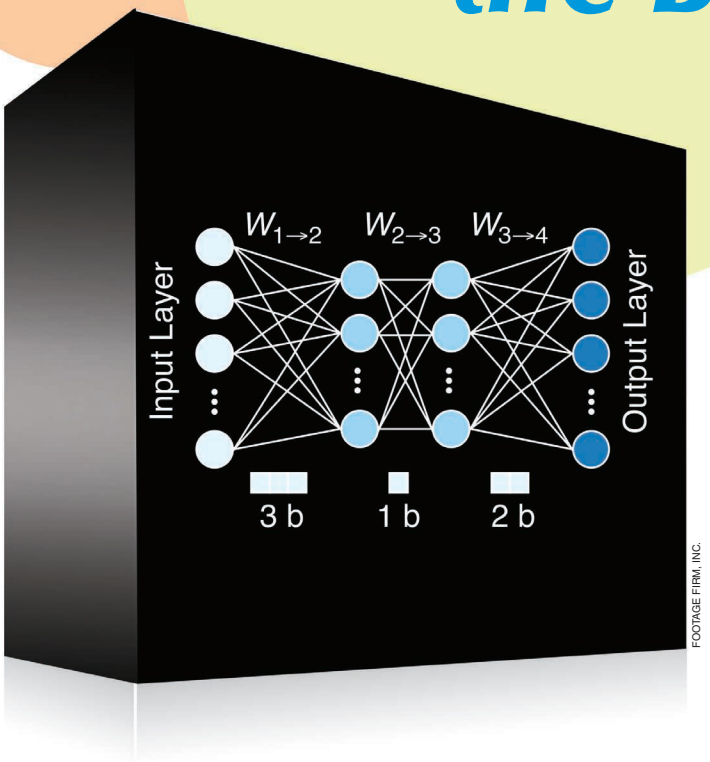
Peeking Into the Black Box



FOOTAGE FIRM, INC.

A tutorial on automated design optimization and parameter search

Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar

Automated optimization algorithms are extensively used to search for optimal design parameters in applications ranging from designing compilers and analog circuits to crafting embedded machine-learning technology. System optimization is particularly laborious as the pertinent design space includes various conflicting objectives and a swarm of free parameters. Automating parameter optimization can enhance system quality while ensuring low design cost and high performance. In this tutorial, we review several recent optimization tools for selecting design parameters and architecture. These tools include heuristic methods, reinforcement learning (RL), and evolutionary strategies. We elaborate on the basics of each algorithm and explain the benefits and tradeoffs when each is applied to system design. To demonstrate how these methods can be applied, we review several real-world examples: analog circuit design, neural network compression, and the design of domain-specific accelerators. We also touch on possible research directions and remaining challenges that need to be addressed.

Digital Object Identifier 10.1109/MSSC.2019.2939336
Date of current version: 18 November 2019

Automated Design Optimization Tools

With the ever-growing demand for lower-power, higher-performance, and more compact analog and digital circuits, manual design has become increasingly laborious and time consuming. Meanwhile, as designs become more complex and the number of system components increases, the underlying space of design choices expands exponentially. Therefore, an automated circuit optimization tool that can reduce engineering cost and ensure ease of use, accuracy, and generalizability is highly desirable.

Circuit design can be viewed as an optimization problem consisting of multiple objectives (e.g., minimizing power consumption), constraints (e.g., linearity), and design hyperparameters (e.g., circuit component values). The pertinent constrained multiobjective optimization can be solved using the rich reservoir of optimization tools introduced in the literature. In general, the circuit-design process can be divided into two main phases:

1) topological design, where the desired functionality is implemented by connecting circuit components

The goal of a multiobjective optimization algorithm is to find a set of solutions that lie on the Pareto frontier curve.

2) circuit sizing, where the focus is on parameter-level optimization for a fixed circuit topology.

Optimization methodologies have been applied to circuit sizing, including the sizing of RF circuits [1], [2], operational amplifiers [3], and other analog circuits [4]–[9]. The applications for automated optimization tools are not limited to analog circuit sizing. In general, any system-design task that requires adjusting a set of hyperparameters can benefit from automated optimization methodologies.

System design on a high level can be directly translated to optimizing a score function $F(x)$:$\mathbb{R}^d \rightarrow \mathbb{R}$, where x is a d-dimensional vector of hyperparameters and $F(\cdot)$ is an arbitrary measure of quality. This measure can be formulated by combining several design objectives that characterize system performance, e.g., power consumption, area, and throughput. The underlying optimization task can be regarded as a search over the values of $F(x)$ in a d-dimensional space where the boundaries are specified by the design constraints. A good optimization algorithm is one that can be globally applied to arbitrary problems and can find the global optima of $F(\cdot)$ with a low search overhead.

Optimization can be performed using two different approaches: numerical methods and black-box methods. Numerical methods assume access to the underlying algebraic model that relates design hyperparameters to the score function [10]. The task of finding the underlying mathematical expressions is often exhaustive, if not impossible. Black-box methods address this challenge by assuming that the score function is not explicitly known and thus merely resorting to empirical (noisy) evaluations of $F(x)$. In this article, we explain the fundamentals of three black-box optimization methods. For each method, we provide practical system-design examples to illustrate its applicability in real-world tasks. We conclude by suggesting future research directions and areas worth exploring.

Problem Formulation

System design can be viewed as an optimization problem where design choices represent the optimization variables (hyperparameters). In this setting, system constraints and design requirements represent the objective functions for optimization. After the design hyperparameters and corresponding objective functions are determined, the hyperparameters are translated to a vectorized representation. Various optimization tools can then be used to explore the underlying hyperparameter vector space and find the solution that results in a good performance. For a vector of hyperparameters ($x \in \mathbb{R}^d$), we assume access to an oracle, $F(x)$:$\mathbb{R}^d \rightarrow \mathbb{R}$, that serves as an objective function to the optimization problem. The optimization goal is thus to find the maxima (or minima) of the objective function, with the caveat that $F(\cdot)$ is not known in advance. To provide a better understanding, let us consider the example of circuit design.

To perform automated circuit optimization, we assume an initial expert-designed mapping of circuit elements and seek to enhance the design, given a set of user-defined objectives. Toward this goal, the first step is identifying the design variables, i.e., a subset of hyperparameters $\{x_i\}_{i=1}^{d}$ that can be tuned to improve the design objective. As an example, hyperparameters can be analog component characteristics, e.g., capacitance, resistance, and transistor dimensions. The design hyperparameters are appended to form a vector representation $x \in \mathbb{R}^d$. Enhancing analog circuit performance can then be formally defined as a multiobjective, constrained optimization. Here, the objective functions $f_1(x), \ldots f_m(x)$ represent different circuit performance metrics, i.e., the figure of merit. Examples of the objective function include power, noise resiliency, and linearity.

In reality, the objectives often conflict with one another. The goal of a multiobjective optimization algorithm is thus to find a set of solutions that lie on the Pareto frontier curve. By definition, Pareto optimal solutions are those that cannot be improved in any of the objective functions without harming at least one other objective. In general, extracting all Pareto optimal solutions can be rather costly for real-world complex circuits. A relaxed alternative is to solve a single-objective optimization of a score function instead. The score $F(\cdot)$ is a carefully designed combination of $f_i(\cdot)$s that incorporates the pertinent tradeoffs among the objective functions. More specifically, the score function $F(\cdot)$ represents a goodness measure for a given vector of circuit hyperparameters. Therefore, the optimization goal is equivalent to maximizing $F(\cdot)$ and obtaining a "good" design that adheres to the desired characteristics. A simple example of such a score function is a linear combination of objective functions $F(x) = \Sigma_{i=1}^{m} w_i f_i(x)$. The relaxed optimization objective can, therefore, be formalized as

$$\underset{x}{\text{maximize}}\ F(x) \quad \text{s.t.} \quad \begin{cases} C_i(x) \geq 0 & \forall\ i \in \{1, \ldots, k\} \\ x_i \in [b_i^{\min},\ b_i^{\max}] & \forall\ i \in \{1, \ldots, d\}, \end{cases} \tag{1}$$

where $C_i(x)$ is a set of constraints that are either user-defined limits on certain design metrics, e.g., power, or rules imposed by analog circuit laws, e.g., Kirchhoff's current law and Kirchhoff's voltage law. Another set of constraints corresponds to boundaries on design hyperparameters, $[b^{\min}, b^{\max}]$, as a result of the fabrication process or specific design requirements. In the following sections, we elaborate on three contemporary methods for hyperparameter optimization: greedy algorithms, RL,

and evolutionary strategies. We provide several system design examples for the optimization techniques but emphasize that these methods are generic and applicable to a wide variety of problems.

Grid-Search and Heuristic Algorithms

Grid Search

One way to solve the maximization problem of (1) is to evaluate all possible parameter settings and select the one that renders the best score function. Naively evaluating all possible parameters can be quite costly. That's why smart grid-search algorithms are highly preferred. In many system-design tasks, the total number of possible configurations can be reduced to a much smaller set. As an example, we consider a robustness-assurance task where a set of defender modules is used to protect a machine-learning model against integrity attacks. We briefly discuss the task here (Figure 1); for detailed system specifications, refer to [11]. The system to be designed in this example is a domain-specific accelerator implemented on a field-programmable gate array (FPGA) that can perform the computations of all defender modules as fast as possible. We assume that N_{def} defender modules are to be executed in parallel while each defender itself uses N_{PU} processing units from the underlying hardware accelerator. The limiting factors in this example are hardware resources, such as the available digital signal processing (DSP) units (*#DSP*), memory (*#BRAM*), flip-flops (*#FF*), and logic gates (*#LOGIC*) on the underlying FPGA board. The objective here is to maximize the number of defender modules by setting N_{def} and N_{pu}, while abiding by throughput and platform constraints. This problem can be efficiently solved using a grid-search approach: for a fixed N_{def}, the total available resources for each defender module are $(\#DSP/N_{\text{def}})$, $(\#BRAM/N_{\text{def}})$, $(\#FF/N_{\text{def}})$, and $(\#LOGIC/N_{\text{def}})$. Given these constraints, the maximum values for N_{PU} and the throughput are uniquely determined. Therefore, the dimensionality of the search space can be reduced by one. We evaluate the throughput under different N_{def}s and the corresponding unique N_{PU} and select the configuration with the maximum N_{def} that satisfies the throughout requirement. Domain-specific accelerator design [12] for machine-learning applications is another example use case of grid-search methodologies for system optimization. Unlike the previous examples, many tasks are not reducible to smaller problems due to the interdependency among the optimization parameters. One approach for tackling such tasks is to use a greedy-search algorithm.

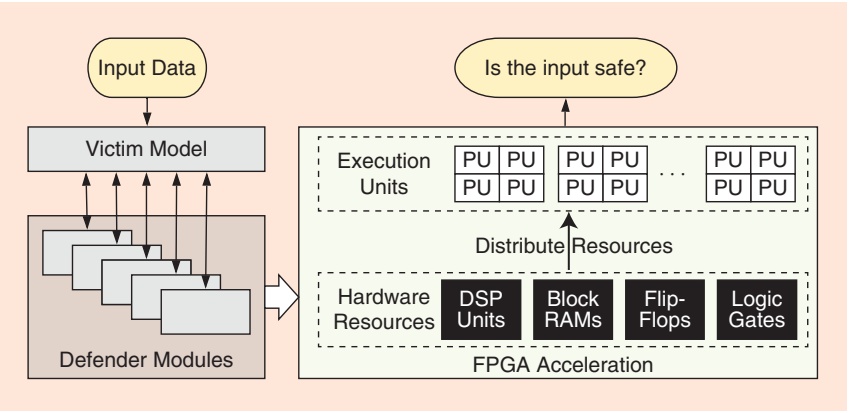


FIGURE 1: A chart illustrating domain-specific FPGA acceleration for defending a victim machine-learning model against integrity attacks. The number of parallel defender modules (N_{def}) and the per-unit parallelism factor (N_{PU}) should be specified according to the FPGA's resource availability to obtain optimal performance. PU: processing unit; RAMs: random access memories.

Heuristic (Greedy) Algorithms

A greedy algorithm starts from an initial hyperparameter vector, x, and modifies this vector toward a better solution in a step-by-step manner. Let us consider the task of deep neural network (DNN) compression [13], [14] as an example optimization problem. DNNs are hierarchical architectures formed by stacking multiple layers (Figure 2). Each layer extracts a set of features from an input vector. In particular, the l^{th} layer performs a linear operation between the layer input and a weight tensor $W_{l\to l+1}$, followed by a nonlinearity. The weight matrix is known as the *layer parameter*. The goal of DNN compression is to reduce the model complexity by applying a compression technique. Figure 2 presents an example compression technique for quantization that aims to reduce the DNN's memory footprint by lowering the number of bits required for model storage and computation. In this scenario, the optimization hyperparameters are the number of bits used to represent the model's parameters/outputs. These hyperparameters should be specified for each layer of the DNN. The parameter space to be searched is, therefore, a vector $x \in \mathbb{R}^L$ for an L-layer DNN, where the i^{th} element $x[i]$ is the bitwidth for layer i.

In this example, there are two conflicting design objectives: high DNN classification accuracy $f_1(x)$ and a small total memory footprint $f_2(x)$. As we reduce the bitwidths, the memory footprint decreases, but the classification accuracy also drops, which is highly undesirable. Our goal is to

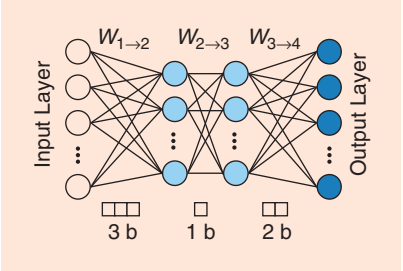


FIGURE 2: An example of a DNN quantization task where the goal is to specify the per-layer bitwidths.

In an RL setting, an autonomous agent interacts with an environment.

find a set of per-layer quantization bitwidths x that maximize a score function $F(x)$, which is a combination of $f_1(x)$ and $f_2(x)$.

The score function is specifically designed to minimize $f_2(x)$ while maximizing $f_1(x)$. The greedy algorithm starts with a vector $x^0 = \{B, B, \ldots, B\}$ with a maximum quantization bitwidth B at all layers. The superscript here denotes the algorithm step. At step t, the parameter vector x^t is updated to x^{t+1}. There are L candidates for the parameter vector x at the next step, where the i^{th} candidate is obtained by changing $x^t[i] \rightarrow x^t[i] - 1$ while keeping all other elements $x^t[j](j \neq i)$ constant. Among these, the candidate that results in the maximum score function $F(x)$ is selected, and the vector x^t is updated accordingly. While this algorithm is fairly simple, it can achieve a better compression rate and accuracy compared to DNNs designed by human experts. See [13] and [14] for a detailed explanation of the algorithm applied to other compression techniques.

Reinforcement Learning

In an RL setting, an autonomous agent interacts with an environment. Through this interaction, the agent gradually learns to adjust its behavior based on the consequences of its decisions (actions). Figure 3 demonstrates the RL loop. The agent, which is generally controlled by a machine-learning algorithm, observes a state s_t of the environment at each time step. The agent then uses a policy π to choose an action a_t based on the observed state s_t. More specifically, a policy specifies a probabilistic distribution for possible actions a given the state S, i.e., $\pi: S \longrightarrow P(A = a|S)$. As a result of the agent's action under policy π, the environment state transitions to s_{t+1}, and a reward r_{t+1} is produced. The reward acts as a feedback for the RL agent that determines the quality of the chosen action in the previous time step. The RL agent thus aims to learn an optimal policy, π^*, that maximizes the cumulative (discounted) reward. In the context of automated optimization, the state s_t is the current choice of design hyperparameters, i.e., the vector x. The action a changes x with the hope of increasing $F(x)$. The per-step reward r_t is, therefore, defined as the increase in the score function from one step to the next.

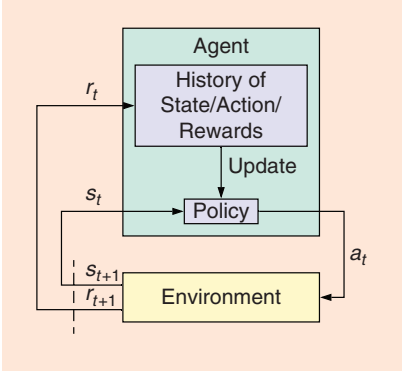


FIGURE 3: An illustration of the RL setting.

Let us consider an example use case of RL in circuit optimization [15] where the optimized circuit parameters are learned after several rounds of interaction with a circuit-simulation environment. During each interaction, the RL agent collects from the simulated circuits a set of observations that include

- the global and high-level simulation results, e.g., the dc operating point voltage and current at each node and the ac amplitude/phase responses
- the local element-level results, e.g., the feature of each transistor containing V_{th}, g_m, V_{dsat}, and so on.

Based on the acquired observations from the simulation environment, the RL agent outputs an action that determines the new value for circuit hyperparameters, e.g., transistor width and lengths, capacitances, and resistances. The optimization objectives in this problem are the set of specifications that need to be satisfied by the design. For each action chosen by the RL agent, the simulation environment outputs a set of performance metrics achieved by the circuit with the corresponding parameter values. The RL agent's reward is then defined to represent the extent to which the chosen circuit parameters satisfy the desired design specifications. Besides circuit optimization, RL-based methods have shown success in a variety of other optimization tasks, e.g., device placement [16] and battery management [17]. The RL approach generally has a higher sample efficiency compared to grid-search approaches, meaning that it requires fewer circuit simulations to obtain a design with similar performance quality. However, the complexity of the incorporated machine-learning models renders the training of the RL agent computationally expensive and excessively time consuming. Furthermore, due to the sequential and iterative nature of RL training, the runtime of the optimization algorithm cannot be reduced by parallelism techniques.

Evolutionary Strategies

Evolutionary strategies are metaheuristic optimization approaches inspired by natural evolution and the notion of survival of the fittest. These methods can be used to perform automated parameter tuning for various system-design tasks. The core idea is to create a collection of hyperparameter vectors $\mathbb{P} = \{x_1, x_2, \ldots, x_n\}$ and gradually change them to achieve a better system design on average. In this context, each vector x_i is called an *individual* and the collection $\mathbb{P}$ is called a *population*. At step t of the algorithm, the population is updated $\mathbb{P}^t \rightarrow \mathbb{P}^{t+1}$ in a way that the hyperparameter vectors in $\mathbb{P}^{t+1}$ are equally good or better than those in $\mathbb{P}^t$ on average.

The process of evolving a population (Figure 4) consists of four consecutive steps. In the evaluation step, the optimization score function $F(\cdot)$

is calculated over the population, and individuals are assigned scores representing their quality, i.e., fitness. The selection step then samples from the current population based on each individual's fitness score. In this process, superior individuals (with high scores) are passed to the next phase, and inferior individuals (with low scores) are eliminated. The next two steps, dubbed *crossover* and *mutation*, operate on the surviving individuals. Crossover and mutation aim to explore the proximity of the selected individuals by injecting small random perturbations. More specifically, crossover simultaneously combines two individuals and generates two offspring hyperparameter vectors. Mutation acts on each individual by randomly tweaking the vector elements. By continuously updating the population using the aforementioned steps, the average score across the population gradually increases. The optimization algorithm converges once the average score stops improving, and the individual with the highest score renders the optimal hyperparameter configuration.

Evolutionary strategies offer several benefits when used for automated system design:

- The simplicity of evolutionary algorithms makes them amenable to many system-design tasks. The computational burden of evolutionary strategies is much lower than it is for machine-learning-based approaches, such as RL.
- The computations involved in evolutionary strategies are highly parallelizable, as the evaluation of individuals can be performed independently. As a result, these methods offer scalability, especially in many-core and distributed-computing platforms.
- Evolutionary strategies can effectively learn to adopt multiple optimization objectives simultaneously. This property provides a low optimization cost for various engineering tasks [18].

We consider a DNN-compression task to illustrate evolutionary strategies. (We briefly explain the problem here. See [19] for a detailed discussion.) Figure 5 presents the overview of this task, where a set of compression techniques is available for application to each layer of the DNN. This optimization task requires choosing a subset of available compression techniques and their corresponding compression intensities. The optimization objective is to effectively compress the DNN such that the overall classification accuracy is minimally affected while the hardware performance is maximally improved. This can be done by formulating DNN compression as an evolutionary process [19].

The individual vectors are obtained by appending the per-layer compression rates into the vector x. For each such vector, the corresponding score $F(x)$ is achieved by compressing the DNN with x, running it on the target hardware, and measuring/estimating the execution cost and classification accuracy. By creating populations of such individual vectors and iteratively applying genetic operations to them, one can achieve superior DNN architectures compared to human-expert designs [19]. While we illustrate the usability of evolutionary strategies for DNN compression, we emphasize that this method can also be applied in other domains, including those involving analog circuit optimization [8], [20], neural architecture search [21], and robotic control [22].

Conclusions and Future Direction

We reviewed here several black-box optimization tools and showed their potential in automating hyperparameter selection for various applications. We illustrated each optimization scheme using a practical system-design task. While contemporary researchers have applied these intelligent schemes to system designs on a small scale, current optimization

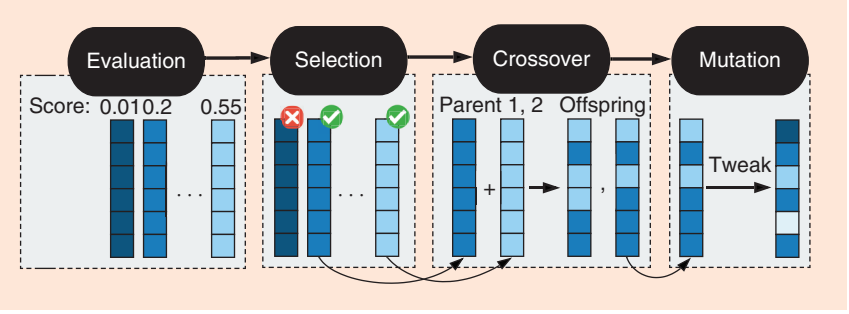


FIGURE 4: The evolution process of a population.

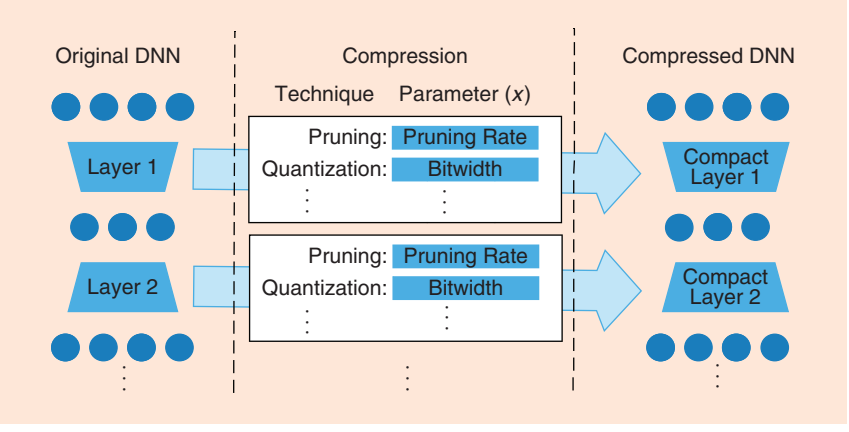


FIGURE 5: A DNN compression task. Each layer of the network should be compressed using a set of compression techniques. Each compression technique has a hyperparameter, e.g., pruning rate and quantization bitwidth, that controls the amount of compression applied.

methods can perform poorly for large-scale designs with millions of hyperparameters. In addition, for many design tasks, simulating system performance for each hyperparameter configuration is quite costly and time consuming. Therefore, devising sample-efficient optimization methodologies that can find the near-optimal hyperparameters by means of few simulations can greatly increase the applicability of these tools in the design of more complex systems. On a separate note, most research articles assume a predetermined architecture and aim to optimize the hyperparameters in that setting. Designing the underlying system architecture still highly relies on human experts, and automating this process remains a standing challenge. As such, developing novel optimization techniques that can effectively perform low-level design choices is highly desirable.

References

[1] S. H. Yeung, W. S. Chan, K. T. Ng, and K. F. Man, "Computational optimization algorithms for antennas and RF/microwave circuit designs: An overview," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 216–227, 2012.

[2] P. Mandal and V. Visvanathan, "CMOS op-amp sizing using a geometric programming formulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 1, pp. 22–38, 2001.

[3] M. delM. Hershenson, S. P. Boyd, and T. H. Lee, "Optimal design of a CMOS op-amp via geometric programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 1, pp. 1–21, 2001.

[4] A. Jafari, E. Bijami, H. R. Bana, and S. Sadri, "A design automation system for CMOS analog integrated circuits using new hybrid shuffled frog leaping algorithm," *Microelectron. J.*, vol. 43, no. 11, pp. 908–915, 2012.

[5] R. Vural and T. Yildirim, "Analog circuit sizing via swarm intelligence," *AEU-Int. J. Electronics Commun.*, vol. 66, no. 9, pp. 732–740, 2012.

[6] D. Ghai, S. P. Mohanty, and E. Kougianos, "Design of parasitic and process-variation aware nano-CMOS RF circuits: A VCO case study," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 9, pp. 1339–1342, 2009.

[7] D. Ghai, S. P. Mohanty, and G. Thakral, "Fast optimization of nano-CMOS voltage-controlled oscillator using polynomial regression and genetic algorithm," *Microelectron. J.*, vol. 44, no. 8, pp. 631–641, 2013.

[8] P. K. Rout, D. P. Acharya, and G. Panda, "A multiobjective optimization based fast and robust design methodology for low power and low phase noise current starved VCO," *IEEE Trans. Semicond. Manuf.*, vol. 27, no. 1, pp. 43–50, 2013.

[9] W. Lyu et al., "An efficient Bayesian optimization approach for automated optimization of analog circuits," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 65, no. 6, pp. 1954–1967, 2017.

[10] A. Mirhoseini, B. D. Rouhani, E. M. Songhori, and F. Koushanfar, "Perform-ML: Performance optimized machine learning by platform and content aware customization," in *Proc. 2016 53nd ACM/EDAC/IEEE Design Automation Conf. (DAC)*, pp. 1–6.

[11] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, "Deepfense: Online accelerated defense against adversarial deep learning," in *Proc. 2018 IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, pp. 1–8.

[12] M. Samragh, M. Ghasemzadeh, and F. Koushanfar, "Customizing neural networks for efficient FPGA implementation," in *Proc. 2017 IEEE 25th Annu. Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, pp. 85–92.

[13] M. Samragh, M. Javaheripi, and F. Koushanfar, CodeX: Bit-flexible encoding for streaming-based FPGA acceleration of DNNs. 2019. [Online]. Available: arXiv:1901.05582

[14] M. Samragh, M. Javaheripi, and F. Koushanfar, "Autorank: Automated rank selection for effective neural network customization," in *Proc. ML-for-Systems Workshop 46th Int. Symp. Computer Architecture (ISCA)*, 2019, pp. 1–6. [Online]. Available: https://mlforsystems.org/assets/papers/isca2019/MLforSystems2019_Mohammad_Samragh.pdf

[15] H. Wang, J. Yang, H.-S. Lee, and S. Han, Learning to design circuits. 2018. [Online]. Available: arXiv:1812.02734

[16] A. Mirhoseini et al., "Device placement optimization with reinforcement learning," in *Proc. 34th Int. Conf. Machine Learning,* 2017, pp. 2430–2439.

[17] A. Mirhoseini and F. Koushanfar, "Hypoenergy. Hybrid supercapacitor-battery power-supply optimization for energy efficiency," in *Proc. 2011 Design, Automation & Test in Europe*, pp. 1–4.

[18] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[19] M. Javaheripi, M. Samragh, T. Javidi, and F. Koushanfar, "Ascai: Adaptive sampling for acquiring compact AI," in *Proc. AutoML Workshop 36th Int. Conf. Machine Learning (ICML)*, 2019, pp. 1-8. [Online]. Available: https://www.automl.org/wp-content/uploads/2019/06/automlws2019_Paper29.pdf

[20] B. Liu et al., "Analog circuit optimization system based on hybrid evolutionary algorithms," *Integration, VLSI J.*, vol. 42, no. 2, pp. 137–148, 2009.

[21] L. Xie and A. Yuille, Genetic CNN. 2017. [Online]. Available: https://arxiv.org/abs/1703.01513

[22] B. Patle, D. Parhi, A. Jagadeesh, and S. K. Kashyap, "Matrix-binary codes based genetic algorithm for path planning of mobile robot," *Comput. Electr. Eng.*, vol. 67, pp. 708–728, 2018.

About the Authors

Mojan Javaheripi (mojan@ucsd.edu) received her B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2017. She is a Ph.D. student in the Department of Electrical and Computer Engineering, at the University of California, San Diego. Her research interests include the intersection of machine learning and computer architecture. Her projects aim at codeveloping machine learning algorithms and their corresponding specialized hardware with the goal of maximizing efficiency and performance. She is the recipient of the 2019 Qualcomm Innovation Fellowship for her efforts in black-box optimization.

Mohammad Samragh (msamragh@ucsd.edu) received his B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2015 and his master's degree from the University of California, San Diego (UCSD), in 2018. He is a Ph.D. student in the Department of Electrical and Computer Engineering, at UCSD. His research interests include hardware–software codesign for embedded machine learning, adversarial deep learning, and privacy-preserving deep learning. He is the recipient of the 2019 Qualcomm Innovation Fellowship for his efforts in black-box optimization.

Farinaz Koushanfar is a professor and Henry Booker Faculty Scholar in the Electrical and Computer Engineering Department at the University of California, San Diego (UCSD), where she directs the Adaptive Computing and Embedded Systems Lab. She is the cofounder and codirector of the UCSD Center for Machine-Integrated Computing and Security. Her research addresses aspects of efficient computing and embedded systems, with a focus on hardware and system security and real-time/energy-efficient and big data analytics under resource constraints, among others. She has received a number of awards and honors, including the Presidential Early Career Award for Scientists and Engineers from President Obama and the Cisco IoT Security Grand Challenge Award. She is an IEEE Fellow and a fellow of the Kavli Foundation Frontiers of the National Academy of Engineering.

SSC