Addressing Task Prioritization in Model-based Reinforcement Learning

Artem Zholus[1] and Yaroslav Ivchenkov[1] Aleksandr I. Panov[2,3]

[1] Moscow Institute of Physics and Technology,
zholus.aa@phystech.edu, ivchenkov.yap@phystech.edu
Moscow Russia
[2] AIRI, Moscow, Russia
[3] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia

Abstract. World models facilitate sample-efficient reinforcement learning (RL) and, by design, can benefit from the multitask information. However, it is not used by typical model-based RL (MBRL) agents. We propose a data-centric approach to this problem. We build a controllable optimization process for MBRL agents that selectively prioritizes the data used by the model-based agent to improve its performance. We show how this can favor implicit task generalization in a custom environment based on MetaWorld with a parametric task variability. Furthermore, by bootstrapping the agent's data, our method can boost the performance on unstable environments from DeepMind Control Suite. This is done without any additional data and architectural changes outperforming state-of-the-art visual model-based RL algorithms. Additionally, we frame the approach within the scope of methods that have unintentionally followed the controllable optimization process paradigm, filling the gap of the data-centric task-bootstrapping methods.

Keywords: Reinforcement Learning, Model-based Reinforcement Learning, Generalization in RL

1 Introduction

Deep reinforcement learning requires millions of online samples to converge, and the results are often task overfitted and unstable. Modern advances in deep RL are primarily associated with using model-free approaches operating in fully observable environments [1]. However, their practical use, for example, in mobile robotics [2,3] and unmanned transport systems [4,5], is limited primarily due to they are task overfitted [6]. Many approaches were proposed to address the problem of the task generalization in RL, namely, goal conditioned RL [7,8], hierarchical RL [9,10], meta-RL [11,12], imitation learning [13,14] and model-based RL [15,16,17]. The latter, however, has not been studied in the context of the task generalization in RL [18]. One of the possible reasons for that is a non-atomic structure of model-based agents with which both policy and the world model should generalize [19].

In this paper, we focus on the *implicit* task generalization of model-based RL agents. Classic task generalization is defined in terms of the zero-shot policy transfer [18], i.e.,

after training the agent on a training tasks distribution $p_{\text{train}}(\tau)$ one should evaluate its performance on the test task distribution $p_{\text{test}}(\tau)$. The generalization is then measured as the performance on the test task distribution. The compound structure of the model-based agents appeals for implicit generalization, that is, how much can agent benefit from the use of task data from $p_{\text{train}}(\tau)$ while learning on the tasks from $p_{\text{test}}(\tau)$. Implicit task generalization is a more broad term as it does not specify how to use the data from the training task distribution. Hence, the zero-shot policy transfer is just an instantiation of the way to do this.

To implement of the implicit generalization method, we propose a controllable optimization of the model-based agent. We build a formalism of meta-Markov Decision Processes (meta-MDP) that black boxes the training of a machine learning model. In this meta-MDP, which we call the *training MDP*, a training policy that optimizes some reward serves as a tool for influencing the optimization of the internal model. We show how this formalism applied to model-based RL fills the gap in specific research directions by framing several prior works into the training MDP context [20,21,22,12]. To evaluate the agent's performance in the training MDP, we perform two main lines of experiments. First, we treat the agent's training dataset as the multitask buffer for the external agent to take samples from. This ensures how well the model can prioritize the agent's buffer to bootstrap the task-specific performance of the model. Second, we assess how well our model can utilize the tasks with parametric variability. We provide the model with a dataset with many experiences of agents solving tasks with different parameters. Then, we train the extended agent on different tasks from the same distribution and provide it access to the experiences of the prior agents. Given that, we show how the model can leverage the existing task data to boost its performance.

Our model uses the training MDP agent to maximize the reward of the internal task as the training MDP reward. This is in contrast with the usual approach of TD-error maximization. We chose to maximize the reward instead since our goal is to find the most promising experiences possibly coming from different performance tasks but not the model prediction improvement potential. Also, we aim to build a practically valuable algorithm that can be used for real-world reinforcement learning problems. Therefore we consider only visual reinforcement learning problems.

The main results of the work can be summarized as follows:

1. We ground the training of the model-based agent [23,24] into the training MDP. The training of the model-based agent is then equivalent to acting in this meta-MDP.
2. We propose a trainable agent that learns to maximize reward in such meta-MDP by using an attention-style prioritization over the training data to propose training samples for the model-based agent.
3. We shape existing methods [25,21,22,12] into the context of the meta-MDP acting through prioritization.
4. We show how the proposed meta-agent can be used to bootstrap the performance of the RL agent and implicitly generalize over parametric variation tasks.

2 Background

2.1 Reinforcement Learning

We formulate the problem of reinforcement learning in the context of the Partially-Observable Markov Decision Process (POMDP). Formally, POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, O, \gamma \rangle$, where $\mathcal{S}$ is the set of states of POMDP, $\mathcal{A}$ is the set of actions, and $\mathcal{O}$ is the set of observations. The environment changes its state according to the conditional transition distribution $P(s' \mid s, a)$, but the agent only has access to the output of the observation function $o = O(s', a)$, $o \in \mathcal{O}$. The agent is defined by a policy $\pi(a_t \mid o_{\leq t}, a_{<t})$. It interacts with a partially observable environment by taking actions on the environment and getting a next observation. We write $o_t, r_t \sim p(o_t, r_t \mid o_{<t}, a_{<t})$ as a shorthand for $s_t \sim P(s_t \mid s_{t-1}, a_{t-1}),\ o_t = O(s_t, a_{t-1}),\ r_t = R(s_t, a_{t-1})$. The goal of the agent is to maximize its expected discounted sum of rewards $\mathbb{E}_\pi \sum_t \gamma^t r_t$.

2.2 Model-Based RL via World Models

World models explicitly learn environment dynamics to generate novel experience [26,24]. Visual control tasks require efficient approaches to state prediction. When observations are non-Markovian, this can be achieved only by incorporating latent states. For latent dynamics learning, we use the Recurrent State-Space Model (RSSM) [23], which learns the dynamics by building a Markovian latent state for each timestep s_t given previous action a_{t-1} by autoencoding observations o_t and rewards r_t, which are non-Markovian. The world model consists of a representation model, or an encoder $q(s_t \mid s_{t-1}, a_{t-1}, o_t)$, a transition model $p(s_t \mid s_{t-1}, a_{t-1})$, an observation model $p(o_t \mid s_t)$, and a reward model $p(r_t \mid s_t)$. The representation and transition models share common parameters in the RSSM network [23]. The world model is trained to maximize the variational lower bound on the likelihood (ELBO) of the observed trajectory conditioned on the actions $\mathbb{E}_p \log p(o_{1:T}, r_{1:T} \mid a_{1:T})$. This is done by incorporating an approximate posterior model $q(s_t \mid s_{t-1}, a_{t-1}, o_t)$, which is known as a representation model. This model acts as a proposal distribution for states s_t. Dreamer is trained using a continuously growing training buffer that contains all experiences collected during the environment interaction.

For behavioral training, we use the Dreamer agent [27] which trains the policy and its value function by *latent imagination*, an approach where the policy $\pi(a_t \mid s_t)$ is optimized by predicting both actions $a_t \sim \pi(a_t \mid s_t)$ and states $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$ without interaction with environment. In particular, it uses a state-value function estimate on the latent state s_t on which the policy is trained to maximize, i.e. $\mathbb{E}_\pi \sum_t V_\lambda(s_t)$, where $V_\lambda(s)$ is a multi-step value estimate with a hyperparameter λ which controls the bias variance trade-off [28]. As the transition and value models are parameterized by neural networks, we can backpropagate through the value function, the transition model, and the action sampling to compute symbolic gradients of the value estimates w.r.t. policy parameters.

3 Task Addressing for Prioritized Implicit Generalization

In this section, we define a meta-MDP of the agent. As the usual MDP, it is a tuple $\left\langle \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{P}, \tilde{\rho}, \tilde{R} \right\rangle$ where everything is defined as in usual MDPs ($\tilde{\rho}$ is an initial state distribution). We draw a correspondence between meta-MDP and the training process of the RL agents. The state-space $\tilde{\mathcal{S}}$ is the set of all possible configurations of the agent, usually defined as $\tilde{\mathcal{S}} = \Theta \times \mathcal{B}$, where Θ is agent's parameter space and $\mathcal{B}$ is the set of all possible training buffers. The action space $\tilde{\mathcal{A}}$ can vary for different algorithms, but in general, it represents the way of how we can update the agent. Usually, it is a training sample or a batch of training samples. Finally, the transition distribution represents the implementation of the optimization algorithm as it uses the previous state of the model (e.g., its parameters) and the update object (e.g., training batch) to stochastically or deterministically map them into the next "state", i.e., the set of parameters. Therefore, a policy: $\tilde{\pi} : \tilde{\mathcal{S}} \to \Delta(\tilde{\mathcal{A}})$ here is any rule that determines which batch to select next given the current model. In a sense, such encoding is quite natural as it builds a direct resemblance between the code for training the model and the RL framework. For example, the initial state distribution corresponds to the model initialization and the dataset creation in this case. Moreover, random walks over the state space, i.e., sampling uniformly random actions, corresponds to simple model optimization (as we hide the internal optimization into the state transition distribution).

The most important part of this paradigm is a reward function of the meta-MDP $\tilde{R}$. First, the reward function defines the general property we want our algorithm to have, such as robustness or adaptability. Second, this allows us to harness the power and diversity of the vast amount of reinforcement learning algorithms to optimize the training procedure directly. Our key insight is that we can tie the internal reward of the task with an external one of meta-MDP. This can help us in several ways. First, it can additionally bootstrap the performance of the RL agent by selecting promising training data. Next, it can facilitate learning general, inter-task features that helps agent to use the multitask data to speed up the current task learning.

Our formulation means the reward prediction as we are not restricted to the case of one task, which is why we use the model-based RL. Next, to test a wider range of the RL algorithms, we may need to differentiate through the learned dynamics. Finally, as we aim to build an algorithm that applies to real-world problems, it is essential to enable visual RL. Summing this up, we use the Dreamer [27] algorithm as an instance of the state-of-the-art RL algorithm satisfying these requirements.

3.1 Addressing Mechanism

The agent in the training MDP must act on the instance of the state space, i.e., the whole parameter set of the neural network of the internal agent. This may lead to an impractical algorithm as we need to predict external reward in a very high dimensional feature space. Therefore, the design of a proper state representation is not trivial and should be motivated by the specifics of a high-level task. In order to implement efficient meta-reward (to avoid confusion, we will refer to a reward in the training MDP

as meta-reward) optimization, we shift representation into the data space. $\tilde{R}$ is an expected agent's reward over internal trajectories, so the use of the agent's trajectories as representatives of its state can help to optimize this value directly.

The meta agent should output a subsample of the dataset of potentially unbounded size. The output is also of variable size where different components do not interfere each other. This motivates us to use an attention-like architecture for the model. In particular, the agent's trajectory serves as a query and the data sample is a key. We use hard attention as we need to retrieve whole data samples.

Formally, for selecting trajectories from the dataset, we use the learned attention model parameterized by a neural network. The model compares trajectory $x = \{(a_t, o_t)\}_{t=1}^{T}$ from the current dataset with a batch of trajectories M from possibly different dataset. The model projects x and each M_j into the embedding space and then calculates softmax logits for each (x, M_j) pair as a dot product in the embedding space. To select the multitask trajectory, we sample index j from distribution $q(j \mid x, M)$ that has the form:

$$q_\phi(j \mid x, M) \propto \exp(g_\psi(f_\phi(x))^T h_\theta(f_\phi(M_j))),$$

where f_ϕ is the learnable embedding of trajectory x or M_j parametrized by a recurrent neural network with parameters ϕ. f_ϕ consumes a sequence of actions concatenated with observation embeddings obtained from convolutional neural network (CNN). As the embedding, we use the last hidden vector of the recurrent neural network (RNN). This embedding is then passed to g_ψ and h_θ function parameterized as multilayer perceptrons (MLP). These two serve a role of different heads acting on a common RNN backbone. In other words, we project trajectory x and each M_j into the embedding space. Next, we calculate the inner product between the embedding of x and the embedding of each M_j, and finally, we pass the resulting vector of the inner products to the softmax that gives probabilities for a categorical distribution. At preliminary experiments, we have observed that a trained CNN is crucial here as its presence significantly boosts the performance. This indicates that the features that are key to a good representation are not those that are required for good addressing. Finally, without g_ψ and h_θ the optimization would lead to just learning abstract behavioral features and a similarity search without reward maximization, as we observed during early experiments.

3.2 Expected Reward Optimization

We consider different approaches to meta-reward optimization. The most direct way for doing this is using the simplest policy gradient estimator. This facilitates faster and more stable learning because the task of training MDP is relatively simple.

The loss for a policy gradient has the form:

$$\mathcal{L}_{\text{PG}} = -\mathbb{E}_{q(j|x,M)} R(M)_j \log q(j \mid x, M) \tag{1}$$

where $R(M)_j$ is a predicted return of the trajectory M_j.

Another possible approach considered in this work is using deterministic policy gradient for training the addressing model. The sampled batch as the meta-agent's action

goes through the differentiable world model to the value head. We feed each trajectory $M_j = \{(a_t^j, o_t^j)\}_{t=1}^T$, which is selected with respect to a particular trajectory x_i (i.e. $j \sim q(j \mid x_i, M)$) to the RSSM and run latent imagination procedure to obtain its value estimates $V_\lambda(s_t^j)$. Then we sum the estimates over the imagination timesteps t and backpropagate gradients of the resulting scalar loss up to the gradients w.r.t. the weights ϕ of the address network. We train the address network to minimize the objective:

$$\mathcal{L}_\text{V} = -\mathbb{E}_{q(j|x_i,M)} \sum_t V_\lambda(s_t^j) \tag{2}$$

To enable efficient gradient computation, we use the straight-through gradient estimator [29] to obtain differentiable samples j returned as one-hot vectors. We then multiply each of these one-hot vectors by a batch of expert trajectories M to obtain selection by index in a differentiable fashion. We considered this approach in early experiments but we made final evaluations using the REINFORCE based approach as we find that more promising.

To train a model-based agent using the addressing model, we first sample a number of the agent's trajectories from its training buffer. Next, we run the address inference $j \sim q(j \mid x_i, M)$ given a batch of external experiences M. Finally, we do an optimization step using a batch of several M_j. The overall process is outlined in Figure 1. We refer to this method as the Task Addressing for Prioritized Implicit Generalization (TAPIG).

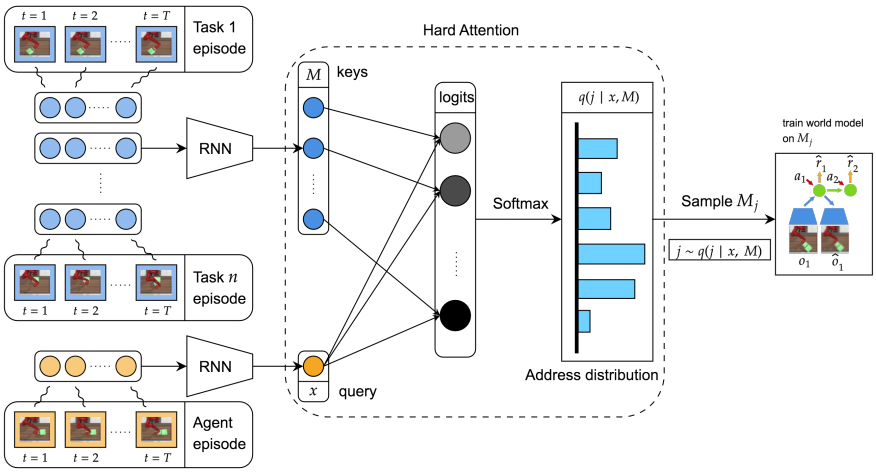

Fig. 1: A principal scheme for training TAPIG augmented agents.

An important observation is that it is not enough to use the same model f_ϕ to encode both x and each M_j. Using the same model for both would mean that for a fixed index

j the logit $f_\phi(x)^T f_\phi(M_j)$ would not change if x were used in place of M_j and M_j in place of x. However, the target which REINFORCE is training towards will change after such a swap. Despite the corresponding gradients w.r.t. x and w.r.t M_j still will differ from each other, this motivates us to use different heads for processing x_i and M_j.

4 Related Work

Many approaches leverage environment modeling, use the multitask approach for reinforcement learning, unintentionally follow the training MDP perspective, or reweight training samples to yield better performance of the agent's learning process.

Multitask reinforcement learning. Plan2Explore [19] adds the exploration stage maximizing a latent disagreement in order to build a better task-agnostic world model. RL^2 [12] uses RNN to encode information about the task into a fixed-size latent vector.

Meta reinforcement learning. We emphasize that our approach is different from the classical meta-learning formulation. We address the problem of implicit task generalization, which may or may not be solved using a meta-learning approach. In our case, we stick to a data-centric approach that is different from the meta-RL. MAML [11] alleviates meta reinforcement learning by training a model with meta-objective, optimizing weights to be easily fine-tuned toward any task. Though, the meta RL can be cast into the training MDP terms. Our formulation serves the purpose of influencing the training process through reward optimization.

Agents in the training MDP. Prioritized DQN [20] is the simplest example of the agent in the training MDP, which is defined implicitly. DQN is trained on prioritized batches, which corresponds to a heuristic meta-agent that selects batches with probability proportional to a TD error. Prioritized Level Replay [21] uses the same idea but is applied to levels in MDP to improve generalization. Data Valuation algorithm (DVRL) [22] uses actor-critic in the training MDP that is trained to achieve robust supervised learning. This is done opposite to the previous two works on a static dataset. RL^2 [12] fuses internal and external agents in one neural network to facilitate multitask RL. In this context, our approach fills the gap of methods that set the external reward to be the same as the internal reward and work on the level of data-centric methods [25,21,22].

Memory in reinforcement learning. Episodic memory models [30] leverage the memorization mechanism to build better Q-value estimates using similarity search in the state-action representation space. Compared to this approach, we store whole episodes into memory and set up memory to contain episodes that represent solutions for different tasks.

Reweighting training samples. DVRL [22] is a meta-learning algorithm that trains a proposal distribution for data samples optimized jointly with the main model. In contrast to our work, it uses one dataset to focus on robustifying the supervised model. Learning-to-Reweight [31] is a model-agnostic approach to reweighting data samples that aim at increasing the robustness.

5 Experiments

Setup. We conducted two lines of experiments to test different research hypotheses. All the experiments were conducted using the Dreamer model either as a baseline or as a base model for TAPIG. In the first setting, we assume that our agent doesn't have any prior experience, and the addressed dataset is the agent's training buffer. This setting can test the addressing model without any inter-task discrepancies, i.e., its capabilities to bootstrap the training dataset of the agent. The second setting assumes that we are given a multitask experience buffer from some agent trained on different task distribution. The model is then trained from scratch using this experience dataset to solve both these and new tasks leveraging the similarity between the tasks. This way, we can evaluate the prioritization in the case of tasks with a common causal structure. To test the first hypothesis, we run experiments on three environments from DMC [32]: Hopper Hop, Finger Spin, and Acrobot Swingup. For the second one, we build a custom environment based on Metaworld [33] which we call the Rotated Drawer World. In the environment, the goal is to open the drawer placed on a circle depending on the task vector.

Self-prioritization experiment. In the first experiment, we test the model in the self-prioritizing regime, i.e., when the addressing model bootstraps the training dataset of the agent. We compare the performance of the TAPIG agent and vanilla Dreamer on Finger Spin, Hopper Hop, and Acrobot Swingup from the DMC. The performance of the Dreamer on Hopper Hop and Finger Spin shows volatile performance among different runs. This is probably due to the extreme sensitivity of these environments to the initial conditions and other factors of variation. On the contrary, our approach optimizes the performance for the Finger Spin domain and, second, makes it more predictable in terms of guaranteed performance achieved, which is higher for the TAPIG, as can be observed on Hopper Hop. On the Acrobot Swingup task, the performance and learning process is more stable for both models, which, however, perform equally. We hypothesize that the task is too unstable, and both agents have converged to the same suboptimal strategy of repeatedly rotating the joints. We repeated the runs three times for each task and algorithm with different random seeds. The results of this experiment are shown in Figure 2. We plot the median performance and fill the area between the best and the worst runs.

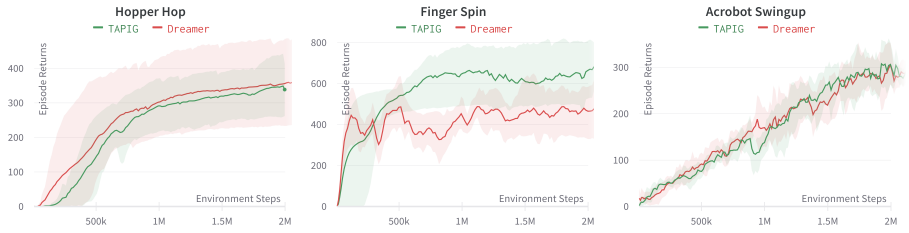

Fig. 2: Performance plots for self-prioritization experiment. For each task, we report the performance of the Dreamer baseline and TAPIG in absence of external data.

Even in the absence of external data, our algorithm shows stable improvements over baseline in different qualities. Our results show that the proposed algorithm has such benefits as a more stable learning process and lower variance over different runs while having comparable or better performance than the base algorithm. We should note that the algorithm's performance depends on the softmax temperature parameter used by the addressing network. While we find the choice of $t = 1/70$ to be good in most cases, for Hopper Hop, the temperature value of $t = 1/10$ has shown the best performance. Such squeezing values of temperature turned to be optimal since the size of the categorical distribution to sample from is proportional to the number of environment samples.

In Figure 3 we show three metrics that qualitatively describe the behavior of the addressing model. The pairwise JSD is is a Jensen-Shannon distance between the address distributions for different agent trajectories, i.e., $\text{JSD}(q(j \mid x, M) \| q(j \mid y, M))$, averaged over many (x, y) pairs. This metric measures the flexibility of the addressing distribution given different chunks of episodes and different tasks. The selected advantage is an average difference between the addressed and uniform rewards, i.e., $\mathbb{E}_{q(j|x,M)} R(M_j) - \mathbb{E}_{M \sim \mathcal{M}} R(M)$. This metric shows the level of reward improvement of the trajectories selected by the addressing model. The last metric is the entropy of the addressing distribution averaged over multiple agent trajectories. The key insight from working with the described addressing model is that its distribution should be highly adaptive to the agent's trajectory. In that case, the model can select randomized experiences by using randomized agent trajectories. We found this to be more critical for the model to have good performance than a high entropy. This is probably because high entropy of the addressing model means a wide range of behavioral features that are treated to be similar, i.e., they are too abstract, so the addressing cannot rely on them for robust learning. In opposition, high distribution adaptability can result in equally high entropy through the marginalization over x of $q(j \mid x, M)$ allowing much broader feature diversity. Another observation is that the addressing model should maximize reward by selection to succeed. Ideally, it should balance a trade-off between reward maximization, high entropy, and adaptability. For example, in the Hopper task, the model has reached only high entropy and only high reward in the Acrobot task. On the other hand, the Finger spin task had reached a balance and thus succeeded.

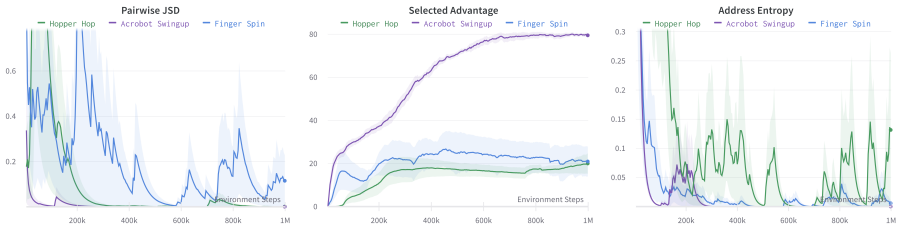

Fig. 3: The metrics describing qualitative behaviour of the addressing model collected during the training of the TAPIG model in DMC environments.

Task prioritization experiment. In order to test how the model can benefit from the use of the multitask data, we run the Rotated Drawer tasks experiment. The TAPIG model was given access to multitask experiences where tasks filled only half of the task space. This was done to test how the agent could first gain from the multitask experience and, second, efficiently reuse its own experience as the agents were trained on the whole tasks range. The results are shown in Figure 4, left, the model show nearly the same convergence speed as the one of the Dreamer baseline. However, the Dreamer baseline's resulting performance is lower in terms of the generalization ability. In Figure 4, right, we plot the evaluation reward for all angles within the whole range. Grey zones indicate the task set provided to the TAPIG model as a multitask dataset for the addressing model. However, the Dreamer baseline has high performance at the end of the training. Its generalization abilities have a zone of performance decline for a horizontal drawer position at the zero angle. On the other hand, the TAPIG model can guarantee a better performance for the whole range of tasks. These two models have equal training rewards but different evaluation rewards. This can be explained by the fact that the training reward is equivalent to an average radius among all angles. Therefore, an average evaluation return is almost the same for these two models, whereas the minimal guaranteed performance is much higher for the TAPIG model.

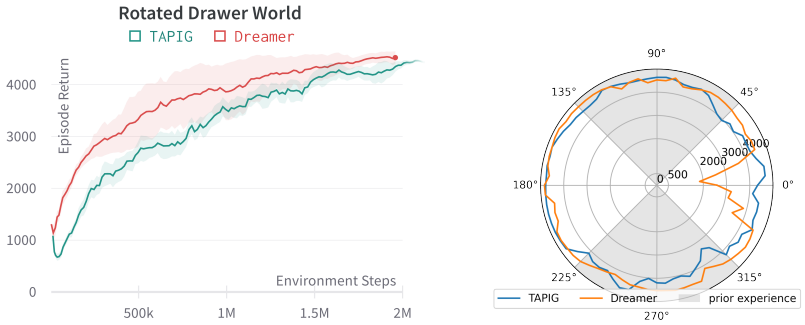

Fig. 4: Performance plots for multitask experiment. For each task, we report the performance of the Dreamer baseline and TAPIG.

6 Discussion and conclusion

The results suggest that the addressing model can facilitate an implicit task generalization and guarantee better performance. However, many tasks are needed to tune the softmax temperature, which we fund very sensitive to the environment. One other limitation is a requirement of the prior data. However, as we have shown in the first experiment, this can be partially mitigated by bootstrapping the data. The method is applicable for a harder environment and can boost the performance without any architectural changes of

the underlying model. We guess that the model reorganizes its capacity to handle better other features of the environment that lead to better rewards. This, however, means that the resulting model is less transferable to different tasks in a zero-shot manner. We then outline a prominent future research direction of making the model more robust in this sense. The metrics we have shown reveal that the addressing can mix different behavioral features that increase the diversity of training samples and the performance of the agent.

In this paper, we considered the problem of influencing the training process of the model-based RL agent. We identified a gap in the current research directions and proposed a new method that can be applied in both single and multitask settings to boost the performance of the model-based RL algorithms. We proposed a mechanism that selectively samples training batches that can gain the performance of the model-based RL algorithm. This model can be trained in two ways: based on value function and REINFORCE loss. We show how the model can increase its sample efficiency without any architectural changes using the DMC suite. We demonstrate how the model can facilitate implicit task generalization using a crafted Rotated Drawer World environment. As a potential future direction, we suggest making the addressing more adaptive to the training in more diverse tasks sets e.g., by providing it the task information explicitly.

References

1. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.
2. Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object Goal Navigation using Goal-Oriented Semantic Exploration. pages 1–11, jul 2020.
3. Aleksei Staroverov and Aleksandr Panov. Hierarchical Landmark Policy Optimization for Visual Indoor Navigation. *IEEE Access*, 10:70447–70455, 2022.
4. Lingli Yu, Xuanya Shao, Yadong Wei, and Kaijun Zhou. Intelligent land-vehicle model transfer trajectory planning method based on deep reinforcement learning. *Sensors (Basel, Switzerland)*, 18, 09 2018.
5. Gregory Gorbov, Mais Jamal, and Aleksandr Panov. Learning Adaptive Parking Manevours for Self-Driving Cars. In *Proceedings of the Sixth International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'22). IITI 2022. Lecture Notes in Networks and Systems*, 2022.
6. Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. In *ICLR 2020*, pages 1–20, 2020.
7. Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.
8. Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. *CoRR*, abs/1706.05064, 2017.
9. Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, 2016.
10. Alexander C Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. In *ICLR 2020*, pages 1–15, 2020.

11. Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.
12. Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.
13. Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov, and Aleksandr I. Panov. Forgetful experience replay in hierarchical reinforcement learning from expert demonstrations. *Knowledge-Based Systems*, 218:106844, 2021.
14. Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov, and Aleksandr I. Panov. Hierarchical Deep Q-Network from imperfect demonstrations in Minecraft. *Cognitive Systems Research*, 65:74–78, 2021.
15. Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Model-based reinforcement learning: A survey, 2021.
16. Artem Zholus and Aleksandr I Panov. Case-based Task Generalization in Model-based Reinforcement Learning. In Ben Goertzel, Matthew Iklé, and Alexey Potapov, editors, *Artificial General Intelligence. AGI 2021. Lecture Notes in Computer Science*, volume 13154, pages 344–354. Springer International Publishing, 2022.
17. Aleksandr I. Panov. Simultaneous Learning and Planning in a Hierarchical Control System for a Cognitive Agent. 83(6):869–883, 2022.
18. Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *CoRR*, abs/2111.09794, 2021.
19. Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020.
20. Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.
21. Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *CoRR*, abs/2010.03934, 2020.
22. Jinsung Yoon, Sercan Ömer Arik, and Tomas Pfister. Data valuation using reinforcement learning. *CoRR*, abs/1909.11671, 2019.
23. Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.
24. David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. https://worldmodels.github.io.
25. Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.
26. Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.
27. Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.
28. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
29. Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.
30. Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control, 2017.
31. Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning, 2019.

32. Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. `dm_control`: Software and tasks for continuous control. 2020.
33. Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021.