Meta-learning from Learning Curves Challenge 2022
Fact sheet template

AIpert

March 2022

1 Team details

- **Team name *:** AIpert
- **Team leader name *:** Giorgia Franchini
- **Team affiliation *:** UNIMORE University of Modena and Reggio Emilia
- **Team leader address *:** Via Campi, 213/B 41125 Modena (MO)
- **Team leader phone number *:** 059 205 5243
- **Team leader email *:** giorgia.franchini@unimore.it
- **Name of other team members (if any)** Elena Govi, Davide Sapienza, Carmelo Scribano
- **Team website URL (if any):**

2 Contribution details

- **Title of the contribution:** Q-learning fixed-time agent
- **Summary ***
  The proposed strategy is based on uncovering good algorithms as fast as possible. The strength of the approach lies on the low-computational cost of the process and on its simplicity. Moreover, different simple tools are combined together in an unusual, but performing way. In a Reinforcement Learning framework different Q-learning matrices are computed during the meta-learning process, one for each cluster. The clustering is performed using the dataset features.
- **Motivation**
  The common purpose of meta-learning is to create a learning algorithm over multiple learning episodes, able at *learning to learn*. Particularly, in this challenge, the strategy had to consider not only the best performance-algorithms, but also the algorithm-efficiency and

how much time they spent reaching that performance. There exists many meta-learning settings, in this particular case meta-data are represented as Model Evaluations. The general purpose is to rapidly find an algorithm that performs best on an unknown dataset. An agent actively requests to train and test algorithms and an environment reveals their performances scores on a given dataset. This approach can be divided into two principal phases. During the first phase a clustering is computed by a K-means model. We had 16 features which describes each dataset and 2 features representing each algorithm. K-means model, with $K = 12$, divided our meta-dataset in 12 groups created by computing distances between dataset's features. There were different types of features: categorical or numerical, string or number. For this reason a scaling phase was needed before clustering. After the Standard Scaler performing, the meta-training dataset was divided into twelve groups and a clustering model was created in order to predict which group a new scaled dataset belongs to.
This first phase comes from the assumption that similar datasets usually have the same learning-behaviour, as explained in [4], so recognizing groups of similar datasets is fundamental. As in [6], performance scores are usually preferred to dataset-features during the meta-training, because they are considered more significant. However, dataset features are available and could be an adding information's resource. In the second phase we used a strategy based on a standard Reinforcement Learning strategy: Q-learning. Since the focal point of the challenge is the exploration-exploitation trade-off, a RL agent able to make the right decision at each step is suitable. The ALC metric computed does not only ask for a good performance, but it also requires a fast good performance score.
The K Q matrices are $n \times m$ matrices where n is the number of states (portion of the time budget) and m is the number of actions (different algorithms). Once the matrices have been computed, they are able to predict the best action given the state. Before computing it, with the standard Q-learning algorithm, [8], [5], [1], some elements had to be defined: state and reward. The state was represented by the time-step. Available time-steps, in the meta-dataset, were not fixed and were not based on a strategy, distribution or any kind of mathematical model, but at the same moment the time choice reveals to be really important for the final ALC score. In an initial attempt a Random Forest Regressor was implemented in order to predict the right time-step request, but it was not enough. After many experiments, we fixed time-step portions and we used them for each dataset. These portions alternates longer and shorter portion-values, starting from very little steps. Percentages of the total time budget are more significant than the absolute time value, because not to loose a large amount of time-budget is fundamental, especially at the beginning. We observed that a logaritmic spaced vector performs better than a linear spaced one. The reward chosen was inspired by the paper [6]:

$$r(t) = [V^*(t) - V^*(t - \Delta t)][(\tau - t)]$$

where $V^*(t) = \max_{k<=t} V(k)$ and V is the vector which represents the current test score of each algorithm in the current episode and τ corresponds to the total time budget. In our method the time-based part was irrelevant because time percentages are fixed at the beginning. The RL agent is not supposed to choose also the time-step, therefore the time is not needed in the reward. The resulting reward is then:

$$r(t) = [V^*(t) - V^*(t - \Delta t)]$$

Once state and reward are defined, the Q learning matrix can be computed and should converge. The Q *updating* is done only during the meta-training, which is an exploration phase. In fact, different algorithms are explored with very high randomness. For each dataset, given a state, an action is predicted with the ϵ-greedy strategy, according to [1]:

$$\begin{cases} random \quad if \quad p < \epsilon \\ argmax_a Q(S,a) \quad if \quad p > \epsilon \end{cases}$$

After receiving the observation, the correspondent Q is updated with the reward R :

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma(\max_a Q(S,a)) - Q(S,A))$$

Moreover, parameters optimization is a central point of this algorithm. More details are explained in the **Detailed method description**.

During the meta-testing, the Q matrices computed in the meta-training phase are used and they don not change. The saved algorithm for each step is the one with the best performance score, obtained from the *validation last score* vector.

Another interesting method for choosing the best action given fixed time percentages is implemented with a performance predictor strategy [2, 3]. In particular, a Random Forest Regressor (RFR) was implemented and it was able to predict the future performance given dataset meta-features, algorithm meta-features and time. Samples of the dataset, for the RFR training phase, were vectors as the following: $[ds_0, ..., ds_{16}, alg_0, alg_1, time]$. During the meta-training, where performance scores are available without any constraint, the RFR is trained. 20% of the dataset, for the RFR, is saved for the RFR test computed with Mean Squared Error. The Mean Squared Error was very low: $1e-3$. This means that the performance predictor is very accurate and it could be a great resource for the best action. However, it has not a complete strategy but only a '*in-that-moment*' strategy. Therefore some errors were found, for example it insists on the same algorithm many times, even if it doesn't observe anything good. For these reasons the performance predictor idea was left behind. Despite this, the performance predictor could be a great resource if associated to a Reinforcement Learning Agent.

- **Contributions** *

  - Unusual and innovative combination of a off-policy Reinforcement Learning method and K-means clustering model. This choice allows us to extract information from the dataset meta-features, with clustering, and then to learn during the meta-training the best strategy, with the RL agent.
  - Time is not chosen as an absolute value, but it depends on the percentage that value represents with respect to the total time-budget and it is fixed. Time percentages or portions are chosen as logarithmic spaced vectors.

- In terms of computational complexity, our approach is really basic: at each step a single element of a tensor is updated. The absence of deep learning methods training is an advantage. Moreover pre-trained neural networks are not used. For that reason this code should enter in the context of GreenAI (Green Artificial Intelligence [7]).

- **Detailed method description** *

1. **Preprocessing.** Data meta-features are preprocessed and represented as numerical vectors. In particular some features represented as strings are linked to the correspondent class number. Then the *Standard Scaler* of the Sci-kit Learn package is used. It standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:
$$z = \frac{(x-u)}{s}$$
where u is the mean of the training samples. Mean and variance are stored and will be useful to scale dataset meta-features during the meta-testing.

2. **Data Characterization.** Given samples representing scaled meta-features, K-Means clustering (unsupervised learning) is applied with the standard euclidean metrics. K-Means is available in the same python package *sk-learn*. K-Means benefits from OpenMP based parallelism through Cython. Small chunks of data are processed in parallel, which in addition yields a low memory footprint.
Each dataset of the meta-training is labelled with a cluster and the centroids are saved in order to label also dataset from the meta-testing phase.

3. **Q-computation.** The Q-learning matrix is a 3-dimensional tensor $c \times s \times a$ where c is the number of clusters, s is the number of states and a is the number of available actions (i.e. algorithms). 5000 is the number of iteration chosen. Each dataset of the meta-training is seen 5000 times and the correspondent Q-learning matrix is updated. Overfitting is not a problem because even if the same dataset are seen each iteration, the observation and rewards continuously change, thanks to randomness. Actions are chosen with 20% of randomness, explained by the ϵ hyperparameter, and each time they give the reward
$$r(t) = [V^*(t) - V^*(t - \Delta t)]$$
, which will be used for the **Q-update** function. Other important parameters of a q-learning method, chosen for the **Q-update**, are α, the learning rate, with a value between 0 and 1, and γ, the discount factor. After many trials, the best performing hyperparameters combination is the following:
$$\alpha = 0.01 \quad \gamma = 0.7 \quad \epsilon = 0.2.$$
On this way the Q-learning tensor converges.

4. **Meta-Testing.** During the meta-testing phase a new dataset is considered. First, the Standard Scaler created in the preprocessing phase is used in order to preprocess dataset features. Secondly, the dataset is associated to one of the $K = 12$ clusters with the previous K-mean model. It explains the Q-learning matrix first dimension $Q(k, :, :)$, which

will remain the same for that dataset. At this point the agent makes different steps according to the time-portions vector of total time budget: $[0.0, 0.00269, 0.0061, 0.0106, 0.0163, 0.0235, 0.0327, 0.0445, 0.0595, 0.0785, 0.1028, 0.1338, 0.1732, 0.2235, 0.289, 0.3692, 0.39, 0.4732, 0.49, 0.6058, 0.69, 0.774, 0.84, 0.89, 0.96, 1.01]$. and choose for each step the best algorithm to reveal. It finds the

$$\max_{a \in N_A} Q(K, S, a) \quad \text{where} \quad N_A = \text{number of actions}$$

and choose that index-value as action.

- **Representative image / workflow diagram of the method** *

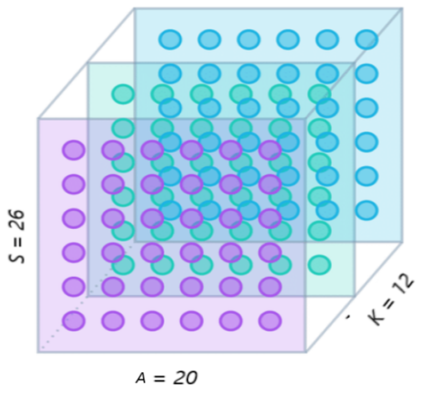

Figure 1: 3D Q-learning matrix. K represents the number of clusters, A represents the number of actions (or the algorithms) and S represents the number of states (or time-percentages seen).

- **Code repository** *
  https://github.com/EleGo9/Meta-Learning-Curve_AIpert

3 Technical details

In this sections, multiple questions are asked on the technical details of your method. Please fill out this Google Forms *: https://forms.gle/KVq5sZ7uWc23DYbW9

REMEMBER: After you filled out the form, you will get a link for later modification. Please right click on "Edit your response" and copy the link address for later modification and put it below:

https://docs.google.com/forms/d/e/1FAIpQLSe752_rX9-ePoSD9Fz3APreyThaooKc1
viewform?pli=1&pli=1&edit2=2_ABaOnudSMaP4bMk3U7ML8OX-6h918EmWPFzzOh_
29jlQB9MpdUfWvab9MofouBUzMw

References

[1] Gaspard Harerimana JongWook Kim Beakcheol Jang, Myeonghwi Kim. *Q-Learning Algorithms: A Comprehensive Classification and Applications*. 2019.

[2] Binxin Ru Yang Liu Frank Hutter Colin White, Arber Zela. *How Powerful are Performance Predictors in Neural Architecture Search?* arXiv:2104.01177, 2021.

[3] Porta F. Zanni L. Franchini G., Ruggiero V. *Neural architecture search via standard machine learning methodologies*, volume 5, Mathematics in Engineering. 2023.

[4] Vanschoren J. *Meta-Learning: a survey*. 2018.

[5] Maxim Lapan. *Deep Reinforcement Learning Hands-on*.

[6] Isabelle Guyon Manh Hung Nguyen, Nathan Grinsztajn and Lisheng Sun-Hosoya. *MetaREVEAL:RL-based Meta-learning from Learning Curves*.

[7] Noah A. Smith Oren Etzioni Roy Schwartz, Jesse Dodge. *Green AI*. 2019.

[8] Csaba Szepesvari. *Algorithms for Reinforcement Learning*.