Meta-learning from Learning Curves: Challenge Design and Baseline Results

Manh Hung Nguyen, Lisheng Sun-Hosoya, Nathan Grinsztajn, Isabelle Guyon

Meta-learning from Learning Curves: Challenge Design and Baseline Results

Manh Hung Nguyen
Chalearn
California, USA
0000-0001-7342-6430

Lisheng Sun-Hosoya
Chalearn
California, USA
0000-0002-5335-1547

Nathan Grinsztajn
Inria, Univ. Lille, CNRS
Lille, France
0000-0001-6817-5972

Isabelle Guyon
Univ. Paris-Saclay, Paris, France, and ChaLearn, California, USA
0000-0002-9266-1783

***Abstract*—Meta-learning has been widely studied and implemented in many Automated Machine Learning systems to improve the process of selecting and training Machine Learning models for new tasks, by leveraging expertise acquired on previously observed tasks. We design a novel meta-learning challenge aiming at learning-to-learn from one of the most essential model evaluation data, the learning curve. It consists of multiple model evaluations collected during the process of training. A meta-learner is expected to apply a learned policy to learning curves of partially trained models on the task at hand, to rapidly find the best task solution, without training all potential models to convergence. This implies learning the exploration-exploitation trade-off. Our challenge is split into two phases: a development phase and a final test phase. In each phase, a meta-learner is meta-trained and meta-tested on validation learning curves (development phase) or test learning curves (final test phase). During meta-training, the meta-learner is allowed to learn from the provided learning curves in any possible way. In meta-testing, we borrowed the common Reinforcement Learning setting in which an agent (a meta-learner) learns by interacting with an environment storing pre-computed learning curves. A meta-learner must pay a cost (corresponding to the actual training and testing time) to reveal learning curve information progressively. The meta-learner is evaluated and ranked based on the average area under its learning curves. This challenge was accepted as part of the official selection of WCCI 2022 competitions.**

***Index Terms*—AutoML, machine learning, meta-learning, learning curves, learning to learn, reinforcement learning.**

I. Introduction

Artificial learning systems are good at learning to solve mono-task problems, e.g. recognizing faces, playing video games, driving cars, translating languages, and assisting doctors to diagnose diseases. However, they are not yet capable of performing a wide diversity of tasks, unlike intelligent living beings. Learning from scratch every new task is obviously wasteful of computational resources. There is hope that leveraging experience from past learned tasks may both accelerate the learning process and yield better performance on new tasks. This is currently an active area of research, encompassing transfer learning [37], representation learning [3], few-shot learning [34], life-long learning [6], continual learning [7], and meta-learning [33] to name a few.

Meta-learning regroups a wide variety of techniques permitting learning systems to capitalize on the experience gained on previous tasks to train models on a new task faster, with fewer examples, and possibly with better performance. Approaches include learning from **algorithm evaluations** (e.g. accuracy scores), from **task and/or model properties** (e.g. dataset meta-features, model hyper-parameters), and from **model priors** (e.g. pre-trained model parameters) [33]. In particular, methods based on **algorithm evaluation**, which inspire this paper have been widely and efficiently exploited in meta-learning. One may simply select the algorithm that performed best on previously seen datasets, e.g. based on the average rank [1], [18]. Some approaches use such data to build recommender systems [10], [21], [22], [28], [30], [36]. REVEAL [29] explored a very different direction by formulating the meta-learning problem as a special Reinforcement Learning (RL) problem and solved it using existing RL methods. However, the aforementioned methods require evaluating fully trained models, a limitation that we address in this paper.

In our proposed setting, we consider multiple evaluations of the learning process, more specifically, the **algorithm learning curve**. A learning curve evaluates an algorithm's incremental performance improvements, as a function of training cycles (e.g. a number of iterations over the training set or epochs, number of examples, or simply wall time). It is an important tool that has been utilized for making decisions in supervised machine learning [24], such as early stopping or early discarding. Some existing learning-curve-based performance predicting methods, such as, [14]–[16], use "hard-coded" policies relying on pairwise comparisons of algorithm learning curves, which do not scale well. Our setting pushes that idea one step further: we want to stimulate the community to produce meta-learners capable of delivering **trained policies**.

In the "meta-learning from learning curves" setting, **meta-learners** should learn to optimize time management: rather than waiting until all potential algorithms are fully trained to be evaluated, they should interrupt training and take advantage of the information on partially trained algorithms (i.e. partial learning curves) to avoid wasting time on less promising algorithms. This should speed up the algorithm selection process for the given task at hand [8], [23], [31]. In other words, we allow a meta-learner to switch between the learning curves of algorithms. It actively requests to train and test algorithms to reveal their performance on a given dataset, which implies an "active meta-learning" setting. Besides, the meta-learner should learn to spend smartly and efficiently a

given time budget $\mathcal{T}$ for training algorithms. If it gives an insufficient amount of time to an algorithm, it may waste it as the algorithm training process might be interrupted before producing a new evaluation score.

The angle that we are taking to stimulate the community to work on this problem is to organize a challenge. This challenge is the third of the meta-learning challenge series (https://metalearning.chalearn.org) of ChaLearn [2], and was officially selected as part of the competition program of WCCI 2022. Its goal is to develop meta-learning algorithms (meta-learning agents) that can leverage learning curve information of partially trained algorithms, hence reducing the wasteful time of training them to convergence. Our challenge was much inspired by MetaREVEAL [25], a reinforcement learning-based meta-learning from learning curves method, which combines the following three ideas: (i) "active meta-learning" implemented in ActivMetal [30], (ii) formulating meta-learning as a special Reinforcement Learning problem in REVEAL [29], and (iii) exploiting partial learning curve information in Freeze-Thaw Bayesian Optimization [31]. In contrast to allocating a fixed-time budget Δt in each step in MetaREVEAL, in our challenge, meta-learners must learn a "smart" policy to allocate sufficient Δt to each algorithm in order to be successful. This implicitly increases the difficulty of the challenge, as the meta-learner needs to learn to solve two problems at the same time: **algorithm selection** and **budget allocation**.

To facilitate running the challenge, we created a benchmark dataset consisting of pre-computed learning curves as a function of time. Though actual algorithm training and testing are not conducted during the challenge, meta-learners must pay a cost corresponding to the actual computational time, for revealing their performance. Hence, meta-learners are expected to learn the trade-offs (see Figure 1) between:

- *exploitation* = continuing "training" an already tried good candidate algorithm (requesting its next learning curve point) and

- *exploration* = checking new candidate algorithms (asking its first learning curve point).

Although our challenge focuses on meta-learning from learning curves, we provide meta-features of datasets and hyperparameters of algorithms to offer more possibilities for meta-learning to a wide range of methods.

II. Challenge Meta-datasets

A. Real-world meta-dataset

We created a meta-dataset of pre-computed learning curves by running 20 algorithms with different hyperparameters on 30 cross-domain datasets[1] in the AutoML challenge [11]. The descriptions of these datasets can be found in [11]. Each algorithm is either a Random Forest (RF) [5] or a Gradient Boosting (GB) [9] algorithm, using implementations provided by Scikit-learn [26]. As the number of features considered when looking for the best split is one of the most important hyperparameters of RF and GB [32], we vary their *"max_feature"* hyperparameters to provide a hyperparameter searching space to meta-learners.

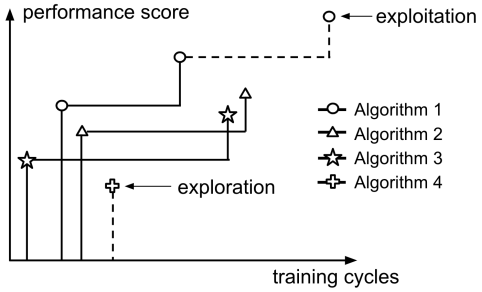


Fig. 1: Exploration-exploitation trade-offs in meta-learning from learning curves.

We also include meta-features that describe well the characteristics of each dataset, such as learning task, metric, time budget, etc. Detailed descriptions of the meta-features can be found in [11].

We respected the data splits of the AutoML challenge into a training set, a validation set, and a test set, for each dataset. We trained on the training set and produced 2 learning curves: one using the validation set and the other using the test set. Consistent with the AutoML challenge setting, the learning curve points on the validation and test sets are time-synchronous, but irregularly spaced because they are chosen by the learning algorithms themselves. Thus the time intervals between two points on a learning curve may vary.

In total, 1200 learning curves were included in our meta-dataset, some of which can be seen in Figure 2. The learning curves cross each other indicating that algorithm ranking on this dataset changes over training time. In addition, the algorithm ranking also varies across datasets, suggesting that a successful meta-learner should learn a dataset-dependent algorithm searching policy that can adapt to a new dataset.

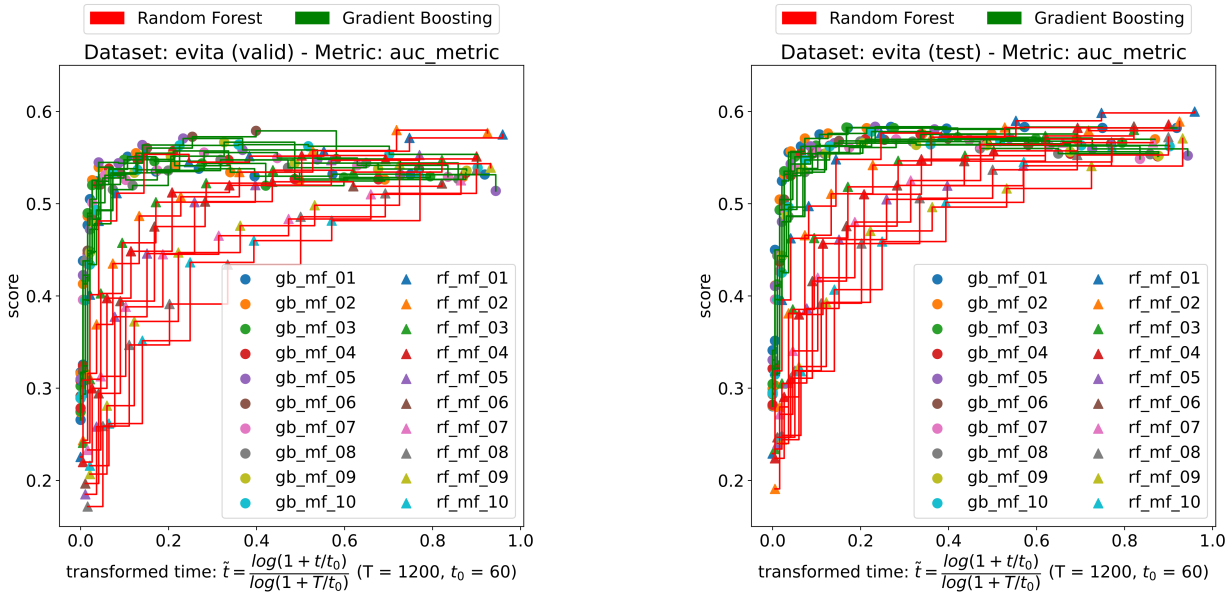
B. Sample meta-dataset

Due to the limited number of real-world (AutoML) datasets (30 datasets) in the AutoML challenge [11], we want to save all of them for evaluating participants' solutions on the challenge platform. We thus synthetically generated learning curves of 20 algorithms on 100 artificial datasets[2], which are used for practice purposes only. The synthetic data is included in the starting kit [3] provided to the participants for

[1]The application fields include medical diagnosis, speech recognition, classification of text, prediction of customer satisfaction, object recognition, etc. The datasets have been preprocessed in suitable fixed-length vectorial representations.

[2]The synthetic learning curves are not included in the meta-dataset used for testing and ranking participants in our challenge. Performing well on the sample (synthetic) meta-dataset does not guarantee that the agent performs well on the real-world meta-dataset in the challenge.

[3]A starting kit consists of necessary materials and instructions to develop your own agent and test it locally in the same way used in the challenge, which includes: sample meta-dataset, sample submissions, ingestion program, and scoring program.



(a) VALIDATION learning curves

(b) TEST learning curves

Fig. 2: **Learning curve samples of our real-world meta-dataset.** They are obtained from the validation set (Figure 2a) and the test set (Figure 2b) of a dataset named "evita". In the legend inside the figures, the first two characters of the algorithm name indicate the algorithm family: Random Forest (rf) or Gradient Boosting (gb). The rest indicates the value of the hyperparameter "max_features". For examples, "gb_mf_01" corresponds to a Gradient Boosting algorithm with "$max\_features$" $= 0.1$, while "rf_mf_10" corresponds to a Random Forest algorithm with "$max\_features$" $= 1.0$. The scores shown on the y-axis depend on the metric used in the dataset, in this case, are AUC (Area Under the ROC Curve) scores. T and t_0 are measured in seconds.

developing and testing their methods. The points on each synthetic learning curve are sampled from a sigmoid function parameterized by three parameters a, b and c as follows:

$$lc(x) = \frac{a}{1 + e^{-b(x-c)}} \tag{1}$$

Using parameterized sigmoid functions gives us abilities to experiment with learning curves of various shapes, by changing their asymptotic performance (determined by a), increasing rate (determined by b), and "warm-up" time (determined by c). Values of a, b, and c were generated from matrix factorizations. The purpose of using matrix factorization is to obtain matrices with some underlying structures indicating that some particular groups of algorithms work well on some groups of datasets. More details of this meta-dataset can be found in [25].

III. Challenge Protocol

A. Challenge phases

We devised a novel two-phase challenge protocol, using both validation and test learning curves to avoid overfitting test data, and k-fold meta-cross-validation to reduce variance in the evaluation:

- **Development phase**: participants submit agents (meta-learners) that will be meta-trained and meta-tested on the validation learning curves (see Figure 3).
- **Final test phase**: no further submissions are made in this phase. The last submitted agent of each participant in the Development phase is forwarded automatically to this phase. It is evaluated on the test learning curves, using the agent's prescriptions made with only the knowledge of partially revealed validation learning curves, following its policy (see Figure 5).

In each phase, the meta-dataset is split using the k-folds meta-cross-validation procedure, with k=6 (as illustrated in Figure 4). More precisely, in each iteration (split), 25 datasets are used for meta-training the agent and the rest (5 datasets) is used for meta-testing. The final results are averaged over the test folds.

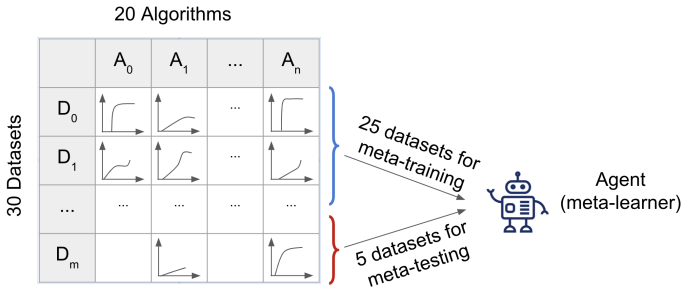


Fig. 4: K-folds meta-cross-validation, with k=6.

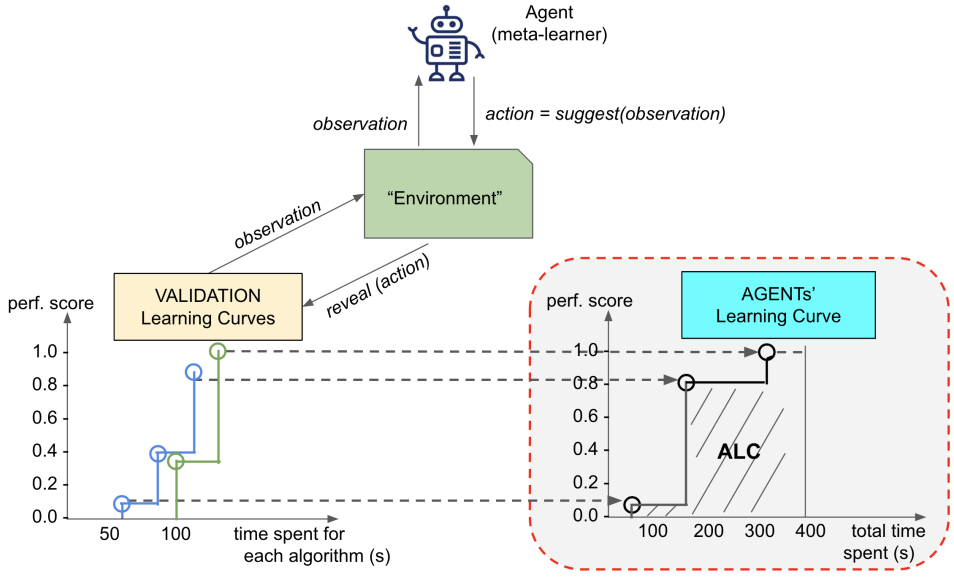


Fig. 3: **Development phase.** The submitted agent interacts back and forth with the environment to reveal validation learning curves progressively. The agents' learning curve is constructed based on the revealed validation learning curves.

B. Meta-training

During meta-training, 25 meta-training datasets are passed to the $meta\_train()$ function implemented in a participant's agent, including validation learning curves, test learning curves, meta-features of datasets, and hyperparameters of algorithms, for each dataset. The agent is free to learn from these data in any possible way.

C. Meta-testing

During meta-testing, an agent is presented with one dataset at a time. To avoid further meta-learning, the agent is reset to its original state after the meta-training phase, at the beginning of each test task. The agent then iteratively interacts with its "environment" (that can supply it with learning curves and meta-features of datasets and algorithms) in a Reinforcement Learning fashion (see Figure 5). In each step, the agent has 2 types of actions: suggesting to reveal a new value of a given algorithm's validation learning curve and choosing the current best performing algorithm. Then, the agent observes the learning curve point revealed to decide its next action. The agent's own validation and test learning curves are built from the agent's choice of best algorithms at each time step.

D. Agent-environment interface

We supplied sample code abiding with an interface that allows RL-style interactions between the agent and the environment (as demonstrated in Figure 3 and Figure 5).

First, an agent must implement its own meta-training procedure in the meta_train() function, using the provided meta-training data:

trained_agent = agent.*meta_train*(validation_learning_curves, test_learning_curves, dataset_meta_features, algorithm_hyperparameters)

Then, the trained agent starts interacting with the environment by suggesting actions and observing feedback from the environment:

action = trained_agent.*suggest*(observation)
observation, done = env.*reveal*(action)

where:

observation (state): includes the information of:

- A: index of the algorithm explored in the previous step
- C_A: a counter that keeps track of the amount of time has been spent for algorithm A
- $R\_validation\_C\_A$: the validation score of algorithm A achieved with respect to C_A

action: consists of three elements:

- A^*: index of the most promising partially trained algorithm that the agent found so far (for constructing the agents' learning curve)
- A: index of the algorithm to be explored next (for revealing the next point on its validation learning curve)
- Δt: amount of time to be spent for training algorithm A in this step

done: indicates whether the current episode is ended (i.e. when the agent exceeds the time budget $\mathcal{T}$ given for the dataset at hand). In each episode, the agent is meta-tested on one dataset at a time (corresponding to one row in Figure 4).

IV. Challenge Evaluation & Metrics

A. Evaluation

In this challenge, an agent is evaluated by accumulated rewards obtained on meta-testing datasets. The reward is given based on the Area under the agents' Learning Curve (ALC), which is computed from validation learning curves or test learning curves, depending on the challenge phase. The

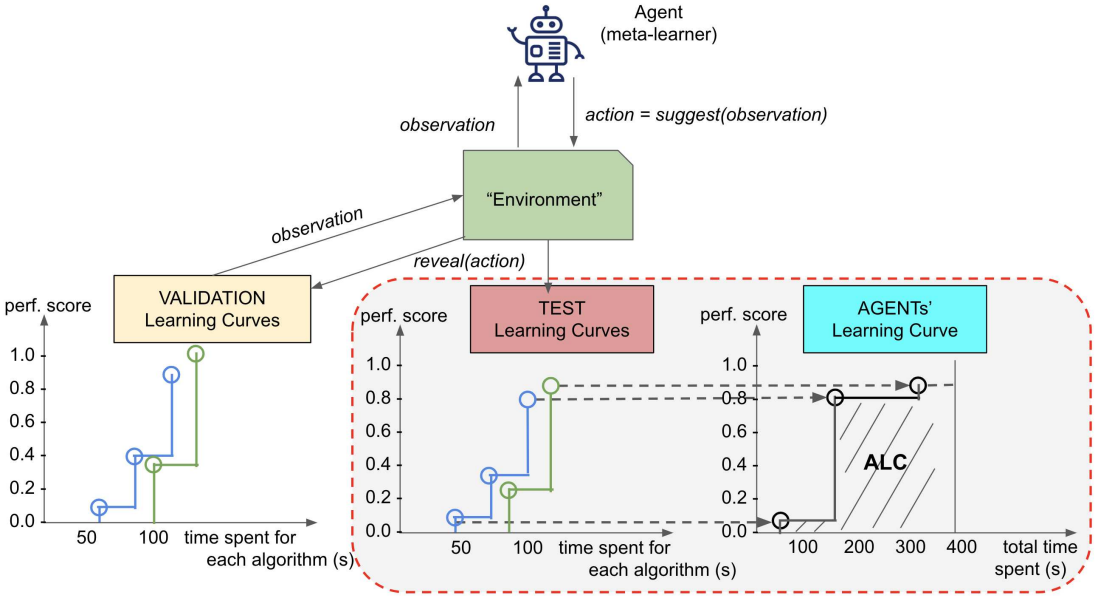


Fig. 5: **Final test phase.** The interaction between the agent and the environment is similar to the Development phase. However, the agents' learning curve is computed from test learning curves which are hidden from the agent.

computation of the ALC is explained in Section IV-B. Having separate validation sets and test sets of learning curves is novel in our protocol and is not common in usual Reinforcement Learning (RL) settings. RL agents are usually evaluated in the same environment where they are trained, which may lead to environment overfitting [35] and poor generalization [13]. Hence, our challenge protocol is designed to avoid these potential problems.

In the Development phase, the agents' learning curve is established from the results of the most promising algorithm chosen by the agent at each time step, using the validation scores (see Figure 3). The final score used for ranking on the leaderboard is the average accumulated reward obtained on the meta-testing datasets.

The protocol is similar in the Final test phase, except that, the agents' learning curve is built from test learning curves which are hidden during the Development phase to avoid overfitting the test data (see Figure 5). The predictions of which algorithm should perform best at each time step are used to compute the test learning curves on which they are evaluated.

B. Metrics

We are interested in agents with high "any-time learning" capacities, which means the ability to perform well if they were to be interrupted at any point in time. Hence, the agent is evaluated by the accumulated reward, which is equal to the ALC. This puts more emphasis on performance improvements at the beginning of an episode, taking inspiration from [19], [20]. The computation of the ALC is shown in Figure 6). The reward in each step is thus defined by:

$$r(t) = [R^*(t) - R^*(t - \Delta t)] \; [(\mathcal{T} - t)] \tag{2}$$

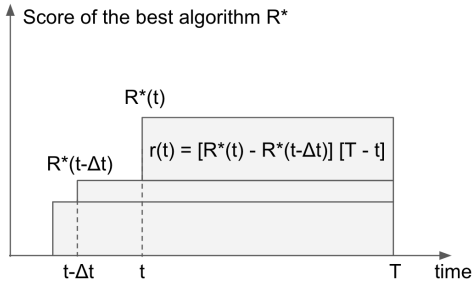


Fig. 6: **Computation of the Area under the Learning Curve (ALC).** It is carried out by integrating the learning curve using horizontal rectangles, in the style of Lebesgue integrals.

where $R^*(t)$ and $R^*(t - \Delta t)$ are the best algorithm performances achieved in this step and the previous step respectively:

$$R^*(t) = \max_{k \leq t} R(k) \; , \tag{3}$$

$$R^*(0) = 0 \; , \tag{4}$$

The x-axis is scaled logarithmically to stress more on the beginning performance, which makes the reward function become:

$$r(t) = [R^*(t) - R^*(t - \Delta t)] \; [(1 - \tilde{t})] \tag{5}$$

with the normalized time:

$$\tilde{t} = \frac{\log(1 + t/t_0)}{\log(1 + \mathcal{T}/t_0)} \tag{6}$$

The larger t_0 is, the more important the beginning of the learning curve is. In our challenge, t_0 is set to 60 seconds.

V. Baseline methods and results

A. Baseline methods

We implemented and tested various methods with simple to sophisticated exploration-exploitation strategies on our challenge, including: uniform-distribution-based (*Random Search* agent), sampling-based (*Best on Samples* agent), heuristic-based (*Freeze-Thaw Bayesian Optimization* agent), exploitation-only (*Average Rank* agent), RL-based (*Double Deep Q-Network* agent). We describe each of them in a subsection below.

1) Random Search: This is a simple baseline that performs a random search on the algorithm space. It randomly chooses an algorithm and assigns an arbitrary amount of time for training and testing the algorithm. As this baseline has a huge variance, we run it 5 times and report its average performance. This baseline is implemented only for comparison purposes. In a realistic setting, one would not average over several runs.

2) Best on Samples: An intuitive way of selecting algorithms for a new task is to try each of them with some samples first, then, select the algorithm that performed best on the samples to run on the entire dataset [27]. We adapted it to make a baseline, by selecting the algorithm that performed best within a fixed (and small) amount of time $t_{sampling}$ instead. More specifically, at the beginning of each episode, it trains every algorithm with the same amount of time $\Delta t = t_{sampling}$ (we set $t_{sampling} = 0.02 \times \mathcal{T}$). Based on the observed results, it selects the one that achieved the highest evaluation score to spend the rest of the given time budget: $\Delta t = \mathcal{T} - (|A| \times t_{sampling})$, with $|A|$ is the number of algorithms.

3) Freeze-Thaw Bayesian Optimization [31]: In a meta-learning setting where a hyperparameter space is available, one may use the standard Bayesian Optimization to quickly and efficiently search for good hyper-parameters. Freeze-Thaw successfully applied Bayesian Optimization to find the best model among a set of "frozen" models (with different sets of hyperparameters) that are partially trained. At the beginning, it tries each algorithm with a small amount of time Δt to quickly explore the algorithm performance space (we set $\Delta t = 0.025 \times \mathcal{T}$). Then it leverages the partial learning curve information in a Bayesian Optimization fashion to decide which ones to "thaw" and resume training it. By doing so, it avoids wasting too much time on not promising models and focuses on high potential models. In our experiments, this is the only method that uses provided algorithm hyperparameter space to perform its searching strategy. However, it does not learn a policy from experience.

4) Average Rank: Another simple and natural idea to select algorithms for a new dataset is based on past algorithm rankings [1], [4], [17], [18], which motivated us to build an Average Rank baseline. In meta-training, it constructs a global average ranking of algorithms from the terminal performance scores of algorithms on datasets. The rank of each algorithm A_j is defined by:

$$global\_rank(A_j) = \frac{\sum_{i=1}^{D_{meta-train}} rank_j^i}{D_{meta-train}} \quad (7)$$

where $D_{meta-train}$ is the total number of meta-training datasets, and $rank_j^i$ is the rank of algorithm A_j on the dataset D_i. In meta-testing, only the algorithm having the highest global rank is chosen to run on the dataset at hand with the entire given time budget: $\Delta t = \mathcal{T}$. In a real-life scenario, running this baseline is very time-consuming as it requires training and testing the whole set of algorithms on all meta-training datasets to have a complete ranking.

5) Double Deep Q-Network [12]: As the meta-testing procedure is designed with inspiration from RL, using an RL method as a baseline is essential. Double Deep Q-Network (DDQN) is a well-known and easy to be implemented RL method. We designed a specific environment based on the one from the meta-testing phase to meta-train this DDQN agent. During the meta-testing phase, the agent uses its learned policy to explore and exploit algorithms. For allocating the given time budget, it keeps track of the amount of time that has been spent for each algorithm $C(A)$, and doubles it every time it re-selects the algorithm: $\Delta t = 2 \times C(A)$.

B. Baseline results

We ran baseline methods on our real-world meta-dataset in the Final test phase and report the detailed results in Table I. In the Final test phase, an agent is evaluated by its ALC computed from test learning curves of algorithms on datasets, hidden from the agent.

The experiments indicate the overall superiority of **DDQN** to other baselines, which not only has the highest average ALC score (0.38), but also reaches the best ALC score on 22 out of 30 datasets among the baselines. We attribute this superiority to the fact that it leverages meta-learning from learning curve data of previous tasks. Remarkably, **DDQN** achieved a relatively good score on dataset 'macro' compared to other techniques, which failed to select an algorithm returning any result within the given time budget. Another agent that learns from experience is the **Average Rank** agent. However, it learns a policy that acts identically for all datasets. This exploitation-only strategy does not work well in our challenge, as **Average Rank** obtained an average ALC score of only 0.23.

In contrast, both the **Freeze-Thaw** agent and the **Best on Samples** agent rely only on partially revealed learning curve information on the dataset at hand, not leveraging prior tasks. Although both of them achieved almost the same average ALC scores (0.33 and 0.34 respectively), the latter won more times than the former. The relatively good performance of the Best on Samples agent can be explained by the fact that the algorithm selection policy is based on time-dependent performances of algorithms, which requires tuning the $t_{sampling}$ hyperparameter.

The **Random Search** agent is at the bottom of the table with an extremely low average ALC score of 0.05. This confirms the need for more elaborate policies to select suitable

algorithms and distribute the time budget for a given dataset, which is the ultimate goal of our challenge.

C. Summary of the challenge results

We briefly present the results of the top 5 approaches, with comparisons to our baselines in Table I. Our analysis of methods employed revealed that it is more effective to learn policies (with reinforcement learning) than rely on hard-coded policies. This holds both for choosing algorithms and spending wisely the time budget, as done by teams **MoRiHa** and **neptune**. In contrast, using pre-defined lists of Δt, as done by **AIpert** and our **DDQN** baseline, did not yield as good results. Team **MoRiHa**, which finished in 1st place, achieved the highest average ALC score of 0.43 and the best score on 21 out of 30 datasets. More in-depth analyses will be made available in our technical report [4].

VI. Conclusion

We present the design of the first challenge on meta-learning from learning curves. The challenge protocol is implemented based on the common RL paradigm in which a meta-learner iteratively selects an algorithm to be "trained" and reveals more information on the chosen algorithms' learning curve. To encourage participants to develop meta-learners with high "any-time learning" capacities, we use the ALC metric to evaluate and rank the meta-learners. Our initial baseline results indicate that **DDQN**, an RL method that can meta-learn from previously seen learning curves, has advantages over other baseline methods. We compared against simpler baselines that either *did not rely on past experience* (no meta-learning), but used the performance of *all* algorithms on a sub-sample of the dataset at hand to make their choice (Best on samples) and methods that *did rely on past experience* but *indiscriminately of datasets* (Average Rank). Since neither performed well, we can see that the advantage of **DDQN** is to learn smart policies from past experience including tactics that are specific to given algorithms. The effectiveness of this strategy has been further confirmed by the results of the top-ranked teams in our challenge.

We are preparing of more in-depth analysis of this challenge, which we will make available shortly on our website https://metalearning.chalearn.org/. From the lessons learned from this challenge, we prepared a second round (in conjunction with AutoML-Conf 2022), currently on-going as of the publication of this paper. It includes larger meta-datasets and some modifications of the challenge protocol suggested by participants. After the second round terminates, we intend to prepare a more complete analysis, including ablation studies to uncover the critical ingredients of the most successful agents.

Acknowledgment

We would like to thank beta-testers for providing feedback, including Ihsan Ullah and Adrien Pavao. We also appreciate the help from the AutoML organizers to create our meta-dataset. We want to thank Chalearn for donating prizes. We are grateful to the Codalab staff for helping with the implementation, and particularly to Adrien Pavao. We also wish to thank Google for supporting us with computing resources. This work was supported by ChaLearn, the ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022, and TAILOR EU Horizon 2020 grant 952215.

[4] https://metalearning.chalearn.org/

References

[1] S. Abdulrahman, P. Brazdil, J. van Rijn, and J. Vanschoren, "Speeding up algorithm selection using average ranking and active testing by introducing runtime," *Machine Learning*, vol. 107, 01 2018.

[2] A. E. Baz, I. Guyon, Z. Liu, J. van Rijn, S. Treguer, and J. Vanschoren, "Advances in metadl: Aaai 2021 challenge and workshop," *arXiv preprint arXiv:2202.01890*, 2022.

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1798–1828, 08 2013.

[4] P. B. Brazdil and C. Soares, "A comparison of ranking methods for classification algorithm selection," in *Machine Learning: ECML 2000*, R. López de Mántaras and E. Plaza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 63–75.

[5] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2004.

[6] Z. Chen, B. Liu, R. Brachman, P. Stone, and F. Rossi, *Lifelong Machine Learning*, 2nd ed. Morgan amp; Claypool Publishers, 2018.

[7] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[8] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *IJCAI*, 2015.

[9] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001.

[10] N. Fusi, R. Sheth, and M. Elibol, "Probabilistic matrix factorization for automated machine learning," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[11] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, and E. Viegas, *Analysis of the AutoML Challenge Series 2015–2018*. Cham: Springer International Publishing, 2019, pp. 177–219. [Online]. Available: https://doi.org/10.1007/978-3-030-05318-5_10

[12] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[13] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning," 2022.

[14] R. Leite and P. Brazdil, "Predicting relative performance of classifiers from samples," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 497–503. [Online]. Available: https://doi.org/10.1145/1102351.1102414

[15] ——, "An iterative process for building learning curves and predicting relative performance of classifiers," in *Proceedings of the Aritficial Intelligence 13th Portuguese Conference on Progress in Artificial Intelligence*, ser. EPIA'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 87–98.

[16] ——, "Active testing strategy to predict the best classification algorithm via sampling and metalearning." NLD: IOS Press, 2010, p. 309–314.

[17] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," vol. 7376, 07 2012, pp. 117–131.

[18] S. Lin, "Rank aggregation methods," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, pp. 555 – 570, 09 2010.

[19] Z. Liu, A. Pavao, Z. Xu, S. Escalera, F. Ferreira, I. Guyon, S. Hong, F. Hutter, R. Ji, T. Nierhoff, K. Niu, C. Pan, D. Stoll, S. Treguer, J. Wang, P. Wang, C. Wu, and Y. Xiong, "Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 17, 2020.

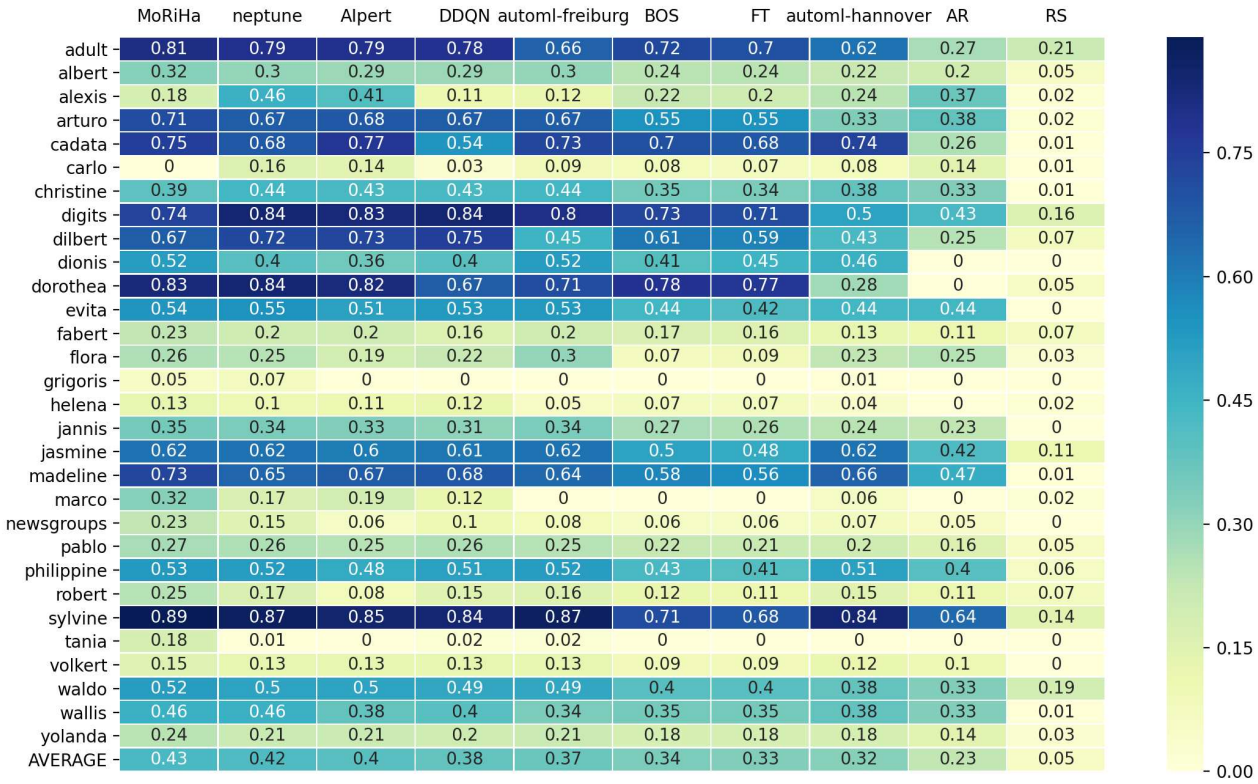


TABLE I: **ALC scores of the top 5 methods: MoRiHa, neptune, AIpert, automl-freiburg, automl-hannover, and our baselines: Double Deep Q Network (DDQN), Best on Samples (BOS), Freeze-Thaw BO (FT), Average Rank (AR), Random Search (RS)**. The reported scores correspond to the worst of 3 runs for each method. The last row shows the average ALC scores (in descending order, from left to right) over 30 datasets.

[20] Z. Liu, Z. Xu, S. Rajaa, M. Madadi, J. C. S. J. Junior, S. Escalera, A. Pavao, S. Treguer, W.-W. Tu, and I. Guyon, "Towards automated deep learning: Analysis of the autodl challenge series 2019," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, H. J. Escalante and R. Hadsell, Eds., vol. 123. PMLR, 08–14 Dec 2020, pp. 242–252. [Online]. Available: http://proceedings.mlr.press/v123/liu20a.html

[21] M. Misir and Sebag, "Algorithm selection as a collaborative filtering problem," Research Report, Dec. 2013. [Online]. Available: https://hal.inria.fr/hal-00922840

[22] ——, "Alors: An algorithm recommender system," *Artif. Intell.*, vol. 244, pp. 291–314, 2017.

[23] F. Mohr and J. N. van Rijn, "Fast and informative model selection using learning curve cross-validation," 2021.

[24] ——, "Learning curves for decision making in supervised machine learning – a survey," 2022.

[25] M. H. Nguyen, N. Grinsztajn, I. Guyon, and L. Sun-Hosoya, "Metareveal: Rl-based meta-learning from learning curves," in *Workshop on Interactive Adaptive Learning co-located with European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2021)*, Bilbao/Virtual, Spain, Sep. 2021. [Online]. Available: https://hal.inria.fr/hal-03502358

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] J. Petrak, "Fast subsampling performance estimates for classification algorithm selection," in *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 2000, pp. 3–14.

[28] D. Stern, H. Samulowitz, R. Herbrich, T. Graepel, L. Pulina, and A. Tacchella, "Collaborative expert portfolio management," vol. 1, 12 2010.

[29] L. Sun-Hosoya, "Meta-Learning as a Markov Decision Process," Theses, Université Paris Saclay (COmUE), Dec. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/tel-02422144

[30] L. Sun-Hosoya, I. Guyon, and M. Sebag, "Activmetal: Algorithm recommendation with active meta learning," in *IAL@PKDD/ECML*, 2018.

[31] K. Swersky, J. Snoek, and R. Adams, "Freeze-thaw bayesian optimization," 06 2014.

[32] J. N. van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2367–2376. [Online]. Available: https://doi.org/10.1145/3219819.3220058

[33] J. Vanschoren, "Meta-learning: A survey," 10 2018.

[34] Y. Wang, Q. Yao, J. Kwok, and L. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys*, vol. 53, pp. 1–34, 06 2020.

[35] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone, "Protecting against evaluation overfitting in empirical reinforcement learning," *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 120–127, 2011.

[36] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell, "Oboe: Collaborative filtering for automl model selection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1173–1183. [Online]. Available: https://doi.org/10.1145/3292500.3330909

[37] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. PP, pp. 1–34, 07 2020.