NAS-Bench-x11 and the Power of Learning Curves

Shen Yan *
Michigan State University
yanshen6@msu.edu

Colin White *
Abacus.AI
colin@abacus.ai

Yash Savani
Carnegie Mellon University
ysavani@cs.cmu.edu

Frank Hutter
University of Freiburg and Bosch Center for Artificial Intelligence
fh@cs.uni-freiburg.de

1. Introduction

In the past few years, algorithms for neural architecture search (NAS) have been used to automatically find architectures that achieve state-of-the-art performance on various datasets. In 2019, there were calls for reproducible and fair comparisons within NAS research [15, 16] due to both the lack of a consistent training pipeline between papers and experiments with not enough trials to reach statistically significant conclusions. These concerns spurred the release of tabular benchmarks, such as NAS-Bench-101 [25] and NAS-Bench-201 [4], created by fully training all non-isomorphic architectures in search spaces of size 423 624 and 6 466, respectively. These benchmarks allow researchers to easily simulate NAS experiments, making it possible to quickly run fair NAS comparisons and to run enough trials to reach statistical significance at very little computational cost [9]. Recently, to extend the benefits of tabular NAS benchmarks to larger, more realistic NAS search spaces which cannot be evaluated exhaustively, it was proposed to construct *surrogate NAS benchmarks*. The first such surrogate benchmark is NAS-Bench-301 [21], which was created by training 60 000 architectures from the DARTS [17] search space, and then fitting a surrogate model which can be used to estimate the performance of all 10^{18} architectures in the DARTS search space. Since 2019, dozens of papers have used these NAS benchmarks to develop new algorithms.

A downside of these benchmarks is that the main type of algorithms that can be benchmarked are *single fidelity* algorithms: when the NAS algorithm chooses to evaluate an architecture, the architecture is fully trained and only the final validation accuracy is outputted. This is because NAS-Bench-301 only contains the architectures‘ accuracy at epoch 100, and NAS-Bench-101 only contains the accuracies at epochs 4, 12, 36, and 108 (allowing single-fidelity or very constrained multi-fidelity approaches). NAS-Bench-201 does allow queries on the entire learning curve, but it is smaller in size (6 466) than NAS-Bench-101 (423 624) or NAS-Bench-301 (10^{18}). In particular, learning curve extrapolation (LCE) [3, 1] and other multi-fidelity techniques [14, 7] have been under-utilized by the NAS community in recent years.

In this work, we fill in this gap by releasing NAS-Bench-111, NAS-Bench-311, and NAS-Bench-NLP11, surrogate benchmarks with full learning curve information for train/validation/test loss and accuracy for all architectures, significantly extending NAS-Bench-101, NAS-Bench-301, and NAS-Bench-NLP [13], respectively. With these benchmarks, researchers can easily incorporate multi-fidelity techniques, such as early stopping and LCE into their NAS algorithms. Our technique for creating these benchmarks can be summarized as follows. We use a training dataset of architectures (drawn randomly and chosen by top NAS methods) with good coverage over the search space, along with full learning curves, to fit a model that predicts the full learning curves of the remaining architectures. We employ three techniques to fit the model: *(1)* dimensionality reduction of the learning curves, *(2)* prediction of the top singular value coefficients, and *(3)* noise modeling. These techniques can be used in the future to create new NAS benchmarks as well. To ensure that our surrogate benchmarks are highly accurate, we statistically compare the difference between ground truth learning curves and predicted learning curves on separate test sets (see Figure 1).

To demonstrate the power of using the full learning curve information, we present a framework for converting single-fidelity NAS algorithms into multi-fidelity algorithms using LCE. We apply our framework to popular single-fidelity NAS algorithms which claimed state-of-the-art upon release, showing that they can be further improved. Overall, our work

*Equal contribution. This work was done while SY was an intern at Abacus.AI and while YS was employed at Abacus.AI. FH acknowledges support by the European Research Council (ERC) under the European Union Horizon 2020 research and innovation programme through grant no. 716721, and by BMBF grant DeToL.

bridges the gap between different areas of AutoML and will allow researchers to easily develop effective multi-fidelity and LCE techniques in the future.

2. Related Work

NAS has been studied since at least the late 1980s [18] and has recently seen a resurgence [27]. Weight sharing algorithms have become popular due to their computational efficiency [17]. Recent advances in performance prediction and other iterative techniques [7, 24] have reduced the runtime gap between iterative and weight sharing techniques. For a survey on NAS, see [6].

NAS-Bench-101 [25], a tabular NAS benchmark, is a search space of size 423 624 with accuracies recorded for epochs 4, 12, 36, and 108. The accuracies for the other epochs are not recorded. NAS-Bench-1shot1 [26] defines a subset of the NAS-Bench-101 search space that allows one-shot algorithms to be run. NAS-Bench-201 [4] contains 6 466 architectures unique up to isomorphisms. It comes with full learning curve information on three datasets: CIFAR-10, CIFAR-100, and ImageNet16-120. The DARTS search space [17] consists of 10^{18} neural architectures, making it computationally prohibitive to create a tabular benchmark. To overcome this fundamental limitation, NAS-Bench-301 [21] evaluates various regression models trained on a sample of 60 000 architectures that is carefully created to cover the whole search space. The surrogate models allow users to query the validation accuracy (at epoch 100) for any of the 10^{18} architectures in the DARTS search space. NAS-Bench-NLP [13] is a search space for language modeling tasks. The search space consists of 10^{53} LSTM-like architectures, of which 14 000 are evaluated on Penn Tree Bank, containing the accuracies from epochs 1 to 50. Since only 14 000 architectures can be queried, this dataset cannot be directly used for NAS experiments.

3. Generating Learning Curves

Given a search space $\mathcal{D}$, let $(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim \mathcal{D}$ denote one datapoint, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the architecture encoding (e.g., one-hot adjacency matrix), and $\boldsymbol{y}_i \in [0,1]^{E_{\text{max}}}$ is a learning curve of validation accuracies drawn from a distribution based on training the architecture for E_{max} epochs on a fixed training pipeline with a random initial seed. Each learning curve $\boldsymbol{y}_i$ can be decomposed into two parts: one part that is deterministic and depends only on the architecture encoding, and another part that is based on the inherent noise in the architecture training pipeline. Formally, $\boldsymbol{y}_i = \mathbb{E}[Y(\boldsymbol{x}_i)] + \boldsymbol{\epsilon}_i$, where $\mathbb{E}[Y(\boldsymbol{x}_i)] \in [0,1]^{E_{\text{max}}}$ is fixed and depends only on $\boldsymbol{x}_i$, and $\boldsymbol{\epsilon}_i \in [0,1]^{E_{\text{max}}}$ comes from a noise distribution Z_i with expectation 0 for all epochs. In practice, $\mathbb{E}[Y(\boldsymbol{x}_i)]$ can be estimated by averaging a large set of learning curves produced by training architecture $\boldsymbol{x}_i$ with different initial seeds. We represent such an estimate as $\bar{\boldsymbol{y}}_i$.

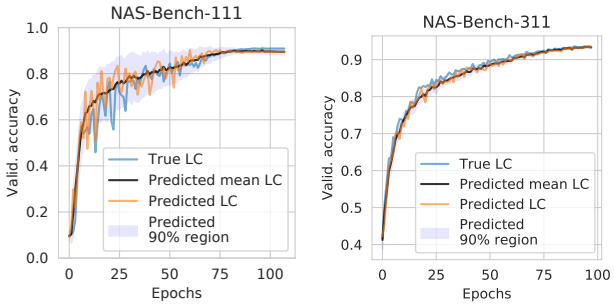

Figure 1: Real vs. predicted learning curves.

Our goal is to create a surrogate model that takes as input any architecture encoding $\boldsymbol{x}_i$ and outputs a distribution of learning curves that mimics the ground truth distribution. We assume that we are given two datasets, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, of architecture and learning curve pairs. We use $\mathcal{D}_{\text{train}}$ to train the surrogate, and we use $\mathcal{D}_{\text{test}}$ for evaluation. In order to predict a learning curve distribution for each architecture, we split up our approach into two separate processes: we train a model $f : \mathbb{R}^d \to [0,1]^{E_{\text{max}}}$ to predict the deterministic part of the learning curve, $\bar{\boldsymbol{y}}_i$, and we train a noise model $p_\phi(\boldsymbol{\epsilon} \mid \bar{\boldsymbol{y}}, \boldsymbol{x})$, parameterized by ϕ, to simulate the random draws from Z_i.

Training a model f to predict mean learning curves is a challenging task, since the training datapoints $\mathcal{D}_{\text{train}}$ consist only of a single (or few) noisy learning curve(s) $\boldsymbol{y}_i$ for each $\boldsymbol{x}_i$. Furthermore, E_{max} is typically length 100 or larger. We propose a technique to help with both of these challenges: we use the training data to learn compression and decompression functions $c_k : [0,1]^{E_{\text{max}}} \to [0,1]^k$ and $d_k : [0,1]^k \to [0,1]^{E_{\text{max}}}$, respectively, for $k \ll E_{\text{max}}$. The surrogate is trained to predict *compressed* learning curves $c_k(\boldsymbol{y}_i)$ of size k from the corresponding architecture encoding $\boldsymbol{x}_i$, and then each prediction can be reconstructed to a full learning curve using d_k. For a good compression model, $(d_k \circ c_k)(\boldsymbol{y}_i)$ should be a less noisy version of $\boldsymbol{y}_i$. Therefore, models trained on $c_k(\boldsymbol{y}_i)$ have better generalization ability.

We test two compression techniques: singular value decomposition (SVD) [8] and variational autoencoders (VAEs) [12]. Next, we train a surrogate model with $\boldsymbol{x}_i$ as features and $c_k(\boldsymbol{y}_i)$ as the label, for architectures in $\mathcal{D}_{\text{train}}$. We test LGBoost [10], XGBoost [2], and MLPs for the surrogate model. Finally, we add a noise model so that the outputs are realistic, *noisy* learning curves. Since the training data only contains one (or few) learning curve(s) per architecture $\boldsymbol{x}_i$, it is not possible to accurately estimate the noise distribution for each architecture without making further assumptions. We test three noise modeling techniques which make use of different assumptions: *(i)* a simple sample standard deviation statistic, *(ii)* a Gaussian kernel density estimation (GKDE)

model [20], and *(iii)* a sample standard deviation statistic based on sliding windows.

Overall, we test two compression methods, three surrogate models, and three noise models. For each approach, we evaluate both the predicted mean learning curves and the predicted noisy learning curves using held-out test sets $\mathcal{D}_{\text{test}}$. To evaluate the mean learning curves, we measure the Kendall Tau (KT) rank correlation [11], both at the final epoch and averaged over all epochs. To evaluate the noisy learning curves, we measure the Kullback Leibler (KL) divergence between the ground truth distribution and the predicted distribution.

Surrogate Benchmark Creation. Now we describe the creation of the NAS benchmarks using the above techniques. For all search spaces, SVD-LGB-GKDE performed the best. First, we describe NAS-Bench-111. Since the NAS-Bench-101 tabular benchmark [25] consists of accuracies for epochs at 4, 12, 36, and 108 without the other accuracies, we train a new set of architectures and save the full learning curves. Inspired by prior work [5, 21], we sample a set of architectures with good overall coverage by sampling 861 architectures generated uniformly at random, 149 architectures discovered by BANANAS [22], local search [23], and regularized evolution [19], and all 91 architectures which contained less than five nodes. We keep our training pipeline as close as possible to the original pipeline. Because a tabular benchmark already exists, we can substantially improve the accuracy of our surrogate by utilizing the information from the tabular benchmark (at epochs 4, 12, 36, 108) as additional features. We achieve average and final epoch KT values of 0.611 and 0.794, respectively, and a KL divergence of 1.710.

Next, we create NAS-Bench-311 by using the training data from NAS-Bench-301, which consists of $40\,000$ random architectures along with $26\,000$ additional architectures discovered by different NAS algorithms [17, 22, 19]. We achieve average and final epoch KT values of 0.728 and 0.788, respectively, and a KL divergence of 0.905.

Finally, we create NAS-Bench-NLP11 by using the NAS-Bench-NLP dataset consisting of $14\,000$ architectures drawn uniformly at random. Due to the extreme size of the search space (10^{53}), we restrict architectures to a maximum of 12 nodes (reducing the size to 10^{22}). To create a stronger surrogate, we add the first three epochs of the learning curve as features in the surrogate. We achieve average and final epoch KT values of 0.878 and 0.844, respectively. Since there are no architectures trained multiple times on the NAS-Bench-NLP dataset [13], we cannot compute the KL divergence.

4. Experiments

A Framework for Learning Curve Extrapolation We describe a simple, novel framework for converting single-fidelity NAS algorithms to multi-fidelity NAS algorithms using learning curve extrapolation techniques, which is able to substantially improve the performance of popular algorithms. A single-fidelity algorithm is an algorithm which iteratively chooses an architecture based on its history, which is then fully trained to E_{max} epochs. Many single-fidelity algorithms iteratively output several architectures at a time, instead of just one. Our framework makes use of learning curve extrapolation (LCE) techniques to predict the final validation accuracies of all architecture choices after only training for a small number of epochs. The architectures predicted to be in the top percentage of validation accuracies are then fully trained. This simple modification can substantially improve the runtime efficiency of popular NAS algorithms. Any LCE technique can be used, and in our experiments, we use LcSVR [1], which trains a support vector regressor to extrapolate the learning curve.

NAS experiments. For single-fidelity algorithms, we implemented random search (RS) [15], local search (LS) [23], regularized evolution (REA) [19], and BANANAS [22]. For multi-fidelity algorithms, we implemented Hyperband [14], and Bayesian optimization Hyperband (BOHB) [7]. For all methods, we used the original implementation whenever possible. We used our LCE framework to create BANANAS-SVR, LS-SVR, and REA-SVR. For each search space, we run the equivalent runtime of training 500 architectures. We run 30 trials of each NAS algorithm. See Figure 2. We see that SVR always improves over the corresponding single-fidelity algorithm, and SVR-based methods achieve top performance in all search spaces.

5. Conclusions and Guidelines

In this work, we released three benchmarks for neural architecture search based on three popular search spaces, which substantially improve the capability of existing benchmarks due to the availability of the full learning curve for train/validation/test loss and accuracy for each architecture. Our techniques to generate these benchmarks can be used to model the full learning curve for future surrogate NAS benchmarks. Furthermore, we demonstrated the power of the full learning curve information by introducing a framework that converts single-fidelity NAS algorithms into multi-fidelity NAS algorithms that make use of learning curve extrapolation techniques. While we believe our surrogate benchmarks will help advance scientific research in NAS, a few guidelines are important to keep in mind. We discourage evaluating NAS methods that use the same internal techniques as those used in the surrogate model. As the surrogate benchmarks are likely to evolve as new training data is added, we recommend reporting the surrogate benchmark version number whenever running experiments.

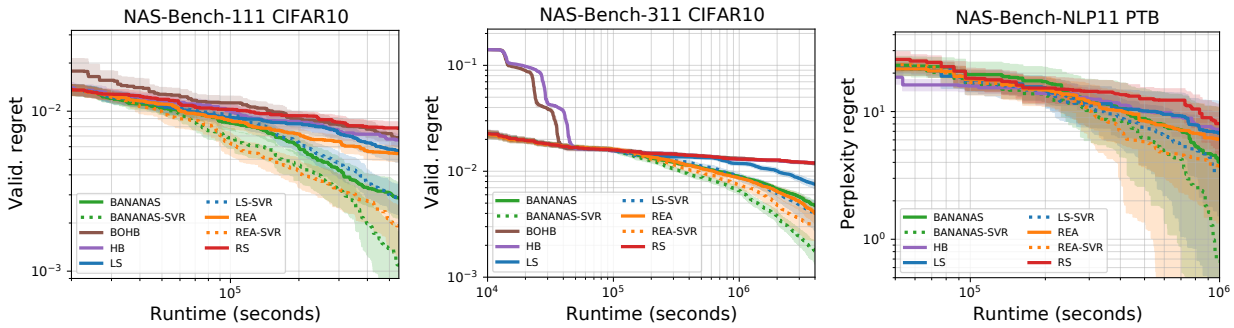

Figure 2: NAS results on three search spaces. For every setting, an SVR augmented method performs best.

References

[1] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *ICLR Workshop*, 2018. 1, 3

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 2

[3] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015. 1

[4] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020. 1, 2

[5] Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, 2015. 3

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. In *JMLR*, 2019. 2

[7] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*, 2018. 1, 2, 3

[8] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Numerical Analysis*, 1965. 2

[9] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*, 2019. 1

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017. 2

[11] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. 3

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 2

[13] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *arXiv preprint arXiv:2006.07116*, 2020. 1, 2, 3

[14] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *JMLR*, 2018. 1, 3

[15] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019. 1, 3

[16] Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. In *JMLR*, 2020. 1

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 1, 2, 3

[18] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989. 2

[19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 3

[20] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015. 3

[21] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020. 1, 2, 3

[22] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *AAAI*, 2021. 3

[23] Colin White, Sam Nolen, and Yash Savani. Local search is state of the art for nas benchmarks. In *UAI*, 2021. 3

[24] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv:2104.01177*, 2021. 2

[25] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019. 1, 2, 3

[26] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *ICLR*, 2020. 2

[27] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2