

Model-free front-to-end training of a large high performance laser neural network

Anas Skalli^{1*}, Satoshi Sunada², Mirko Goldmann¹,
Marcin Gebski³, Stephan Reitzenstein⁴, James A. Lott⁴,
Tomasz Czyszanowski³, Daniel Brunner¹

¹Institut FEMTO-ST, Université Marie et Louis Pasteur, CNRS UMR,
6174, Besançon, France.

²Faculty of Mechanical Engineering, Institute of Science and
Engineering, Kanazawa University, Kakuma-machi Kanazawa, Ishikawa
920–1192, Japan.

³Institute of Physics, Lodz University of Technology, ul. Wólczańska
219, 90-924 Lodz, Poland.

⁴Technical University of Berlin, Hardenbergstraße 36, D-10623 Berlin,
Germany.

*Corresponding author(s). E-mail(s): anas.skalli@femto-st.fr;
anasskalli2@gmail.com;

Abstract

Artificial neural networks (ANNs), have become ubiquitous and revolutionized many applications ranging from computer vision to medical diagnoses. However, they offer a fundamentally connectionist and distributed approach to computing, in stark contrast to classical computers that use the von Neumann architecture. This distinction has sparked renewed interest in developing unconventional hardware to support more efficient implementations of ANNs, rather than merely emulating them on traditional systems. Photonics stands out as a particularly promising platform, providing scalability, high speed, energy efficiency, and the ability for parallel information processing. However, fully realized autonomous optical neural networks (ONNs) with in-situ learning capabilities are still rare. In this work, we demonstrate a fully autonomous and parallel ONN using a multimode vertical cavity surface emitting laser (VCSEL) using off-the-shelf components. Our ONN is highly efficient and is scalable both in network size and inference bandwidth towards the GHz range. High performance hardware-compatible optimization algorithms are necessary in order to minimize reliance

on external von Neumann computers to fully exploit the potential of ONNs. As such we present and extensively study several algorithms which are broadly compatible with a wide range of systems. We then apply these algorithms to optimize our ONN, and benchmark them using the MNIST dataset. We show that our ONN can achieve high accuracy and convergence efficiency, even under limited hardware resources. Crucially, we compare these different algorithms in terms of scaling and optimization efficiency in term of convergence time which is crucial when working with limited external resources. Our work provides some guidance for the design of future ONNs as well as a simple and flexible way to train them.

Keywords: Model-free optimization, Online learning, Photonic neural networks, Semiconductor laser

Contents

1	Introduction	4
2	Ceiling Analysis: Moving Beyond Reservoir Computing and Extreme Learning Machines	6
2.1	Motivation and Methodology	6
2.2	Results of the ceiling analysis	7
3	Survey of hardware-compatible training methods	10
4	Selected strategies and their implementation	11
4.1	Gradient measurement perturbative methods	12
4.1.1	Finite difference method	12
4.1.2	Simultaneous Perturbation Stochastic Approximation (SPSA)	12
4.2	Population and sampling based methods	13
4.2.1	Evolutionary strategy basics	14
4.2.2	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	15
4.2.3	Parameter Exploring Policy Gradients (PEPG)	17
4.2.4	Particle Swarm Optimization (PSO)	19
4.3	Application to Toy problems	23
4.3.1	Objective Functions	23
4.3.2	Neural network optimization	24
5	Towards a Fully Tunable Optical Neural Network	30
5.1	Large area VCSELs	30
5.1.1	Circular cavity VCSELs	30
5.1.2	Chaotic cavity VCSELs	32
5.2	Updated experimental setup	33
5.2.1	Input layer	34
5.2.2	Output layer	36
6	Benchmarking Training Strategies	37
6.1	Hyperparameter tuning of training algorithms	37
6.1.1	Finite difference	38
6.1.2	Simultaneous Perturbation Stochastic Approximation	40
6.1.3	Covariance Matrix Adaptation Evolution Strategy	41
6.1.4	Parameter-exploring Policy Gradients	42
7	Comparison of training algorithms	44
8	Conclusion	50

1 Introduction

Over the past decade, artificial neural networks (ANNs) have transformed numerous fields, ranging from natural language processing [1, 2] and image recognition [3], to autonomous driving [4] and advanced game strategies [5]. Perhaps the best indicator of the success of ANNs and other machine learning algorithms in general, is that they have become so ubiquitous that they are now part of our daily lives, often without us even realizing it. Their success was the product of a convergence of several key factors: the availability of vast datasets, the rise of affordable and powerful computational resources, and their remarkable ability to tackle complex tasks without requiring explicit programming, making their flexibility unparalleled.

ANNs represent a different computing paradigm, grounded in a connectionist and parallel approach to information processing, where a network learns to adapt to perform a task rather than being explicitly programmed to do so. As such, they are fundamentally different from the traditional von Neumann computers used to emulate them.

This mismatch, coupled to fundamental physical properties of electronics, results in significant additional overhead when running ANN algorithms on conventional electronic hardware. In addition, the rapid growth in size and complexity of ANNs has led to a surge in computational demands which has outpaced improvements on the computing front. This has driven the rapid development and adoption of more parallel von Neumann architectures, such as graphical processing units (GPUs) and tensor processing units (TPUs) [6]. Yet even these are starting to show their limits. Consequently, there has been a surge in interest in researching unconventional hardware platforms to efficiently implement ANNs [7]. Motivated by these fundamental physical limitations, both academia and industry have been exploring unconventional physical computing substrates that harness nonlinear effects for continuous, analog, in-memory computing [8].

Optics has re-emerged as a highly promising platform [9, 10], offering scalability [11–13], high-speed performance [14–16], energy efficiency [17], and the ability for parallel information processing [15, 18]. Recent advancements in photonic hardware include integrated tensor cores on-chip [19] and high-dimensional optical pre-processors [20, 21]. Notably, semiconductor lasers are promising candidates for implementing optical neural networks (ONNs) due to their ultra-fast modulation rates and complex dynamics [15]. Vertical-cavity surface-emitting lasers (VCSELs), in particular, stand out for their efficiency, speed, intrinsic nonlinearity, and well-established CMOS-compatible fabrication process [22, 23].

A very popular approach when pursuing unconventional computing is reservoir computing (RC)[24, 25], which simplifies recurrent neural networks by transforming input data through a fixed, high-dimensional network of nonlinear nodes—the reservoir—and training only the output weights. This minimal complexity, closely related to extreme learning machines[26], has enabled implementations on a wide array of physical substrates ranging from electronics [27] and spintronics [28] to various optical approaches—time-multiplexed [15, 27], frequency-multiplexed [18], spatially multiplexed [29, 30]—and even mechanical systems [31]. Recent explorations also include

quantum RC to exploit the exponential scaling of Hilbert space for information processing [32].

While RC offers flexibility and value through its simplified design, researchers are increasingly exploring fully trainable and more complex architectures to push performance boundaries. In pursuit of harnessing unconventional ANN hardware for specific tasks, significant advances in hardware-compatible training algorithms have emerged [33–37]. These developments include model-based techniques such as back-propagation using a digital twin [33] and augmented direct feedback alignment [36], as well as experimental methods for implementing in-situ backpropagation [37, 38]. Simultaneously, model-free or black-box strategies are gaining traction [29, 34, 39, 40]. However, fully realized in-situ hardware implementations of these methods remain scarce. For unconventional neural networks to be truly competitive, integrating in-situ training techniques is essential to overcome existing bottlenecks and reduce reliance on external high-performance computers—a dependency that often undermines the inherent benefits of these novel computing substrates.

In our earlier works [29, 30, 30], we demonstrated RC via spatial multiplexing of modes on a large-area VCSEL (LA-VCSEL), establishing a parallel, autonomous network that minimizes reliance on external computing by employing hardware learning rules with Boolean weights. In this work, we investigate different ways by which we can improve its performance. As such, in section 2 we conduct a ceiling analysis on a software ANN of representative size to understand which parts of the ANN are most crucial to performance. Among these, we find that the presence of both positive and negative weights, a tunable input connectivity as well as sufficient weight resolution are all crucial parameters. This analysis will influence our subsequent ONN designs. The changes to our physical ONN implied by the ceiling analysis require us to rethink the way we train our network. We therefore give a brief overview of the methods used to train physical neural networks in section 3. Next in section 4, we present different hardware-compatible, model-free training algorithms in detail. We use them to optimize a toy-problem and to train a software ANN in order to compare them in terms of performance and convergence efficiency. The different evaluated algorithms include: perturbative methods such as finite difference and simultaneous perturbation stochastic approximation (SPSA), evolutionary algorithms such as CMA-ES and parameter-exploring policy gradients (PEPG), as well as particle swarm optimization (PSO). Among these algorithms, PEPG, CMA-ES and SPSA proved to be the most promising in terms of accuracy and convergence on MNIST, while PSO showed less effective results, struggling to converge efficiently.

In section 5 we apply all the conclusions from the ceiling analysis to alter the physical design of our VCSEL-based ONN, going beyond the RC architecture by implementing trainable input weights as well as implementing positive and negative weights with 8-bit resolution using spatial light modulators. We then use the training algorithms studied previously to optimize our ONN. We start with an extensive hyperparameter study of these algorithms in section 6 giving guidance and intuition regarding the setting of said hyperparameters. Additionally, we benchmark the different training algorithms using the well-known MNIST hand written digit dataset. Here, rather than simply focusing on pure accuracy, we also study the convergence efficiency

of these algorithms in terms of time in section 7. Indeed, the time spend to optimize the network is linearly proportional to its energy consumption. This is of particular interest when energy efficiency is crucial, particularly when it comes to online learning under limited hardware resources, which is extremely relevant. Indeed, in physical computing, optimization typically focuses on the efficiency of the hardware, often overlooking the energy impact of the computations required for its training. However, some algorithms, being too resource-intensive, can undermine the expected energy savings. By evaluating various algorithms in terms of both efficiency and computational cost, we introduce a useful perspective on optimizing hardware neuromorphic networks. This approach, which extends beyond the realm of optics, is applicable to a wide range of physical systems, from electronic circuits to quantum devices.

We demonstrate a scalable ONN both in size and bandwidth reaching up to 10 000 neurons in parallel, which will be capable of GHz inference due to the fast response time of the laser. We find that PEPG is the most efficient algorithm offering the best performance and convergence efficiency. Interestingly CMA-ES while a widely popular algorithm offers similar performance to PEPG but its heavy quadratic overhead significantly hampers its efficiency in high-dimensional optimization scenarios taking up to 4 times longer to converge in our ONN. SPSA while deceptively simple, offers a compromise between the two algorithms. Finally, we compared the performance of PEPG and SPSA for all classes of the MNIST dataset and compared them to two baselines: a digital linear classifier and a hardware linear system (ONN with LA-VCSEL off). PEPG surpassed the digital linear classifier, while SPSA approached but did not exceed it highlighting the almost unfair advantage of offline weights. Importantly, both algorithms significantly outperformed the hardware linear system, showcasing the high-dimensional transformations enabled by the LA-VCSEL.

2 Ceiling Analysis: Moving Beyond Reservoir Computing and Extreme Learning Machines

In our previous work [29, 30, 41] we have shown that a simple LA-VCSEL coupled with information injection and optimized through optically implemented readout weights can be used to solve simple classification tasks. We also studied the influence of physical hyperparameters on performance. As a proof of concept, we chose the classification of binary patterns for their simplicity and tunability. We would now like to move beyond this simple task and show that our system can move towards more complex and "real world" tasks such as the MNIST dataset. To that end, we start by setting the scene by conducting a ceiling analysis on a software NN to know which parts of the NN are most crucial for performance. We then motivate and explain several hardware-compatible training strategies and benchmark them on toy and benchmark problems, paving the way for their future use in our ONN.

2.1 Motivation and Methodology

Let us first benchmark our physical system on the MNIST dataset. This requires no physical change to the experiment, and just requires us to display input sequences comprised of MNIST digits on the input DMD. Using our improved learning strategy,

we optimize the Boolean readout on DMD_b. Figure 1 shows the results on one-vs-all classification for each MNIST digit and compares it to a simple linear regression model trained and tested on the same experimental data. We use one-vs-all classification as the output of our ONN is a scalar. Each classifier is trained and tested on sequences comprising 1000 images of the target digit and 1000 images of the other digits. Figure 1, shows that the linear model outperforms the LA-VCSEL for every digit, achieving an accuracy of $\sim 92\%$ on average, compared to 84 % for the LA-VCSEL.

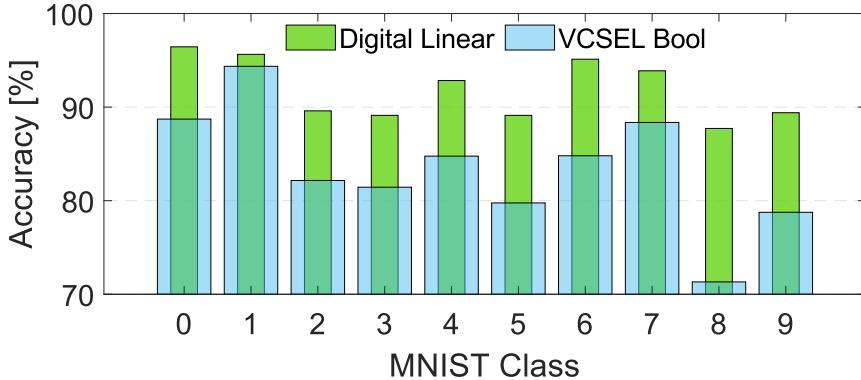


Fig. 1 Results of the MNIST classification task using Boolean readout weights of the LA-VCSEL.

The questions facing us now is how do we improve the performance of our system on this task? Are we limited by hardware factors such as Boolean weights, or our NN architecture, i.e. the fact that we have fixed random input and non-optimized recurrent internal weights? To answer this question, we study the behavior of a software NN with a size comparable to our ONN, i.e. 100 nodes, trained with the back-propagation algorithm. This process is known as conducting a ceiling analysis. By systematically removing or restricting parts of the NN and observing the impact on overall performance, it allows to pinpoint bottlenecks or areas where improvements would have the most significant impact. This method is particularly useful in complex systems like NNs, where it can help us understand how different factors contribute to the system's limitations, guiding future improvements to our ONN.

2.2 Results of the ceiling analysis

As shown in Fig. 2(a), we first evaluate the performance of a fully connected feed-forward neural network (FFNN) of 100 nodes on the MNIST dataset using Pytorch. This FFNN is fully trained using the backpropagation algorithm with no restriction on the weights. In this reference configuration, the FFNN achieves an accuracy of $\sim 97.5\%$, as a reminder the accuracy of a linear model is $\sim 93\%$. If the input weights are fixed, as is the case in our experimental setup, the FFNN becomes an extreme learning machine (ELM) which works similarly to a reservoir in its steady state. The

ELM achieves an accuracy of $\sim 87\%$, showing that the input weights are crucial for performance when working with spatially diverse data. Finally, to simulate a more realistic scenario, we train the FFNN while restricting all weights to be positive only, which proves extremely detrimental to performance, only achieving an accuracy of $\sim 60\%$. This shows that negative weights are crucial for performance, as they can leverage neurons which are negatively correlated with the target.

To further show the importance of input weights, we study how the performance of, both, FFNN and ELM scale with the number of neurons, as shown by Fig. 2(b). By simply training the input weights we are able to leverage more computational capability from the NN. As such, we can decrease the amount of required neurons by a factor of $\approx 30\times$ if we implement trainable input weights, see Fig. 2(c). Crucially, the ELM only outperforms the linear model when comprising more than 300 neurons. This has considerable implications concerning the use of unconventional physical substrates as means of computing. Indeed, it shows that increasing the number of neurons, which might be difficult and often entails changing substrates, e.g. getting bigger LA-VCSELs, is not the most efficient way to increase performance, while implementing some tunability at the input can lead to better improvements. This raises a potential discussion on the tradeoff between trainable connectivity and nonlinearity in NNs that perhaps deserves a more in-depth study. This tradeoff should in principle be highly task and architecture dependent. Intuitively, we can understand how input weights are crucial to performance when processing spatially diverse data like images. Yet, they may not be as crucial when handling scalar inputs such as some time series. In the latter case, tuning dynamical properties might become more advantageous, and one could for instance study the impact of training internal weights of RNNs. A detailed ceiling analysis for different types of tasks and architectures, focusing on different parts of the NN would be highly interesting, yet is beyond the scope of our study.

In the context of a software NN, input weights are potentially trivial to implement and benefit from double precision representation. In contrast, hardware weights would most certainly be limited in resolution, thus we need to study the impact of weight resolution on performance. In the context of optics, 8-bit or even 10-bit resolution is readily achieved by commercially available spatial light modulators (SLM) using liquid crystal on Silicon technology (LCOS). Yet, when it comes to new integrated photonics technologies, such as phase change materials, the resolution is more towards 5-bit [42]. Figure 2(d) shows how performance degrades with lower bit resolution and demonstrates that only marginal improvements in performance are achieved above 4-bit weight resolutions, for a simple single-layer FFNN of 100 neurons applied to the MNIST dataset. In contrast, 2-bit weights cannot solve the MNIST task with the current architecture. It should be noted that quantization of NNs have garnered great interest [43, 44] for its memory saving capabilities. Moreover, production models are often trained with full 32-bit resolution gradients and shipped with reduced resolution such as 8-bit to save on memory, especially for NNs running on restricted hardware such as smartphones or microcontrollers [45].

Another crucial aspect of layers in the FFNN is that they implement fully programmable matrix vector operations, which are not trivial to implement in hardware. Indeed, implementing a fully tunable matrix vector multiplication on the input data is

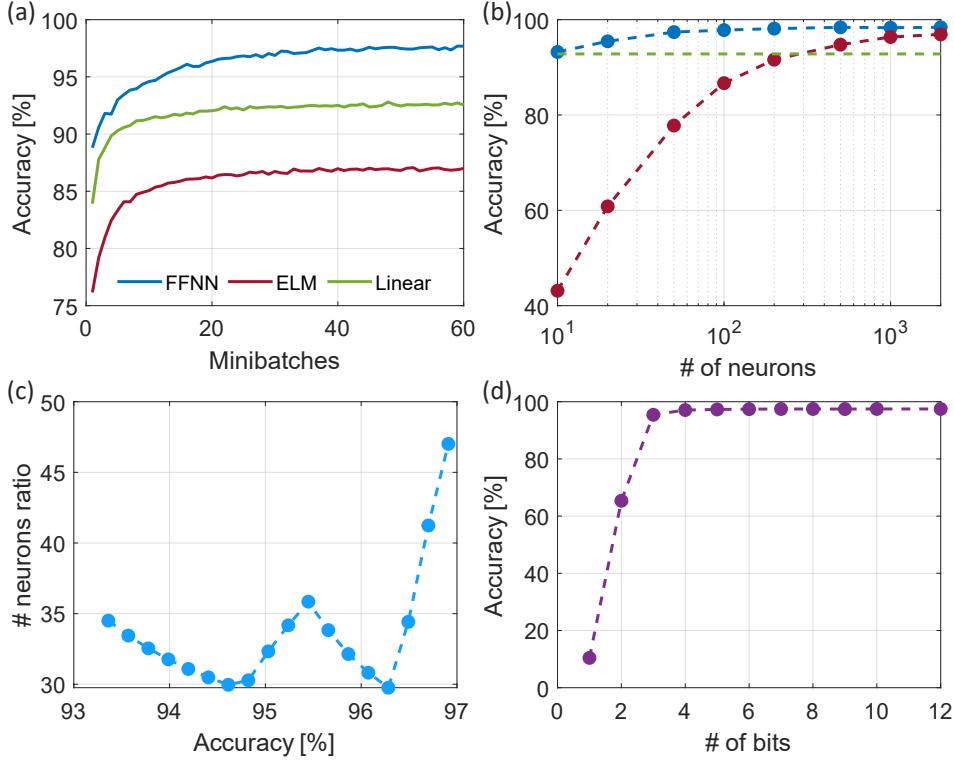


Fig. 2 (a) Comparison between a single-layer FFNN in blue and an ELM in red on the MNIST dataset, the baseline performance for a linear classifier is shown in green. (b) Performance for FFNN and ELM architectures as a function of the number of neurons. (c) Ratio between the number of neurons needed for the ELM to reach the same accuracy compared to the FFNN. (d) Performance of the FFNN as a function of the weight resolution.

non-trivial. In contrast, simply imaging the input data on a SLM yields a Hadamard or element wise product, which can be a good starting point but is vastly inferior in terms of information processing capability.

The results discussed thus far are presented in Table 1.

Architecture	Performance
Fully trained, 32-bit	~97.5%
Fully trained, 4-bit	~97%
Fully trained positive weights only	60%
ELM - training W^{out} only	~87%

Table 1 Summary of the main results of the ceiling analysis.

These conclusions may seem trivial in the context of conventional machine learning, yet they have profound consequences when it comes to building a setup that leverages

a physical system i.e., the LA-VCSEL as the central piece of an ONN. Firstly, negative weights, while crucial for accuracy, are not trivial to implement optically. Secondly, implementing trainable input weights is a challenge because one cannot rely on error backpropagation to train them as is done in digitally simulated NNs. Therefore, a hardware-friendly optimization algorithm that is sufficiently efficient is required. As a result, we will discuss such optimization algorithms in the next sections.

3 Survey of hardware-compatible training methods

In the context of physical / hardware NNs, online training strategies are usually required to account for physical imperfections such device variability and noise. Figure 3 aims to give a brief summary of the different training strategies that are relevant for hardware NNs, while listing their strengths and weaknesses.

Training Mechanism	Advantages	Disadvantages
Reservoir Computing	<ul style="list-style-type: none"> • Easy to implement • Demonstrated in hardware 	<ul style="list-style-type: none"> • Does not exploit the full potential of the hardware
BP with backward pass in a digital twin	<ul style="list-style-type: none"> • Allows to train the full system • Approximates the true gradient • Demonstrated in hardware 	<ul style="list-style-type: none"> • Needs a lot of data to capture the physics of the system, big overhead • Need to retrain said model as the physical hardware drifts / changes • Need to train a model per physical device due to variability
Augmented Direct Feedback Alignment	<ul style="list-style-type: none"> • Allows to train the full system • Only minimal knowledge of physical system is required • Partially demonstrated in hardware 	<ul style="list-style-type: none"> • Partial knowledge of the model is still needed, full knowledge of states and activation functions is required. • Gradients are misaligned, reduced performance • Only works on certain types of neural networks even in software
Forward-Forward training	<ul style="list-style-type: none"> • Allows to train deep models layer-wise • Partially demonstrated in hardware • No specific model needed 	<ul style="list-style-type: none"> • Requires generating "negative" examples, adding complexity to the training process. • Reduces inference efficiency, as each input must be compared to all labels, lowering bandwidth.
Equilibrium propagation	<ul style="list-style-type: none"> • Performs Backpropagation based on system dynamics = true gradient 	<ul style="list-style-type: none"> • Energy function of the system needs to match certain criteria, maybe not applicable to all kinds of devices
Black-box / model-free methods	<ul style="list-style-type: none"> • Easy to implement, demonstrated in hardware • Flexible = hardware agnostic • Trains the full system • Can be memory efficient 	<ul style="list-style-type: none"> • Converges slower because updates are not based on the true gradient • Plethora of algorithms to choose from, can be overwhelming to test them all • Potential scalability issues

Fig. 3 Summary of different hardware compatible strategies with their advantages and disadvantages, compiled from [33, 36, 46–48].

In our case, we want to achieve a fully autonomous NN, and as such we will focus on model-free, or “black-box” training algorithms. Indeed, only model-free methods minimize the reliance on an external computer in contrast to model-based methods e.g. digital twin training [33], which fully relies on a pretrained model to compute the update for hardware weights. Moreover, continuous in-the-loop or online learning should compensate for physical system drift, which could hinder the performance of a digital twin over time. As a result, model-free methods have been singled out as a flexible solution for training physical NNs, yet fully fledged implementations of these algorithms in hardware remain scarce. These algorithms can be grouped into two broad categories, on the one hand, perturbative methods with the objective to estimate the

gradient by sampling a target function to optimize (i.e. loss function) at different coordinates (i.e. weights). On the other hand, the second class of gradient-free methods consist of population-based sampling, which relies on entirely different concepts and dynamics to optimize functions. These include popular classes of algorithms such as genetic algorithms (GA), evolutionary strategies (ES), swarm optimization and reinforcement learning (RL) algorithms. These are not directly concerned with achieving an approximation of the gradient, but rather iteratively generate better candidate solutions to an optimization problem. They achieve this either according to heuristic criteria in the case of GA, ES, and swarm-type algorithms, or according to an iteratively improved candidate generation policy in the case of RL. A detailed study comparing the computational cost of training a physical system using model-free versus model-based methods and examining the tradeoff between the high initial cost of pretrained models and the less accurate parameter updates from model-free methods requiring more steps to converge, is still sorely needed at this point but is beyond the scope of this work.

4 Selected strategies and their implementation

In this section, we present different perturbative and population-based strategies. We will measure the performance of these strategies on optimization problems to showcase their strengths and weaknesses.

Our first toy problem is a non-convex function, namely the well-established Rastrigin function, first proposed by Leonid Rastrigin in 1963 [49]. It is defined as follows:

$$f(x) = An + \sum_{i=1}^n (x_i^2 - A \cos(2\pi x_i)). \quad (1)$$

This function has a global minimum at $f(x = 0) = 0$ and is highly non-convex. We will use it to showcase the performance of our training strategies. Figure 4 shows the Rastrigin function in 2D, with an equal scale for both dimensions, demonstrating the complex nature of the function with many periodically placed local minima. Intuitively, we can see how a gradient-based optimization method could get stuck in the many local minima the Rastrigin function presents.

Optimization competitions routinely feature the Rastrigin function as a benchmark, benchmark, but they furthermore use a plethora of other functions each one more convoluted than the other to showcase the performance of optimization algorithms. Due to significant progress in the field of optimization, these functions are often combined to create composition functions in order to increase complexity for competitions [50].

Lastly, to provide a more realistic comparison between the different optimization methods, we will use a small NN and train it using the different optimization methods that follow.

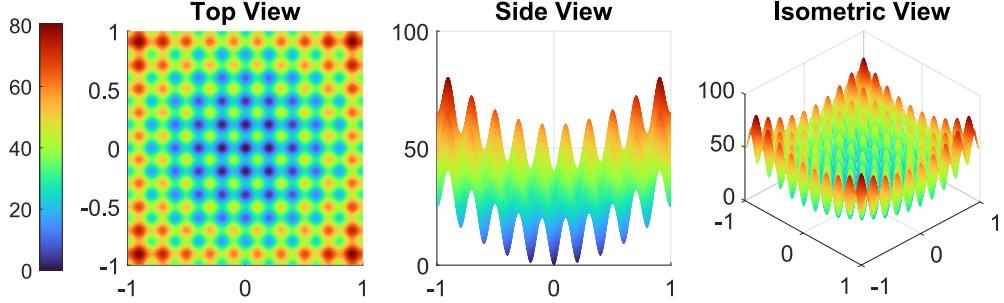


Fig. 4 The Rastrigin function in 2D.

4.1 Gradient measurement perturbative methods

First, let us discuss the simplest methods. In software NNs, the backpropagation (BP) algorithm is used to determine gradients, subsequently, gradient descent (GD) is then used to update the weights. Although, both, BP and GD are often used interchangeably, they are two distinct algorithms. In hardware, the backpropagation algorithm is not directly implementable for most systems, yet nothing prevents us from implementing a simple GD algorithm, provided we find a way to compute, estimate or measure said gradient, thus we will present two perturbative methods that estimate the gradient.

4.1.1 Finite difference method

The simplest estimation of a gradient can be measured by a finite difference scheme. Simply by perturbing a weight by a small amount $\pm\epsilon$ we can measure a difference in error and therefore compute a gradient according to the basic definition of a numerical derivative for a function L around a point w given by

$$\frac{\delta L}{\delta w} \simeq \frac{L(w + \epsilon) - L(w - \epsilon)}{2\epsilon}. \quad (2)$$

Using this perturbative method on every weight will therefore give us an approximation of the gradient for all parameters, allowing us to perform gradient descent, with the gradient estimate being more accurate for small values of ϵ . Yet, backpropagation makes use of the forward pass of the NN, and can utilize massive hardware vectorization to compute the gradients in a parallel fashion. In contrast, the finite difference method needs to sequentially perturb each weight individually and requires 2 forward passes for each weight. This optimization method while uninspired and has the merit of being local while using very little memory overhead storage as only one weight is perturbed at a time. The pseudocode for an optimization loop using finite difference gradients is given in Algorithm 1, describing the optimization of a given function L .

4.1.2 Simultaneous Perturbation Stochastic Approximation (SPSA)

A more sophisticated perturbative method is the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm. This method was introduced by J. C. Spall in

Algorithm 1 Finite Difference Gradient Descent Optimization

```
1: Initialize  $\mathbf{w}_0$ , set constant learning rate  $\alpha$  and a small value for  $\epsilon$ 
2:  $K \leftarrow$  maximum number of iterations
3: Specify number of parameters to perturb each epoch  $P$ 
4: for  $k = 0, 1, \dots, K - 1$  do
5:   Initialize gradient vector  $\mathbf{g}$  to zeros
6:   Select  $P$  unique parameters randomly from  $\mathbf{w}$  to form subset  $\mathbf{w}_{\text{sub}}$ 
7:   for each  $w_j$  in  $\mathbf{w}_{\text{sub}}$  do
8:     Compute  $L^+ \leftarrow L(w_j + \epsilon)$  other parameters stay fixed
9:     Compute  $L^- \leftarrow L(w_j - \epsilon)$  other parameters stay fixed
10:     $g_j \leftarrow \frac{L_+ - L_-}{2\epsilon}$  ▷ Estimate gradient for  $w_j$ 
11:   end for
12:   Update  $w$  for all  $w_j$  in  $\mathbf{w}_{\text{sub}}$ :  $w_j = w_j - \alpha * g_j$ 
13: end for
14: return Optimized parameters vector  $\mathbf{w}_{k_{\min}}$ , and optimal Loss value  $L(\mathbf{w}_{k_{\min}})$ 
```

1987 [51, 52] and is a stochastic gradient descent based method. It is particularly well suited for hardware-based optimization, as it only requires two function evaluations to estimate the gradient, regardless of the search space dimensionality. It is thus quite a computationally efficient algorithm that has been used to optimize simple NN controllers [53–56].

SPSA estimates the gradient vector of a multivariate function by perturbing all parameters \mathbf{w} simultaneously along a randomly chosen direction vector Δ :

$$\mathbf{g} = \frac{L(\mathbf{w} + \epsilon\Delta) - L(\mathbf{w} - \epsilon\Delta)}{2\epsilon \text{VAR}(\Delta)} \Delta \simeq \nabla L(\mathbf{w}), \quad (3)$$

where \mathbf{w} is the parameter vector, ϵ is a small perturbation, and L is the loss function. The direction vector Δ is chosen to be a random vector with elements ± 1 , and VAR is the variance operator. We can then use the gradient estimate vector \mathbf{g} to update the parameter vector \mathbf{W} using the standard gradient descent update rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}, \quad (4)$$

where α is the step size at iteration or learning rate. Naturally, we can use momentum methods or Adam for updates rather than vanilla gradient descent to speed up learning. The pseudocode for an optimization loop using SPSA is given in Algorithm 2.

4.2 Population and sampling based methods

The second class of optimization methods we explore are so called population-based methods. These methods use information from a population of coordinates to optimize a target function and are as such more global in their nature than the perturbative methods previously introduced. Among these, we can find many algorithms such as the genetic algorithm, different versions of ESs and other nature inspired methods

Algorithm 2 Simultaneous Perturbation Stochastic Approximation (SPSA)

```
1: Initialize  $\mathbf{w}_0$ , set constant learning rate  $\alpha$  and a small value for  $\epsilon$ 
2:  $K \leftarrow$  maximum number of iterations
3: for  $k = 1, 2, \dots, K$  do
4:   Generate  $\Delta_k$ , a vector with each element drawn from a Bernoulli  $\pm 1$  distribution
5:   Compute  $L^+ \leftarrow L(\mathbf{w}_k + \epsilon\Delta_k)$ 
6:   Compute  $L^- \leftarrow L(\mathbf{w}_k - \epsilon\Delta_k)$ 
7:   Estimate gradient:  $\mathbf{g}_k \leftarrow \frac{L_+ - L_-}{2\epsilon \text{VAR}(\Delta)} \odot \Delta$ 
8:   Update parameters:  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha * \mathbf{g}_k$ 
9: end for
10: return Optimized parameters vector  $\mathbf{w}_{k_{\min}}$ , and optimal Loss value  $L(\mathbf{w}_{k_{\min}})$ 
```

such as particle swarm optimization (PSO), the ant colony algorithm [57], the bat algorithm [58] and other heuristic methods. ESs are, as the name suggest, a class of "black-box" optimization methods inspired from natural selection with their goal being the optimization of a target function. Before delving into more advanced evolutionary strategies, we lay the general framework for ESs. To develop an intuition for evolutionary strategies, we recommend the illuminating and very well written blog "A visual guide to evolutionary strategies." by D. Ha [59].

4.2.1 Evolutionary strategy basics

First, let us start conceptually. Following the principle of natural selection, a generation or population of candidates is evaluated with a certain criterion. Following the performance evaluation comes a selection process where only a number of best individuals are selected. These then undergo a recombination process where their best features are combined, with potential random mutation to increase the diversity of candidates. Using this new recombined and mutated population, the next generation of offspring candidates is generated. This new generation embodies the characteristics of their parents and, thanks to their added genetic diversity, some of them will fulfill the selection criterion better than their ancestors. This process of selection, recombination then mutation and finally new generation is repeated, in a "survival of the fittest" loop until a satisfactory condition is met. From this nature-inspired loop let us explain perhaps the simplest yet very illustrative example of an ES. Our goal will be to optimize an objective function $L(\mathbf{w})$.

First, a random starting position \mathbf{w}_0 in the search space is set. Subsequently, a new population of solutions is created around that starting position. Mathematically this is usually done by drawing p sample points $\mathbf{w}_i, i = \{1, \dots, p\}$ for a random distribution $\mathbf{w} \sim \mathcal{N}(\mathbf{w}_{\text{mean}} = \mathbf{w}_0, \sigma^2)$, and usually a normal distribution is used. For clarity, we chose p to refer to the population size, usually denoted by λ , which can be confusing in the context of optics. In practice, the population is simply a list of p points each of which is a candidate solution to our optimization problem. These candidates are subsequently evaluated using the objective function $L(\mathbf{w})$. Following evaluation, a selection process is required, the simplest of which is to choose the best candidate

of said population $\mathbf{w}_{\text{best}} = \arg \min_{\mathbf{w}_i} L(\mathbf{w}_i)$. We then set this best candidate as the new mean for the next generation. In general, for all ESs, increasing p enhances the likelihood of generating better candidate points, albeit at the cost of more sampling. The pseudocode for this simple algorithm is described in Algorithm 3

Algorithm 3 Simple ES Optimization

```

1: Initialize  $\mathbf{w}_{\text{mean}} = \mathbf{w}_0$  randomly, choose  $\sigma^2$ 
2: Set number of generations  $G$  and population size  $p$ 
3: for  $g = 1$  to  $G$  do
4:   for  $i = 1$  to  $p$  do
5:     Generate  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ 
6:     Set  $\mathbf{w}_i = \mathbf{w}_{\text{mean}} + \epsilon_i$ 
7:     Evaluate fitness  $L(\mathbf{w}_i)$ 
8:   end for
9:   Select  $\mathbf{w}_{\text{best}} = \arg \min_{\mathbf{w}_i} L(\mathbf{w}_i)$ 
10:  Update  $\mathbf{w}_{\text{mean}} = \mathbf{w}_{\text{best}}$ 
11: end for
12: return  $\mathbf{w}_{\text{mean}}$  as the optimized solution

```

While very simple, Algorithm 3 lays the framework for more advanced ESs and can be improved with simple tweaks. First, instead of selecting only $\mathbf{w}_{\text{best}} = \arg \min_{\mathbf{w}_i} L(\mathbf{w}_i)$ we can select a subset \mathbf{W}_{best} of size denoted here by l of best candidates $\mathbf{W}_{\text{best}} = \{\mathbf{w}_{i_1}, \mathbf{w}_{i_2}, \dots, \mathbf{w}_{i_\mu}\}$ and take their mean for the next iteration $w_{\text{mean}} = \frac{1}{l} \sum_{j=1}^l \mathbf{w}_{i_j}$. In the literature, l is often denoted by μ , which can lead to confusion when dealing with Gaussian distributions. This variant of the simple ES is usually referred to as the μ, λ (in our case l, p) mutation strategy in the literature.

We will next present more advanced variants of this basic algorithm that include changes to normal distribution's standard deviation, elongating and changing its orientation to sample more along the direction of the best candidates.

4.2.2 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Building upon the previously described simple ES, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES or simply CMA) is a strategy that adapts, both, the mean and shape of a multivariate gaussian distribution. As its name suggests, this is done by adapting the covariance matrix of the distribution. It was first introduced in 2001 by N. Hansen and A. Ostermeier [60]. A remarkably well written tutorial has been published by the original authors of CMA in [61], and the authors also maintain libraries, both, in MATLAB and Python.

The CMA-ES algorithm is analytically quite heavy and as such we will not focus on specific mathematical equations and rather give an intuition for the algorithm. First, the central idea is to adapt a multivariate Gaussian distribution in a space of dimension D for which the general equation is given by:

$$f(\mathbf{w}) = \frac{1}{\sqrt{(2\pi)^D |\mathbf{C}|}} \exp\left(-\frac{1}{2}(\mathbf{w} - \mu)^T \mathbf{C}^{-1}(\mathbf{w} - \mu)\right), \quad (5)$$

$$\mathcal{N}(\mu, \sigma^2 \mathbf{C}), \quad (6)$$

where \mathbf{m} is the mean vector, \mathbf{C} is the covariance matrix and σ is the step size. We should note that in the context of CMA, σ does not refer to the standard deviation of the distribution but is a scalar that is used to determine how far we are from the mean. It is the covariance matrix \mathbf{C} that scales σ along certain directions to shape the distribution. Moreover, sigma is adapted independently from \mathbf{C} , and it's update depends on the history of the search and, in this sense, it works like a momentum term in optimization algorithms.

A simple pseudocode for the CMA-ES algorithm is given in algorithm 4, below:

Algorithm 4 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

```

1: Initialize population size  $p$ , number of elites  $l$ , mean vector  $\mu$ , covariance matrix
    $\mathbf{C}$ , step size  $\sigma$ 
2: Evaluate the initial population based on  $\mu$  and  $\mathbf{C}$ 
3: while stopping criteria not met do
4:   Generate  $p$  new offspring by sampling from  $\mathcal{N}(\mu, \sigma^2 \mathbf{C})$ 
5:   Evaluate fitness of each offspring
6:   Select top  $l$  elite offspring based on fitness
7:   Update  $\mu$  towards mean of selected elite
8:   Update  $\mathbf{C}$  to reflect distribution of selected elite
9:   Adapt step size  $\sigma$  based on success of the search
10: end while
11: return Best solution found

```

The CMA-ES algorithm is one of the most advanced ESs and is a widely used algorithm for black-box optimization, in the context of RL or hyperparameter optimization. Yet, its main drawback is that the computation of the covariance matrix \mathbf{C} is quite heavy and scales with $O(D^2)$. This makes it unsuitable for very high dimensional problems when computational resources are limited. The original authors proposed a modification simplifying the algorithm and reducing its scaling to $O(D)$, called the Separable CMA-ES (Sep-CMA-ES) [62], which can be seen as only computing the diagonal elements of covariance matrix \mathbf{C} .

A simple illustration of the CMA algorithm is provided below in Fig. 5. For illustration purposes, we start the algorithm at the upper right corner of the Rastrigin function and let it evolve and plot the distribution of the population on top of a contour colormap of the objective function. An additional ellipsoid is superimposed to represent the shape of the distribution. We can see how the ellipsoid first elongates along the direction of best rewards and then gradually contracts as the search converges to the global minimum.

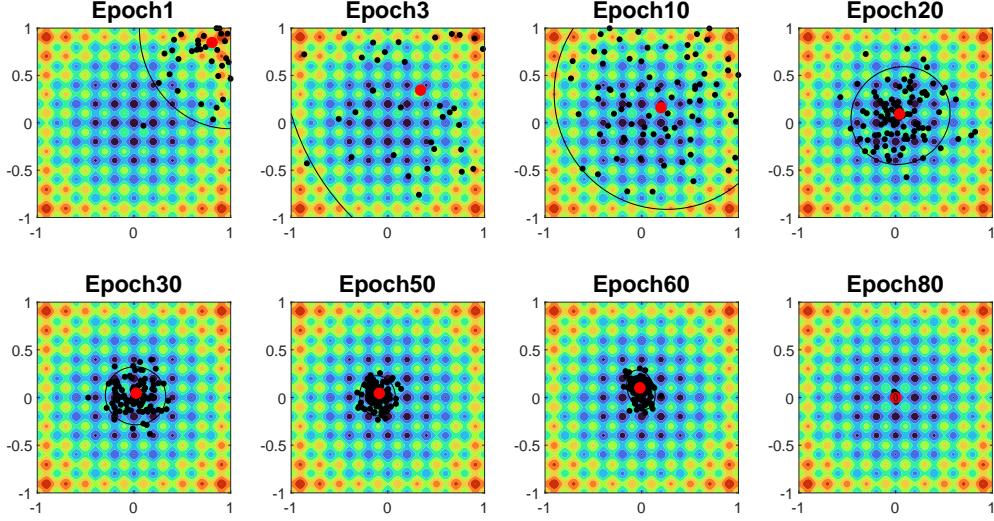


Fig. 5 CMA-ES algorithm behavior on the Rastrigin function in $D = 10$ dimensions, only the first two are shown. The mean of the distribution is represented by the red dot, the population by the black dots and the ellipsoid helps to visualize the shape of the distribution.

4.2.3 Parameter Exploring Policy Gradients (PEPG)

The Parameter Exploring Policy Gradients (PEPG) is quite similar to CMA in so far as it is a population-based method that uses information from the population to update the mean as well as standard deviation of said distribution. Yet, instead of modifying the covariance matrix directly, PEPG leverages the policy gradient theorem to estimate the gradient of an objective function with respect to the parameters of its distribution [63, 64]. This concept is used extensively in RL, where optimization problems can be high dimensional and rewards very sparse. In our subsequent explanation, we disregard the technical jargon, as it can be quite confusing and is irrelevant for our purposes.

Let us now explain how PEPG optimization is done in simple terms following the explanation presented in [59, 65]. First, we have an objective or fitness function we want to optimize $L(\mathbf{w})$. The central insight is that we will maximize the expected value of $L(\mathbf{w})$ when sampled according to distribution $P_\theta(\mathbf{w})$, where θ represents the parameters of this distribution. For example, it can represent a normal distribution $\mathcal{N}(\mu, \sigma)$, with $\theta = (\mu, \sigma^2)$. The expected value of $L(\mathbf{w})$ which we want to maximize is given by:

$$\mathbb{E}[L(\mathbf{w})] = \int L(\mathbf{w}) P_\theta(\mathbf{w}) d\mathbf{w}. \quad (7)$$

We now rewrite our optimization goal with θ as our optimization parameter. In simple terms we now have to optimize function $J(\theta) = \mathbb{E}[L(\mathbf{w})]$ with respect to θ . By acting on θ we influence $P_\theta(\mathbf{w})$ and improve the expected value of our fitness function. If the expected value of the fitness function improves, then the best candidate of the

population will be even better! Our goal is therefore to perform gradient ascent on $J(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta). \quad (8)$$

The derivative of $J(\theta)$ is given by:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \mathbb{E}[L(\mathbf{w})] = \nabla_{\theta} \int P_{\theta}(\mathbf{w}) L(\mathbf{w}) d\mathbf{w}. \quad (9)$$

We can permute the gradient ∇_{θ} and the integral $\int d\mathbf{w}$ since θ and \mathbf{w} are independent and get:

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} P_{\theta}(\mathbf{w}) L(\mathbf{w}) d\mathbf{w}. \quad (10)$$

We now use a "log derivative trick" given by:

$$\frac{\partial \ln(L(\mathbf{w}))}{\partial \mathbf{w}} = \frac{1}{L(\mathbf{w})} * \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}. \quad (11)$$

Multiplying by $\frac{P_{\theta}(\mathbf{w})}{P_{\theta}(\mathbf{w})} = 1$ inside the integral we get:

$$\nabla_{\theta} J(\theta) = \int \frac{P_{\theta}(\mathbf{w})}{P_{\theta}(\mathbf{w})} \nabla_{\theta} P_{\theta}(\mathbf{w}) L(\mathbf{w}) d\mathbf{w} \Leftrightarrow \nabla_{\theta} J(\theta) = \int P_{\theta}(\mathbf{w}) \nabla_{\theta} \ln(P_{\theta}(\mathbf{w})) L(\mathbf{w}) d\mathbf{w}. \quad (12)$$

We notice that Eq.12 can be seen as the expected value of a new variable $\nabla_{\theta} \ln(P_{\theta}(\mathbf{w})) L(\mathbf{w})$, therefore we have:

$$\nabla_{\theta} J(\theta) = \mathbb{E}[\nabla_{\theta} \ln(P_{\theta}(\mathbf{w})) L(\mathbf{w})]. \quad (13)$$

The expectation value $\mathbb{E}[\nabla_{\theta} \ln(P_{\theta}(\mathbf{w})) f(\mathbf{w})]$, can be approximated by sampling a sufficient number p of samples \mathbf{w}_i from $P_{\theta}(\mathbf{w})$:

$$\nabla_{\theta} J(\theta) \simeq \frac{1}{p} \sum_{i=1}^p \nabla_{\theta} \ln(P_{\theta}(\mathbf{w}_i)) L(\mathbf{w}_i). \quad (14)$$

Equation 14 is known as the policy gradient theorem. Using this theorem, we can now estimate the gradient on $\nabla_{\theta} J(\theta)$ and optimize our distribution $P_{\theta}(\mathbf{w})$ by acting on its parameters θ to maximize the expected value of the fitness function $L(\mathbf{w})$. In the case of a normal distribution, $P_{\theta}(\mathbf{w}) = \mathcal{N}_{\theta=(\mu, \sigma)}(\mathbf{w})$. We can therefore update μ and σ using the following update rules:

$$\mu \leftarrow \mu + \alpha \nabla_{\mu} J(\mu) \quad \text{and} \quad \sigma \leftarrow \sigma + \alpha \nabla_{\sigma} J(\sigma). \quad (15)$$

For a normal distribution,

$$P_{\theta}(x) = \mathcal{N}_{(\mu, \sigma)}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad (16)$$

we obtain, according to[63] :

$$\nabla_\mu \ln(P_\theta(\mathbf{w})) = \frac{\mathbf{w} - \mu}{\sigma^2} \quad \text{and} \quad \nabla_\sigma \ln(P_\theta(\mathbf{w})) = \frac{(\mathbf{w} - \mu)^2 - \sigma^2}{\sigma^3}. \quad (17)$$

The key insight here is that $\nabla_{\mathbf{w}} L(\mathbf{w})$ is unknown, yet we use the policy gradient theorem to estimate it using $\nabla_\theta \ln(P_\theta(\mathbf{w}))$. This can be thought of as an optimization by proxy, and works without making any assumptions on $L(\mathbf{w})$ itself. The pseudocode for a simple policy gradient optimization loop is given in Algorithm 5.

Algorithm 5 Simple Policy Gradient optimization

```

1: Initialize  $\theta = (\mu, \sigma)$ , set constant learning rate  $\alpha$ , set population size  $p$ 
2:  $K \leftarrow$  maximum number of iterations
3: for  $k = 1, 2, \dots, K$  do
4:   Generate  $p$  samples  $w_i \sim \mathcal{N}_\theta(\mathbf{w})$ 
5:   Compute  $f(\mathbf{w}_i)$  for each  $\mathbf{w}_i$ 
6:   Estimate gradient:  $\nabla_\mu J(\mu) \leftarrow \frac{1}{p} \sum_{i=1}^p \frac{\mathbf{w}_i - \mu}{\sigma^2} L(\mathbf{w}_i)$ 
7:   Estimate gradient:  $\nabla_\sigma J(\sigma) \leftarrow \frac{1}{p} \sum_{i=1}^p \frac{(\mathbf{w}_i - \mu)^2 - \sigma^2}{\sigma^3} L(\mathbf{w}_i)$ 
8:   Update parameters:  $\mu \leftarrow \mu + \alpha \nabla_\mu J(\mu)$ 
9:   Update parameters:  $\sigma \leftarrow \sigma + \alpha \nabla_\sigma J(\sigma)$ 
10: end for
11: return Optimized parameters vector  $\theta = (\mu, \sigma)$ , and optimal Loss value  $L(\theta)$ 

```

A notable flaw in the simple version of policy gradient optimization presented in Algorithm 5 is that both gradients are inversely proportional to the standard deviation σ . As the algorithm gradually converges to the global optimum, the standard deviation will shrink which will cause the gradients to explode. This instability similar to an overshooting oscillation problem can be mitigated by using additional momentum terms and applying a time dependent decay on the standard deviation as introduced in [65].

Figure 6 shows the PEPG algorithm in action on the Rastrigin function. We can see how the mean of the distribution is updated towards the global minimum and how the standard deviation is adapted and contracts when the search converges. In practice the distribution remains quite symmetric when using PEPG as opposed to CMA, which is because CMA adapts the off-diagonal terms of the covariance matrix as well. Yet, as PEPG does not compute any covariance matrix, it is more suitable for high dimensional problems as it scales in $O(D)$.

4.2.4 Particle Swarm Optimization (PSO)

PSO, unlike the previous methods, is inspired by the behavior and movements of animals such as flocks of birds. As such, it tries to mimic their movements as a way to "patrol" an optimization landscape looking for an optimal solution. Interestingly, the PSO algorithm was introduced by an electrical engineer and social psychologist, Kennedy and Eberhart in 1995 [66].

Rather than using a population of candidates and mutation, it uses a swarm of particles, each representing a candidate solution in the search space. Each particle has

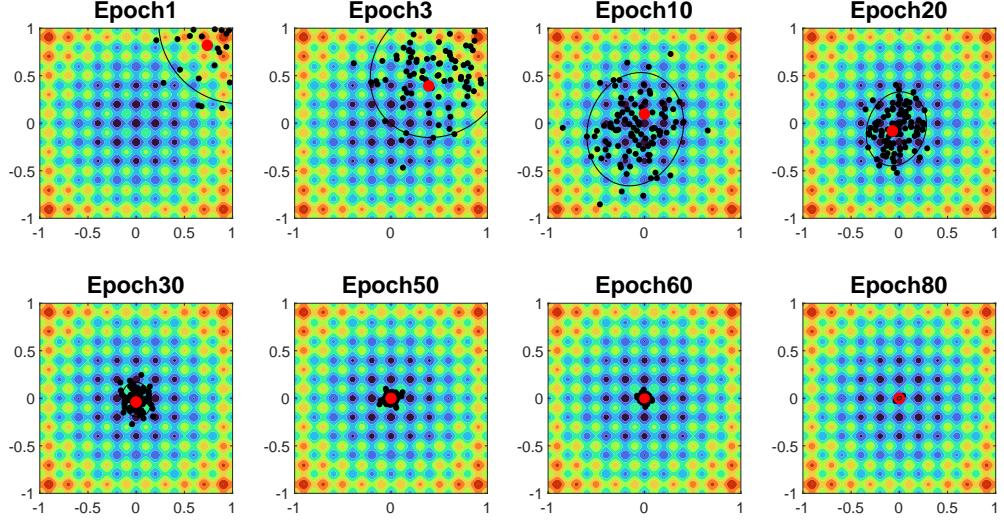


Fig. 6 PEPG algorithm behavior on the Rastrigin function in $D = 10$ dimensions, only the first two are shown. The mean of the distribution is represented by the red dot, the population by the black dots and the ellipsoid helps to visualize the shape of the distribution.

a velocity, patrols the search space and is attracted to, both, its personal best position and the global best position found by the swarm. These two components influencing the movement of each particle are referred to as the cognitive and social components, the former influencing a more local search, and the latter a more global one. A careful balance between cognitive and social components is needed to efficiently find optimal solutions. Each particle's velocity and position are updated at each iteration of the optimizing loop according to the following equations:

$$\mathbf{v}^i(t+1) = \omega \mathbf{v}^i(t) + c_1 r_1^i (\mathbf{p}_{\text{best}}^i - \mathbf{w}^i) + c_2 r_2^i (\mathbf{g}_{\text{best}} - \mathbf{w}^i), \quad (18)$$

$$\mathbf{w}^i(t+1) = \mathbf{w}^i(t) + \mathbf{v}^i(t+1), \quad (19)$$

where \mathbf{v}^i and \mathbf{w}^i are the velocity and position of particle i , $\mathbf{p}_{\text{best}}^i$ its personal best position of particle i , \mathbf{g}_{best} the global best position found by the swarm. All of these being vectors fulfill $\in \mathbb{R}^D$, D being the dimensionality of the search space. $\omega < 1$ is the inertia weight and tries to keep the particle on its current trajectory. c_1 and c_2 are the cognitive and social constants and control how much importance is given to refining the search of a particle i versus acknowledging the best result found by the swarm, and in literature their balancing is often referred to as the tradeoff between 'exploration' and 'exploitation'. r_1 and r_2 are random numbers drawn from a uniform distribution in $[0, 1]^D$. c_1 , c_2 , and ω are hyperparameters that require tuning depending on the optimization problem.

Figure 7 describes the update of the particles' velocity and position. The process can be illustrated by thinking of $(\mathbf{p}_{\text{best}}^i - \mathbf{w}^i)$, $(\mathbf{g}_{\text{best}} - \mathbf{w}^i)$ and $\mathbf{v}^i(t)$ as three vectors

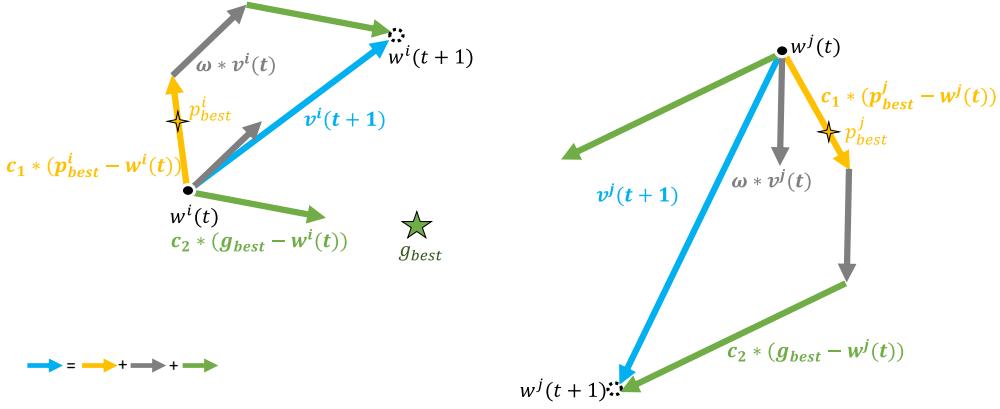


Fig. 7 PSO update diagram, showing the update of the velocity of two particles in the swarm w^i and w^j . The particles are attracted to their personal best p_{best}^i and the global best g_{best} . The inertia weight ω tries to keep the particle on its current trajectory.

scaled by their respective constants c_1 , c_2 , and ω yielding vector $v^i(t+1)$. The particle then moves in the direction of the sum of these 3 scaled vectors $v^i(t+1)$. The pseudocode for PSO is given in Algorithm 6.

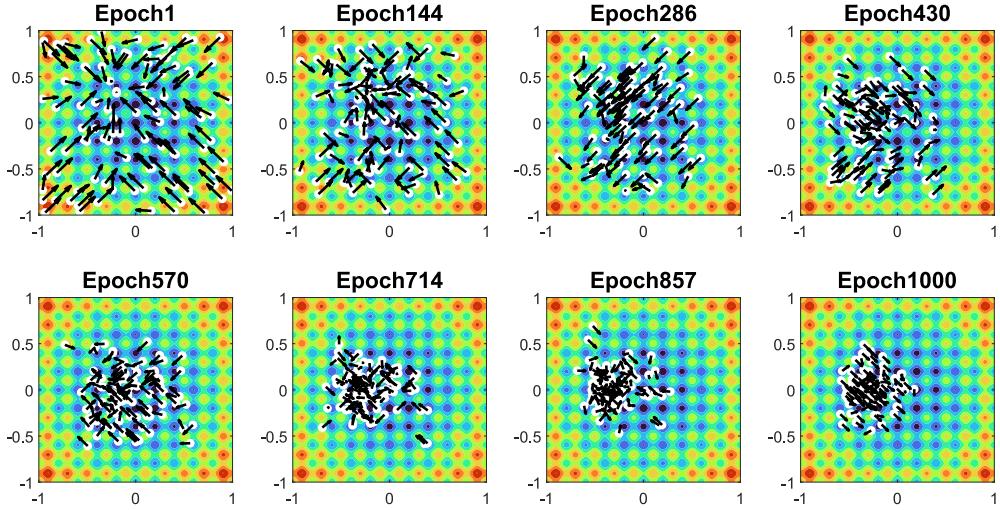


Fig. 8 PSO algorithm behavior on the Rastrigin function in $D = 10$ dimensions, only the first two are shown. The particles are attracted to their personal best p_{best}^i and the global best g_{best} . The inertia weight ω tries to keep the particle on its current trajectory.

PSO has been used for various optimization problems such as training NNs[67, 68], reservoir computers [69] as well architecture search [70] where the architecture and hyperparameters of NNs are optimized. PSO main advantage is its propensity towards

Algorithm 6 Particle Swarm Optimization (PSO)

```
1: Set swarm size  $p$ , maximum number of iterations  $K$ .
2: Initialize each particle  $i$  in swarm with random positions  $\mathbf{w}^i$  and velocities  $v^i$ .
3: Initialize hyperparameters  $c_1$ ,  $c_2$ , and  $\omega$ .
4: Evaluate the fitness of each particle at  $\mathbf{w}^i$ .
5: Set personal best  $\mathbf{p}_{\text{best}}^i = \mathbf{w}^i$  and global best  $\mathbf{g}_{\text{best}}$  to the best  $\mathbf{w}^i$  among all particles.
6: while termination criteria not met do
7:   for each particle  $i$  in the swarm do
8:     Update velocity:
9:      $v^i = \omega v^i + c_1 r_1 (\mathbf{p}_{\text{best}}^i - \mathbf{w}^i) + c_2 r_2 (\mathbf{g}_{\text{best}} - \mathbf{w}^i)$ 
10:    Update position:
11:     $\mathbf{w}^i = \mathbf{w}^i + v^i$ 
12:    Evaluate the new fitness of particle  $i$  at  $\mathbf{w}^i$ .
13:    if new fitness is better than  $\mathbf{p}_{\text{best}}^i$  then
14:      Update  $\mathbf{p}_{\text{best}}^i$  to the new position  $\mathbf{w}^i$ .
15:    end if
16:    if new fitness is better than the fitness of  $\mathbf{g}_{\text{best}}$  then
17:      Update  $\mathbf{g}_{\text{best}}$  to  $\mathbf{w}^i$ .
18:    end if
19:  end for
20: end while
21: return  $\mathbf{g}_{\text{best}}$ .
```

global search as the swarm is uniformly initialized in the search space. Yet, it can often result in a premature convergence to a local minimum. Figure 8 shows the convergence behavior of the PSO algorithm on the Rastrigin function. At first, individual particles are uniformly distributed and each of them has a given velocity and direction, the swarm-like behavior is quite apparent and over time particles tend to converge and group together to offer a more focused search. Yet, as shown in Fig. 8, the algorithm is unable to find to global minimum and prematurely converges to a local one.

4.3 Application to Toy problems

4.3.1 Objective Functions

Let us start by benchmarking the optimization algorithms on the Rastrigin function in $D = 1000$ dimensions, which offers a significant challenge, but is still quite low dimensional compared to problems like NN optimization. For population-based methods, we use a population size of 100. Figure 9(a) shows the so called score i.e. minimal value of the Rastrigin function achieved by each algorithm at every epoch i.e. iteration of the algorithm, while panel (b) shows the same information but now plotted against true time in seconds. This therefore includes the computational overhead of each algorithm.

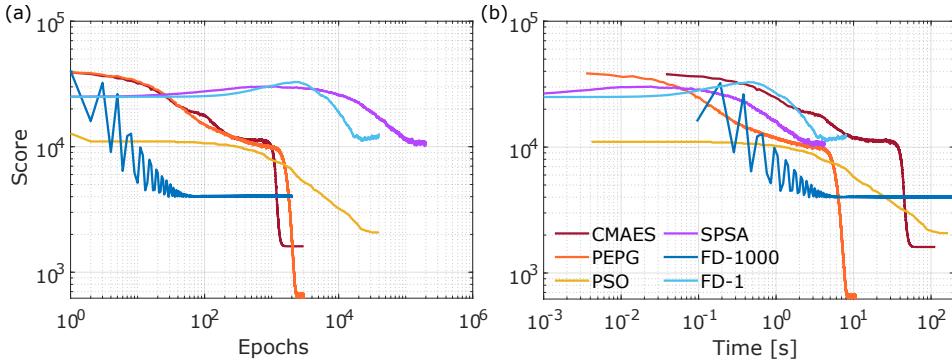


Fig. 9 Performance on the Rastrigin function for each optimization algorithm as a function of epochs (a) and of time in seconds (b).

The general trend is that population-based methods achieve lower values of the Rastrigin function than perturbative methods. Perturbative gradient estimates methods like FD and SPSA break down for this optimization problem. Indeed, the Rastrigin function is highly symmetric, which makes it hard for gradient based methods to escape local minima even when using momentum-based optimizers like ADAM. When increasing the number of directions along which the gradient is estimated to 1000 the FD method performs significantly better thanks to a more accurate gradient estimate, yet it also converges prematurely to a local optimum. It should be noted that probing all available dimension of an optimization problem independently in a sequential manner would take too long and is not feasible in practice.

The CMA-ES and PEPG algorithms are the best performing, with PEPG performing slightly better and converging significantly faster by requiring only $10s$, compared to CMA-ES that requires up to $60s$ even in this relatively low dimensional case. This sharp increase in convergence time for the CMA-ES algorithm stems from the computational overhead presented by the computation of the covariance matrix, and due to the scaling, this difference is exacerbated as the dimensionality of the optimization problem grows. In contrast, the particle swarm algorithm shows poorer performance as it struggles to navigate the landscape in a meaningful way. Even if PSO has the

same overhead as PEPG, it ends up converging slower than CMA-ES due to its poor ability to use information from the landscape. Moreover, PSO is the only algorithm that required us to do extensive hyperparameter tuning. Noteworthy is also that it is extremely challenging to develop an intuition for PSO hyperparameters, making their tuning process highly cumbersome.

Although the Rastrigin function is a good toy problem for benchmarking, studying and understanding optimization algorithms, it does not really fit our purposes since it is highly symmetric and is not representative of a real error landscape as would be found in a NN. The highly symmetric nature of the version of the Rastrigin function makes it easy to ‘cheat’. Indeed, increasing ϵ to $\epsilon = 0.1$ in the SPSA or FD algorithm allows it to ‘jump’ over local minima, leveraging more global information, and yielding significantly better performance than even PEPG as shown in Fig. 10. It should be stressed, however, that this is only possible because the structure of the Rastrigin function is pseudo periodic, moreover the higher ϵ leads to significant instability. Generally, such a high ϵ value no longer yields an accurate estimate of the gradient.

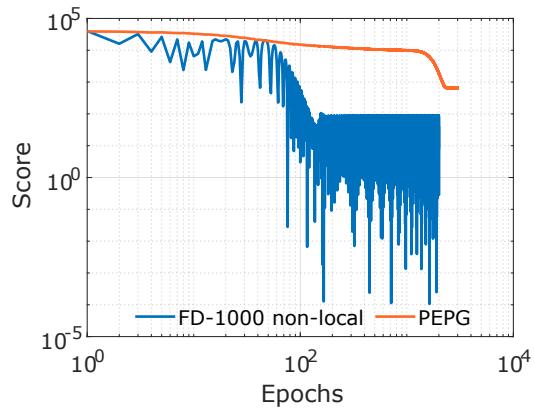


Fig. 10 Performance on the Rastrigin function for PEPG and FD using $\epsilon = 0.1$.

4.3.2 Neural network optimization

As stated in the previous section, rather than arbitrary optimization functions, a more useful test would be to benchmark all optimization algorithms in the context of NN training. A detailed breakdown of the shape of NN error landscapes is provided in [71]. Figure 11 illustrates the rough, irregular and steep landscape of a convolutional NN. Therefore, it becomes quite apparent that an artificial function, such as given by Rastrigin, will only provide us with limited information.

As such, we chose 2 classification tasks for a NN to solve, namely the IRIS [72] and MNIST [73] datasets. We chose these two datasets since MNIST provides high dimensional input images with 784 input features, while the IRIS dataset gives us the limit scenario of a very low dimensional dataset with only 4 input features. Table 2 bellow offers a quick overview of each dataset:

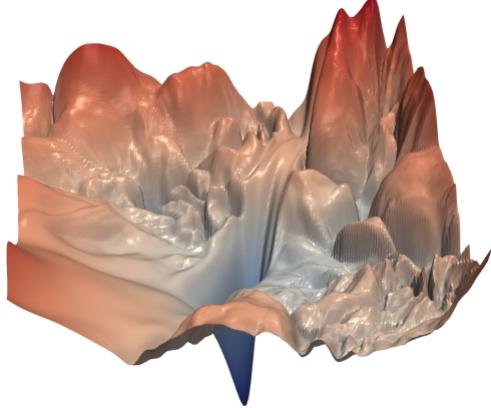


Fig. 11 Illustration of the error landscape of a convolutional NN, reproduced from [71].

Table 2 Summary of MNIST and Iris Datasets.

Dataset Feature	MNIST	Iris
Input Features	28x28 = 784 pixel images	4 flower measurements
Output Features	Digit labels (0-9)	Iris species (setosa, versicolor, virginica)
Number of Classes	10	3
Training Samples	60,000	105
Testing Samples	10,000	45

Like most image-based datasets, the MNIST dataset is quite high dimensional with a total of 784 input features. If we were to consider a FFNN of 100 nodes, then the number of trainable parameters would total $784 * 100 + 100 + 100 * 10 + 10 = 79510$, which include the input weights, node biases, the output weights, and output biases. Such a high number of parameters would make our study highly time-consuming, and would render the CMA-ES algorithm unusable. Therefore, to significantly reduce the computational cost of our analysis and particularly that of the CMA-ES algorithm, we will use a ConvNet architecture of ~ 12000 parameters on the MNIST dataset. The specific architecture used is presented in table 3 and is similar to the one used in [59]. On the contrary, the IRIS dataset is quite low dimensional with only 4 input features, and we will use a simple FFNN with 1 hidden layers of 10 nodes. The architecture is presented in table 4.

We can then compare all training algorithms in terms of convergence efficiency measured in seconds for the two datasets and compare them with BP for reference. Performance on the IRIS dataset, shown in Fig. 12 is less interesting and was performed as a sanity check, to see how all algorithms would perform on a low dimensional dataset. As expected, all algorithms can reach 100% classification accuracy in barely more than 1s. The task is too low dimensional to explicitly reveal any specific shortcomings of these algorithms.

Figure 13 showcases performance on the MNIST dataset. Here, BP achieves $\sim 99\%$ classification accuracy, which constitutes the ceiling against which we can evaluate other algorithms. BP is followed by PEPG at $\sim 98\%$ and CMA-ES at $\sim 96\%$. Crucially,

Layer	Type	Output Shape	Number of Parameters	Description
Input	Input Layer	(28, 28, 1)	0	Takes an input of size 28x28x1 (grayscale image)
Conv1	Conv2D	(28, 28, 8)	208	A convolutional layer with 8 filters of size 5x5, stride 1, and padding 2
MaxPool1	MaxPooling2D	(14, 14, 8)	0	A max pooling layer with a 2x2 window and stride 2
Conv2	Conv2D	(14, 14, 16)	3216	A convolutional layer with 16 filters of size 5x5, stride 1, and padding 2
MaxPool2	MaxPooling2D	(7, 7, 16)	0	A max pooling layer with a 2x2 window and stride 2
Flatten	Flatten	(784)	0	Flattens the output to a vector of size 784
FC	Fully Connected	(10)	7850	A fully connected layer with 10 output neurons
Output	Output Layer	(10)	0	Produces the final output of size 10

Table 3 Summary of the convolutional network architecture used on the MNIST dataset.

because of its heavy quadratic overhead, each iteration of the CMA-ES algorithm takes a significant amount of time, there is a factor ~ 30 between updates computed with PEPG and CMA-ES. In general, CMA-ES is seen as more robust and should achieve higher performance than PEPG, yet because of its expensive overhead we are not able to tune its hyperparameters in the present task. In practice, this makes PEPG more robust for high dimensional optimization problems under limited resources. The PSO algorithm performs poorly in general and is not able to converge to a satisfactory

Layer	Type	Output Shape	Number of Parameters	Description
Input	Input Layer	(4)	0	Takes an input of size 4 (features of the Iris dataset)
Hidden1	Linear	(10)	50	Fully connected layer with 10 output neurons
ReLU	ReLU Activation	(10)	0	Applies the ReLU activation function
Output	Linear	(3)	33	Fully connected layer with 3 output neurons (corresponding to the 3 classes of the Iris dataset)

Table 4 Summary on the FFNN architecture used on the IRIS dataset.

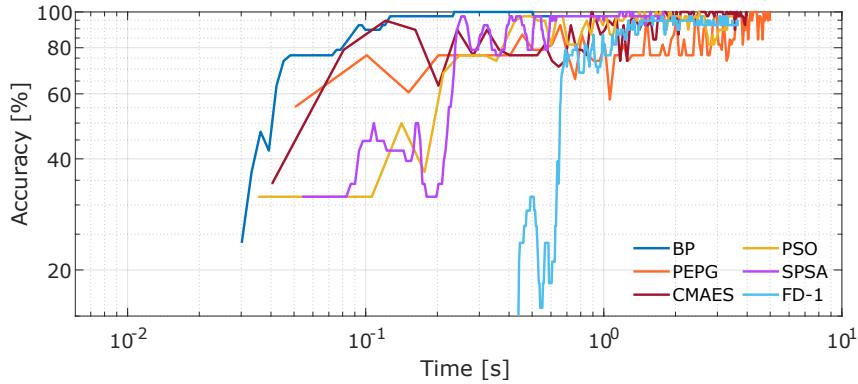


Fig. 12 Convergence of the different optimization algorithms on the IRIS dataset.

solution, it seems that it is not well suited to high dimensional optimization problems. All population-based methods were evaluated with a population size of 200.

Regarding perturbative methods, it should be stressed that in the context of digital NNs, there is no difference between the true gradient given by BP and a finite difference method that would probe all parameters because ϵ can be arbitrarily small. Indeed, as a method to debug digital NNs, FD can be used to perform so called "gradient checking" to make sure that computed NN gradients are correct. However, gradient

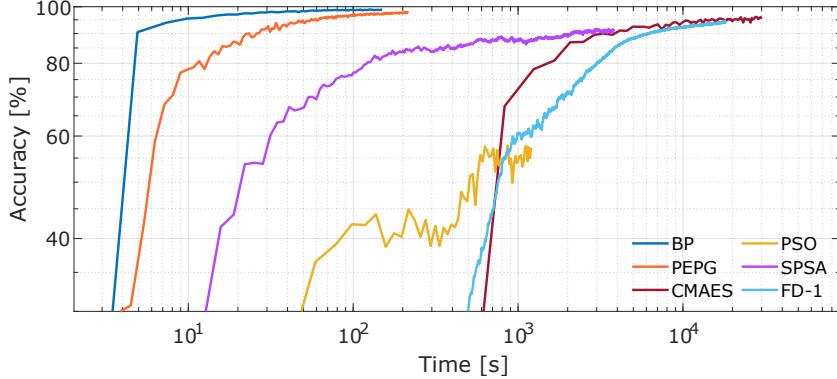


Fig. 13 Convergence of the different optimization algorithms on the MNIST dataset.

checking has largely become obsolete due to the widespread adoption of automatic differentiation techniques, which no longer require users to explicitly program gradient computations when using conventional NN libraries such as Pytorch or TensorFlow.

Surprisingly, SPSA converges significantly faster than FD-1 while having the same overhead, since both probe a single direction. The main difference here is that FD-1 probes a single parameter while SPSA probes a direction that is a linear combination of all parameters, enabling it to use more information and converge quicker. Indeed, since MNIST data is quite sparse, i.e. a significant portion of the image is just a dark background, FD-1 can spend quite some time optimizing irrelevant parameters, unlike SPSA which perturbs all parameters at the same time.

SPSA reaches $\sim 92\%$ accuracy on the MNIST dataset, which is quite impressive given its simplicity, while FD-1 also reaches $\sim 92\%$ but takes ~ 3 times longer, and then outperforms SPSA due to its highly accurate estimate of the true gradient. Regarding perturbative methods, it should be noted that ϵ values such as $\epsilon = 10^{-6}$ are unrealistic in hardware and would require more than 16 bits of resolution. A more detailed study of this aspect is presented in section 6.

Finally, we can study how performance scales with population size for the best population-based algorithm, PEPG. Figure 14(a) shows a study of accuracy as a function of the ratio between the population size and the total number of parameters to optimize in the convolutional network for the MNIST dataset. Interestingly, for this task performance saturates at $\sim 1\%$ corresponding to a population size of 113. It should be noted that this ratio is not absolute in any sense and should, in principle, be highly task dependent.

Intuitively, we can understand how only a small population size is needed by looking at the MNIST dataset itself, shown in Fig. 14(b). The images are quite sparse and mostly consist of a dark background, therefore only a small number of pixels corresponding to the digits themselves are actually relevant for classification. One could imagine how with another dataset, such as fashion-MNIST, where the images are fuller and more complex, a larger population size would be needed to optimize the network. Indeed, when using PEPG on the fashion-MNIST dataset, we found that performance

saturates for a population size ratio of $\sim 2\%$ corresponding 226 samples, confirming our hypothesis.

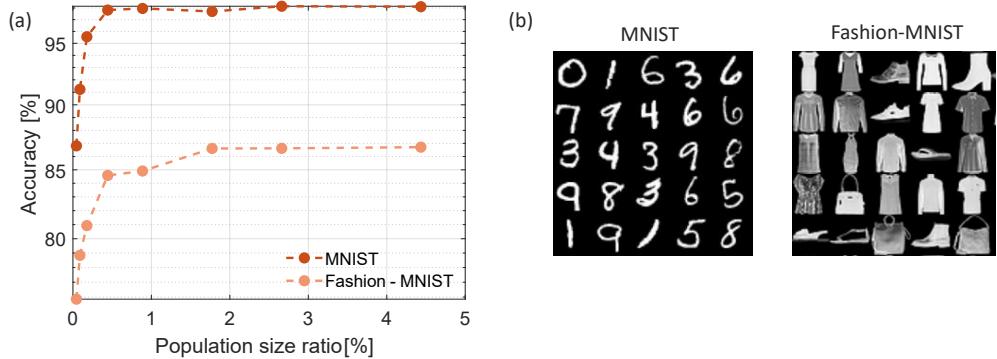


Fig. 14 Accuracy of the PEPG algorithm on the MNIST dataset as a function of the population size ratio to the total number of parameters in the network.

To summarize, in these previous sections, we laid the groundwork for future improvements to our ONN experimental setup. These improvements stem from insights given by a ceiling analysis performed on a software NN. This ceiling analysis led us to three central conclusions. Firstly, having both positive and negative weights is crucial in a NN. Secondly, these weights should have sufficient resolution to account for the complexity of the task. And finally, having trainable input weights as opposed to a fixed random mapping is crucial for performance. These conclusions, while seemingly trivial, have a profound impact on the design of our ONN, particularly when it comes to weight optimization. We therefore presented several training strategies of varying complexity and applied them to, both, toy problems and in the context of NN optimization on the MNIST and IRIS dataset. We were able to verify that several of these algorithms, particularly PEPG, SPSA and CMA-ES show promising performance and varying degrees of overhead cost. In the next sections, we will use these insights to improve our ONN experimental setup and use the hardware compatible algorithms presented here to train a fully tunable ONN on the MNIST dataset.

5 Towards a Fully Tunable Optical Neural Network

5.1 Large area VCSELs

For our proof-of-concept hardware NN implementation, we built an optical reservoir or extreme learning machine for image processing, with a VCSEL playing the starring role. This, in turn, begs the following question, "What type of VCSEL should we use?" Indeed, VCSELs come in various shapes and sizes, yet, as stated previously, we want our NN to process information in a parallel fashion. To achieve that we will use a spatially multimode VCSEL. In this spatially-multiplexed approach, transverse modes of the VCSEL will be related to neurons of the network. In other words, the more modes our VCSEL can support, the larger its potential information processing capabilities.

As a first approximation the mode profile, i.e. the spatial distribution of light that a semiconductor laser exhibits, results from interplay between two physical effects. First, the geometrical properties of the cavity provide confinement conditions that support certain optical modes. These are solutions to Maxwell's equations inside the medium, and are referred to as "cold cavity" modes, as they do not take into consideration any carrier effects via current injection, i.e. conditions as encountered as the device is pumped. When current is injected into the cavity, charge carriers are created and, via stimulated emission, they provide gain to certain modes of the cold cavity which in turn become lasing modes. This means that in practice, when the device is switched on and imaged on a camera, the VCSEL's emission mode profile is determined by the optical modes that overlap the most adequately with the distribution of carriers in the device. In simple terms, there is a number of carriers available to provide gain and different optical modes "compete" for these carriers. Finally, other physical properties such as temperature and carrier distribution can also influence the refractive index distribution which in turn influences the mode profile.

The multimodal nature of the LA-VCSEL is a central point in our NN design. As a thought experiment, let us consider the case of a single mode VCSEL. For such a device, spatially diverse input data is mapped onto the only mode that the VCSEL supports, making it therefore impossible to spatially distinguish different inputs. Our VCSEL should therefore be sufficiently multimode in order to spatially separate input data. Finally, the multimodal property of the laser is a product of the cavity geometry, namely its dimensions and shape. The following discussion explores various LA-VCSEL designs, highlighting those best suited for our application.

5.1.1 Circular cavity VCSELs

Circular cavity VCSELs represent the overwhelming majority of, if not all devices in commercial use today. Two other types of cavities, namely chaotic cavities and photonic crystal cavities, are generally confined to academic research [74]. Circular cavity VCSELs are robust and well-studied devices, yet because our goal is to build an ONN based on the spatial multiplexing of modes, we need to choose devices which inherently maximize the number of supported spatial modes. In practice, this means choosing bigger devices. For technological applications, circular VCSELs are generally designed to operate in a single mode (or near single mode) regime which means

the commercial circular oxide aperture VCSELs are quite small, usually under $\sim 5 \mu\text{m}$ in diameter. Indeed, while LA-VCSELs are desirable as they provide more optical power, they are heavily multimode. This is quite advantageous for our purposes, but makes them undesirable for industrial applications such as camera auto-focus optics or LiDAR pointers [75]. As a result, the industry has shifted towards arrays of single mode VCSELs rather than engineering bigger VCSELs.

Figure 15 shows VCSEL emission mode profiles for 3 VCSELs of different emitting diameters, respectively with apertures of 8, 20, and 60 μm . The smallest device shows a single mode Gaussian intensity distribution, while the bigger devices show increasingly high order ring-like mode profiles, also referred to as daisy modes[76].

Bigger VCSELs support more optical modes, should we therefore just use or design the biggest VCSEL possible? Unfortunately, the answer is not obvious, as small devices have a higher concentration of charge carriers close to the center of the cavity. However, as the diameter of the device increases, the emission profile becomes exceedingly ring-like and the center of the device exhibits a drop in power density. Indeed, carriers are injected into the device's active region, the quantum well, by the electrical contacts located at the rim around the VCSEL's aperture. Unfortunately, carrier diffusion is limited to $\sim 10 \mu\text{m}$ [77], and increasing the device's diameter beyond this distance results in a diminishing carrier concentration towards the device's center. As a result, high order ring modes have access to most of the device's gain.

As a consequence, with substantially bigger devices, most of the LA-VCSEL's central area is dark. The result is an increasingly fraction of the device area that cannot achieve lasing, which consequently results in weaker input-output nonlinear transformation. Therefore, for this type of cavity design, there is a tradeoff between the total area of the VCSEL and its lasing or 'active' area. Furthermore, for the same reason, bigger devices have higher threshold currents, and in our experience, devices exceeding diameters of 100 μm and higher in diameter were not able to lase because before crossing the lasing threshold, the thermal rollover [78] resulted in a too-strong reduction in optical gain. How then can we make more efficient use of an LA-VCSEL's area, i.e. turn more of the VCSEL area active? The so called 'chaotic cavity VCSEL' design, addresses this point [79, 80].

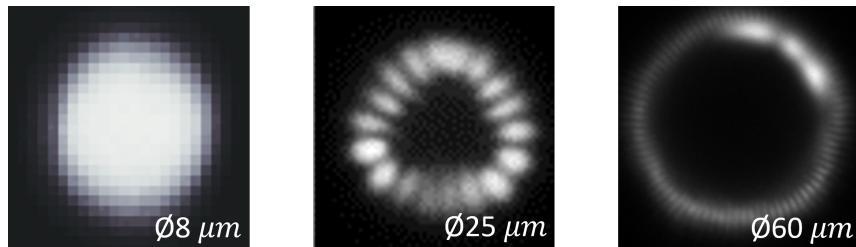


Fig. 15 Emission mode profile for 3 different VCSELs with oxide aperture diameters (ϕ) of, respectively, ~ 8 , 20, and 60 μm .

5.1.2 Chaotic cavity VCSELs

Historically, chaotic cavities were studied and proposed as a solution to for example, provide more directional emission from the cavity, especially in the context of micro-lasers [81, 82]. In the context of LA-VCSELs, chaotic cavities, i.e. cavities breaking their circular symmetry, have been used primarily as a way to achieve a more uniformly active mode profile, a higher density of supported optical states [83], as well as reduction in the mutual coherence between these modes. The reduced coherence in particular is interesting for applications in speckle-free imaging and free-space optical communications [79, 80].

In our case, by breaking the circular symmetry of the cold cavity, the high order ring modes are no longer supported for certain designs, which in turn means that optical gain can be "used" by other modes which have a more distributed profile. In practice, this means that a higher fraction of the VCSEL's area is active and supports lasing emission. In turn, this should in principle increase the number of neurons we can use. In [79, 80], the authors report an increased fraction of the device's lasing surface and an increase in the number of modes when switching to a chaotic LA-VCSEL cavity design. In addition, in [83] the authors report an increase in emitted power and in quantum efficiency when switching to a chaotic cavity design. In addition the more uniform density of optical modes creates less modal competition, which in turn helps the overall efficiency of the device as the recombination of carriers through non-radiative processes is reduced [83].

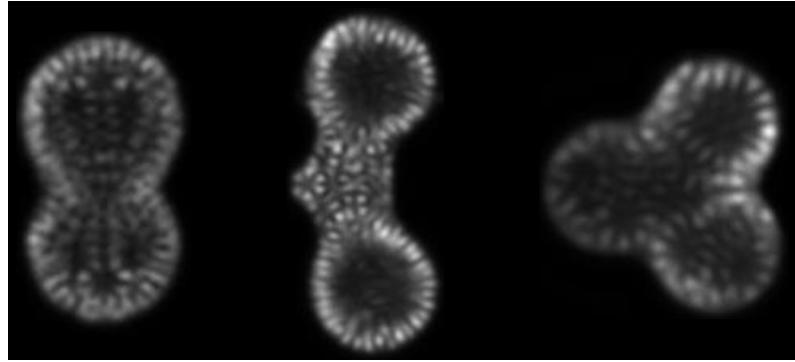


Fig. 16 Chaotic cavity VCSELs with different geometries, showcasing a more active mode profile with less non-lasing regions as compared to the circular devices in Fig. 15.

Figure 16 shows the emission mode profile of several LA-VCSELs with non-circular symmetric geometries, showcasing a more active mode profile with less dark regions. The study of the cavity geometry's influence on computational performance remains an extremely interesting and so far unexplored research direction. In [84] the authors discuss possibilities of tuning the dynamical response of microlasers by adjusting the cavity geometry, with a potential avenue for chaos-based applications leveraging compact microlasers. Potential applications for these devices are the development of

high-power broad-area lasers with stable dynamics and compact lasers for chaos-based applications [83].

5.2 Updated experimental setup

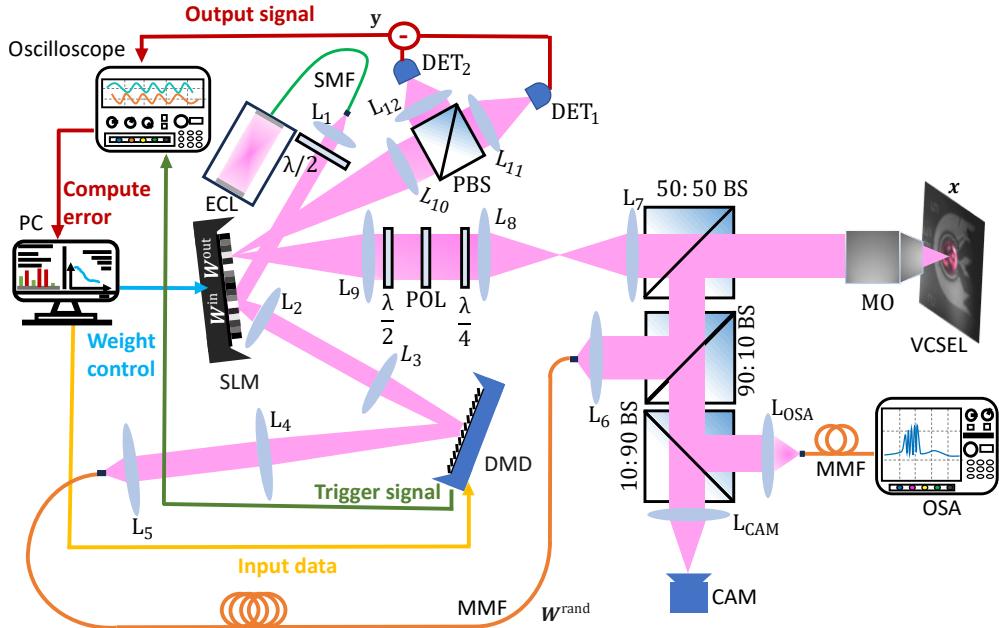


Fig. 17 Updated ONN setup. The ONN is now fully tunable and no longer is a reservoir, a single SLM is used to provide, both, input and output weights using, respectively, phase and intensity modulation via a PBS.

The experimental setup presented in Fig. 17 is an updated version of the first experimental setup presented in [29, 30, 41]. Taking into account the results of the ceiling analysis presented in section 2, we have updated our experimental setup to include several new features. An SLM was added before the input DMD to provide tunable input weights, while the output DMD was replaced by the same SLM. The SLM screen is large enough to accommodate two beams located on distinct spatial positions to prevent crosstalk. The first corresponding to the input laser beam, the second to the LA-VCSEL. The input weights apply up to 2π phase modulation to the input data displayed on the DMD before mixing in the MMF, therefore providing positive and negative input weights. Output weights are realized using an intensity modulation configuration, using a polarizing beamsplitter (PBS) and two detectors on each of the PBS-outputs. The detectors' outputs are subtracted electronically in real-time, which in turn provides positive and negative output weights.

The number of pixels used on the SLM is determined by the size of the input beam and the LA-VCSEL image on its surface, respectively, and their position is determined

by a knife edge measurement while the SLM is in intensity modulation. Finally, using an SLM (Santec LCOS-SLM200) provides up to 10-bit resolution on the weights. The following subsections will discuss the implementation of these improvements in more detail. A simplified diagram illustrating the scheme of our fully tunable ONN is presented in Fig. 18.

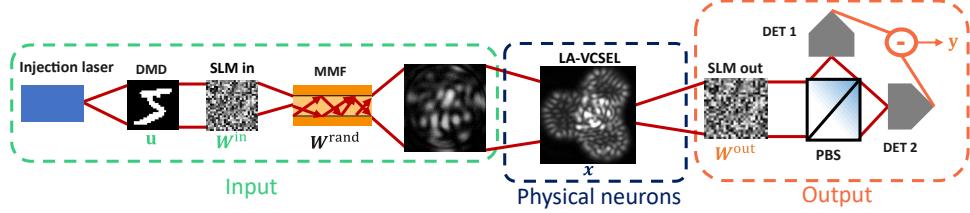


Fig. 18 Simplified experimental scheme of the fully tunable ONN.

5.2.1 Input layer

In our implementation of optical input weights, a collimated input beam was reflected off the SLM's surface onto the DMD. The modulated input data was then coupled into the MMF via imaging. To ensure we are fully modulating the phase of the input beam, we need to align the input beam's polarization to the SLM's slow axis. This can be achieved by displaying a blazed grating on the SLM and rotating the input polarization using a half-waveplate to maximize the intensity of diffraction orders, as shown in Fig. 19.

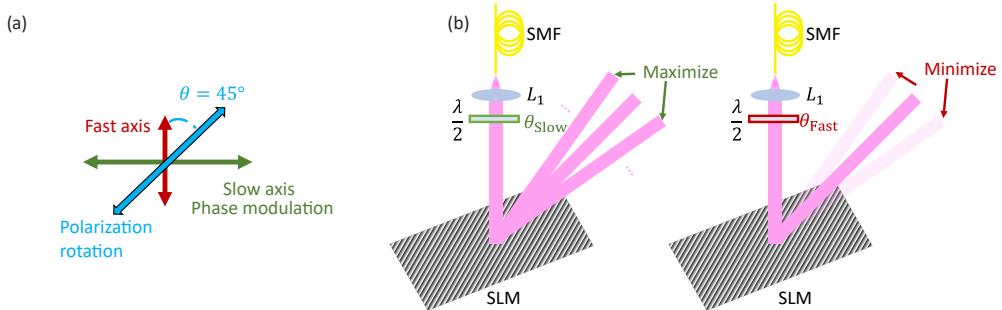


Fig. 19 (a) Schematic of the the SLM's refractive index-ellipsoid. (b) SLM displaying a blazed grating, the input polarization is rotated using a half-waveplate to maximize the intensity of high diffraction orders to ensure phase-only modulation.

Via a knife edge measurement, we determine the number of SLM pixels, i.e. input weights that can be used on the input beam. Furthermore, we can adjust the number of input weights by defining a software weight matrix of an arbitrary size smaller than

the number of illuminated SLM pixels and resizing it via interpolation to fit the area corresponding to the input beam. In this context, each weight corresponds to a super pixel on the surface of the SLM.

Yet, because of their size of $8 \mu\text{m}$, the SLM pixels cause diffraction, which in turn creates a loss of coupling efficiency at the MMF due to high NA components of the reflected beam. This ultimately results in a loss of information and reduces the injected power ratio and includes a weight-dependent artificial loss, which is detrimental in terms of performance.

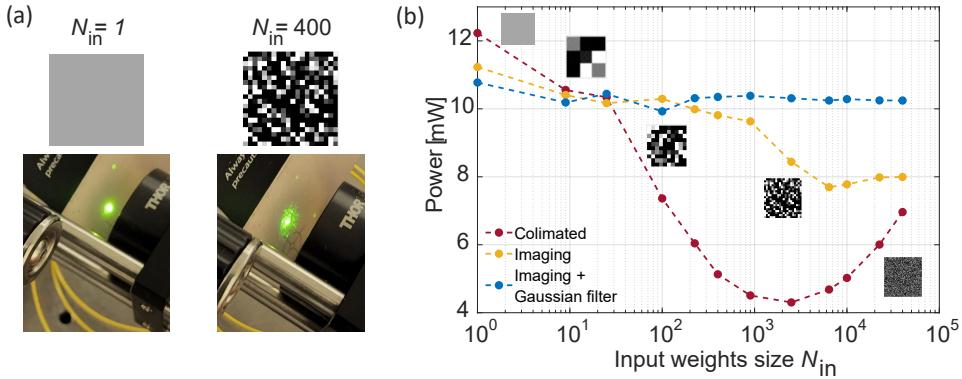


Fig. 20 (a) Picture showing the effect of diffraction for larger input weight matrices. (b) Power coupled into the MMF as a function of the input weight matrix size.

Figure 20 shows the effect of diffraction on the input beam. The number of input weights N_{in} is progressively increased, while the power coupled into the fiber is continuously monitored. As N_{in} grows, the size of super pixels decreases, which in turn increases the effective NA incident on the MMF, which reduces the coupling efficiency, as when the beam's NA starts exceeding the MMF's NA or when the resulting collimated beam diameter exceeds the imaging lens's diameter, see red curve in Fig. 20(b). After a certain point, the power coupled into the MMF starts to increase again, this is due to an increased spatial frequency that is so high that the phase is no longer modulated, i.e. there is a drop in diffraction efficiency linked to high spatial frequency. Figure 20 (a) shows a picture of the beam before L_5 in the case where $N_{\text{in}} = 1$ and $N_{\text{in}} = 400$ respectively, and the effect of diffraction is quite clear; a higher NA translates into a bigger beam diameter in collimated space in between L_4 and L_5 . To mitigate this effect, we decided to image the surface of the SLM onto the DMD using a 2-lens Relay-lens system. This means that the beam no longer propagates in collimated space from the SLM to the DMD, reducing diffraction-induced losses between lenses due to eventual beam clipping. This greatly alleviate the power loss at the MMF coupling stage, as shown by the yellow curve on panel (b).

Furthermore, knowing the lenses in our system we can estimate the smallest feature size on the SLM that will couple into the MMF before the corresponding NA will

exceed the MMF's NA. Knowing this characteristic feature size, we can apply Gaussian filtering to the input weights to prevent features from going below said size, which yields the blue curve on Fig. 20(b). To estimate the characteristic feature size, we use the following simple approximations:

$$\text{NA}_{\text{DMD}} = \frac{f_2}{f_3} * \text{NA}_{\text{SLM}}, \quad (20)$$

$$\text{NA}_{\text{MMF}} = \frac{f_4}{f_5} * \text{NA}_{\text{DMD}}, \quad (21)$$

$$\text{NA}_{\text{MMF}} = \frac{f_2 * f_4}{f_3 * f_5} * \text{NA}_{\text{SLM}}. \quad (22)$$

Next, we can estimate the diffraction limit of the SLM's characteristic feature size x_{SLM} by using the Rayleigh criterion:

$$\text{NA}_{\text{SLM}} = \frac{\lambda}{x_{\text{SLM}}}, \quad (23)$$

Where $\lambda \simeq 975$ nm is the wavelength of the input laser. Combining previous equations yields:

$$x_{\text{SLM}} = \frac{\lambda * f_2 * f_4}{\text{NA}_{\text{MMF}} * f_3 * f_5}. \quad (24)$$

Finally, using 8 μm as the SLM's pixel pitch, we get the number of pixels that have to be used to define our Gaussian filter kernel. Using $\lambda \simeq 975$ nm, $f_2 = 100$ mm, $f_3 = 100$ mm, $f_4 = 250$ mm, $f_5 = 20$ mm, $\text{NA}_{\text{MMF}} = 0.22$, we find that the Gaussian kernel should be at least ~ 2 pixels wide to ensure that the smallest feature size on the SLM still couples to the MMF. We use a Gaussian kernel of width 4 to filter the input weights. While operating the setup, input weights are rescaled to the interval [0, 1023] to match the 10-bit resolution 2π modulation of the SLM.

5.2.2 Output layer

The modifications to the output layer were quite significant. We now use intensity modulation, and two detectors, one for positive, the other for negative weights. First, to ensure we are in intensity modulation, the incident LA-VCSEL light needs to be linearly polarized and aligned at 45 degrees with respect to the slow axis. To achieve this, the LA-VCSEL goes through a quarter waveplate and linear polarizer, which are then adjusted to maximize the intensity transmitted through the linear polarizer. Subsequently, a half waveplate is rotated to align the linearly polarized LA-VCSEL emission to 45 degrees with the SLM slow axis. This means that the SLM is effectively doing polarization rotation of the LA-VCSEL light. Finally, a PBS acts as a second polarizer to achieve intensity modulation. The two arms of the PBS are polarized perpendicularly to each other, and the output of the detectors DET_1 and DET_2 are subtracted electronically in real-time to achieve positive and negative weights.

Figure 21 shows how we achieve positive and negative weights. Output weights in the interval $[-1, 1]$ are rescaled to the interval corresponding to the range between the minimum and maximum of intensity modulation the SLM provides i.e. $[\sim 170, \sim 660]$.

Finally, the position of the output weights is determined via a knife edge measurement as with one of the two photodetectors to ensure that input and output weights do not overlap.

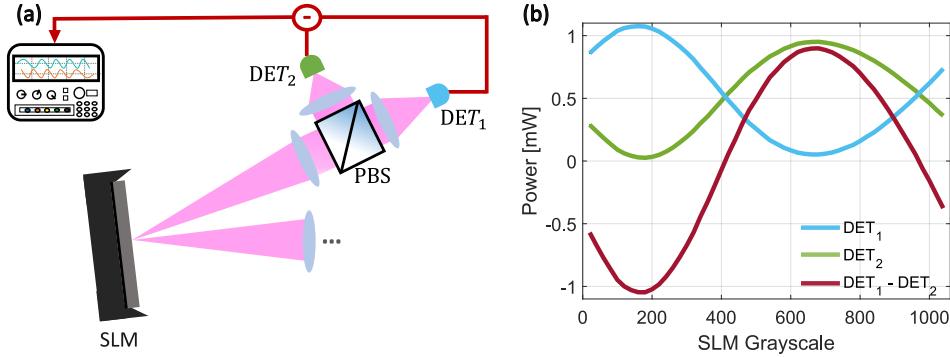


Fig. 21 (a) ONN output layer using a SLM, a PBS and two photodetectors. (b) Optical power on DET₁ and DET₂ as a function of the SLM grayscale showing 2π intensity modulation. The signals from DET₁ and DET₂ are subtracted to yield positive and negative output weights.

6 Benchmarking Training Strategies

After updating the experimental setup to include input and output weights, we now conduct an analysis of the training strategies presented in section 4. We study the impact of the main hyperparameters within each optimization algorithm on performance. Due to the high number of hyperparameters, we use "one-vs-all" classification on the hardest digit of the MNIST dataset, which is digit eight. The main limiting factor in this regard is the slow response time of the SLM of ~ 3 Hz and the scalar output given by the single photodetector. In conjunction, these two factors make hyperparameter tuning a time-consuming process that is impractical on the full MNIST dataset. This section aims to provide guidance for the hyperparameters in each optimization algorithm and gives valid ranges for good performance in a given context. We then use the best hyperparameters for each strategy to compare them in terms of performance.

6.1 Hyperparameter tuning of training algorithms

In this section, we solely focus on finding appropriate hyperparameter ranges for all training strategies. Due to hyperparameter scans taking a significant amount of time, numbers presented here are not comparable across different algorithms. The next section focuses on a direct comparison of relevant algorithms using the best hyperparameters found in our analysis. All hyperparameter dependencies were conducted under optimal injection locking conditions, as presented in [29].

6.1.1 Finite difference

The finite difference (FD) method is the easiest to implement conceptually, nevertheless, in-hardware learning adds a significant layer of complexity. Crucially, as a result of hardware resolution and noise, infinitesimal values of ϵ would not yield any change in the output as measured by the oscilloscope with an 8-bit resolution when using its full range. This effectively sets a lower bound on ϵ . Moreover, the optimal value of ϵ should, in principle, depend on the size of the super pixels on the SLM representing the weights. Indeed, the bigger the superpixel, the more optical intensity it receives, which makes it easier to detect changes at the output when perturbing said weight. Therefore ϵ should increase as the number of superpixel increases (i.e. size of superpixels decreases). First, let us focus on the output parameters, and Fig. 22 shows the results of the hyperparameter scan for the output weights.

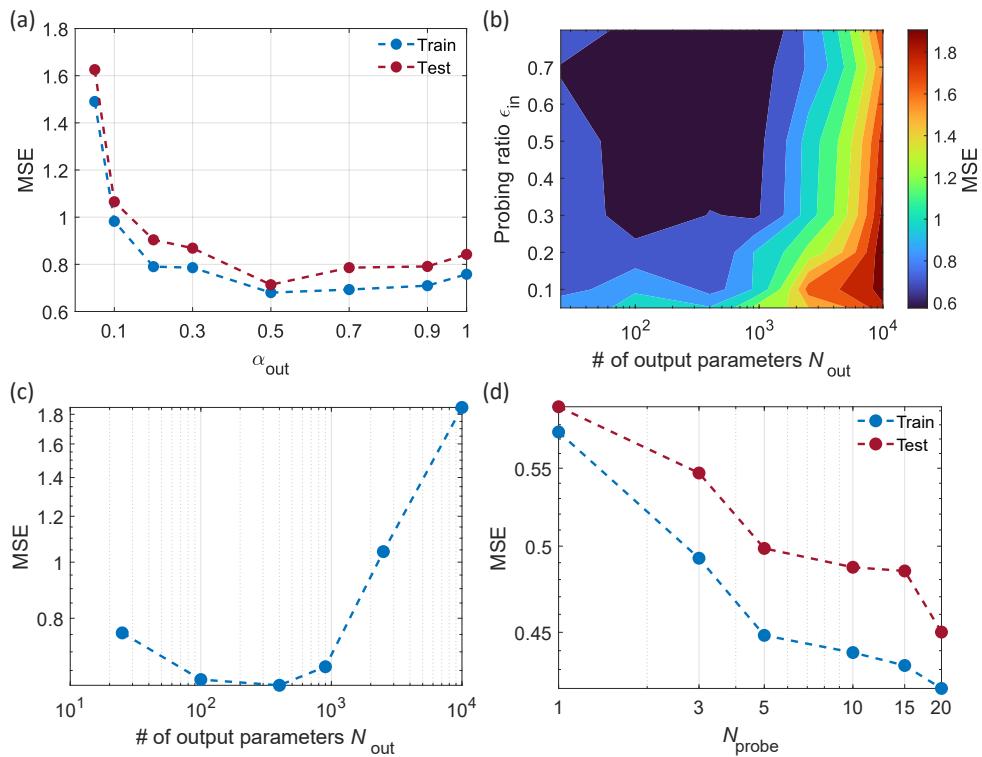


Fig. 22 Hyperparameter scan for the output weights using the finite difference method. (a) MSE as a function of the learning rate α_{out} . (b) Colormap of the MSE as a function of ϵ and N_{out} . (c) MSE as a function of N_{out} . (d) MSE as a function of N_{probe} .

First, the influence of the learning rate α_{out} is studied in Fig. 22(a), showing that $\alpha_{\text{out}} = 0.5$ is an optimal value. Then, we show how ϵ changes with the size of the output weight matrix in Fig. 22(b). Interestingly, despite what we might assume, as the size

of the output weight matrix increases, the optimal values of ϵ remain unchanged, our initial assumption is therefore false. In addition, optimal values of ϵ are quite high at around ~ 0.5 , while output weights are in the interval $[-1, 1]$, this highlights the rather counterintuitive nature of hardware learning as such a high ϵ value would be absurd in the context of a software NN. Furthermore, performance degrades significantly when increasing the number of output weights. As the number of superpixels increase, the dimensionality of the search problem also increases, as a consequence the FD methods should need more iterations to reach the same level of performance. Consequently, for a fixed training time, increasing the dimensionality of the search space i.e. number of output weights results in a significant decrease in performance, showing that the optimal size of the output weight matrix for the FD method is only $N_{\text{out}} = 400$ as shown in Fig. 22(c). This constitutes a major limitation of the FD algorithm in the context of our ONN and severely diminishes its applicability.

Next, we studied the impact of the number of probed weights per epoch N_{probe} . Here N_{probe} weights are each perturbed sequentially, i.e. one at a time, to build a gradient vector and they are then updated at the same time. In terms of scaling this means that if $N_{\text{probe}} = 20$, each epoch takes 20 times longer, as 20 superpixels are perturbed one after the other. Figure 22(d) shows the impact of N_{probe} on the MSE for the output weights when training for a fixed number of epochs. We can see that performance increases with N_{probe} , as sampling more coordinates i.e. superpixels yields a more accurate estimate of the gradient vector before updating the weights at every epoch. Yet this comes at the cost of a longer training time. To investigate the influence of N_{probe} on convergence speed, we study the convergence of the FD method as a function of training time for different values of N_{probe} , as shown in Fig. 23. Crucially, all curves overlap and converge to the same level showing that $N_{\text{probe}} = 1$ should reach the same performance level as $N_{\text{probe}} = 20$ provided they are both trained for the same amount of time.

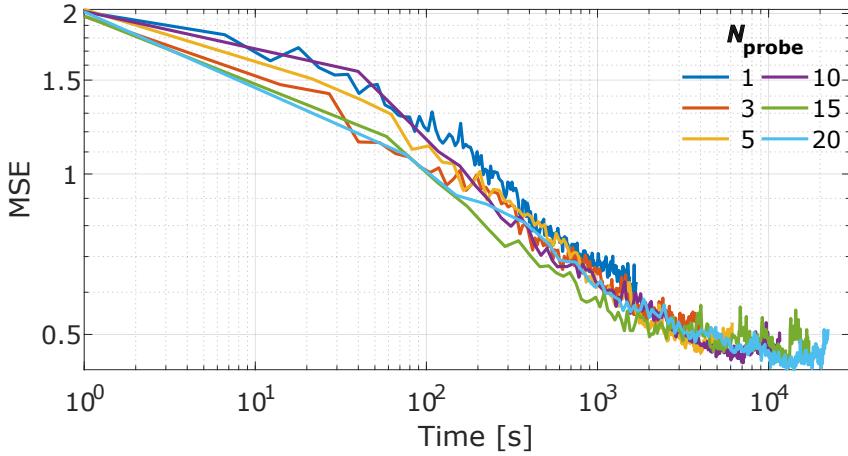


Fig. 23 Performance of the FD method as a function of training time for different values of N_{probe} .

We did not study the influence of the FD methods on the input weights, as that would add an additional layer of complexity to the problem, and as shown by our measurements, the FD method is not well suited to high dimensional problems.

6.1.2 Simultaneous Perturbation Stochastic Approximation

SPSA can be viewed as a more robust and significantly more efficient variation on the FD method. Moreover, since all weights are perturbed at the same time, we expect ϵ to be smaller. Finally, we should not in principle assume that input and output weights have the same impact on the loss function minimization and probing. In practice, this would mean that we would have 4 different hyperparameters, ϵ_{in} , α_{in} , ϵ_{out} and α_{out} . For simplicity, we will assume input and output parameters are independent and scan them independently in order to avoid high dimensional hyperparameter grid which can be highly time-consuming.

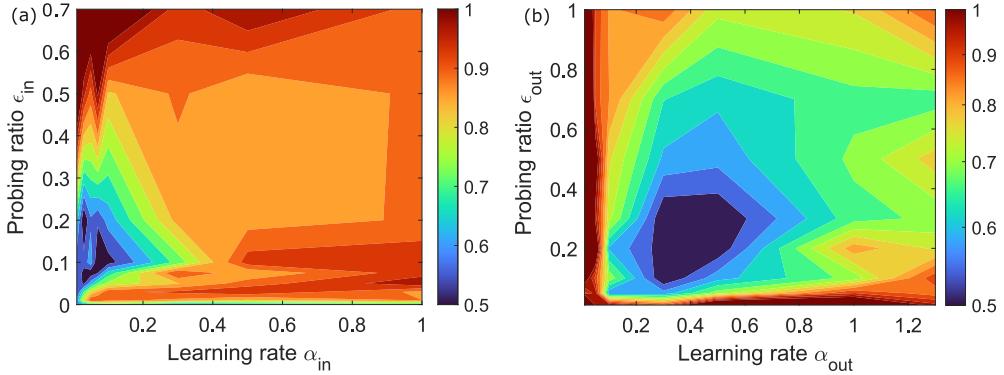


Fig. 24 Hyperparameter scan for the input and output weights using the SPSA algorithm. (a) Colormap of the MSE as a function of ϵ_{in} and α_{in} . (b) Colormap of the MSE as a function of ϵ_{out} and α_{out} .

Figure 24 shows the results of the hyperparameter scan for the input and output weights with the MSE as a colormap. The learning rate α is scanned on the x-axis and the perturbation ϵ on the y-axis for, both, input and output weights in Fig. 24(a) and (b), respectively. First, output weights are scanned with an output matrix size of $N_{\text{out}} = 2500$. Optimal values are $\epsilon_{\text{out}} \in [0.2, 0.3]$ and $\alpha_{\text{out}} \in [0.3, 0.4]$. These values are then fixed within the optimal range to $\alpha_{\text{out}} = 0.35$ and $\epsilon_{\text{out}} = 0.25$ and ϵ_{in} , α_{in} , are scanned in the same way, with $N_{\text{in}} = 400$. Crucially, we see that the optimal values for the input hyperparameters are significantly smaller than for the output, with optimal values $\epsilon_{\text{in}} \in [0.08, 0.1]$ and $\alpha_{\text{in}} \in [0.05, 0.1]$, highlighting the complexity of tuning the input weights, i.e. highly non-convex and ragged error landscape. Moreover, in our experience, these hyperparameters are quite robust for nearly all values of N_{in} and N_{out} , and the optimal values are stable even across different LA-VCSELs.

Finally, contrary to the FD method, SPSA is able to leverage a greater number of parameters and still converge, even for a fixed number of training epochs, making it a

simple yet powerful hardware-compatible optimization strategy. As illustrated in Fig. 25(b), SPSA is able to converge to a lower MSE than the FD method and performance increases with the number of parameters, and saturates at around $N_{\text{out}} = 2500$ whereas FD saturated at $N_{\text{out}} = 400$. The advantages of SPSA over FD become even more apparent when considering that, in terms of the number of measurements per epoch, SPSA is equivalent to FD with $N_{\text{probe}} = 1$.

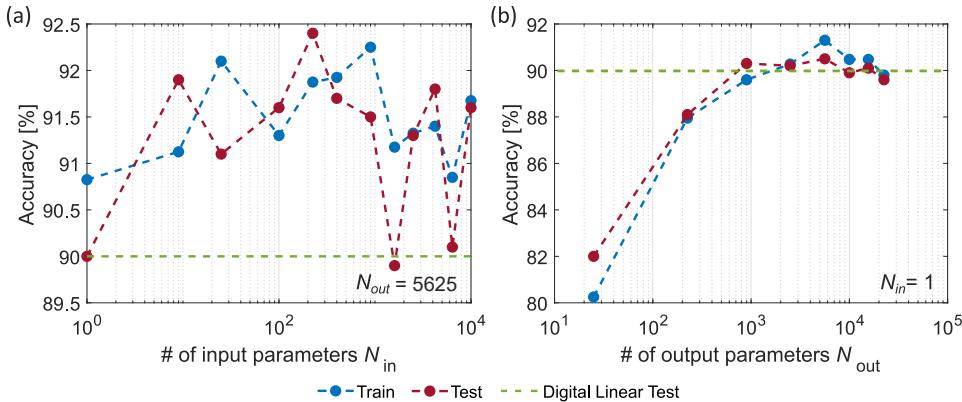


Fig. 25 Performance of the SPSA algorithm as a function of (a) input N_{in} and (b) output N_{out} parameters.

While SPSA can be a powerful algorithm, it is not without its limitations. When trying to study the influence of the number of input weights, no clear trend can be extracted as shown in Fig. 25(a). It seems that SPSA is not really able to leverage the additional degrees of freedom provided by the input weights.

6.1.3 Covariance Matrix Adaptation Evolution Strategy

For the CMA-ES algorithm, the standard deviation along each direction in the search space is learned through sampling. Therefore, we do not need to consider different hyperparameters for input and output weights. The main parameters influencing CMA-ES convergence are the initial standard deviation of the distribution σ_{init} and the population size p . In general, CMA-ES is appreciated for its robustness in terms of hyperparameter choice, and many theoretical estimates or rules of thumb for hyperparameter choice as a function of dimensionality exist in the literature [61].

Since CMA-ES is an advanced optimization algorithm we study the behavior of these two hyperparameters for the following output and input weight settings respectively; $N_{\text{out}} = 2500$ and $N_{\text{in}} = 2500$. Figure 26 shows the impact of σ_{init} on the performance of the CMA-ES algorithm. We see that the optimal value of σ_{init} is quite stable and is around $\sigma_{\text{init}} = 0.2$.

The last and perhaps most impactful parameter on performance is the population size. Performance increases with larger populations and then saturates, which is a

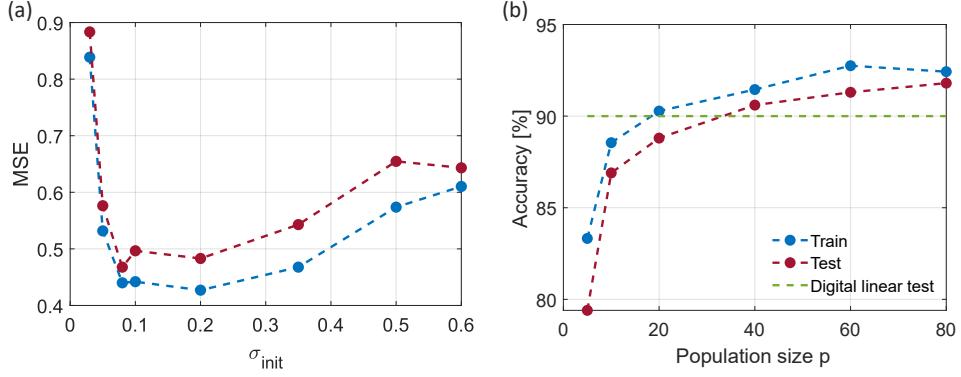


Fig. 26 (a) MSE as a function of the initial standard deviation σ_{init} . (b) Accuracy as a function of population size p .

general feature of population-based optimization algorithms. For our purposes, the saturation point is at around $p \in [40, 60]$.

Because of its quadratic scaling with the dimensionality of the optimization problem, tuning the few hyperparameters of CMA-ES can be quite time consuming. As such, we did not explicitly study the performance scaling of CMA-ES with respect to the number of parameters, but we can infer from the previous section that CMA-ES is able to leverage a large number of parameters. Furthermore, we provide in the following section a comparison of all relevant training strategies in terms of performance.

6.1.4 Parameter-exploring Policy Gradients

Much like CMA-ES, PEPG has the merit of being quite a robust strategy, with a noticeable advantage being that its computational overhead is much smaller than that of CMA-ES. At first glance, it may seem that tuning PEPG hyperparameters is a cumbersome task compared to CMA-ES. Yet in practice, the algorithm is robust and its hyperparameter behavior is quite predictable. Furthermore, its reduced computational overhead allows us to study a greater number of hyperparameters and dependencies in higher dimensional settings. As such we study all hyperparameters with $N_{\text{out}} = N_{\text{in}} = 2500$.

We start by studying the impact of the initial standard deviation of the normal distribution defining the population σ_{init} , as well as the learning rate which adapts the mean α_μ . We then study the impact of α_σ , the learning rate which adapts the standard deviation. As shown in Fig. 27(a) and (b), respectively. Optimal values are $\sigma_{\text{init}} \sim 0.2$ similar to CMA-ES, $\alpha_\mu \in [0.4, 0.5]$ and the choice of α_σ is not as critical as it shows a rather flat dependency with optimal values ranging between $\alpha_\sigma \in [0.2, 0.4]$.

We then studied the impact of population size, highlighting the same behavior measured for CMA-ES as shown in Fig. 28. For our purposes, the saturation point is at around $p = 40$, which corresponds to $\sim 1\%$ of the 5000 dimensional search space, confirming the same trend we measured on a digital convolutional NN on the MNIST dataset.

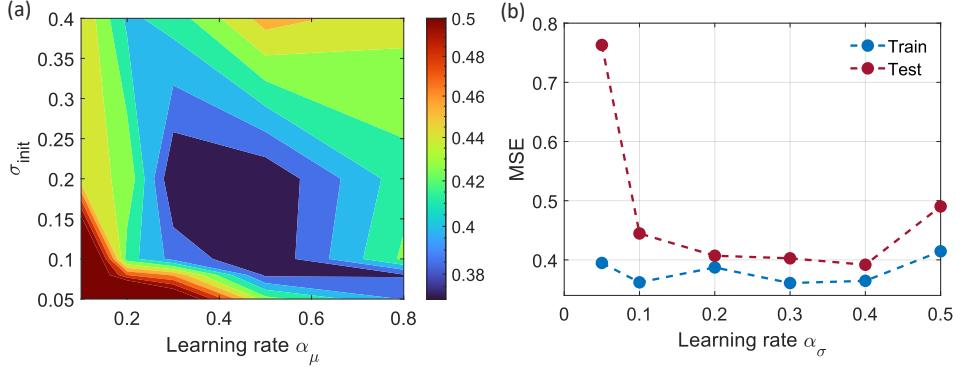


Fig. 27 Hyperparameter scan for the input and output weights using the PEPG algorithm. (a) Colormap of the MSE as a function of the initial standard deviation σ_{init} and mean learning rate α_μ . (b) MSE as a function of the standard deviation learning rate α_σ .

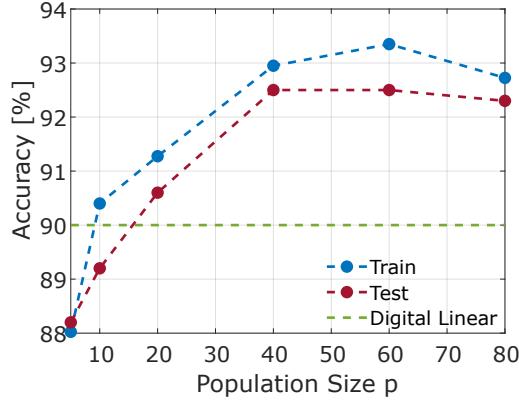


Fig. 28 Classification accuracy as a function of the population size p

Then, by fixing the population size at $p = 40$ to reduce measurement time, we study how the performance of PEPG scales with the number of input and output parameters respectively. As shown in Fig. 29, performance increases with the number of parameters and saturates at around $N_{\text{out}} = 5625$ and $N_{\text{in}} = 2500$, showcasing the powerful nature of the PEPG algorithm as opposed to the FD method which saturates at $N_{\text{out}} = 400$, or even SPSA which can leverage more output weights but struggles greatly when it comes to making use of the input weights or reaching similar performance levels.

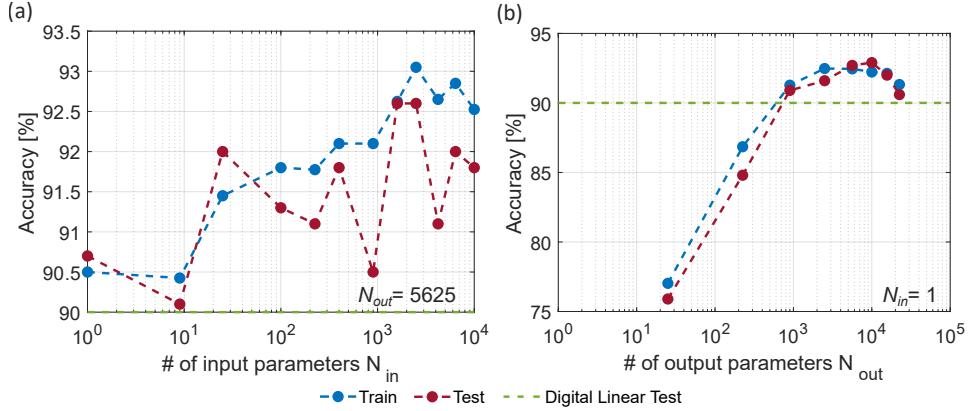


Fig. 29 Classification accuracy as a function of the number of (a) input N_{in} and (b) output N_{out} parameters.

7 Comparison of training algorithms

We can now compare these different optimization algorithms in terms of convergence efficiency. In this analysis, we specifically focus on SPSA, PEPG and CMA-ES since they show the most promise and can handle high dimensional optimization. Previously, we showed convergence in terms of epochs, which are equivalent to individual optimization steps. Crucially, different algorithms compute these update steps in vastly different ways, and as such, the energy cost of each update or the time spent per epoch is different. By studying convergence as a function of physical time for convergence, we begin to understand the real-world efficiency of each strategy in a hardware context. Indeed, SPSA only takes two measurements per epoch, while CMA-ES and PEPG require a full population evaluation. It would appear therefore that SPSA is more efficient in terms of convergence, but as we will see, the situation is not so trivial.

Indeed, while SPSA provides quicker updates, these updates are noisier and contain less information than updates computed via PEPG or CMA, making them less efficient in terms of convergence. And yet, one cannot simply neglect the computational overhead incurred by the CMA-ES algorithm, as this overhead scales quadratically with the number of parameters. We extensively compared these algorithms across a wide variety of devices of different shapes and sizes as shown in Fig. 30. As expected, both CMA-ES and PEPG reach better performance levels than SPSA. We confirmed PEPG to be the most efficient optimization algorithm as it is the most robust and provides the fastest convergence, being anywhere from ~ 3 to ~ 4 times faster than CMA-ES in this present test, and reaching significantly better performance than SPSA more efficiently. It should be stressed however that convergence efficiency is highly task dependent.

For reference, our performance evaluation was conducted under optimal injection locking conditions to ensure maximum performance as presented in [29], the free running and locked mode profile as well as spectra are shown in figure 31 for two selected LA-VCSELs.

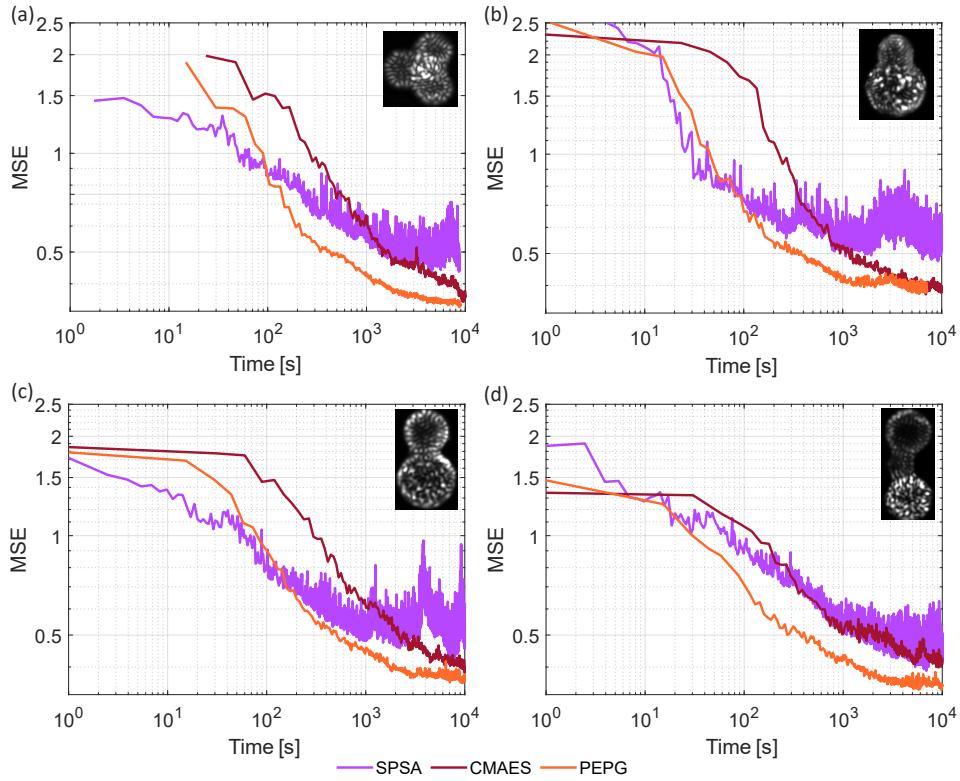


Fig. 30 Convergence efficiency for four different chaotic cavity LA-VCSELs trained using SPSA, PEPG and CMA-ES.

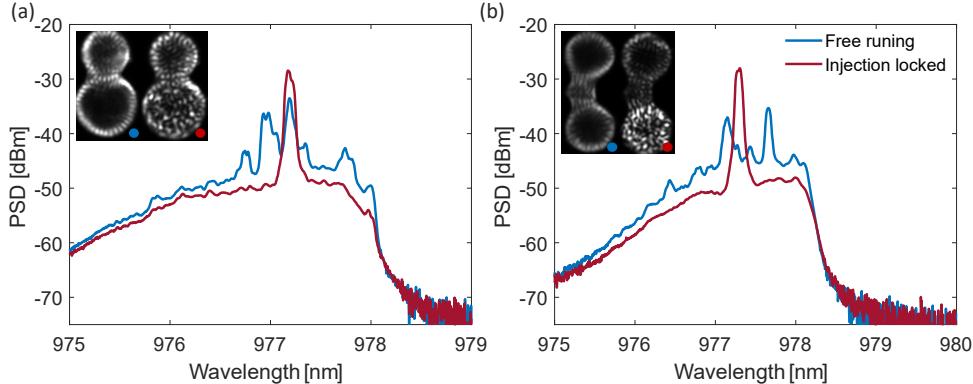


Fig. 31 Spectra / mode profiles of two different free running and injection locked chaotic cavity LA-VCSELs.

To illustrate how misleading a comparison in terms of epochs can be, we show the convergence of a selected LA-VCSEL, both, in terms of time and epochs in Fig. 32(a) and (b), respectively. First, if we only consider convergence in terms of epochs, it appears that SPSA is always significantly worse than the other two algorithms. However, if training time is restricted, then SPSA performs better than CMA-ES for this task. Moreover, the difference between PEPG and CMA-ES is greatly diminished when only considering epochs. Indeed, while PEPG takes \sim 140 epochs to reach an MSE of 0.4 CMA-ES takes \sim 200 epochs. This might seem like a small difference, yet if we look at the time spent to reach each value, PEPG takes \sim 1900 s while CMA-ES takes \sim 6240 s making PEPG \sim 3.3 times faster which cannot be deduced when only considering epochs.

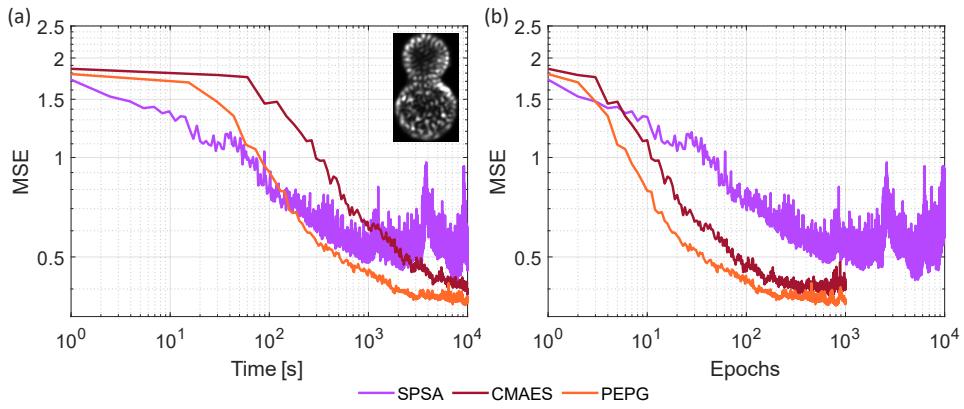


Fig. 32 Convergence of a selected LA-VCSEL in terms of time (a) and epochs (b) for the different optimization algorithms.

To further investigate the behavior and scaling of these different algorithms, Fig. 33(a) shows the time spent per epoch for each algorithm as a function of the number of parameters N , while (b) shows a detailed breakdown of the time spent at each step of the forward pass for our ONN.

In Fig. 33(a), the offset between the population based algorithms and SPSA stems from the larger number of measurements per epoch for PEPG and CMA-ES. Indeed, SPSA keeps a constant epoch time since it only requires two measurements per epoch. In contrast, in Fig. 33(a) the population size is either $p = 10$ for $N \leq 1000$, or scales with the dimensionality of the search space according to $p = \lceil 0.01 \times N \rceil$, when $N > 1000$. Crucially, there is a stark difference between PEPG and CMA-ES as the number of parameters increases due to the additional quadratic overhead of CMA-ES induced by the covariance matrix calculation. We were able to experimentally verify that PEPG scales with $O(N^{0.98})$ and CMA-ES with $O(N^{1.85})$. This stark difference reaches close to an order of magnitude for parameter numbers relevant to our experiment i.e. \sim 10000 parameters.

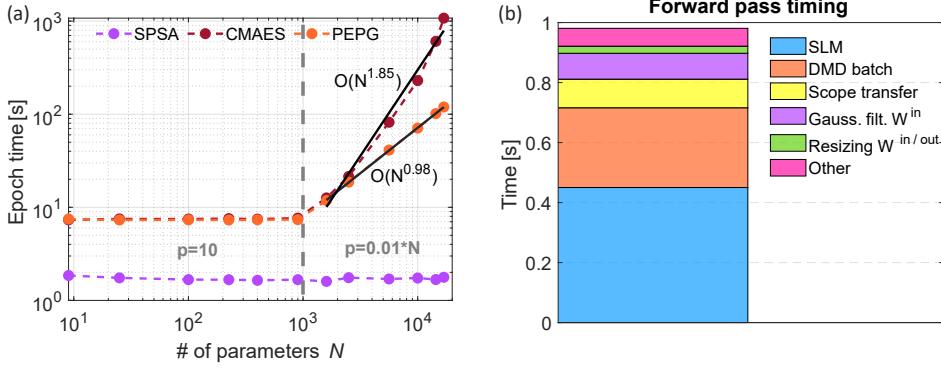


Fig. 33 (a) Time taken by each optimization epoch for different training algorithms in seconds. (b) Bar chart detailing the time taken by each step during the forward pass on the experimental setup.

To partially mitigate this heavy overhead, several variants of the original CMA-ES algorithm have been explored to achieve sub-quadratic scaling, and a detailed comparison for a wide range of synthetic benchmark tasks is provided in [85, 86]. However, it should be stressed however that no such comparison exists in the context of high dimensional NN optimization. This simple characterization is another way of confirming the results shown in Fig. 30. Even if CMA-ES is able to leverage a large number of parameters, its quadratic overhead can automatically disqualify it in the context of optimization under limited digital resources, for all but moderate dimensional physical systems. In addition, we are still comparing these algorithms using a conventional digital computer, Dell Precision i5-6400 8Gb RAM, which means that this difference will be exacerbated when considering limited digital resources such as a Raspberry Pi. Our comparison of different optimization algorithms and their corresponding overhead is crucial when designing autonomous hardware NN, where the reliance on an external digital computer needs to be minimized. We feel it remains a largely unexplored question in the context of unconventional hardware optimization and deserves more in-depth investigation.

Figure 33(b), showcases how the SLM we use is our main bottleneck in terms of speed followed by the DMD batch, referring to the total time it takes the DMD to show a batch of 4000 images in this present case. A fundamental limiting factor in our setup is that population samples have to be evaluated sequentially. In contrast, software NNs can benefit from hardware acceleration such as vectorization, and massive parallelization to perform evaluation of population samples in parallel. However, such parallelization only addresses the time-bottleneck, while it is inefficient and counter-productive in terms of energy efficiency [87, 88]. For the output weights, one could imagine creating copies of the LA-VCSEL and using a detector array to evaluate the population in parallel, yet this would result in a significant increase in experimental complexity and cost. Moreover, multiplexing the input weights in such a way is not possible. At first glance, it would seem that the only immediate avenue for addressing this problem is to increase the speed at which the weights are addressed, i.e. the speed

of the SLM, which is currently limited to ~ 3 Hz. Crucially, for high dimensional problems, using a faster SLM would only help with the convergence speed of PEPG and SPSA, the CMA-ES algorithm would be largely unaffected due to its computational overhead in a high dimensional optimization scenario.

For a given LA-VCSEL, we can study how performance scales with the number of input and output parameters for, both, SPSA and PEPG as shown in Fig. 34. Interestingly, for the output weights, both strategies can overcome the digital linear classifier limit, but PEPG is able to consistently reach better performance at high numbers of parameters and it also saturates later at $N_{\text{out}} = 5625$ as opposed to $N_{\text{out}} = 2500$, meaning that it can leverage these new dimensions whereas SPSA cannot. For the input weights, our measurement is very noisy and we would need to conduct a more comprehensive study averaging several times for each datapoint to get more reproducible results, which is currently out of scope due to the slow SLM.

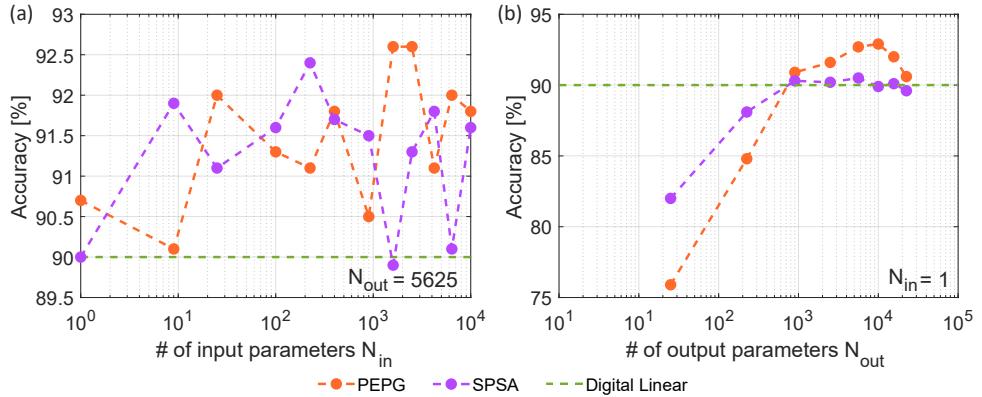


Fig. 34 Performance comparison between SPSA (purple) and PEPG (orange) as a function of the number of input (a) and output (b) parameters.

Finally, we can benchmark relevant strategies like PEPG and SPSA on all classes of the MNIST dataset in a sequential manner using one-vs-all classification, and compare them to the baseline of a digital linear classifier trained and tested directly on the same Boolean MNIST images that are sent to the LA-VCSEL. In one-vs-all classification, for n_{classes} classes, n_{classes} independent classifiers are trained from scratch, each distinguishing one class from the rest. The results from this comparison are shown in Fig. 35.

PEPG marks itself as the most efficient online learning algorithm and converges to a lower error faster than the SPSA algorithm while also providing more consistent behavior. Interestingly, only PEPG is able to overcome the digital linear classifier even if SPSA gets quite close with only a $\sim 0.6\%$ difference on average across all classes while PEPG overcomes the digital linear classifier limit by 2%. Table 5 shows the averaged classification accuracy for each class of the MNIST dataset for the linear classifier, PEPG and SPSA, as well as with the LA-VCSEL switched off, which constitutes the

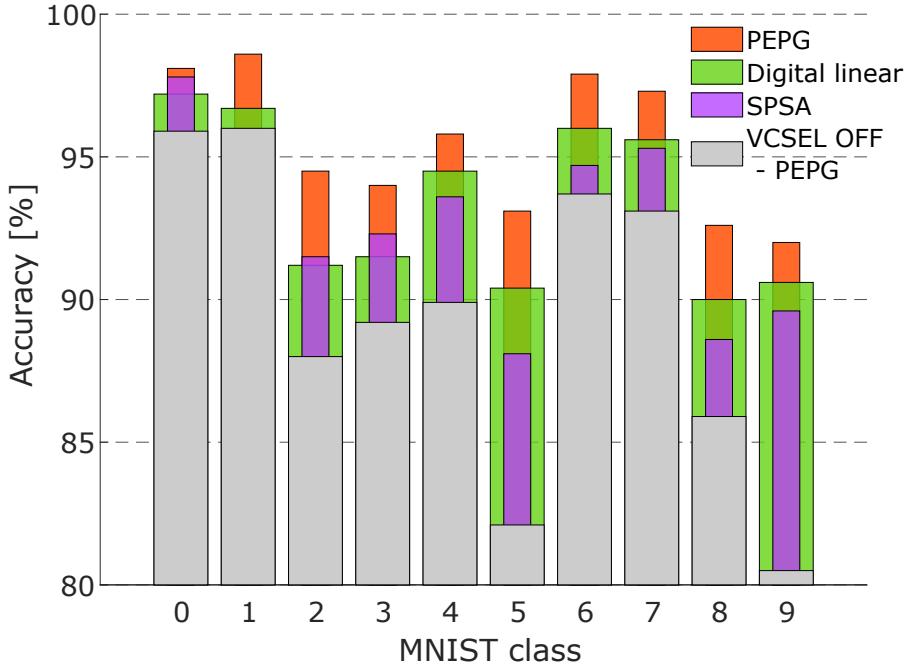


Fig. 35 One-vs-all test classification accuracy on the MNIST dataset, using PEPG (orange) and SPSA (purple) compared to a digital linear classifier (green). The VCSEL off baseline is shown for reference (gray).

baseline for a hardware linear system, comprising only the linear random mixing via the MMF. Crucially, when the LA-VCSEL is off, we specifically train the MMF system with PEPG, which is the most efficient and high-performance algorithm. Moreover, we use the same number of input and output parameters for, both, VCSEL ON and OFF. Therefore, the gain in performance when switching the LA-VCSEL on is solely due to its nonlinearity and high dimensional mapping. In principle, even with the LA-VCSEL switched off, there still is a nonlinearity present in the physical system. Indeed, our output detector by measuring the light's intensity $I^{\text{out}} = \sum_i W_i^{\text{out}} * |E_i^{\text{MMF}}|^2$, implements a nonlinear operation on the MMF field, yet as shown in our present measurement, this nonlinearity is not sufficient at all and cannot even result in performance close to a digital linear classifier.

Digital Linear Classifier	PEPG	SPSA	VCSEL OFF PEPG
93.37	95.39	92.71	89.43

Table 5 Averaged classification accuracy across all classes of the MNIST dataset for the linear classifier, PEPG and SPSA, as well as with the LA-VCSEL switched off.

These results might seem trivial or obvious given the LA-VCSEL’s highly nonlinear behavior. Yet, because the linear classifier is trained in software on a digital computer, it is completely exempt of noise, instabilities, drifts, and benefits from floating point precision digits. In addition, it is trained in one step with a ridge regression algorithm to prevent overfitting and help its generalization properties. For these reasons, we feel that the fact that a fully hardware trained ONN with no offline components can overcome a digital, albeit linear, classifier is a significant achievement.

8 Conclusion

In summary, our work presents a comprehensive exploration of model-free, hardware-compatible training algorithms implemented on a laser-based optical neural network (ONN). The investigation began with a detailed ceiling analysis performed on a representative software neural network, which revealed that the inclusion of both positive and negative weights, at least 4 to 5-bit weight resolution, and tunable input connectivity are fundamental for achieving high performance in complex tasks. In our simulations, a fully trainable feed-forward neural network achieved near 97.5% accuracy on the MNIST dataset, whereas fixed random mappings as found in extreme learning machines (ELM) resulted in a significantly lower performance 87%. These observations underscore the importance of trainable input weights and the critical role of weight precision, especially when processing spatially diverse and high-dimensional data.

Building on the insights from this ceiling analysis, we realized significant improvements to our ONN experimental setup. We designed both input and output weights with high resolution as well as positive and negative values, to unlock the full potential of our LA-VCSEL based NN concept. These hardware improvements allowed us to move beyond hardware RC and implement a fully trainable ONN. In addition, we implemented and extensively studied the hyperparameter behavior of different hardware compatible optimization algorithms, namely SPSA, PEPG, CMA-ES and FD, to train our ONN. We also compared the convergence efficiency of these strategies in the context of hardware optimization under limited digital resources for a wide range of LA-VCSELS, and found that PEPG is the most efficient and high-performance algorithm. In contrast, due to its computational overhead, CMA-ES is not well suited for high dimensional optimization problems in an autonomous hardware context. Finally, SPSA offers somewhat of a happy medium between two. We then studied the performance of PEPG and SPSA for all classes of the MNIST dataset and compared them to two baselines. The first being a digital linear classifier, while the second was our ONN with the LA-VCSEL switched off, which corresponds to a hardware linear system. We found that PEPG is able to overcome the digital linear classifier limit of 93.37%, while SPSA approaches it yet falls short. Crucially, both outperform the linear hardware system accuracy of 89.43% by a significant margin achieving 95.39% and 92.71% for PEPG and SPSA respectively, highlighting the high dimensional nature of the transformation produced by the LA-VCSEL. Notably, a significant portion of physical NNs in the literature still rely on offline, i.e. software weights on a digital computer. This approach, while understandable since it offers great flexibility, is not in

the spirit of building autonomous computing systems. Crucially, it artificially inflates the performance of physical NNs while tying them down, in most cases, to a slow digital computer handling the states of the system and applying software weights, which undermines the potential gains in terms of speed and energy efficiency that physical systems might otherwise offer. In our view, offline weights should be reserved for fundamental testing and troubleshooting, which is illustrated by the stark performance difference between the digital linear classifier and the VCSEL OFF.

In conclusion, our study not only provides a robust framework for training large-scale physical neural networks but also introduces experimental innovations that overcome longstanding challenges in the field. The combination of model-free optimization, high-resolution tunable weight control, and an advanced VCSEL-based platform culminates in an ONN that is both high-performing and energy-efficient. We hope that these findings will stimulate further advancements in photonic neural computing and foster interdisciplinary collaboration across physics, photonics, and machine learning, ultimately driving the next generation of unconventional computational hardware.

Acknowledgments

This work was supported by the Region Bourgogne Franche-Comté, the EUR EIPHI program (Contract No. ANR-17-EURE-0002), by the German Research Foundation (via Phase III of the Collaborative Research Center project 787), the European Union's and Research Council's Horizon research and innovation program under the Marie Skłodowska-Curie grant agreement No 860830 (POST DIGITAL) and the ERC Consolidator Grant (Grant agreement No. 101044777 (INSPIRE)) as well as JSPS KAKENHI (Grant No. JP22H05198) and JST CREST (Grant No. JPMJCR24R2)

Data availability

The data generated and/or analyzed during the current study are not publicly available for legal/ethical reasons but are available from the corresponding author on reasonable request.

Code availability

The code used in the simulation section is available in the following open Github repository : https://github.com/ASkalli/learning_strategies

Competing interests

Competing interests: We declare that none of the authors have competing financial or non-financial interests as defined by Nature Portfolio.

Author contributions

AS and DB conceived the experiment and in combination with SS and MG implemented the learning algorithms. MG and TC fabricated and characterized the

LA-VCSEL devices designed by JL with help from SR. AS carried out all experiments under the supervision of DB. AS wrote the manuscript with contributions of all authors.

References

- [1] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
- [2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. *Advances in neural information processing systems* **30** (2017)
- [3] ARD, V., GR, J.: Five ways deep learning has transformed image analysis. *Nature* **609** (2022)
- [4] Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L., Berriel, R., Paixao, T.M., Mutz, F., et al.: Self-driving cars: A survey. *Expert systems with applications* **165**, 113816 (2021)
- [5] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanzelotti, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**(6419), 1140–1144 (2018)
- [6] Reuther, A., Michaleas, P., Jones, M., Gadepally, V., Samsi, S., Kepner, J.: Survey of machine learning accelerators. In: 2020 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–12 (2020). IEEE
- [7] Hooker, S.: The hardware lottery. *Communications of the ACM* **64**(12), 58–65 (2021)
- [8] Jaeger, H.: Towards a generalized theory comprising digital, neuromorphic and unconventional computing. *Neuromorphic Computing and Engineering* **1**(1), 012002 (2021)
- [9] McMahon, P.L.: The physics of optical computing. *Nature Reviews Physics* **5**(12), 717–734 (2023)
- [10] Abreu, S., Boikov, I., Goldmann, M., Jonuzi, T., Lupo, A., Masaad, S., Nguyen, L., Picco, E., Pourcel, G., Skalli, A., et al.: A photonics perspective on computing with physical substrates. *Reviews in Physics*, 100093 (2024)
- [11] Dinc, N.U., Psaltis, D., Brunner, D.: Optical neural networks: The 3d connection. *Photoniques* (104), 34–38 (2020)

- [12] Rafayelyan, M., Dong, J., Tan, Y., Krzakala, F., Gigan, S.: Large-scale optical reservoir computing for spatiotemporal chaotic systems prediction. *Physical Review X* **10**(4), 041037 (2020)
- [13] Moughames, J., Porte, X., Thiel, M., Ulliac, G., Larger, L., Jacquot, M., Kadic, M., Brunner, D.: Three-dimensional waveguide interconnects for scalable integration of photonic neural networks. *Optica* **7**(6), 640–646 (2020)
- [14] Shen, Y., Harris, N.C., Skirlo, S., Prabhu, M., Baehr-Jones, T., Hochberg, M., Sun, X., Zhao, S., Larochelle, H., Englund, D., Soljacic, M.: Deep learning with coherent nanophotonic circuits. *Nature Photonics* **11**(7), 441–446 (2017)
- [15] Brunner, D., Soriano, M.C., Mirasso, C.R., Fischer, I.: Parallel photonic information processing at gigabyte per second data rates using transient states. *Nature communications* **4**(1), 1–7 (2013)
- [16] Chen, Z., Sludds, A., Davis III, R., Christen, I., Bernstein, L., Ateshian, L., Heuser, T., Heermeier, N., Lott, J.A., Reitzenstein, S., *et al.*: Deep learning with coherent vcsel neural networks. *Nature Photonics* **17**(8), 723–730 (2023)
- [17] Miller, D.A.: Attojoule optoelectronics for low-energy information processing and communications. *Journal of Lightwave Technology* **35**(3), 346–396 (2017)
- [18] Lupo, A., Picco, E., Zajnulina, M., Massar, S.: Deep photonic reservoir computer based on frequency multiplexing with fully analog connection between layers. *Optica* **10**(11), 1478–1485 (2023)
- [19] Feldmann, J., Youngblood, N., Karpov, M., Gehring, H., Li, X., Stappers, M., Le Gallo, M., Fu, X., Lukashchuk, A., Raja, A.S., *et al.*: Parallel convolutional processing using an integrated photonic tensor core. *Nature* **589**(7840), 52–58 (2021)
- [20] Wang, T., Sohoni, M.M., Wright, L.G., Stein, M.M., Ma, S.-Y., Onodera, T., Anderson, M.G., McMahon, P.L.: Image sensing with multilayer nonlinear optical neural networks. *Nature Photonics* **17**(5), 408–415 (2023)
- [21] Xia, F., Kim, K., Eliezer, Y., Shaughnessy, L., Gigan, S., Cao, H.: Deep learning with passive optical nonlinear mapping. *arXiv preprint arXiv:2307.08558* (2023)
- [22] Muller, M., Hofmann, W., Grundl, T., Horn, M., Wolf, P., Nagel, R.D., Ronneberg, E., Bohm, G., Bimberg, D., Amann, M.-C.: 1550-nm high-speed short-cavity vcsels. *IEEE Journal of selected topics in Quantum Electronics* **17**(5), 1158–1166 (2011)
- [23] Vatin, J., Rontani, D., Sciamanna, M.: Enhanced performance of a reservoir computer using polarization dynamics in vcsels. *Optics letters* **43**(18), 4497–4500 (2018)

- [24] Jaeger, H.: The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report **148**(34), 13 (2001)
- [25] Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation* **14**(11), 2531–2560 (2002)
- [26] Ortín, S., Soriano, M.C., Pesquera, L., Brunner, D., San-Martín, D., Fischer, I., Mirasso, C., Gutiérrez, J.: A unified framework for reservoir computing and extreme learning machines based on a single time-delayed neuron. *Scientific reports* **5**(1), 14945 (2015)
- [27] Appeltant, L., Soriano, M.C., Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I.: Information processing using a single dynamical node as complex system. *Nature communications* **2**(1), 1–6 (2011)
- [28] Marković, D., Leroux, N., Riou, M., Abreu Araujo, F., Torrejon, J., Querlioz, D., Fukushima, A., Yuasa, S., Trastoy, J., Bortolotti, P., et al.: Reservoir computing with the frequency, phase, and amplitude of spin-torque nano-oscillators. *Applied Physics Letters* **114**(1) (2019)
- [29] Skalli, A., Porte, X., Haghghi, N., Reitzenstein, S., Lott, J.A., Brunner, D.: Computational metrics and parameters of an injection-locked large area semiconductor laser for neural network computing. *Optical Materials Express* **12**(7), 2793–2804 (2022)
- [30] Porte, X., Skalli, A., Haghghi, N., Reitzenstein, S., Lott, J.A., Brunner, D.: A complete, parallel and autonomous photonic neural network in a semiconductor multimode laser. *Journal of Physics: Photonics* **3**(2), 024017 (2021)
- [31] Nakajima, K., Hauser, H., Kang, R., Guglielmino, E., Caldwell, D.G., Pfeifer, R.: A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm. *Frontiers in computational neuroscience* **7**, 91 (2013)
- [32] Marković, D., Grollier, J.: Quantum neuromorphic computing. *Applied physics letters* **117**(15) (2020)
- [33] Wright, L.G., Onodera, T., Stein, M.M., Wang, T., Schachter, D.T., Hu, Z., McMahon, P.L.: Deep physical neural networks trained with backpropagation. *Nature* **601**(7894), 549–555 (2022)
- [34] Momeni, A., Rahmani, B., Scellier, B., Wright, L.G., McMahon, P.L., Wanjura, C.C., Li, Y., Skalli, A., Berloff, N.G., Onodera, T., et al.: Training of physical neural networks. arXiv preprint arXiv:2406.03372 (2024)
- [35] Momeni, A., Rahmani, B., Malléjac, M., Del Hougne, P., Fleury, R.:

Backpropagation-free training of deep physical neural networks. *Science* **382**(6676), 1297–1303 (2023)

- [36] Nakajima, M., Inoue, K., Tanaka, K., Kuniyoshi, Y., Hashimoto, T., Nakajima, K.: Physical deep learning with biologically inspired training method: gradient-free approach for physical hardware. *Nature communications* **13**(1), 7847 (2022)
- [37] Xue, Z., Zhou, T., Xu, Z., Yu, S., Dai, Q., Fang, L.: Fully forward mode training for optical neural networks. *Nature* **632**(8024), 280–286 (2024)
- [38] Pai, S., Sun, Z., Hughes, T.W., Park, T., Bartlett, B., Williamson, I.A., Minkov, M., Milanizadeh, M., Abebe, N., Morichetti, F., *et al.*: Experimentally realized in situ backpropagation for deep learning in photonic neural networks. *Science* **380**(6643), 398–404 (2023)
- [39] Andreoli, L., Porte, X., Chrétien, S., Jacquot, M., Larger, L., Brunner, D.: Boolean learning under noise-perturbations in hardware neural networks. *Nanophotonics* **9**(13), 4139–4147 (2020)
- [40] McCaughan, A.N., Oripov, B.G., Ganesh, N., Nam, S.W., Dienstfrey, A., Buckley, S.M.: Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation. *APL Machine Learning* **1**(2) (2023)
- [41] Skalli, A., Goldmann, M., Haghghi, N., Reitzenstein, S., Lott, J.A., Brunner, D.: Annealing-inspired training of an optical neural network with ternary weights. *Communications Physics* **8**(1), 68 (2025)
- [42] Chen, R., Fang, Z., Perez, C., Miller, F., Kumari, K., Saxena, A., Zheng, J., Geiger, S.J., Goodson, K.E., Majumdar, A.: Non-volatile electrically programmable integrated photonics with a 5-bit operation. *Nature Communications* **14**(1), 3465 (2023)
- [43] Guo, Y.: A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:1808.04752 (2018)
- [44] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of machine learning research* **18**(187), 1–30 (2018)
- [45] Novac, P.-E., Boukli Hacene, G., Pegatoquet, A., Miramond, B., Gripon, V.: Quantization and deployment of deep neural networks on microcontrollers. *Sensors* **21**(9), 2984 (2021)
- [46] Hinton, G.: The forward-forward algorithm: Some preliminary investigations, 2022. URL <https://arxiv.org/abs/2212.13345> (2022)

- [47] Oguz, I., Ke, J., Weng, Q., Yang, F., Yildirim, M., Dinc, N.U., Hsieh, J.-L., Moser, C., Psaltis, D.: Forward-forward training of an optical neural network. *Optics Letters* **48**(20), 5249–5252 (2023)
- [48] Scellier, B., Bengio, Y.: Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience* **11**, 24 (2017)
- [49] Rastrigin, L.A.: Systems of extremal control. Nauka (1974)
- [50] Liang, J.J., Qu, B.Y., Suganthan, P.N.: Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore **635**(2) (2013)
- [51] Spall, J.C.: A stochastic approximation technique for generating maximum likelihood parameter estimates. In: 1987 American Control Conference, pp. 1161–1167 (1987). IEEE
- [52] Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control* **37**(3), 332–341 (1992)
- [53] Choy, M.C., Srinivasan, D., Cheu, R.L.: Simultaneous perturbation stochastic approximation based neural networks for online learning. In: Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749), pp. 1038–1044 (2004). IEEE
- [54] Maeda, Y., Hirano, H., Kanata, Y.: A learning rule of neural networks via simultaneous perturbation and its hardware implementation. *Neural Networks* **8**(2), 251–259 (1995)
- [55] Martinez, J.I.M., Nakano, K., Higuchi, K.: Parameter estimation in neural networks by improved version of simultaneous perturbation stochastic approximation algorithm. In: 2009 ICCAS-SICE, pp. 4567–4572 (2009). IEEE
- [56] Wouwer, A.V., Renotte, C., Remy, M.: On the use of simultaneous perturbation stochastic approximation for neural network training. In: Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251), vol. 1, pp. 388–392 (1999). IEEE
- [57] Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. *IEEE computational intelligence magazine* **1**(4), 28–39 (2006)
- [58] Yang, X.-S., Hossein Gandomi, A.: Bat algorithm: a novel approach for global engineering optimization. *Engineering computations* **29**(5), 464–483 (2012)

- [59] Ha, D.: A visual guide to evolution strategies. blog.otoro.net (2017)
- [60] Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* **9**(2), 159–195 (2001)
- [61] Hansen, N.: The cma evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772 (2016)
- [62] Ros, R., Hansen, N.: A simple modification in cma-es achieving linear time and space complexity. In: International Conference on Parallel Problem Solving from Nature, pp. 296–305 (2008). Springer
- [63] Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* **8**, 229–256 (1992)
- [64] Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. *Neural Networks* **23**(4), 551–559 (2010)
- [65] Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural evolution strategies. *The Journal of Machine Learning Research* **15**(1), 949–980 (2014)
- [66] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-international Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995). IEEE
- [67] Khan, K., Sahai, A.: A comparison of ba, ga, pso, bp and lm for training feed forward neural networks in e-learning context. *International Journal of Intelligent Systems and Applications* **4**(7), 23 (2012)
- [68] Rauf, H.T., Bangyal, W.H., Ahmad, J., Bangyal, S.A.: Training of artificial neural network using pso with novel initialization technique. In: 2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), pp. 1–8 (2018). IEEE
- [69] Wang, H., Yan, X.: Optimizing the echo state network with a binary particle swarm optimization algorithm. *Knowledge-Based Systems* **86**, 182–193 (2015)
- [70] Wang, B., Xue, B., Zhang, M.: Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020). IEEE
- [71] Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. *Advances in neural information processing systems* **31** (2018)
- [72] Fisher, R.A.: The use of multiple measurements in taxonomic problems. *Annals of eugenics* **7**(2), 179–188 (1936)

- [73] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. *Neural computation* **1**(4), 541–551 (1989)
- [74] Skalli, A., Robertson, J., Owen-Newns, D., Hejda, M., Porte, X., Reitzenstein, S., Hurtado, A., Brunner, D.: Photonic neuromorphic computing using vertical cavity semiconductor lasers. *Optical Materials Express* **12**(6), 2395–2414 (2022)
- [75] Chang-Hasnain, C., Orenstein, M., Von Lehmen, A., Florez, L., Harbison, J., Stoffel, N.: Transverse mode characteristics of vertical cavity surface-emitting lasers. *Applied physics letters* **57**(3), 218–220 (1990)
- [76] Degen, C., Fischer, I., Elsäßer, W.: Transverse modes in oxide confined vcsels: Influence of pump profile, spatial hole burning, and thermal effects. *Optics Express* **5**(3), 38–47 (1999)
- [77] Xiong, Y., Lo, Y.-H.: Current spreading and carrier diffusion in long-wavelength vertical-cavity surface-emitting lasers. *IEEE Photonics Technology Letters* **10**(9), 1202–1204 (1998)
- [78] Baveja, P.P., Kögel, B., Westbergh, P., Gustavsson, J.S., Haglund, Å., Maywar, D.N., Agrawal, G.P., Larsson, A.: Assessment of vcsel thermal rollover mechanisms from measurements and empirical modeling. *Optics express* **19**(16), 15490–15505 (2011)
- [79] Alkhazragi, O., Dong, M., Chen, L., Liang, D., Ng, T.K., Zhang, J., Bagci, H., Ooi, B.S.: Modifying the coherence of vertical-cavity surface-emitting lasers using chaotic cavities. *Optica* **10**(2), 191–199 (2023)
- [80] Alkhazragi, O., Dong, M., Chen, L., Kong, M., Melinte, G., Liang, D., Ng, T.K., Zhang, J., Bagci, H., Ooi, B.S.: Chaotic-cavity surface-emitting lasers for optical wireless communication and low-speckle illumination. *APL Photonics* **8**(8) (2023)
- [81] Song, Q., Fang, W., Liu, B., Ho, S.-T., Solomon, G.S., Cao, H.: Chaotic microcavity laser with high quality factor and unidirectional output. *Physical Review A* **80**(4), 041807 (2009)
- [82] Shinohara, S., Hentschel, M., Wiersig, J., Sasaki, T., Harayama, T.: Ray-wave correspondence in limaçon-shaped semiconductor microcavities. *Physical Review A* **80**(3), 031801 (2009)
- [83] Brejnak, A., Gebski, M., Sokol, A., Marciniaik, M., Wasiak, M., Muszalski, J., Lott, J., Fischer, I., Czyszanowski, T.: Boosting the output power of large-aperture lasers by breaking their circular symmetry. *Optica* **8**(9), 1167–1175 (2021)
- [84] Kim, K., Bittner, S., Jin, Y., Zeng, Y., Wang, Q.J., Cao, H.: Impact of cavity

geometry on microlaser dynamics. arXiv preprint arXiv:2304.01313 (2023)

- [85] Loshchilov, I.: A computationally efficient limited memory cma-es for large scale optimization. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 397–404 (2014)
- [86] Varelas, K., Auger, A., Brockhoff, D., Hansen, N., ElHara, O.A., Semet, Y., Kassab, R., Barbaresco, F.: A comparative study of large-scale variants of cma-es. In: Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part I 15, pp. 3–15 (2018). Springer
- [87] Gustafson, J.L.: Reevaluating amdahl’s law. Communications of the ACM **31**(5), 532–533 (1988)
- [88] Hill, M.D., Marty, M.R.: Amdahl’s law in the multicore era. Computer **41**(7), 33–38 (2008)