A resource-efficient method for repeated HPO and NAS problems

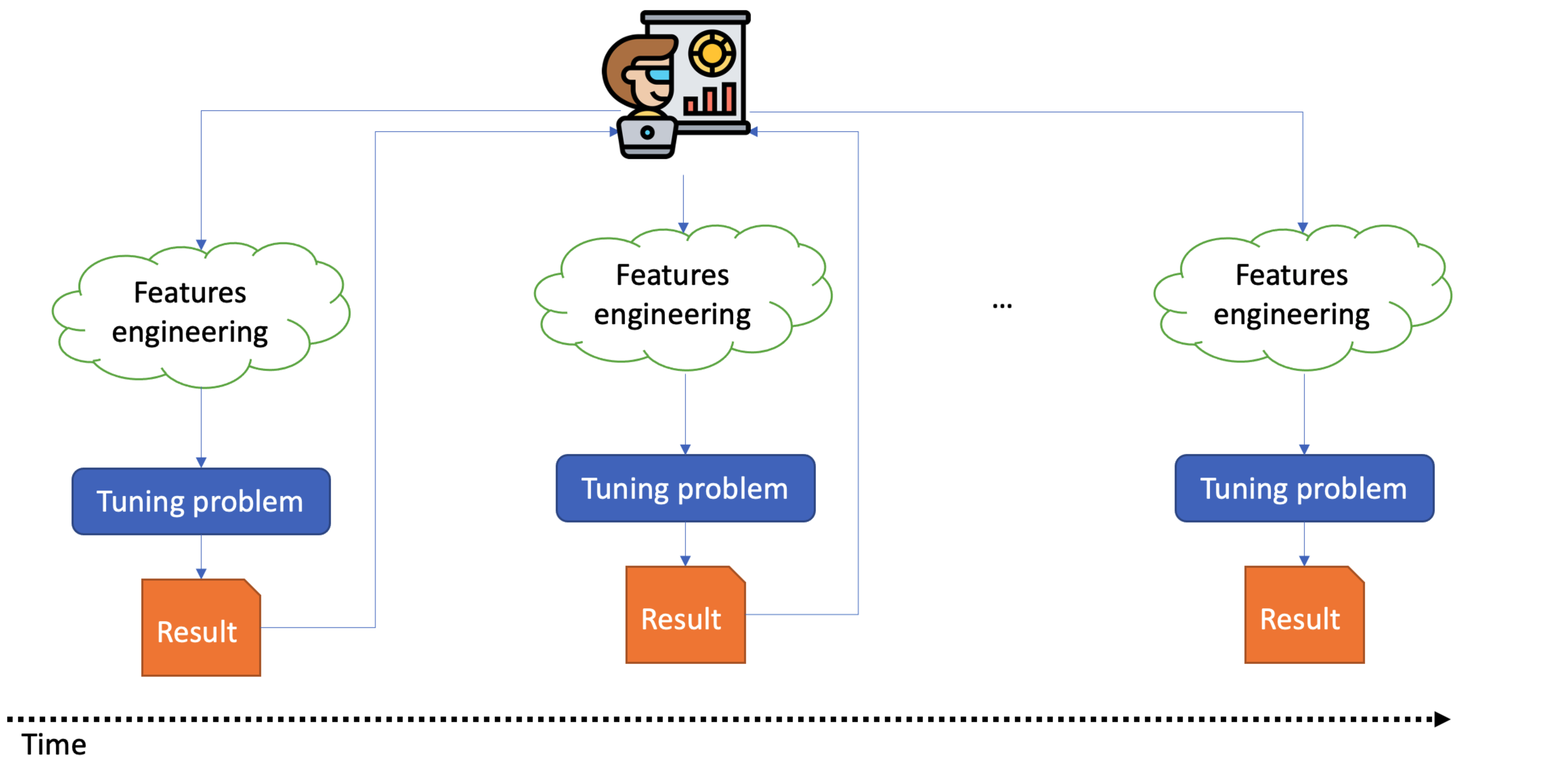
Giovanni Zappella, David Salinas, Cédric Archambeau

Amazon Web Services

Repeated HNAS

Data scientists do not create machine learning models in a single shot. They try different features, transformations, target variables, etc. During this procedure they often avoid using running HPO and NAS procedures due to their high cost and long execution time.

At the same time, hyperparameter configurations often perform similarly if the learning tasks on which they are used are similar. However, defining the similarity between learning tasks is far from trivial and it is often impossible for non-expert users.



Data scientist icon from Flaticon.com

Can we speedup HNAS when it is performed on a sequence of related tasks?

Setting

We formalize the problem as a sequence of "Best arm identification" (BAI) problems. Each hyperparameter configuration (or architecture) is an arm in the bandit problem and the goal of the learner is to identify the one with the highest reward.
Every BAI problem has an optimal arm and we assume that the arms previously identified as optimal will be quickly outperformed by other configurations if they are not the best arm for the problem at hand.

Our Solution: RUSH

Input: η (halving hyper-parameter), B (budget)
$|A_0^*| \leftarrow \emptyset$
$s \leftarrow 0$
while a new task is available **do**
  $A_s^{new} \leftarrow$ set of new arms
  $A_s^0 \leftarrow A_s^{new} \cup A_s^*$
  $n \leftarrow |A_s^0|$
  for $k = 0, \dots, \lceil \log_\eta n \rceil - 1$ **do**
    $\forall a \in A_s^k$, pull it $\left\lfloor \frac{B}{\max(1, \lfloor n/\eta^{k+1} \rfloor) \lceil \log_\eta n \rceil} \right\rfloor$ times
    $\forall a$, $r_a \leftarrow$ position of a in ranking by loss
    $r^* \leftarrow \min(r_i), \forall i \in A_s^*$
    $A_s^{k+1} =$
    $\{i \in A_s^k : r_i < \max(\min(r^* + 1, \lfloor n/\eta^{k+1} \rfloor), 1)\}$
  end for
  $\tilde{a} \leftarrow$ best arm from $A_s^{\lceil \log_\eta n \rceil - 1}$
  if $\tilde{a} \notin A_s^*$ **then**
    $A_{s+1}^* \leftarrow A_s^* \cup \{\tilde{a}\}$
  end if
  $s \leftarrow s + 1$
end while

Guarantees

Theorem 1 *If the budget B provided to the algorithm for each step of the sequence $1, \dots, S$ is larger than $\lceil \log n \rceil \max\left(2n + \sum_{a=2,\dots,n} \bar{\gamma}^{-1}(\Delta_a/2), zn\right)$ then RUSH will correctly identify the best arm.*

We can guarantee that when the budget is "large enough", RUSH will identify the best arm. (see the paper for all the details).

Tuning Tasks

We evaluate the algorithms on NAS tasks concatenating in a sequence the tuning tasks from FCNET and NAS201 and also HPO tasks by tuning an XGBoost classifier on a sequence of "similar" datasets. To crate datasets similar to each other, we pre-process the same dataset with different features transformers and encoders.
All sequences are formed by 20 tuning tasks and the results are averaged over 25 runs with different seeds (and tasks permutations).

Experimental Results: Predictive Performance

As first step, we would like to verify that our algorithm can identify configurations leading to models with competitive predictive performance.

Dataset	BALMIX	BANK	CCDEFAULT	COVTYPE	FCNET	NAS201
SH-Autorange(BB)	0.630	0.691	0.950	0.181	0.077	–
HB-Autorange(BB)	0.611	0.666	0.932	0.174	0.075	–
RUSH	0.546	0.693	0.978	0.209	0.079	31.897
SH	0.553	0.698	0.980	0.210	0.080	32.433
RUSH-HB	0.537	0.678	0.968	0.186	0.075	31.355
HB	0.542	0.691	0.978	0.186	0.078	31.908

Table 1: Average prediction performance obtained by different optimizers. For BANK and CCDEFAULT we report 1-F1, for COVTYPE we report 1-F1micro, for FCNET and NAS201 the prediction error. In all cases lower is better.

Experimental Results: Number of Evaluations

Since RUSH performs an aggressive pruning by leveraging the optimal configurations identified over the sequence, it save resources by performing less evaluations at higher resource levels.

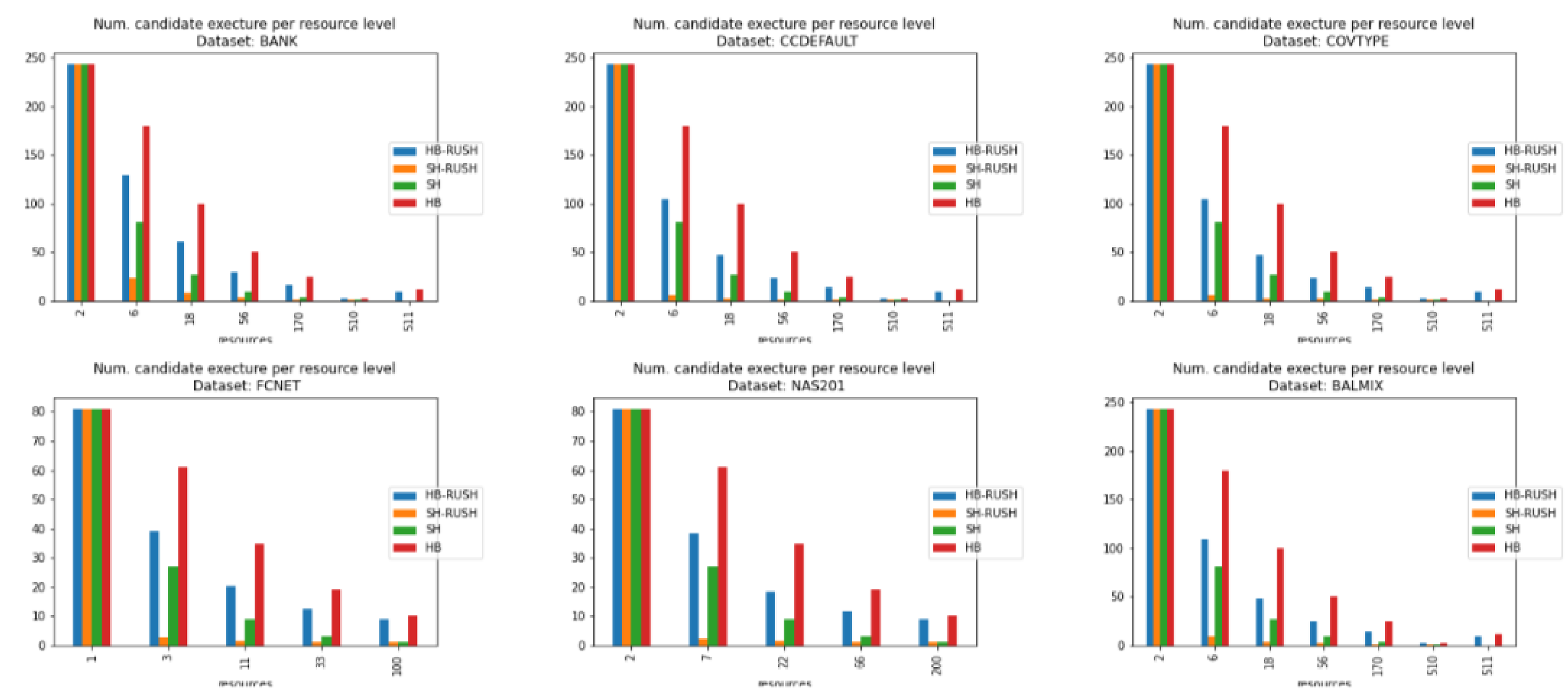


Figure 1: Number of evaluated candidates per resources level. Lower is better.

Experimental Results: Time Gained

Our algorithm does not explicitely account for the different cost associated to different hyperparameters configurations. However, the total time necessary for the tuning is related to the waiting time and often, especially in cloud environments, to cost.
In the following table we report the time gained (in percentage) by using RUSH instead of Successive Halving and Hyperband.

Dataset	RUSH vs SH	HB-RUSH vs HB
BALMIX	41.044	18.336
BANK	34.465	17.345
CCDEFAULT	45.627	18.646
COVTYPE	45.932	26.941
FCNET	0.587	0.657
NAS201	48.547	19.357

References

References

[1] Kevin Jamieson et al. "Non-stochastic best arm identification and hyperparameter optimization". In "Artificial Intelligence and Statistics", pp. 240–248. 2016.
[2] Zohar Karnin et al. "Almost optimal exploration in multi-armed bandits". In "International Conference on Machine Learning", pp. 1238–1246. 2013.
[3] Valerio Perrone et al. "Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning". In "Advances in Neural Information Processing Systems", pp. 12771–12781. 2019.
[4] Danny Stoll et al. "Hyperparameter Transfer Across Developer Adjustments", 2020.