DIFFERENTIAL EVOLUTION FOR NEURAL ARCHITECTURE SEARCH

Noor Awad[1][*], **Neeratyoy Mallik**[1][*] **& Frank Hutter**[1,2]
[1]Department of Computer Science, University of Freiburg
[2]Bosch Center for Artificial Intelligence
Freiburg, Germany
{awad,mallik,fh}@cs.uni-freiburg.de

ABSTRACT

Neural architecture search (NAS) methods rely on a *search strategy* for deciding which architectures to evaluate next and a *performance estimation strategy* for assessing their performance (e.g., using full evaluations, multi-fidelity evaluations, or the one-shot model). In this paper, we focus on the search strategy and demonstrate that the simple yet powerful evolutionary algorithm of *differential evolution* (DE) yields state-of-the-art performance for NAS, comparing favourably to regularized evolution and Bayesian optimization. It yields improved and more robust results for 13 tabular NAS benchmarks based on NAS-Bench-101, NAS-Bench-1Shot1, NAS-Bench-201 and NAS-HPO bench.

1 INTRODUCTION

Evolutionary algorithms have a long history for neural architecture search (NAS), for combined search of architectures and weights (Stanley et al., 2019; 2009), (Mason et al., 2017a; Baioletti et al., 2020), search of just the architecture (Real et al., 2017; Elsken et al., 2018; Liu et al., 2017), and multi-objective optimization of performance and resource consumption (Elsken et al., 2019b). Recently, regularized evolution (Real et al., 2018) has been shown to yield very robust performance on NAS benchmarks (Ying et al., 2019) and found novel neural architectures for object recognition in CIFAR-10 (Real et al., 2018) and to an improved version of the transformer (So et al., 2019).

Here, we study the use of the popular evolutionary algorithm of *differential evolution* (DE, Storn & Kenneth (1997)) for NAS. DE has previously been used to search basic neural network architectures. For example, Mineu et al. (2010) used DE to search for layers, neurons, weights and connections for architectures using a special local and global neighbourhood strategy for the mutation operation, Bhuiyan (2009) introduced a simple DE algorithm without the use of a crossover operation, and Zhang et al. (2019) used DE to jointly evolve architectures and weights, followed by the Levenberg-Marquardt algorithm to finetune the generated weights. Other works that use DE to evolve basic neural architectures can be found in (Dhahri et al., 2012; Mason et al., 2017b). In this paper and different from the above, rather than developing a customized DE version for a specific task, we standardize and benchmark the use of a simple, yet effective DE, for a wide range of NAS benchmarks .

DE has been used as one of many algorithms for a recent benchmark of joint hyperparameter optimization and NAS Klein et al. (2018), and did not yield state-of-the-art performance there. However, that study used a simple SciPy Virtanen et al. (2020) implementation, and we demonstrate that with a better implementation and a fixed, robust hyperparameter setting, DE does indeed achieve state-of-the-art performance on a wide range of recent NAS benchmarks compared to other blackbox optimizers.

Most recent progress in NAS focuses on exploiting the one-shot model introduced by Pham et al. (2018), prominently based on extensions of differentiable architecture search (DARTS (Liu et al., 2018)). However, the one-shot model in general (Sciuto et al., 2019) and DARTS in particular (Zela et al., 2020b) feature several failure modes. For this reason, using the terminology of Elsken et al.

---

[*]Equal contribution

(2019a), we do not employ the one-shot model as a performance estimation strategy to evaluate different search strategies, but rather stick to the simpler performance estimation strategy of full evaluations. While a large-scale evaluation would normally be completely infeasible in this setting due to the high computational cost of full evaluations, this analysis is made possible by the recent availability of tabular NAS benchmarks (Ying et al., 2019).

We first describe a canonical version of differential evolution (DE; Section 2), then describe how to apply DE to NAS (Section 3), and then Section 4 demonstrates that the resulting algorithm outperforms the previous best search strategies on a wide range of 13 benchmarks based on NAS-Bench-101 (Ying et al., 2019) NAS-Bench-1Shot1 (Zela et al., 2020c), NAS-Bench-201 (Dong & Yang, 2020), and NAS-HPO-Bench (Klein & Hutter, 2019). The appendix can be found at: Appendix Link.

2 CANONICAL DIFFERENTIAL EVOLUTION

Differential Evolution (DE, Storn & Kenneth (1997)) is an evolutionary algorithm that is based on four steps (initialization, mutation, crossover and selection). We describe these below, deferring details to Appendix A. In its canonical form, DE is described for continuous optimization.

Initialization. DE is a population-based meta-heuristic algorithm which consists of a population of NP individuals. Each individual is considered a solution and expressed as a vector of D-dimensional decision variables, which are initialized uniformly at random in the search range.

Mutation. A new child/offspring is produced using the mutation operation for each individual in the population by a so called mutation strategy. The classical DE uses $rand/1$ mutation, in which three random individuals/parents $X_{r_1}, X_{r_2}, X_{r_3}$ are chosen to generate a new vector $V_{i,g}$ as follows:

$$V_{i,g} = X_{r_1,g} + F \cdot (X_{r_2,g} - X_{r_3,g}) \qquad (1)$$

where $V_{i,g}$ is the mutant vector generated for each individual $X_{i,g}$ in the population, F is the scaling factor (which usually takes values within the range $[0, 1]$), and r_1, r_2, r_3 are the indices of different randomly selected individuals. The subscript g indicates the generation index, or iteration number.

Crossover. After the mutation, a crossover operation is applied to each target vector $X_{i,g}$ and its corresponding mutant vector $V_{i,g}$ to generate a trial vector $U_{i,g}$. We use a simple binomial crossover, which chooses the value for each dimension i from $V_{i,g}$ with probability Cr and from $X_{i,g}$ otherwise.

Selection. After generating the trial vector $U_{i,g}$, DE computes its function value $f(U_{i,g})$, keeping $U_{i,g}$ if it performs at least as well as $X_{i,g}$ and reverting back to $X_{i,g}$ otherwise.

3 DIFFERENTIAL EVOLUTION FOR NAS

Recent NAS approaches and benchmarks parameterize cell structures of deep neural networks as directed graphs (Zoph et al., 2018; Ying et al., 2019; Zela et al., 2020a; Dong & Yang, 2020). The realisation of a candidate cell structure can be seen as an assignment of operations from a set of choices or a range of values, such as the choice of operator on an edge or the choice of predecessors of a node in the directed graph.

We found the best way of applying DE when parameters are discrete or categorical is to keep the population in a continuous space, perform canonical DE as usual as described in Section 2, and only discretize copies of individuals to evaluate them. If we instead dealt with a discrete population space, then the diversity of population would drop dramatically, leading to many individuals having the same parameter values; the resulting population would then have many duplicates, lowering the diversity of the difference distribution and making it hard for DE to explore effectively.

The modified canonical DE we used for NAS is presented in Algorithm 1 of Appendix B. Figure 1 shows the general framework of our DE implementation. We scale all NAS parameters to [0, 1] to let DE work on individuals from a uniform, continuous space. The continuous value for $U_{i,g}$ needs to be mapped back to the original space of the NAS parameters before the function evaluation. In Algorithm 1, we use a method *discretized_architecture* to do this; this method retrieves the following values X^i depending on the parameter's type:
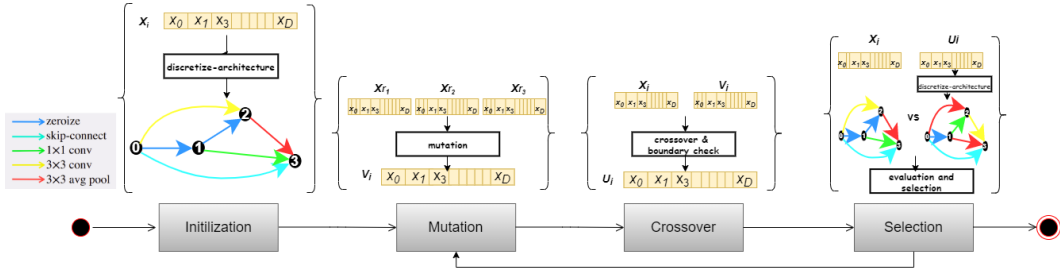
Figure 1: DE for NAS Framework

- *Integer* and *float* parameters: $X^i \in [a_i, b_i]$ are retrieved as: $a_i + (b_i - a_i) \cdot U_{i,g}$, where the integer parameters are additionally rounded.
- *Ordinal* and *categorical* parameters $X^i \in \{x_1, ..., x_n\}$: the range [0, 1] is divided uniformly into n bins.

We illustrate the discretization in Figure 2. For the categorical parameter $X^2 \in \{$'*1x1 conv*', '*skip*', '*3x3 conv*'$\}$, the corresponding continuous DE space maps to $[0, 1/3)$ for '*1x1 conv*', $[1/3, 2/3)$ for '*skip*', and $[2/3, 1]$ for '*3x3 conv*'. As seen in Figure 2, the *difference vector* and the randomly sampled candidate individuals determine how the search space is spanned to find a *mutant vector* that participates in the selection process. The resultant mutant can lie on any of the 9 grids formed in Figure 2 for the 2-dimensional case.
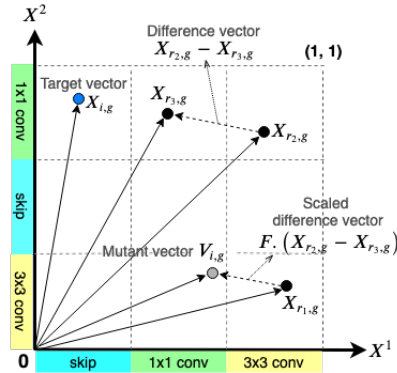

Figure 2: Illustration of DE mutation on categorical parameterization of NAS cell space

One drawback for such an approach might arise in the case of a conditional parameter space. However, just like in NAS-Bench-101 (Ying et al., 2019), the function value for an *invalid architecture* can simply be a maximal error of 1 (at no computational cost, even in a non-tabular benchmark). Such individuals will be guaranteed to lose in the selection process, thereby implicitly avoiding invalid architectures over time.

4 EXPERIMENTS

We evaluate DE's performance on four recent NAS benchmarks: NAS-Bench-101 (Ying et al., 2019), NAS-HPO (Klein & Hutter, 2019), NAS-Bench-1shot1 (Zela et al., 2020a) and NAS-Bench-201 (Dong & Yang, 2020). We compare against several baseline algorithms, namely Random Search (RS) (Bergstra & Bengio, 2012), BOHB (Falkner et al., 2018), Tree Parzen Estimator (TPE) (Bergstra et al., 2011), Hyperband (HB) (Li et al., 2018) and regularized evolution (RE) (Real et al., 2018). Appendix C has more details about the used algorithms and their hyperparameter settings. For DE, we set scaling factor F and crossover rate Cr to 0.5 over all the generations. For the population size NP, we tested several values (provided in Appendix F and chose 20 for our experiments. We consider RE as the *primary* baseline algorithm (run until $10Ms$) since it belongs to the same family of algorithms as DE and has been shown to perform robustly many times before. We provide a comparison of the robustness between RE and DE in Appendix E. For each algorithm, we performed 500 independent runs and report the mean performance of the immediate validation regret (Ying et al., 2019). Throughout, we evaluate algorithms in the anytime setting, showing performance of the best found configuration over time as suggested by Ying et al. (2019) and Lindauer & Hutter (2019). In all our plots, the x-axis shows *estimated wall-clock time*, as the cumulative time taken for training each of the architectures found as returned by the NAS benchmarks. Due to the space limitation, we show the test regret plots for NAS-101 and NAS-1shot1,

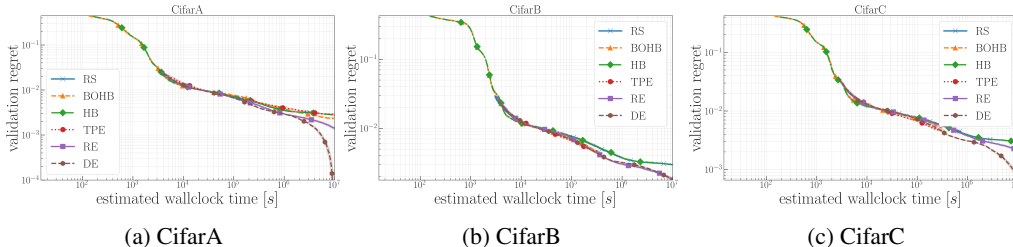(a) CifarA   (b) CifarB   (c) CifarC

Figure 3: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-Bench-101 on CifarA, CifarB and CifarC.

and also discuss the experiments for NAS-Bench-201 and NAS-HPO in Appendix D.1, D.2, D.3, D.4, respectively. We compare our implementation with the popular SciPy-DE code Virtanen et al. (2020) in Appendix G. Our code for DE and for reproducing our experiments is publicly available at `https://github.com/automl/DE-NAS`.

4.1 NAS-BENCH-101

In this experiment we investigated DE's performance on the cell search space of 423k unique cell architectures of a convolutional neural network for CIFAR-10 defined by NAS-Bench-101 (Ying et al., 2019). We study three different search spaces: *CifarA* contains the main search space discussed by Ying et al. (2019), and *CifarB* and *CifarC* are variants of the same space with alternative encodings (treating the edge parameters as categorical parameters with 21 choices and continuous $\in [0, 1]$, respectively). Figure 3 presents a comparison of the performance of compared algorithms showing the mean validation regret of 500 independent runs as a function of the estimated training time. We show our results for test regret in Appendix D.1. HB and BOHB are multi-fidelity optimization algorithms which evaluate at fewer epochs while other algorithms evaluate only at E_{max}. However, NAS-Bench-101 features a low rank correlation between the performance obtained with different budgets (Ying et al., 2019), and thus these algorithms do not perform better than the other algorithms that only use the maximum number of epoch. The other algorithms (RS, TPE, and RE) follow the same behaviour at the beginning of the search for all 3 encodings of the search space, and in the end the evolutionary algorithms RE and DE clearly yield the best performance. DE shows much better final performance for *CifarA* and *CifarC* and competitive performance with RE for *CifarB*. It appears that DE is able to exploit high-dimensional spaces well and handle mixed-types better. This may be attributed to NAS-Bench-101's locality property (Ying et al., 2019) along with DE's search method, since a DE population with individuals from a good region will be able to exploit further and get near the global optimum.

4.2 NAS-BENCH-1SHOT1

NAS-Bench-1Shot1 (Zela et al., 2020c) was created from the search space of NAS-Bench-101 by keeping the network-level topology intact and modifying the cell-level topology to allow the application of modern weight sharing algorithms for three search spaces with $6, 240$ (search space 1), $29, 160$ (search space 2), and $363, 648$ (search space 3) architectures. Figure 4 shows our results for the mean performance on validation regret while we present a comparison on test regret in Appendix D.2. For search space 1, all the algorithms achieve nearly the same error at the beginning of the search, then DE converges faster until other algorithms catch up. For search space 2, RE and DE converge fastest. For search space 3, the most complex (high-dimensional) and largest (10x more architectures than space 2, and 100x more than space 1), DE clearly outperforms all other algorithms and converges fastest.

5 CONCLUSION

We demonstrated that Differential Evolution can be utilised as an alternative search strategy for the growing field of NAS. We also demonstrated DE's ability to handle mixed data types and high-dimensional spaces. DE may thus be a good candidate for NAS in very large spaces that may help

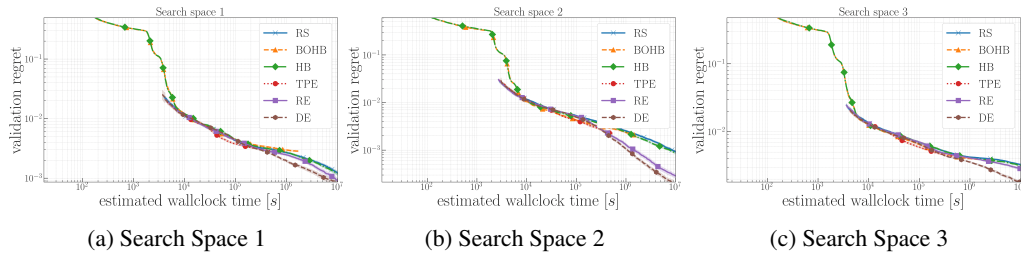(a) Search Space 1   (b) Search Space 2   (c) Search Space 3

Figure 4: A comparison of the mean validation regret performance of 500 independent runs as a function of estimated training time for NAS-1Shot1 on the three different search spaces.

discover new, yet unknown, architectural design patterns. Since DE naturally lends itself well to parallelization, future work includes providing a parallel implementation. We are also interested in combining DE with different performance estimation strategies, such as multi-fidelity methods and the one-shot model. Our reference implementation of the code is available at `https://github.com/automl/DE-NAS`.

ACKNOWLEDGMENTS

REFERENCES

M. Baioletti, G. Di Bari, A. Milani, and Valentina Poggioni. Differential evolution for neural networks optimization. *Mathematics*, 8(1), 2020.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proc. of NeurIPS'11*, pp. 2546–2554, 2011.

Md. Zakirul Alam Bhuiyan. An algorithm for determining neural network architecture using differential evolution. In *International Conference on Business Intelligence and Financial Engineering*, July 2009.

H. Dhahri, A. Alimi, and A. Abraham. Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network. *Neurocomputing*, 97:131–140, November 2012.

X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv:2001.00326 [cs.CV]*, 2020.

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv:1804.09081 [stat.ML]*, 2018.

T. Elsken, J. Metzen, and F. Hutter. Neural architecture search: A survey. *JMLR*, 20:1–21, 2019a.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=ByME42AqK7.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proc. of ICML'18*, pp. 1437–1446, 2018.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv:1905.04970 [cs.LG]*, 2019.

A. Klein, E. Christiansen K. Murphy, and F. Hutter. Towards reproducible neural architecture and hyperparameter search. In *ICML Reproducibility in Machine Learning Workshop*, 2018.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(185):1–52, 2018.

Marius Lindauer and Frank Hutter. Best Practices for Scientific Research on Neural Architecture Search. *arXiv e-prints*, art. arXiv:1909.02453, Sep 2019.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv:1711.00436 [cs.LG]*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. URL http://arxiv.org/abs/1806.09055.

K. Mason, J. Duggan, and E. Howley. Neural network topology and weight optimization through neuro differential evolution. In *Genetic and Evolutionary Computation Conference*, pp. 213–214, July 2017a.

K. Mason, J. Duggan, and E. Howley. Evolving multi-objective neural networks using differential evolution for dynamic economic emission dispatch. In *Genetic and Evolutionary Computation Conference*, pp. 1287–1294, July 2017b.

Nicole L. Mineu, Teresa B. Ludermir, and Leandro M. Almeida. Topology optimization for artificial neural networks using differential evolution. In *International Joint Conference on Neural Networks*, October 2010.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL http://arxiv.org/abs/1802.03268.

E. Real, S. Moore, A. Selle, S. Saxena, Y. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv:1703.01041 [cs.NE]*, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. Le. Regularized evolution for image classifier architecture search. *arXiv:1802.01548 [cs.NE]*, 2018.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142, 2019. URL http://arxiv.org/abs/1902.08142.

David R. So, Chen Liang, and Quoc V. Le. The evolved transformer. *CoRR*, abs/1901.11117, 2019. URL http://arxiv.org/abs/1901.11117.

K. Stanley, D. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(8):24–35, 2019.

R. Storn and P. Kenneth. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *J GLOBAL OPTIM*, 11:341–359, 1997.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. URL https://doi.org/10.1038/s41592-019-0686-2.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/ying19a.html.

A. Zela, J. Siems, and F. Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv:1902.09635 [cs.LG]*, 2020a.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=H1gDNyrKDS.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search, 2020c.

L. Zhang, H. Li, and X. Kong. Evolving feedforward artificial neural networks using a two-stage approach. *Neurocomputing*, 360:25–36, September 2019.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.