Deep Uncertainty Estimation for Model-based Neural Architecture Search

Colin White
RealityEngines.AI
colin@realityengines.ai

Willie Neiswanger
Carnegie Mellon University
willie@cs.cmu.edu

Yash Savani
RealityEngines.AI
yash@realityengines.ai

1 Introduction

In this paper, we explore an application of deep uncertainty estimation for the task of model-based neural architecture search. We first develop a deep ensemble model to predict and provide uncertainty estimates for the performance of cell-based neural architectures. We then use the predictive uncertainty estimates from this model in a Bayesian optimization procedure for efficient neural architecture search. In this section, we give background on model-based optimization, neural architecture search, and uncertainty quantification in deep neural networks, and then summarize the main contributions of this work. [1]

Background. Neural architecture search (NAS) is the task of finding a neural network architecture that yields good performance (such as accuracy on a validation set) for a given dataset. A variety of strategies have been proposed for NAS, including random search, evolutionary search, reinforcement learning, gradient descent, and Gaussian process-based Bayesian optimization [26, 13, 18, 14, 9, 4, 8].

In this paper we focus on Bayesian optimization methods for NAS. Bayesian optimization (BO) is surrogate-model based technique for zeroth order optimization and search, which uses a Bayesian model to iteratively choose points to query [19, 5]. By leveraging this model to manage the tradeoff between exploration and exploitation, BO has the potential to perform efficient optimization using a minimal number of queries. Finding the next point to query at each iteration of BO involves optimizing an acquisition function, which is a function typically defined with respect to the posterior predictive distribution of a Bayesian model. Some common acquisition functions include the expected improvement (EI) [16], upper confidence bound (UCB) [21], and probability of improvement (PI) [10] acquisitions, as well as Thompson sampling (TS) [22].

In previous work, BO has been applied to NAS using a Gaussian process (GP) prior to provide predictive uncertainty estimates for architecture performance [9, 8]. To define the GP model, these strategies require developing an appropriate and accurate distance function between architectures, which may be difficult to define and costly to compute. In this paper, we instead explore and develop deep models for use in BO-based NAS, where we leverage recently developed methods to estimate the uncertainty in these deep models, and use these estimates to compute acquisition functions for use in BO.

Our contributions. In this paper, we present the following contributions.

- We develop a deep predictive uncertainty model for the performance of neural architectures. This involves a novel path encoding feature representation of neural architectures, and an ensemble-based model for uncertainty.
- We integrate our predictive uncertainty estimates into a Bayesian optimization procedure, which allows us to perform iterative acquisition optimization using a number of acquisition functions, including EI, UCB, PI, and Thompson sampling [22] (TS).

[1] This work was extended to a full-length paper here: `https://arxiv.org/abs/1910.11858`. All of the code for this work is available here: `https://github.com/naszilla/bananas`.

4th workshop on Bayesian Deep Learning (NeurIPS 2019), Vancouver, Canada.

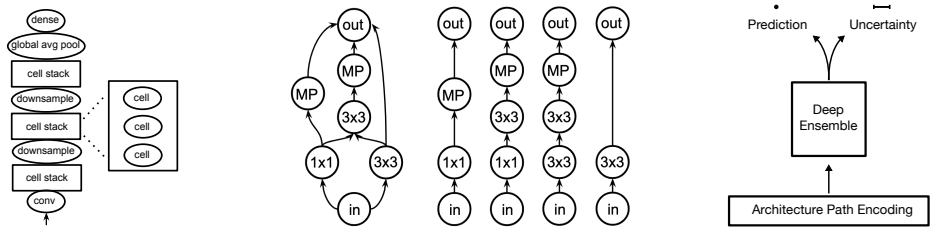

Figure 1: Hyper-architcture of a NASBench neural network (left). Example of a path encoding of a neural network (middle). A diagram of our deep predictive uncertainty model (right).

- We compare the performance of our method against a host of benchmark NAS algorithms including random search, regularized evolution, and reinforcement learning. We show significant improvements against these benchmarks on the NASBench dataset [25].

2 Methods

We first provide a formalism for neural architectures and the search space, and then describe a feature representation for architectures, which is used in a deep predictive model. After, we describe ensemble-based uncertainty estimates for this predictive model, show the use of these estimates in computing approximate acquisition functions, and finally use these acquisition estimates in a Bayesian optimization procedure for NAS.

Architecture formalism and search space. In this work, we consider convolutional cell-based search spaces [26, 18, 14]. A *cell* consists of a relatively small section of a neural network, usually 6-12 nodes forming a directed acyclic graph (DAG). A neural architecture is then built by repeatedly stacking one or two different cells on top of each other sequentially, separated by downsampling layers. The layout of cells and downsampling layers is called a *hyper-architecture*, and this is fixed, while the NAS algorithm searches for the best cells. The search space over cells consists of all possible DAGs of size 6-12, where each node can be one of several operations. It is also common to set a restriction on the number of total edges or the in-degree of each node [25, 14]. In this work, we focus on searching over a cell of size 7 with up to 10 edges, where the middle five nodes can be set to **conv1x1**, **conv3x3**, or **max-pool3x3**, consistent with the NASBench [25] search space (Fig. 1 (left)).

Architecture featurization via a path encoding. Prior works aiming to encode or featurize neural networks have proposed using a binary encoding of the adjacency matrix and either a categorical or a one-hot encoding for the operations on each node [23, 25, 3, 1]. It is challenging even for a deep network to learn graph topologies from an adjacency encoding, and we show in the next section that this featurization does not perform well. We introduce a path-based encoding and show it substantially increases the performance of our deep predictive uncertainty model. A path encoding of a cell is created by enumerating all possible paths from the input node to the output node, in terms of the operations (see Figure 1). The total number of paths is $\sum_{i=0}^{n} q^i$ where n denotes the number of nodes in the cell, and q denotes the number of operations for each node. For example, the NASBench [25] dataset has $\sum_{i=0}^{5} 3^i = 364$ possible paths.

Predictive model of accuracy and uncertainty estimates. Using our path-based featurization of neural architectures, we develop a deep predictive model with uncertainty estimates, which will be used downstream in a Bayesian optimization procedure. In our model, we define a collection of neural networks that each map from the path encoding of an architecture to its accuracy on a validation dataset. We train each network according to a proper scoring rule [7] to produce an ensemble of predictors, using a strategy similar to some previous work [6, 11] for generating uncertainty estimates. We visualize this model in Fig. 1 (right). We plot predictions and uncertainty estimates given by this model, and compare them with true values, in Fig. 2, where as a simple sanity check we compare the predictive uncertainty on training vs held-out test architectures. We next describe how we use this collection of deep predictors for uncertainty estimation in Bayesian optimization.

Acquisition estimation and Bayesian optimization. We take our predictive uncertainty estimate to be a proxy for the posterior predictive distribution of a Bayesian model. In practice, we treat the predictive given by each network as a sample from the posterior predictive distribution of a Bayesian model, and use these samples to approximately compute a few common acquisition functions [17].

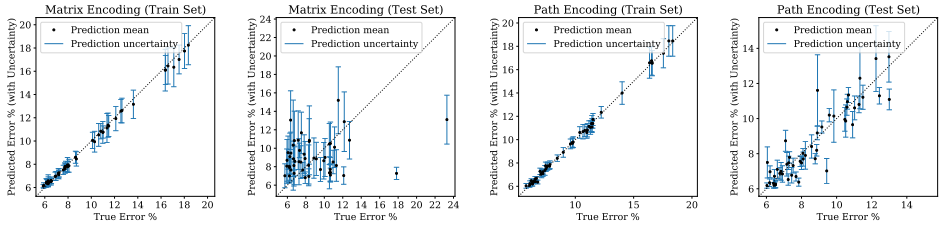

Figure 2: Predictive uncertainty estimates for architecture validation error under our model.

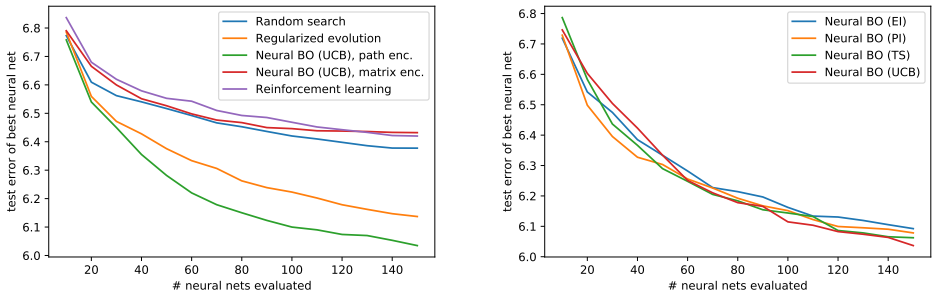

Figure 3: Comparison of NAS methods on the NASBench dataset (left), and a comparison of different acquisition functions for our neural BO method (right).

For example, we can compute the expected improvement (EI) and probability of improvement (PI) acquisition functions via sample-based approximate expectations with respect to the posterior predictive, and we can compute the upper-confidence bound (UCB) acquisition function by estimating a lower confidence bound on the posterior predictive distribution. Given an acquisition function estimate, we can carry out the following Bayesian optimization procedure: iteratively optimize the approximate acquisition functions to choose new points to query (i.e. new architectures to validate), and then retrain our deep predictive model on the new queried architectures. We optimize our approximate acquisition functions using an evolutionary algorithm. The acquisition estimates and full BO procedure are described in detail in the Appendix Sec. A and B.

3 Empirical Results

We show results on the performance of our deep predictive uncertainty model, as well as the full neural BO procedure compared to several baselines. We use the NASBench [25] dataset, in which there are approximately 423,000 architectures after removing isomorphisms, and the best neural network in this dataset achieves 5.68% test error on CIFAR-10.

Performance of Deep Model with Predictive Uncertainty. We evaluate the performance of our deep model with predictive uncertainty. This model consists of an ensemble of dense fully-connected neural network, with 10 layers, where each layer has width 20. We use the Adam optimizer with a learning rate of 0.1. These hyperparameters were chosen through random hyperparameter search. We trained our deep model on 200 neural networks, and tested the standard adjacency matrix encoding as well as the path encoding (Sec. 2) on a held-out set of 50 architectures as well as a subset of the training set. We plot this in Fig. 2. When using our path encoding representation, we see a low uncertainty for architectures in the training set, and a higher uncertainty for held-out test points, where incorrectly predicted points tend to have higher uncertainty. The path-based encoding outperforms the adjacency matrix encoding by a factor of 3 in mean absolute error.

NAS Results. We compare our neural BO algorithm against random search, regularized evolution, and reinforcement learning (see Appendix C for a description of each baseline). Each NAS algorithm is given a budget of 150 queries. That is, each algorithm can train and query the validation accuracy of at most 150 architectures. Every 10 iterations, we return the architecture with the best validation error found by the algorithm so far. After all NAS algorithms complete, we return the test error for each returned architecture. We repeated this process for 200 trials for each algorithm.

Our BO procedure with path encoding significantly outperforms all other baselines, and is state-of-the-art on NASBench for the 100-150 queries setting [25, 2, 15] (Fig. 3 (left)). The standard deviation among 200 trials for all NAS algorithms were between 0.16 and 0.22. Our procedure had the lowest standard deviation, and regularized evolution had the highest standard deviation. We also compared different acquisition functions (Fig. 3 (right)).

References

[1] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[2] Zalán Borsos, Andrey Khorlin, and Andrea Gesmundo. Transfer nas: Knowledge transfer between search spaces with transformer agents. *arXiv preprint arXiv:1906.08102*, 2019.

[3] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

[4] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[5] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[7] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[8] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

[9] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

[10] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[11] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[12] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[13] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[14] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[15] Krzysztof Maziarz, Andrey Khorlin, Quentin de Laroussilhe, and Andrea Gesmundo. Evolutionary-neural hybrid agents for architecture search. *arXiv preprint arXiv:1811.09828*, 2018.

[16] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[17] Willie Neiswanger, Kirthevasan Kandasamy, Barnabas Poczos, Jeff Schneider, and Eric Xing. Probo: a framework for using probabilistic programming in bayesian optimization. *arXiv preprint arXiv:1901.11515*, 2019.

[18] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[19] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[20] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[21] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[22] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[23] Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

[24] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, pages 229–256, 1992.

[25] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.

[26] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

A Full Bayesian Optimization Procedure

Bayesian optimization is a popular technique for query-efficient zeroth order optimization, such as in the task of hyperparameter tuning of machine learning models. In Bayesian optimization for deep learning, the goal is to find the neural network and/or set of hyperparameters in the search space which lead to the best validation accuracy. Formally, Bayesian optimization seeks to optimize a function $\max_{a \in A} f(a)$, where A is the set of neural architectures and/or hyperparameters, and $f(a)$ denotes the validation accuracy of architecture a after training a fixed dataset for a fixed number of epochs.

In a typical Bayesian optimization setting, t_0 architectures are drawn at random from the search space and trained, forming an initial dataset. Then the objective $f(A)$ is modeled via the posterior distribution of a chosen Bayesian model. Typically a Gausssian process posterior is used, while in this work we use an ensemble of neural networks to define a posterior. An acquisition function, defined with respect to this posterior, is used to choose the next neural network to query. We optimize the acquisition function using a evolutionary algorithm. See Algorithm 1.

Algorithm 1 Neural BO

Input: Search space A, dataset D, parameters $t_0,\ T,\ c,\ e$, acquisition function ϕ
Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from a and train them on D
Denote $f(a)$ as the validation accuracy of a after training
For t from t_0 to T,

1. Generate a set of c candidate neural architectures from A by drawing $c/2$ architectures at random, and creating $c/2$ mutations of the k architectures a from $\{a_0, \ldots, a_t\}$ with the highest value of $f(a)$
2. Train an ensemble of deep networks on $f(a_0), \ldots, f(a_t)$
3. For each candidate architecture a, evaluate the acquisition function $\phi(a)$
4. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$ and evaluate $f(a_{t+1})$

Output: $a^* = \text{argmin}_{a_i} f(a_i)$ and the test accuracy of a^*

B Acquisition Function Estimates

Suppose we have trained a collection of M predictive models, $\{f_m\}_{m=1}^M$, where $f_m : \mathcal{X} \to \mathbb{R}$. Following previous work [17], we use the following acquisition function estimates for an input architecture $x \in \mathcal{X}$:

$$\hat{a}_{\text{EI}}(x) = \mathbb{E}\left[\mathbb{1}\left[f_m(x) > y_{\text{min}}\right](y_{\text{min}} - f_m(x))\right] = \int_{-\infty}^{y_{\text{min}}} (y_{\text{min}} - y)\mathcal{N}\left(\hat{\mu}, \hat{\sigma}^2\right) dy \tag{1}$$

$$\hat{a}_{\text{PI}}(x) = \mathbb{E}\left[\mathbb{1}\left[f_m(x) > y_{\text{min}}\right]\right] = \int_{-\infty}^{y_{\text{min}}} \mathcal{N}\left(\hat{\mu}, \hat{\sigma}^2\right) dy \tag{2}$$

$$\hat{a}_{\text{UCB}}(x) = \widehat{\text{LCB}}\left(\{f_m(x)\}_{m=1}^M\right) = \hat{\mu} - \beta\hat{\sigma} \tag{3}$$

$$\hat{a}_{\text{TS}}(x) = f_{\tilde{m}}(x), \quad \tilde{m} \sim \text{Unif}\,(1, M) \tag{4}$$

In these acquisition function definitions, $\mathbb{1}(x) = 0$ if x is true and 0 otherwise, and we are making a normal approximation for our model's posterior predictive density, where we estimate parameters $\hat{\mu} = \frac{1}{M}\sum_{m=1}^M f_m(x)$, and $\hat{\sigma} = \sqrt{\frac{\sum_{m=1}^M (f_m(x) - \hat{\mu})^2}{M-1}}$. In the UCB acquisition function, we use $\widehat{\text{LCB}}$ to denote some estimate of a lower confidence bound for the posterior predictive density (note that we use a lower confidence bound rather than an upper confidence bound since we are performing minimization), and β is a tradeoff parameter. In experiments we set $\beta = 0.5$.

C NAS Experimental Details.

We give a brief description of each of the baselines we show in our empirical results in Sec. 3.

Random search (RS). The simplest baseline, random search draws n architectures at random and outputs the architecture with the highest validation accuracy. Despite its simplicity, multiple papers have concluded that random search is a competitive baseline for NAS algorithms [12, 20].

Regularized evolution(RE). We used the nasbench implementation of regularized evolution [25]. The algorithm consists of drawing an initial set of cells at random, and then iteratively mutating the best architecture out of a sample of all architectures evaluated so far. The architectures with the worst validation accuracy in each iteration are removed from the population.

Reinforcement Learning (RE). We used the nasbench implementation of reinforcement learning [25], based on the REINFORCE algorithm [24]. We chose this algorithm because prior nasbench experiments have shown that a 1-layer LSTM controller trained with PPO is not effective on the NASBench dataset [25].

Validation accuracy queries In the nasbench dataset, each architecture was trained to 108 epochs three separate times with different random seeds. The nasbench paper conducted experiments by querying a random validation error, when querying each architecture, and then reporting the mean test error at the conclusion of the NAS algorithm. We found that the mismatch (choosing random validation error, but mean test error) added extra noise that would not be present during a real-life NAS experiment. Therefore, we used mean validation error and mean test error for the results presented in Section 3.