbf{I}_{N_u}$. The hyperparameter ρ should not be very large because $\mathbf{\Sigma}_u$ should be different from $\mathbf{I}_{N_u}$ according to our aim that replaces (8) with (9).

As a consequence, the covariance matrix $\mathbf{\Sigma}_u$ is given by

$$\mathbf{\Sigma}_u = (1+\rho)\left(\frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\mathbf{F}_u^{(i)} + \rho\cdot\mathbf{I}_{N_u}\right)^{-1}, \quad (18)$$

with the Fisher information matrices $\mathbf{F}_u^{(i)}$ averaged over R_{in} random input vectors $\boldsymbol{x}_{iu}$. How to derive (18) from (16) will be explained in the next subsection. As already explained when we introduced $\mathbf{F}_u^{(i)}$ in (13), $\mathbf{F}_u^{(i)}$ is computed first by sampling output perturbations $\delta\boldsymbol{y}_{iup} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{M_u})$, $p = 1, \ldots, R_{\text{out}}$ as (14), then by backpropagating them up to parameters $\delta\boldsymbol{\theta}_{up}^{(i)}$ as (12), and finally averaging their outer products over $p = 1, \ldots, R_{\text{out}}$ as (13).

Derivation of (18)

Let us first rewrite the two equations in Sec. 3. The first one (13) can be written as

$$\mathbf{F}_u^{(i)} = \mathbf{J}_{iu}^{\mathsf{T}}\,\mathbb{E}_p\left[\delta\boldsymbol{y}_{iup}\delta\boldsymbol{y}_{iup}^{\mathsf{H}}\right]\mathbf{J}_{iu}^{*} = \mathbf{J}_{iu}^{\mathsf{T}}\,\mathbf{J}_{iu}^{*}, \quad (19)$$

where * is the element-wise conjugate operator. Here, we use (12) and $\mathbb{E}_p\left[\delta\boldsymbol{y}_{iup}\delta\boldsymbol{y}_{iup}^{\mathsf{H}}\right] = \mathbf{I}_{M_u}$ because of $\delta\boldsymbol{y}_{iup} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{M_u})$. The second one (15) can be written as

$$\mathbf{C}_{\boldsymbol{y}_{iu}} = \mathbf{J}_{iu}\,\mathbb{E}_q\left[\delta\boldsymbol{\theta}_{uq}\delta\boldsymbol{\theta}_{uq}^{\mathsf{T}}\right]\mathbf{J}_{iu}^{\mathsf{H}} = \mathbf{J}_{iu}\,\mathbf{\Sigma}_u\,\mathbf{J}_{iu}^{\mathsf{H}}, \quad (20)$$

where we use (11) and $\mathbb{E}_q\left[\delta\boldsymbol{\theta}_{uq}\delta\boldsymbol{\theta}_{uq}^{\mathsf{T}}\right] = \mathbf{\Sigma}_u$ because of $\delta\boldsymbol{\theta}_{uq} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_u)$.

According to the definition of LogDet divergence (17) and the new form (20) of the output covariance matrix, the cost function (16) with ignoring constant terms can be written as

$$\mathcal{C}(\mathbf{\Sigma}_u) \overset{c}{=} \frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\left[\text{trace}\left(\mathbf{J}_{iu}\,\mathbf{\Sigma}_u\,\mathbf{J}_{iu}^{\mathsf{H}}\right) - \log\det(\mathbf{\Sigma}_u)\right] + \rho\cdot\left[\text{trace}(\mathbf{\Sigma}_u) - \log\det(\mathbf{\Sigma}_u)\right]. \quad (21)$$

The complex gradient matrix (Petersen and Pedersen 2008) with respect to $\mathbf{\Sigma}_u$ is given by

$$\begin{aligned}\frac{\partial\mathcal{C}(\mathbf{\Sigma}_u)}{\partial\mathbf{\Sigma}_u^{*}} &= \frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\left[\mathbf{J}_{iu}^{\mathsf{T}}\,\mathbf{J}_{iu}^{*} - \mathbf{\Sigma}_u^{-1}\right] + \rho\cdot\left[\mathbf{I}_{N_u} - \mathbf{\Sigma}_u^{-1}\right] \\ &= \frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\mathbf{F}_u^{(i)} + \rho\cdot\mathbf{I}_{N_u} - (1+\rho)\cdot\mathbf{\Sigma}_u^{-1}, \quad (22)\end{aligned}$$

where we use (19). Setting (22) to a zero matrix gives the covariance matrix (18) as the minimizer of (16).

Overall Procedure of ZO Optimization

Algorithm 1 shows the overall procedure of ZO optimization that generates *layered-parameter perturbations*. Lines 2, 3, 14-17 concerns the basic procedure of ZO optimization explained in Sec. 2. Lines from 4 to 10 are specific to our new proposal. Line 9 is the key operation of our proposal, which samples perturbations $\delta\boldsymbol{\theta}_{uq}$ for a module u with layered parameters by (9) as $\delta\boldsymbol{\theta}_{uq} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_u)$. For a module u with non-layered parameters, we sample perturbations $\delta\boldsymbol{\theta}_{uq}$ by (8) as with the ordinary ZO optimization (line 12). The procedure reduces to the ordinary one if we skip lines from 4 to 10 and apply line 12 for all the modules u.

Algorithm 1: ZO optimization with *layered-parameter perturbations*

```
Require: training dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^{D}$
Ensure: parameters $\boldsymbol{\theta}$
 1: while not converged, in $\tau$-th iteration do
 2:     Sample mini-batch $\mathcal{B}$ from $\mathcal{D}$
 3:     Evaluate loss $\ell_{\mathcal{B}}(\boldsymbol{\theta})$ (5)
 4:     for all module $u$ with layered parameters do
 5:         if $\tau \bmod T_{\text{ud}} = 0$ then
 6:             Update $\mathbf{F}_u$ by (24), (13), (12), and (14)
 7:             Update $\mathbf{\Sigma}_u$ by (23)
 8:         end if
 9:         Sample perturbations $\delta\boldsymbol{\theta}_{uq},\ q = 1,\ldots,Q$ by (9)
10:     end for
11:     for all module $u$ with non-layered parameters do
12:         Sample perturbations $\delta\boldsymbol{\theta}_{uq},\ q = 1,\ldots,Q$ by (8)
13:     end for
14:     Concatenate module perturbations $\delta\boldsymbol{\theta}_{uq}$ into the
        whole perturbation vectors $\delta\boldsymbol{\theta}_q,\ q = 1,\ldots,Q$
15:     Get $\delta\ell_q,\ q = 1,\ldots,Q$ by (7)
16:     Calculate the ZO gradient estimate (6)
17:     Update parameters $\boldsymbol{\theta}$ (4)
18: end while
```

In the iterations (lines 1 through 18), exponential smoothing (Gardner Jr 2006) leads to robust estimations of some statistics. In our particular case, we modify (18) to

$$\mathbf{\Sigma}_u = (1+\rho)\left(\mathbf{F}_u + \rho\cdot\mathbf{I}_{N_u}\right)^{-1}, \quad (23)$$

$$\mathbf{F}_u \leftarrow \alpha\cdot\frac{1}{R_{\text{in}}}\sum_{i=1}^{R_{\text{in}}}\mathbf{F}_u^{(i)} + (1-\alpha)\cdot\mathbf{F}_u, \quad (24)$$

where $\mathbf{F}_u$ is initially set as an identity matrix and $0 \le \alpha \le 1$ is a smoothing hyperparameter. In addition, we have empirically confirmed that the performance of ZO optimization is not degraded much even if we reduce the update frequency of these statistics. Therefore, we introduce another hyperparameter T_{ud} (line 5) to reduce the computational burden.

5 Experiments

Compared Methods

Four methods "ZO-**I**", "ZO-**co**", "ZO-$\mathbf{\Sigma}_u$" and "CMA" are compared. The first three are ZO optimization based on an identity matrix $\mathbf{I}$ as (8), **co**ordinate-wise perturbations where $\delta\boldsymbol{\theta}_q$ is a one-hot vector (only one element is 1 and all other elements are 0), and proposed special covariance matrices $\mathbf{\Sigma}_u$ as (9). "CMA" is CMA-ES (Hansen, Akimoto,

$B = 100$	Mini-batch size	Eq.(5)
$Q = K$	Number of perturbation vectors	Eq.(6)
$\lambda = 1/N$	Scale (ZO optimization)	Eq.(6)
$\mu = 0.001/\sqrt{N}$	Smoothing (ZO optimization)	Eq.(7)
$\alpha = 0.01$	Exponential smoothing of $\mathbf{F}_u$	Eq.(24)
$\rho = 0.1$	Regularizing weight for $\mathbf{\Sigma}_u$	Eq.(23)
$T_{\mathrm{ud}} = 100$	Update interval of $\mathbf{F}_u$ and $\mathbf{\Sigma}_u$	Alg.(1)
$R_{\mathrm{in}} = 100$	Number of random input vectors	Eq.(24)
$R_{\mathrm{out}} = 100$	Num. of output perturbation vectors	Eq.(14)

Table 2: Hyperparameter settings

and Baudis 2019), which also computes special covariance matrices, but in a fundamentally different way.

The parameters of layered modules Clements(K,L) were randomly initialized, while those of non-layered modules, e.g., PSdiag(K) and modReLU(K), were initialized to zero. The training procedure consisted of two steps. In the first step, we used backpropagation with error-free assumptions, i.e., $\zeta = 1$ and $\gamma = 0$, for a small number of 10 epochs. The intention of the first step was to quickly and roughly minimize the loss function from the initial parameters, even though the calculated gradients were inaccurate due to the wrong error assumption. In the second step, we ran some of the four methods, taking into account the actual error situation, with a sufficient number of 100 epochs.

For better convergence of the ZO optimization algorithm, we used the Adam optimizer (Kingma and Ba 2014) instead of vanilla SGD (4). The learning rate η of Adam and the step-size `sigma0` of CMA-ES were optimized by using Optuna (Akiba et al. 2019) for each combination of task, dimensionality ($K = 16, 32, 64$), and method. Table 2 shows the default settings of other hyperparameters. The upper four settings were common to the three ZO methods. The lower five settings were specific to the proposed method "ZO-$\mathbf{\Sigma}_u$".

Experimental Settings

We simulated ONN circuits with fabrication variations in an ordinary computer. For this purpose, we set the splitting angle errors γ and the attenuation-phase errors ζ (see Fig. 1) randomly by

$$\gamma = \sigma_\gamma \cdot r_0, \quad \zeta = (1 - \sigma_{\zeta,r} r_1) \exp[j \cdot \sigma_{\zeta,a}(2r_2 - 1)]\,, \quad (25)$$

where r_0 is a random number sampled from a normal distribution $\mathcal{N}(0, 1)$, r_1 and r_2 are ones sampled from a uniform distribution $[0, 1)$. We examined the following error settings:

$$\sigma_\gamma = 10^{-2}\beta\,,\ \sigma_{\zeta,r} = 10^{-3}\beta\,,\ \sigma_{\zeta,a} = 10^{-1}\beta \qquad (26)$$

where β controlled the amount of errors. The setting $\beta = 1$ corresponded to our estimate on an actual calibrated Clements mesh on silicon photonics. We set $\beta = 1$ unless otherwise noted.

We employed PyTorch (Paszke et al. 2019) to simulate and train ONNs. We built our customized CUDA kernels (Luebke 2008) for computational acceleration, and ran the program on an NVIDIA RTX A6000 (48 GB) as the GPU.

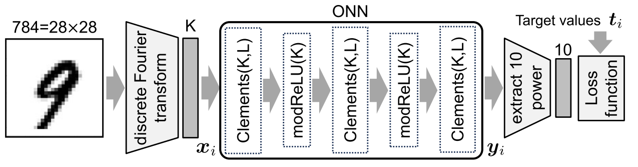


Figure 5: ONN configuration for image classification tasks.

Image Classification Task

Figure 5 shows our configuration for image classification tasks. We used the MNIST (LeCun and Cortes 2010) and FashionMNIST (Xiao, Rasul, and Vollgraf 2017) as image datasets, which were of appropriate difficulties for ONN circuits without memory. We first applied the discrete Fourier transform to a $784 = 28 \times 28$ pixel image and took the frequency bins from the second lowest to the $K + 1$ lowest (discarding the 0 Hz) as an input vector $\boldsymbol{x}_i$. Then, $\boldsymbol{x}_i$ was fed into the ONN to produce the output vector $\boldsymbol{y}_i$, whose central 10 dimensions were extracted. Finally, the powers of the extracted elements were calculated to produce a 10-dimensional real-valued vector indicating the image classification result. As in many related studies, test accuracy and training loss are used as evaluation metrics.

Table 3 shows the test accuracies over eight independent runs with settings $L = K$. In addition to the four methods compared, the row labeled "BP w error info" shows the results by backpropagation with perfect error information, which are unrealistic but can be considered as upper bounds. We observe that the proposed method "ZO-$\mathbf{\Sigma}_u$" generally outperformed the conventional methods (italic values indicate that they are statistically different from the corresponding best bold values according to the Mann-Whitney U test at a significance level of $p = 0.05$). We consider that "ZO-$\mathbf{I}$" suffered from the interrelation effect discussed in this paper. Also that "ZO-**co**" avoided the interrelation effect, but could not train (or change) parameters efficiently because its ZO gradient estimates had only Q non-zero elements. "CMA" did not scale well as one with K=64 did not complete a run even in one day. Let us examine the three ZO methods in more detail below.

	MNIST			FashionMNIST		
method	K=16	K=32	K=64	K=16	K=32	K=64
ZO-$\mathbf{I}$	*79.18%* ±0.43	*91.95%* ±0.19	*94.37%* ±0.09	*64.73%* ±0.34	*76.73%* ±0.39	*81.51%* ±0.15
ZO-**co**	79.37% ±0.21	92.17% ±0.16	*94.43%* ±0.15	*64.89%* ±0.34	*77.12%* ±0.26	*81.48%* ±0.18
ZO-$\mathbf{\Sigma}_u$	**79.60%** ±0.32	**92.29%** ±0.21	**94.70%** ±0.14	**65.34%** ±0.29	**77.43%** ±0.18	**81.87%** ±0.22
CMA	*78.88%* ±0.33	*91.92%* ±0.22	- -	*64.69%* ±0.35	*76.62%* ±0.35	- -
BP w error info	80.27% ±0.65	93.96% ±0.11	95.91% ±0.20	66.69% ±0.64	79.01% ±0.75	83.74% ±0.41

Table 3: Test accuracies @epoch=100. Mean ± standard deviation of eight runs. The best mean values among the four methods compared are shown in bold.

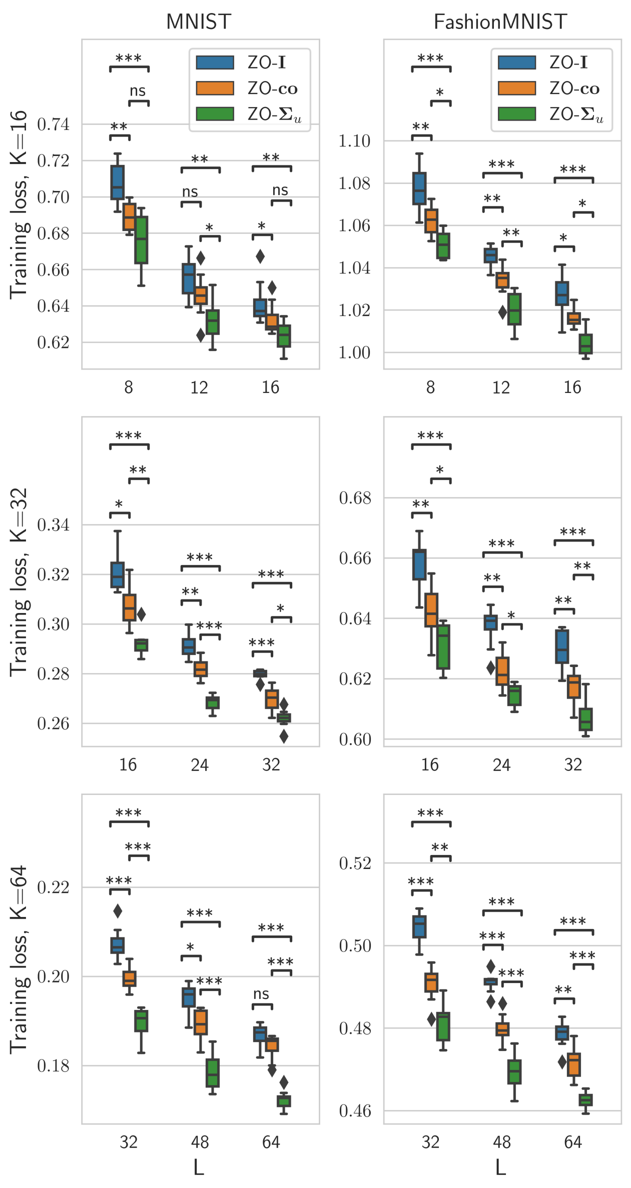


Figure 6: Training losses for image classification tasks. The p-value annotations are based on the Mann-Whitney U test. The legends are: *** for $10^{-4} < p \leq 10^{-3}$, ** for $10^{-3} < p \leq 10^{-2}$, * for $10^{-2} < p \leq 0.05$, and ns for $0.05 < p \leq 1$.

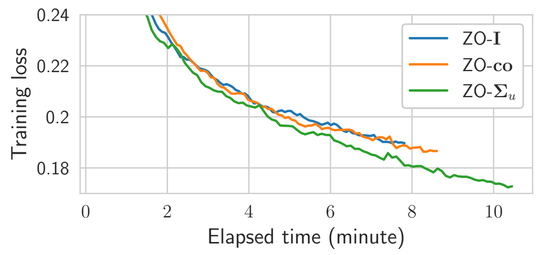


Figure 7: Convergence behaviors for $K = L = 64$ MNIST

The box plots in Fig. 6 show the distributions of the training losses with p-value annotations by the Mann-Whitney U test. We performed eight independent runs for each combination of dataset, method, dimensionality K and the number L of layers of Clements(K,L). We observe that "ZO-**co**" outperformed "ZO-**I**" and then the proposed method "ZO-$\mathbf{\Sigma}_u$" outperformed the winner "ZO-**co**" among the conventional methods. These were with statistical significance in many cases except for the four ns cases in the MNIST plots. Most notably, the proposed method enabled some truncated Clements meshes such as Clements(K=64,L=48) to outperform the full Clements meshes Clements(K,K) trained by the conventional methods, leading to significant circuit size savings without sacrificing classification performance.

Figure 7 shows some examples of convergence behavior as a function of the elapsed time. Although the proposed method "ZO-$\mathbf{\Sigma}_u$" apparently had a computational overhead over the other two ZO methods to complete the 100 epochs, the overhead was worth paying because the proposed method outperformed the other two methods most of the time, even with the same amount of elapsed time.

6 Conclusion

To better train MZI-based ONNs by ZO optimization in a black-box manner, we have discussed a new notion *layered-parameter perturbation* with some motivating examples. We have then proposed to perturb the parameters $\boldsymbol{\theta}_u$ of a layered module u using a special covariance matrix $\mathbf{\Sigma}_u$ computed by our novel method. Experimental results show that the proposed method using $\mathbf{\Sigma}_u$ significantly outperformed the conventional methods. The more layers the modules had, the greater the advantage of the proposed method. The computational overhead of the proposed method was not very large in the experiments and was worth paying for the improvement of the ZO optimization.

As the first proposal of *layered-parameter perturbation*, this paper demonstrated its effectiveness with a simple, small-scale feed-forward network. To achieve higher accuracy in the image classification task, optical convolution operations (Feldmann et al. 2021; Wirth-Singh et al. 2024; Zheng et al. 2024; Liu et al. 2024a) or diffractive optical neural networks (Lin et al. 2018) could be used. In such situations, error compensation networks (Mengu et al. 2020; Liu et al. 2024b) can be implemented with an MZI-based ONN, whose parameters can be efficiently trained by the proposed method. Future work also includes verifying the proposed method on an actual silicon photonic device.

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2623–2631.

Arjovsky, M.; Shah, A.; and Bengio, Y. 2016. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, 1120–1128. PMLR.

Ashtiani, F.; Geers, A. J.; and Aflatouni, F. 2022. An on-chip photonic deep neural network for image classification. *Nature*, 606: 501–506.

Bandyopadhyay, S.; Sludds, A.; Krastanov, S.; Hamerly, R.; Harris, N.; Bunandar, D.; Streshinsky, M.; Hochberg, M.; and Englund, D. 2022. Single chip photonic deep neural network with accelerated training. *arXiv preprint arXiv:2208.01623*.

Banerjee, S.; Nikdast, M.; and Chakrabarty, K. 2023. Characterizing Coherent Integrated Photonic Neural Networks Under Imperfections. *Journal of Lightwave Technology*, 41(5): 1464–1479.

Baydin, A.; Pearlmutter, B.; Radul, A.; and Siskind, J. 2018. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153): 1–43.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Bogaerts, W.; Pérez, D.; Capmany, J.; Miller, D. A. B.; Poon, J.; Englund, D.; Morichetti, F.; and Melloni, A. 2020. Programmable photonic circuits. *Nature*, 586: 207–216.

Bottou, L. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, 421–436. Springer.

Carolan, J.; Harrold, C.; Sparrow, C.; Martín-López, E.; Russell, N. J.; Silverstone, J. W.; Shadbolt, P. J.; Matsuda, N.; Oguma, M.; Itoh, M.; et al. 2015. Universal linear optics. *Science*, 349(6249): 711–716.

Chen, M. K.; Liu, X.; Sun, Y.; and Tsai, D. P. 2022. Artificial intelligence in meta-optics. *Chemical Reviews*, 122(19): 15356–15413.

Clements, W. R.; Humphreys, P. C.; Metcalf, B. J.; Kolthammer, W. S.; and Walmsley, I. A. 2016. Optimal design for universal multiport interferometers. *Optica*, 3(12): 1460–1465.

Davis, J.; and Dhillon, I. 2006. Differential entropic clustering of multivariate gaussians. *Advances in Neural Information Processing Systems*, 19.

Davis, J.; Kulis, B.; Jain, P.; Sra, S.; and Dhillon, I. 2007. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, 209–216.

Fang, M. Y.-S.; Manipatruni, S.; Wierzynski, C.; Khosrowshahi, A.; and DeWeese, M. R. 2019. Design of optical neural networks with component imprecisions. *Optical Express*, 27(10): 14009–14029.

Feldmann, J.; Youngblood, N.; Karpov, M.; Gehring, H.; Li, X.; Stappers, M.; Le Gallo, M.; Fu, X.; Lukashchuk, A.; Raja, A. S.; et al. 2021. Parallel convolutional processing using an integrated photonic tensor core. *Nature*, 589(7840): 52–58.

Gardner Jr, E. 2006. Exponential smoothing: The state of the art—Part II. *International journal of forecasting*, 22(4): 637–666.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Gu, J.; Feng, C.; Zhao, Z.; Ying, Z.; Chen, R. T.; and Pan, D. Z. 2021. Efficient on-chip learning for optical neural networks through power-aware sparse zeroth-order optimization. In *Proc. AAAI Conf. Artificial Intelligence*, volume 35, 7583–7591.

Gu, J.; Zhao, Z.; Feng, C.; Li, W.; Chen, R. T.; and Pan, D. Z. 2020. FLOPS: Efficient on-chip learning for optical neural networks through stochastic zeroth-order optimization. In *Proc. 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6.

Hansen, N. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

Hansen, N.; Akimoto, Y.; and Baudis, P. 2019. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*, 521: 436–444.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database.

Lin, X.; Rivenson, Y.; Yardimci, N. T.; Veli, M.; Luo, Y.; Jarrahi, M.; and Ozcan, A. 2018. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406): 1004–1008.

Liu, S.; Chen, P.-Y.; Kailkhura, B.; Zhang, G.; Hero III, A. O.; and Varshney, P. K. 2020. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5): 43–54.

Liu, Y.; Qin, J.; Liu, Y.; Liu, Y.; Liu, X.; Ye, F.; and Li, W. 2024a. Optical Fourier convolutional neural network with high efficiency in image classification. *Optics Express*, 32(13): 23575–23583.

Liu, Y.; Qin, J.; Liu, Y.; Yue, X.; Liu, X.; Wang, G.; Li, T.; Ye, F.; and Li, W. 2024b. Physics-Constrained Comprehensive Optical Neural Networks. In *Advances in Neural Information Processing Systems*.

Luebke, D. 2008. CUDA: Scalable parallel programming for high-performance scientific computing. In *2008 5th IEEE international symposium on biomedical imaging: from nano to macro*, 836–838. IEEE.

Lupo, A.; Picco, E.; Zajnulina, M.; and Massar, S. 2023. Deep photonic reservoir computer based on frequency multiplexing with fully analog connection between layers. *Optica*, 10(11): 1478–1485.

Maduranga, K. D. G.; Helfrich, K. E.; and Ye, Q. 2019. Complex unitary recurrent neural networks using scaled Cayley transform. In *Proc. AAAI Conf. Artificial Intelligence*, 4528–4535.

Martens, J. 2020. New insights and perspectives on the natural gradient method. *The Journal of Machine Learning Research*, 21(1): 5776–5851.

Mengu, D.; Zhao, Y.; Yardimci, N. T.; Rivenson, Y.; Jarrahi, M.; and Ozcan, A. 2020. Misalignment resilient diffractive optical networks. *Nanophotonics*, 9(13): 4207–4219.

Mikkelsen, J. C.; Sacher, W. D.; and Poon, J. K. S. 2014. Dimensional variation tolerant silicon-on-insulator directional couplers. *Opt. Express*, 22(3): 3145–3150.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

Petersen, K.; and Pedersen, M. 2008. The matrix cookbook. *Technical University of Denmark*, 7(15): 510.

Rasmussen, C.; and Williams, C. 2006. *Gaussian processes for machine learning*. MIT press Cambridge, MA.

Reck, M.; and Zeilinger, A. 1994. Experimental realization of any discrete unitary operator. *Phys. Rev. Lett.*, 73(1): 58–61.

Sawada, H.; Aoyama, K.; and Ikeda, K. 2024. Zeroth-Order Optimization of Optical Neural Networks with Linear Combination Natural Gradient and Calibrated Model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 1–6.

Sawada, H.; Kameoka, H.; Araki, S.; and Ueda, N. 2013. Multichannel extensions of non-negative matrix factorization with complex-valued data. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(5): 971–982.

Shen, Y.; Harris, N. C.; Skirlo, S.; Prabhu, M.; Baehr-Jones, T.; Hochberg, M.; Sun, X.; Zhao, S.; Larochelle, H.; Englund, D.; and Soljačić, M. 2017. Deep learning with coherent nanophotonic circuits. *Nat. Photonics*, 11: 441–446.

Tang, R.; Tanomura, R.; Tanemura, T.; and Nakano, Y. 2021. Ten-Port Unitary Optical Processor on a Silicon Photonic Chip. *ACS Photonics*, 8(7): 2074–2080.

Vadlamani, S. K.; Englund, D.; and Hamerly, R. 2023. Transferable learning on analog hardware. *Science Advances*, 9(28): eadh3436.

Williamson, I. A. D.; Hughes, T. W.; Minkov, M.; Bartlett, B.; Pai, S.; and Fan, S. 2020. Reprogrammable Electro-Optic Nonlinear Activation Functions for Optical Neural Networks. *IEEE Journal of Selected Topics in Quantum Electronics*, 26(1): 1–12.

Wirth-Singh, A.; Xiang, J.; Choi, M.; Fröch, J.; Huang, L.; Shlizerman, E.; and Majumdar, A. 2024. Compressed meta-optical encoder for image classification. In *CLEO: Fundamental Science*, FF1J–1. Optica Publishing Group.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Zhang, T.; Wang, J.; Dan, Y.; Lanqiu, Y.; Dai, J.; Han, X.; Sun, X.; and Xu, K. 2019. Efficient training and design of photonic neural network through neuroevolution. *Optics Express*, 27(26): 37150–37163.

Zheng, H.; Liu, Q.; Kravchenko, I. I.; Zhang, X.; Huo, Y.; and Valentine, J. G. 2024. Multichannel meta-imagers for accelerating machine vision. *Nature nanotechnology*, 19(4): 471–478.

Zheng, Z.; Duan, Z.; Chen, H.; Yang, R.; Gao, S.; Zhang, H.; Xiong, H.; and Lin, X. 2023. Dual adaptive training of photonic neural networks. *Nature Machine Intelligence*, 5(10): 1119–1129.

Zhou, H.; Zhao, Y.; Wang, X.; Gao, D.; Dong, J.; and Zhang, X. 2020. Self-configuring and reconfigurable silicon photonic signal processor. *ACS Photonics*, 7(3): 792–799.