Our Attempt to Improve DEHB with Exploration and Regularization

Luca Pfrang[1] **Akshay Chandra Lagandula**[1] **Sri Harsha Koribille**[1]

[1]University of Freiburg, Germany

Abstract Our decision to use DEHB (Awad et al. (2021)) as the main backbone HPO algorithm for this project was easy, given its many useful properties. From being conceptually simple to computationally cheap, DEHB fulfils all the desiderata of a good HPO optimizer. Additionally, through this project, we wanted to understand if DEHB can benefit from more exploration at different stages of its iteration. To that end, we proposed three exploration hungry methods - SoftDEHB (DEHB with softmax sampling of promotion population), RandDEHB (DEHB with fractional random sampling of promotion population) and RestartDEHB (DEHB with exploration only at lowest budgets). While all DEHB variants beat the baseline CNN's test accuracy without data augmentation, SoftDEHB and RandDEHB experiments on FashionMNIST show that DEHB doesn't need to explore any more than it already does. Our SoftDEHB and RestartDEHB experiments on HPOBench (Klein and Hutter (2019)) further confirm this as their performance is comparable to, albeit no better or worse than, DEHB. Our project repository is at `https://github.com/automl-classroom/final-project-lash/`.

1 Introduction & Motivation

DEHB (Awad et al. (2021)) intelligently combines the conceptually simple but effective evolutionary optimization method of differential evolution (DE) and the computationally efficient bandit-based Hyperband (HB) approach, explained in detail in Sections 2.1 and 2.2. As claimed in Awad et al. (2021), DEHB is 32× faster than BOHB (Falkner et al. (2018)), one of the best off-the-shelf HPO methods. BOHB has the advantage of Bayesian Optimization which can allow a probabilistic model to trade off exploration and exploitation while modelling the objective function. To that end, we think DEHB achieves exploration through its *rand/1* crossover-based mutation strategy, but we wanted to understand if DEHB can benefit from more exploration. To verify that, we defined three exploration hungry methods - SoftDEHB (DEHB with softmax sampling of promotion population), RandDEHB (DEHB with fractional random sampling of promotion population), RestartDEHB (DEHB with exploration only at lower budgets), explained in Section 2.3.

2 Methodology

2.1 Background

Differential Evoution (DE): DE is a population based evolutionary algorithm which uses 3 iterative steps mutation, crossover and selection. At the beginning of the search on D-dimensional problem, we initialize a population space of N individuals. *Mutation* uses *rand/1* strategy to select 3 random parents $x_{r_1}, x_{r_2}, x_{r_3}$ and generates a new mutant vector based on the weighted difference vector

$$v_{i,g} = x_{r_1,g} + F.(x_{r_2,g} - x_{r_3,g})$$

where $F \in (0, 1]$ is the scaling factor. *Crossover* performs a binomial operation to generate the offspring $u_{i,g}$ by selecting parameter values from mutant $v_{i,g}$ with a probability p(crossover rate)

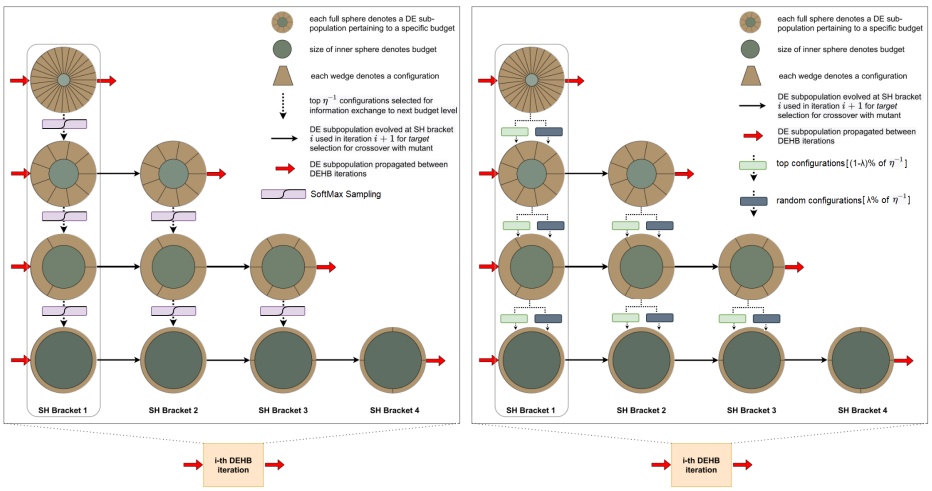


Figure 1: Overview of SoftDEHB (left) and RandDEHB (right) iterations. In SoftDEHB, configurations are promoted across fidelities through the softmax sampling (top-down) and from left to right by updating the sub-population. In RandDEHB, configurations are promoted across fidelities through a mix of greedy parent selection and random parent selection (top-down). Figures are slightly modified from Awad et al. (2021).

and from the individual $x_{i,g}$ otherwise. *Selection* uses the better of $x_{i,g}$ and $u_{i,g}$ as the population of the next generation

Hyperband (HB): Hyperband is a multi-fidelity optimization technique with Successive Halving (SH) at its core. A vanilla SH bracket samples N configurations, evaluates them at the lowest budget and forwards a fraction of top $1/\eta$ configurations to the next higher budget which is η more expensive than the previous budget. This process repeats iteratively until the highest budget is reached. HB runs different instantiations of SH with successively larger lower budgets, thereby finding configs that perform well on higher budgets but poorly on lower ones. One iteration of HB is a sequence of SH brackets starting with different starting budgets and different number of configurations per bracket.

2.2 DEHB

DEHB, as the name suggests, combines the evolutionary search approach of DE with the multi-fidelity-based speedup capabilities of HB adding minor variations to both methods. It maintains a subpopulation for every fidelity level such that no random sampling occurs except for the lowest fidelity. The first iteration of DEHB is similar to vanilla HB as random search is used to initialise the subpopulation of lowest fidelity. But the subsequent SH brackets starting at higher budgets begin by reusing the subpopulation updated in the previous SH bracket and carrying out a DE evolution. The top performing configurations are not directly promoted for evaluation on higher budgets; rather they're collected in a parent pool. Unlike vanilla DE, the mutants involved in the modified DE evolution are extracted by sampling parents from the parent pool instead of the current subpopulation directly. This allows the evolution to incorporate and combine information from the current budget and also from the lower budgets.

2.3 Regularized DEHB

Inspired by the simplicity and success of Regularized Evolution (RE) (Real et al. (2019)), we attempt to transfer the idea of RE to DEHB. As proposed in Real et al. (2019), RE drops the oldest member

of the population to make room for a new member, therefore dubbing this procedure as *regularized* or *ageing evolution*. Consequently, this shifts the search for a good architecture on an exploitation-exploration spectrum towards the latter. As the authors in Real et al. (2019) hypothesize, exploration makes the evolutionary algorithm less sensitive to noise in the training procedure of architectures.

In a related attempt, we try to enforce more exploration in DEHB by regularizing the parent pooling process. The two methods we propose are described subsequently.

RandDEHB: In RandDEHB, we modify the parent pooling process of DEHB to sample a fraction of the promotion population according to the best configurations of the lower brackets and the rest ($\lambda \times$ top η^{-1} configurations precisely) randomly. λ controls the scope of regularization of this method.

SoftDEHB: In SoftDEHB, we evolve RandDEHB to sample the promotion population according to a softmax (with temperature τ) distribution over the fitness values. Note that we can control the output distribution by tuning τ of the softmax, where high temperatures push the output distribution to be more uniform leading to more exploration. This particularly allows for different exploration exploitation trade-offs. Additionally, we also introduce temperature schedules to change the degree of exploration over time. Please refer to Section 3.2 for more information.

RestartDEHB: Since SoftDEHB and RandDEHB introduce the same amount of regularization on the parent pooling process on all budget levels, in RestartDEHB we restrict regularization to the lowest budget. More specifically, we re-initialize the subpopulation of the lowest budget of a DEHB iteration in every other iteration. By restricting regularization to the lowest budget we respect the hierarchical nature of DEHB and do not disregard information that has already flown to higher budgets, lightly.

Unfortunately, we couldn't test RestartDEHB on FashionMNIST since we came up with it at the last moment. However, you can find RestartDEHB's results on HPOBench in Section 3.2.

3 Experiments & Results

3.1 FashionMNIST

Due to space constraints, we refer the readers to Append A for the experimental details, HPO search space and more. As seen in Table 1, all DEHB variants (without data augmentation) beat the baseline's test accuracy by a decent margin, while DEHB is the best of them all. These test accuracy statistics were averaged over two random seed runs. Looking at optimizer performances over time, it is abundantly clear in Figure 2a that DEHB doesn't require any more exploration as it converges to the lowest validation regret around 4000 seconds while its regularized variants take almost four times more time to converge to their lowest validation regret. This suggests that DEHB is robust and doesn't need to explore at all.

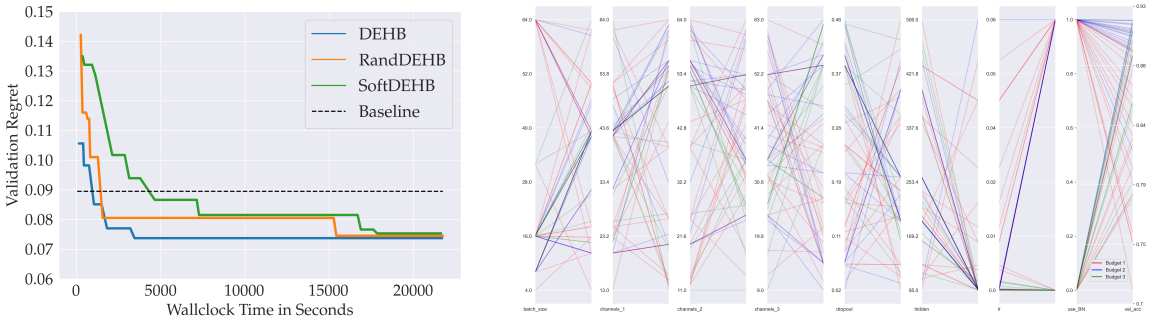
Method	**Test Accuracy (%)**
Baseline	90.5
DEHB	**92.75**
SoftDEHB	92.40
RandDEHB	92.35

Table 1: Best Incumbent's Test Error (50 Epochs, No Data Augmentation, Average of Two Random Seeds)

We also visualized HP choices and their final validation accuracy across budgets of a DEHB run with a parallel coordinate plot, shown in Figure 2b. We couldn't spot a particular trend to discuss it in detail. Note that colour signifies budget, and the final column has each configuration's corresponding validation accuracy. Also note that we excluded kernel and stride HPs from the plot for clarity.

3.2 HPOBench

Since evaluating our methods for many runs on FashionMNIST is costly, we revert to tabular benchmarks to develop and test our methods conceptually. More specifically, we use HPOBench (Klein



(a) Runtime comparison of DEHB, SoftDEHB and RandDEHB on FashionMNIST Results.

(b) Parallel Coordinate Plot of one DEHB run on FashionMNIST.

Figure 2: FashionMNIST Results

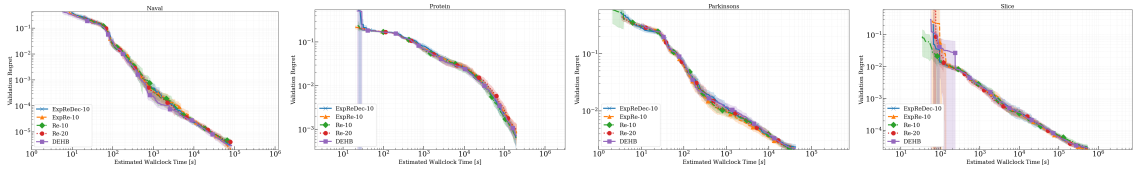


Figure 3: SoftDEHB with temperature schedules and RestartDEHB evaluated on HPOBench (Klein and Hutter (2019)).

and Hutter, 2019), which contains evaluated joint architecture and hyperparameter configurations of feed forward networks on four different regression datasets.

Experimental Details: We chose two variants of SoftDEHB and two variants of RestartDEHB for our main experiments. The SoftDEHB variants use a exponential decaying schedule with restarts denoted as *expRe* and a similar version where we additionally decay the restarting temperature denoted as *expReDec*. For RestartDEHB we chose a variant with re-initialization of the lowest budget sub-population every 10 and 20 iterations, denoted as *Re-10* and *Re-20*. For more details on the experimental setup we refer to the Appendix B. Subsequently, we present the results

Results and Discussion: Figure 3 shows the final experiments on HPOBench. Although there are occasional phases where our proposed methods find better configurations more quickly, there are no signs of improvement over standard DEHB. In fact, DEHB once again proves to be quite robust to slight disturbances of the parent pool selection and the lowest budget configurations. Our methods perform neither worse nor better. The randomness introduced by differential evolution sufficiently explores the configuration space and does not show the need to be supplemented by additional regularization as introduced by our methods. We give more details on further experiments in Appendix B.

4 Conclusion

In this project, we jointly optimized architecture and hyperparameters on FashionMNIST with DEHB, a state-of-the-art off-the-shelf optimizer at the moment, and beat the baseline CNN test performance. Along with that, we also proposed methods and designed experiments to understand if DEHB can benefit from more exploration. Through our experiments on FashionMNIST and HPOBench, we conclude that DEHB is robust and powerful on its own and doesn't need to explore any more than it already does.

References

Acknowledgements.

Awad, N., Mallik, N., and Hutter, F. (2021). Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. *arXiv preprint arXiv:2105.09821*.

Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*.

Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

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

A Futher FashionMNIST Experiment Details

Our chosen HP search space for FashionMNIST experiments is shown in Table 2. Since DEHB doesn't support conditional HPs, we resort to a fixed backbone architecture of three CNN-layers and two FC-layers but each layer's properties are optimized. In all our experiments we did not use data augmentation on purpose as we wanted to isolate the optimizer induced performance gain over the provided CNN baseline which doesn't use data augmentation. Due to Adam's consistent dominant performance on FashionMNIST, we excluded optimizer choice form the search space. We set $\tau = 1$ for SoftDEHB and $\lambda = 0.3$ for RandDEHB. For all experiments, we set minimum budget of 3 epochs and maximum budget of 50 epochs and a maximum allowed run-time of 6 hours.

	Name	Range	log	Type
Architecture	kernel1, kernel2, kernel3	[2, 3, 5]	-	ordinal
	channels1, channels2, channels3	[8, 64]	-	int
	stride1, stride2, stride3	[1, 2]	-	int
	hidden units (two fc layers)	[64, 512]	✓	int
	batch normalization	[0, 1]	-	ordinal
	dropout rate	[0, 0.5]	-	float
Hyperparameters	batch size	[4, 8, 16, 32, 64]	-	ordinal
	learning rate (Adam)	[1e-6, 1e-1]	✓	float

Table 2: Hyperparamter Search Space for FashionMNIST experiments.

B Further Details of HPOBench Experiments

Experiment Details: We ran SoftDEHB and RestartDEHB methods for 200 DEHB iterations on the Slice, Protein, Parksinsons and Naval dataset. We average each method over 50 runs if not stated otherwise. We plot the validation regret over time, where the validation regret is the difference in error of the best model seen across the search and the current incumbent of the method being run.

For SoftDEHB we try different initial temperatures, as well as more advanced temperature schedules. All schedules operate on the number of DEHB iterations, but could also be adjusted based on time or function evaluations. We test linear decaying and ascending schedules, exponential decay and cosine decay schedules, which we abbreviate as *scheduleName_startTemperature_endTemperature* in our plots. Additionally, we introduce restarts for the exponential decay temperature schedules, restarting the decay every *x-th* DEHB iteration, abbreviated by *expRe-x*. Finally, we test an exponential restart schedule (*expReDec-x*) which decays the

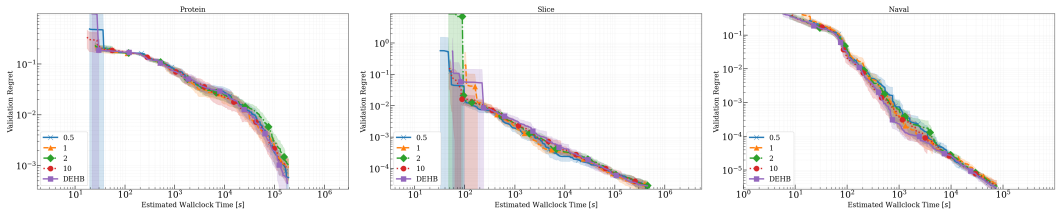


Figure 4: Different initial temperatures for SoftDEHB.

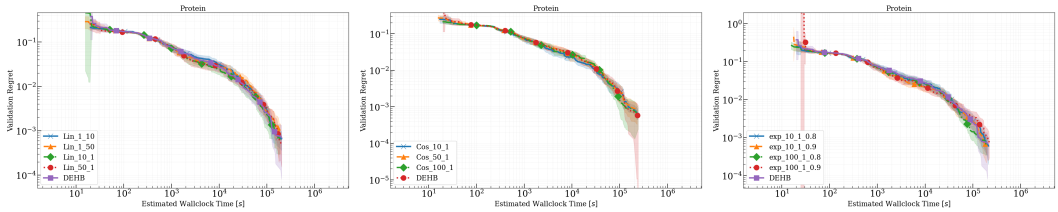


Figure 5: Linear, cosine and exponential temperature schedule for SoftDEHB.

restart temperature over time. The exponential restart schedules in Figure 3 use a decay rate of *0.05*, initial temperature of *10* and revert to regular DEHB once the temperature is close to the *end temperature*, which we chose to be *0.5*. We simply denote RestartDEHB as *Re-x* where we re-initialize the lowest fidelity subpopulation every *x-th* DEHB iteration.

Different Initial Temperatures for SoftDEHB: Figure 4 shows different initial temperatures for SoftDEHB on the Protein, Slice and Naval dataset. Even though DEHB is beaten for a considerable amount of time ($10^3 - 10^4$) on the Naval dataset for some of the SoftDEHB runs, the behavior is not robust across different datasets. Additionally, SoftDEHB converges into DEHB for smaller temperatures. Therefore SoftDEHB shows no signs of consistent improvement over standard DEHB.

Temperatures Schedules for SoftDEHB: Figure 5 shows a linear, cosine and exponential temperature schedule for SoftDEHB on the Protein dataset. The abbreviations are to read as follows: *scheduleName_startTemperature_endTemperature_decayRate*, where *decayRate* refers to the rate of decay for the exponential schedule. For the linear schedule we test an increasing and decreasing temperature. Since we do not observe diverging behavior between the two, we restrict further experiments to decreasing schedules. We believe that descending the temperature makes conceptually more sense, since we reduce exploration over time, converging to the same level of exploration as introduced standard DEHB. We further observe similar behavior to previous experiments, where some schedules do perform slightly better for some time spans of the search and some perform slightly worse.

C Author Contributions

Pfrang. wrote the code for SoftDEHB and RestartDEHB and was responsible for all things HPOBench (from experimental design to ablations).

Lagandula. wrote the code for RandDEHB and was responsible for FashionMNIST code (choosing the HPO search space, model code, running ablations, DEHB code etc.). He created Figure 1.

Koribille. was responsible for bringing DEHB to the table. He wrote the plotting code for graphs and figures in the report, executed all our experiments in his RTX 3050 machine. He also formatted and reorganized the code repository and added necessary documentation.

All authors contributed equally in coming up with the Regularised DEHB variants and in writing the report.