Evaluating supernets for neural architecture search

Anonymous[1]

[1]Anonymous Institution

Abstract In order to address the costly process of evaluating candidate architectures in neural architecture search (NAS), ENAS introduced supernets, a single network representing the entire search space. DARTS then added a set of architecture weights to this structure, allowing an entirely gradient-descent-based approach to NAS and leading to supernets becoming a common feature of efficient NAS algorithms. Although following research has demonstrated numerous improved techniques for training supernets, these methods are typically compared solely through the performances of the final architectures selected from the supernets without any direct comparison of the supernets themselves. We provide methods which exploit the availability of NAS benchmark data sets to provide a robust direct evaluation of trained supernets, both of the weights shared across candidate architectures as well as architecture weights. We also use these methods to analyze the process of NAS training and provide insight on how the top supernet NAS algorithms are able to effectively exploit the supernet structure.

1 Introduction

Neural architecture search (NAS) provides a data-driven algorithmic approaches to the complex design problem of building a neural network. However, the initial approach of evaluating candidate architectures by training each one resulted in search algorithms requiring 1000s of GPU-hours for a single run (Zoph et al., 2018).

Pham et al. (2018) provided a method to search for an architecture in a comparable amount of time as required to train the final model by training a single supernetwork instead of individual candidate networks. The supernet is a single model representing the entire architecture search space, essentially a superposition of every allowed candidate model. Then Liu et al. (2019) took advantage of this structure sharing weights across all candidates by adding trainable architecture weights to each possible operation in the supernet, allowing for search and training to be conducted via gradient descent. These innovations combined to offer a practical and promising approach to NAS.

However, while the marriage of gradient-based NAS with supernets may simplify the implementation of NAS algorithms, it also serves to obscure the search process. While NAS algorithms can generally be described as an iterative process of search over candidate architectures and obtaining estimates of their performance (Elsken et al., 2019), in DARTS this only occurs implicitly. DARTS does not actually evaluate individual candidate architectures at any point in training, instead training over a "continuous relaxation" of possible architectures Liu et al. (2019). This collapse of the NAS training loop to gradient-based training of two sets of interdependent weights makes it more difficult to ascertain why, in various contexts, some NAS algorithms are highly effective and others fail.

We address these limitations through the direct evaluation of trained supernets using NAS benchmark datasets. We propose two separate general-purpose methods focused on separate components of the supernet. One method is focused on evaluating the shared weights of the supernet, determining how suitable they are for the various architectures in the search space which share them. The second method evaluates the architecture weights (or architecture distribution)

Submitted to AutoML 2023

learned by DARTS-like algorithms during the process of training the supernet. Both methods make use of NAS benchmarks to analyze how the learned weights vary across the architecture search space in comparison to trained-from-scratch test accuracy. We implement these methods in the NASLib library with the intention of making them widely available as evaluation tools.

The methods we demonstrate provide additional insight into the algorithms evaluated, revealing not only differences in performance obfuscated by similarities in the final test accuracy, but also insight into how these algorithms are able to successfully exploit the structure of the supernet. We are able to demonstrate a greater capacity for supernets to identify top performing architectures than previously described (White et al., 2021).

The primary contributions of this work are thus to demonstrate and highlight that:

- That algorithms that succeed in supernet training do so by specializing to maximize performance estimation on (only) the top architectures in the search space (Sec. 2)
- This specialization of the supernet is enabled by the structure of the search space (Sec. 3)

2 Evaluating supernets

Prior work on issues with NAS evaluation has produced a set of best practices to support fair and reliable comparison between NAS methods, including evaluating in the same search space as comparable methods, using the same routine to train the final selected architecture, reporting results of multiple random seeds, and performing ablation studies (Yu et al., 2020) Even following all of these practices and, even better, evaluating your algorithm using a NAS benchmark data set, we argue that reporting the final train-from-scratch test accuracy of your model as the sole metric of evaluation will still fail to reveal key information about the results. For example, final test accuracy measurements reveal little about the dependence of the result on the search space used, such as DARTS' notable failure on NAS-Bench-201 (Dong and Yang, 2020). Evaluating the supernet directly has the potential to provide more generalizable insights into the performance of NAS algorithms. By understanding the mechanisms which a NAS algorithm is able to obtain its result, we are better equipped to anticipate how it'll perform in other search spaces and training contexts.

Our main approach to analyzing the shared weights of the supernet is based on model ranking. This approach, ranking candidate architectures based on validation accuracy obtained with the shared weights, is used for architecture selection in supernet NAS methods that don't train arch weights, such as one-shot NAS Bender et al. (2018). We apply this method across a broader set of supernet NAS algorithms, arguing that while DARTS may never use its shared weights in training to evaluate an actual candidate architecture, examining how suitable the shared weights are for varying architectures allows us to utilize the search space to analyze how learning the architecture weights has trained the shared weights.

Supernet NAS methods have been previously shown to produce model rankings with low correlation White et al. (2021). We argue that correlation is a poor indicator of the efficacy of a supernet's shared weights. Making a relative distinction between poor performing architectures is of little value to supernet NAS methods while distinguishing top architectures from the rest is key. Correlation metrics do not account for how the significance of correct ranking varies over the architecture space. Instead, we report the "top architecture ID rate", which we define as the proportion of the top k architectures, ranked by benchmark test accuracy, which are correctly ranked in the top k by validation accuracy via the shared weights. Note that this metric was proposed prior by Ning et al. (2021) as "precision@topk", a name which we do not adopt as this metric corresponds to both precision and recall in this case. We expand on this work by applying this metric to a broader range of supernet NAS methods, providing novel insights into the process of supernet training.

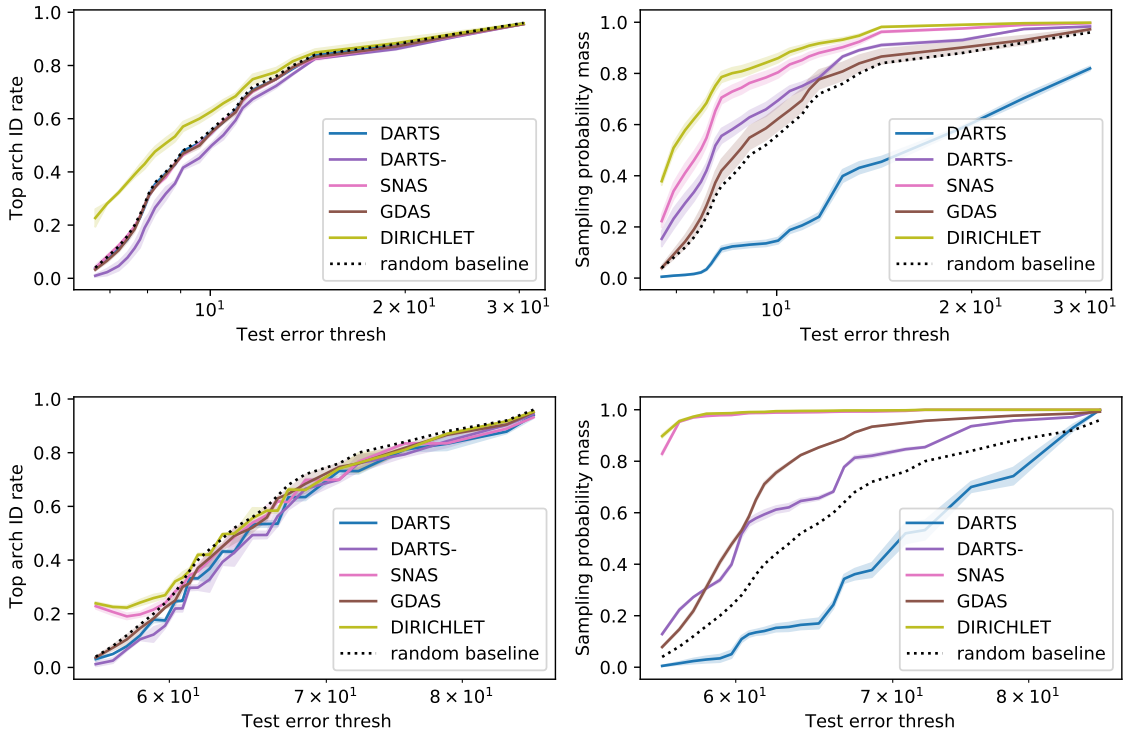

Figure 1: *(Left)* The ability of an algorithm to identify the top architectures in a search space as a function of the threshold for "top" architecture. *(Right)* Cumulative sampling probability across test error threshold groups of the search space, showing these two algorithms as most likely to sample the highest performing architectures. *(Top)* plots show results on CIFAR10 and *(Bottom)* plots show results on ImageNet16-120, all in the NAS-Bench-201 search space.

Our method of evaluating learned architecture weights is simply to compute a sampling probability for each architecture from these weights and view how NAS algorithms assign probability mass over the architecture space. For search spaces, like that of DARTS/NAS-Bench-301, which apply further constraints to the architecture space than NAS-Bench-201 does, this will require defining a sampling process based on the discretization steps used to derive a final architecture.

Both of these measurements, for the shared-weights and the architecture-weights, were implemented as a variant Trainer class for the NASLib library in order to make them widely accessible for use as a general evaluatory tool (Ruchte et al., 2020; Mehta et al., 2022). Between the range of search spaces and data sets already provided through NASLib and these metrics, future researchers who implement their proposed NAS algorithms on NASLib will have a full suite of methods for understanding how and why their algorithms performance varies across contexts.

Most methods for supernet training do not outperform the naive random baseline for most top-architecture thresholds, which aligns with previous work reporting supernet methods to be ineffective at predicting performance in this way (Yu et al., 2020; White et al., 2021). However, some algorithms (e.g. DirichletNAS (Chen et al., 2021) and SNAS (Xie et al., 2019)) are able to identify more accurately the very top architectures in the search space (left side of the subfigure) but show no additional ability in distinguishing between fairly low-performing architectures (right side of the subfigure). This highlights that shared-weight models need not be (and likely cannot be) trained to be well suited for all architectures within a search space. Rather, they will likely be most

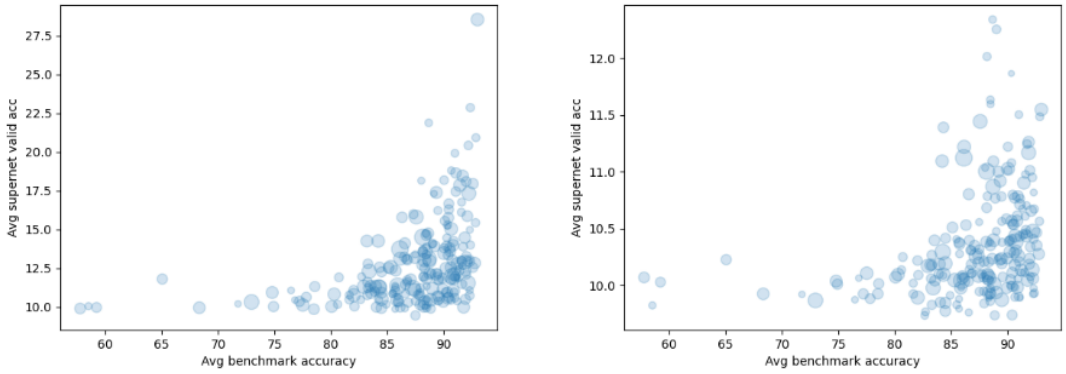

Figure 2: *(Left)* Supernet validation accuracy vs benchmark test accuracy averaged over architecture clusters for a sample SNAS run. *(Right)* Supernet validation accuracy vs benchmark test accuracy averaged over architecture clusters for a sample DrNAS run.

informative for helping us to identify top architectures if those shared weights are specialized to maximize performance estimation of the highest performing architectures. This capability is missed in previous studies by the ranking correlation metrics used to assess performance predictions, and in Ning et al. (2021) as a result of only evaluating a less effective supernet training algorithm than SNAS and DirichletNAS.

A second metric for accessing quality of trained supernet models is how likely it is to sample a top architecture (if sampled at the end of training). Again using the ranking of architectures by the benchmarked test accuracy, we parameterize a probability distribution over the search space using the supernet's architecture weights to assess the likelihood of sampling architectures of a given rank or better. Fig 1 (*right*) shows the cumulative sampling probability as the top-architecture threshold is relaxed, relative to a baseline of randomly selecting architectures from the search space. Algorithms above the cumulative baseline curve (e.g. DirichletNAS and SNAS again, and – to a lesser extent – GDAS (Dong and Yang, 2019) and DARTS- (Chu et al., 2021)) oversample higher performing architectures, while curves below the baseline (e.g. DARTS (Liu et al., 2019)) oversample lower performing discrete architectures.

3 Understanding supernet specialization

The dirichlet sampling technique used by DrNAS clearly proves more effective at guiding the search process than the Gumbel-Softmax approach to differentiable categorical sampling used by SNAS. Both of these algorithms, do however, display some capacity to estimate performance and, as shown in their performance prediction results over the search space, specifically are able to estimate performance for better performing architectures. These results underscore the utility of the top architecture ID rate measure, as the ineffectiveness of these methods for predicting the performance of the lowest ranking architectures would lead to a low ranking correlation across the entire architecture space. These results indicate that these algorithms are able to make successful use of the supernet structure by specializing the shared weights to work for a set of structurally similar high performing architectures.

We can explore this supernet specialization hypothesis by clustering the architectures by their structural similarity. To do so, we define the pairwise similarity between architectures as the number of overlapping operations in their encodings. Then, performing hierarchical clustering on the encodings, for a given threshold we can obtain a set of clusters and examine the relationship

between the average benchmark test accuracy of each cluster and the average validation accuracy for the architectures of the cluster obtained using the shared weights. Shown above for sample runs from SNAS and DrNAS in Fig 2, it is apparent that it is indeed the case that, while the shared weights are far from optimal for every architecture, for many architectures they do not support inference better than selection at random but for a subset of top architectures they do. The existence of clusters of structurally similar, high-performing architectures which are able to be distinguished from lower-performing architectures through obtaining validation accuracy measurements from the supernet indicates that it is the structure of the search space, specifically, the presence of these families of similarly effective designs, which enables this effective specialization of the supernet.

4 Conclusion

Although SNAS and DrNAS both attain final test accuracy scores near the optimal score for CIFAR10 on NAS-Bench-201, deeper analysis of the supernet algorithms trained by each method reveals a more significant underlying disparity: DrNAS is considerably more successful at identify top architectures, both through its shared weights and its architecture weights. This suggests that the extent of the performance boost attained by DrNAS's switch to dirichlet architecture sampling is obfuscated when viewing final test accuracy results alone by the limited size of the benchmark search space. Together, these measures provide a set of tools supporting more robust evaluation of NAS methods, made accessible for broad use through their implementation on top of the NASLib library.

5 Limitations

This work only examines the proposed evaluation measures on a single NAS benchmark data set, NAS-Bench-201. Therefore, the generalizability of the insights obtained from these measures across search spaces cannot be confirmed. Additionally, these measures are computed over the entirety of the search space, which is only possible for tabular benchmarks, so further work is needed to extend these measures to surrogate NAS benchmarks, such as NAS-Bench-301 (Siems et al., 2022). Additionally, while the specific set of supernet NAS methods evaluated are able to provide general insights into the performance of supernet NAS algorithm through analysis based in the specfic differences between algorithm, the use of a wider set of supernet NAS algorithms could provide even broader and more generalizable insights.

6 Broader Impact

Our work demonstrates a method for evaluating supernet-based NAS algorithms using benchmark data sets, made easily accessible to researchers through implementation in the NASLib library. Through this contribution to open-source NAS research we hope to encourage deeper NAS benchmark use and more thorough evaluation in future supernet NAS research. A stronger scientific understanding of supernet NAS techniques will aid in both the development and proliferation of efficient architecture search methods. Ultimately, the widespread availability of highly effective efficient architecture search methods has the potential to make the development of deep learning models more widely accessible. While this does have the potential to make easier the development of potentially harmful models, we believe that architecture search techniques also offers the potential for a widely accessible method of auditing and guaranteeing properties of the architecture of neural networks, therefore leading to more socially beneficial deep learning.

References

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In Dy, J. and Krause, A., editors, *Proceedings of the*

35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 550–559, Stockholmsmässan, Stockholm Sweden. PMLR.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. (2021). Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., and Yan, J. (2021). Darts-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations (ICLR)*.

Dong, X. and Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*.

Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K., and Hutter, F. (2022). Nas-bench-suite: Nas evaluation is (now) surprisingly easy. In *International Conference on Learning Representations*.

Ning, X., Li, W., Zhou, Z., Zhao, T., Zheng, Y., Liang, S., Yang, H., and Wang, Y. (2021). Evaluating efficient performance estimators of neural architectures.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *ICML*.

Ruchte, M., Zela, A., Siems, J., Grabocka, J., and Hutter, F. (2020). Naslib: A modular and flexible neural architecture search library. `https://github.com/automl/NASLib`.

Siems, J. N., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., and Hutter, F. (2022). Surrogate {nas} benchmarks: Going beyond the limited search spaces of tabular {nas} benchmarks. In *International Conference on Learning Representations*.

White, C., Zela, A., Ru, B., Liu, Y., and Hutter, F. (2021). How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*.

Xie, S., Zheng, H., Liu, C., and Lin, L. (2019). SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. (2020). Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.

Appendices

A Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes]

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] Code for experiments is available at https://anonymous.4open.science/r/NASLib-F26E/

   (b) Did you include the raw results of running the given instructions on the given code and data? [No] Data is excluded from code due to significant size without extensive computational requirements.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [No] Plotting code is not included at the time of submission, however more general visualization code to pair with the measures is in development.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See Implementation details in Appendix B

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [N/A]

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [No] This is a key component of this project in development for future work.

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes]

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [No] Licensing was not mentioned in the main text, however, the only asset used, NASLib, is licensed via Apache 2.0.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

B Implementation details

Source code is available at https://anonymous.4open.science/r/NASLib-F26E/. The runs presented used random seed [1001, 1002, 1003], run hyperparameters were the default hyperparameters used in NASLib for DARTs-like methods and are available in the source code in

```
scripts/exp_hyperparam
```

All tests were run on AMD Radeon Instinct MI50 Accelerators, requiring roughly 925 GPU hours for all experiments.

C Further Supernet Evaluation Results and Details

Our independent evaluation of Stage-1 algorithms utilizes two statistics we developed, explained in further detail below, which show two algorithms, SNAS and DrNAS, to be dominant. These algorithms are the only two to display a top arch ID rate above the baseline. This indicates that the shared-weights of their supernets have been trained such that they are capable of distinguishing the best architectures from the rest by computing validation accuracies. They also display the highest probability of sampling the best architectures in the search space. We hypothesize that these results are interrelated. As these are sampling based algorithms, sampling top architectures during training

allows the shared weights to be updated to be more suitable for these top architectures, which in turn improves the quality of gradient updates guiding the architecture weights toward top models.

The significant improvement in the cumulative sampling probability shown from the addition of an auxiliary skip connection to DARTS, represented by the DARTS- results supports the hypothesis that overvaluing skip connections is a key issue for DARTS. As correcting for this overweighting seems to close the gap between DARTS and the baseline, it seems the cumulative sampling probability results for DARTS simply reflect a high probability of sampling an architecture which is low performing due to excessive skip connections.

Fig. 3 shows the plot of Stage-1 search algorithm performance estimation and architecture sampling metrics on CIFAR-100 in the NASBench-201 search space, corresponding to the plots on CIFAR-10 and ImageNet16-120 shown in Sec. 2. These results are consistent with those shown for CIFAR-10.

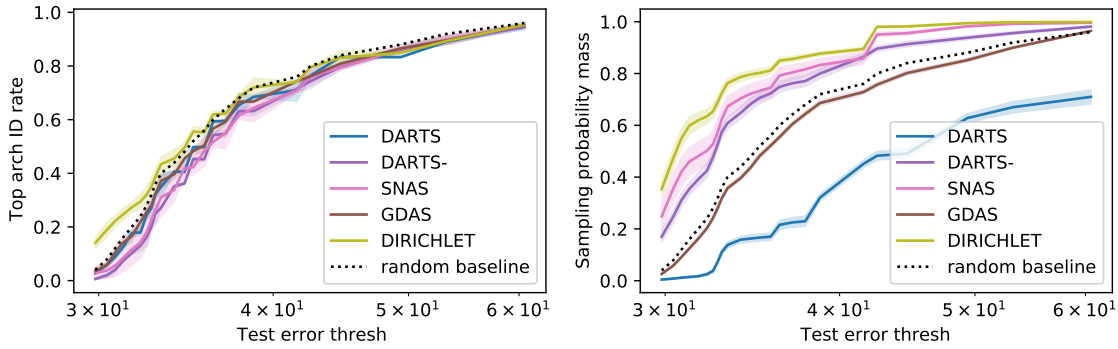

Figure 3: Left: Top arch ID rate across test regret threshold groups of the architecture space. Right: Sampling probability mass across test regret threshold groups of the probability space.

C.1 Top Architecture ID Rate

Previous NAS papers have utilized Spearman or Kendall-Tau ranking correlation to evaluate the capacity of NAS algorithms to estimate the performance of candidate models. Although these statistical are powerful tools to evaluate rankings, we argue that, when applied to the entirety of a NAS search space, they do not necessarily reflect the characteristics of performance estimation which are important to NAS algorithms.

For example, consider in the 15,625 architecture search space of NAS-Bench-201 one NAS algorithm which ranks the top 625 architectures correctly and the bottom 15,000 in reverse order compared to second NAS algorithm which ranks the architectures correctly except the first 625 models are, in reverse order, assigned the ranks of the last 625. Here the former NAS algorithm would result in a Spearman correlation of -0.7695 and a Kendall Tau correlation of -0.8432 while the latter would result in a Spearman correlation of 0.7695 and a Kendall Tau correlation of 0.8432. But we argue that the 1st example NAS algorithm has succeeded at performance estimation: it has correctly identified the global maximum and the ordering of the top 4% of the search space while the 2nd example NAS algorithm has failed catastrophically, as it is maximally incorrect in its estimation of the relative performance of the top 4% of the search space.

By utilizing a threshold-based statistic we can configure our measurement such that it reflects an NAS algorithm's capacity to estimate performance over specific regions of the search space, thus ignoring the algorithms inability to distinguish between similarly poorly performing models and capturing, in a coarser sense, the ability of the algorithm to distinguish better from worse models.

For a given index based threshold in the true architecture ranking k, we compute a a pair of error thresholds corresponding to this index using the benchmark test accuracies and the validation accuracies obtained using the shared weights of the trained supernet. We then identify the sets of architectures falling under each error threshold, which we denote T_k and V_k, respectively. Our top architecture ID rate metric is then given by:

$$a_{1,k} = \frac{|T_k \cap V_k|}{|T_k|}$$

C.2 Sampling probability mass

Due to the design of NAS-Bench-201's search space, which allows a single operation at each edge, the architecture weights, after a softmax, correspond exactly to set of categorical distributions (which we model as independent) over operations at each edge. If we consider the use of these distributions to sample architectures, we find that the probability of sampling any given architecture from a set of architecture weights can be computed efficiently as the inner product of the one-hot representation of that architecture and the architecture weights.

Note that, while a wide range of architecture parameterizations permit the creation of a sampler based on learned architecture weights, it is often less straightforward than in NASBench-201. For example, in the search space of DARTS, each node selects two incoming edges out of all operations across all nodes with potential in-edges. Creating a sampler in this search space would then require designing a scheme for combining and comparing the weights across different in-edges and a method to sample a pair of operations rather than a single operation.

Following the notation given above for the top architecture ID rate based on the set of architectures (T_k) under an error benchmark test error threshold of k, we can also denote our cumulative sampling probability metric. Our metric for architecture sampling is:

$$a_{2,k} = \sum_{\alpha \in T_k} Pr(\alpha)$$

Note that in our toy example from the previous section, for $k \leq 625$, a_1 will be 1 for the first, successful, NAS algorithm and 0 for the second, failing one.