CASH: Cache Alignment with Specified Horizons

Anonymous submission

Abstract

In this work, we offer a new reinforcement learning algorithm: CASH (Cache Alignment with Specified Horizons) to effectively leverage prior knowledge. In this framework, previously computed policies are stored in a cache and compared against rollouts to determine similarity based on a novel metric derived from the Kendall tau distance. We prove error bounds on the estimated value function which allows for an optimizable rollout length connected to classical cost-accuracy tradeoff. We then introduce a second horizon to specify execution time of the best cached policy. We show that even in the case of adversarial caches, the algorithm performs no worse than standard Q-learning. This preliminary work is provides a new perspective at the intersection of transfer learning and model-based reinforcement learning, opening avenues for further investigations.

Introduction

The application of reinforcement learning (RL) to sequential decision-making tasks has seen significant growth in recent years, showcasing its capacity to uncover innovative solutions in complex and unfamiliar environments (Schrittwieser et al. 2020). In RL, model-based methods – which focus on learning or utilizing a model of the environment's dynamics – have consistently demonstrated greater sample efficiency compared to model-free approaches, especially in scenarios with constrained interaction budgets. This efficiency arises from the ability of model-based methods to perform planning over imagined trajectories, employing techniques like Monte Carlo tree search to evaluate and refine policies. Unfortunately, these methods incur higher computational cost and depend heavily on the precision of the learned dynamics model.

In the traditional reinforcement learning (RL) paradigm, solutions to previously learned tasks are often ignored or discarded when tackling a new task. However, new tasks are frequently related to or share features with earlier ones. Consequently, in many cases a significant amount of useful information is thrown away and relearned, causing inefficiencies in training. This has motivated previous work to incorporate learned solutions into training. However, these techniques usually require some understanding of how the current task is related to prior tasks (e.g. in terms of their reward functions, which limits the utility of these approaches. Furthermore, if past tasks do not contain useful information to solve the new task, their incorporation can potentially distract the agent and impede performance instead of improving it. As a result, it remains an open question of how to best incorporate previously learned policies into the training for new tasks.

To remedy this inefficiency, we propose a modified training approach which leverages previously learned policies for general tasks to guide training for the new task. In this approach, short parallelized rollouts are performed for each possible action. The optimal trajectory is then selected based off of the accumulated reward, and then compared with the stored policy cache to identify the most similar corresponding stored policy. We propose a new method to measure the distance betweeen value functions, allowing solutions in the cache to be aligned with the recently experienced online data. Additionally, we propose to include the current solution estimate in the cache itself. Critically, if the cached tasks do not aid in performance, they are ignored, allowing this algorithm to succeed even in the presence of an adversarial cache.

In the following sections, we review related prior work, give background material on the RL problem, then state our results and conclusions.

Prior Work

In previous work, a set of solutions has been used to transfer knowledge to the agent (cf. the review (Taylor and Stone 2009)). Closely related to our approach is the notion of the policy cache with explicit choices for transfer. With the successor feature (Barreto et al. 2017) structure, a policy cache was used to generate bounds and determine additions to the cache (Nemecek and Parr 2021). Other work has focused on combining multiple "subtasks" when a direction relationship over reward functions is known (e.g. linear or logical compositions) (Haarnoja et al. 2018; Adamczyk et al. 2023; Tasse, James, and Rosman 2020; Peng et al. 2019). Compared to such work, we do not assume any functional relationship between the reward functions, and instead use a new metric to define similarity between tasks. The notion of "similarity" is quite broad, and has been discussed e.g. in the context of reward spaces (Wulfe et al. 2022; Skalse et al. 2023), whereas we consider similarity between value functions.

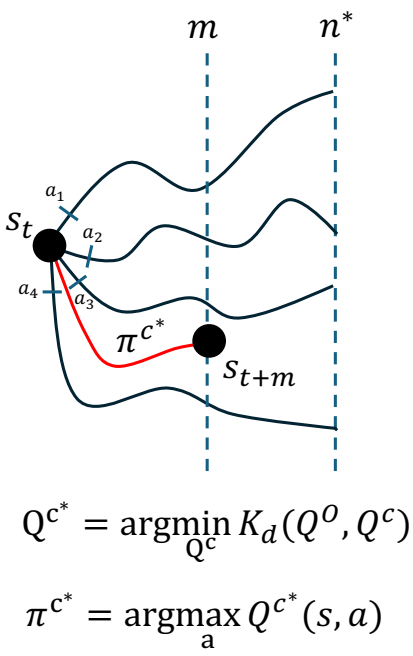

Figure 1: Illustration of the proposed algorithm. As the agent is trained, a trajectory (of length n^*) beginning with each possible action is rolled out, using the online estimate of π^*. The initial actions are then ranked based on the corresponding trajectory returns. Then, the Kendell Tau distance is used to measure the discrepancy between this ranking, and those induced by the cached policies at the same state. The closest policy is used for the next m steps in online environment interaction.

Preliminaries/Background

Reinforcement Learning

In this section we briefly introduce the reinforcement learning framework. The reader is directed to standard texts for more information (Sutton and Barto 2018). We consider the RL problem for discrete state-action spaces. The RL problem can then be modeled by a Markov Decision Process (MDP), represented by the information $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ with state space $\mathcal{S}$; action space $\mathcal{A}$; potentially stochastic transition function (dynamics) $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$; bounded, real reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$; and the discount factor $\gamma \in [0, 1)$.

The primary objective of RL is to maximize the total discounted reward received under the control policy π. Specifically, we wish to find π^* which maximizes the following expectation, where trajectories τ are sampled from the Markov chain induced by the fixed dynamics and optimized policy:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim p, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \quad (1)$$

In the present work, we consider value-based RL methods, where the solution to the RL problem is equivalently defined by its optimal action-value function ($Q^*(s, a)$). Note that we will need the Q function to fully calculate the relative ordering of actions, defining a distance metric between Q functions. The aforementioned optimal policy $\pi^*(a|s)$ is derived from Q^* through a greedy maximization over actions. The optimal value function can be obtained by iterating the following recursive Bellman equation until convergence:

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \max_{a'} \left(Q^*(s', a') \right). \quad (2)$$

In the tabular setting, the exact Bellman equation can be applied until convergence. In the function approximator setting, the Q table is replaced by a parameterized function approximator, Q_θ and the temporal difference (TD) loss is minimized instead. To simplify the discussion, we will neglect the details of precisely how the Q function is derived, and we focus on general value functions alongside their approximations. We note that although this initial investigation is presented in the "standard" Q-learning setting, extensions to maximum entropy setting may also be possible.

Now, we introduce a new metric to relate two tasks. As solutions Q values encode a ranking between any two actions. If two solutions agree, their relative rankings should also be in agreement. This intuition is encapsulated by the following definition. We draw on a well-established definition, stemming from the bioinformatics literature (Kendall 1938):

Definition 1 (Kendall Tau Distance Between Value Functions)**.** The (un-normalized) Kendall tau distance between two policies is defined as

$$K_d(Q_1, Q_2; s) = \sum_{\{i,j\} \in P, i<j} \bar{K}_{ij}(Q_1(s, \cdot), Q_2(s, \cdot)), \quad (3)$$

where P is the set of all (unordered) pairs of actions; $\bar{K}_i j = 0$ if the rankings $Q_1(s, i) > Q_1(s, j)$ and $Q_2(s, i) > Q_2(s, j)$ agree, and otherwise takes on the value 1.

Here we make the state explicit, but we will drop this indexing when it is clear. In the following, we propose an algorithm that aligns task solutions in the cash to the observed data stream, based on this new notion of distance.

Comparing the action rankings rather than direct Q value magnitudes has an important implication for measuring similarity: this distance is invariant to any potential-based reward shaping (Ng, Harada, and Russell 1999) or positive scaling in the reward function. Instead, it directly encodes preferences at the individual action, or policy, level. This definition also allows comparisons in the discrete action case, where typical notions of policy distance would be ill-defined.

CASH Algorithm Description

Next, we describe the CASH algorithm. Starting from state s_t with initialized online Q-function Q_0, compute $|A|$ parallelized rollouts for n^* steps. To gather the Q function over action space (as dictated to compare against the cache), each one of the $a \in |A|$ possible actions, and subsequent actions are dictated by the online Q-function ($\text{argmax}_a\, Q_0(s, a)$). Of the $|A|$ resulting trajectories, the optimal trajectory is

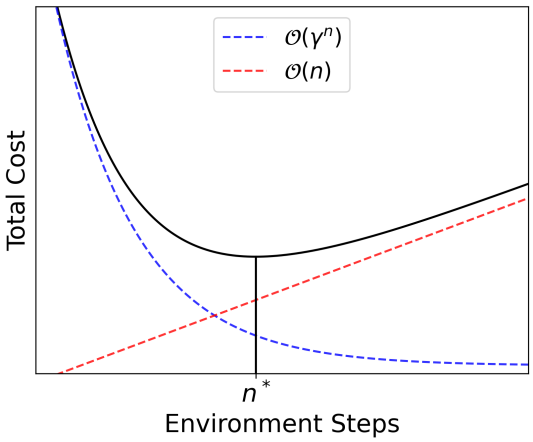

Figure 2: Tradeoff between larger n-step returns for reducing error geometrically, and linearly increasing the cost of compute for rollout length n. The combined cost and error rate yield an optimal choice for trajectory length, denoted n^*.

then identified by selecting the one which maximizes the accumulated return $\sum_{i=0}^{n^*} \gamma^t r_{t+i}$. An estimate of the Q-function $\hat{Q}(s_t, a)$ at state s_t is then given by the n^*-step return and bootstrapped next value. This estimate allows the quality of each action at time t (a_t) to be ranked.

The most similar cached policy, as measured by the minimum Kendall tau distance between the ranked list of actions given by the n^*-step return and each cached policy, is then identified and used to determine action selection for the next m steps. Critically, the online policy given by Q_0 is also in the cache, so the algorithm is guaranteed to not perform worse than naive Q-learning. This procedure is then repeated, starting at state s_{t+m}.

Theory

We first motivate the proposed algorithm by providing an intuition for an improved sample complexity: In the initial lookahead steps, $|A|$ trajectories each of length N, (where N denotes the maximum episode length before guaranteed termination) are rolled out, giving a complexity of $|A|N$. Then, the trajectory length is optimized against the cost-accuracy tradeoff (cf. Fig 2), leading to a complexity of $|A|n^*$. Finally, we further reduce the total lookahead ("planning") steps by a factor of m, since the cached policy is trusted for m steps, giving a complexity of $|A|n^* m^{-1}$. Note that in principle m grows with the size of the cache because there are a finite number of policies, and thus a larger unique cache must lead to distinct policies. This ultimately leads to an algorithm with a well-reduced scaling in environment steps.

Next, we derive an error bound on the difference between the online task's action-value function and those from the cache.

Theorem 1. *Suppose the online policy π (greedy with respect to Q_0 is used to collect trajectories of length n for estimating the return beginning from state s and action a. Let the cached value functions with Kendall tau distance be denoted by ε_c. Then the error in the value function can be bounded as*

$$\left|Q_0^\pi(s,a) - Q^c(s,a)\right| \leq \gamma^n \frac{R_{max}}{1-\gamma} + \varepsilon_c$$

(The proof is given in the Appendix.) Note that running the trajectories used for this estimation yield a cost linear in the number of timesteps, cn. Similar to classical literature (cf. discussion in (Dutta et al. 2018)), a cost-accuracy tradeoff is now presented, which can be used to determine the optimal trajectory length, n^*, defined below:

Proposition 1. *The cost of computing the proposed rollouts is proportional to the number of timesteps: cn.*

Here, we essentially assume the $|A|$ trajectories can be performed in parallel, in simulation. Then, with this cost addition, the previous bound can be optimized as a function of n:

Lemma 1. *Let $B = 2(1-\gamma)^{-1} R_{max}$, $g = \log \gamma^{-1}$ and c denote the per-step cost of the trajectory. For conditions satisfying Theorem 1, the trajectory length that minimizes the cost-accuracy tradeoff is:*

$$n^* = \left\lfloor g^{-1} \log \frac{Bg}{c} \right\rceil, \tag{4}$$

where $\lfloor \cdot \rceil$ denotes the nearest integer. this result follows immediately from Theorem 1 by adding the rollout cost cn and differentiating with respect to n. So long as $n \geq 1$, there is a possible advantage in using this approach. This can be ensured by satisfying the following condition: $B \leq \gamma^{-1} Cg$.

A natural question then follows: For how long can the policy π_{c^*} be "trusted" to yield similarly optimal trajectories? Note that when the error in using some "best" cached policy exceeds the next-best cached policy's error, the bound becomes void. This allows us to determine the number of steps for which the cached policy can be safely deployed, which we denote m. With some additional assumptions on the structure of the MDP, we can provide the following lemma, giving insight into this secondary rollout horizon.

Assumption 1 (Dynamical Locality)**.** There exists a constant $d > 0$ such that

$$|s - s'| < d \tag{5}$$

for all s' satisfying $p(s'|s,a) > 0$.

Lemma 2. *Let $Q(s,a)$ be Lipschitz continuous with constant L_Q. For all trajectories $\tau = (s_0, a_0, \ldots s_n, a_n)$, the difference in initial and terminal Q values is bounded by:*

$$\left|Q(s_0,a_0) - Q(s_n,a_n)\right| \leq L_Q \cdot d \cdot n. \tag{6}$$

The Lipschitz property of value functions is discussed in detail by (Rachelson and Lagoudakis 2010). The proof of this follows simply from the definitions, but is given in the Appendix for completeness.

Lemma 3. *If the Q functions each have Lipschitz constant upper bounded by L_Q, and Assumption 1 is satisfied, then*

the optimized cache policy c^* remains optimal for at least m steps, with

$$m = \frac{L_Q d}{2\left(\varepsilon_{c^*} - \varepsilon_{\bar{c}}\right)} \tag{7}$$

where $\bar{c}$ denotes the cached value function with next-smallest distance.

Proof. The optimized policy is only guaranteed to be optimal when its error does not exceed that of the second-best. Since (at worst) the best Q values can diverge at a rate $L_Q d$, and the second-best Q values can converge at a rate $L_Q d$, the gap in question $\varepsilon_{c^*} - \varepsilon_{\bar{c}}$ can be closed in no fewer than m steps. □

Conclusions

In this work, we introduced CASH, a new algorithm for reusing old solutions in an online model-based setting. We offer a new similarity metric which is used to compare rankings of simulated rollouts to action-rankings dictated by the cached policies. This notion of similarity is inspired by a relative ranking between actions, instead of focusing only on the greedy maximizing actions. We have established a relationship between cached solutions, value error bounds and compute cost, leading to an optimizable trajectory length. This method of generalization for planning appears novel, and has already led to theoretical statements revealing connections between core ideas across RL.

Recognizing the preliminary nature of this work, we are interested in continuing in several promising directions. For example, we aim to provide a practical implementation of the CASH algorithm to study it empirically, comparing its performance to the suggested theoretical results. We also intend to further extend the theoretical results by providing formal sample complexity analysis and relating to model-based techniques. We believe that this algorithm at the intersection of model-based RL and transfer learning can enhance training efficiency in complex problem settings.

References

Adamczyk, J.; Makarenko, V.; Arriojas, A.; Tiomkin, S.; and Kulkarni, R. V. 2023. Bounding the optimal value function in compositional reinforcement learning. In Evans, R. J.; and Shpitser, I., eds., *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, volume 216 of *Proceedings of Machine Learning Research*, 22–32. PMLR.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H. P.; and Silver, D. 2017. Successor features for transfer in reinforcement learning. *Advances in Neural Information Processing Systems*, 30.

Dutta, S.; Joshi, G.; Ghosh, S.; Dube, P.; and Nagpurkar, P. 2018. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. In *International conference on artificial intelligence and statistics*, 803–812. PMLR.

Haarnoja, T.; Pong, V.; Zhou, A.; Dalal, M.; Abbeel, P.; and Levine, S. 2018. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*, 6244–6251. IEEE.

Kendall, M. G. 1938. A New Measure of Rank Correlation. *Biometrika*, 30(1/2): 81–93.

Nemecek, M.; and Parr, R. 2021. Policy caches with successor features. In *International Conference on Machine Learning*, 8025–8033. PMLR.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, volume 99, 278–287.

Peng, X. B.; Chang, M.; Zhang, G.; Abbeel, P.; and Levine, S. 2019. Mcp: Learning composable hierarchical control with multiplicative compositional policies. *Advances in Neural Information Processing Systems*, 32.

Rachelson, E.; and Lagoudakis, M. G. 2010. On the Locality of Action Domination in Sequential Decision Making. In *11th International Symposium on Artificial Intelligence and Mathematics (ISIAM 2010)*, 1–8. Fort Lauderdale, US.

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

Skalse, J.; Farnik, L.; Motwani, S. R.; Jenner, E.; Gleave, A.; and Abate, A. 2023. STARC: A General Framework For Quantifying Differences Between Reward Functions. *arXiv preprint arXiv:2309.15257*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tasse, G. N.; James, S.; and Rosman, B. 2020. A Boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 9497–9507.

Taylor, M. E.; and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10(56): 1633–1685.

Wulfe, B.; Balakrishna, A.; Ellis, L.; Mercat, J.; McAllister, R.; and Gaidon, A. 2022. Dynamics-aware comparison of learned reward functions. *arXiv preprint arXiv:2201.10081*.

Proofs

Proof of Lemma 3

Proof. In the discrete action case, we ignore the action distances for simplicity.

$$\begin{aligned}
\big|Q(s_0,a_0) - Q(s_n,a_n)\big| &= \big|Q(s_0,a_0) - Q(s_1,a_1) + Q(s_1,a_1) + \cdots + Q(s_{n-1},a_{n-1}) - Q(s_{n-1},a_{n-1}) + Q(s_n,a_n)\big| \\
&= \big|\sum_{t=0}^{n-1} Q(s_t,a_t) - Q(s_{t+1},a_{t+1})\big| \\
&\leq \sum_{t=0}^{n-1} \big|Q(s_t,a_t) - Q(s_{t+1},a_{t+1})\big| \\
&\leq L_Q \sum_{t=0}^{n-1} \big|d(s_t,s_{t+1})\big| \\
&\leq L_Q dt
\end{aligned}$$

We have used the triangle inequality, followed by the Lipschitz condition and locality assumption at each timestep. □

For convenience we restate the result of Theorem 1 before its proof:

Theorem.

$$\big|Q_0^*(s,a) - Q_1^*(s,a)\big| \leq 2\gamma^n \frac{R_{\max}}{1-\gamma} + n(L_1+L_2)c + \varepsilon_1$$

Proof. First, we describe the separate error terms being incorporated in our bound. In estimating the return for a rollout of length n, this induces an error in the corresponding Q function of

$$|Q^\pi(s,a) - R_{1:N}| \leq \gamma^n \frac{R_{\max}}{1-\gamma} \tag{8}$$

$$\begin{aligned}
\big|Q_0^*(s,a) - Q_1^*(s,a)\big| &\leq \big|Q_0^*(s,a) - \hat{Q}_0^\pi(s,a) + \hat{Q}_0^\pi(s,a) - Q_1^*(s,a)\big| \\
&\leq \big|Q_0^*(s,a) - \hat{Q}_0^\pi(s,a)\big| + \big|\hat{Q}_0^\pi(s,a) - Q_1^*(s,a)\big| \\
&\leq \gamma^n \frac{R_{\max}}{1-\gamma} + \eta + \big|\hat{Q}_0^\pi(s,a) - Q_1^*(s,a)\big| \\
&\leq \gamma^n \frac{R_{\max}}{1-\gamma} + \eta + \varepsilon_1.
\end{aligned}$$

Since η is a global error in the current estimate of the Q function based on the number of optimization steps, rather than trajectory length or cache index, it can be ignored.

□