Zeroth-Order Optimization of Optical Neural Networks with Linear Combination Natural Gradient and Calibrated Model

Hiroshi Sawada
NTT Corporation
Communication Science Laboratories
Japan

Kazuo Aoyama
NTT Corporation
Communication Science Laboratories
Japan

Kohei Ikeda
NTT Corporation
Basic Research Laboratories
Japan

ABSTRACT

Optical neural networks (ONNs) have attracted great attention due to their low energy consumption and high-speed processing. The usual neural network training scheme leads to poor performance for ONNs because of their special parameterization and fabrication variations. This paper contributes to extend zeroth-order (ZO) optimization, which can be used to train such ONNs, in two ways. The first is to propose linear combination natural gradient, which mitigates the optimization difficulty caused by the special parameterization of an ONN. The second is to generate a guided direction vector by calibration for better guessing than random vectors generated in ZO optimization. Experimental results show that the two extensions significantly outperformed the existing ZO optimization and related methods with little computational overhead.

KEYWORDS

Optical neural network (ONN), Neural network training, Zeroth-order (ZO) optimization, Natural gradient, Calibration

ACM Reference Format:
Hiroshi Sawada, Kazuo Aoyama, and Kohei Ikeda. 2024. Zeroth-Order Optimization of Optical Neural Networks with Linear Combination Natural Gradient and Calibrated Model. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655664

1 INTRODUCTION

Optical neural networks (ONNs) based on silicon photonics, which are programmable integrated circuits, are one of promising systems for various applications due to their intrinsic characteristics of low energy consumption and high-speed processing in principle [3, 8]. In particular, ONNs using coherent optical signals can perform complex-valued vector-matrix multiplication in an analog way [11, 29, 30]. Figure 1 shows the structure of an ONN, which consists of linear and nonlinear units. A linear unit in a coherent ONN is an array of tunable Mach-Zehnder interferometers (MZIs) connected with waveguides [10, 28]. The MZI consists of sequentially coupled two pairs of a programmable phase shifter (PS) with a parameter θ_{n_p} and a directional coupler (DC) illustrated as (PS-DC)(PS-DC). The processing by the ONN is impacted by fabrication variations of its components [6, 11]. Let us model such variations by errors

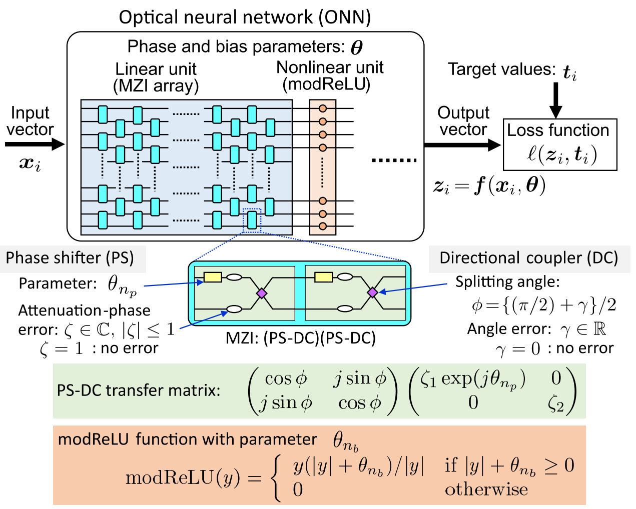
Optical neural network (ONN)
Phase and bias parameters: $\boldsymbol{\theta}$
Linear unit (MZI array)
Nonlinear unit (modReLU)
Input vector $\boldsymbol{x}_i$
Output vector
Target values: $\boldsymbol{t}_i$
Loss function $\ell(\boldsymbol{z}_i, \boldsymbol{t}_i)$
$\boldsymbol{z}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta})$
Phase shifter (PS)
Parameter: θ_{n_p}
Attenuation-phase error: $\zeta \in \mathbb{C}$, $|\zeta| \leq 1$
$\zeta = 1$: no error
MZI: (PS-DC)(PS-DC)
Directional coupler (DC)
Splitting angle: $\phi = \{(\pi/2) + \gamma\}/2$
Angle error: $\gamma \in \mathbb{R}$
$\gamma = 0$: no error
PS-DC transfer matrix: $\begin{pmatrix} \cos\phi & j\sin\phi \\ j\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \zeta_1 \exp(j\theta_{n_p}) & 0 \\ 0 & \zeta_2 \end{pmatrix}$
modReLU function with parameter θ_{n_b}
$\mathrm{modReLU}(y) = \begin{cases} y(|y| + \theta_{n_b})/|y| & \text{if } |y| + \theta_{n_b} \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Figure 1: Optical neural network (ONN)

from ideal component characteristics, such as splitting angle errors $\gamma \in \mathbb{R}$ [25, 31] and attenuation-phase errors $\zeta \in \mathbb{C}$ that include phase variations and additional attenuation in waveguides.

Training such ONNs is much more difficult than training usual neural networks (NNs) built in ordinary computers. Most notably, the component errors caused by fabrication variations cannot be accurately modeled and observed. Therefore, the gradients with respect to parameters cannot be calculated by the backpropagation algorithm [7, 17], which needs perfect internal information of the NN. One approach is *in situ* training that evaluates approximate gradients directly on an ONN chip [5, 29]. Zeroth-order (ZO) optimization [19], which is a broader concept including the *in situ* training, has been widely used to train ONNs [13, 14, 34]. ZO optimization approximates the gradients by a set of random perturbations to the parameters in a black-box manner.

Another difficulty comes from the structure of the MZI array, which is different from a matrix used in a usual NN in terms of parameterization, and whose parameters θ_{n_p} may be interrelated by the repeated structure (shown in blue in Fig. 1). Natural gradient [1, 23] is known to make training robust to re-parameterization of NNs [26]. This property would contribute to training ONNs because an MZI array can be seen as a re-parameterization of a matrix used in a usual NN. A straightforward combination of ZO optimization and natural gradient has been proposed for another task [33].

In this paper, we improve ONN training in terms of efficiency and stability by extending the ZO optimization in two ways. The first is to propose *linear combination natural gradient* (LCNG), a novel optimization method derived from the notion of natural gradient and suitable for ZO optimization (Sec. 3.2). As shown in the experiments, both the existing combination [33] and our proposed LCNG outperformed the ordinary ZO optimization (Table 2), but the computational cost of the LCNG is much less than the existing combination (Fig. 4). The second extension is to employ a model NN that tries to mimic the ONN by calibration, and to generate a *guided direction vector* by backpropagation for better guessing than random vectors (Sec. 3.3). Experimental results show that the second extension further improved LCNG when the component error settings were not so severe (the lower plot of Fig. 5).

The rest of this paper is organized as follows. Section 2 explains the basics of ONN, NN training, and ZO optimization. We propose our new optimization method in Sec. 3, and its effectiveness is demonstrated in Sec. 4. Section 5 concludes the paper.

Table 1 lists the notations used in this paper.

2 PRELIMINARIES

2.1 Optical Neural Network (ONN)

As shown in Fig. 1, an ONN consists of linear and nonlinear units. In this paper, we use the Clements mesh [10] as the MZI array of a linear unit to implement a unitary matrix. And we use modReLU [2, 20, 32] as a nonlinear unit whose function is described in the figure. Each MZI consists of two pairs of PS-DC. One pair of PS-DC serves as the transfer matrix shown in the figure, where j is the imaginary unit. The PS and modReLU have a phase parameter and a bias parameter denoted as θ_{n_p} and θ_{n_b}, respectively. To model fabrication variations, we assume component errors in the MZI: $\zeta_1, \zeta_2 \in \mathbb{C}, |\zeta_1|, |\zeta_2| \leq 1$ are attenuation-phase errors, and $\gamma \in \mathbb{R}$ is a splitting angle error.

Let us denote the phase or bias parameter defined above by $\theta_n \in \mathbb{R}$ for unified management, and let $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_N] \in \mathbb{R}^N$ be a parameter vector. The ONN implements a function $\boldsymbol{f}$ with N parameters $\boldsymbol{\theta}$, and produces an output vector $\boldsymbol{z}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}) \in \mathbb{R}^M$ for an input vector $\boldsymbol{x}_i$. Outside the ONN, a loss function $\ell(\boldsymbol{z}_i, \boldsymbol{t}_i)$ with a target vector $\boldsymbol{t}_i$ is defined to train the ONN.

2.2 Training NN and Natural Gradient

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 total loss

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

A standard way to optimize the parameters $\boldsymbol{\theta}$ is by mini-batch stochastic gradient descent (SGD) [9, 12] in which we iterate the parameter updates

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta})\,, \tag{2}$$

$$\ell_{\mathcal{B}}(\boldsymbol{\theta}) = \tfrac{1}{B} \textstyle\sum_{i=1}^B \ell(\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}), \boldsymbol{t}_i)\,, \tag{3}$$

where $\mathcal{B} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^B \subset \mathcal{D}$ is a mini-batch drawn from the training dataset, $\nabla_{\boldsymbol{\theta}}$ is the gradient with respect to $\boldsymbol{\theta}$ typically calculated by backpropagation [7, 17], and $\eta > 0$ is a learning rate.

For NNs with possibly interrelated parameters such as ONNs with MZI arrays, gradient descent may exhibit slow or unstable convergence. Natural gradient [1, 23] is attractive for such situations because it is less affected by parameterization [26]. As summarized in [4], many existing optimization methods, including gradient descent and natural gradient, can be derived from the following update of the proximal point method

$$\boldsymbol{\theta} \leftarrow \operatorname{argmin}_{\boldsymbol{u}} \left\{ \ell_{\mathcal{B}}(\boldsymbol{u}) + \frac{\lambda_P}{\eta} \frac{1}{2} ||\boldsymbol{u} - \boldsymbol{\theta}||_2^2 + \frac{\lambda_F}{\eta} \xi_{\mathcal{B}}(\boldsymbol{u}, \boldsymbol{\theta}) \right\}\,. \tag{4}$$

Table 1: Notations

$\boldsymbol{f}$	Function implemented by neural network
$\boldsymbol{x}_i$	Input vector to neural network
$\boldsymbol{z}_i \in \mathbb{R}^M$	Output vector of neural network
$\boldsymbol{\theta} \in \mathbb{R}^N$	Phase and bias parameters to be trained
$\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^D$	Training dataset
$\boldsymbol{t}_i$	Target vector
$\mathcal{B} \subset \mathcal{D}$	Mini-batch drawn from dataset
ℓ	Loss function for training neural network
$\eta > 0$	Learning rate
$\lambda_P \geq 0, \lambda_F \geq 0$	Weights for PSD and FSD terms
$\mathbf{F}_{\boldsymbol{\theta}}, \mathbf{F}_{\boldsymbol{z}_i}$	Fisher information matrices (FIMs)
$\delta\boldsymbol{\theta}_q$	Direction vector for ZO optimization
$\delta\ell_q$	Difference quotient for ZO optimization
$\delta\boldsymbol{z}_{iq}$	Change of neural network output
μ	Smoothing hyperparameter
$\boldsymbol{a}$	LCNG coefficients
$\boldsymbol{\psi} = \{\boldsymbol{\zeta}, \boldsymbol{\gamma}\}$	Component error parameters to be calibrated
$\boldsymbol{\zeta}$	Attenuation-phase errors ($\zeta = 1$: no error)
$\boldsymbol{\gamma}$	Splitting ratio errors ($\gamma = 0$: no error)
$\boldsymbol{e}$	Discrepancy for calibrating model network
$0 < \alpha \leq 1$	Hyperparameter to control guided randomness

Here, the first term in the curly brackets corresponds to the loss function, the second term with a weight $\lambda_P \geq 0$ is the parameter space discrepancy (PSD), and the third term with a weight $\lambda_F \geq 0$ is the function space discrepancy (FSD). The gradient descent update (2) can be derived with a first-order approximation to the loss term and by setting $\lambda_P = 1,\ \lambda_F = 0$, which means that the FSD term is ignored. In the case of natural gradient, we introduce a probability distribution $p_{\boldsymbol{\theta}}$, and take into account the FSD term

$$\xi_{\mathcal{B}}(\boldsymbol{u}, \boldsymbol{\theta}) = \tfrac{1}{B} \textstyle\sum_{i=1}^B d_{\mathrm{KL}} \left[p_{\boldsymbol{u}}(\boldsymbol{t} | \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u})), p_{\boldsymbol{\theta}}(\boldsymbol{t} | \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta})) \right]\,, \tag{5}$$

where d_{KL} is the Kullback-Leibler divergence [7] between two distributions. By applying the second order approximation to the FSD term, we have a natural gradient update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot (\lambda_P \mathbf{I} + \lambda_F \mathbf{F}_{\boldsymbol{\theta}})^{-1} \nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta})\,, \tag{6}$$

where $\mathbf{I}$ is an identity matrix and

$$\mathbf{F}_{\boldsymbol{\theta}} = \frac{1}{B} \sum_{i=1}^{B} \mathbb{E}_{\boldsymbol{t} \sim p_{\boldsymbol{\theta}}(\boldsymbol{t}|\boldsymbol{z}_i)} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{t} \mid \boldsymbol{z}_i) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{t} \mid \boldsymbol{z}_i)^{\mathsf{T}} \right] \tag{7}$$

is the Fisher information matrix (FIM) with respect to the parameters $\boldsymbol{\theta}$. As noted in [16], we do not use the target vector $\boldsymbol{t}_i$ but draw samples $\boldsymbol{t}$ from $p_{\boldsymbol{\theta}}(\boldsymbol{t} \mid \boldsymbol{z}_i)$.

2.3 Zeroth-Order Optimization

Zeroth-order (ZO) optimization [19] is a useful technique for situations where the gradient of a loss function ℓ cannot be calculated by backpropagation but only function queries $\ell(\boldsymbol{z}_i, \boldsymbol{t}_i)$, $\boldsymbol{z}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta})$ are allowed. It approximates the gradient with the ZO gradient estimate:

$$\nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta}) \approx \hat{\nabla}_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta}) = \textstyle\sum_{q=1}^Q \delta\ell_q \cdot \delta\boldsymbol{\theta}_q\,, \tag{8}$$

where $\delta\boldsymbol{\theta}_q$, $q = 1, \ldots, Q$ are direction vectors typically randomly drawn from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}/N)$, and

$$\delta\ell_q = \frac{\ell_{\mathcal{B}}(\boldsymbol{\theta} + \mu \cdot \delta\boldsymbol{\theta}_q) - \ell_{\mathcal{B}}(\boldsymbol{\theta})}{\mu}\,, \quad q = 1, \ldots, Q \tag{9}$$

are difference quotients with a smoothing hyperparameter $\mu > 0$. Note that we need $B \times (Q + 1)$ queries for a mini-batch of size B and

Q direction vectors. The parameters are updated similarly to (2)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \hat{\nabla}_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta}) \tag{10}$$

with the ZO gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta})$ defined in (8).

3 PROPOSED METHOD

3.1 Overview

Our proposed method aims to train the parameters $\boldsymbol{\theta}$ of an ONN by a variant of ZO optimization extended in two ways. The first extension is by the newly proposed LCNG, which efficiently achieves the merit of natural gradient in the context of ZO optimization. The second is to employ a model NN to produce a guided direction vector that may be better than those drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I}/N)$.

Algorithm 1 shows a pseudocode of the proposed method. The lines from 3 to 11 represent a sequence of operations for a mini-batch $\mathcal{B}$. The ordinary ZO optimization concerns lines 3, 8, 9, and 11, except that $\boldsymbol{z}_i$ and $\delta\mathbf{Z}_i$ do not have to be kept. For LCNG, the explicit representation of $\boldsymbol{z}_i$ and $\delta\mathbf{Z}_i$ as well as the execution of lines 4 and 10 are additionally required. To create a guided direction vector $\delta\boldsymbol{\theta}_1$, lines from 5 to 7 are also executed, followed by line 8. If no guided direction vector is created, let the random vectors in line 8 be $\delta\boldsymbol{\theta}_1, \ldots, \delta\boldsymbol{\theta}_Q$.

3.2 Linear Combination Natural Gradient

Our proposed LCNG follows the principle of natural gradient, but differs in some aspects as listed below.

a) Both are derived from (4) taking into account the FSD term.
b) LCNG is specifically designed for ZO optimization, where the gradient estimate is a linear combination of direction vectors such as (8). Consequently, the gradient estimate is optimized within the subspace spanned by the Q direction vectors. It has less flexibility than the natural gradient whose flexibility is the whole N-dimensional parameter space.
c) Natural gradient needs to compute the inverse $(\lambda_P \mathbf{I} + \lambda_F \mathbf{F}_{\boldsymbol{\theta}})^{-1}$ as (6) whose size $N \times N$ depends on the number of parameters N. Since the inverse computation is impractical for large N, it is practically approximated, e.g., [24]. On the other hand, LCNG computes the inverse of a $Q \times Q$ matrix as shown in (13), which is small because the number Q of direction vectors is practically up to hundreds.

Let us denote the direction vectors $\delta\boldsymbol{\theta}_q$ and the difference quotients $\delta\ell_q$ introduced in Sec. 2.3 in matrix and vector forms as $\delta\boldsymbol{\Theta} = [\delta\boldsymbol{\theta}_1, \ldots, \delta\boldsymbol{\theta}_Q] \in \mathbb{R}^{N \times Q}$ and $\delta\boldsymbol{\ell} = [\delta\ell_1, \ldots, \delta\ell_Q]^\mathsf{T} \in \mathbb{R}^Q$, where $^\mathsf{T}$ is a transpose operator. Moreover, let us define $\delta\mathbf{Z}_i = [\delta z_{i1}, \ldots, \delta z_{iQ}] \in \mathbb{R}^{M \times Q}$ with

$$\delta z_{iq} = \frac{\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta} + \mu \cdot \delta\boldsymbol{\theta}_q) - \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta})}{\mu}, \quad q = 1, \ldots, Q, \tag{11}$$

each of which indicates how the NN output is changed by the parameter change with a direction vector $\delta\boldsymbol{\theta}_q$. Then, LCNG updates the parameters by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \delta\boldsymbol{\Theta}\, \boldsymbol{a} \tag{12}$$

with coefficients

$$\boldsymbol{a} = \left(\lambda_P \cdot \delta\boldsymbol{\Theta}^\mathsf{T} \delta\boldsymbol{\Theta} + \frac{\lambda_F}{B} \sum_{i=1}^{B} \delta\mathbf{Z}_i^\mathsf{T} \mathbf{F}_{z_i} \delta\mathbf{Z}_i \right)^{-1} \delta\boldsymbol{\ell}, \tag{13}$$

Algorithm 1: LCNG-based guided ZO optimization

```
  Data: training dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{t}_i\}_{i=1}^D$
  Result: parameters $\boldsymbol{\theta}$
1 repeat
2     forall mini-batches $\mathcal{B}$ sampled from $\mathcal{D}$ do
3         Evaluate loss $\ell_{\mathcal{B}}(\boldsymbol{\theta})$ (3) and outputs $\boldsymbol{z}_i$ by queries;
4         Calculate output FIM $\mathbf{F}_{z_i}$ (14);
5         Update error parameters $\boldsymbol{\psi}$ (32);
6         Calculate model gradient $\boldsymbol{g}$ (34);
7         Draw a guided direction vector $\delta\boldsymbol{\theta}_1$ from (35);
8         Draw random direction vectors $\delta\boldsymbol{\theta}_2, \ldots, \delta\boldsymbol{\theta}_Q$;
9         Get $\delta\boldsymbol{\ell}$ (9) and $\delta\mathbf{Z}_i$ (11) by additional queries;
10        Calculate LCNG coefficients $\boldsymbol{a}$ (13);
11        Update parameters $\boldsymbol{\theta}$ (12);
12    end
13 until convergence;
```

where

$$\mathbf{F}_{z_i} = \mathbb{E}_{\boldsymbol{t} \sim p(\boldsymbol{t} | z_i)} \left[\nabla_{z_i} \log p(\boldsymbol{t} \mid z_i) \, \nabla_{z_i} \log p(\boldsymbol{t} \mid z_i)^\mathsf{T} \right] \tag{14}$$

is the FIM with respect to the output $\boldsymbol{z}_i$. The derivation of (13) and the procedure for calculating (14) are explained in Subsections 3.2.1 and 3.2.2, respectively.

We see that the ordinary ZO gradient estimate (8) is a special case of LCNG in which $\delta\boldsymbol{\Theta}^\mathsf{T}\delta\boldsymbol{\Theta} = \mathbf{I}$, $\lambda_P = 1$, and $\lambda_F = 0$. It is similar to the gradient descent in a sense that both ignore the FSD terms, i.e., the third term in the curly brackets of (4) and the second term in the parenthesis of (13). In contrast, LCNG takes care of these FSD terms with some weight λ_F, and therefore puts relatively less importance on the PSD term, which is the second term in the curly brackets of (4) and is directly affected by the parameterization. Consequently, optimization with LCNG is less sensitive to the MZI array parameterization of the ONN than with ordinary ZO.

3.2.1 LCNG coefficients (13) derivation. LCNG aims to optimize the coefficients $\boldsymbol{a} = [a_1, \ldots, a_Q]^\mathsf{T}$ such that the parameter update (12) approximately solve the minimization problem (4). Let

$$\boldsymbol{u} = \boldsymbol{\theta} - \eta \cdot \delta\boldsymbol{\Theta}\, \boldsymbol{a}, \tag{15}$$

and we are looking into the three terms in the curly brackets of (4). The first term is approximated by linearization as

$$\ell_{\mathcal{B}}(\boldsymbol{u}) \approx \ell_{\mathcal{B}}(\boldsymbol{\theta}) - \eta \cdot \delta\boldsymbol{\ell}^\mathsf{T} \boldsymbol{a}. \tag{16}$$

The second term is given by

$$\frac{\eta \cdot \lambda_P}{2} ||\delta\boldsymbol{\Theta}\, \boldsymbol{a}||_2^2. \tag{17}$$

The third term is approximated by the second-order Taylor series with $\boldsymbol{a}$ as

$$\frac{\lambda_F}{\eta} \xi_{\mathcal{B}}(\boldsymbol{u}, \boldsymbol{\theta}) \approx \frac{\eta \cdot \lambda_F}{2} \boldsymbol{a}^\mathsf{T} \delta\boldsymbol{\Theta}^\mathsf{T} \mathbf{F}_{\boldsymbol{\theta}}\, \delta\boldsymbol{\Theta}\, \boldsymbol{a}. \tag{18}$$

Therefore, the objective function to be minimized with respect to $\boldsymbol{a}$ is given by

$$C(\boldsymbol{a}) = -\delta\boldsymbol{\ell}^\mathsf{T} \boldsymbol{a} + \frac{1}{2} \left(\lambda_P ||\delta\boldsymbol{\Theta}\, \boldsymbol{a}||_2^2 + \lambda_F \cdot \boldsymbol{a}^\mathsf{T} \delta\boldsymbol{\Theta}^\mathsf{T} \mathbf{F}_{\boldsymbol{\theta}}\, \delta\boldsymbol{\Theta}\, \boldsymbol{a} \right) \tag{19}$$

and solving $\frac{\partial}{\partial \boldsymbol{a}} C(\boldsymbol{a}) = 0$ gives

$$\boldsymbol{a} = \left(\lambda_P \cdot \delta\boldsymbol{\Theta}^\mathsf{T} \delta\boldsymbol{\Theta} + \lambda_F \cdot \delta\boldsymbol{\Theta}^\mathsf{T} \mathbf{F}_{\boldsymbol{\theta}} \delta\boldsymbol{\Theta} \right)^{-1} \delta\boldsymbol{\ell} \tag{20}$$

that minimizes $C(\boldsymbol{a})$.

Now, let us introduce a Jacobian matrix

$$\mathbf{J}_{z_i\theta} = \frac{\partial z_i}{\partial \boldsymbol{\theta}} = \frac{\partial f(\boldsymbol{x}_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{M\times N} \tag{21}$$

so that $\mathbf{F}_{\boldsymbol{\theta}}$ can be expresses as

$$\mathbf{F}_{\boldsymbol{\theta}} = \frac{1}{B}\sum_{i=1}^{B} \mathbf{J}_{z_i\theta}^{\mathsf{T}} \mathbf{F}_{z_i} \mathbf{J}_{z_i\theta} \,. \tag{22}$$

The Jacobian matrix $\mathbf{J}_{z_i\theta}$ can typically be calculated by backpropagation. However, in the context of ZO optimization, we need to resort to another way, and we employ a least-squares and minimum-norm solution of $||\delta \mathbf{Z}_i - \mathbf{J}_{z_i\theta}\,\delta\Theta||_2^2$, i.e., the Moore-Penrose inverse

$$\mathbf{J}_{z_i\theta} = \delta\mathbf{Z}_i \left(\delta\Theta^{\mathsf{T}}\delta\Theta\right)^{-1} \delta\Theta^{\mathsf{T}} \,. \tag{23}$$

Substituting (23) into (22) and then (22) into (20) gives (13).

3.2.2 *Probability distributions and output FIMs (14).*

We have two options for calculating the FIM (14) with respect to the output $\boldsymbol{z}_i$ as there are two typical probability distributions that can be defined at the output of an NN. The first is a multivariate normal distribution

$$p(\boldsymbol{t} \mid \boldsymbol{z}_i) = \mathcal{N}(\boldsymbol{t} \mid \boldsymbol{z}_i, \mathbf{I}) = \frac{1}{(2\pi)^{\frac{M}{2}}} \exp\left\{-\frac{||\boldsymbol{t}-\boldsymbol{z}_i||^2}{2}\right\}, \tag{24}$$

and its FIM is obtained as $\mathbf{F}_{z_i} = \mathbf{I}$ [22].

The second is a multinomial distribution, which can be expressed in either of the following two forms [7]

$$p(\boldsymbol{t} \mid \boldsymbol{z}_i) = \left(1 + \sum_{k=1}^{M-1} \exp(z_{ki})\right)^{-1} \exp(\boldsymbol{z}_i^{\mathsf{T}}\boldsymbol{t}) \tag{25}$$

$$p(\boldsymbol{t} \mid \boldsymbol{z}_i) = \prod_{m=1}^{M} \left[\mathit{softmax}(\boldsymbol{z}_i)\right]_m^{[\boldsymbol{t}]_m} \tag{26}$$

where $\boldsymbol{z}_i = [z_{1i}, \ldots, z_{Mi}]^{\mathsf{T}}$, $\boldsymbol{t}$ is a one-hot vector, and

$$\left[\mathit{softmax}(\boldsymbol{z}_i)\right]_m = \frac{\exp(z_{mi})}{\sum_{k=1}^{M}\exp(z_{ki})} \tag{27}$$

is the softmax function. The derivative of the log-likelihood is obtained from (25) as

$$\nabla_{\boldsymbol{z}_i} \log p(\boldsymbol{t} \mid \boldsymbol{z}_i) = \boldsymbol{t} - \mathit{softmax}(\boldsymbol{z}_i)\,. \tag{28}$$

With $\mathcal{T} = \{[1, 0, \ldots, 0]^{\mathsf{T}}, \ldots, [0, \ldots, 0, 1]^{\mathsf{T}}\}$ being all possible outcomes, the calculation of (14) can be done by

$$\mathbf{F}_{z_i} = \frac{1}{M}\sum_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t} \mid \boldsymbol{z}_i) \nabla_{\boldsymbol{z}_i} \log p(\boldsymbol{t} \mid \boldsymbol{z}_i)\, \nabla_{\boldsymbol{z}_i} \log p(\boldsymbol{t} \mid \boldsymbol{z}_i)^{\mathsf{T}} \tag{29}$$

using (26) and (28).

3.3 Guided Direction Vector

This subsection describes a way to generate a guided direction vector $\delta\boldsymbol{\theta}_1$. Let us construct a model NN that simulates the ONN in an ordinary computer. The model network has the same structure as the ONN (Fig. 1) with unknown error parameters $\boldsymbol{\psi} = \{\boldsymbol{\zeta}, \boldsymbol{\gamma}\}$, where $\boldsymbol{\zeta}$ and $\boldsymbol{\gamma}$ are vectors containing all the attenuation-phase errors and angle errors, respectively. Let $\boldsymbol{f}^{(\mathrm{m})}$ be a function that the model network implements. We calibrate the error parameters $\boldsymbol{\psi}$ so as to minimize the discrepancy

$$e_{\mathcal{D}}(\boldsymbol{\psi}) = \frac{1}{D}\sum_{i=1}^{D} e(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi})\,, \tag{30}$$

$$e(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi}) = ||\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta}) - \boldsymbol{f}^{(\mathrm{m})}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi})||_2^2 \tag{31}$$

between the ONN output $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{\theta})$ and the model output $\boldsymbol{f}^{(\mathrm{m})}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi})$ for all the inputs $\boldsymbol{x}_i$ in the training dataset $\mathcal{D}$. The minimization can be performed by mini-batch stochastic gradient descent as

$$\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \eta^{(\mathrm{m})} \cdot \nabla_{\boldsymbol{\psi}} e_{\mathcal{B}}(\boldsymbol{\psi})\,, \tag{32}$$

$$e_{\mathcal{B}}(\boldsymbol{\psi}) = \frac{1}{B}\sum_{i=1}^{B} e(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi})\,. \tag{33}$$

Then, the gradient of the model loss with respect to the parameters

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}^{(\mathrm{m})}(\boldsymbol{\theta}),\ \ \ell_{\mathcal{B}}^{(\mathrm{m})}(\boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B} \ell(\boldsymbol{f}^{(\mathrm{m})}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\psi}), \boldsymbol{t}_i) \tag{34}$$

is calculated as a good candidate of a direction vector that is expected to approximate the true gradient $\nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}(\boldsymbol{\theta})$ of the ONN. Note that these two types of gradients, $\nabla_{\boldsymbol{\psi}} e_{\mathcal{B}}(\boldsymbol{\psi})$ and $\nabla_{\boldsymbol{\theta}} \ell_{\mathcal{B}}^{(\mathrm{m})}(\boldsymbol{\theta})$, can be calculated by backpropagation because the model network $\boldsymbol{f}^{(\mathrm{m})}$ is in the ordinary computer and not a black-box function.

The model gradient $\boldsymbol{g}$ is a biased approximation because a perfect calibration is practically impossible. Therefore, the direct use of $\boldsymbol{g}$ is not appropriate, and we instead follow the idea [21] of trading off the randomness and the model information with a hyperparameter $0 < \alpha \leq 1$. More specifically, we calculate a covariance matrix

$$\Sigma = \frac{\alpha}{N} \cdot \mathbf{I} + \frac{1-\alpha}{||\boldsymbol{g}||_2^2} \cdot \boldsymbol{g}\,\boldsymbol{g}^{\mathsf{T}} \tag{35}$$

and draw a guided direction vector $\delta\boldsymbol{\theta}_1$ from a distribution $\mathcal{N}(\mathbf{0}, \Sigma)$.

4 EXPERIMENTS

4.1 Settings

To demonstrate the effectiveness of LCNG (Sec. 3.2), we set up four ZO optimization methods, `ZO`, `ZO-NG`, `LCNG`, and `LCNG-I`, which did not utilize a model network. `ZO` [19] was based on the ZO gradient estimate (8) and parameter update (10). `ZO-NG` [33] was the combination of the ZO gradient estimate (8) and the natural gradient update (6). `LCNG` and `LCNG-I` were based on our new LCNG updates (12) and (13). The difference between the two methods was in the options for calculating the output FIM (14): `LCNG` used (29) and `LCNG-I` used $\mathbf{F}_{z_i} = \mathbf{I}$ (Subsubsection 3.2.2). Furthermore, to examine how helpful guided direction vectors from a calibrated model were (Sec. 3.3), we set up additional four methods `ZO-m`, `ZO-NG-m`, `LCNG-m`, and `LCNG-I-m`, where `-m` indicates the use of a model NN $\boldsymbol{f}^{(\mathrm{m})}$. We set the hyperparameters $\mu = 10^{-3}$ for (9) and (11), and $\alpha = 0.5$ for (35). All the methods were implemented using PyTorch [27] with some modules being accelerated by customized CUDA kernels.

We performed a classification task with cross-entropy loss using the MNIST handwritten digit database [18]. Although the cross-entropy loss corresponds to the multinomial distribution, we still used both `LCNG(-m)` and `LCNG-I(-m)` because the loss function and the FSD term in (4) could be specified independently. We employed a feed-forward ONN consisting of three Clements meshes as MZI

Table 2: Training losses. Every entry represents mean ± standard deviation of five runs. The best mean values in bold.

H	`ZO`	`ZO-NG`	`LCNG`	`LCNG-I`	`ZO-m`	`ZO-NG-m`	`LCNG-m`	`LCNG-I-m`
16	0.782 ± 0.023	0.737 ± 0.022	**0.712** ± 0.025	0.714 ± 0.010	0.714 ± 0.026	0.683 ± 0.014	**0.680** ± 0.017	0.684 ± 0.011
32	0.376 ± 0.018	0.332 ± 0.002	0.325 ± 0.012	**0.322** ± 0.011	0.353 ± 0.058	0.301 ± 0.009	0.293 ± 0.009	**0.292** ± 0.011
64	0.255 ± 0.005	0.225 ± 0.004	0.217 ± 0.005	**0.211** ± 0.005	0.419 ± 0.050	0.208 ± 0.004	0.205 ± 0.005	**0.195** ± 0.005
128	0.212 ± 0.005	out of memory	0.177 ± 0.006	**0.165** ± 0.006	0.846 ± 0.051	out of memory	0.175 ± 0.002	**0.163** ± 0.003

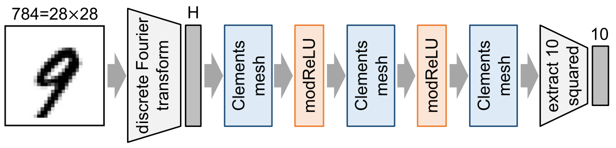

Figure 2: Feed-forward ONN recognizing handwritten digits

arrays and two modReLU layers as nonlinearity (Fig. 2). Let H be the dimensionality of the unitary matrix implemented by each of the meshes. We applied the discrete Fourier transform to a $784 = 28 \times 28$ pixel MNIST image and took the lowest H frequency bins as input. We examined four dimensionalities $H = 16, 32, 64, 128$. The number N of parameters $\boldsymbol{\theta}$ to be trained can be calculated as $N = 3 \times H(H-1) + 2 \times H$. All parameters $\boldsymbol{\theta}$ were initialized to zero. We set the number Q of direction vectors as $Q = H/2$ to compromise the gradient estimation accuracy and query efficiency. The mini-batch size was set at $B = 100$. The number of epochs for training was 100. Instead of vanilla SGD, we used the Adam optimizer [15] to mitigate the impact of magnitude variations of estimated gradients. We searched for the best learning rate η for `ZO` with each dimensionality H, and used the same learning rate for the other seven methods as well.

We simulated ONNs in an ordinary computer for the variety of dimensionalities H and therefore the MZI array sizes. For the component errors of ONNs, we set the attenuation-phase errors ζ and the splitting errors γ randomly by

$$\zeta = (1 - \sigma_{\zeta,r} r_{u1}) \exp[j \cdot \sigma_{\zeta,a}(2r_{u2}-1)], \ \gamma = \sigma_\gamma \cdot r_n, \qquad (36)$$

where r_{u1} and r_{u2} are random numbers sampled from a uniform distribution $[0, 1)$, and r_n is from a standard normal distribution $\mathcal{N}(0, 1)$. We examined the following error settings:

$$\sigma_{\zeta,r} = 10^{-3}\beta, \ \sigma_{\zeta,a} = 10^{-1}\beta, \ \sigma_\gamma = 10^{-2}\beta, \qquad (37)$$

where β controlled the amount of errors. The setting $\beta = 1$ corresponded to our estimate on an actual calibrated Clements mesh on silicon photonics. To simulate situations where perfect calibration was impossible, we limited the calibration capability of model networks. More specifically, we randomly selected about half of the total angle error parameters γ, and fix them to zero.

4.2 Results

Table 2 shows the training losses for all combinations of the eight methods and the dimensionalities H. The error setting was $\beta = 1$. We examined five independent runs to see statistical significance. For six methods related to natural gradient `ZO-NG(-m)`, `LCNG(-I(-m))`, we set $\lambda_F = 100$ to weight the FSD term. In general, all the six methods performed significantly better than the corresponding ordinary ZO methods `ZO(-m)`. Among them, `LCNG-I(-m)` performed the best except the smallest dimensionality $H = 16$. Figure 3 shows the convergence behaviors when $H = 64$ sampled from the five runs. We observe that `ZO-m`, which utilizes guided direction vectors

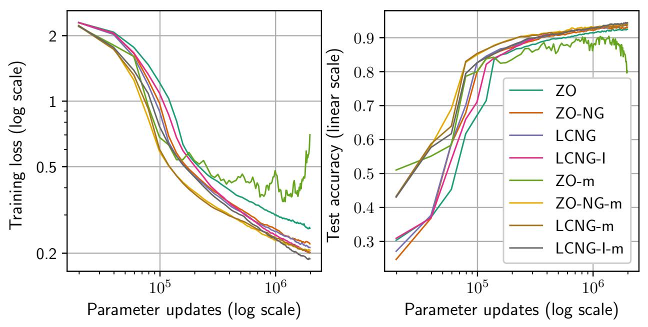

Figure 3: Convergence behaviors for $H = 64$: training losses and test accuracies along the number of parameter updates.

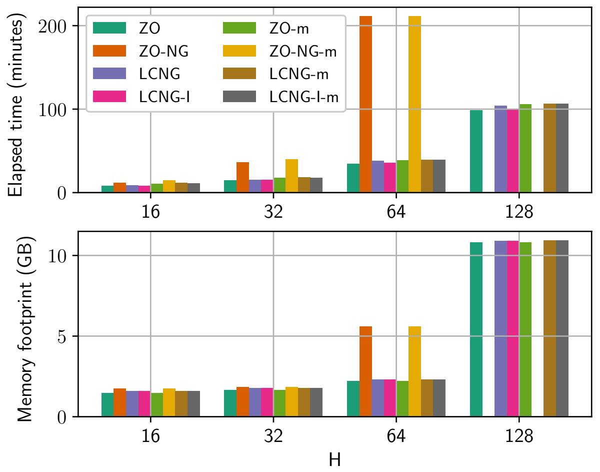

Figure 4: Elapsed time and memory footprint for training. `ZO-NG(-m)` with $H = 128$ ran out of memory (48 GB).

without taking care of the FSD term, did not converge well. In this sense, the effect of natural gradient becomes more prominent when we use guided direction vectors in a high dimensionality.

Figure 4 shows the elapsed time and memory footprint for training with NVIDIA RTX A6000. We see that those of `ZO-NG(-m)`, i.e., the existing combination [33] of ZO optimization and natural gradient, rapidly increased as the dimensionality H increased. And for $H = 128$, `ZO-NG(-m)` ran out of memory (48 GB). The proposed methods `LCNG(-I(-m))`, on the other hand, could be executed with very little overhead compared to the ordinary methods `ZO(-m)`.

So far, we have recognized that `LCNG-I(-m)` performed the best. Figure 5 shows the sensitivity of `LCNG-I(-m)` to the settings and situations. The upper plot shows the effect of the FSD weight λ_F with dimensionalities $H = 32, 64, 128$. The left most $\lambda_F = 0$ indicates that we did not care the FSD term and thus the results were very similar to those of the ordinary method `ZO(-m)`. We observe that the training losses were robustly minimized as long as $10^0 \le \lambda_F \le 10^4$

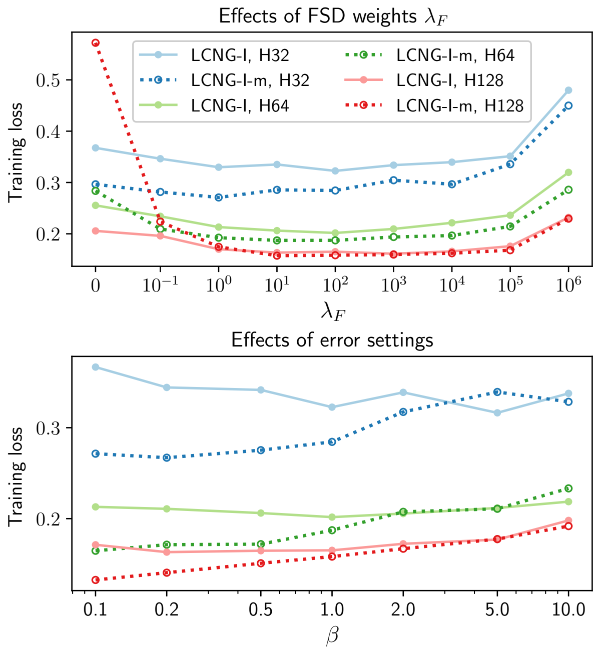

Figure 5: Effects of FSD weights λ_F (upper) and error settings (37) (lower) on training losses.

although the optimal λ_F depended on the situation. The lower plot shows the effects of error settings by varying β in (37). We observe that the results of `LCNG-I`, which did not use a model network, had no clear tendency. On the other hand, the results of `LCNG-I-m` clearly depended on the error settings, and when the fabrication variations became smaller ($\beta < 1$) than what we experienced ($\beta = 1$), the effectiveness of using a model network became more prominent.

5 CONCLUSION

To train ONNs based on ZO optimization, we have newly proposed LCNG, which takes care of the FSD term in (4). We have also proposed the use of a model NN to generate guided direction vectors. These two extensions have achieved efficient and stable ZO-based training (Table 2 and Fig. 3) of ONNs with the MZI array structures having unknown fabrication variations. We can enjoy the benefits of these proposals with little overhead, since the computational cost is almost the same as that of the ordinary ZO optimization (Fig. 4). Future work includes the verification of the proposed method on an actual silicon photonic device.

REFERENCES

[1] S. Amari. 1998. Natural gradient works efficiently in learning. *Neural Computation* 10, 2 (1998), 251–276.
[2] M. Arjovsky, A. Shah, and Y. Bengio. 2016. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*. PMLR, 1120–1128.
[3] F. Ashtiani, A. J. Geers, and F. Aflatouni. 2022. An on-chip photonic deep neural network for image classification. *Nature* 606 (2022), 501–506. https://doi.org/10.1038/s41586-022-04714-0
[4] J. Bae, P. Vicol, J. Z. HaoChen, and R. B. Grosse. 2022. Amortized proximal optimization. *Advances in Neural Information Processing Systems* 35 (2022), 8982–8997.
[5] S. Bandyopadhyay, A. Sludds, S. Krastanov, R. Hamerly, N. Harris, D. Bunandar, M. Streshinsky, M. Hochberg, and D. Englund. 2022. Single chip photonic deep neural network with accelerated training. *arXiv preprint arXiv:2208.01623* (2022).
[6] S. Banerjee, M. Nikdast, and K. Chakrabarty. 2023. Characterizing Coherent Integrated Photonic Neural Networks Under Imperfections. *Journal of Lightwave Technology* 41, 5 (2023), 1464–1479. https://doi.org/10.1109/JLT.2022.3193658
[7] C. M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.
[8] W. Bogaerts, D. Pérez, J. Capmany, D. A. B. Miller, J. Poon, D. Englund, F. Morichetti, and A. Melloni. 2020. Programmable photonic circuits. *Nature* 586 (2020), 207–216. https://doi.org/10.1038/s41586-020-2764-0
[9] L. Bottou. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 421–436.
[10] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley. 2016. Optimal design for universal multiport interferometers. *Optica* 3, 12 (2016), 1460–1465.
[11] M. Y.-S. Fang, S. Manipatruni, C. Wierzynski, A. Khosrowshahi, and M. R. DeWeese. 2019. Design of optical neural networks with component imprecisions. *Optical Express* 27, 10 (2019), 14009–14029.
[12] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT press.
[13] J. Gu, C. Feng, Z. Zhao, Z. Ying, R. T. Chen, and D. Z. Pan. 2021. Efficient on-chip learning for optical neural networks through power-aware sparse zeroth-order optimization. In *Proc. AAAI Conf. Artificial Intelligence*, Vol. 35. 7583–7591.
[14] J. Gu, Z. Zhao, C. Feng, W. Li, R. T. Chen, and D. Z. Pan. 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.
[15] D. P. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[16] F. Kunstner, P. Hennig, and L. Balles. 2019. Limitations of the empirical Fisher approximation for natural gradient descent. *Advances in Neural Information Processing Systems* 32 (2019).
[17] Y. LeCun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature* 521 (2015), 436–444. https://doi.org/10.1038/nature14539
[18] Y. LeCun and C. Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist
[19] S. Liu, P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney. 2020. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine* 37, 5 (2020), 43–54.
[20] K. D. G. Maduranga, K. E. Helfrich, and Q. Ye. 2019. Complex unitary recurrent neural networks using scaled Cayley transform. In *Proc. AAAI Conf. Artificial Intelligence*. 4528–4535.
[21] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein. 2019. Guided evolutionary strategies: Augmenting random search with surrogate gradients. In *International Conference on Machine Learning*. PMLR, 4264–4273.
[22] L. Malagò and G. Pistone. 2015. Information geometry of the Gaussian distribution in view of stochastic optimization. In *Proc. ACM Conference on Foundations of Genetic Algorithms XIII*. 150–162.
[23] J. Martens. 2020. New insights and perspectives on the natural gradient method. *The Journal of Machine Learning Research* 21, 1 (2020), 5776–5851.
[24] J. Martens and R. Grosse. 2015. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*. PMLR, 2408–2417.
[25] J. C. Mikkelsen, W. D. Sacher, and J. K. S. Poon. 2014. Dimensional variation tolerant silicon-on-insulator directional couplers. *Opt. Express* 22, 3 (Feb 2014), 3145–3150. https://doi.org/10.1364/OE.22.003145
[26] R. Pascanu and Y. Bengio. 2013. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584* (2013).
[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).
[28] M. Reck and A. Zeilinger. 1994. Experimental realization of any discrete unitary operator. *Phys. Rev. Lett.* 73, 1 (1994), 58–61.
[29] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, and M. Soljačić. 2017. Deep learning with coherent nanophotonic circuits. *Nat. Photonics* 11 (2017), 441–446.
[30] R. Tang, R. Tanomura, T. Tanemura, and Y. Nakano. 2021. Ten-Port Unitary Optical Processor on a Silicon Photonic Chip. *ACS Photonics* 8, 7 (2021), 2074–2080. https://doi.org/10.1021/acsphotonics.1c00419
[31] S. K. Vadlamani, D. Englund, and R. Hamerly. 2023. Transferable learning on analog hardware. *Science Advances* 9, 28 (2023), eadh3436. https://doi.org/10.1126/sciadv.adh3436
[32] I. A. D. Williamson, T. W. Hughes, M. Minkov, B. Bartlett, S. Pai, and S. Fan. 2020. Reprogrammable Electro-Optic Nonlinear Activation Functions for Optical Neural Networks. *IEEE Journal of Selected Topics in Quantum Electronics* 26, 1 (2020), 1–12. https://doi.org/10.1109/JSTQE.2019.2930455
[33] P. Zhao, P.-Y. Chen, S. Wang, and X. Lin. 2020. Towards query-efficient black-box adversary with zeroth-order natural gradient descent. In *Proc. AAAI Conf. Artificial Intelligence*, Vol. 34. 6909–6916.
[34] H. Zhou, Y. Zhao, X. Wang, D. Gao, J. Dong, and X. Zhang. 2020. Self-configuring and reconfigurable silicon photonic signal processor. *ACS Photonics* 7, 3 (2020), 792–799.