PABO: Pseudo Agent-Based Multi-Objective Bayesian Hyperparameter Optimization for Efficient Neural Accelerator Design

Maryam Parsa[1], Aayush Ankit[1], Amirkoushyar Ziabari[1,2], Kaushik Roy[1]
1. Purdue University
2. Oak Ridge National Laboratory
{mparsa,aankit,kaushik}@purdue.edu
ziabariak@ornl.gov

ABSTRACT

The ever increasing computational cost of Deep Neural Networks (DNN) and the demand for energy efficient hardware for DNN acceleration has made accuracy and hardware cost co-optimization for DNNs tremendously important, especially for edge devices. Owing to the large parameter space and cost of evaluating each parameter in the search space, manually tuning of DNN hyperparameters is impractical. Automatic joint DNN and hardware hyperparameter optimization is indispensable for such problems. Bayesian optimization-based approaches have shown promising results for hyperparameter optimization of DNNs. However, most of these techniques have been developed without considering the underlying hardware, thereby leading to inefficient designs. Further, the few works that perform joint optimization are not generalizable and mainly focus on CMOS-based architectures.

In this work, we present a novel pseudo agent-based multi-objective hyperparameter optimization (PABO) for maximizing the DNN performance while obtaining low hardware cost. Compared to the existing methods, our work poses a theoretically different approach for joint optimization of accuracy and hardware cost and focuses on memristive crossbar based accelerators. PABO uses a supervisor agent to establish connections between the posterior Gaussian distribution models of network accuracy and hardware cost requirements. The agent reduces the mathematical complexity of the co-optimization problem by removing unnecessary computations and updates of acquisition functions, thereby achieving significant speed-ups for the optimization procedure. PABO outputs a Pareto frontier that underscores the trade-offs between designing high-accuracy and hardware efficiency. Our results demonstrate a superior performance compared to the state-of-the-art methods both in terms of accuracy and computational speed (~100x speed up).

1 INTRODUCTION

Advances in computing engines and graphic processing units (GPUs) as well as massively produced data from smart devices, social media, and internet, create an immense opportunity for machine learning and in particular deep learning techniques to solve brain-inspired tasks such as recognition and classification. Deep neural networks (DNNs) are computationally expensive, and require substantial resources. Therefore, domain-specific energy-efficient accelerators built with CMOS [1–4], resistive crossbars [5–7] and spintronics [8] based technologies were proposed to boost the performance and speed of DNNs.

Performance of DNNs is highly dependent on the selection of hyperparameters (HPs). These include, but are not limited to, the number of hidden layers, kernel sizes, the choice of optimizer and non-linearity function. On the other hand, designing a neuromorphic hardware is reliant not only on a networks' HPs, but also on hardware specific parameters such as memory bandwidth, pipelining in CMOS technologies [9], and the number of bits, the number of crossbars and the crossbar sizes in memristive crossbar accelerators [7]. Over the past few years, there has been a growing interest in optimizing DNNs' HPs to obtain their optimum performance (in terms of accuracy, speed, etc). Most of these HP optimization techniques, however, have been developed with little or even no attention to the underlying hardware implementation.

Optimizing the network performance without considering the strong correlation between its HPs and the corresponding hardware specific parameters results in a sub-optimal and inefficient hardware architecture. In fact, designing an energy-efficient accelerator for implementing a high performance DNN (i.e. with minimum error) is a multi-objective optimization task with expensive black-box objective functions, and solution to such optimization problems is a Pareto frontier. The Pareto frontier is a set that consists of solutions in which no other is superior in optimizing both objective functions (minimizing the DNN's error and maximizing the hardware energy efficiency). In other words, each member of the Pareto set is not dominated by other members of the set, where the dominance is defined as: The vector $\vec{a}$ dominates vector $\vec{b}$ notated as $\vec{a} \succ \vec{b}$ or $\vec{b} \prec \vec{a}$, iff $\forall i; f_i(a) \leq f_i(b)$ where f_i is the i-th objective function [10]. Each point in the Pareto set represents an HP combination and signifies a trade-off between the two objective functions (minimizing the DNN's error and maximizing the hardware energy efficiency). Designer may select any HP combination from the Pareto frontier set according to the design requirements, DNN accuracy, and hardware energy efficiency.

Bayesian optimization is one of the most effective techniques to optimize HPs of a DNN [11]. This stochastic approach is an online optimization of an expensive objective function f, when f is unknown. Starting with a random selection of HPs from the search space (observation), a Gaussian process based posterior model is estimated. For each posterior model, an acquisition function is defined based on a trade-off between exploring the search space, and exploiting the current status. The optimum point of the acquisition function directs us to the next set of HPs to evaluate. The posterior model is updated with each new experiment until adding an extra observation does not improve the model [12].

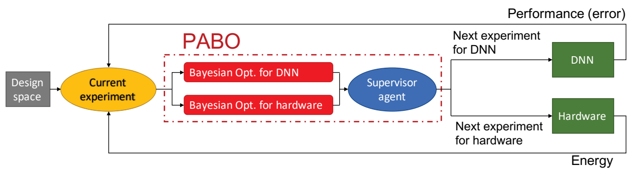

Figure 1: Overview of pseudo agent-based multi-objective Bayesian optimization (PABO) search process.

In this paper, we propose a pseudo agent-based multi-objective Bayesian optimization technique using Gaussian distribution (PABO). Figure 1 summarizes the PABO search process. The core of this approach is the PABO block which uses Bayesian optimization to estimate posterior models, and thus, find optimum HPs for the two black-box objective functions, related to DNN's performance and hardware's energy requirement. This process is then followed by a supervisor agent that evaluates the impacts of output HPs on the other posterior model and decides which HPs it must pass along. The PABO block decides on the set of HPs to evaluate at each step, the direction of the search process, and when to stop the technique. Using a supervisor agent in the PABO block reduces the complexity of the joint optimization problem, which in turn speeds up the algorithm to obtain the Pareto frontier compared to the state-of-the-art methods. In fact, in all the experiments performed, the output Pareto frontier is calculated with few iterations. Such capability is further beneficial for solving multi-objective problems with more than two objective functions.

We made the following contributions:

(1) A novel pseudo agent-based multi-objective Bayesian optimization technique that estimates the Pareto frontier with only few evaluations of network training and hardware energy calculation. Further, the proposed multi-objective black-box optimization approach can be applied to any number of black-box functions. For example, user can optimize neural network accuracy, hardware energy efficiency, and area requirements simultaneously.
(2) To the best of our knowledge, for the first time, a highly efficient memristive crossbar architecture [7] is selected to demonstrate the need of higher level optimization techniques for using neural networks. With memristive crossbar architectures, we demonstrate significant enhancement in hardware energy requirements when the network and hardware HPs are optimized jointly.
(3) We compare PABO with state-of-the-art methods and show that it estimates the Pareto frontier with superior computational speed.

2 RELATED WORK

In this section we provide a brief overview of the available multi-objective optimization techniques that have been used to co-optimize the neural network performance and hardware energy requirement.

The traditional approaches to solve an HP optimization problem, are grid search, random search and stochastic grid search [12–14]. The grid search technique is a manual search that estimate the optimum hyperparameter (HP) set by training a DNN for all possible combinations of HPs. This approach is only usable in small networks with few HPs. As its names indicates, the random search technique performs a random search over some domain of the exhaustive search space. Although this approach is more practical compared to the grid search, it is not necessarily an efficient approach since it might search spaces that are not promising [15]. Stochastic grid search [14] is another traditional approach that adds stochasticity to the manual search method to reduce the number of required network training for different combinations of HPs. This technique is also inefficient as it blindly searches the space.

Heuristic and sequential approaches such as simulated annealing (SA), genetic algorithm (GA), and Bayesian optimization techniques are more common to optimize a network HPs [16–19]. Among them GA and Bayesian approaches are favorable. A key limitation of the SA is that it might trap in a local minima for complex problems such as DNNs. GA technique speeds up the process of search for the next HP combination by isolating the evaluated HP sets with good network performance and build the next HP set based upon those good genes. Bayesian approaches are leveraging the stochasticity of Gaussian processes and conditional probability to efficiently limit the search space to HP sets that produce the best network performance.

More recently, for multi-objective optimization problems in which the objective functions are expensive to evaluate, design space exploration (DSE) [15, 20], non-dominated sorting genetic algorithm (NSGA-II) [21], evolutionary optimization [22], and Bayesian optimization techniques [9, 23–25] were developed. DSE [15] uses deep neural networks to find the optimum HP set for the best network performance while designing an energy-efficient hardware. However, designing such a DNN is application dependent and it cannot efficiently be generalized to other problems. NSGA-II algorithm is a modified GA based search technique for multi-objective optimization problems with expensive black-box functions. Evolutionary processes such as genetic crossover and mutation is utilized to guide the population toward the Pareto frontier [21]. We used this technique as a benchmark to compare with the PABO results. Both techniques estimate Pareto frontier, but the PABO is ∼100x faster.

A thorough review on Neural Architecture Search (NAS) for multi-objective optimization is given at [22]. Different approaches such as reinforcement learning based for NAS problems, evolutionary algorithm based, and search acceleration were among the techniques reviewed in [22] with the main focus on the most recent algorithms of MONAS [26] and DPP-Net [27]. The experimental results show that these approaches are able to find Pareto fronts with respect to different objectives imposed by devices and networks' performances. However, since these techniques are all based on evolutionary optimization, they require substantial computational resources. In PABO we use fundamentally different optimization technique (Bayesian-based) and able to find Pareto frontier set much faster than evolutionary based approaches.

Bayesian optimization based techniques were already used in the literature for optimizing HPs [9, 23, 24]. Reagen in [9] designed Spearmint package that outputs the Pareto frontier for a design space that minimizes energy consumption in Aladdin accelerator and optimizes the DeepNet performance. While promising results are demonstrated using this technique, the Spearmint package is not readily available to be used with the other networks and accelerator architectures. In addition, the method cannot be extended to more than two objective functions.

Inspired by the success of the Bayesian optimization techniques and to serve our purpose, which is optimizing HPs for a memristive crossbar accelerator [7], we designed a novel agent-based Bayesian optimization algorithm to find the optimum set of HPs that maximize the DNN performance and energy efficiency of the underlying hardware. The proposed hardware-aware optimization method is not limited to the number of functions to optimize (network performance, training speed, hardware area requirements or energy consumption) , and it is generic enough that can be used with different neural networks and accelerator architecture designs.

3 BACKGROUND

In this section, we provide some background on the Bayesian optimization and in particular Gaussian processes with acquisition functions.

Bayesian optimization (BO) is a derivative-free sequential optimization technique which is widely used when the objective function f is black-box and/or expensive to evaluate. One of the main variants of the BO is the Gaussian processes (GP) with acquisition function (AF). A GP is a stochastic process in which each combination of set of random variables follows a multivariate Gaussian distribution and can be defined by its mean and variance [12]. A key property they hold is that, the conditional distribution, and in turn the corresponding posterior distribution, of an expensive black-box function f at finite number sampling points will be a GP, if the prior distribution of it is assumed to be a GP [12].

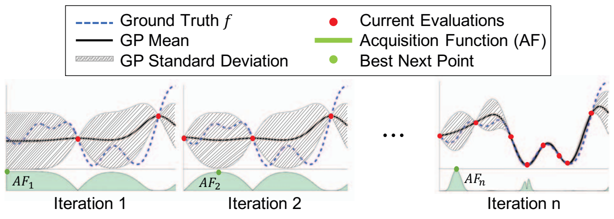

Figure 2: An overview of the Bayesian optimization using Gaussian Process (GP) with Acquisition Function (AF) method. The figure is slightly modified from its original in [28].

Figure 2 shows the procedure to estimate f using the GP with AF. Here, we would like to estimate the blue dashed curve that is the ground truth for the unknown function f. We assume that the observation y_i is the output of function f for the input θ_i. For all current observations of $D_n = \{(\theta_1, y_1), ..., (\theta_n, y_n)\}$, we implement BO to predict the best next point θ_{n+1} to evaluate f conditioned on D_n. This predictive conditional distribution is called posterior Gaussian model, and it is calculated by conditional probability of $p(y_{n+1}|\theta_{n+1}, D_n)$. In the Figure 2, mean and standard deviation of posterior Gaussian model is shown in solid black line and the shaded area, respectively. This model is then used to calculate AF, which directs the BO to the best next observation by exploiting and exploring the design space. Exploitation is seeking for places with the lowest mean (to stay near high quality observations), and exploration is to explore the search space where the variance is high (to cover the unknown regions and avoid trapping in local optima). AF is computed by the expected utility of evaluating function f, and it is a way to predict how useful it is to explore θ_{n+1} in order to optimize the black-box function f [9].

The acquisition function is shown in green in Figure 2. Optimum point of the AF shows the best next HP set to evaluate in the next iteration (θ_{n+1}). The posterior Gaussian distribution model is updated accordingly with the value of function f at θ_{n+1}. A new surrogate acquisition function is calculated for the updated posterior Gaussian model to obtain the next evaluation point. We observe that as we evaluate more points, the variance of the posterior Gaussian model is reduced, and the estimated mean gets closer to the ground truth function f that we would like to approximate. This procedure iteratively continues until the next point derived from optimizing the updated acquisition function cannot change to the estimated mean and variance of the posterior Gaussian model.

In PABO, we use Bayesian optimization processes along with a supervisor agent as the core processing unit to calculate the Pareto frontier of the multi-objective optimization of maximizing the DNN performance while maintaining the hardware energy requirements minimum. Details of our method is given in the following section.

4 METHODOLOGY

The overview of the proposed PABO method was given in Figure 1. In this section we explain this technique in more details. Algorithm 1 shows the PABO algorithm flow. We assume ***err*** (DNN's performance) and ***eng*** (hardware's energy requirement) are the two expensive black-box functions we would like to optimize.

The PABO process can be explained as follows. PABO has two key elements, the BO processes and the supervisor agent. In the n^{th} iteration, two BO processes are run. The first estimated BO with Gaussian processes computes an acquisition function to find the optimum HPs, θ_{n+1} set, for an unknown ***err*** function. The second estimated BO, on the other hand, calculates an acquisition function to find the optimum HPs, Γ_{n+1} set, for an unknown ***eng*** function. The sets D_{err} and D_{eng} keep track of θ_{n+1} and Γ_{n+1}, respectively, so that they can be used as training data in the BO processes of the next iteration. Before, running the next iteration, a supervisor agent is used to link these two seemingly separate processes. The supervisor agent evaluates sets $[err(\theta_{n+1}), eng(\theta_{n+1})]$, and $[err(\Gamma_{n+1}), eng(\Gamma_{n+1})]$, and examines whether these sets are dominated by all previous observations ($[err(\theta_n), eng(\theta_n)]_{n=1}^{n}$ or $[err(\Gamma_n), eng(\Gamma_n)]_{n=1}^{n}$) or not. The non-dominated sets may belong to the Pareto frontier of the problem and the corresponding HP set (θ_{n+1}, or Γ_{n+1}) will be added to D_{eng}, or D_{err}, respectively. Hence, the BO optimization processes of the next iteration has a more complete training set to estimate the acquisition functions and in turn optimized HPs in the direction of optimizing both objective functions. In what follows we explain the entire process in more details.

Algorithm 1: PABO algorithm flow

```
Inputs:      Black-box functions: $\boldsymbol{err}$, and $\boldsymbol{eng}$
             Initial training dataset: $\boldsymbol{\theta_i}|_{i=1}^{2} = \boldsymbol{\Gamma_i}|_{i=1}^{2}$ from design space $\mathcal{D}_N$ where $N$ is the number of all HPs
             Corresponding observations: $[err(\theta_1), eng(\Gamma_1)]$, and $[err(\theta_2), eng(\Gamma_2)]$

Initialize:  $n \leftarrow 1$;                            ▷ n is the number of evaluations
             $D_{err}$: Set for storing all selected HPs for estimating err
             $D_{eng}$: Set for storing all selected HPs for estimating eng

1:    Estimate posterior Gaussian distributions $\widetilde{err}_N = p(\widetilde{err}_N | err, D_{err})$, and $\widetilde{eng}_N = p(\widetilde{eng}_N | eng, D_{eng})$
2:    improve_flag ← TRUE
3:    while improve_flag do:
        • Calculate Acquisition Functions $AF_n(\widetilde{err})$, and $AF_n(\widetilde{eng})$
        • Determine $\theta_{n+1} = argmax_{\theta \in D_{err}}\ AF_n(\widetilde{err})$, and $\Gamma_{n+1} = argmax_{\Gamma \in D_{eng}} AF_n(\widetilde{eng})$
        • if $\theta_{n+1} = \theta_n$ or $\Gamma_{n+1} = \Gamma_n$ , then improve_flag ← FALSE
        • else:
      Add $\theta_{n+1}$ to $D_{err}$, and $\Gamma_{n+1}$ to $D_{eng}$
      Evaluate $[err(\theta_{n+1}), eng(\theta_{n+1})]$, and $[err(\Gamma_{n+1}), eng(\Gamma_{n+1})]$        ▷ Expensive step
      if $[err(\theta_{n+1}), eng(\theta_{n+1})]$ is not dominated by $[err, eng]_{n=1}^{n}$ observations, then add $\theta_{n+1}$ to $D_{eng}$
      if $[err(\Gamma_{n+1}), eng(\Gamma_{n+1})]$ is not dominated by $[err, eng]_{n=1}^{n}$ observations, then add $\Gamma_{n+1}$ to $D_{err}$
      $n \leftarrow n + 1$
      Update posterior Gaussian distributions $\widetilde{err}_N$, and $\widetilde{eng}_N$
```

Bayesian Optimization (BO) with Gaussian Processes (GP): In each iteration, BO finds the optimum HP sets that optimize the unknown ***err*** and ***eng*** objective functions, respectively. To start the BO process, we need to have at least two initial training dataset with their corresponding set of observations. $\theta_1 = \Gamma_1$ and $\theta_2 = \Gamma_2$ are randomly selected HP sets from design space D_N with observations of $[err(\theta_1), eng(\Gamma_1)]$, and $[err(\theta_2), eng(\Gamma_2)]$. For N possible HP sets, D_N is the exhaustive search space (grid search) that includes all HP combinations. D_{err}, and D_{eng} initially store the selected sets of HPs to optimize ***err*** and ***eng*** objective functions, respectively. They get updated after each iteration based on the supervisor agent's decision described later.

As mentioned in section 3, for each objective function, we estimate a posterior Gaussian model (shown with black line and shaded grey area in Figure 2). In the $n - th$ evaluation, these predictive conditional distributions are defined by $p(\widetilde{err}_N | err_n, D_{err})$ and $p(\widetilde{eng}_N | eng_n, D_{eng})$ for ***err*** and ***eng*** objective functions, respectively. The former is the conditional probability of the approximated error function, given the current estimate of error values for the input HP sets. Similarly, the latter is the conditional probability of the the approximated energy function, given the current estimate of energy values for the input HP sets. We calculate separate acquisition functions for each posterior Gaussian model. Details of defining this function is given in section 3. The optimum points of the acquisition functions, are the best selected HP sets to evaluate in the next iteration (θ_{n+1}, and Γ_{n+1}).

Supervisor Agent: Before running the next iteration, the next HP sets (θ_{n+1}, and Γ_{n+1}) are input to a supervisor agent. θ_{n+1} is the new HP combination chosen by the BO algorithm for estimating ***err*** function and will be added to D_{err} set. Similarly, Γ_{n+1}, which is chosen by the BO algorithm for estimating ***eng*** function, will be added to D_{eng} set.

In addition, Γ_{n+1} (the next selected HP set for optimizing the ***eng*** function) will also be added to D_{err} set if and only if the set $[err(\Gamma_{n+1}), eng(\Gamma_{n+1})]$ is not dominated by any other previous observations and might belong to the Pareto frontier set. This means that although Γ_{n+1} is chosen by posterior Gaussian model and its corresponding AF for ***eng*** function, it is also creating a solution set of $[err(\Gamma_{n+1}), eng(\Gamma_{n+1})]$ that up to this point belongs to the Pareto set. This set might be substituted in the future evaluations if the new set creates superior solutions in at least one of the objective functions. Moreover, θ_{n+1} will also be added to D_{eng} set if and only if the set $[err(\theta_{n+1}), eng(\theta_{n+1})]$ is not dominated by any other previous observation and might belong to the Pareto frontier set. We repeat this process until the acquisition function vanishes.

Throughout the process, the supervisor agent guides the search process to the Pareto frontier region and speeds up the procedure without adding extra complexity to the underlying mathematics.

5 BASELINE ACCELERATOR

In this work, the underlying hardware of choice is a neural accelerator. In the following, we overview the hardware architecture, and explain the energy consumption computation for a memristive crossbar accelerator.

5.1 Accelerator Overview

This section provides an overview of the memristive crossbar based accelerator used in our evaluations. Figure 3 shows a memristive crossbar based DNN accelerator. Typical memristive accelerators employ a spatial architecture, where the DNN is executed by mapping the model across the on-chip crossbar storage in a spatial manner [7]. This is because the memristive devices have high storage density, but are limited by the high write cost. Consequently, the high storage density enables mapping DNNs spatially in practical die sizes while alleviating the high write cost which would be required if a crossbar was reused for different parts of the model in a time-multiplexed fashion. At the lowest level, N Matrix Vector Multiplication Units (MVMU) are grouped into a single core. Each MVMU is composed of multiple crossbars and performs a 16-bit 128×128 matrix-vector multiplication. Note that multiple crossbars are needed to store high precision data required for DNN inference, since typical memristive crossbars store low-precision data such as 2-bits [5, 7]. At the next level, M cores are grouped into a single tile with access to a shared memory, which enables data movement between cores (inter and intra tile). At the highest level, T tiles are connected via a network-on-chip that enables data movement between tiles within a single node. For large scale applications, multiple nodes can be connected using suitable chip-to-chip interconnect.

5.2 Energy consumption

We use an abstract energy consumption model to evaluate the efficiency for PABO, where we consider the energy consumption of the MVMUs only. First, the abstract model enables evaluating the impact of hyperparameter optimization while isolating the benefits obtained from microarchitectural techniques. This isolation enables widespread applicability where DNNs optimized with PABO can be executed over a wide range of memristive accelerators, where each accelerator may be leveraging different dataflow, compute to control granularity etc. Second, while a typical memristive accelerator expends significant energy in shared memory, network on chip and chip-chip interconnect due to the data movements in a spatial architecture, reducing the number of MVMU operations typically reduces the total energy consumption commensurately [29].

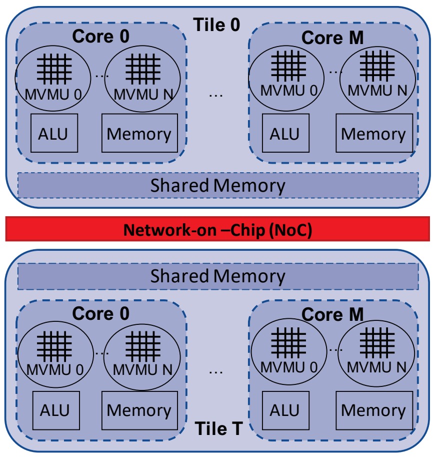

Figure 3: High-level Overview of a Abstract Hybrid Accelerator Architecture

A layer (fully connected layer or convolution layer) is partitioned into smaller blocks of size N×N to fit a MVMU (sized N×N). Each layer will map across multiple MVMUs that may span multiple cores and multiple tiles (see Figure 3). Further, a MVMU may be used multiple times (once) for an input in a convolution layer (fully connected layer) due to weight-sharing. Hence, the number of MVMU operations required to execute an inference of deep neural network will depend on the several HPs such as the number of layers, the number of extracted feature in each convolution layers, and the kernel sizes in the network architecture (Equations 1, and 2).

$$num\_xbar\_c_i = d_i \times d_i \times \lceil \frac{nc_i \times k_i \times k_i}{xs} \rceil \times \lceil \frac{nc_{i+1}}{xs} \rceil \tag{1}$$

$$num\_xbar\_f_i = \lceil \frac{nf_i}{xs} \rceil \times \lceil \frac{nf_{i+1}}{xs} \rceil \tag{2}$$

In these equations, $num\_xbar\_c_i$, and $num\_xbar\_f_i$ are number of crossbars for the i_{th} convolution layer and the fully connected layers, respectively. d_i is the dimension of the output, nc_i is the number of input features for convolution layer i. Similarly, nf_i is the number of input features for the fully connected layer i. k_i is the kernel size in i_{th} convolution layer, and xs is the crossbar size. The term d_i in equation 1 is for inherent weight-sharing property of convolution layers.

Typically, each memristive operation is followed by a vector linear, vector non-linear and data movement operations [7]. Consequently, the number of MVMU operations is proportional to the overall energy consumption and can be used as a metric of computational cost on hardware. We calculate the total energy consumption in each convolution and fully connected layer based on the number of crossbar operations using the following equations. In our selected memristive crossbar accelerator, a 16-bit (inputs and weights) crossbar operation (size 128×128) consumes ≃44nJ energy. *epx* is the energy per matrix vector multiplication operation. The sum of energy consumption for all the convolution and fully connected layers is then used to calculate the total energy consumption of the memristive crossbar accelerator (Equation 3).

Table 1: Experimental Parameters for AlexNet and VGG19 Case Studies

Hyperparameter	Case Study 1	Case Study 2	Case Study 3	Type
Architecture	AlexNet	AlexNet	VGG19	-
Dropout	0.4, 0.5	5e-3, ..., 5e-1	-	DNN
Learning rate	0.001	1e-6, ..., 1e-1	0.01, 0.1	DNN
Dropout, layer 1	-	-	0.3, 0.4	DNN
Learning rate decay	-	-	1e-6, 1e-4	DNN
Weight decay	-	-	0.05, 0.0005	DNN
Momentum	0.85, 0.9, 0.95	7e-3, .., 7e-1	-	DNN
# of FC layers	2, 3	2, 3	-	DNN/HW
# of Conv. layers	4, 5	4, 5	-	DNN/HW
Kernel size, layer 1	7, 5	3, 5, 7, 11	-	DNN/HW
Kernel size, layer 2	3, 5	3, 5	-	DNN/HW
Kernel Size, layer 3	3, 5	3, 5	-	DNN/HW
Kernel Size, layer 4	3	3, 5	-	DNN/HW
Kernel size, layer 6	-	-	3, 5	DNN/HW
Kernel size, layer 7	-	-	3, 5	DNN/HW
Kernel Size, layer 8	-	-	3, 5	DNN/HW
Kernel Size, layer 9	-	-	3, 5, 7	DNN/HW
# of features, layer 1	-	-	64, 128	DNN/HW
# of features, layer 2	-	-	128, 256	DNN/HW
# of features, layer 4	-	-	256, 512	DNN/HW
Search Space Size	192	6912	3072	-

$$tot\_eng\_t = (\sum_i num\_xbar\_c_i + \sum_i num\_xbar\_f_i) \times epx \tag{3}$$

For each set of HPs given in Table 1 case studies, we used Equations 1 through 3 to calculate the total hardware energy consumption.

6 RESULTS

To validate the proposed PABO algorithm, we used two different neural network architectures, AlexNet [30], and VGG19 [31], with flower17 [32] and cifar-10 [33] dataset, respectively. As the hardware of choice we selected a memristive crossbar accelerator [7]. A summary of the HPs that were selected for different case studies is given in Table 1. These choices were only made as the proof of concept for PABO technique. PABO search algorithm is not limited to these choices and depending on designer's decision, any type of neural network architecture, hardware, dataset, number and type of HPs can be selected.

In this section after discussing the limitations of single objective optimization, we present and analyze joint optimization results for different case studies in Table 1.

6.1 Single-Objective Optimization

Before presenting the joint optimization results, it is imperative to answer the question: Why one cannot rely on single objective optimization to minimize the hardware energy consumption with maximum neural network accuracy? Figure 4 demonstrates the limitations of using independent single objective HP optimization techniques to separately design a neural network with optimum performance and a hardware with minimum energy requirements.

Figure 4a shows that designing an energy efficient hardware without optimizing the network accuracy leads to significant decrease in the network performance (large error). The selected HP set is reported after 40 evaluations of hardware energy consumption. For this HP, the minimum energy is ~7.8mJ, while the DNN's

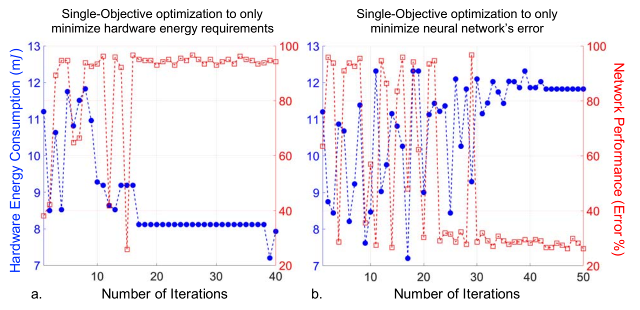

Figure 4: Single-objective optimization results for Table 1, case study 2 with SKOPT python Bayesian optimization package. a. Optimizing HPs for hardware energy consumption only. b. Optimizing HPs for DNN's performance only.

error is ~92%. Similarly, in Figure 4b the network performance -in terms of reducing error- is optimized without considering the energy consumption of the underlying hardware. The inefficient hardware design is evident as the reported minimum error region occurs at high hardware energy consumption. Both of these results are undesirable, and are the main reasons to seek a multi-objective approach to find HPs that minimizes DNN's error while designing an energy-efficient hardware. We used SKOPT python package to solve these single-objective Bayesian optimization for AlexNet on Flower17 dataset with HPs given in Table 1, case study 2.

6.2 Multi-Objective Optimization (PABO)

We used the proposed PABO algorithm to find the optimum DNN accuracy while minimizing the underlying memristive crossbar accelerator energy consumption on three case study. In case study 1, we used AlexNet network with the Flower17 dataset with a small search space of 192 HPs. The range of HPs are provided in Table 1. In this case, we intentionally selected a small search space, so that we can estimate the actual Pareto frontier using the grid search method. Figure 5 shows the results for case study 1. In this Figure, PABO's result compared with grid search, random search, and state-of-the-art NSGA-II (Non-Dominated Sorting Genetic Algorithm) [21]. Red triangles, blue dots, black squares and gray crosses correspond to PABO, random search, NSGA-II, and grid search, respectively. Each point in the figure corresponds to one evaluation of the noted techniques.

With only 17 evaluations (out of 192 possible sets of HPs), PABO estimates the Pareto frontier (red dash line in Figure 5) for the HPs within ~1-2% percent of the actual Pareto set (gray line in Figure 5) obtained using the grid search method. Compared to the NSGA-II approach, PABO not only estimates Pareto frontier more accurately, but also it is 92x faster. A comparison between the execution time for different techniques is shown in Figure 6.

In Table 2, to further illustrate the impact of HPs, we summarized them for the points A, B, C and D that are shown in Figure 5. Point A belongs to the Pareto frontier of the network at which we obtained the optimum DNN performance and hardware energy requirement. Point B corresponds to an HP set with minimized energy requirement for hardware, while producing a sub-optimal DNN design. At point C, HPs result in minimum DNN error but

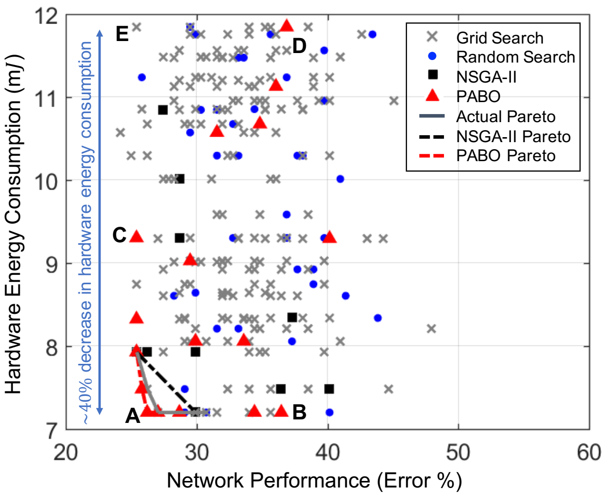

Figure 5: Case study 1: AlexNet on Flower17 dataset with 192 possible set of HPs. Comparison between grid search for all HP combinations (grey cross), random search with evaluating 40 different sets of HPs (blue dots), NSGA-II with population size of 10 and maximum generation of 50 (black squares), and PABO (red triangles). The red dashed line, gray line and the black dashed line are the Pareto frontiers obtained by PABO, grid search and NSGA-II approaches.

with an inefficient hardware design, and the corresponding HP at point D neither optimizes the DNN performance nor hardware energy consumption. It is clear from Table 2, that using a joint optimization approach is indispensable for optimal design of both the DNN and the hardware. Moreover, in this case study, selecting HPs given in point A (from Table 2) results in up to 40% decrease in energy requirements for the memristive crossbar accelerator compared to the case where DNN is not optimized for the hardware architecture design (point E shown in Figure 5).

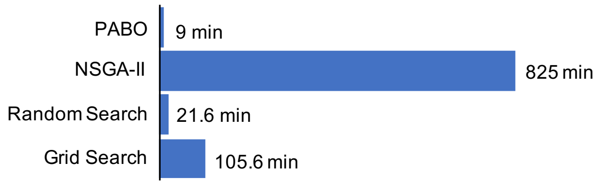

Figure 6: Execution time for case study 1. PABO is 92x faster than state-of-the-art NSGA-II technique

Figures 7 and 8 show the results for two additional case studies with more realistic choices of HPs. Details of the HP selections are given in Table 1. Figure 7 is a case study with 6912 choices of HP combinations on AlexNet architecture with Flower17 dataset. PABO results are shown in red triangles and compared with random search with blue dots and NSGA-II with black squares. Random search is performed with 40 evaluations, NSGA-II had population size of 20 with maximum generation of 100. PABO approximates the Pareto frontier with only 33 function evaluations (Training the DNN and computing the hardware energy consumption). Compare

Table 2: Hyperparameter Analysis for case study 1

Hyperparameter	A	B	C	D
Dropout	0.5	0.4	0.5	0.4
Learning rate	0.001	0.001	0.001	0.001
Momentum	0.95	0.85	0.95	0.9
Batch size	64	64	64	64
# of FC layers	2	2	2	3
# of conv. layers	4	4	5	5
Kernel size, layer 1	5	5	7	7
Kernel size, layer 2	3	3	3	5
Kernel size, layer 3	3	3	5	5
Kernel size, layer 4	3	3	3	3

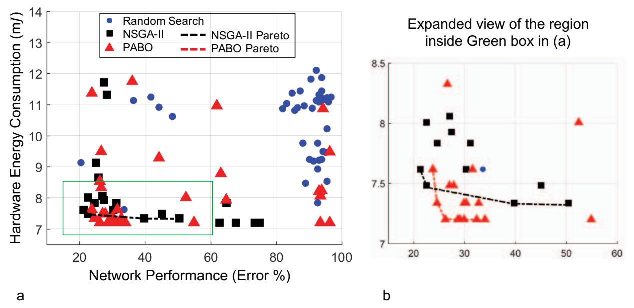

Figure 7: a. Case study 2: Comparison between PABO, NSGA-II, Grid Search and Random Search for AlexNet netork on Flower17 dataset with 6912 different set of HP combination. b. The expanded view of the region inside the green box in panel (a).

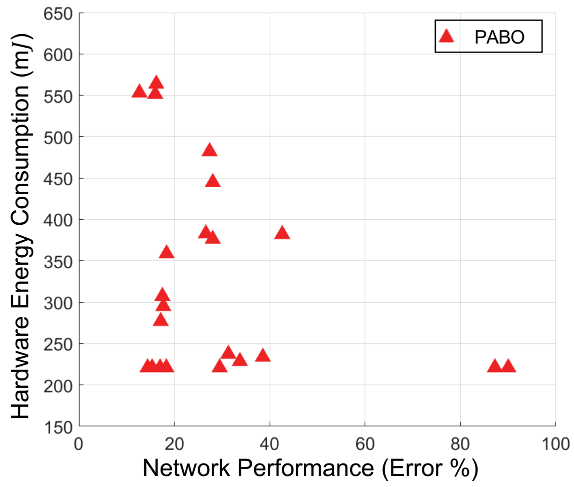

Figure 8: Case study 3: PABO result for VGG19 network on CIFAR-10 dataset with 3702 different set of HP combination. Other methods cannot be run on this case study due to significant computational requirements.

to 6000 function evaluations of NSGA-II technique, this leads to 183× faster execution time. Note that the results obtained for NSGA-II are the best we could get with the same computational resources we used to perform PABO and other methods. We may improve the NSGA-II's results using supercomputers and significantly longer hours of computation.

In Figure 8, we used VGG19 network on CIFAR-10 data with 3702 HPs combinations listed in Table 1. Since the network is significantly larger than AlexNet, it is computationally prohibiting to perform NSGA-II using our computational resources. On the other hand, PABO was able to estimate the Pareto frontier with only 22 evaluations.

7 CONCLUSION

We proposed a novel pseudo agent-based multi-objective hyperparameter optimization technique, deemed PABO, that can maximize the neural network performance while minimizing the energy requirements of the underlying hardware. PABO uses Bayesian optimization with Gaussian processes and acquisition function along with a supervisor agent, to estimate the Pareto frontier that shows the optimum HP sets for maximum neural network performance and minimum hardware energy consumption.

We tested PABO on both AlexNet and VGG19 neural network while designing a memristive crossbar accelerator, and compared it with other algorithms. Superior performance of our method both in terms of accuracy and computational time was demonstrated. 100x increase in computational speed compared to the NSGA-II algorithm was demonstrated. It is important to note that PABO is not limited to a specific neural network or hardware architecture, and can be applied to multiple (more than two) black-box functions corresponding to different design requirements.

REFERENCES

[1] Yu-Hsin Chen and et al., "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*. IEEE Press, 2016, vol. 44, pp. 367–379.
[2] Norman P. Jouppi and et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2017, ISCA'17, pp. 1–12, ACM.
[3] Swagath Venkataramani and et al., "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *ACM SIGARCH Computer Architecture News*. ACM, 2017, vol. 45, pp. 13–26.
[4] Jeremy Fowers and et al., "A configurable cloud-scale dnn processor for real-time ai," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.
[5] Ali Shafiee and et al., "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.
[6] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proceedings of*

the 43rd International Symposium on Computer Architecture, Piscataway, NJ, USA, 2016, ISCA'16, pp. 27–39, IEEE Press.

[7] Aayush Ankit and et al., "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[8] Shankar Ganesh Ramasubramanian and et al., "Spindle: Spintronic deep learning engine for large-scale neuromorphic computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 15–20.

[9] Brandon Reagen and et al., "A case for efficient accelerator design space exploration via bayesian optimization," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 2017, pp. 1–6.

[10] Tatsuya Okabe, Yaochu Jin, and Bernhard Sendhoff, "A critical survey of performance indices for multi-objective optimisation," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*. IEEE, 2003, vol. 2, pp. 878–885.

[11] Ilya Loshchilov and Frank Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[12] James S Bergstra and et al., "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[13] Hugo Larochelle and et al., "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 473–480.

[14] James Bergstra and et al., "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[15] Sean C Smithson and et al., "Neural networks designing neural networks: multi-objective hyper-parameter optimization," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 104.

[16] Jasper Snoek and et al., "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[17] José Miguel Hernández-Lobato and et al., "A general framework for constrained bayesian optimization using information-based search," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 5549–5601, 2016.

[18] José Miguel Hernández-Lobato and et al., "Predictive entropy search for bayesian optimization with unknown constraints," 2015.

[19] José Miguel Hernández-Lobato and et al., "Predictive entropy search for efficient global optimization of black-box functions," in *Advances in neural information processing systems*, 2014, pp. 918–926.

[20] Tobias Domhan and et al., "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.," in *IJCAI*, 2015, vol. 15, pp. 3460–8.

[21] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[22] Guillaume Michel, Mohammed Amine Alaoui, Alice Lebois, Amal Feriani, and Mehdi Felhi, "Dvolver: Efficient pareto-optimal neural network architecture search," *arXiv preprint arXiv:1902.01654*, 2019.

[23] Eric Bradford and et al., "Efficient multiobjective optimization employing gaussian processes, spectral sampling and a genetic algorithm," *Journal of Global Optimization*, vol. 71, no. 2, pp. 407–438, 2018.

[24] Daniel Hernández-Lobato and et al., "Predictive entropy search for multi-objective bayesian optimization," in *International Conference on Machine Learning*, 2016, pp. 1492–1501.

[25] José Miguel Hernández-Lobato and et al., "Designing neural network hardware accelerators with decoupled objective evaluations," in *NIPS workshop on Bayesian Optimization*, 2016, p. l0.

[26] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[27] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun, "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 517–531.

[28] Ryan P. Adams, "A Tutorial on Bayesian Optimization," 2014.

[29] Aayush Ankit, Abhronil Sengupta, and Kaushik Roy, "Transsformer: Neural network transformation for memristive crossbar-based neuromorphic system design," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 533–540.

[30] Alex Krizhevsky and et al., "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[31] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] M-E Nilsback and Andrew Zisserman, "A visual vocabulary for flower classification," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. IEEE, 2006, vol. 2, pp. 1447–1454.

[33] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.