2024 IEEE 31st International Conference on High Performance Computing, Data, and Analytics (HiPC) | 979-8-3315-0909-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/HIPC62374.2024.00020

Training Photonic Mach Zehnder Meshes for Neural Network Acceleration

Andy Wolff and Avinash Karanth
School of Electrical Engineering and Computer Science
Ohio University Athens, OH, 45701
Email: aw415517@ohio.edu, karanth@ohio.edu

***Abstract*—Photonic neural networks enable faster and energy-efficient inferences for deep neural network implementations when compared to electrical counterparts. Prior training methods for photonic neural networks modify the network or decompose the inputs or weights to simplify the training, however that could impact accuracy. Electrical training uses backpropagation that needs to partially differentiate each of the weights with respect to the loss function on a number of examples to achieve high accuracy. While the traditional backpropagation algorithm can be applied to electrical networks, it is difficult to directly apply to photonic neural network with Mach-Zehnder Interferometer (MZI) meshes that have different parameters (frequency and phase). In this paper, we implement the traditional backpropagation algorithm to train the parameters of the MZI meshes with gradient descent. Parameters of the mesh are trained by implementing a modified backpropagation algorithm on the hardware, and those operations may be performed on device. The implementation does not modify the circuits or add any additional lasers and therefore, self-trains the MZI to achieve the desired accuracy and achieve O(n^2) speedup over electrical methods. We test our training algorithms on MNIST datasets and show that our method improves the training when compared to state-of-the-art electrical backpropagation.**

I. Introduction

Deep learning is well established for solving complex problems [7] in many applications such as autonomy, language models, video and image recognition and several others. With demands of higher accuracy, neural models have continuously evolved to become complex networks that have a significant number of weights, inputs, connections and layers which impact both inference and training time [7]. Taken together, large models put tremendous pressure on the computation and memory requirements to meet accuracy and throughput demands of applications. Electronic accelerators have implemented several optimizations for improving throughput and reducing the computation and memory complexity by applying quantization, exploiting sparsity, pre-processing data, and other techniques. However, training neural network models consume both power and latency in electrical models which becomes challenging with large models.

Emerging technology such as silicon photonics has the potential to lower the energy cost while allowing parallel computations which could result in both higher energy-efficiency and throughput. Photonics has proven to substantially improve the communication bottleneck with improved $performance-per-watt$ for long and short distances (within chassis or board) for inter-processor communication. The intrinsic properties of light also make photonics a potential contender for parallel compute tasks such as linear algebra and matrix multiplication. Several research groups from both academia and industry have proposed photonic neural network acceleration to scale deep neural network (DNN) inference in terms of energy efficiency and throughput, achieving more than an order of magnitude latency improvement over electronic accelerators. For example, Clement's or Reck's arrangement of Mach-Zehnder interferometers can represent any general matrix multiplication [14] [2]. Each MZI within the grid will need to be trained (amplitude and phase) to compute the linear algebra transformations or matrix multiplication.

As MZIs have two parameters, phase and amplitude, prior work have shown on how to adjust/correct the phase/amplitude of MZIMs to overcome any imperfections in device fabrication when meshes of MZIs are connected [11]. Other approaches have shown techniques on how to train the non-linear device parameters with a numerical model, and then to use this numerical model to train the analog devices as in Zheng et.al. [20]. Such methods allows for the gradient to be computed more accurately than existing training methods, but conversions from a numerical model to photonic device parameters may make this combined numerical and photonic approach more expensive than a purely numerical training. Such training also does not take into account any device imperfections. Training a photonic device without access to a numerical representation requires additional information about the device's behavior than the result of an inference [6]. Hughes et. al. implement a training algorithm that takes samples of each MZI's output and uses this result over multiple passes to compute the gradient for each MZI parameter collectively [6]. This method allows for a MZI mesh to be trained by itself, but it requires significant modifications and may suffer in non-ideal cases [3] [6]. Conversely, Zhang et. al. approach this problem by using an algorithm that does not require the numerical device model to be known with a genetic algorithm [19]. This allows the device to be trained over time without measuring how each parameter impacts individually, but requires a family of candidate parameters to be created, which implies that the number of calculations increases significantly compared to inferences.

In this paper, we propose a method to train any MZI-based photonic neural network (PNN) using an additional dot product circuit. A system such as the MZI mesh in Figure 1 may be be used to implement the training of a PNN

2640-0316/24/$31.00 ©2024 IEEE
DOI 10.1109/HiPC62374.2024.00020

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

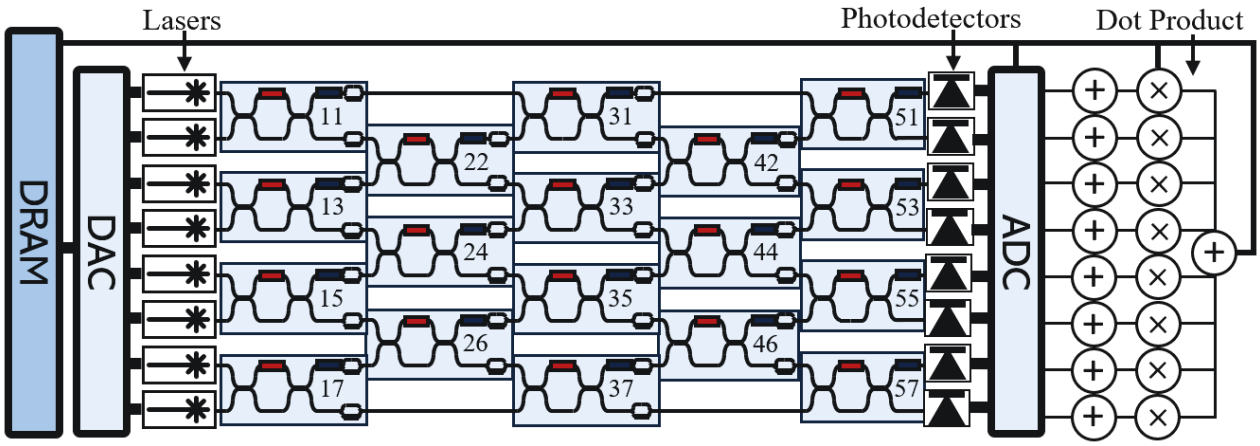

Fig. 1. **Model of MZI mesh and peripherals including lasers, photodetectors, DACs, ADCs and dot products required for training of mesh parameters. Numbering of MZI's corresponds to the index of the MZI and its parameters.**

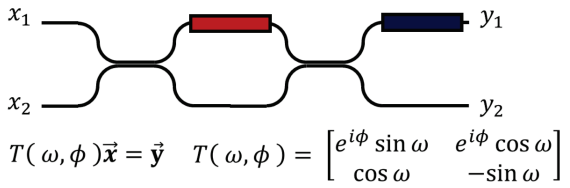

Fig. 2. **Model of MZI and its effect on the electric field component of input signals.**

through this algorithm. We accomplished this by modifying backpropagation to a form that may be implemented on the mesh using only the summation of partial sums, and the addition of constant values to MZI parameters. This modified backpropagation allows the gradient with respect to device parameters to be computed exactly, allowing a gradient descent approach adapted to PNNs. This allows for an entire layer's gradient to be calculated after a number of passes equal to the number of parameters plus two times the number of inputs. Parameters of the mesh are trained by implementing a modified backpropagation algorithm on the hardware, and those operations may be performed on device. The implementation does not modify the circuits or add any additional lasers and therefore, self-trains the MZI to achieve the desired accuracy and achieve O(n^2) speedup over electrical methods. We test our training algorithms on MNIST and CIFAR-10 datasets and show that our method improves the training when compared to state-of-the-art electrical backpropagation. The major contributions of this work as follows:

- **Gradient Descent:** Implements gradient descent in a number of passes that scales linearly with the number of layer parameters. This improves the efficiency of the proposed training algorithm.
- **Unmodified Mesh:** Allows for the MZI mesh to be trained without significantly modifying its circuits or any adding additional lasers or photodetectors beyond what is required for inferences. This allows on-device training which simplifies the architecture for both training and inference.
- **Energy-efficient Training:** Since the training is on the device parameters rather than comparison to a numerical representation, the implementation is similar to those used for static matrices, and we can avoid costly calibrations. Our results indicate that the training algorithm is comparable to both existing photonic and digital training results in terms of accuracy and throughput.

II. Background

A. MZI Mesh Architecture

The effect of a single Mach-Zehnder interferometer (MZI) on the electric field may be modeled by two parameters, phase and amplitude, as described in Figure 2. MZI's may be arranged to form a multiport interferometer in an exponentially large number of configurations, including those described by Clements et. al. [2] and Reck et. al. [14]. The effects of any mesh on electric field can be described as a matrix multiplication:

$$\mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j) =$$

$$\begin{bmatrix} 1 & 0 & \dots & & & & \dots & 0 \\ 0 & 1 & \dots & & & & \dots & 0 \\ \vdots & & \ddots & & & & & \vdots \\ \vdots & & & e^{i\phi_j}\sin\omega_j & e^{i\phi_j}\cos\omega_j & & & \vdots \\ \vdots & & & \cos\omega_j & -\sin\omega_j & & & \vdots \\ \vdots & & & & & \ddots & & \vdots \\ 0 & \dots & & & & \dots & 1 & 0 \\ 0 & \dots & & & & \dots & 0 & 1 \end{bmatrix}$$

where ϕ_i and ω_i denote the MZI parameters in Figure 2 for each MZI device in the mesh, and n_j and m_j denote the lanes of the mesh that connect the MZIs. While the meshes utilized here and described by Clements et. al. only connect adjacent lanes [2], the training algorithm introduced in this paper may

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

be applied to any arrangement of MZIs. The arangements of MZI introduced by Clements et. al. may represent any unitary matrix [2], and this property is desirable for using the multiport interferometers in photonic neural networks.

Since we cannot easily measure the electric field directly, a PNN computed on an MZI mesh measures the optical power with photodiodes. This means that the layers compute the magnitude squared of the result of the matrix multiplication. This gives us for every layer:

$$\mathbf{M}^{(l)} = \prod_{j=1} \mathbf{T}(n_j^{(l)}, \phi_j^{(l)}, \omega_j^{(l)})$$

$$\mathbf{z}^{(l+1)} = |\mathbf{M}^{(l)}\mathbf{a}^{(l)}|^2 + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l+1)} = f(\mathbf{z}^{(l+1)})$$

where l denotes the particular layer computed, $\mathbf{z}^{(l+1)}$ is the output of the matrix multiplication plus the bias for layer l, and $\mathbf{a}^{(l+1)}$ denotes the output of the layer l where f is its activation function. Note that for the purposes of this paper, the magnitude operator always refers to the element-wise complex magnitude. This result, $\mathbf{a}^{(l+1)}$ may be used as the input to the next layer and this may be used to compute inferences through the entire network. This model allows for an entire PNN to be computed on a MZI mesh such as the one described in Figure 1.

B. Backpropagation Algorithm

We briefly review the backpropagation algorithm since we propose to modify some of the steps to compute the gradients with supervised training. Supervised training in this case consists of minimizing an error function, denoted here as $\mathbf{J}$, which evaluates the error as the residual squared, although this process works for any differentiable error function. Next, the gradient of the error function is taken with respect to every network parameter, and this value is used to update each parameter. For a set of parameters, $\boldsymbol{\omega}$, the updated weights are:

$$\omega_i(t+1) = \omega_i(t) - \alpha\frac{\partial \mathbf{J}}{\partial \omega_i}$$

$$\boldsymbol{\omega}(t+1) = \boldsymbol{\omega}_i(t) - \alpha\nabla_\omega \mathbf{J}$$

For the last layer, the gradient with respect to its parameters is trivial, but for the preceding layers, this task is less straightforward. A method for generalizing this process uses an intermediate vector $\boldsymbol{\delta}^{(l)}$, to track the derivative of each layer's outputs with respect to the loss. This can then be multiplied by the derivative of a layer with respect to its parameters to get the final gradient:

$$\boldsymbol{\delta}_i^{(l+1)} = \frac{\partial \mathbf{J}}{\partial \mathbf{z}_i^{(l+1)}}$$

$$\frac{\partial \mathbf{J}}{\partial \omega_j^{(l)}} = \sum_{i=1}^{|\mathbf{a}|} \boldsymbol{\delta}_i^{(l+1)} \frac{\partial \mathbf{z}_i^{(l+1)}}{\partial \omega_j^{(l)}}$$

In order to do this for every layer l, the values of the $\boldsymbol{\delta}^{(l)}$ may be found for every layer recursively from the output using the chain rule of partial derivatives:

$$\boldsymbol{\delta}^{(L)} = (\mathbf{a}^{(L)} - \mathbf{y}) \odot f'(\mathbf{z}^{(L)})$$

$$\boldsymbol{\delta}^{(l)} = f'(\mathbf{z}^{(l)}) \odot \sum_{i=1}^{|\delta^{(l)}|} \boldsymbol{\delta}_i^{(l+1)} \frac{\partial \mathbf{z}_i^{(l+1)}}{\partial \mathbf{a}^{(l)}}$$

The algorithm for computing the gradient using this method is straightforward. Calculate the vector $\boldsymbol{\delta}^{(l)}$ for every layer, l, and use it to compute the gradient and update for each layer's parameters. This process is then repeated until training is completed according to the version of backpropagation that is being applied.

III. Proposed Backpropagation Training Algorithm for PNN

In order to implement gradient descent on the networks described in Figure 1, we need to compute the gradient with respect to each mesh parameter and increment it in the form:

$$\boldsymbol{\phi}(t+1) = \boldsymbol{\phi}(t) - \alpha\nabla_\phi \mathbf{J}$$

$$\boldsymbol{\omega}(t+1) = \boldsymbol{\omega}(t) - \alpha\nabla_\omega \mathbf{J}$$

for parameter vectors $\boldsymbol{\phi}$ and $\boldsymbol{\omega}$. Utilizing our derivative of the output, we find the gradient as the dot product of the intermediate vector $\boldsymbol{\delta}^{(l)}$ and the derivative of the mesh itself with respect of its parameters.

Finding the values of both of these vectors is challenging because the mesh can only measure the modulus squared of its matrix multiplication, which removes both the complex component of the result, and additionally implies the mesh can only return positive values, whereas both $\boldsymbol{\delta}^{(l)}$ and the derivative of $\mathbf{z}^{(l+1)}$ may be signed.

Computing Gradient: In order to compute the gradient, we compute two mesh iterations for every MZI parameter, and take the difference of the result as done in Figure 3 for the derivative of the MZI in the first row, third column. For each $\boldsymbol{\omega}_i$, we add a constant value of $\pi/4$ to its value, and modulate channels that do not connect to the MZI containing the target parameter by a factor of $1/\sqrt{2}$ and store the result of the mesh in an intermediate vector $\mathbf{q}$. Next, we repeat but instead subtract the target parameter by $\pi/4$ and store this second result in the intermediate vector $\mathbf{p}$. The factor of $1/\sqrt{2}$ can be accomplished by the addition of MZM's between any connecting MZI's. Since the multiplicative effect of this MZM needs to be only 1 or $1/\sqrt{2}$, its size can much smaller than would be required otherwise, as its state is effectively binary. As can be seen in the full algorithm proof in the appendix, the difference of $\mathbf{q}$ and $\mathbf{p}$ is exactly equal to the derivative of $\mathbf{z}^{(l+1)}$ with respect to the target parameter. This difference can be computed by a PE connected to each output of the mesh. These two steps need to be repeated for every parameter, ω_i. Figure 3 shows the calculation for ω_{31} by adding and subtracting $\pi/4$. This step needs to be implemented for each of MZIs.

The process is continued for every ϕ_i in the mesh in the manner described in Figure 3. This version is slightly simpler, adding $\pi/6$ to the target parameter, and storing the mesh result in the vector $\mathbf{u}$. Then again we compute the layer subtracting $\pi/6$ from the target parameter and store it in $\mathbf{v}$. Finally,

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

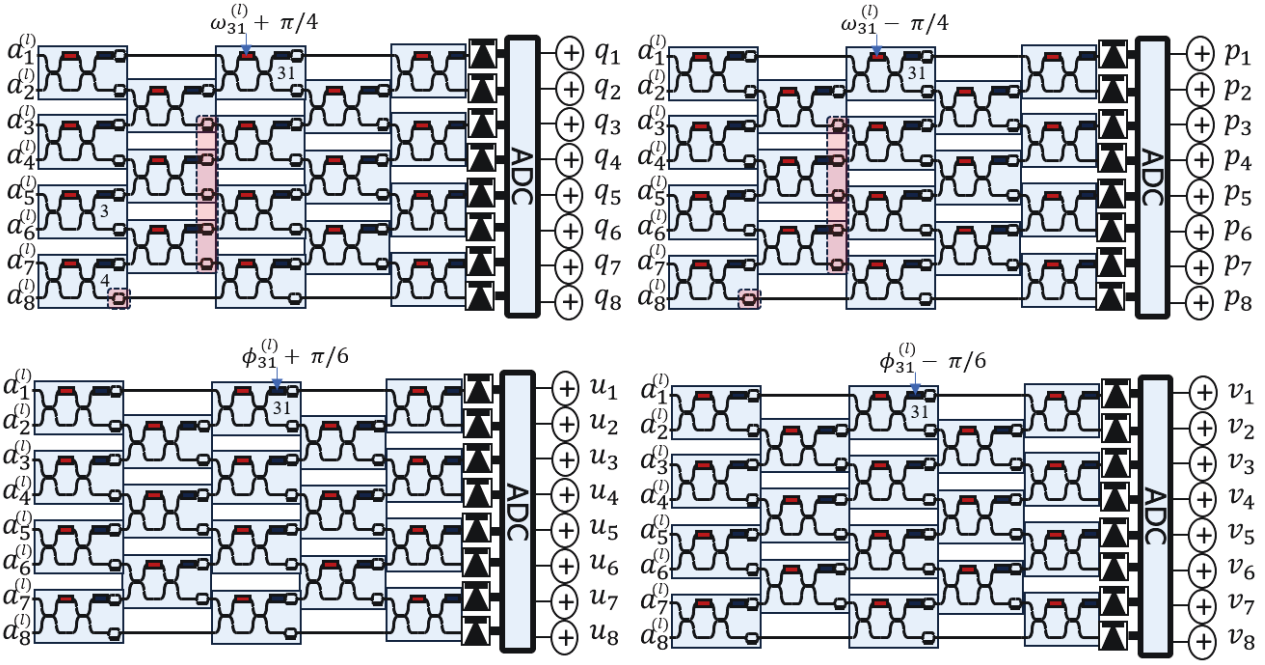

Fig. 3. **(a) and (b) Method for taking derivative of ω_i. Difference of two passes with an offset of a constant value to the target weight. Parallel paths must be modulated by a factor of $\frac{1}{\sqrt{2}}$, indicated at the sections highlighted red. (c) and (d) Corresponding process for derivative of ϕ_i. For both processes, the derivative is taken with respect to paramters of the MZI in index (3,1) and all other mesh parameters are unchanged.**

the difference of the intermediate vectors **u** and **v** is exactly equal to the derivative of the output with respect to the target parameter $\boldsymbol{\phi_i}$. These two steps need to be repeated for every parameter, ϕ_i. Figure 3 shows the calculation for phi_{31} by adding and subtracting $\pi/6$. This step needs to be implemented for each of MZIs.

Both of these methods allow the computing of the gradient by exploiting the fact that for any complex matrix **M**, and for any complex vector, a, and any MZI, the derivative of the modulus squared of their product can be computed by only modifying the MZI and possibly inserting the diagonal matrix D with the following identity:

$$\frac{\partial}{\partial \boldsymbol{\omega}}|MT(\omega,\phi)\mathbf{a}|^2 = |MT(\omega+\frac{\pi}{4},\phi)D\mathbf{a}|^2 - |MT(\omega-\frac{\pi}{4},\phi)D\mathbf{a}|^2$$

$$D_{i_k} = \begin{cases} 1 & k = n_i, k = m_i \\ \frac{1}{\sqrt{2}} & else \end{cases}$$

$$\frac{\partial}{\partial \boldsymbol{\phi}}|MT(\omega,\phi)\mathbf{a}|^2 = |MT(\omega,\phi+\frac{\pi}{6})\mathbf{a}|^2 - |MT(\omega,\phi-\frac{\pi}{6})\mathbf{a}|^2$$

which yields

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \boldsymbol{\omega}_i} = \mathbf{q} - \mathbf{p}$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \boldsymbol{\phi}_i} = \mathbf{u} - \mathbf{v}$$

This allows for the derivatives of the output with respect to the target parameter vector to be computed in two times as many steps as there are parameters to take the derivative. This also means that in both cases the inputs to the layer are exactly what they were on the inference step in all stages of this process. The derivative of the bias vector is unmodified from standard backpropagation and is trivially proven to be one if the output index matches the bias parameter index and zero otherwise.

Finally once the derivative of the vector $\mathbf{z}^{(l+1)}$ is computed, the final derivative of error with respect to the parameter is computed by taking the dot product of the our result from the previous step with $\boldsymbol{\delta}$ as discussed already.

$$\frac{\partial \mathbf{J}}{\partial \phi_j^{(l)}} = \sum_{i=1}^{|\mathbf{a}|} \frac{\partial \mathbf{J}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial \phi_j^{(l)}} = \sum_{i=1}^{|\mathbf{a}|} \boldsymbol{\delta}_i^{(l+1)} \frac{\partial \mathbf{z}_i^{(l+1)}}{\partial \phi_j^{(l)}}$$

$$\frac{\partial \mathbf{J}}{\partial \omega_j^{(l)}} = \sum_{i=1}^{|\mathbf{a}|} \frac{\partial \mathbf{J}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial \omega_j^{(l)}} = \sum_{i=1}^{|\mathbf{a}|} \boldsymbol{\delta}_i^{(l+1)} \frac{\partial \mathbf{z}_i^{(l+1)}}{\partial \omega_j^{(l)}}$$

Once each layer is computed, the value for the preceeding layer's $\boldsymbol{\delta}^{(l)}$ may be calculated by following the method described in Figure 4. It describes how in three steps a single value of the $\boldsymbol{\delta}$ vector may be computed first by setting the input to all input channels except the target index to zero, and storing the result at each output channel in the temporary vector **x**. The second step is to repeat the first doing the opposite, setting only the target parameter to zero and storing the result in the temporary vector **y**. The third and final step is to take the difference of the values of **x** and **y** and also to add the result from the multiplication from the inference step, and then take the dot product of the resulting vector with the $\boldsymbol{\delta}^{(l)}$ from the

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

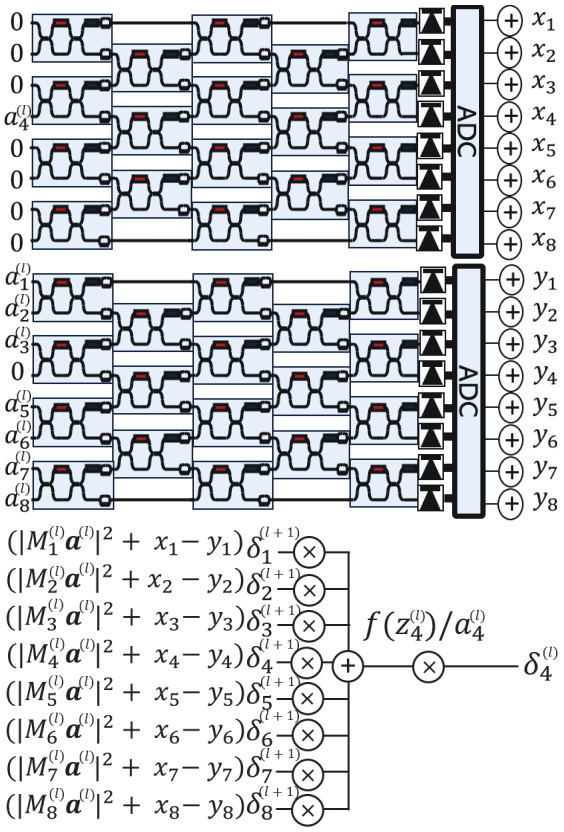

Fig. 4. **Method for obtaining δ_i. Difference of two passes with a_i set to zero for the first, and all of a except a_i set to zero. These are added to the mesh results from the inference step to obtain the δ_i element for each layer. Finally, the dot product of the summation and the next layer's δ gives the current layer's value for that index. Finally, the result is multiplied by the derivative of the activation function and divided by the a_i.**

current layer to get a single scalar value, which is multiplied by the activation function's derivative and the inverse of the result from the inference step. This gives the value of the vector $\boldsymbol{\delta}^{(l-1)}$ for the preceding layer and allows the gradient of that layer to be computed. The algorithm depicted here and in Figure 4 functions by exploiting the following identity:

$$\delta_j^{(l-1)} = \frac{f'(z_j^{(l)})}{a_j^{(l)}} \boldsymbol{\delta}^{(l)} \cdot [|M^{(l)}\mathbf{a}^{(l)}|^2 + |M^{(l)}\mathbf{s}|^2 - |M^{(l)}(\mathbf{a}^{(l)} - \mathbf{s})|^2]$$

$$\mathbf{s}_k = \begin{cases} \mathbf{a}_j^{(l)} & k = j \\ 0 & else \end{cases}$$

which yields

$$\delta_j^{(l-1)} = \frac{f(z_j^{(l)})}{a_j^{(l)}} \boldsymbol{\delta}^{(l)} \cdot (\mathbf{x} - \mathbf{y})$$

Key Insight: With both of the stages repeated for each layer, the gradient of every parameter in the network will be calculated. If one calculates the number of MZI runs, it will will be two times the number of parameters plus two times the number of inputs. We calculate **q**, **p**, **u** and **v** vectors for ω_i and ϕ_i for each MZI. Then **x** and **y** vectors are calculated by setting one of the inputs to zero and then setting the remaining inputs to zero as discussed previously. Now, both Clements et. al. and Rick et. al. could also be used to train the MZIs using their proposed decomposition method. The proposed decomposition method relies on two important properties of the $T_{m,n}$ matrices. First, for any given unitary matrix **U**, there are specific values of ω and ϕ that make any target element in row m or n of matrix $T_{m,n}$U zero [14]. Second, any target element in column n or m of **U** can also be nulled by multiplying U from the right by a $T_{m,n}^{-1}$. As seen in [2], for a 5 $\times$ 5 case, nulling elements of **U** one by one in such a way that every $T_{m,n}$ and $T_{m,n}^{-1}$ matrix used in the process completely determines both the reflectivity and phase shift of **one** beam splitter and phase shifter. The sequence of $T_{m,n}$ and $T_{m,n}^{-1}$ matrices must both correspond to the desired order of beam splitters in the interferometer and guarantee that the nulled elements of U are not affected by subsequent operations. In the Reck decomposition, the entire matrix can be nulled using either only $T_{m,n}$ matrices or only$T_{m,n}^{-1}$ matrices while still making sure nulled elements of **U** are not affected by subsequent operations. Therefore, the nulling of elements has to be implemented for each beam splitter for each parameter. This increases the costs for Clement's and Rick's approaches compared to our proposed approach. The total number of mesh iterations for both of these stages combined will be equal to two plus the number of inputs squared minus the number of inputs. Given that there are intermediate values computed after every layer, the output of each layer must be held in memory each epoch. Moreover, the result for the mesh before the bias is added and activation must also be stored in memory. This is true for other training methods with photonics [6] where the constant parameter values are calculated and then utilized at the end. These processes collectively allow for the gradient with respect to every parameter in the network to be computed, and once this is done, the gradient descent may occur as it would via any other method.

IV. Performance Evaluation

In this section, we evaluate the performance of the proposed training algorithm on MZI meshes. We first describe the training of the MZI meshes and then evaluate the latency and energy costs for different deep neural network models such as VGG-16 and MobileNet. We also compare the MZI meshes with state-of-the-art electrical and photonic accelerators.

A. Training Evaluation

We numerically evaluate the training method with our own simulator built in C. We extend the simulator to include the MZI meshes with the parameters (ω and ϕ) for training. We test the simulator with the proposed training method and compare against existing backpropagation methods. All evaluation with different fixed, varying training rates for MNIST and Imagenet datasets. We run the training both for numerical method and photonic models for different epochs to evaluate the training performance.

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

The photonic neural network was modeled by considering all the optical component losses and modeled using DSENT [18] tool. This tool captures losses for various components including external laser, losses through MZIs, waveguide losses and electrical frontend and backend circuitry (ADCs/DACs, TIA, etc). The overall power consumption Ptotal includes three parts: laser power Plaser, power consumption of transmitting circuitry PTX, and power consumption of receiving circuitry PRX. This was carefully calculated for each pass through the MZI meshes (and also for different size of MZIs). This provided the inference energy and then we calculated the gradient costs for each layer. We have calculated the energy consumption for each of our photonic devices (losses, external laser) and then modeled the inference and gradient cost.

This method for training MZI architectures was tested against conventional training methods such as backpropagation with numerical matrices, and its performance was evaluated. Due to the relatively small space of angle values for each MZI, it can increase performance substantially to normalize the angle increments. This improves performance by reducing the likelihood of increments beyond the target. Alternatively, a sufficiently low learning rate can approximate this effect without the additional calculations to normalize the weight increment vectors.

With these optimizations in mind, the algorithm is trained against MNIST and compared to conventional training methods. The training consists of samples of the MNIST dataset, and is trained for up to 100 epochs for most of the design space exploration. Various methods of backpropagation are applied, such as fixed learning rate, variable learning rate, and others to test the effectiveness of the proposed approach. One key training method is to normalize the increment for layer parameters. What this method entails is taking the vector of the gradient, and if the euclidean norm exceeds a threshold, to divide the vector by a scalar to reduce its norm to the target threshold. This is particularly applicable for this case as the space of possible angle values for phase-shifters is very small, and excessively large increments are unlikely to converge. With neural networks of identical size, the training rates are comparable on average, with the accuracy of the photonic neural network converging at a slightly reduced rate, with the converged values being roughly equivalent on average, as can be seen in Figure 5.

For a fixed learning rate of 0.02 the parameterized gradient descent for the MZI mesh converged slightly more slowly than the purely numeric model, but it converged at a roughly comparable rate. After 100 epochs the numeric model only predicted the incorrect result 15% of the time, and the photonic model had an error rate of 21% on the same verification set. The second result in Figure 5 models a learning rate that is inversely proportional to time for training. Here the results are very similar, with both models being correct 88% of the time. The space of angle values is very small, so smaller weight increments typically improve accuracy whenever the error is high, as the high error of initial results can cause angle values to overshoot ideal values in many cases. Finally, the last result in Figure 5 is the best results for both the numeric and photonic model analysis. In this sample of the dataset, the best method for the numeric model is the inversely proportional learning rate, and the best method for the photonic model is one with smaller, or normalized increments.

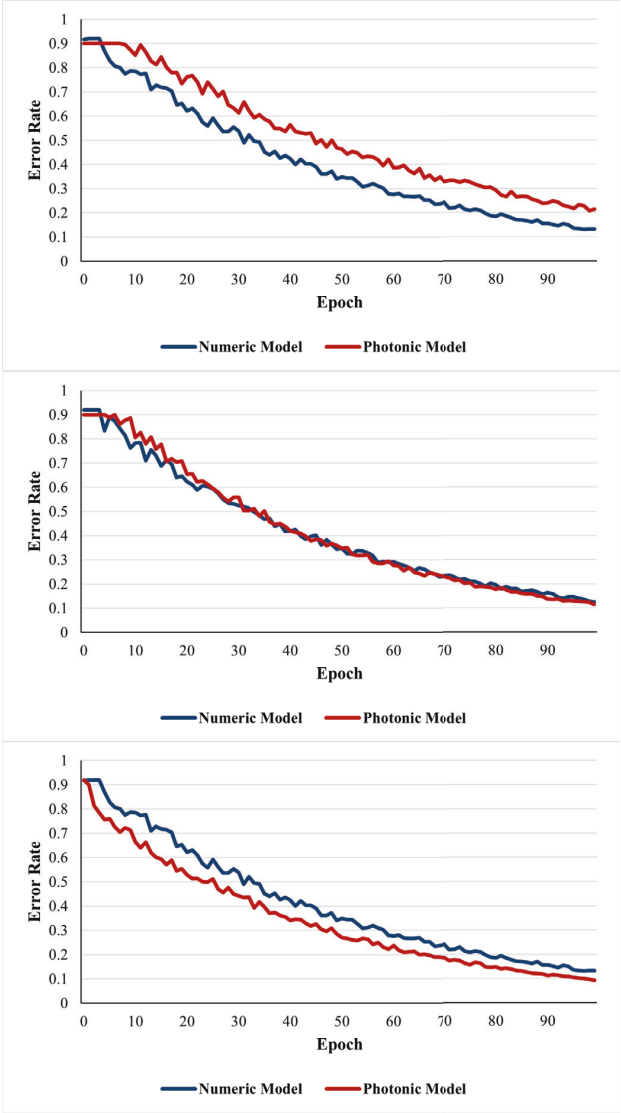

Fig. 5. **(a) Training on random sample sets from MNIST dataset with fixed learning rate of 0.02 for both numeric and photonic models. (b) Training with learning rate that scales inversely each epoch. Training where the best hyperparemeters for both numeric and photonic models are used seperately. In this case the best version is the inversely scaling learning rate for the numeric model, and the normalized increment model for the photonic model.**

Additionally, the training algorithm is implemented on a MZI mesh based implementation of ResNet18, and trained on the Imagenet dataset. For this larger dataset, the differences are much less pronounced, while individual training iterations take different paths, on average they converge at essentially the same rate, as can be seen in Figure 6.

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

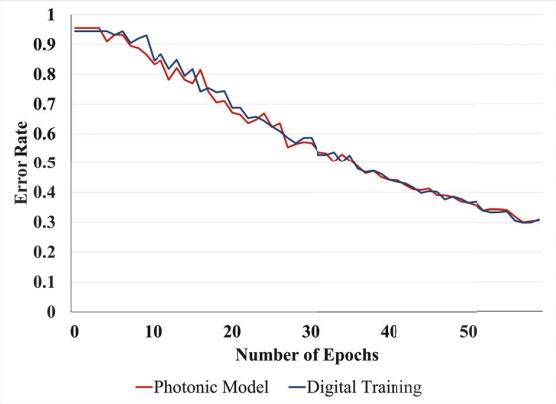

Fig. 6. **Training error rate for imagenet dataset. Using batch size of 200 for both, decreasing learning rate for both numeric and photonic training, and normalized increments for the photonic model as well. In both cases using the best particular learning rates. Error rates are averaged from 5 randomly reordered training passes of the dataset. Both models used a numeric or photonic implementation of ResNet18**

TABLE I
ENERGY COST PARAMETERS FOR MZI AND LASER COST, AND LOSS PARAMETERS FOR MZIS AND LINKING WAVEGUIDES

Layer Parameters	Value
MZI Loss	2 dB
MZI Energy Cost [12] [4]	0.45-5pJ / Op
Waveguide Loss	0.1 dB/MZI
Laser Cost	∼7.93 W

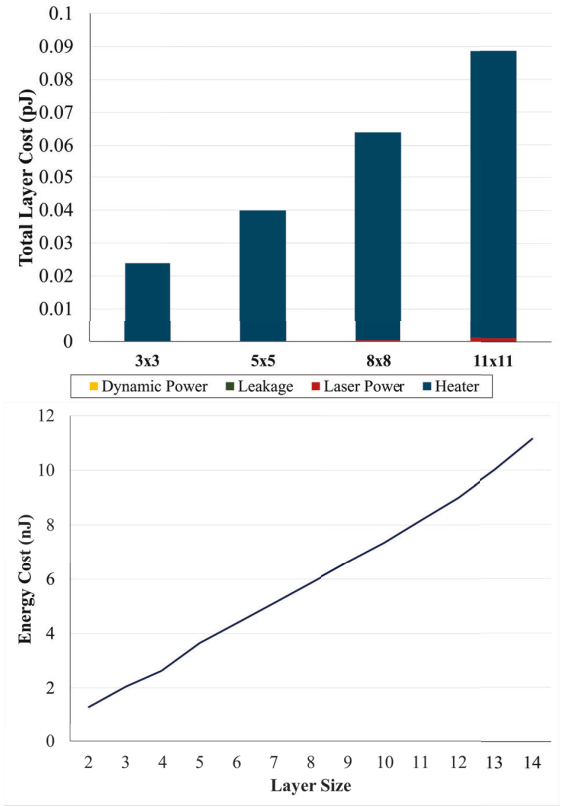

Fig. 7. **(a) Energy consumption for the inference stage for a layer of given size. Heaters make up an overwhelming amount of the power consumption. (b) Inference cost for for a variety of integer layer sizes.**

B. Energy and Latency Estimations

The performance of the system was evaluated both for inference and gradient descent use cases. First circuits were modeled of a series of MZI sizes, each along the algorithm prescribed in Clements et. al. in order to minimize optical depth [2].

The power loss is measured for each size and in this conservative model the loss is calculated to increase by approximately 2 decibels for every increase in optical depth. The energy consumption of the lasers is modeled using DSENT [18] from the data on signal losses obtained from the circuit model. The resulting component parameters are in Table I. We must factor in the fact that for every laser source, there are photodiodes at every possible output it must reach, and again the power consumption is larger for these analog signals than for purely digital sources. It also accounts for the dynamic power consumption for control devices, analog-to-digital converters (ADCs) and various small miscellaneous power drains, but these remain a relatively insignificant contribution to the total power cost.

In order to model the total power consumption for each layer, we must account for the cost of the circuits that control the phase shifters. For the purposes of phase-shifter settings, the matrices the layers compute are static because the phase shifters are controlled by the parameters directly rather than calibrated to fit some external matrix. The phase shifter power consumption varies depending on the system, ranging from 0.45 pJ to 5 pJ per operation for this system [4] [12]. We will be using the higher power consumption value for the modelling of this system. Note that this method of setting a MZI mesh to a matrix is both significantly more energy efficient and affords a higher bandwidth than setting the MZI to an external weight matrix rather than predefined parameters [12]. We break the power consumption into dynamic power, leakage, laser power and heater costs for different sizes of MZI (3×3, 5×5, 8×8 and 11×11). The energy consumption for inference grows linearly for larger sizes of the MZI as seen in the Figure 7(a) with the heaters needed to stabilize the lasers consuming maximum energy. As can be seen in Figure 7(b), the energy cost for the inference step grows roughly linearly as the layer size increases. This is because the dominant source of power consumption is the heaters for modulating the lasers, and the number of lasers grows linearly. The power consumption is roughly 8 mW per laser for all tested bandwidths, as cited in Table I. It can also be seen in Figure 7(b) that as the laser power consumption increases, it is a small contribution to the total power consumption.

Once the cost of inference is known, determining the energy cost of the gradient step is a simple matter. The gradient is taken by using the mesh repeatedly. Calculating the value for $\boldsymbol{\delta}$ and the gradient for each layer can be done by the methods described in the previous sections. The power consumption for each step of this process is the same as it is in the

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

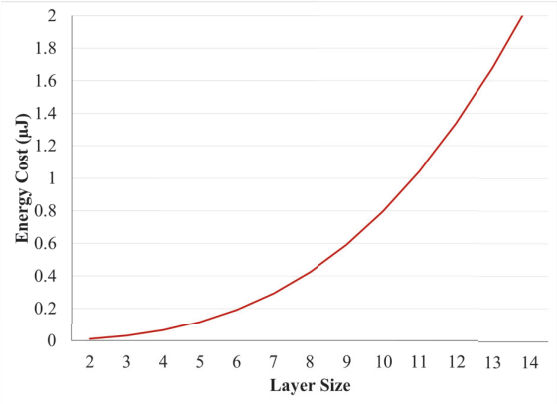

Fig. 8. **Energy consumption scaling as layer size increases. Uses upper bound for MZI tuning cost of 5pJ/Op.**

TABLE II
ENERGY COST FOR INFERENCE AND GRADIENT STEPS IN VGG-16 TRAINING SEPARATED BY LAYER

VGG-16			
Layer Output	**Kernel**	**Layer Cost (mJ)**	**Gradient Cost (mJ)**
224x224x64	3x3	0.058	5.69
112x112x128	3x3	0.010	0.95
56x56x256	3x3	0.005	0.47
28x28x256	3x3	0.001	0.12
14x14x512	3x3	0.001	0.06
4096	FC	11.1	890.4
1000	FC	0.663	53.1
Total		11.9	958.8

TABLE III
ENERGY COST FOR INFERENCE AND GRADIENT STEPS IN MOBILENET TRAINING SEPARATED BY LAYER

MobileNet			
Layer Output	**Kernel**	**Layer Cost (mJ)**	**Gradient Cost (mJ)**
112x112x32	3x3	0.0024	0.24
112x112x32	3x3	0.0024	0.24
56x56x64	3x3	0.0012	0.12
56x56x128	3x3	0.0024	0.24
28x28x128	3x3	0.0006	0.06
28x28x256	3x3	0.0012	0.12
14x14x256	3x3	0.0003	0.03
14x14x512	3x3	0.0006	0.06
7x7x512	3x3	0.0001	0.01
7x7x1024	3x3	0.0003	0.03
9 Layers	1x1xN	0.1620	12.96
1000	FC	0.0054	0.43
Total		0.179	14.54

TABLE IV
INFERENCE PER SECOND (IPS) AND GRADIENT TIME FOR VGG-16 AND MOBILENET FOR MZI MESHES.

Network	**IPS**	**Gradient Time(sec)**
VGG-16	0.33	160.7
Mobilenet	0.30	237.4

inference step, but the mesh is used repeatedly for each layer as described before: two times plus the size of the layer squared minus the size of the layer. Figure 8 shows for various MZI sizes and as observed, larger MZI sizes consume more power for gradient computation. We also consider different power estimates, conservative by assuming optical power parameters (losses) to become worse and aggressive by assuming that optical parameters to become much better in the future. The cost of the gradient for a variety of layer sizes is in Figure 8. As can be seen, the cost of the gradient increases extremely quickly as size of the layer increases, which is unsurprising as the cost of each mesh operation scales linearly, and the number of mesh iterations increases quadratically. The cost of the dot products is extremely small because the results can be passed into it directly from the mesh without ever needing to make a DRAM access. This makes the power consumption of the dot product very low as there are not any memory accesses [1]. The result is that the dot product has marginal impact on the cost of the gradient compared to the optoelectronic power costs.

The models for the cost of gradients and the cost of the inferences were applied to various existing network architectures in order to compare performance on existing benchmarks. The results for inference and gradient energy costs are in Tables II, II, and IV. All of these are operated using the 10 GHz benchmark used by Mojaver et. al. [12]. With this operating speed, the inference step for VGG-16 and Mobilenet take around 3 seconds each, as detailed in Table V. The most significant sources of energy cost are the largest layers. Large layers can be broken apart into smaller matrix multiplications in the same manner as is commonly used in processing element array methods, such as those used by Eyeriss or similar accelerators [1]. While it may be possible to reduce energy cost by doing this, exploring this method for reducing power consumption is beyond the scope of this paper. In this instance, it is only used for the extremely large fully connected layers. Fortunately, this has no effect on how the gradient is calculated, the task is just broken up into each partial sum. The most efficient layers in all cases are the smallest convolutions, as the cost of layer inferences increases as the size does, as previously discussed. However, decreasing layer size in favor of more neurons increases the time it takes for this method to perform inferences. Because of this, VGG-16 and Mobilenet both require more time for both inference and gradient despite having a lower energy cost. The primary method to improve energy consumption is to reduce the size of the layers, and the method to improve inferences per second is to increase layer size.

As far as physical dimensions of the proposed architecture is concerned, there have been prior programmable MZIs designed and fabricated [5]. For example, the programmable nanophotonic processor (PNP) is composed of 176 individually tunable phase modulators and 88 interferometers spanning a chip area of 4.9 mm by 2.4 mm. These are moderate dimensions of the chip to build photonic neural network accelerators. Therefore, we expect that the proposed accelerator with associated components (laser, DAC, ADC) should easily fit within chip dimensions [9], [17].

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

TABLE V
INFERENCE TIME AND COST PER INFERENCE COMPARISON FOR PROPOSED MZI MESHES AND OTHER ELECTRICAL AND OPTICAL ACCELERATORS.

Accelerator	Inference Time (ms)	Cost per Inference (mJ)
MZI Mesh	280	51.19
Eyeriss [1]	29	8.01
Res-DNN [16]	2.90	2.34
UNPU [8]	2.89	0.91
HolyLight [9]	0.02	1.11
Albireo [17]	0.13	2.91

TABLE VI
TIME COMPLEXITY AND MESH BANDWIDTH COMPARISONS.

Mesh Training Method	Time Complexity	Mesh Bandwidth
Clements et. al.	$O(N^4)$	∼10 MHz
Dynamic Calibration	$O(N^2)$	∼2 kHz
Parameterized Gradient	$O(N^2)$	10 MHz

Finally, this system is compared against numerous existing methods for hardware acceleration, as can be seen in Table V. Both the time for inferences and the energy consumption of this conservative model of the system is greater than the compared electrical accelerators [1] [16] [8]. This difference is more significant for the photonic accelerators, which is unsurprising for the relatively low bandwidth for this model [9] [17]. It may be possible to improve the performance of the MZI mesh to match these benchmarks since the mesh is mostly unmodified. Further work to analyze the maximum performance this system might find the limits to the possibilities of this system.

Table VI shows the different mesh training methods, time complexity and mesh bandwidths. It must be noted that programming weights can be an order of magnitude more expensive and imposes limits on bandwidth [12] [4]. In the dynamic calibrated meshes analyzed by Hassan et. al., the mesh bandwidths are on the order of 2kHz which is a difference of three orders of magnitude on the time it takes each training step and the time complexity is O(N^2). Both the calibration process described by Hassan et. al., and the method of setting parameters described by Clements et. al. increase the cost of training meshes by existing methods significantly. Our parameterized gradient approach discussed in previosuly in Section III shows higher mesh bandwidth of 10 MHz (similar to Clements et.al. [2]) with a time complexity of O(N^2). In order to set a multiport interferometer's parameters, the values of the matrix multiplication must be found to zero an element of an intermediate matrix, and this must be repeated for every MZI in the layer. As discussed already, the calibration method of obtaining this is extremely time intensive, but the nulling parameters of an element otherwise requires solving the system of linear equations, which is a task with quadratic time complexity and therefore, Clements et.al. will require O(N^4).

V. RELATED WORK

Reck's decomposition shows how to transform N input states into N output states using an arrangement of beam splitters, phase shifters and mirrors. The methodology transforms the input state with modes (k_1, k_2) into output states with modes ($k_1^{'}$, $k_2^{'}$) as shown [14]:

$$\begin{bmatrix} e^{i\phi}sin\omega & e^{i\phi}cos\omega \\ cos\omega & -sin\omega \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} k_1^{'} \\ k_2^{'} \end{bmatrix}$$

The parameter ω describes the reflectivity ($\sqrt{R}$ = sin ω) and transmittance ($\sqrt{T}$ = cos ω) of the beam splitter. The parameter ϕ can be realized as an external phase shifter after the beam splitter. Reck et. al. adopts this approach where the beam splitter with variable reflectivity can be substituted by a Mach-Zehnder interferometer (MZI) using symmetric 50:50 beam splitters. The most important conclusion of this work is that one can setup the unitary matrix transformation by using these beam splitters or MZIs. This seminal work proposed to apply this methodology to crytography and quantum teleportation in quantum information science and not to neural network modeling.

David Miller [10], [11] showed how to design an optical device that can perform any linear function or coupling between inputs and outputs. The work showed the key concept with successive beam splitters that have phase shifts and reflectivities which can be calibrated such that each device self-configures. This process requires only local feedback loops each operating on a single measurable parameter. Such feedback loops can be left running during device operation, allowing continuous optimization and compensation for drifts in devices. This self-configuration is also progressive, requiring no global calculations or optimization. While the initial concept applied to beam splitters, successive designs applied to path-length adjusted programmable MZIs and was experimentally proven [13], [15]. This is self-training MZIs, but needs additional detectors and equal path lengths for all signal transmission. This approach has not been directly applied to different neural network layers for training.

Clements et. al. [2] builds on Reck's model and improves in some of the key areas. First, Clement's design achieves the minimal optical depth, requiring roughly half of the depth of the Reck design, which is important for minimizing optical losses and reducing fabrication resources. Second, the natural symmetry of this new design makes it significantly more robust to fabrication errors caused by mismatched optical losses. Clement's finding is based on a new mathematical decomposition of a unitary matrix and this decomposition is used to prove the universality of the design and to construct an efficient algorithm to program interferometers based on it. The decomposition method in Clement's can be applied to both row-wise (as Reck's) $T_{m,n}$ and column-wise $T_{m,n}^{-1}$ for a unitary matrix of U with size of m,n. This decomposition does not delve into many details on training the layers or with varying number of inputs.

Other in-situ training approaches have been proposed to train MZIs as seen in [6]. The proposed techniques uses back-propagation in a techniques called time-reversal interference method (TRIM) to compute the cost of the gradient function. After calculating the original amplitude, the time reversed adjoint input is calculated by sending the loss function from the output port. After the interference between the original and time reversed input, the intensities are measured and subtracted

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

from the original amplitudes. This process while has a constant time, the reversing of signals sent into the MZI meshes requires lasers and detectors to be reversed which is a much harder problem. When compared to our approach, keeping the lasers and detectors in identical locations simplifies the hardware architecture.

VI. Conclusions

This paper introduces a method to perform parameterized gradient descent on a multiport interferometer composed of MZIs. The proposed algorithm accomplishes training without adding any additional photodiodes or lasers. This allows for training of the system parameters directly, circumventing extremely costly calibrations during the training process. The proposed method can accomplish the inference phase of photonic neural networks for allowing the PNN to be trained via backpropagation. This algorithm for MZI training introduces the possibility for parameterized gradient descent for any circuit which utilizes MZI's for linear algebra.

VII. Acknowledgement

This research was partially supported by NSF grants CCF-1901192, CCF-1936794, CCF-2324645, and CCF-2311544. We sincerely thank the anonymous reviewers for their excellent feedback.

References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[2] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "Optimal design for universal multiport interferometers," *Optica*, vol. 3, no. 12, pp. 1460–1465, 2016.

[3] L. De Marinis, M. Cococcioni, P. Castoldi, and N. Andriolli, "Photonic neural networks: A survey," *IEEE Access*, vol. 7, pp. 175 827–175 841, 2019.

[4] W. M. Green, M. J. Rooks, L. Sekaric, and Y. A. Vlasov, "Ultra-compact, low rf power, 10 gb/s silicon mach-zehnder modulator," *Optics express*, vol. 15, no. 25, pp. 17 106–17 113, 2007.

[5] N. C. Harris, G. R. Steinbrecher, M. Prabhu, Y. Lahini, J. Mower, D. Bunandar, C. Chen, F. N. C. Wong, T. Baehr-Jones, M. Hochberg, S. Lloyd, and D. Englund, "Quantum transport simulations in a programmable nanophotonic processor," *Nature Photonics*, vol. 11, no. 7, p. 447–452, Jun. 2017. [Online]. Available: http://dx.doi.org/10.1038/nphoton.2017.95

[6] T. W. Hughes, M. Minkov, Y. Shi, and S. Fan, "Training of photonic neural networks through in situ backpropagation and gradient measurement," *Optica*, vol. 5, no. 7, pp. 864–871, 2018.

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

[8] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2018.

[9] W. Liu, W. Liu, Y. Ye, Q. Lou, Y. Xie, and L. Jiang, "Holylight: A nanophotonic accelerator for deep learning in data centers," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1483–1488.

[10] D. A. B. Miller, "Self-aligning universal beam coupler," *Opt. Express*, vol. 21, no. 5, pp. 6360–6370, Mar 2013. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-21-5-6360

[11] ——, "Self-configuring universal linear optical component," *Photon. Res.*, vol. 1, no. 1, pp. 1–15, Jun 2013. [Online]. Available: https://opg.optica.org/prj/abstract.cfm?URI=prj-1-1-1

[12] K. H. R. Mojaver, B. Zhao, E. Leung, S. M. R. Safaee, and O. Liboiron-Ladouceur, "Addressing the programming challenges of practical interferometric mesh based optical processors," *Optics Express*, vol. 31, no. 15, pp. 23 851–23 866, 2023.

[13] F. Morichetti, A. Annoni, S. Grillanda, N. Peserico, M. Carminati, P. Ciccarella, G. Ferrari, E. Guglielmi, M. Sorel, and A. Melloni, "4-channel all-optical mimo demultiplexing on a silicon chip," in *Optical Fiber Communication Conference*. Optica Publishing Group, 2016, p. Th3E.7. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=OFC-2016-Th3E.7

[14] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, "Experimental realization of any discrete unitary operator," *Physical review letters*, vol. 73, no. 1, p. 58, 1994.

[15] A. Ribeiro, A. Ruocco, L. Vanacker, and W. Bogaerts, "Demonstration of a 4 × 4-port universal linear circuit," *Optica*, vol. 3, no. 12, pp. 1348–1357, Dec 2016. [Online]. Available: https://opg.optica.org/optica/abstract.cfm?URI=optica-3-12-1348

[16] N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Res-dnn: A residue number system-based dnn accelerator unit," *IEEE Transactions on Circuits and Systems I: regular papers*, vol. 67, no. 2, pp. 658–671, 2019.

[17] K. Shiflett, A. Karanth, R. Bunescu, and A. Louri, "Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 860–873.

[18] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE, 2012, pp. 201–210.

[19] H. Zhang, J. Thompson, M. Gu, X. D. Jiang, H. Cai, P. Y. Liu, Y. Shi, Y. Zhang, M. F. Karim, G. Q. Lo *et al.*, "Efficient on-chip training of optical neural networks using genetic algorithm," *Acs Photonics*, vol. 8, no. 6, pp. 1662–1672, 2021.

[20] Z. Zheng, Z. Duan, H. Chen, R. Yang, S. Gao, H. Zhang, H. Xiong, and X. Lin, "Dual adaptive training of photonic neural networks," *Nature Machine Intelligence*, vol. 5, no. 10, pp. 1119–1129, 2023.

Appendix A
Derivative Algorithm

The transfer matrix for a circuit of MZIs is the product of $\mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)$ such that:

$$\mathbf{U} = \prod_{j=1}^{M} \mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)$$

For a mesh with input vector $\mathbf{a}$, and an output vector $\mathbf{z}$, the output for the layer can be defined with element-wise complex magnitude as:

$$\vec{\mathbf{z}} = |U\vec{\mathbf{a}}|^2$$

$$\mathbf{z}_j = |\sum_{k=1}^{N} \mathbf{U}_{jk}\mathbf{a}_k|^2$$

Neglecting bias in notation as its derivative is trivial. To find the derivative of each output with respect to ϕ_x

$$\text{let } \mathbf{A} = \prod_{j=x+1}^{M} \mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)$$

$$\text{let } \vec{\mathbf{q}} = \prod_{j=1}^{x-1} \mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)\vec{\mathbf{a}}$$

Substituting in $\vec{\mathbf{z}}$:

$$\vec{\mathbf{z}} = |\mathbf{A}\mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)\vec{\mathbf{q}}|^2$$

let w = $\mathbf{n}_x$, let v = $\mathbf{n}_x + 1$, let $\vec{\mathbf{m}} = \mathbf{U}\vec{\mathbf{a}} = \mathbf{A}\mathbf{T}(\mathbf{n}_j, \boldsymbol{\phi}_j, \boldsymbol{\omega}_j)\vec{\mathbf{q}}$

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.

This yields for each row, $\mathbf{m}_j$

$$\mathbf{m}_j = \mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x)+$$

$$\mathbf{A}_{vx}(\mathbf{q}_v \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x) + \sum_{k=1}^{w-1}\mathbf{A}_{jk}\mathbf{q}_k + \sum_{k=v+1}^{N}\mathbf{A}_{jk}\mathbf{q}_k \qquad z_j$$

$$\text{let } \mathbf{c} = \sum_{k=1}^{w-1}\mathbf{A}_{jk}\mathbf{q}_k + \sum_{k=v+1}^{N}\mathbf{A}_{jk}\mathbf{q}_k \qquad z_j$$

$$\mathbf{m}_j = \mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x)+ \qquad \boldsymbol{\omega}_x$$

$$\mathbf{A}_{vx}(\mathbf{q}_v \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x) + \mathbf{c}$$

Substituting into the output vector $\vec{\mathbf{z}}$:

$$\mathbf{z}_j = |\mathbf{m}_j|^2 = |\mathbf{c} + \mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x)+$$
$$\mathbf{A}_{vx}(\mathbf{q}_v \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x)|^2$$

$$\mathbf{z}_j = |\mathbf{c}|^2 + |\mathbf{A}_{wx}|^2|(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x)|^2$$
$$+|\mathbf{A}_{vx}|^2|(\mathbf{q}_w \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x)|^2$$
$$+[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x), \mathbf{c}]+$$
$$[\mathbf{A}_{vx}(\mathbf{q}_w \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x), \mathbf{c}]$$
$$+[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x+\mathbf{q}_v \cos\boldsymbol{\omega}_x), \mathbf{A}_{vx}(\mathbf{q}_w \cos\boldsymbol{\omega}_x-\mathbf{q}_v \sin\boldsymbol{\omega}_x)]$$

Take the result for power from Appendix A, for each term, seperately:

$$|\mathbf{A}_{wx}|^2|(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x)|^2 =$$
$$|\mathbf{A}_{wx}|^2(|\mathbf{q}_w|^2 \sin^2\boldsymbol{\omega}_x+[\mathbf{q}_w, \mathbf{q}_v] \sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x+|\mathbf{q}_v|^2 \cos^2\boldsymbol{\omega}_x)$$

$$|\mathbf{A}_{vx}|^2|(\mathbf{q}_w \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x)|^2 =$$
$$|\mathbf{A}_{vx}|^2(|\mathbf{q}_w|^2 \cos^2\boldsymbol{\omega}_x-[\mathbf{q}_w, \mathbf{q}_v] \sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x+|\mathbf{q}_v|^2 \sin^2\boldsymbol{\omega}_x)$$

$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x + \mathbf{q}_v \cos\boldsymbol{\omega}_x), \mathbf{c}] =$$
$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_w, \mathbf{c}] \sin\boldsymbol{\omega}_x+$$
$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_v, \mathbf{c}] \cos\boldsymbol{\omega}_x$$

$$[\mathbf{A}_{vx}(\mathbf{q}_w \cos\boldsymbol{\omega}_x - \mathbf{q}_v \sin\boldsymbol{\omega}_x), \mathbf{c}] =$$
$$[\mathbf{A}_{vx}\mathbf{q}_w, \mathbf{c}] \cos\boldsymbol{\omega}_x - [\mathbf{A}_{vx}\mathbf{q}_v, \mathbf{c}] \sin\boldsymbol{\omega}_x$$

And finally

$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}(\mathbf{q}_w \sin\boldsymbol{\omega}_x+\mathbf{q}_v \cos\boldsymbol{\omega}_x), \mathbf{A}_{vx}(\mathbf{q}_w \cos\boldsymbol{\omega}_x-\mathbf{q}_v \sin\boldsymbol{\omega}_x)] =$$
$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}, \mathbf{A}_{vx}](\mathbf{q}_w^2 - \mathbf{q}_v^2) \sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x+$$
$$[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_v, \mathbf{A}_{vx}\mathbf{q}_w] \cos^2\boldsymbol{\omega}_x - [\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_w, \mathbf{A}_{vx}\mathbf{q}_v] \sin^2\boldsymbol{\omega}_x$$

$$\text{let } t_1 = |\mathbf{A}_{wx}|^2|\mathbf{q}_w|^2 + |\mathbf{A}_{vx}|^2|\mathbf{q}_v|^2 - [\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_w, \mathbf{A}_{vx}\mathbf{q}_v]$$
$$\text{let } t_2 = |\mathbf{A}_{wx}|^2|\mathbf{q}_v|^2 + |\mathbf{A}_{vx}|^2|\mathbf{q}_w|^2 + [\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_v, \mathbf{A}_{vx}\mathbf{q}_w]$$
$$\text{let } t_3 = |\mathbf{A}_{wx}|^2[\mathbf{q}_w, \mathbf{q}_v]-|\mathbf{A}_{vx}|^2[\mathbf{q}_w, \mathbf{q}_v]+[\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}, \mathbf{A}_{vx}](\mathbf{q}_w^2-\mathbf{q}_v^2)$$
$$\text{let } t_4 = [\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_w, \mathbf{c}] - [\mathbf{A}_{vx}\mathbf{q}_v, \mathbf{c}]$$
$$\text{let } t_5 = [\mathbf{A}_{wx}e^{i\boldsymbol{\phi}_x}\mathbf{q}_v, \mathbf{c}] + [\mathbf{A}_{vx}\mathbf{q}_w, \mathbf{c}]$$

Combine like terms in z_j

$$= t_1 \sin^2\boldsymbol{\omega}_x+t_2 \cos^2\boldsymbol{\omega}_x+t_3 \sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x+t_4 \sin\boldsymbol{\omega}_x+t_5 \cos\boldsymbol{\omega}_x$$

Derivative of z_j:

$$= t_1 \sin 2\boldsymbol{\omega}_x-t_2 \sin 2\boldsymbol{\omega}_x+t_3 \cos 2\boldsymbol{\omega}_x+t_4 \cos\boldsymbol{\omega}_x-t_5 \sin\boldsymbol{\omega}_x$$

Moving one term at a time for two passes of the mesh, set to $\boldsymbol{\omega}_x$ plus $\pi/4$ for the first, and $\boldsymbol{\omega}_x$ minus $\pi/4$ for the second. For the t_1 terms:

$$\sin^2(\boldsymbol{\omega}_x + \frac{\pi}{4}) - \sin^2(\boldsymbol{\omega}_x - \frac{\pi}{4})$$
$$\frac{(\sin\boldsymbol{\omega}_x + \cos\boldsymbol{\omega}_x)^2}{2} - \frac{(\sin\boldsymbol{\omega}_x - \cos\boldsymbol{\omega}_x)^2}{2}$$

$$2\sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x = \sin 2\boldsymbol{\omega}_x$$

For the t_2 terms:

$$\cos^2(\boldsymbol{\omega}_x + \frac{\pi}{4}) - \cos^2(\boldsymbol{\omega}_x - \frac{\pi}{4})$$
$$\frac{(\cos\boldsymbol{\omega}_x - \sin\boldsymbol{\omega}_x)^2}{2} - \frac{(\cos\boldsymbol{\omega}_x + \sin\boldsymbol{\omega}_x)^2}{2}$$

$$-2\sin\boldsymbol{\omega}_x \cos\boldsymbol{\omega}_x = -\sin 2\boldsymbol{\omega}_x$$

For the t_3 terms:

$$\sin(\boldsymbol{\omega}_x + \frac{\pi}{4})\cos(\boldsymbol{\omega}_x + \frac{\pi}{4}) - \sin(\boldsymbol{\omega}_x - \frac{\pi}{4})\cos(\boldsymbol{\omega}_x - \frac{\pi}{4})$$
$$\frac{(\cos^2\boldsymbol{\omega}_x - \sin^2\boldsymbol{\omega}_x)}{2} - \frac{(\sin^2\boldsymbol{\omega}_x - \cos^2\boldsymbol{\omega}_x)}{2}$$
$$\cos^2\boldsymbol{\omega}_x - \sin^2\boldsymbol{\omega}_x = \cos 2\boldsymbol{\omega}_x$$

For the t_4 terms:

$$\sin(\boldsymbol{\omega}_x + \frac{\pi}{4}) - \sin(\boldsymbol{\omega}_x - \frac{\pi}{4})$$
$$\frac{\sin\boldsymbol{\omega}_x + \cos\boldsymbol{\omega}_x}{\sqrt{2}} - \frac{\sin\boldsymbol{\omega}_x - \cos\boldsymbol{\omega}_x}{\sqrt{2}}$$
$$\sqrt{2}\cos\boldsymbol{\omega}_x$$

For the t_5 terms:

$$\cos(\boldsymbol{\omega}_x + \frac{\pi}{4}) - \cos(\boldsymbol{\omega}_x - \frac{\pi}{4})$$
$$\frac{\cos\boldsymbol{\omega}_x - \sin\boldsymbol{\omega}_x}{\sqrt{2}} - \frac{\cos\boldsymbol{\omega}_x + \sin\boldsymbol{\omega}_x}{\sqrt{2}}$$
$$-\sqrt{2}\sin\boldsymbol{\omega}_x$$

This means that for nearly all terms that do not contain $\mathbf{c}$, this method obtains the derivative with respect to $\boldsymbol{\omega}_x$ exactly, and for terms that do contain $\mathbf{c}$, it is off by a factor of $\sqrt{2}$ If we modulate channels that do not pass through the target MZI by $1/\sqrt{2}$, then taking two passes of the mesh with the original value of $\boldsymbol{\omega}_x$ plus $\pi/4$ and then minus $\pi/4$, we obtain the derivative with respect to that parameter, $\boldsymbol{\omega}_x$. Therefore we can implement the derivative with two mesh iterations adding and subtracting $\pi/4$, and multiplying channels that do not pass through the mesh by $1/\sqrt{2}$.

Authorized licensed use limited to: IEEE Editors-in-Chief. Downloaded on June 25,2025 at 23:49:01 UTC from IEEE Xplore. Restrictions apply.