MT-Dreamer: Efficient Multi-Task Replay for Model-Based Deep Reinforcement Learning

Anonymous Authors[1]

Abstract

Reinforcement learning agents operating in real environments may be asked to solve many tasks over their operational lifetimes. While attempting to perform one task, such an agent often collects experience relevant to many others, but such data is typically off-policy and therefore challenging to exploit; nevertheless, it should be exploited. Therefore, we first formalize the problem setting as a shared environment with multiple tasks and reward streams. Then, we provide a taxonomy of replay strategies in this setting and propose a novel approach to shared environment multi-task replay, where off-policy task completions are balanced with on-policy task assignments during replay relabeling. We compare our method's performance to alternative task relabeling strategies in a modified Crafter domain, where tasks are assigned in a random sequence until the agent dies or all the tasks are completed. Rewards and termination conditions are provided for each task simultaneously, although the agent is only evaluated on the sequence of assignments. Our results show that our novel replay strategy can exploit multiple streams of sparse reward without neglecting assigned tasks when combined with the deep model-based RL algorithm DreamerV2.

1. Introduction

Reinforcement learning agents operating in real environments may be asked to solve many tasks over their operational lifetimes. While attempting to complete one of these assigned tasks, an agent may encounter data relevant to tasks other than the one currently assigned. If available, an agent may find it useful to have access to the counterfactual answer to the question, "If I was currently assigned task A (although I am solving task B), how would I be doing?". As a motivating example, consider an agent in a kitchen, who has been tasked to make a series of dishes, including baking a cake, finding a mug, and brewing coffee. While exploring the kitchen to bake a cake, the agent may encounter and manipulate a variety of items, such as a coffee mug. If the next task is to find the mug, then the experience of previously finding the coffee mug collected during the task "bake a cake" is of course highly relevant, despite the current assignment being essentially unrelated. By relabeling the experience with the other task, and changing the rewards, an agent can learn to exploit such chance encounters systematically.

Goal relabeling is one such method which might seem appropriate for such a situation (Andrychowicz et al., 2017). However, not all kitchen configurations result in a finished dish, and so goal relabeling would require an additional transformation of the task space, and may not be able to exploit the original task structure in the organic manner as above. In this paper, we propose a new method which can exploit task structure in the multi-task learning setting.

Most reinforcement learning algorithms were not designed to handle multiple, simultaneous reward streams. For example, many popular model-free deep reinforcement learning techniques, such as PPO (Schulman et al., 2017) or soft-actor critic (Haarnoja et al., 2018), may struggle to make efficient use of multiple reward streams, due to the off-policy nature of the alternative rewards. Others, like DQN (Mnih et al., 2013), are capable in principle of off-policy learning but may still have difficulty in practice (Van Hasselt et al., 2018). One early paper in this direction (Silver et al., 2017), learns a latent predictive model from multiple reward streams, although it stops short of addressing the full control aspect of the reinforcement learning problem. On the other hand, many components of model-based (Atkeson & Santamaria, 1997; Moerland et al., 2020) deep reinforcement learning algorithms, such as MuZero (Schrittwieser et al., 2020), or DreamerV2 (Hafner et al., 2020), translate well out of the box to multi-task learning, as learning a latent transition function and observation function need little to no modification with the introduction of multiple rewards. However, many of these model-based algorithms utilize model-free algorithms to learn a policy in an inner loop, and were likely not designed with multi-task environments particularly in mind.

Finally, there is the complex question of when or how best to relabel. In this work, we explore the performance of several different relabeling strategies, paired with the model-based deep reinforcement learning algorithm DreamerV2, and propose our solution: MT-Dreamer. We provide a comprehensive framework for generating and comparing replay strategies that focuses on the temporal relationship that a single piece of experience has with a variety of tasks. Each replay strategy considered makes different decisions based on those relationships. To thoroughly evaluate these strategies, we compare MT-Dreamer with these strategies on a novel multi-task version of Crafter. (Hafner, 2021). Empirically, we find that a strategy that balances both the on-policy nature of assigned task experience, while replaying successful task completions regardless of assignment is the most effective.

2. Preliminaries

2.1. Reinforcement Learning with Multiple Tasks

We consider a shared environment, multi-task reinforcement learning setting, in which an agent interacts with a single environment to achieve various tasks. An environment is described by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, an initial state distribution ρ, transition probabilities $p(s_{t+1} \mid s_t, a_t)$, and a discount factor $\gamma \in [0, 1]$.

In the multi-task setting, there is a task space $\mathcal{T}$ corresponding to the space of possible goals. Each task $\tau \in \mathcal{T}$ maps to some reward function $R_\tau : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a termination function $d_\tau : \mathcal{S} \times \mathcal{A} \to [0, 1]$. If a task τ is terminal at time m and the last reward $r_\tau(s_m, a_m)$ is positive, we call that task *completed*, and if the last reward is negative, we call it *not completed*. This can be easily generalized to an arbitrary binary classifier, but we omit it for clarity. At each timestep t, the agent receives as input the current state s_t and the current task τ, such that $\pi : \mathcal{S} \times \mathcal{T} \to \mathcal{A}$. After executing an action, the agent receives the task-specific reward $r_t = r_\tau(s_t, a_t)$. If the current task terminates, then the next task is sampled from any task in $\mathcal{T}$ that is not already terminated. If there are no more tasks to sample, than the episode terminates.

Reinforcement learning algorithms fall into two categories: model-free and model-based. Model-free algorithms estimate the value function and/or policy through directly interacting with the environment. In contrast, model-based methods approximate the environment dynamics, such as the transition and reward, with a model to assist in this learning process. After the agent repeatedly interacts with the environment, the experienced transitions are stored in a dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, \gamma_t\}$. The agent then uses these experiences to estimate a model $\tilde{M}$. We discuss the specific model-based learning algorithm used as the backbone of our work below.

2.2. DreamerV2

DreamerV2 (Hafner et al., 2020), an algorithm in the Dreamer family (Hafner et al., 2019a; 2023), was the first MBRL method to achieve human-level performance on Atari (Bellemare et al., 2013; Machado et al., 2018). It learns behaviors from the compact latent space of a learned world model, which is trained independently from the policy. DreamerV2 consists of an experience dataset $\mathcal{D}$ for training the world model, a world model $\tilde{M}$ for imagining sequences of compact model states, and an actor π and critic Q^π for behavior learning.

The world model $\tilde{M}$ consists of multiple model components: an image encoder, a recurrent state-space model (RSSM) (Hafner et al., 2019b) for dynamics learning, and a set of predictors to reconstruct the image, reward, and discount factor. The RSSM consists of three parts: a recurrent model that produces the deterministic recurrent state h_t, a representation model that produces a stochastic latent posterior state z_t by explicitly incorporating information about the current image x_t, and a transition predictor that produces the prior state $\hat{z}_t$, which aims to predict the posterior without access to the current image.

2.3. Experience Replay

Experience replay is a standard technique in deep reinforcement learning to improve sample efficiency and stability of training (Lin, 1992; Fedus et al., 2020). It consists of a fixed-size replay buffer that typically holds a large number of the most recent transitions collected by the policy. During training, the algorithm extracts samples from the buffer to perform updates. Because data can be resampled multiple times, this technique offers the benefit of sample efficiency. The randomized sampling also increases stability, as consecutive gradient updates become more decorrelated than if they were applied to the data in a strictly temporal order.

The introduction of a deep, generative world model (Ha & Schmidhuber, 2018) allows for the generation of "imagined" data which can augment or in some cases completely replace directly sampled data from the environment. Other more modern model-based RL papers, like the Dreamer family and MuZero, continue to build on model-free techniques for training policies, using an experience replay buffer to initially sample data for replay, then rolling the latent model out some fixed number of timesteps before computing a loss. Training these more competitive methods on off-policy data is still a concern, and may require modifications.

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) introduces a number of potential strategies for goal relabeling, although it assumes that every state fulfills at least one goal, unlike our multi-task setting. Prioritized experience replay (Schaul et al., 2015) provides a framework for more frequently replaying important transitions. When applied to DQN, this approach generally outperformed DQN with standard uniform replay. Curriculum guided HER (Fang et al., 2019) is an approach for enabling agents to learn from failed experiences in a goal-based, sparse-reward setting that employs a curriculum of assigned goals. likelihood free importance weights (Sinha et al., 2022b), experience replay optimization (Zha et al., 2019), competitive experience replay (Liu et al., 2019), continuous transition (Lin et al., 2021), surprisingly simple self-supervised reinforcement learning (S4RL) (Sinha et al., 2022a), neighborhood mixup experience replay (Sander et al., 2022), are all additional techniques which extend and build upon the foundational ideas of experience replay.

3. Improving Multi-Task Replay

We present MT-Dreamer, an extension of the Dreamer family of algorithms to the multi-task setting. We first specify the changes needed to adopt Dreamer to this setting, then delve into the additional considerations implicated in performing experience replay in the context of multiple reward streams. To that end, we present a taxonomy for thinking about and organizing multi-task experience replay strategies. We then instantiate a subset of potential replay strategies and explain them in detail.

3.1. MT-Dreamer

Like DreamerV2, MT-Dreamer consists of the standard components for a model-based agent: a world model that is learned from data, an actor and a critic that are trained using the model-generated sequences of latent states, and an experience dataset that is collected by the actor to continue training the model. However, we introduce a few critical changes to utilize Dreamer in the multi-task setting. We detail these changes here.

Model Replay Buffer. Previous versions of Dreamer have no notion of an assigned task. In all experiments, we represent the assigned task as a one-hot encoding. However, our framework can easily accommodate other task representations. Although an agent is assigned an initial task $\tau \in \mathcal{T}$ at the start of an episode, upon completion, another task will be assigned. Because the temporal assignment of tasks factors heavily in the differences between candidate replay strategies, we must store the assigned task at each timestep $\tau_{1:T}$ in our model replay buffer. We store these assigned tasks in addition to the standard components that are stored in the world model replay buffer (images, actions, rewards, task completions, and discount factors). Task rewards $\vec{r} \in \mathbb{R}^{|\mathcal{T}|}$ and completions $\vec{d} \in \{0,1\}^{|\mathcal{T}|}$ are stored as vectors. These changes allows us to explore a variety of replay strategies, as we can distinguish between replaying the *assigned* task and alternative tasks, and carefully consider the temporal order of events as an episode plays out. We define the function used for the replay strategy as f.

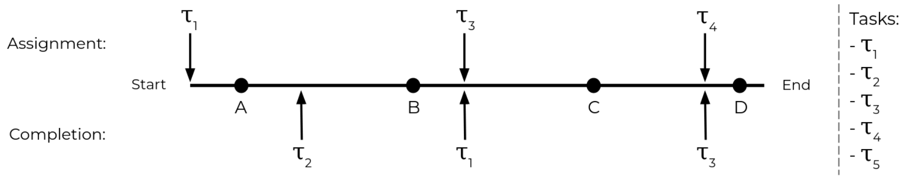
Model Components. To handle the multi-task setting, we introduce an additional task predictor that takes as input a one-hot vector corresponding to the assigned task τ_t and aims to produce a reconstruction of the task $\hat{\tau}_t$. This task predictor is implemented as an MLP. We believe that task conditioning will be useful for representation learning; as a result, we additionally augment the representation model in DreamerV2. In DreamerV2, the representation model serves to output a distribution over the posterior state z_t. This posterior state z_t is given access to the current image x_t; the prior state $\hat{z}_t$ is not. We augment the representation model to additionally take as input the assigned task τ_t, such that the representation model becomes: $z_t \sim q_\phi(z_t|h_t, x_t, \tau_t)$. The prior state $\hat{z}_t$ aims to predict the posterior without access to the current image or the task vector. We therefore keep the transition predictor, which outputs the prior state, unchanged from DreamerV2. This choice enables us to learn behaviors by predicting sequences of model states with the RSSM without needing to observe or generate images or task vectors.

Model Learning. We jointly optimize all components of the world model. Given that we are in a multi-task setting, we must now introduce the concept of a task in the loss function. The chosen sampling strategy f determines the task τ that is used for replay at each timestep t. The loss of all predictors, including our proposed task predictor, are the log-likelihoods of their corresponding targets. We also include the KL-balancing loss β KL $[\cdot]$ for training the prior toward the representations and regularizing the representations toward the prior. However, we include one key change: the representation model (or approximate posterior) now takes as input the task vector τ_t in addition to the current image x_t and deterministic recurrent state h_t, giving us $q_\phi(z_t|h_t, x_t, \tau_t)$. The MT-Dreamer loss function is therefore:

$$\begin{aligned}\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(z_{1:T}|a_{1:T}, x_{1:T}, \tau_{1:T})}[\sum_{t=1}^{T} &-\ln p_\phi(x_t|h_t, z_t) \\ &- \ln p_\phi(r_t|h_t, z_t) - \ln p_\phi(\gamma_t|h_t, z_t) \\ &- \ln p_\phi(\tau_t|h_t, z_t) \\ &+ \beta \text{ KL } [q_\phi(z_t|h_t, x_t, \tau_t)||p_\phi(z_t|h_t)]].\end{aligned}$$

3.2. Replay with Task Assignment

The adoption of the multi-task framework means there are additional possibilities for experience replay during training. We now present a taxonomy for thinking about and



Time	τ_1	τ_2	τ_3	τ_4	τ_5
A	Past assigned Future completed	Never assigned Future completed	Future assigned Future completed	Future assigned Never completed	Never assigned Never completed
B	Past assigned Future completed	Never assigned Past completed	Future assigned Future completed	Future assigned Never completed	Never assigned Never completed
C	Past assigned Past completed	Never assigned Past completed	Past assigned Future completed	Future assigned Never completed	Never assigned Never completed
D	Past assigned Past completed	Never assigned Past completed	Past assigned Past completed	Past assigned Never completed	Never assigned Never completed

Figure 1. A sample timeline for a sample episode. T denotes tasks; A, B, C, and D are time points corresponding to four intervals determined by task completion and task assignments. The arrows on the upper part of the timeline show when the tasks are assigned, whereas the bottom part shows when the tasks are completed.

organizing replay strategies in this setting. In a multi-task sequential decision-making problem, where multiple tasks are assigned and completed during an episode, for any given timestep t, tasks τ can be categorized into three categories with respect to their assignment time (past, future, and never) and also into three categories similarly with respect to their completion time. To illustrate these possible categorizations, we provide an example in Figure 1. There are five possible tasks (τ_1, τ_2, τ_3, τ_4, and τ_5) in the environment. We highlight four points in time during the environment rollout, $t = A, B, C, D$, where A occurs before B, B occurs before C, and C occurs before D.

At the start of the episode $t = 0$, a task τ_1 is assigned to the agent according to some task assignment distribution $\rho_{\mathcal{T}}$. For the time point $t =$ A, therefore, τ_1 is assigned in the past (**past assigned**). Given that τ_1 is completed after $t = B$, it is also completed in the future with respect to $t = A$, making it also **future completed**. This assignment is also reflected with respect to time point $t = B$. However, note that the agent completes τ_1 between time points B and C. This task completion means that, with respect to time point C, τ_1 has now been completed in the past (**past completed**). This change is also reflected in its categorization with respect to time point D.

Now consider τ_2. During the course of the entire episode, the agent is never assigned τ_2, so it is always categorized as **never assigned** for all time points in the episode. However, the agent accomplishes this task after $t = A$. Therefore, τ_2 is categorized as **future completed** with respect to $t = A$. For all other time points, it is categorized as **past completed**.

We now contrast τ_3 and τ_4. With respect to timepoint A, τ_3 and τ_4 are assigned in the future, making them both **future assigned**. However, the agent completes τ_3 after timepoint C but before timepoint D, so it is considered **future completed** with respect to time points A, B, and C. In contrast, the agent never accomplishes τ_4, so it is categorized as **never completed** with respect to all considered time points.

Finally, consider τ_5. It is never assigned over the course of the episode, and the agent never accomplishes it. Therefore, it is categorized as **never assigned** and **never completed** for all considered time points.

3.3. Replay Strategies

Our task taxonomy implies a variety of replay strategies. In particular, different combinations act as filters for data to consider as replay. For example, when we perform experience replay, we can choose to only consider tasks that were assigned and completed over the course of that episode. In this case, we do not consider other tasks that were assigned but not completed or tasks that were completed but not assigned. More generally, under our proposed taxonomy, there are nine possible task categorizations: for assignment and completion, we choose from past, future, and never. However, including data from tasks that have been completed in the past for any assignment category would break the causality component of the sequential learning problem. For that reason, we focus on the six remaining categorizations, which we show in Table 1.

We note that these filters can be combined to include potentially even more data for replay. For example, we may want to include data for tasks that have been assigned in the past and not completed, as well as tasks that have been assigned in the past and completed. Considering data in this

Strategies	Task Variations					
	Never a. Never c.	Future a. Never c.	Never a. Future c.	Future a. Future c.	Past a. Never c.	Past a. Future c.
No relabel	✗	✗	✗	✗	✓	✓
Future completed	✗	✗	✓	✓	✗	✓
Completed or past assigned	✗	✗	✓	✓	✓	✓
All assigned	✗	✓	✗	✓	✓	✓
All assigned or completed	✗	✓	✓	✓	✓	✓
All tasks	✓	✓	✓	✓	✓	✓

Table 1. Strategy variations

way yields 64 possible ways to include data for replay. Each way corresponds to a different replay strategy. Because accomplishing an assigned task is the primary goal of task-conditioned RL, we always include this data in learning, which decreases the candidate strategies to 32. To reduce them even further, we categorize task variations based on whether they are positive experiences or negative experiences or whether they are on-policy or off-policy. Then, we select the strategies that are distinctive for these categories to see their contribution to the performance, such as all positive, all on-policy, on-policy and positive, etc. Consequently, we choose six to thoroughly empirically evaluate. We now describe each of the six included strategies in turn.

No Relabel (NR). This strategy includes experiences with the assigned task labels where tasks can be succeeded in the future or never succeeded. There is no relabeling done for unassigned tasks. Therefore, this is the closest strategy to the standard training of the RL methods in this environment because it only retains on-policy data.

Future Completed (FC). This strategy considers only the tasks that are completed in the future where they can be assigned anytime or not assigned at all. Past successes are not included because that would again break the causality. The models are trained with sequences relabeled with tasks whose solution they lead to as if these tasks are assigned. In case there are multiple solved tasks, one of them is randomly sampled. Therefore, this strategy focuses only on positive experiences.

Completed or Past Assigned (CPA). This strategy extends the FC strategy with one difference: it also includes tasks that were assigned in the past and never completed. By including this data, we selectively test the contribution of on-policy negative experiences.

All Assigned (AA). The all assigned (AA) strategy includes all tasks that have been assigned at any point, regardless of whether they have been completed. In comparison with no relabel strategy, this strategy relabels experiences with future assigned tasks and considers that these off-policy experiences are related even if those tasks are not currently assigned.

Figure 2. A sample picture from test environment Crafter. The agent (blue shirt, center of the screen) has successfully made a wood pickaxe and sword using the crafting table.

All Assigned and Completed (AC). This strategy extends the completed or past assigned strategy with the second-order related negative experiences, where a task is assigned in the future but never completed. Like AA, this also considers that off-policy experiences are related to future assigned tasks.

All Tasks This strategy includes the remaining experiences where tasks are never assigned and never completed. In the end, it includes all task variations.

4. Experiments

Equipped with our MT-Dreamer model, we now investigate which of the experience replay strategies yields the best performance on a modified version of Crafter (Hafner, 2021).

4.1. Experimental Setup

Multi-Task Crafter Crafter (Hafner, 2021) is a relatively new benchmark for key challenges in reinforcement

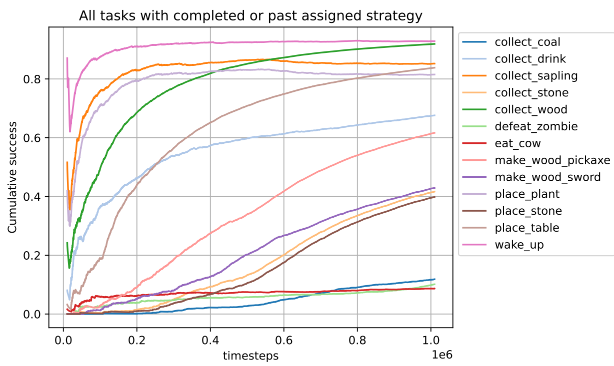


Figure 3. 1M training run for the best strategy.

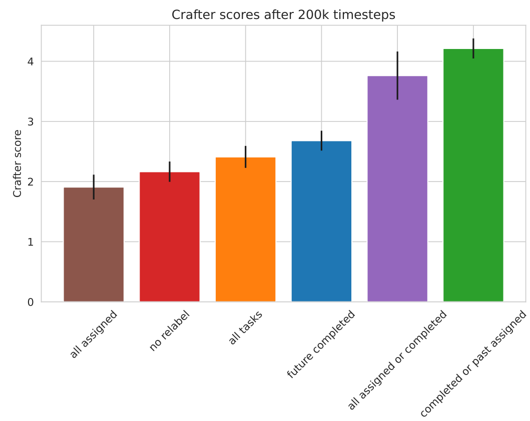


Figure 4. Overall performances of the strategies. We compute the performance as the geometric mean over all tasks, which enables rarer tasks to have a higher influence on the score. Error bars represent the standard deviation across 3 trials. The *completed or past assigned* strategy exhibits a higher task performance than the other strategies.

learning, including generalization, exploration, representation learning, long-term reasoning, and credit assignment. Crafter is inspired by Minecraft, a challenge that is of great interest to the research community (Aluru et al., 2015; Johnson et al., 2016; Guss et al., 2019). Like Minecraft, Crafter includes an item hierarchy that an agent traverses by walking around, collecting objects, and crafting. At the same time, the agent must stay alive by warding off enemies, finding shelter, and eating. There are 22 skills in total, ranging from simply waking up from sleep, all the way to mining diamond. Figure 2 shows a sample picture of the environment, where the agent was able to collect wood and made wood pickaxe and sword using the crafting table. Although Crafter has an implicit notion of skills through the reward signal — the first time each of the 22 possible skills is achieved in an episode provides the agent with some reward — the agent has no explicit notion of task. To adopt Crafter to the multi-task setting, we introduce Multi-Task Crafter. This environment is the same as Crafter, except we restructure the reward function to evaluate the episode for all 22 tasks separately. This change enables the evaluation of various task relabeling strategies. To further emphasize the focus on sample efficiency, we propose an additional, more data-constrained setting: we reduce the original sample budget of 1 million timesteps to 200,000 to evaluate all strategies. Then, we train the best 2 strategies with 1 million timesteps to distinguish them clearly in longer time horizons.

4.2. Results

We now present our main findings. We find that overall, task relabeling is more helpful than not for most of the strategies. Including the positive reward trajectories has a greater impact than off-policy negative reward trajectories, although the on-policy negative reward trajectories are still important. We use two different metrics to assess performance. The first metric computes the score over all completed tasks in that episode (**Overall Score**). The second metric only assesses which of the assigned tasks have been successfully completed (**Assigned Score**). In both cases, we present the Crafter score in order to assess an agent's capabilities on all tasks. This score assesses whether an agent completes a task at least once during an episode, then takes a geometric mean over the task completions. We further decompose the score into task-specific completion, which we detail in Appendix A.

Figure 3 illustrates the typical evolution of task accuracies for a given run of multi-task Crafter. Shown is a run for one million timesteps for the strategy *completed or past assigned*. The performance drop at beginning of the graph indicates the start of the training, where the agent switches from the random policy that filled the replay buffer initially. The tasks that are instantly available with possible actions are easily learned, such as waking up or collecting a plant. Then, it learns to do more complex tasks at the bottom of the hierarchy that requires moving to the correct position, like collecting water and wood. Later, it starts to pick up hierarchically more complex tasks like making a wooden pickaxe and collecting stones. For clarity, tasks that were not successfully completed at least once are omitted.

In Figure 4, we see the various strategies sorted by their Crafter score achieved after 200k timesteps. The *completed and past assigned* strategy has the highest mean Crafter Score at 4.2, followed by *all assigned and completed* at 3.76. *future completed* has the third place at 2.68, and *no relabel* is at 2.16. Finally, *all assigned* is last, with a score of only 1.9.

From the Crafter scores, we can see an overall improvement from the use of relabeling. Figure 5 shows the success rate

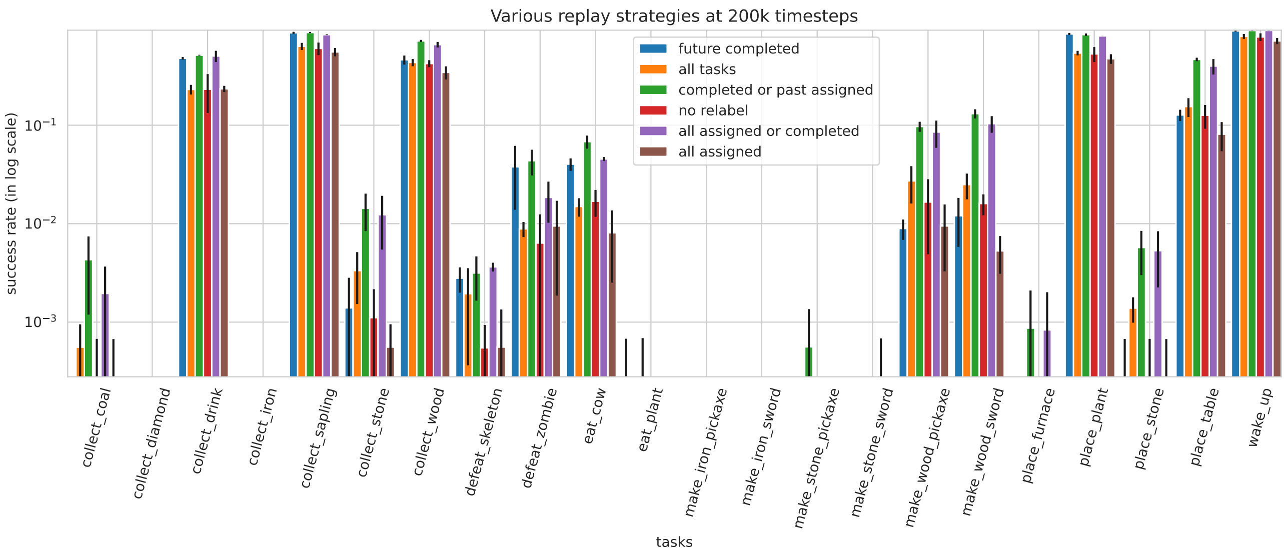


Figure 5. Full task results

	AC	CPA
Assigned Crafter Score	9.23	9.76
Overall Crafter Score	9.04	9.56

Table 2. Crafter scores for the two best replay strategies after 1M timesteps. As the agents are rewarded on the basis of the assigned tasks, they are more likely to complete a task during an episode it is assigned. Changes to the task-conditioning could likely improve this gap, although it was not the primary aim of this paper.

for each task/strategy pair independently. Looking at the figure, we see that although the overall gap between *completed or past assigned* and *all assigned or completed* is not outside of a standard deviation, we see that for 15 out of 22 tasks, *completed or past assigned* is outperforming *all assigned or completed*. We also see that *future completed* is nearly matching *completed or past assigned* and *all assigned or completed* in the easier tasks, like wake up, collect sapling, or place plant, but is far behind in the rarer tasks, like place furnace, collect coal, or even place table.

We selected the two best strategies and trained for an additional 1 million timesteps, in order to better compare the strategies. We present the results of this experiment in Table 2. In this regime, *completed or past assigned* maintained its small but superior performance over *all assigned or completed* for both **Assigned Score** and **Overall Score**. We find that both replay strategies achieve higher **Assigned Scores** than **Overall Scores**.

4.3. Discussion

Our results show that the performance of the model depends on several data factors such as: the data amount, positive-negative experience, and off-policy. The two worst strategies are *all assigned* and *no relabel* because the only positive experience they get is completion of a past assigned task. This data is very sparse, especially at the early stages of training. *All assigned* does worse than *no relabel* because off-policy data (**future assigned**) hurt the training. However, this effect is not large because the agent must complete an assigned task in order to get a future assignment, and this case is rare (as explained before). Although *all future completed* does not utilize any negative experiences like (past/future assigned never completed), task relabeling of the off-policy positive experiences (never assigned future completed) improves the performance because these experiences happen a lot since the agent often solves unassigned easy tasks. *Completed or past assigned* adds only the on-policy negative experiences (past assigned never completed), but this improves the performance greatly and makes it the best strategy. *All assigned and completed* does slightly worse than *completed or past assigned* in both 200K and 1M runs because the off-policy negative experiences (**future assigned** and **never completed**) apparently hurt training. *all tasks* performs worse than all future completed because we add too much off-policy negative experiences with never assigned and never completed since this data type dominates the dataset. To sum up, the positive experiences greatly help training and justify the use of task relabeling whether they are on or off-policy. Negative experiences are helpful when the data is on-policy, but can substantially hurt the model if they are off-policy, and dominate the dataset.

5. Conclusion

We studied replay strategies for multi-task reinforcement learning in the model-based setting. We first formalized the problem setting as a shared environment of multi-tasks with multiple reward streams and employed a leading model-based RL algorithm to test candidate replay strategies. We provided a comprehensive framework to distinguish replay strategies based on the temporal aspect of task assignment and completion. Our experiments show that the strategy that balances positive experience with on-policy negative experience performs best.

References

Aluru, K. C., Tellex, S., Oberlin, J., and MacGlashan, J. Minecraft as an experimental world for ai in robotics. In *2015 aaai fall symposium series*, 2015.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., et al. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Atkeson, C. G. and Santamaria, J. C. A comparison of direct and model-based reinforcement learning. In *Proceedings of the International Conference on Robotics and Automation*, 1997.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, 2013.

Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, 32, 2019.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, 2020.

Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., et al. The MineRL competition on sample efficient reinforcement learning using human priors. *NeurIPS Competition Track*, 2019.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hafner, D. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019b.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The malmo platform for artificial intelligence experimentation. In *Ijcai*, pp. 4246–4247, 2016.

Lin, J., Huang, Z., Wang, K., Liang, X., Chen, W., and Lin, L. Continuous transition: Improving sample efficiency for continuous control problems via mixup. In *International Conference on Robotics and Automation*, 2021.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3), 1992.

Liu, H., Trott, A., Socher, R., and Xiong, C. Competitive experience replay. *arXiv preprint arXiv:1902.00528*, 2019.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Moerland, T. M., Broekens, J., and Jonker, C. M. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Sander, R., Schwarting, W., Seyde, T., Gilitschenski, I., Karaman, S., and Rus, D. Neighborhood mixup experience replay: Local convex interpolation for improved sample efficiency in continuous control tasks. In *Learning for Dynamics and Control Conference*, 2022.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pp. 3191–3199. PMLR, 2017.

Sinha, S., Mandlekar, A., and Garg, A. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, 2022a.

Sinha, S., Song, J., Garg, A., and Ermon, S. Experience replay with likelihood-free importance weights. In *Learning for Dynamics and Control Conference*, 2022b.

Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Zha, D., Lai, K.-H., Zhou, K., and Hu, X. Experience replay optimization. *arXiv preprint arXiv:1906.08387*, 2019.

A. Additional Experimental Results

Achievements	No Relabel	All Assigned	CPA	Future Completed	All Assigned or Completed	All Tasks
collect coal	0.0003	0.0003	0.0043	0.0000	0.0020	0.0006
collect diamond	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
collect drink	0.2319	0.2341	0.5145	0.4780	0.5042	0.2310
collect iron	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
collect sapling	0.6031	0.5549	0.8741	0.8674	0.8300	0.6336
collect stone	0.0011	0.0006	0.0143	0.0014	0.0123	0.0033
collect wood	0.4228	0.3438	0.7182	0.4615	0.6586	0.4340
defeat skeleton	0.0005	0.0006	0.0032	0.0028	0.0036	0.0019
defeat zombie	0.0064	0.0095	0.0436	0.0378	0.0185	0.0089
eat cow	0.0169	0.0081	0.0681	0.0402	0.0454	0.0150
eat plant	0.0000	0.0000	0.0003	0.0003	0.0000	0.0000
make iron pickaxe	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
make iron sword	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
make stone pickaxe	0.0000	0.0000	0.0006	0.0000	0.0000	0.0000
make stone sword	0.0000	0.0000	0.0000	0.0000	0.0003	0.0000
make wood pickaxe	0.0166	0.0095	0.0970	0.0089	0.0851	0.0272
make wood sword	0.0160	0.0053	0.1311	0.0120	0.1037	0.0250
place furnace	0.0000	0.0000	0.0009	0.0000	0.0008	0.0000
place plant	0.5300	0.4730	0.8313	0.8445	0.8059	0.5405
place stone	0.0003	0.0003	0.0057	0.0003	0.0053	0.0014
place table	0.1263	0.0810	0.4660	0.1270	0.3986	0.1546
wake up	0.7871	0.7174	0.9158	0.9066	0.9179	0.7988

Table 3. Success rates on the augmented Multi-Task Crafter.

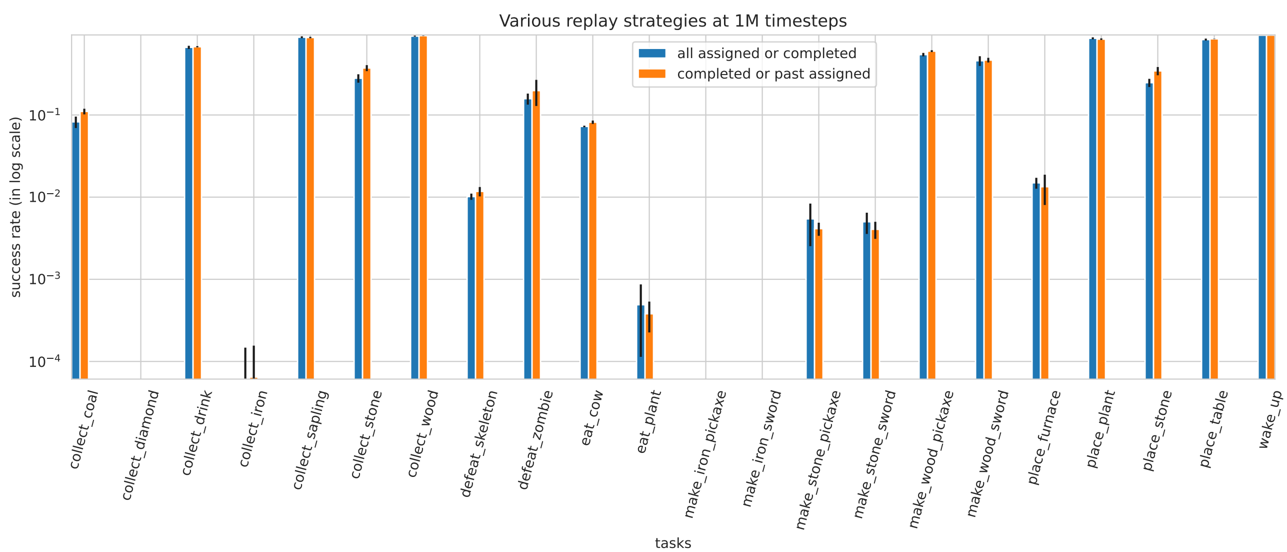


Figure 6. Full task results of the two best strategies in a one million training run. Error bars represent the standard deviation across 3 trials. The *completed or past assigned* strategy exhibits a higher task performance than *all assigned or completed*.