Hyperparameter Optimization: BO with High-deimensional DNN

Jiarong Liang, Fiona Xu, Haoran Song, Yi Qu, Mingjian Zhao

[1]Washington University in St. Louis

One Brookings Dr

St. Louis, MO 63130

Abstract

In order to avoid finding the best hyperparameter for Bayesian optimization (BO) based on Gaussian processes (GPs), we propose one model including three layers of deep neural networks, which can dramatically decrease the cubic computing time, compared uncertained models like the random forest, linear regression... + results

1. Introduction

In recent years, there has been a growing interest in the development of efficient hyperparameter optimization algorithms for machine learning models. Various approaches have been proposed to address this issue, which can be broadly categorized into gradient-based methods, evolutionary algorithms, and Bayesian optimization methods.

Gradient-based methods, such as Hyperband (Li et al. 2017), use gradient information to optimize hyperparameters. While these methods can be efficient for certain tasks, they may not be suitable for optimizing non-differentiable hyperparameters or those with complex search spaces.

Evolutionary algorithms, such as Genetic Algorithms (GAs) (Such et al. 2017), and Particle Swarm Optimization (PSO) (Kennedy and Eberhart 2010), have also been applied to hyperparameter optimization. These methods generally perform well in terms of global search but may suffer from slower convergence rates due to the nature of the evolutionary process.

Bayesian optimization (BO) has emerged as a popular choice for hyperparameter optimization due to its ability to handle high-dimensional, non-convex, and non-differentiable search spaces. Gaussian Process-based BO, as used in Spearmint (Snoek, Larochelle, and Adams 2012) and SMAC (Hutter, Hoos, and Leyton-Brown 2011), has shown success in a wide range of applications. However, the scalability of GPs remains a challenge due to their cubic complexity with respect to the number of data points.

To address this limitation, researchers have explored alternative models for BO, such as using deep neural networks to approximate GPs, as in Deep Gaussian Processes (DGPs) (Damianou and Lawrence 2013), and Bayesian Neural Networks (BNNs) (Neal 2012). Despite these advances, maintaining well-calibrated uncertainty estimates remains a challenge.

In this work, we developed a high-dimensional deep neural network bayesian optimization model (HDBO) that maintains the well-calibrated uncertainty estimates of GPs for efficient hyperparameter optimization.

After presenting the background information in Section 2, we contribute the following: We develop a general framework for Bayesian Optimization (BO) using deep neural networks(Section 3). We introduce the details of this network and how we built it, and compare it with traditional models.(Section 4). Through extensive experiments, we demonstrate exceptional performance across HPOlib benchmark.

2. Related theory and practice

2.1. Bayesian Optimization

Bayesian Optimization (BO) is a well-known strategy to optimize objective functions (f) which are very expensive and slow to optimize (MacKay 1995). The main idea behind this approach is to limit the time cost in the evaluation of f by spending more time choosing the new set of hyperparameters (HPs) values. BO builds a surrogate model of the objective function, quantifies the uncertainty in the surrogate using a regression model (e.g., Gaussian Process Regression), and uses an acquisition function to decide where to sample the new set of HPs (Frazier 2018). The focus of BO is solving the problem:

$$x^* = \arg\min_x f(x), \tag{1}$$

where x^* represents the optimal set of hyperparameters that minimize the objective function f.

To achieve this, BO follows these key steps:

Define the surrogate model: A surrogate model is a probabilistic model that approximates the expensive objective function f. Gaussian Process Regression (GPR) is a popular choice for the surrogate model due to its ability to capture the uncertainty in the function's estimate. GPR assumes that the function values follow a multivariate Gaussian distribution, and it models the covariance between points using a kernel function. By learning the function's behavior with fewer evaluations, the surrogate model saves time and resources.

Choose the acquisition function: The acquisition function is responsible for guiding the search toward the optimal solution. It balances exploration (sampling in regions with high uncertainty) and exploitation (sampling where the surrogate model predicts the lowest function value). Common acquisition functions include Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB). The acquisition function is used to select the next query point x_{t+1}:

$$x_{t+1} = \arg\max_x \alpha(x), \tag{2}$$

where $\alpha(x)$ is the acquisition function and x_{t+1} is the new set of hyperparameters to be evaluated.

Update the surrogate model: After querying the true objective function at the new point x_{t+1}, the surrogate model is updated to include this new information. The updated model is then used to decide the next query point, and the process is repeated iteratively until a stopping criterion is met, such as a maximum number of iterations or reaching a convergence threshold. Bayesian optimization is particularly suited for optimizing functions with a high degree of noise, non-convexity, and expensive evaluations. It has been successfully applied to various optimization problems, including hyperparameter tuning for machine learning models, neural architecture search, and material discovery (Shahriari et al. 2016).

2.2. Deep Neural Network

Deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized the field of computer vision by offering the ability to automatically learn hierarchical feature representations from raw data in an end-to-end manner. CNNs have proven to be particularly effective in image recognition tasks, eliminating the need for manual feature engineering. However, designing high-performing CNN architectures requires significant expertise and effort, as the search space for possible architectures is enormous. This has led to the development of Neural Architecture Search (NAS) techniques as part of the Automated Machine Learning (AutoML) paradigm to aid researchers in designing more effective architectures for image recognition tasks.

In recent years, there has been a surge of research on NAS techniques (Liu et al. 2018). These approaches can be broadly divided into two main categories: those based on Reinforcement Learning (RL) (Zoph and Le 2016) and those based on Evolutionary Algorithms (EA) (Real et al. 2019). Both methodologies aim to automate the process of discovering optimal network architectures, but they differ in their optimization techniques.

RL-based NAS methods leverage the power of RL to explore and exploit the search space of possible architectures. By formulating the NAS problem as a Markov Decision Process, RL-based methods learn a policy to generate architectures with high performance on the task of interest. Despite their success, RL-based approaches can be computationally expensive and may require a large number of samples to converge to a good solution.

EA-based NAS methods, on the other hand, draw inspiration from the principles of natural selection and genetic variation to optimize network architectures. These methods iteratively evolve a population of candidate architectures by applying mutation and crossover operations, followed by selection based on performance. EA-based approaches have shown promising results in comparison to other NAS methods, including RL-based ones. However, they also suffer from the need to evaluate a large number of candidate networks, which can be computationally expensive and consume considerable GPU resources.

In summary, NAS techniques have emerged as a promising approach to automating the design of effective CNN architectures for image recognition tasks. While both RL-based and EA-based methods have shown success in discovering high-performing architectures, they each have their own challenges in terms of computational cost and search efficiency. Continued research in this area will be essential to further advance the state of the art and make image recognition accessible to a wider community of researchers and practitioners.

2.3. Deep Networks for Global Optimization(DNGO)

The objective of this research is to create a method that allows Bayesian optimization to scale effectively while preserving its valuable adaptability and uncertainty characterization. To achieve this, the authors suggest using neural networks to learn an adaptive set of basis functions for Bayesian linear regression. This method is referred to as Deep Networks for Global Optimization (DNGO). Unlike conventional Gaussian processes, DNGO's scalability is linear with respect to the number of function evaluations, which corresponds to the number of trained models in the context of hyperparameter optimization and is compatible with stochastic gradient training. Although it may appear that we are simply shifting the issue of setting the model's hyperparameters to setting them for the tuner, we demonstrate that with an appropriate set of design choices, it is feasible to develop a robust, scalable, and effective Bayesian optimization system that generalizes across numerous global optimization problems.

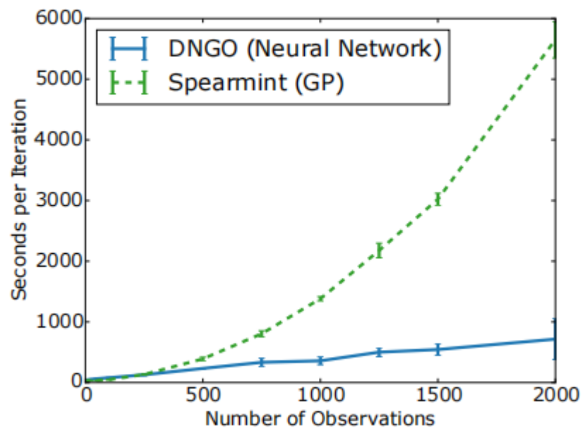

Figure 1: The speed difference between DNGO and GP

2. Technical details

3.1. TanH Smooth Function Applied in Network Layers

A significant drawback of GP-based Bayesian optimization is its cubic computational cost scaling with the number of observations, restricting its application to objectives that necessitate relatively few observations for optimization. In this study, our objective is to substitute the traditional GP used in Bayesian optimization with a model that scales more favorably, while preserving most of the GP's appealing attributes, such as adaptability and well-calibrated uncertainty. Bayesian neural networks naturally come to mind, particularly due to the theoretical connection between Gaussian processes and infinite Bayesian neural networks (Neal 1995) (Williams 1996). However, implementing these on a large scale is computationally demanding.

A common concern when using deep networks is that they often demand considerable effort to adapt and fine-tune for specific problems. Adjusting the architecture and tuning the neural network's hyperparameters can be seen as a complex hyperparameter optimization problem in itself. For instance, Figure 1 illustrates that the widely used rectified linear (ReLU) function can result in poor uncertainty estimates, leading the Bayesian optimization process to explore excessively. In DNGO, they utilize the bounded tanh function as it produces smooth functions with realistic variance. However, if the smoothness assumption needs to be relaxed, a combination of rectified linear functions with a tanh function only on the last layer can also be employed to bound the basis.

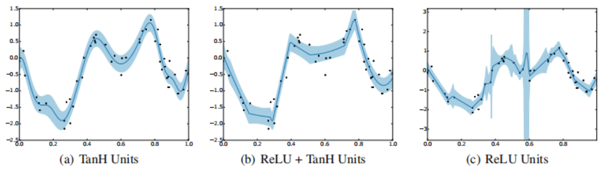

Figure 2: A comparison of the predictive mean and uncertainty learned by the model when using 2(a) only tanh, 2(c) only rectified linear (ReLU) activation functions, or 2(b) ReLU's but a tanh on the last hidden layer. The shaded regions correspond to standard deviation envelopes around the mean. The choice of activation function significantly modifies the basis functions learned by the model. Although the ReLU, which is the standard for deep neural networks, is highly flexible, we found that its unbounded activation can lead to extremely large uncertainty estimates.

To fine-tune the remaining hyperparameters, such as the width of the hidden layers and the amount of regularization, we employed GP-based Bayesian optimization. For each layer, from one to four, we ran Bayesian optimization using the Spearmint package (Snoek et al. 2014) to minimize the average relative loss on a series of benchmark global optimization problems. We adjusted the global learning rate, momentum, layer sizes, L2 normalization penalties for each set of weights, and dropout rates (Hinton et al. 2012) for each layer.

Interestingly, the optimal configuration included no dropout and only moderate L2 normalization. We speculate that dropout, despite having an approximate correction term, introduces noise in the predicted mean, leading to a loss of precision. The optimizer instead favored restricting capacity through a small number of hidden units. Specifically, the optimal architecture is a deep and narrow network with three hidden layers and around 50 hidden units per layer.

3.2. Robust stochastic gradient HMC via scale adaptation

Stochastic gradient Hamiltonian Monte Carlo (SGHMC) sampling methods (Hutter, Hoos, and Leyton-Brown 2011) can be used for estimating the posterior distribution of the neural network's weights. This approach helps to capture the uncertainties in predictions more effectively, making the method suitable for BO tasks. We first summarize the general formalism behind SGHMC(Chen, Fox, and Guestrin 2014) and then derive a more robust variant suitable for BO.

3.2.1. Stochastic gradient HMC HMC introduces a set of auxiliary variables, r, and then samples from the joint distribution

$$p(\theta, r|D) \propto \exp\left(-U(\theta) - \frac{1}{2}r^T M^{-1} r\right), \quad \text{with } U(\theta) = -\log p(D, \theta) \tag{3}$$

By simulating a fictitious physical system described by Hamilton's equations, the negative log-likelihood $U(\theta)$ acts as potential energy, r as the system's momentum, and M as the mass matrix. Introducing a user-defined friction matrix C, Chen et al. showed that Hamiltonian dynamics can be modified to sample from the correct distribution using a noisy estimate $\nabla\tilde{U}(\theta)$. Their discretized system of equations is given by

$$\Delta\theta = \epsilon M^{-1} r, \tag{4}$$

$$\Delta r = -\epsilon\nabla\tilde{U}(\theta) - \epsilon C M^{-1} r + N(0, 2(C - \hat{B})\epsilon), \tag{5}$$

where $N(0, \Sigma)$ represents a sample from a multivariate Gaussian with zero mean and covariance matrix Σ. This approach is particularly appealing for large models and datasets, and the distribution of (θ, r) guarantees that θ is distributed according to $p(\theta|D)$.

3.2.2. Scale adapted stochastic gradient HMC SGHMC has caveats, such as the correct setting of user-defined quantities like friction term C, gradient noise $\hat{B}$, mass matrix M, the number of MCMC steps, and step-size ϵ. These quantities can be highly model and dataset dependent, which is not suitable for BO that requires robust estimates across different functions F. Equation (5) shows the crucial impact of step-size on SGHMC robustness. To correct for unequal parameter scales, a pre-conditioner reflecting the metric underlying the model's parameters is desired. However, gSGRHMC requires the computation and storage of the full Fisher information matrix of U, which is prohibitively expensive.

A pragmatic approach involves a pre-conditioning scheme increasing SGHMC's robustness with respect to ϵ and C, while avoiding costly computations of gSGRHMC(Girolami and Calderhead 2011)(Ma, Chen, and Fox 2015). Recent adaptive pre-conditioning methods have been combined with stochastic gradient Langevin dynamics and HMC(Li et al. 2015)(Chen et al. 2016) sampling, but these approaches introduce additional hyperparameters or do not guarantee unbiased sampling.

4. Experiment and Discussions

4.1. HPOLib Benchmarks

HPOLib (Hyperparameter Optimization Library) Benchmarks is a suite of optimization problems that were specifically designed to evaluate hyperparameter optimization algorithms. It provides a collection of standardized benchmark problems, enabling a fair comparison of different hyperparameter optimization methods.

These benchmark problems often consist of supervised learning tasks with various machine learning models (e.g., support vector machines, random forests, and neural networks) and datasets from different domains. The goal of an optimization algorithm in the context of these benchmarks is to find the best set of hyperparameters for a given model, which would minimize a chosen objective function such as validation error or test error.

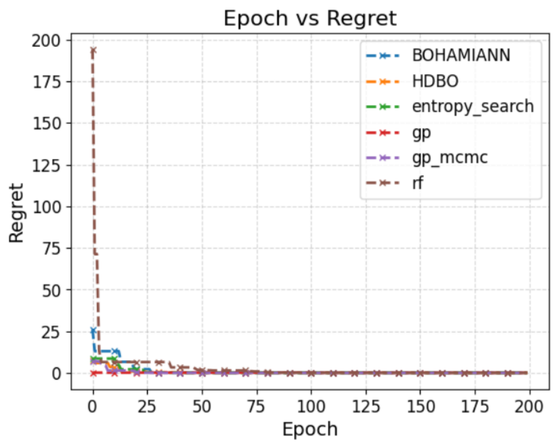

Figure 3: Regret of 6 algorithms changes with epoch

4.2. HDBO model

To showcase the efficacy of our algorithm, we conducted our algorithm HDBO a comparison with other optimization algorithms, and the input-warped Gaussian process technique proposed by (Snoek et al. 2014)using the benchmark set of continuous problems from the HPOLib package (Eggensperger 2013). As illustrated in Table 1, The HDBO algorithm converges faster than other methods. This indicates that HDBO maintains the statistical efficiency of the Gaussian process method concerning the number of evaluations needed to identify the minimum, even with substantial enhancements in scalability.

From the graph, it can be seen that the HDBO algorithm converges in just 25 epochs, and the algorithm significantly reduces the regret during initialization.

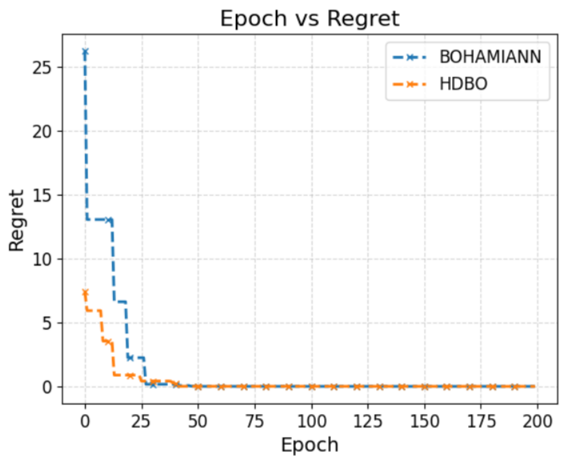

Figure 4: Regret comparison between BO and HDBO

Algorithm	**# hyperparams**	**contin./discr.**	**Dataset**	**Size(Train/Valid/Test)**	**runtime**
Branin	2(-)	2/-	-	-	$<$ 1s
Camelback function	2(-)	2/-	-	-	$<$ 1s
Hartmann 6d	6(-)	6/-	-	-	$<$ 1s
LDA ongrid	3(-)	-/3	wikipedia articles	-	$<$1s
SVM ongrid	3(-)	-/3	UniPROBE	-	$<$1s
Logistic Regression	4(-)	4/-	MNIST	50k/10k/10k	$<$1m
hp-nnet	14(4)	7/7	MRBI	10k/2k/50k	$\sim$25m
			convex	6.5k/1.5k/50k	$\sim$6m
hp-dbnet	36(27)	19/17	MRBI	10k/2k/50k	$\sim$15m
			convex	6.5k/1.5k/50k	$\sim$10m
autoweka	786(784)	296/490	convex	6.5k/1.5k/50k	$\sim$15m
Surrogate Benchmarks	as original	as original	as original	-	$<$1sec

Table 1: Available Benchmarks in HPOLib

Method	x_opt	f_opt
HDBO	(3.1356, 2.2572)	0.3986
BOHAMIANN	(3.1441, 2.2767)	0.3979
Entropy Search	(9.3943, 2.5101)	0.4060
GP	(9.3921, 2.3276)	0.4174
GP_MCMC	(-3.1419, 12.2748)	0.3979
RF	(-3.1316, 12.2283)	0.3989

Table 2: Optimal points and function values for different methods on branin function

We optimize various learning parameters, including learning rate, momentum, and batch size; regularization parameters such as dropout and weight decay for both word and image representations; as well as architectural parameters like context size, the choice between additive or multiplicative versions, word embedding size, and the size of the multi-modal representation. The final parameter, the number of factors, is only applicable to the multiplicative model, adding a unique challenge since it is relevant to only half of the hyperparameter space. This results in a total of 11 hyperparameters. Although this number may appear small, the problem presents several challenges that make optimization quite difficult. To maintain generality, we opt for broad box constraints for the hyperparameters, which consequently renders most of the model space infeasible. Furthermore, several hyperparameters are categorical, leading to significant non-stationarities in the objective surface.

Layer type	**in_feature**	**out_feature**
Linear	50	50
Linear	50	50
Linear	50	50
Tanh	50	50
MaxPooling		

Table 3: Network architecture, the above is one unit of the network, and the complete network requires repeating this unit three times.

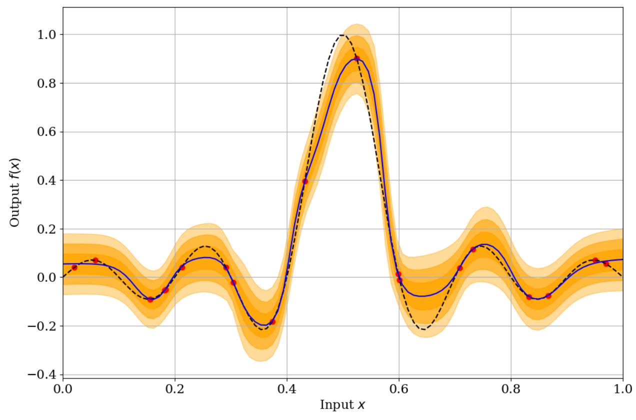

Figure 5: Fit of the sinOne function using HDBO

After increasing the number of layers in our network and changing its structure, our network has significantly improved compared to traditional Bayesian optimization in terms of both convergence speed in finding the global optimal solution and the accuracy of the results. Even comparing to the latest Bohamiann Bayesian optimization algorithm, there is a substantial improvement as well, with faster convergence speed and better initial point selection.

Our optimal models run on the benchmark HPOlib with P100 GPU Colab, each training session requires 200 epochs, taking approximately ten hours. The control experiment is conducted under the same conditions.

5. Conclusion

In conclusion, our study has successfully demonstrated the effectiveness of Hierarchical Deep Bayesian Optimization (HDBO) as a powerful optimization method. The research has shown that HDBO outperforms traditional Bayesian optimization techniques and Bohamiann algorithm in terms of convergence speed and initial point selection.

6. Contributions of team members

Jiangrong Liang:Designing and implementing the HDBO model, writing the code, and running the model along with a comparison experiment on the HPOlib benchmark. Writing the parts related theory and practice, technical details and experiment and discussions.

Fiona Xu: Mainly focusing on the theory part, writing the part on ”introduction & related theory and practice”, also doing part of the experiments.

Haoran Song: Gathering background related to Bayesian optimization, combining it with previous research, and conduct a literature review to clarify the direction of model improvement.

Yi Qu:

Mingjian Zhao: Writing the abstract and Combining the related work, write an introduction that provides a detailed background and objectives of the HDBO model.

References

Chen, C.; Carlson, D.; Gan, Z.; Li, C.; and Carin, L. 2016. Bridging the gap between stochastic gradient mcmc and stochastic optimization.

Chen, T.; Fox, E. B.; and Guestrin, C. 2014. Stochastic gradient hamiltonian monte carlo. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*.

Damianou, A. C., and Lawrence, N. D. 2013. Deep gaussian processes. In *International Conference on Artificial Intelligence and Statistics*.

Eggensperger, K. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters.

Frazier, P. I. 2018. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Girolami, M., and Calderhead, B. 2011. Riemann manifold langevin and hamiltonian monte carlo methods. In *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'11)*.

Kennedy, J., and Eberhart, R. 2010. *Particle swarm optimization*. Springer.

Li, C.; Chen, C.; Carlson, D.; and Carin, L. 2015. Preconditioned stochastic gradient langevin dynamics for deep neural networks.

Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *International Conference on Learning Representations*.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, 19–34.

Ma, Y.; Chen, T.; and Fox, E. 2015. A complete recipe for stochastic gradient mcmc. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*.

MacKay, D. J. 1995. Probable networks and plausible predictions-a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems* 6(3):469.

Neal, R. M. 1995. *Bayesian learning for neural networks*. Ph.D. Dissertation, University of Toronto.

Neal, R. M. 2012. Bayesian learning for neural networks. *Springer Science Business Media*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, 4780–4789.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and de Freitas, N. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.

Snoek, J.; Swersky, K.; Zemel, R. S.; and Adams, R. P. 2014. Input warping for bayesian optimization of non-stationary functions. In *Proceedings of the AAAI Conference on Artificial Intelligence (ICML)*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*.

Such, F. P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K. O.; and Clune, J. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Williams, C. K. I. 1996. Computing with infinite networks. In *Advances in Neural Information Processing Systems*.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.