Proceedings of Machine Learning Research 191:62–74, 2022 ECMLPKDD Workshop on Meta-Knowledge Transfer

Trust Region Meta Learning for Policy Optimization

Manuel Occorso MANUEL.OCCORSO@MAIL.POLIMI.IT
Luca Sabbioni LUCA.SABBIONI@POLIMI.IT
Alberto Maria Metelli ALBERTOMARIA.METELLI@POLIMI.IT
Marcello Restelli MARCELLO.RESTELLI@POLIMI.IT
Politecnico di Milano, Italy

Editors: P. Brazdil, J. N. van Rijn, H. Gouk and F. Mohr

Abstract

Reinforcement Learning aims to train autonomous agents in their interaction with the environment by means of maximizing a given reward signal; in the last decade there has been an explosion of new algorithms, which make extensive use of hyper-parameters to control their behaviour, accuracy and speed. Often those hyper-parameters are fine-tuned by hand, and the selected values may change drastically the learning performance of the algorithm; furthermore, it happens to train multiple agents on very similar problems, starting from scratch each time. Our goal is to design a Meta-Reinforcement Learning algorithm to optimize the hyper-parameter of a well-known RL algorithm, named Trust Region Policy Optimization. We use knowledge from previous learning sessions and another RL algorithm, Fitted-Q Iteration, to build a policy-agnostic Meta-Model capable to predict the optimal hyper-parameter for TRPO at each of its steps, on new unseen problems, generalizing across different tasks and policy spaces.

Keywords: Meta-Reinforcement Learning; Hyperparameter Optimization; Trust Region Policy Optimization; Fitted-Q Iteration.

1. Introduction

Machine Learning (ML) is a very prolific field that aims to *autonomously and approximately* solve different kinds of tasks; in the last decade there has been an explosion of new algorithms, techniques, and applications in a large variety of fields. ML algorithms make extensive use of *hyper-parameters* to let the user control their behaviour, learning speed, accuracy and many other aspects. Usually, those hyper-parameters are *fine-tuned by hands*, which means that the same algorithm is run multiple times to then select the best model in a validation instance. The choice of those values may change drastically the learning performance of the algorithm, and the same values may not perform equally on very *similar tasks*; this may require to perform very careful fine-tuning sessions.

Hyper-Parameter Optimization (HO) is an area of studies that has the goal of automatize, speed-up and optimize the fine-tuning process. Meta-Learning instead is an approach whose purpose is to automatically learn how to optimally solve a set of different *tasks*: in other words, Meta-Learning objective is *to automatically learn how to learn* (Schmidhuber, 1987). Even though treated as separate fields, HO and Meta-Learning are sometimes interconnected (Franceschi et al., 2017).

In this work we focus on Reinforcement Learning (RL) problems, where the goal is to *train autonomous agents* in their interaction with the *environment* by means of maximizing a

given *reward signal*. RL can be used to train physical robots, software bots, simulators, game-playing AI systems, and any other kind of agent that needs to interact with an environment; often the same agent must be adapted to interact with very similar environments, e.g., a robot may need to solve the same task at different speeds, a recommender system may need to adapt to changes in a store, *etc*. This aspect of RL is what makes it a perfect environment for HO and Meta-Reinforcement Learning; furthermore, RL is usually high demanding from a computational point of view, raising the advantage of learning a way to automatically provide the optimal hyper-parameters for each task, without having to train multiple times the same model with different hyper-parameters.

The main contribution of this work places itself in this exact situation: our proposed Meta-Reinforcement Learning approach tries to optimize the hyper-parameter of a well-known RL algorithm, named *Trust Region Policy Optimization* $\left(TRPO\right)$. At each iteration of the *TRPO* optimization process, the algorithm defines a *Trust Region* around the previous solution, then builds a local approximation of the performance measure and searches for the next candidate agent inside the trust region. *TRPO* provides generally good performances in learning many different kinds of tasks, but heavily relies on one hyper-parameter that is used to constrain the trust region at each step: depending on the learning task it may have a huge impact on the learning capabilities of the algorithm, and this is why it may be important to optimize its usage. Another important aspect of the trust region constraint is that it is commonly *fixed across all the learning steps*: since we will automatically solve the fine-tuning process, we can give to it new degrees of freedom by letting it change at each *TRPO* learning step. This may lead to increase the maximum performances of *TRPO*, instead of only reach its full potential.

This work is organised as follows: we will start in Section 2 by introducing the necessary knowledge required to comprehend the contribution of our work; afterwards the context and the related works are briefly discussed in Section 3; moving on to Section 4, where we present our framework to model the learning process of *TRPO*, called Meta-MDP, and our proposed solution, consisting in the application of an offline RL algorithm, namely *Fitted Q-Iterations* (FQI), to approximate the expected performance gain obtained from the selection of each hyperparameter on a set of different tasks and different policy architectures; finally, the experimental evaluation is provided in Section 5.

2. Background Knowledge

In this section we briefly expose the required knowledge needed to understand the contributions of this work.

Markov Decision Processes. A Markov Decision Process $\left(\text{MDP}\right)$ is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$; $\mathcal{S}$ and $\mathcal{A}$ are respectively the *state space* and the *action space*; $\mathcal{P}\left(\cdot \mid s, a\right)$ is the *transition probability kernel*, which defines for each tuple $(s, a) \in \mathcal{S} \times \mathcal{A}$ a distribution over the next state $s' \in \mathcal{S}$; $R\left(s, a\right)$ is the reward function, $\gamma \in [0, 1]$ is the *discount factor* and μ is the *initial state* probability distribution. The agent is modelled by a *Stochastic Policies* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, whose *Expected Return* is defined as $\eta^{\pi} := \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]$ and ρ^{π} defines the related (discounted) visitation frequencies. It is commonly defined the *state-action value function* $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as the expected return

got using action a in state s and then following π:

$$Q^{\pi}(s,a) \coloneqq \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{T}\gamma^{t}R(s_t,a_t) \mid s_0 = s, a_0 = a\right],$$

The goal of a *learner* in RL is to find the policy π^* that maximizes the expected return η^{π} and, consequently, the maximum state-action value function: $\forall (s,a) \in \mathcal{S}\times\mathcal{A} : Q^*(s,a) = \sup_{\pi} Q^{\pi}(s,a)$. The optimal value function can be recursively computed thanks to the Bellman Equation:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[\sup_{a'\in\mathcal{A}} Q^*(s',a')\right]. \tag{1}$$

Trust Region Policy Optimization. *TRPO* (Schulman et al., 2015) algorithm belongs to the important class of *Policy Gradient* (PG) algorithms (Sutton et al., 1999), which rely on policies *parametrized* by a vector $\boldsymbol{\theta}$. The optimization of the policy π thus follows the gradient of the return η^{π} with respect to the parameters $\boldsymbol{\theta}$. In this work we only consider *Multi-Layer Perceptrons* (MLP) policies with a *Gaussian Noise*; in order to simplify the notation we will use $\boldsymbol{\theta}$ instead of $\pi(\boldsymbol{\theta})$ in the following definitions.

At the core of the PG methods lies the so called *Policy Gradient Theorem* (Sutton et al., 1999), which exposes a simple relation between the direction of the gradient of the expected return $\nabla_{\boldsymbol{\theta}}\eta^{\pi}(\boldsymbol{\theta})$ and the gradient of the policy $\nabla_{\boldsymbol{\theta}}\pi(a \mid s;\boldsymbol{\theta})$:

$$\nabla_{\boldsymbol{\theta}}\eta^{\pi}(\boldsymbol{\theta}) \propto \sum_{s\in\mathcal{S}}\rho^{\pi}(s)\sum_{a\in\mathcal{A}} Q^{\pi}(s,a)\,\nabla_{\boldsymbol{\theta}}\pi(a \mid s;\boldsymbol{\theta}). \tag{2}$$

consequently, the update rule to learn the optimal policy parameter $\boldsymbol{\theta}^*$ is:

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_t + \alpha\nabla_{\boldsymbol{\theta}}\eta^{\pi}(\boldsymbol{\theta})$$

where α is the *learning rate* hyper-parameter.

TRPO, as the name suggests, is the respective Trust Region based optimization of PG methods: instead of finding the optimal direction given by the gradient and then following it for a given step-size, *TRPO* defines a trust region around the current parameters and then solves the optimization subproblem of the kind:

$$\begin{cases}\pi_{t+1} = \arg\max_{\tilde{\pi}} J_{\pi_t}(\tilde{\pi}) \\ \mathbb{D}_{\mathrm{KL}}(\pi_t \,||\, \pi_{t+1}) \le \lambda\end{cases} \tag{3}$$

where $\mathbb{D}_{\mathrm{KL}}(\pi_t \,||\, \pi_{t+1})$ is the KL-divergence of the two policies and λ is the main *TRPO* hyper-parameter which constrains the trust region; $J_{\pi_t}(\tilde{\pi})$ is a first order surrogate loss function of the performances $\eta(\tilde{\pi})$ centered in π_t. The core of the surrogate loss consists in the possibility to estimate $J_{\pi_t}(\tilde{\pi})$ using only trajectories from π_t, hence the algorithm is able to compute and differentiate the approximated loss for multiple policies in the trust region without needing to collect new trajectories, but only having the ones got following π_t.

The practical implementation of *TRPO* requires two-phases for each iteration: The first phase uses an iterative conjugate gradient optimization to find the search direction,

to approximate the *natural gradient*, i.e. a variation of Equation 2 by means of the Fisher information matrix (Kakade, 2001). During the second phase, *TRPO* levers a criteria-based line-search in order to obtain a feasible step size; starting from a maximum threshold following the direction found in the previous phase, the stepsize is halved if the trust region constraint is not satisfied or if the surrogate function does not predict a return improvement.

The hyper-parameter λ is kept constant for all iterations, but this may be a limitation for the performances: the trust region concept imposes a limit in which the approximate return J_{π_t} is trusted in order to improve the real performances η; however, different performance surfaces in the parameters space have a different complexity of approximation: a very smooth $\eta(\boldsymbol{\theta})$ surface can be easily approximated in a big trust-region, while the approximation of a really stiff $\eta(\boldsymbol{\theta})$ cannot be trusted in the same area. At the same time, always imposing a very small trust-region makes the algorithm slower, giving up on some advantages of using *TRPO* instead of a classical PG gradient algorithm.

A fixed-size trust region has also some more subtle consequences on the algorithm itself: the double checking procedure carried out during the second phase at each iteration is needed since the first phase is approximate and its accuracy depends on the trust region constraint; the shrinking steps during this second phase are mainly applied to mitigate the bad effects of a too big trust-region. The shrinking process is thus very conservative, slowing down the algorithm.

2.1. Fitted-Q Iteration

FQI (Ernst et al., 2005) takes a very different approach to solve MDPs, being it a *value based* and *offline* algorithm. It uses a batch of experience samples collected by interactions with the environment, of the kind $\mathcal{D}_{\text{fqi}} = \{\langle s_t, a_t, r_t, s_{t+1}\rangle\}$. The algorithm relies on the iterative application of regression techniques (in our case, Extra Trees $\left(\text{ET}\right)$ (Geurts et al., 2006)) to compute an approximation of the Q-value function: at each next step h a new regression dataset is built as $\mathcal{D}_h \coloneqq \{\langle (s_t, a_t); r_t + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_h(s_{t+1}, a')\rangle\}$, using the previous result $\hat{Q}_h$ and then used to train a new approximation $\hat{Q}_{h+1}$, with the effective horizon increased by one step. Notice that the new target for the regression dataset is built using the Bellman Optimal Operator from Definition 1. The number of iterations of an *FQI* model is referred to as the *horizon* H. One important hyper-parameter of the ET algorithm is the *minimum split* ψ, which defines the ratio between the minimum number of nodes in each leaf and the total number of nodes; a smaller value implies more ramifications, which means smaller sub-regions of the input space and less nodes in each leaf; thus, the *minimum split* controls the fitting capabilities of the ET model.

3. Trust Region Optimization

In this section we state the main objectives of our work and their motivations; furthermore, we will describe the related fields and works in the state of the art.

3.1. Goals

In the *TRPO* section we highlighted the fact that it uses an hand-crafted, fixed-size hyper-parameter to constrain its trust region at each step: this implies that there will probably be

some steps during the learning session that uses a trust region that is too small or too big; the algorithm itself is designed to be very conservative on the step size.

The first goal of our proposed solution is to design a method that trains a model able to automatically choose an optimal trust region hyper-parameter at each learning step, not just once for all the learning session. Secondly, the learnt model should work across a set of similar tasks, as common practice in a meta-learning setting. Thirdly, we want to accomplish the first two independently of the inner-policy: *TRPO* optimizes any kind of parametrized policies over any kind of task, and we want our solution to provide a meta-model that predicts the optimal trust region at each step without knowing the inner-parametrization or the architecture of the policy under training.

The last goal is to improve *TRPO* by reducing its conservativity on the step size: we will work on a modified version of *TRPO*, denoted here as *xTRPO*, that does not shrink the step size when the surrogate loss does not predict an improvement; in other words, we are turning off the feature of *TRPO* that attempts to mitigate the effects of a bad trust region, since we aim to provide the optimal trust region at each step.

3.2. Related Works

Our proposed approach is similar to works in hyper-parameter optimization (HO), and similar to works in meta-reinforcement learning in the methodology.

Hyper-Parameter Optimization For what concerns HO, the field deeply changed in the last years (Lorraine et al., 2020): starting from simple methods that search for optimal hyper-parameters by grid-searching or random searching (Bergstra and Bengio, 2012) all the possible combinations, we now have more complex methods that are able to train more efficiently many hyper-parameters (Im et al., 2021). The main difference we propose in our approach w.r.t. usual HO is that optimizing the trust region hyper-parameter for *TRPO* is just our first step; the fact that we are automatically learning the optimal trust region instead of hand-crafting it naturally implies the possibility of giving the hyper-parameter a new degree of freedom, letting it change at each learning step, which would not be practically feasible if hand-crafted.

Meta-Learning Meta-Learning has the objective of improving a learning algorithm over a set of *tasks*, which are different instantiations of similar ML problems (Vanschoren, 2018). We heavily rely on the meta-learning setting: we have a meta-training phase in which we interact with a set of tasks and train a meta-model, and then a meta-testing phase in which new tasks are sampled and the meta-model is tested against some baselines. One of the most famous meta-learning algorithms, *Model Agnostic Meta-Learning* (MAML) (Finn et al., 2017), allows to learn the best parameters initialization in order to be adapted to a new task, and it does so in a *model-agnostic* manner: it does not rely on a particular kind of model or training algorithm, as long as it is a parametrized model and it is differentiable. Differently from MAML and other Meta-Learning algorithms, our approach wants to be *parametrization agnostic* in the sense that it works only with a class of models, parametrized-policies, but the kind of parametrization does not really matter: we can use multiple different parametrizations during meta-training, and then the meta-model can work also with new, never seen parametrizations.

Meta-Reinforcement Learning Meta-Reinforcement Learning is meta-learning applied to a reinforcement learning algorithm, implying that the set $\mathscr{M}$ of similar tasks contains different MDPs $\mathcal{M}_i$. A method that is specific for meta-reinforcement learning is Meta-MDP REINFORCE (Garcia and Thomas, 2019), which proposes a notion of meta-MDP, very similar to ours, with different goals and usage. Meta-MDP here, as in other works, refers to the usage of a MDP inside another MDP: the inner one is a normal RL environment over which we need to optimize a policy, while the outer one models and optimizes the learning process itself.

4. Proposed Solution

In this section, we present our solution to the previously defined goals. We first start defining the notion of Meta-MDP, which we use to model a RL training session; we then detail some critical aspects of the meta-MDP concept and implementation; we then move on to the techniques to solve a meta-MDP.

4.1. Meta-MDP

A meta-MDP is a tuple $\langle \mathcal{X}, \Lambda, \mathcal{L}, (\mathscr{M}, p), (\Pi, q), f \rangle$, where:[1]

- $\mathcal{X}$ and Λ are respectively the *meta-state space* and the *meta-action space*;
- $\mathcal{L} : \mathcal{X} \times \Lambda \to \mathbb{R}$ is the *meta-reward function*;
- $(\mathscr{M}, p)$ and (Π, q) contain respectively the set of inner-tasks[2] and the policy space, together composing the meta-task space, with their respective distributions over the initial task and policy.
- $f : \Pi \times \Lambda \to \Pi$ is the update rule of the inner reinforcement learning algorithm chosen, where the meta action represents the selected hyperparameter.

Given a meta-task (a sampled inner-policy and a inner-task), the meta-state is the distribution over the trajectories given by the interaction of the policy with the inner-task. The internal dynamic depends on the meta-state and meta-action as in a MDP, but is intrinsically complex as it represents a complete policy update with update rule f with hyperpamater $\lambda \in \Lambda$; moreover, since the inclusion of all representation of the trajectories is unfeasible, we will encode the current meta-state using a set of observed metafeatures: this makes the meta-MDP a partially observable MDP. Further, we will learning as meta-reward: $\mathcal{L}(\mathbf{x}_t, \lambda_t) = J(f(\pi_t, \lambda)) - J(\pi_t)$, which is the difference between the next expected return and the current one; notice that the dependence on the meta-state is given by the policy π_t and the J, which depends on the current inner-task.

Update Rule At each step, given the current meta-state and meta-action, the update rule can be applied to the inner-policy as $\boldsymbol{\theta}_{t+1} \coloneqq f(\boldsymbol{\theta}_t, \lambda_t)$: a batch of trajectories $\mathcal{B}_t$ is sampled from the interaction between the current policy and task and given to the learner, producing the update rule, with hyperpamarameter λ_t. We chose to optimize the *xTRPO* algorithm:

1. a meta-discount factor $\tilde{\gamma}$ can be included in the definition, here set as 1.
2. Each *inner-task* $\mathcal{M} \in \mathscr{M}$ is a different MDP.

Measure	Distribution	Timing
$\mathbb{H}$	$p_{(s)}$, $p_{(a\|s)}$, $p_{(s'\|a,s)}$	(t) and $(t-1)$
$\mathbb{C}$	$p_{(s)}$, $p_{(a\|s)}$, $p_{(s'\|a,s)}$	$((t),(t-1))$ and $((t-1),(t))$
$\mathbb{D}_{\mathrm{KL}}$	$p_{(s)}$, $p_{(a\|s)}$, $p_{(s'\|a,s)}$	$((t) \,\|\|\, (t-1))$ and $((t-1) \,\|\|\, (t))$
$\mathbb{E}[J]$	$p_{(r)}$	(t)
$\mathbb{V}\mathrm{ar}[J]$	$p_{(r)}$	(t)

Table 1: Summary of all the metafeatures $\phi(\mathbf{x})$ used as observation for the Meta-MDP. *Distribution* refers to the type of distribution considered instead of the trajectory distribution. *Timing* refers to the learning step considered for the distribution. For example, for $\mathbb{H}$, $p_{(s)}$ and (t) we mean $\mathbb{H}(p_{(s)}(t))$, while for $\mathbb{D}_{\mathrm{KL}}$, $p_{(a|s)}$ and $((t) \,||\, (t-1))$ we mean $\mathbb{D}_{\mathrm{KL}}(p_{(a|s)}(t) \,||\, p_{(a|s)}(t-1))$.

this means that our update rule f is one step of *xTRPO*; the step takes as input the batch $\mathcal{B}_t$ and a trust region constraint λ_t; it computes an approximate natural gradient and then applies a line-search to stay inside the trust region, returning the new policy parameters.

Encoding the Meta-State There is no analytical representation of the meta-state x_t (the distribution over the trajectories defined by the current policy $\pi_{e,t}$ and inner-task $\mathcal{M}_e$): consequently, starting from a batch of sampled trajectories $\mathcal{B}_t$, we need to consider a different set of metafeatures as meta-observations $\phi(\mathcal{B}_t)$.

The distribution of the overall trajectories of a policy in the state space is unfeasible to be computed, hence we can decompose it into simpler distributions that define the trajectories: $p_{(s)}$, $p_{(a|s)}$, $p_{(s'|a,s)}$, which are respectively the state-visit distribution, the policy distribution (the actions chosen by the policy given the state) and the transition distribution (the distribution over the next state given the current state and action); for what concerns the rewards, we can rely on the sample mean and variance of the empirical return.

We compress the meta-state even more computing some metrics over these smaller distributions: the *Shannon-entropy* $\mathbb{H}$, the *cross-entropy* $\mathbb{C}$ and the *KL-Divergence* $\mathbb{D}_{\mathrm{KL}}$ (Shannon, 1948). The Shannon-Entropy is a very common measure of how complex and chaotic is a distribution; the cross-entropy and KL-divergence instead measure the distance between two distributions in two different ways and with two different meanings; since we want to include information about the current learning state and the history of the learning, entropy will be useful to encode each learning state, while cross-entropy and KL-divergence will encode information about the difference between policies. All the metafeatures we use can be estimated from samples (Singh et al., 2003), more specifically using a set of N trajectories performed at current time step t and the previous set, sampled at step $t-1$. The summary of all the metafeatures can be found in Table 1.

4.2. Solving the Meta-MDP

In order to solve the meta-MDP, i.e., learn a meta-model that provides the meta-action that maximises the meta-return, we apply the *FQI* algorithm. We can collect meta-trajectories over a variety of different meta-tasks, including different policy-parametrizations and different inner-tasks. From the meta-trajectories we compute a meta-training dataset containing

samples of the kind $\langle \phi(\mathbf{x}_t), \lambda_t, l_t, \phi(\mathbf{x}_{t+1}) \rangle$, which is the meta-version of the tuples required in input for FQI (meta-observation, meta-action, meta-reward, and next meta-observation). At the beginning of each learning trajectory, a new inner-task $\mathcal{M}_e$ is sampled from the distribution p, and a new policy from q, which may include different types of neural architectures. Furthermore, at each learning step the hyperparameter λ is sampled from a uniform distribution within a fixed range. From the resulting dataset we can train a *FQI* model that estimates the meta-Q-value function

$$\widehat{Q}_{(H,\psi)}(\phi(\mathbf{x}_t), \lambda_t) \ .$$

After a validation process in order to chose suitable values for H and ψ, we can test the model by using it to select meta-actions during an *xTRPO* training session; given the current metafeatures ϕ_t, meta-action is selected as:

$$\widehat{\lambda}_t = \arg\max_{\lambda \in \Lambda} \widehat{Q}(\phi_t, \lambda) \ .$$

5. Experiments

The experimental environments we used for our tests are *Elikoeidis*, *Minigolf* (Tirinzoni et al., 2019) and *Cartpole* (Penner, 2002).

Elikoeidis is an environment we *specifically designed* in order to put the original *TRPO* algorithm in difficulty, and see whether our approach could perform better with only linear policies. More technically, it is a single-state environment, which lets us control the complexity of the reward given the action by means of its task; the reward functions are of the kind $R(a) \sim \mathcal{N}\left(c_1 a \sin(a^{1+\frac{1}{c_2}}) + c_3 a, \quad c_4\right)$, where the vector c is the task. In this case, we want to analyse the effects on a single-task MDP, with $c = [0.5, 3, 0.5, 1]$. FQI dataset collection is made of 100 tuples, and for each one the metafetures are extracted from sets of $N = 8$ linear policy trajectories.

Minigolf is an environment presented in (Tirinzoni et al., 2019) that emulates a minigolf game, in which the agent selects the club angular speed at each hit, with the objective of letting the ball enter the hole; the task is given by the club length and the friction coefficient of the golf field.

Eventually, *Cartpole* is the standard and classical RL environment in which the objective is to control a cart balancing a pole; the agent controls a force applied to the cart horizontally; the task is given by a variable mass and length of the cart pole. For Minigolf and Cartpole environments, the FQI datasets (containing respectively 7000 and 15000 tuples) is made by adopting linear policies and Multi-Layer Perceptrons with a different number of hidden layers (3 maximum) and hidden neurons. N is set to 64 for the evaluation of metafeatures.

The settings are typical of Meta-Learning, with a meta-training phase and a meta-testing phase: in our case, the meta-training is composed by a meta-dataset collection phase, followed by the training phase, since *FQI* is an off-line algorithm; further, we also had a meta-validation phase in order to select the best across different meta-models, trained on the same datasets but using different hyper-parameters H and ψ. As commonly done in meta-learning, during the meta-tests we compare the performances of learning sessions

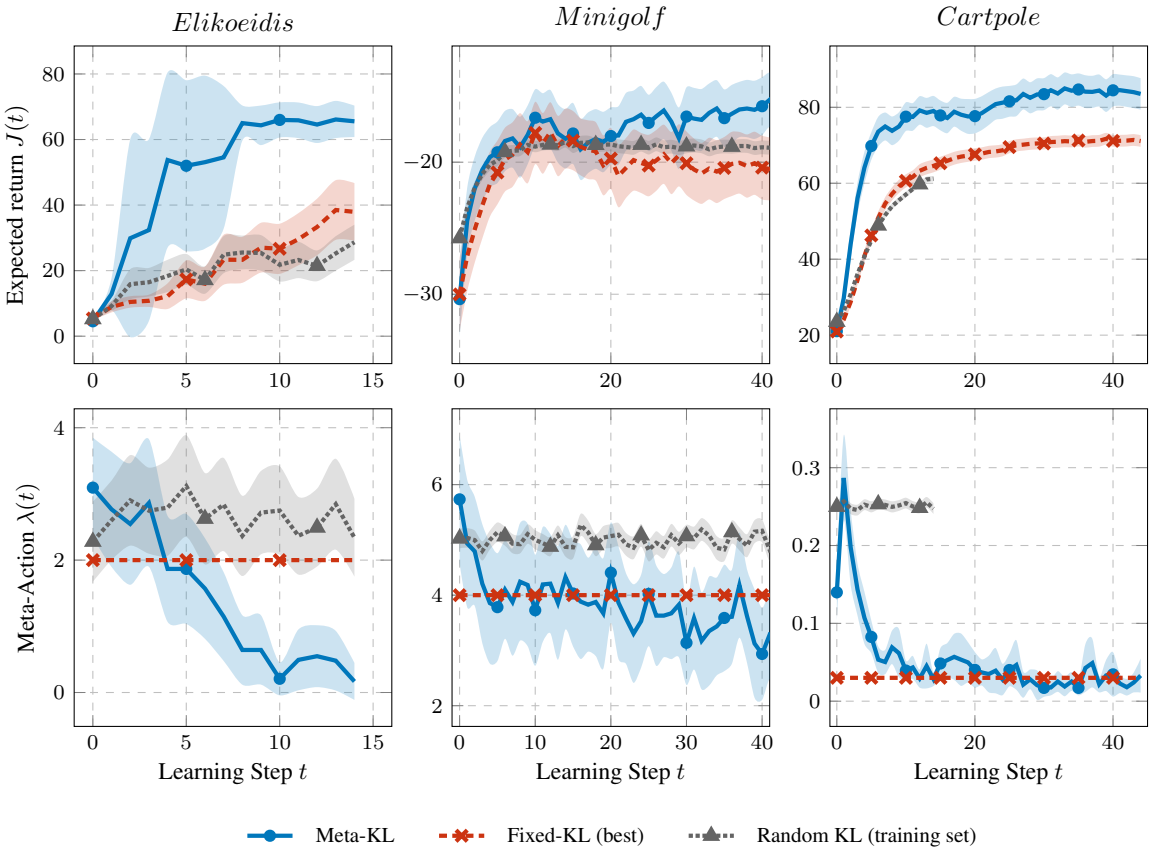

Figure 1: Fine-tuned hyper-parameter *TRPO* vs meta-model with linear policies. *Top Figure:* Comparison of the average performances of the tested FQI models (Meta-KL) w.r.t the baselines (Fixed-KL) and the results from the learning trajectories in the meta-training dataset (random KL) Cartpole training dataset is composed of learning trajectories with 15 only total updates, while the evaluation process is performed on 45 learning steps. *Bottom Figure:* meta-action selected at each learning step. 30 runs, avg $\pm 90\%$ c.i.

collected using our meta-models to those collected using a fixed-value hyper-parameter fine-tuned by means of a grid search, which are referred to as *baselines*.

In Figure 1 we show the main experimental results of our work: for each iteration of *TRPO* algorithm, we report the sample mean of the performances and the average selected hyper-parameter using the resulting model, compared to the *baselines*. The Training Meta-Dataset curves represent the average performances and λ values obtained in the dataset for training *FQI* models, with a random hyper-parameter at each step. We show it in order to provide a glimpse of how much information has been learned: for example, in *Cartpole* the meta-training dataset contained only information about the first 15 steps of learning, while the meta-model has been tested on trajectories 45 steps long, requiring good generalization capabilities. The baselines are composed of a set of trajectories collected

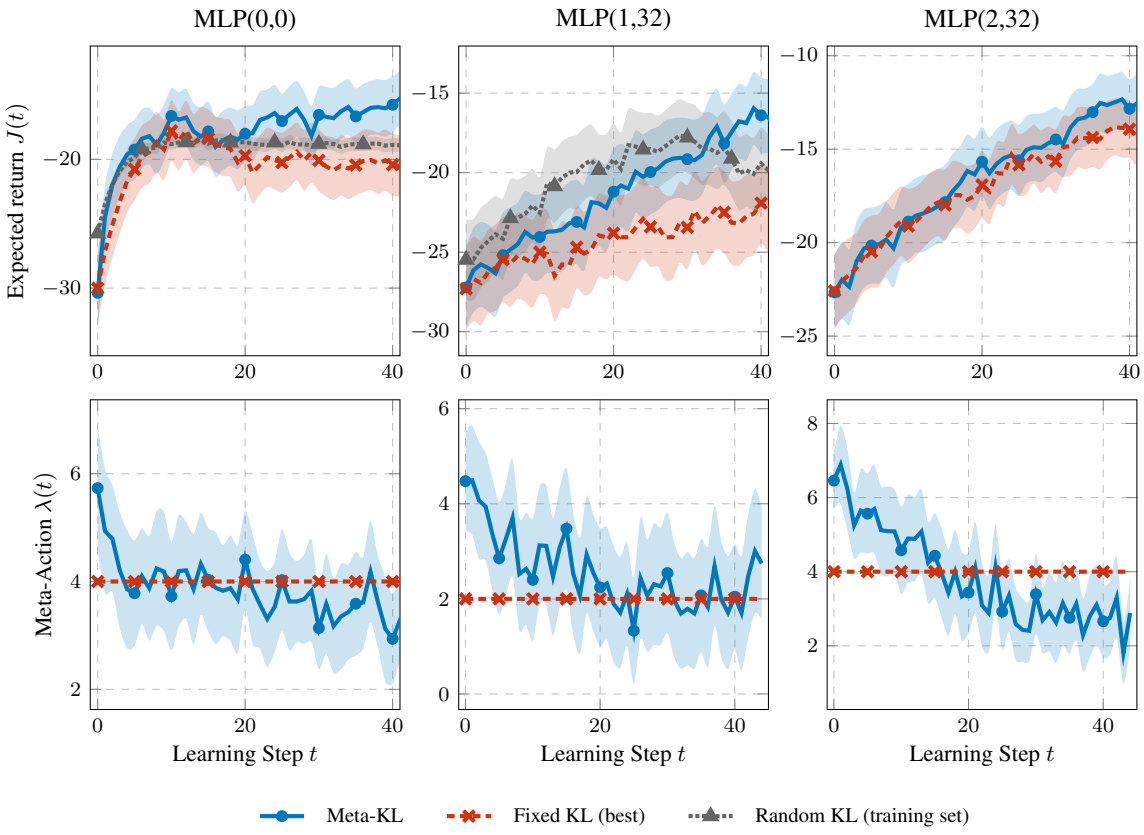

Figure 2: Minigolf: Performances by different Policy Parametrizations. *Top figure*: Comparison, across different architectures, of the FQI model ($H = 3$, $\psi = 10^{-3}$) against the baselines, including training trajectories. *Bottom figure*: meta-action selected at each learning step. 30 runs per architecture. MLP(h, n) denotes an MLP with h hidden layers and n neurons per layer. MLP(0,0) denotes a linear policy. MLP(2,32) (rightmost plot) is only tested, i.e. no policies with this architecture were considered in the training dataset.

applying a constantly constrained trust region *TRPO* across the same set of test meta-tasks. Figure 1 depicts the performances of the best validated FQI model, compared with the best performing baselines. The comparison shows clearly how the meta-model can always perform as good as the best baseline found, and in many times even better, by choosing an adaptive trust region, decreasing with time.

Figures 2 and 3 show the results across different policy architectures respectively regarding the Minigolf and Cartpole environments. In the first scenario, we can observe that the FQI model is able to reach better results than the baselines in the case of linear policies and with MLP with 1 hidden layer. This last case is emblematic, as also selecting a random trust region seem to bring an improvement with respect to the fixed case. The rightmost plot, instead, depicts the results on a set of instances where the selected parametrizations is a MLP with 2 hidden layers and 32 hidden neurons, which was never seen by the FQI model

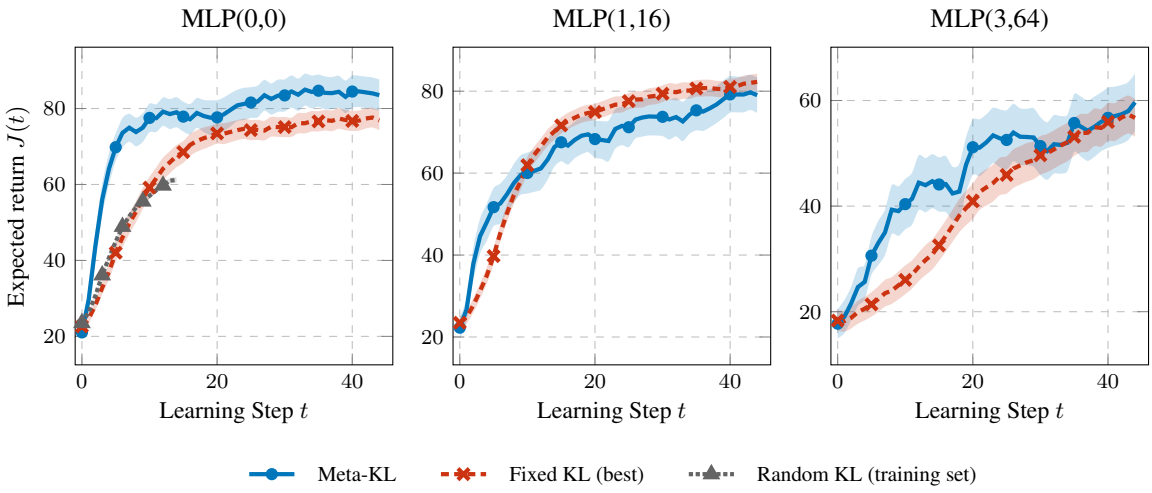

Figure 3: Cartpole: Performances across different Policy Parametrizations. Comparison, split across different architectures, of the FQI model ($H = 1$, $\psi = 5 \cdot 10^{-5}$) against the baselines, including training trajectories. 30 runs per architecture. MLP(h, n) denotes an MLP with h hidden layers and n neurons per layer. MLP(0,0) denotes a linear policy. MLP(1,16) and (3,64) are not in the training dataset, i.e. no policies with these architectures were considered for FQI training.

during training: the model is able to reach the same results as the best baseline, computing using a grid-search, hence proving its generalization capabilities. The same conclusions can be taken for the Cartpole case (Figure 3), where the resulting FQI model is able to outperform the baselines for the linear case (leftmost plot), and to generalize well for unseen architectures (central and rightmost plots), by using only trajectories with 15 updates.

6. Conclusions

The general goal of this work is to improve *TRPO* by means of applying meta-learning to the trust region hyper-parameter optimization, letting it change at each learning step. Furthermore, we wanted our solution to build a parametrization-agnostic meta-model, which can be trained on multiple different policy parametrizations and then used on other parametrizations, even if never seen.

Performing better than the optimal (fixed) baseline found with a grid search means that we developed a method to train a meta-model that can substitute and improve the hand-crafted choice of the trust region hyper-parameter, work that must be done for each new task or policy parametrization.

All the experiments carried out suggest that the meta-MDP direction is promising: the results obtained in the *Elikoeidis* environment, which was designed with the intent to be easily learned by *TRPO* using actions provided by a meta-model on linear policies, show the feasibility of our approach, while the experiments on *Minigolf* and *Cartpole* show how, at least on simple environments, it is possible to fully reach our goals, improving *TRPO* over a set of tasks, and automatizing the hyper-parameter selection.

There are anyhow many situations in which a meta-learning approach does not bring any advantage: when the optimal fixed-size trust region does not change across different parametrizations or different tasks, it is not computationally efficient to take the effort of building the meta-model to automatize the hyper-parameter choice; on the other hand, the new degrees of freedom on the trust region may result in finding optimal models that cannot be found by means of a fixed-size trust region. Moreover, in complex environments it may require a lot of meta-training data, as in the classic applications of FQI. As future research directions, we may wonder how to generate and select more informative metafeatures, and to develop online approaches, that can bring the advantages of a dynamic choice of learning hyperparameters.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 2012.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, pages 1126–1135. PMLR, 2017.

Luca Franceschi, Paolo Frasconi, Michele Donini, and Massimiliano Pontil. A bridge between hyperparameter optimization and larning-to-learn. *CoRR*, abs/1712.06283, 2017.

Francisco M. Garcia and Philip S. Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pages 5692–5701, 2019.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Daniel Jiwoong Im, Cristina Savin, and Kyunghyun Cho. Online hyperparameter optimization by real-time recurrent learning. *CoRR*, abs/2102.07813, 2021.

Sham M. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems 14*, pages 1531–1538. MIT Press, 2001.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020*, pages 1540–1552. PMLR, 2020.

A Raymond Penner. The physics of golf. *Reports on progress in physics*, 66(2):131, 2002.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015,*, pages 1889–1897. PMLR, 2015.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. The MIT Press, 1999.

Andrea Tirinzoni, Mattia Salvini, and Marcello Restelli. Transfer of samples in policy search via multiple importance sampling. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, pages 6264–6274. PMLR, 2019.

Joaquin Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.