A Case for Efficient Accelerator Design Space Exploration via Bayesian Optimization

Brandon Reagen[1], José Miguel Hernández-Lobato[2], Robert Adolf[1],
Michael Gelbart[3], Paul Whatmough[4,1], Gu-Yeon Wei[1], David Brooks[1]

[1]Harvard University [2]University of Cambridge [3]University of British Columbia [4]ARM Research

Abstract—**In this paper we propose using machine learning to improve the design of deep neural network hardware accelerators. We show how to adapt multi-objective Bayesian optimization to overcome a challenging design problem: optimizing deep neural network hardware accelerators for both accuracy and energy efficiency. DNN accelerators exhibit all aspects of a challenging optimization space: the landscape is rough, evaluating designs is expensive, the objectives compete with each other, and both design spaces (algorithmic and microarchitectural) are unwieldy. With multi-objective Bayesian optimization, the design space exploration is made tractable and the design points found vastly outperform traditional methods across all metrics of interest.**

I. Introduction

Deep Neural Networks (DNNs) have recently gained attention due to their success in solving notoriously difficult classification and regression problems. However, their high computation and memory demands make them difficult to deploy and impractical in heavily constrained form factors, such as mobile and IoT devices. Dedicated hardware accelerators are one promising way of lessening the computational burden, offering orders of magnitude higher energy efficiency than general purpose processors.

Building a DNN accelerator is fundamentally a co-design problem: an architect must devise a system that achieves high prediction accuracy on the application task while simultaneously minimizing the amount of energy consumed doing so. Three factors make finding such a system challenging. First, the design space is known to be large and complex for both DNNs [20] and hardware accelerators [15] independently. Combined, the co-design requires properly tuning upwards of 14 parameters (see Table I). Second, the parameters tend to have non-obvious interactions. A parameter which might control a DNN structural trait can indirectly affect the way a datapath can be laid out (see Section II). Finally, evaluating the characteristics of a single design point is expensive, as it involves training a neural network and running hardware simulations. This limits the total number of points which can be explored. Together, these factors paint a picture of a multi-objective problem with a rough optimization landscape. An intelligent search strategy is needed to navigate the design space and arrive at a good solution.

This paper proposes using a machine learning technique, Bayesian optimization (BayesOpt), to solve the DNN accelerator co-design problem. BayesOpt has had success in machine learning circles as a solution for tuning DNN parameters alone [20]. The algorithm works by building a simplified statistical model of the parameter space and iteratively refining it as more data is collected. BayesOpt is able to gradually tease apart complex parameter interactions and avoids running experiments in unprofitable areas of the design space. This paper is the first to apply BayesOpt to a hardware and algorithm co-design problem and demonstrates how BayesOpt can be adapted to efficiently search the combined parameter space, outperforming conventional search algorithms.

TABLE I: Co-design Parameters

Parameter	Min	Max	Step	Type
Neurons per layer	10	200	1	DNN
Initial learning rate	0.001	1	Continuous	DNN
Learning rate decay	10	10^7	Continuous	DNN
Droptrain rate	0	0.4	Continuous	DNN
Dropout rate	0	0.4	Continuous	DNN
Max weight norm	0	10	Continuous	DNN
L2 regularization	0	0.1	Continuous	DNN
Initial momentum	0	0.9	Continuous	DNN
Final momentum	0	0.99	Continuous	DNN
Memory bandwidth	1	32	2^x	HW
Loop parallelism	1	32	2^x	HW
Pipelining	0	1	1	HW
Integer bits	1	32	1	HW
Fractional bits	1	32	1	HW

This paper makes the following contributions:

1) We build an automated framework to co-design DNN hardware accelerators for accuracy and energy efficiency by adapting multi-objective Bayesian optimization. The resulting set of Pareto-optimal designs strictly outperform those found with traditional methods.
2) We demonstrate that Bayesian optimization consistently samples better points than both grid search and a genetic algorithm over the course of the design space search. Bayesian optimization also results with a more densely populated Pareto frontier.
3) As the points are costly to evaluate, we also show that Bayesian optimization finds better designs with fewer samples. After only 42 samples, the designs found by Bayesian optimization outperform those found by other methods even after 200 samples are taken.

II. Background and Motivation

A. *Efficient Design Space Exploration*

The design space exploration required to find good DNN and hardware parameter settings is problematic and exhibits all the makings of a challenging optimization problem. The first challenge is due to the shear magnitude of the design space.

As the DNN accelerator space consists of upwards of 14 free parameters, a thorough exploration is infeasible. The second problem is that evaluating the energy and accuracy of a single design point is costly as it involves both training a DNN and simulating hardware. This expense limits the number of points that can be explored, requiring good designs to be found with minimal point evaluations. Finally, the effects of the parameters on the objectives (energy and accuracy), as well as between the parameters themselves, are intertwined, creating an unintuitive and rough optimization landscape.

Consider optimizing just two parameters: the bitwidth of fixed-point datatypes (a well known energy efficiency optimization [16]), and the L2 regularization parameter, used to prevent overfitting during training. While regularization is only intended to *directly* affect the accuracy objective, it also *indirectly* affects the energy objective: regularizing weights compresses their dynamic range, impacting the number of bits required to store the weights. Likewise, using a fixed-point datatype does not directly affect the regularization parameter, but it does reduce the resolution of a model's weights and the overall model accuracy, which both end up affecting the choice of regularization strength. Nearly all the design parameters in Table I impact both objectives, and the complex interactions between them are difficult to capture using traditional modeling techniques used to reason about hardware design spaces [11], [12].

B. The Optimization Problem

Generally speaking, the optimization problem revolves around finding the global optimal of an unknown objective function. Optimization methods typically work by evaluating a black-box function. However, most challenging optimization problems do not have closed form solutions and can only hope to be optimized via repeated evaluations of the function(s), i.e., exploring the design space by sampling points.

Traditional Solutions: A common starting point is to do a *grid search* (GS) over the design space. While this is a common approach found in computer architecture, it is the least effective. The fidelity of GS results depend on the strides of each parameter, and in high dimensional spaces it is extremely likely that the settings are poor. A provable improvement on GS is *stochastic grid search* (SGS) [3]. In SGS, the setting for each parameter for each iteration is sampled from a uniform random distribution, which provides much better value diversity over each individual dimension.

Optimization is a well-traversed field and SGS is not state-of-the-art. Common best practices include heuristics like simulated annealing, gradient decent, and genetic algorithms. Of the three, *genetic algorithms* (GA) are the most applicable here. While gradient descent and simulated annealing work well in certain circumstances, they tend to fall victim to local minima, and do not work well for the complex, rough optimization landscape of DNNs [20]. GA goes a step further by attempting to isolate *good* parameter combinations, saving them to be interchanged with others, and exploring more suspected-good points. GA is used as a comparison point for state-of-the-art hardware optimization algorithms.

A handful of more advanced solutions have been proposed as well. A solution to accelerator design space exploration based on decomposition was presented by Liu et al. [13]. The approach leverages locally-convex behavior of an optimization landscape to rapidly prune away sub-optimal components. However, this algorithm suffers in high-dimensional, tightly-coupled parameter spaces such as the one considered here. Zuluaga et al. propose a modeling approach similar to the one in this paper, but consider only a limited set of well-behaved hardware parameters in a design space that is small enough to be searched exhaustively [22]. A recent paper proposed using DNNs to model the energy cost of DNN hardware [19]. However, the paper considers a limited design space (at least ten orders of magnitude smaller than this work), and the on-line approach to training a DNN to model the design space is susceptible to becoming data-starved in high-dimensional problems with expensive sample functions.

C. Deep Neural Networks

Deep neural networks are machine learning models that can learn complex, non-linear functions. The basic structure consists of a series of neuron layers connected by weighted edges. Each individual neuron produces an activity output that is computed by summing all activity-weight products of the previous layer's neurons and subjecting the result to a non-linear activation function. The outputs of the final layer represent the classification prediction made by the DNN.

DNNs are used in two modes: *training* and *inference*. In training mode, DNN weight values are fit to labeled training data. As inputs are run through the model, a loss function is computed between the model's predictions and reference output values. The gradient of the loss function is then used to adjust its behavior, propagating backwards through the network proportional to the strength of the weights that produced the prediction. This backpropagation process repeats until weight values converge, at which point the DNN is considered trained, and its weights are fixed. In inference mode, the network only runs in the forward direction and is used to make predictions on new input data. In this paper, we focus on optimizing hardware accelerator designs just for inference, which is the more common use case in low-power devices.

D. Hardware Accelerators

A hardware accelerator is a fixed-function, application specific integrated circuit block that is typically FSM-controlled and expends minimal resources on superfluous circuitry. Accelerators have been shown to provide orders of magnitude more energy efficiency than general-purpose computing. Despite their narrow domain, the lack of predetermined structures and the ability to consider radically different techniques leads to a large design space [15].

Accelerating DNNs: Accelerating DNNs is a well studied topic [4], [5], [16]. While results and approaches vary, each concludes that the use of low-precision fixed-point arithmetic, specialized memories, and hardware/data reuse are essential for efficiency. The problem is that exploring the design space of DNN accelerators and optimizing for both accuracy and efficiency is challenging. The contributions of this paper are not to design the best DNN accelerator, but rather provide hardware designers with a powerful method to more thoroughly evaluate design options in a tractable manner.

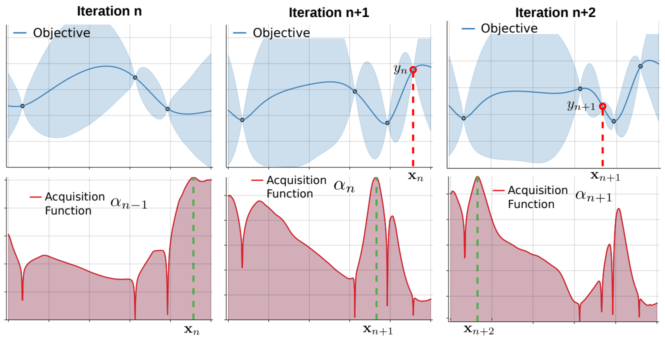

Fig. 1: Three iterations of BayesOpt are shown. Blue dots are sampled data points, the blue line is the GP's mean, and the shaded blue region shows model confidence bands. In Iteration n, the point x_n is chosen to be evaluated as it maximizes the AF. At $n+1$, a Bayesian step updates the GP to include point (x_n, y_n). This process is repeated for each sampled design point.

III. Bayesian Optimization

Bayesian optimization is a statistical framework that uses information gained from past experiments to model and minimize an arbitrary objective function. BayesOpt works by building and querying cheap *surrogate models* which estimate the behavior of real objective functions which are expensive to evaluate. Surrogate models are typically built using Gaussian Processes (GPs) [14]. GPs are fit to previously observed data and used to make predictions about the objectives' values in areas not yet explored. These predictions are easy to compute and can intelligently choose the next set of parameters such that solutions are found with a minimal number of expensive objective function evaluations.

We describe single-objective BayesOpt to provide context for this work. While we use a multi-objective formulation, the differences are beyond the scope of this paper and an overview can be found in Hernández-Lobato et al. [8]. BayesOpt finds the global minimizer $\mathbf{x}_\star = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$ of a function f over some bounded domain, typically $\mathcal{X} \subset \mathbb{R}^d$. It assumes that f can only be evaluated via expensive queries to a black-box that provides an output y_i given input $\mathbf{x}_i$. BayesOpt implements a sequential search algorithm that, after n iterations, proposes to evaluate f at some new location $\mathbf{x}_{n+1}$. To make this decision, the algorithm uses all previous observations $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, and leverages GPs to model the design space and make predictions about y_{n+1} given $\mathbf{x}_{n+1}$ and $\mathcal{D}_n$. The predictive distribution $p(y_{n+1}|\mathbf{x}_{n+1}, \mathcal{D}_n)$ is then used to guide the search for the global minimizer by computing an *acquisition function* (AF), $\alpha_n(\mathbf{x})$. The value of $\alpha_n(\mathbf{x})$ is given by the expected utility of evaluating f at $\mathbf{x}$ which is a measure of how useful it is to observe y_{n+1} at $\mathbf{x}$ with respect to the goal of optimizing f. Intuitively, an AF should balance exploiting known-good points and exploring unknown regions. Thus, the function $\alpha_n(\mathbf{x})$ should take on high values both in areas where the minima is most likely to lie given the past observations but also in areas where there are few samples. Rewarding exploration avoids local optima, and a balance between the two ensures quick convergence.

Example: Figure 1 illustrates the operations performed by BayesOpt. Top plots show the probabilistic predictions $p(y|\mathbf{x}, \mathcal{D}_n)$ of the data collected so far (blue dots) from evaluating the objective function. The mean of $p(y|\mathbf{x}, \mathcal{D}_n)$ is shown as a continuous blue line with plus/minus confidence bands of 3σ in light blue. The surrogate model of the space is used to compute the AF, shown in the bottom plots. This AF is globally maximized to find the next evaluation location for the objective, shown as a vertical green line. Middle and right plots show the result of adjusting the GPs to the available data plus the newly collected data-point, shown as a vertical red line.

A. Adapting Bayesian Optimization to Accelerator Co-design

Bayesian optimization is a framework, not a single algorithm. In this paper, we craft an implementation appropriate for the characteristics of DNN hardware design. First, we must decide which AF to use, as a variety are used practice [17]. A popular choice for tuning DNNs is expected improvement (EI) [20]. However, in the multi-objective setting EI requires objective scalarization (casting the multi-objective as a single-objective), which makes computing Pareto frontiers burdensome. Instead, we use Predictive Entropy Search (PES) [8]. The PES acquisition function approximates the expected information gain of a sample with respect to the optimization problem, where information is measured in terms of entropy. PES has been shown to be effective at tuning DNN parameters alone, and here it also enables a true multi-objective design space search.

In addition, hardware design has several characteristics which required a rethink of prior work. First, most hardware parameters are discrete-valued, whereas BayesOpt typically uses continuous parameters. While some methods exist for BayesOpt to handle discrete parameters, we found that allowing BayesOpt to treat the parameters as continuous and discretizing them afterwards to be more successful. Second, some hardware features can be characterized by multiple parameters, introducing uninformative regions which confuse the model. For example, the precision of fixed-point operations are usually written as a combination of the total number of bits as well as how many are used for the fractional piece. However, we found that parameters expressed in this fashion were unstable when the two values were close, slowing surrogate model convergence. We solved this by expressing overlapping parameters hierarchically; in the fixed-point example, it meant choosing the total number of bits and re-parameterizing the integer-fractional split as a ratio instead.

IV. Methodology and Design Flow

The design parameters considered for optimization are presented in Table I. These hardware parameters encompass the most pertinent aspects of accelerator design. Loop parallelism specifies how much parallel hardware is instantiated, which in the case of DNNs translates to how many neurons can be processed in parallel. Memory bandwidth governs how many ports and banks the SRAMs have. The DNN parameters are those typically used to specify and train a model. Note that while we are only considering accelerating inference, we still need to consider training parameters as they impact both accuracy and energy efficiency.

To measure the benefits of BayesOpt, we compare it with two traditional search techniques: SGS [3] and GA [6]. SGS is a common baseline used in the machine learning community

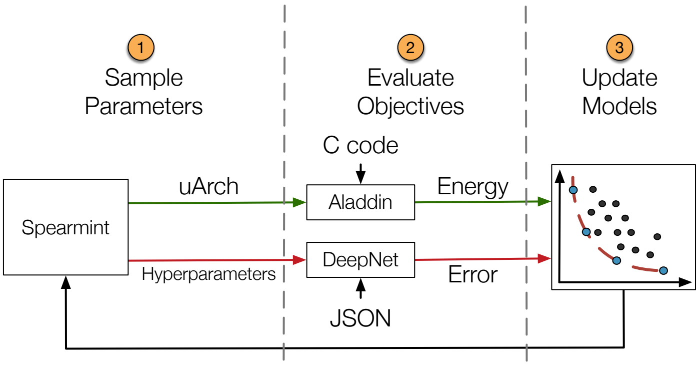

Fig. 2: The proposed design flow to explore the DNN accelerator design space.

and GA is one of the most widely used search optimizers. The MNIST [10] dataset is used as the target application of our accelerator design space search. MNIST is relatively small, especially compared to modern datasets, but it is widely seen as a standard for reasoning about new ideas and methodologies; larger datasets tend to make it difficult to compare and generalize [7]. Three tools were combined to explore the DNN accelerator design space. Spearmint [2] was used to perform BayesOpt and guide parameter sampling. To train and evaluate DNNs, we used the DeepNet [1] GPU library. Finally, the Aladdin [18] simulator was used to estimate energy numbers for accelerators.

Each optimization technique considers the objection functions as black boxes. While we chose to use Aladdin and DeepNet, the optimization methods are agnostic to the actual means by which the numbers are collected. If a user wanted more accurate hardware numbers they could replace Aladdin with a traditional hardware flow. The use of a simulator does not detract from the contributions of this paper as BayesOpt will equally outperform the other methods regardless of whether a simulator or CAD flow is used.

The accelerator design flow is presented in Figure 2. Each pass through this 3-step process produces one sample point, and the cycle is repeated to explore the design space.

Step 1—Sample Parameters: Each exploration iteration begins with Spearmint selecting values for each of the 14 parameters. Parameter values are chosen based on their ability to maximize expected utility (see Section III).

Step 2—Evaluate Objectives: Chosen parameters are then translated to a representation the evaluation tools can interpret. DeepNet uses a declarative JSON file for training DNNs, which is straightforward to generate given the sampled hyperparameters. Aladdin requires a C description of the DNN. We built a templated code generator to produce C implementations of DNNs as well as Aladdin constraint files containing hardware microarchitectural parameter settings.

Step 3—Update Design Space Models: The outputs from the function evaluations, model error and accelerator energy consumption, are fed back to Spearmint. Spearmint uses this information to update the posterior distribution of its surrogate model (i.e., the GPs). The AF can then be recomputed (bottom of Figure 1) and the process repeated.

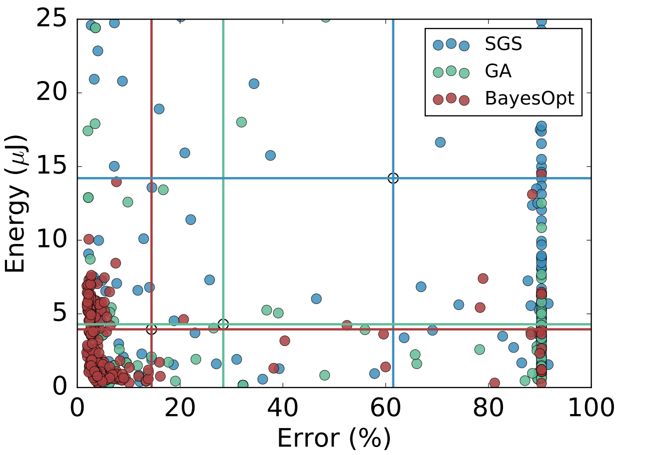

Fig. 3: Exploration results using SGS, GA, and BayesOpt. Crosshair points are the sample means. BayesOpt outperforms SGS and GA, finding more accurate, efficient designs.

V. Results

In this section we demonstrate the merits of applying BayesOpt to hardware design problems. We find that BayesOpt outperforms traditional design search methods (SGS and GA) across all major metrics of interest by: consistently sampling better design points, finding a strictly better Pareto frontier, and achieving both in fewer iterations.

A. *Bayesian Optimization Consistently Samples Better Designs*

Figure 3 depicts each design point evaluated by SGS, GA, and BayesOpt. The objective functions, error and energy, are plotted on the x- and y-axis, respectively (better designs are closer to the origin). Immediately obvious is the tendency for SGS points to cluster around the 90% error mark. This is a reflection of the underlying problem domain and speaks to its inherent challenges: MNIST is a classification problem with 10 categories, so an algorithm that randomly picked classes should choose correctly about 10% of the time. Thus, a DNN configuration with 90% error is effectively useless. The fact that SGS samples tend to fall in this area does not imply that SGS is a poor search algorithm. Rather, DNN optimization is a fundamentally difficult problem, and most configurations are in fact bad. Moreover, while it is hard to discover an accurate DNN, it is easy to ruin one: mis-setting even a single parameter is often sufficient to eliminate any model accuracy. The end result is a peaky optimization landscape where isolated good design points are surrounded by a floor of poor ones. Because SGS is an unbiased sampler of the design space, it is not unreasonable that almost 60% of its networks do no better than random guessing.

The results from using a genetic algorithm give some insight into the optimization space, despite its still-mediocre performance. GA tends to do well optimizing energy, but not accuracy. In Figure 3, it can be seen that there is still a large clustering of points around the 90% error mark, but compared to SGS, points tend to consume considerably less energy. This suggests that of the two objectives, the DNN training space is more difficult to reason about than the hardware space. This finding is somewhat intuitive, as there are several simple

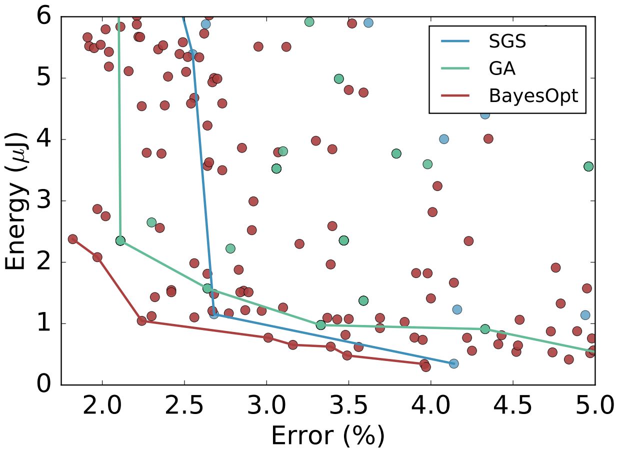

Fig. 4: Pareto frontiers for each optimization method. BayesOpt strictly outperforms both SGS and GA while SGS and GA overtake each other in different regions of the space.

relationships between hardware parameters which even a basic search algorithm could uncover—the tradeoff between datapath parallelism and memory bandwidth, for instance. While GA is able to exploit these simple trends, it has difficulty optimizing both objectives simultaneously. While BayesOpt and GA have similar mean energy consumption, BayesOpt finds designs with substantially lower mean error.

BayesOpt's strength comes from modeling intricate relationships between parameters and avoiding cliffs in the optimization landscape. This is clear from its sample distribution: the majority of its evaluated design points have low error and low energy. While BayesOpt does produce a small number of ineffective network configurations, this is partially by design: BayesOpt seeks to strike a balance between exploitation and exploration. Many of these inaccurate or high-energy points are an insurance plan against getting stuck in a local minimum.

The red, green, and blue cross-hairs represent the average energy consumption and prediction error for points chosen by BayesOpt, GA, and SGS respectively. SGS designs have an average error of 61.4% using 14.2μJ and GA has an average error of 28.4% and consumes 4.29μJ. BayesOpt does best with an average of 14.4% model error and 3.95μJ energy usage.

B. *Bayesian Optimization Produces a Superior Pareto Frontier*

Figure 4 compares the Pareto frontiers selected from the points shown in Figure 3. For every SGS and GA Pareto point, there is at least one point on the BayesOpt Pareto frontier which dominates it in both energy and accuracy. This is intuitive given the observations in the previous section: Pareto frontiers represent the extrema of a sample distribution and thus nearly always have very low probability. Because SGS is a flat distribution over the parameter space, it rarely samples in the Pareto region. GA does reasonably well on the energy objective but significantly under-performs on DNN accuracy. Because BayesOpt intentionally picks points expected to have low-energy and low-error, its resulting sample distribution skews heavily towards that region, producing a higher-quality Pareto frontier.

Consider the best design points achieved for a target energy budget of 1μJ. This results in a DNN model with an error of 3.4% and 2.7% for GA and SGS respectively. BayesOpt does substantially better with a error of 2.25%. Along the other dimension, if a designer has a target error rate of 2.2%, the best BayesOpt design uses 58% less energy than GA, and SGS is unable to find any design which can meet this target.

BayesOpt also produces a larger number of points on and near the frontier. A more populated Pareto frontier and dense surrounding region offers several benefits. First, larger number of points *on* the frontier provides designers more flexibility. For instance, a designer using BayesOpt with an energy budget of 0.5-1.0 μJ would have four different designs to choose from. With SGS, the designer has only one viable solution, and a poor solution at that. While GA offers slightly more flexibility than SGS, these points actually perform worse than the designs found by either of the other methods. BayesOpt provides the best of both worlds: flexibility and superior performance.

Second, a high concentration of points *near* the Pareto frontier gives some confidence that the best designs are not coincidental and that the search is close to the global optimum. The concentration of BayesOpt (red) points near the Pareto frontier is a result of the surrogate model accurately modeling the design space and exploiting good parameter settings. The sparsity of GA and SGS points raises concerns: both are stochastic algorithms, so what if a different seed had been chosen? The paucity of high-quality results offers little reassurance to a designer that their next search will stumble across an optimal configuration. BayesOpt finds *dozens* of points near the Pareto frontier, so even under different circumstances, its results are unlikely to change substantially—it will still identify a satisfactory Pareto set.

C. *Bayesian Optimization Discovers Optimal Designs Faster*

Evaluating a DNN accelerator is a costly task. For our experiments, training and simulation could each take hours per sample, and it is not uncommon for modern DNNs to take days to train a single network configuration. Consequently, we want the benefits from the preceding two sections in as few samples as possible. We can compare the sample efficiency of SGS, GA, and BayesOpt by analyzing the evolution of their Pareto frontiers as a function of the cumulative number of samples evaluated. This is done via the notion of a Pareto *hypervolume* [21]. Hypervolume is the integral between a Pareto frontier and a fixed reference point in the objective space, which captures the region of interest for the design space in question.

The arbitrary units used to measure hypervolume can make it difficult to reason about what the results mean. A good way to interpret the results is to compare the hypervolume curves against each other, which is fair since all three are computed using the same reference point. This allows us to draw the following conclusion: in the converged state, the relative improvement of GA relative to SGS is the same as BayesOpt is to GA. This implies the superiority of designs with respect to quality of results found with BayesOpt are the same and the difference between GA and SGS (a random search). As GA is an intelligent optimization algorithm, this provides significant evidence as to the power and merit of applying BayesOpt to accelerator design problems.

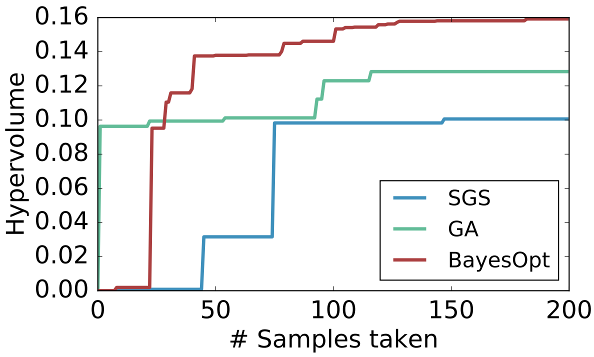

Fig. 5: The hypervolume gained from each sampled point is shown. The evolution of the Pareto frontiers shows the quality of results scales with the number of evaluations.

While both SGS, GA, and BayesOpt achieve nonzero hypervolume after just 11, 2, and 9 samples, respectively, BayesOpt quickly outperforms the traditional solutions. After only 32 samples, BayesOpt reaches a value of 0.12 (arbitrary units), but it takes SGS 97 samples (3× more) to find a Pareto frontier with similar coverage. Once BayesOpt samples 42 points, its Pareto frontier covers more of the objective space than SGS and GA over all our experiments.

Comparing the hypervolumes for GA and BayesOpt provides interesting takeaways from this experiment. Most notable, is that between iterations 2 and 29 *GA actually outperforms BayesOpt*, meaning it found a better point earlier. While this may seem to weaken the justification for using BayesOpt, it in fact strengthens the case. When GA begins sampling points from the space, points are sampled randomly (as in SGS). So the large spike in hypervolume at point 2 is entirely coincidental, and yet after 42 iterations, BayesOpt still outperformed GA for the remaining samples. This means that even though GA had what turned out to be its best point in its entire search by only the second iteration, it was unable to leverage this to effectively explore the design space. On the other hand, once BayesOpt had a reasonable surrogate model of the space, it was able to exploit its samples, consistently finding improvements to its Pareto frontier.

VI. Conclusion

This paper proposes using machine learning methods to design better hardware. We demonstrate how to adapt multi-objective Bayesian optimization to co-design deep neural network parameters and accelerator hardware parameters to simultaneously maximize the accuracy of the DNN and the energy efficiency of the hardware. We find Bayesian optimization substantially outperforms existing optimization methods across all relevant metrics.

We are optimistic about the future applications of machine learning in hardware design and optimization. Many of the statistical building blocks we use in this paper have only matured in the last few years, and continued progress in algorithmic techniques will only reinforce the results seen here. E.g., a recent advancement has shown how decoupled Bayesian optimization can speed up search time by evaluating hardware and algorithmic objectives separately [9]. We believe that techniques like Bayesian optimization will eventually become core components of the next generation of computer-aided hardware design tools.

Acknowledgments

This work was partially supported by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This research was, in part, funded by the U.S. Government under the DARPA CRAFT and PERFECT programs (Contract #: HR0011-13-C-0022). Intel Corporation also provided support. J.M.H.L. acknowledges support from the Rafael del Pino Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government or other sponsors.

References

[1] DeepNet: implementations of deep learning algorithms. https://github.com/nitishsrivastava/deepnet. 2014.

[2] Spearmint: a package to perform bayesian optimization. https://github.com/JasperSnoek/spearmint, 2016.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012.

[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. ASPLOS, 2014.

[5] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. ISCA, 2016.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 2002.

[7] R. Grosse. Which research results will generalize? https://hips.seas.harvard.edu/blog/2014/09/02/which-research-results-will-generalize, 2014.

[8] D. Hernández-Lobato, J. M. Hernández-Lobato, A. Shah, and R. P. Adams. Predictive entropy search for multi-objective bayesian optimization. *ICML*, 2016.

[9] J. M. Hernández-Lobato, M. A. Gelbart, B. Reagen, R. Adolf, D. Hernández-Lobato, P. N. Whatmough, D. Brooks, G.-Y. Wei, and R. P. Adams. Designing neural network hardware accelerators with decoupled objective evaluations. In *NIPS workshop on Bayesian Optimization*, 2016.

[10] Y. Lecun and C. Cortes. The MNIST database of handwritten digits.

[11] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. ASPLOS, 2006.

[12] H.-Y. Liu and L. P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. DAC, 2013.

[13] H. Y. Liu, M. Petracca, and L. P. Carloni. Compositional system-level design exploration with planning of high-level synthesis. In *DATE*, 2012.

[14] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[15] B. Reagen, Y. S. Shao, G.-Y. Wei, and D. Brooks. Quantifying acceleration: Power/performance trade-offs of application kernels in hardware. ISLPED, 2013.

[16] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernndez-Lobato, G.-Y. Wei, and D. Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. ISCA, 2016.

[17] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 2016.

[18] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. ISCA, 2014.

[19] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. ICCAD, 2016.

[20] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

[21] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 1999.

[22] M. Zuluaga, A. Krause, P. Milder, and M. Püschel. Smart design space sampling to predict pareto-optimal solutions. LCTES, 2012.