

Goal-Conditioned Q-Learning as Knowledge Distillation

Alexander Levine¹
Soheil Feizi¹

¹University of Maryland
College Park, Maryland, USA
alevine0@cs.umd.edu

Abstract

Many applications of reinforcement learning can be formalized as goal-conditioned environments, where, in each episode, there is a “goal” that affects the rewards obtained during that episode but does not affect the dynamics. Various techniques have been proposed to improve performance in goal-conditioned environments, such as automatic curriculum generation and goal relabeling. In this work, we explore a connection between off-policy reinforcement learning in goal-conditioned settings and knowledge distillation. In particular: the current Q-value function and the target Q-value estimate are both functions of the goal, and we would like to train the Q-value function to match its target *for all goals*. We therefore apply Gradient-Based Attention Transfer (Zagoruyko and Komodakis 2017), a knowledge distillation technique, to the Q-function update. We empirically show that this can improve the performance of goal-conditioned off-policy reinforcement learning when the space of goals is high-dimensional. We also show that this technique can be adapted to allow for efficient learning in the case of *multiple simultaneous sparse goals*, where the agent can attain a reward by achieving any one of a large set of objectives, all specified at test time. Finally, to provide theoretical support, we give examples of classes of environments where (under some assumptions) standard off-policy algorithms such as DDPG require at least $O(d^2)$ replay buffer transitions to learn an optimal policy, while our proposed technique requires only $O(d)$ transitions, where d is the dimensionality of the goal and state space. Code is available at <https://github.com/alevine0/ReenGAGE>.

Introduction

In recent years, many works have focused on applying deep reinforcement learning to goal-conditioned tasks, through approaches such as goal relabeling (Andrychowicz et al. 2017; Nair et al. 2018; Yang et al. 2021; Fang et al. 2019) and automatic curriculum generation (Florensa et al. 2018; Sukhbaatar et al. 2018; Zhang, Abbeel, and Pinto 2020). In this work, we focus on model-free off-policy goal-conditioned RL, and present a novel technique for improving performance in this setting. Our approach relies on a connection between the standard Bellman update used in off-policy reinforcement learning in a goal-conditioned setting, and *knowledge distillation*, the task of training a student network to model the same function as a (generally

more complex) teacher network.¹ In brief, the Bellman update can be viewed as an instance of (conditional, stochastic) knowledge distillation, where the current Q-value estimate is the student, the target Q-value network (averaged over transitions) is the teacher, and the independent variable is the *goal* that the agent is attempting to reach. We use this insight to develop an novel off-policy algorithm that in some instances has improved performance over baselines for goal-conditioned tasks. Our main contributions are as follows:

1. We propose **ReenGAGE**, a novel technique for goal-conditioned off-policy reinforcement learning, and evaluate its performance.
2. We propose **Multi-ReenGAGE**, a variant of ReenGAGE well-suited for goal-conditioned environments with *many simultaneous sparse goals*.
3. We provide theoretical justification for ReenGAGE by showing that it is in some cases asymptotically more efficient, in terms of the total number of replay buffer transitions required to learn an optimal policy, than standard off-policy algorithms such as DDPG.

In most of this work, we focus on continuous action control problems; we extend our method to discrete action spaces in the appendix. Note that while we mostly focus on using ReenGAGE on top of HER (Andrychowicz et al. 2017) and DDPG (Lillicrap et al. 2016) in this work, it can be easily applied alongside any goal-relabeling scheme or automated curriculum, and can be adapted for other off-policy algorithms such as SAC (Haarnoja et al. 2018) or TD3 (Fujimoto, Hoof, and Meger 2018). In particular, we include an application to SAC in the appendix.

Preliminaries and Notation

We consider control problems defined by goal-conditioned MDPs $(S, A, G, \mathcal{T}, R)$, where S , A , and G denote sets of states, actions, and goals, respectively, G and A are assumed to be continuous spaces, $\mathcal{T} \in S \times A \rightarrow \mathcal{P}(S)$ is a stochastic transition function, and $R \in S \times G \rightarrow \mathbb{R}$ is a reward function. At every step, starting at state $s \in S$, an agent

¹Some works use the term “knowledge distillation” to refer to the particular method for this task proposed by (Hinton, Vinyals, and Dean 2015), while others, such as (Gou et al. 2021) use it to refer to the task in general; we use the latter definition.

chooses an action $a \in A$. The system then transitions to $s' \sim \mathcal{T}(s, a)$, and the agent receives the reward $R(s', g)$.

For now, we assume that the reward function $R(s', g)$ is *known a priori* to the learning algorithm (while the transition function is not): this just means that we know how to interpret the objective which we are trying to achieve. Note that existing goal relabeling techniques, such as HER (Andrychowicz et al. 2017), implicitly make this assumption, as it is necessary to compute the rewards of relabeled transitions. We will discuss cases where this assumption can be relaxed in later sections.

We consider both *shaped* rewards, in which $R(s', g)$ is continuous and differentiable everywhere, as well as *sparse* rewards, where $R(s', g)$ is not assumed to be differentiable, but maintains some constant value c_{low} (e.g., $c_{\text{low}} = -1$ or 0) with $\nabla_g R(s', g) = \mathbf{0}$ for a substantial fraction of inputs. (There may be some boundary points where $R(s', g) = c_{\text{low}}$ but $\nabla_g R(s', g)$ is not defined or $\nabla_g R(s', g) \neq \mathbf{0}$, but in theoretical discussion we will assume these are of measure zero).

The objective of goal-conditioned RL is to find a policy $\pi \in S \times G \rightarrow A$ such that the discounted future reward:

$$r = \sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, g) \quad (1)$$

is maximized in expectation. One common approach is to find the policy π and Q-function $Q \in S \times A \times G \rightarrow \mathbb{R}$ that solve the Bellman equation for Q-learning (Watkins and Dayan 1992), conditioned on a goal g :

$$\forall s, a, g, \quad Q(s, a, g) = \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [R(s', g) + \gamma Q(s', \pi(s', g), g)] \quad (2)$$

$$\forall s, g, \quad \pi(s, g) = \arg \max_a Q(s, a, g). \quad (3)$$

If functions π and Q satisfy these, then π is guaranteed to be an optimal policy. In practice, off-policy RL techniques, notably DDPG (Lillicrap et al. 2016) can be used to solve for these functions iteratively by drawing tuples (s, a, s', g) from a replay buffer:

$$\mathcal{L}_{\text{critic}} = \mathbb{E}_{(s, a, s', g) \sim \text{Buffer}} \left[\mathcal{L}_{\text{mse}} \left[Q_{\theta}(s, a, g), \right. \right. \\ \left. \left. R(s', g) + \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \right] \right] \quad (4)$$

$$\mathcal{L}_{\text{actor}} = \mathbb{E}_{(s, g) \sim \text{Buffer}} -Q_{\theta}(s, \pi_{\phi}(s, g), g), \quad (5)$$

where θ and ϕ are the *current* critic and actor parameters, and θ' and ϕ' are *target* parameters, which are periodically updated to more closely match the current estimates. Note that Equation 2 should ideally hold for *all* (s, a, g) : therefore the distribution of (s, a, g) in the replay buffer does not need to precisely follow any particular distribution, assum-

ing sufficient visitation of possible tuples.² The only necessary constraint on the buffer distribution is that the marginal distribution of s' matches the transition function:

$$\forall s, a, g, \quad \Pr_{\text{Buffer}}[s' | s, a, g] \approx \Pr_{\mathcal{T}}[s' | s, a], \quad (6)$$

so that the relation in Equation 2 is respected. This means that the goal which is included in the buffer need not necessarily reflect a “true” historical experience of the agent during training, but can instead be relabeled to enhance training. (Andrychowicz et al. 2017; Nair et al. 2018; Yang et al. 2021; Fang et al. 2019). Interestingly, (Schroeder and Isbell 2021) shows that HER (Andrychowicz et al. 2017), a popular relabeling technique, actually *does not* respect Equation 6 when the transition function is nondeterministic, and therefore may exhibit “hindsight bias.”

Proposed Method

From Equation 2, we can take the gradient with respect to g :

$$\begin{aligned} \nabla_g Q(s, a, g) &= \\ \nabla_g \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [R(s', g) + \gamma Q(s', \pi(s', g), g)] &= \\ \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [\nabla_g R(s', g) + \gamma \nabla_g Q(s', \pi(s', g), g)] &. \end{aligned} \quad (7)$$

Because the gradient of the Q-value function is equal to the *expectation of the gradient* of the sum of the reward and the next-step Q-value, this suggests that we can augment the standard DDPG critic loss with a gradient term, which estimates this expected gradient using the replay buffer samples:

$$\begin{aligned} \mathcal{L}_{\text{ReenGAGE}} &= \mathbb{E}_{(s, a, s', g) \sim \text{Buffer}} \left[\mathcal{L}_{\text{mse}} \left[Q_{\theta}(s, a, g), \right. \right. \\ &\quad R(s', g) + \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \left. \right] \\ &\quad + \alpha \mathcal{L}_{\text{mse}} \left[\nabla_g Q_{\theta}(s, a, g), \right. \\ &\quad \left. \left. \nabla_g R(s', g) + \gamma \nabla_g Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \right] \right] \end{aligned} \quad (8)$$

where α is a constant hyperparameter. Note that the second MSE term is applied to a vector: thus we are fitting $\nabla_g Q_{\theta}(s_0, a_0, g)$ in all $\dim(g)$ dimensions. This allows more information to flow from the target function to the current Q-function (a $\dim(g)$ -vector instead of a scalar), and may therefore improve training. We call our method **Reinforcement learning with Gradient Attention for Goal-seeking Efficiently**, or **ReenGAGE**.

In the case of shaped rewards, we can use this loss function directly. In the case of sparse rewards, $\nabla_g R(s', g)$ is not necessarily defined or available. However, it is also zero for a substantial fraction of inputs, and, if $R(s', g) = c_{\text{low}}$, then

²In practice, replay buffers which better match the behavioral distribution result in better training, due to sources of “extrapolation error”, including incomplete visitation and model inductive bias; see (Fujimoto, Meger, and Precup 2019).

$\nabla_g R(s', g) = \mathbf{0}$ with high probability. Therefore, we use the gradient loss term only when training on tuples where $R(s', g) = c_{\text{low}}$, and assume $\nabla_g R(s', g) = \mathbf{0}$:

$$\begin{aligned} \mathcal{L}_{\text{ReenGAGE}}^{(\text{sparse})} = & \mathbb{E}_{(s, a, s')} \left[\mathcal{L}_{\text{mse}} \left[Q_{\theta}(s, a, g), \right. \right. \\ & R(s', g) + \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \left. \right] \\ & + \mathbf{1}_{R(s', g) = c_{\text{low}}} \alpha \mathcal{L}_{\text{mse}} \left[\nabla_g Q_{\theta}(s, a, g), \right. \\ & \left. \left. \gamma \nabla_g Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \right] \right]. \end{aligned} \quad (9)$$

In this sparse case, if ReenGAGE is used alone (i.e., without goal relabeling), then the reward function R does not need to be known explicitly *a priori*. Instead, the observed values of the rewards $R(s', g)$ from the training rollouts may be used.

Note that ReenGAGE can only be used in goal-conditioned reinforcement learning problems: in particular, we cannot use gradients with respect to states or actions in a way similar to Equation 7, because, unlike in Equation 7:

$$\nabla_{s, a} \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [\langle \cdot \rangle] \neq \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [\nabla_{s, a} \langle \cdot \rangle] \quad (10)$$

because the sampling distribution depends on s and a .

Connection to Knowledge Distillation

We can view Equation 4 as the loss function of a regression problem, fitting Q_{θ} to the target. We treat g as the independent variable, and s and a as parameters:

$$\begin{aligned} \forall g, \quad Q_{\theta}(g; s, a) &:= \text{Targ.}(g; s, a) \text{ where} \\ \text{Targ.}(g; s, a) &= \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [R(s', g) + \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g)] \end{aligned} \quad (11)$$

This can be framed as a knowledge distillation problem: we can view $\text{Targ.}(g; s, a)$ as a (difficult to compute) *teacher* function, and we are trying to fit the network $Q_{\theta}(g; s, a)$ to represent the same function of g . Note however that conventional “knowledge distillation” (Hinton, Vinyals, and Dean 2015), which matches the logits of one network to another for classification problems in order to provide richer supervision than simply matching the class label output, cannot be applied here because the output is scalar. However, (Zagoruyko and Komodakis 2017) proposes *Gradient-based Attention Transfer* which instead matches the *gradients* of the student to the teacher using a regularization term. Applied to our Q function and target, this is:

$$\begin{aligned} \mathcal{L}_{\text{GAT}} = & \mathcal{L}_{\text{MSE}}(Q_{\theta}(g; s, a), \text{Targ.}(g; s, a)) \\ & + \alpha \|\nabla_g Q_{\theta}(g; s, a) - \nabla_g \text{Targ.}(g; s, a)\|_2^2, \end{aligned} \quad (12)$$

which is in fact the ReenGAGE loss function. Therefore we can think of ReenGAGE as applying knowledge distillation (specifically Gradient-based Attention Transfer) to the Q-value update.³ See Figure 1 for an illustration. While the

³(Zagoruyko and Komodakis 2017) actually uses the ℓ_2 distance between the gradients as the regularization term, rather than

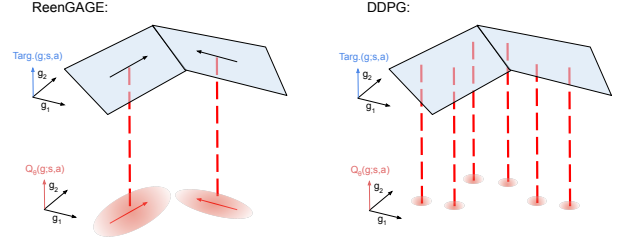


Figure 1: Illustration comparing the information flow from the Q-value target function to the current Q-function in ReenGAGE, compared to standard DDPG. In ReenGAGE, for each (s, a, s', g) tuple, the gradient with respect to the goal is used as supervision, while in standard DDPG, only point values are used. Note that in stochastic environments, each tuple only provides a stochastic estimate of the target gradient (in ReenGAGE) or target point value (in DDPG).

computation of the loss function gradient is somewhat more complex here than in standard training, involving mixed partial derivatives, (Zagoruyko and Komodakis 2017) notes that it can still be performed efficiently using modern automatic differentiation packages; in fact, this “double backpropagation” should only scale the computation time by a constant factor (Etmann 2019). Further discussion of this and empirical runtime comparisons are provided in the appendix.

(Zagoruyko and Komodakis 2017) also propose Activation-based Attention Transfer, which transfers intermediate layer activations from teacher to student network rather than gradients; in fact, they report better performance using this method than the gradient method. However, this is not applicable in our case. Firstly, in the dense reward case, we cannot model the reward function in this way. Secondly, unlike the gradient operator, activations are nonlinear: so, even in the sparse case, we cannot assume that the activations of a “converged” Q-network perfectly modeling the expected target will be the equal to the expected activations of the target network (i.e., there is no activation equivalent to Equation 7.) See (Hsu et al. 2022) for a review of sources of auxiliary network information that can be used for knowledge distillation.

Toy Example Experiments

We first apply ReenGAGE to a simple sparse-reward environment, which we call **ContinuousSeek**. This task is a continuous variant of the discrete “Bit-Flipping” environment proposed in (Andrychowicz et al. 2017). In our proposed task, the objective is to navigate from an initial state in d -dimensional space to a desired goal state, by, at each step, adding an ℓ_{∞} -bounded vector to the current state. Formally:

its square. However, because our gradient estimate is stochastic (in particular, we are using samples of $s' \sim \mathcal{T}(s, a)$ rather than the expectation), we instead use the mean squared error, so that the current Q gradients will converge to the population mean. (Zagoruyko and Komodakis 2017) notes that their particular choice of the ℓ_2 norm is arbitrary: other metrics should work.

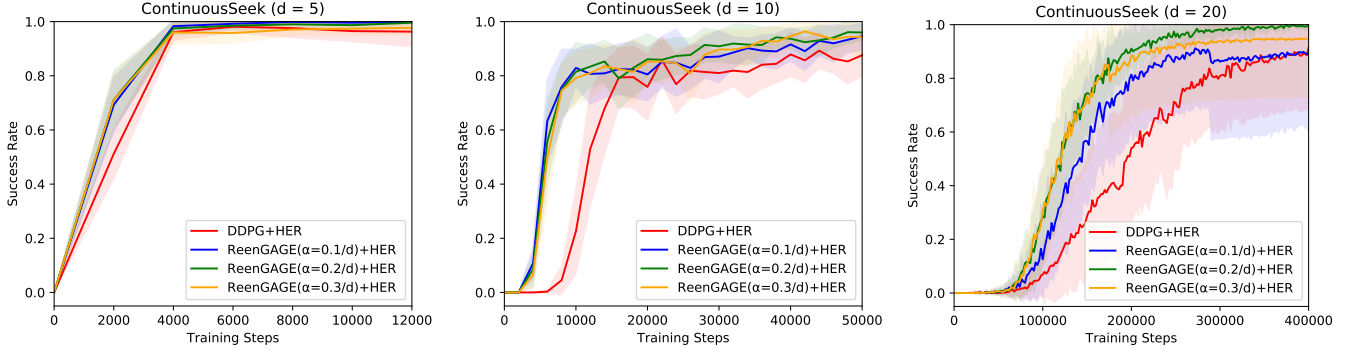


Figure 2: ContinuousSeek results. Lines show the mean and standard deviation over 20 random seeds (kept the same for all experiments.) The Y-axis represents the success rate, defined as the fraction of test episodes for which the goal is ever reached.

- $s, g \in [-D, D]^d$
- $a \in [-1, 1]^d$
- $\mathcal{T}(s, a) = s + a$ (clipped into $[-D, D]^d$)
- $R(s, g) = -1 + \mathbf{1}_{\|s-g\|_\infty \leq \epsilon}$
- initial state $s_0 = \mathbf{0}$.

In our experiments, we use $D = 5$, $\epsilon = 0.1$, and we run for 10 steps per episode. We test with $d = 5, 10$ and 20 . The chance that a random state achieves the goal is approximately $\frac{1}{50^d}$, so this is an extremely sparse reward problem (as sparse as “Bit-Flipping” with $5.6 \times d$ bits). As a baseline, we use DDPG with HER.

See Figure 2 for results. For the baseline and each value of α , we performed a grid search over learning rates $\{0.00025, 0.0005, 0.001, 0.0015\}$ and batch sizes $\{128, 256, 512\}$; the curves shown represent the “best” hyperparameter settings for each α , defined as maximizing the area under the curves. See appendix for results for all hyperparameter settings. We studied the learning rate specifically to ensure that the ReenGAGE regularization term is not simply “scaling up” the loss function with similar gradient updates. Other hyperparameters were kept fixed and are listed in the appendix. We see that ReenGAGE clearly improves over the baseline for larger-dimensionality goals ($d = 10$ and $d = 20$): this shows that ReenGAGE can improve the performance of DDPG in such high-dimensional goal settings. See the appendix for a similar experiment with SAC as the base off-policy learning algorithm instead of DDPG.

Robotics Experiments

We tested our method on **HandReach**, the environment from the OpenAI Gym Robotics suite (Plappert et al. 2018) with the highest-dimensional goal space ($d = 15$). In this sparse-reward environment, the agent controls a simulated robotic hand with 20-dimensional actions controlling the hand’s joints; the goal is to move all of the fingertips to the specified 3-dimensional positions. As a baseline, we use the released DDPG+HER code from (Plappert et al. 2018), with all hyperparameters as originally presented, and only modify the critic loss term. Results are presented in Figure 3. In this environment, we see that ReenGAGE greatly speeds up

convergence compared to the baseline. However, at a high value of α , the success rate declines after first converging. This shows that ReenGAGE may cause some instability if the gradient loss term is too large, and that tuning the coefficient α is necessary (see also the Limitations section below).

We also tried our method on the **HandManipulateBlock** environment from the same paper; however, in this lower-dimensional goal environment ($d = 7$) ReenGAGE was not shown to improve performance. This is compatible with our observation from the ContinuousSeek environment that ReenGAGE leads to greater improvements for higher-dimensional tasks, as the dimensionality of the additional goal-gradient information that ReenGAGE propagates increases. Results are provided in the appendix.

Multi-ReenGAGE: ReenGAGE for Multiple Simultaneous Goals

In this section, we propose a variant of ReenGAGE for a specific class of RL environments: environments where the agent is rewarded for achieving any goal in a large set of arbitrary sparse goals, all of which are specified at test time. Formally, we consider goals in the form $g = \{g_1, \dots, g_n\}$, where n may vary but $n \leq n_{\max}$. We consider $\{0, 1\}$ rewards, where the reward function takes the form:

$$R(s', g) = \begin{cases} 1 & \text{if } \bigvee_{g_i \in g} (R_{\text{item}}(s', g_i) = 1) \\ 0 & \text{otherwise} \end{cases}. \quad (13)$$

In our experiments, we only consider cases where the goals are mutually exclusive, so this is equivalent to:

$$R(s', g) = \sum_{g_i \in g} R_{\text{item}}(s', g_i). \quad (14)$$

We assume that either: (i) the function R_{item} is known *a priori* to the agent, or (ii) the item rewards $R_{\text{item}}(s', g_i)$ are observed separately for each goal g_i at each time step during training. This scenario presents several challenges. Firstly, many goal relabeling strategies cannot be directly applied here: strategies such as HER (Andrychowicz et al. 2017) assume that achieved states can be *projected down* into the space of goals. In this case, the space of goals is *much larger*

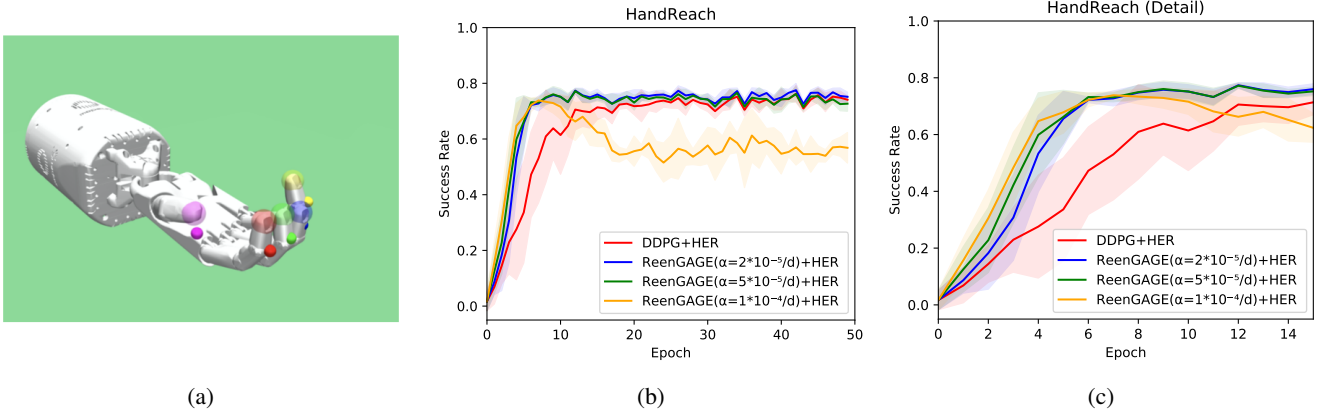


Figure 3: HandReach results. (a) Rendering of the HandReach simulation environment. Figure taken from (Plappert et al. 2018). (b) Performance of ReenGAGE on HandReach, compared to the baseline from (Plappert et al. 2018). Lines represent mean and standard deviation for the same set of 5 seeds. The X-axis is the number of training epochs, as defined in (Plappert et al. 2018), while the Y-axis is the success rate, defined by (Plappert et al. 2018) as the fraction of test episodes where the *final* state satisfies the goal. (c) Detailed view of (b), showing the epochs before convergence, where the advantage of ReenGAGE is most clear.

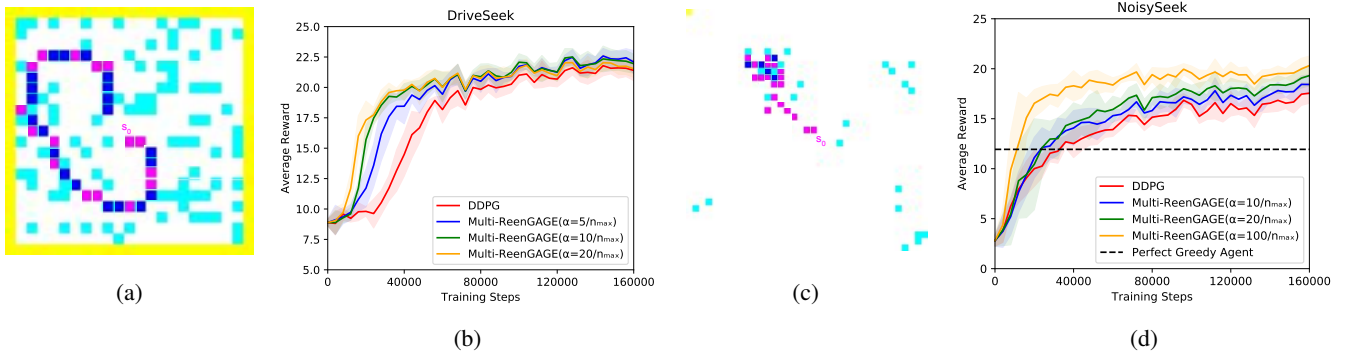


Figure 4: Multi-ReenGAGE results. (a) and (c): Illustrations of DriveSeek and NoisySeek environments, respectively: cyan points show goals that were never reached, blue points show goals that were reached, and magenta points show (rounded) non-goal states that were reached. In (c), we see a NoisyReach agent (correctly) avoiding the trap of going to the nearby isolated points in favor of seeking the larger cluster. (b) and (d): Results for DriveSeek and NoisySeek, respectively. We see that Multi-ReenGAGE substantially improves over standard DDPG for both tasks. Lines are an average of (the same) 5 random seeds. For NoisySeek, we also show the performance of a perfect “greedy” agent, which simply goes towards the nearest individual goal. For NoisySeek, evaluations with more values of α are included in the appendix. Note that for both experiments, the agent takes as input a list of goal coordinates, rather than an image: the agents do not use convolutional layers to interpret the goals. (On DriveSeek, where the coordinates are bounded, we attempted learning from images as well; ReenGAGE still outperformed DDPG, but overall performance was worse for both – this experiment is presented in the appendix.)

than the space of possible states, so this assumption is broken. Secondly, we suggest that standard Q-learning is somewhat unsuited to this kind of problem, because it loses information about *which* goal led to a reward. For instance, if there are 100 goals g_i , and a reward is received for a certain state s' , there is no direct indication of which goal was satisfied. This means that a very large number of episodes may need to be run in order to learn the effect of *each individual* goal on the reward.

We now describe our approach. For concreteness, we will assume that the agent uses an architecture based on DeepSets (Zaheer et al. 2017) to process the goal set input

(although we believe our technique can likely be adapted to using more complex neural set architectures, such as Set Transformer (Lee et al. 2019)). Concretely, this means that our Q-function takes the form:

$$Q_{\theta}(s, a, g) := Q_{\theta^h}^{\text{head}}(s, a, \sum_{g_i \in g} [Q_{\theta^c}^{\text{encoder}}(s, g_i)]) \quad (15)$$

and the policy has a similar architecture. Note that $Q_{\theta^c}^{\text{encoder}}$ outputs a vector-valued embedding for a given goal g_i . From this baseline, introduce a set of scalar *gate variables* b_i :

$$Q_{\theta}(s, a, g) := Q_{\theta^h}^{\text{head}}(s, a, \sum_{i=1}^n [b_i Q_{\theta^c}^{\text{encoder}}(s, g_i)]). \quad (16)$$

Each gate b_i is set to 1. However, if b_i were zero, this would be equivalent to the goal g_i being absent from the set g . We then treat the gate variables as *differentiable*. If a certain goal g_i contributes to the Q function (i.e., if it is likely to be satisfied), then we expect $Q_\theta(s, a, g)$ to be *highly sensitive* to b_i ; in other words, we expect $\frac{\partial Q_\theta}{\partial b_i}$ to be large. Then $\nabla_b Q$ represents the importance of each goal to the Q-value function. Our key insight is that we can use a ReenGAGE-style loss to transfer $\nabla_b Q$ from target to current Q-value estimate, therefore preserving attention on the relevant goals.

However, this requires us to have a value for $\nabla_b R(s', g)$. Note that the reward can be written as:

$$R(s', g) = \sum_{g_i \in g} b_i R_{\text{item}}(s', g_i). \quad (17)$$

Setting $b_i = 0$ is again like g_i being absent. Interpolating:

$$\frac{\partial R}{\partial b_i} := R_{\text{item}}(s', g_i) \quad (18)$$

which gives us a “ground-truth” reward gradient we can compute. This yields the following loss function:

$$\begin{aligned} \mathcal{L}_{\text{Multi-ReenGAGE}} = \mathcal{L}_{\text{DDPG-Critic}} + \alpha \mathcal{L}_{\text{mse}} & \left[\nabla_b Q_\theta(s, a, g), \right. \\ & \left. R_{\text{item}}(s', g) + \gamma \nabla_b Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \right] \end{aligned} \quad (19)$$

where $[R_{\text{item}}(s', g)]_i := R_{\text{item}}(s', g_i)$. In practice, we make two modifications to this algorithm. First, we use b_i^2 as the gate rather than b_i .⁴ While algebraically this should do nothing but multiply the gradient loss term by 4, it is important for vectorized implementation; see the appendix for details.

Second, we share the encoder Q^{encoder} between the Q-function and the policy π . This is so the policy does not have to learn to interpret the goal set “from scratch” and is empirically important (see ablation study in the appendix). We train the encoder only during critic training.

Experiments

We test Multi-ReenGAGE on two environments: **DriveSeek** and **NoisySeek**. Both environments are constructed such that a “greedy” strategy of simply going to the closest individual goal is not optimal, so the entire goal set must be considered. We describe the environments informally here and provide additional detail in the appendix.

DriveSeek is a deterministic environment, where the continuous position $s_{\text{pos.}} \in [-10, 10]^2$ always moves with constant ℓ_2 speed 1, in a direction determined by a velocity vector $s_{\text{vel.}}$ on the unit circle. At each step, the agent takes a 1-dimensional action $a \in [-0.5, 0.5]$, which represents *angular acceleration*: it specifies an angle in radians which is added to the angle of the velocity vector. At the edges of the space, the state position wraps around to the opposite edge.

The objective is to reach any of up to $n_{\text{max}} = 200$ goals. The goals all lie on integer coordinates in $[-10, 10]^2$, and the agent receives a reward if its coordinates round to a goal.

⁴And use $2R_{\text{item}}(s', g)$ instead of $R_{\text{item}}(s', g)$.

In addition to $s_{\text{pos.}}$ and $s_{\text{vel.}}$, the agent receives an observation of its current rounded position. The agent also receives as input the current list of goal coordinates. Note that the agent cannot simply stay at a single goal, or take an arbitrary path between goals: it is constrained to making wide turns. Therefore all goals must be considered in planning an optimal trajectory.

NoisySeek is a randomized environment. In it, $s \in \mathbb{R}^2$, $a \in \mathbb{R}^2$, with $\|a\|_2 \leq 1$, and the transition function is defined as $\mathcal{T}(s, a) \sim \mathcal{N}(s + a, I)$. In other words, the agent moves through space at a capped speed, and noise is constantly added to the position. The goals are defined as integer coordinates in a similar manner to in DriveSeek, but without a box constraint. Additionally, the goal distribution is such that goals tend to be clustered together. Note that a “greedy” agent that simply goes to the nearest goal is suboptimal, because the probability of consistently reaching that one goal is low: it is better to seek clusters.

Results are presented in Figure 4. We see that Multi-ReenGAGE substantially outperforms the baseline of DDPG on both environments.

Theoretical Properties

Bias

In the Preliminaries section, we discuss that goal relabeling strategies can exhibit bias if Equation 6 is not respected. In the dense reward case, our method does not cause bias of this sort (although such bias may be present if our method is combined with a relabeling strategy.) However, in the sparse reward case, if the transitions are nondeterministic, our method may cause a similar bias. In particular, note that, in the sparse case, Equation 9 effectively trains the gradient of the Q-value function to match the following target:

$$\begin{aligned} \nabla_g Q_\theta(s, a, g) &:= \\ \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [\gamma \nabla_g Q_{\theta'}(s', \pi_{\phi'}(s', g), g) | R(s', g) = c_{\text{low}}] &\approx \\ \nabla_g \mathbb{E}_{s' \sim \mathcal{T}(s, a)} [R(s', g) + & \\ \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g) | R(s', g) = c_{\text{low}}] & \end{aligned} \quad (20)$$

where the last line holds exactly if the “boundary” points where $R(s', g) = c_{\text{low}}$ but $\nabla_g R(s', g) \neq \mathbf{0}$ are of measure zero (and the derivative is defined at all such points).

Equation 20 shows us that, in the sparse case, our method trains the gradient of the Q-function to match an expected target gradient where the expectation is taken over a biased distribution: if

$$\Pr_{\mathcal{T}}[s' | s, a; R(s', g) = c_{\text{low}}] \neq \Pr_{\mathcal{T}}[s' | s, a], \quad (21)$$

then this will cause bias in our target gradient estimate, in a similar manner to the hindsight bias of HER described by (Schroecker and Isbell 2021).

Note that this is only an issue in nondeterministic environments: in deterministic environments, for a given (s, a, g) , either $R(s', g)$ is always $\neq c_{\text{low}}$, in which case the gradient

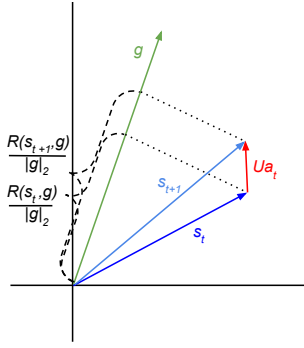


Figure 5: Example dense reward environment class. The agent receives a reward proportional to the projection of the state vector onto the goal vector at each step, and can change the state vector by adding an action vector with ℓ_2 distance up to 1 at each step. However, at each step, this action is distorted by an unknown rotation U : the agent must learn to compensate for this distortion.

term is never involved in training, or $R(s', g)$ is always c_{low} , in which case

$$\Pr_{\mathcal{T}}[s'|s, a; R(s', g) = c_{\text{low}}] = \Pr_{\mathcal{T}}[s'|s, a] = 1. \quad (22)$$

Learning Efficiency

In this section, we provide examples of classes of environments for which our method will result in provably more efficient learning than standard DDPG-style updates. We treat both the dense reward case (in which we have access to the gradient of the reward function) and the sparse reward case (in which we do not).

Dense Reward Example Consider the class of simple, deterministic environments described as follows:

- $g, s, a \in \mathbb{R}^d$; $\|a\|_2 \leq 1$
- $R(s, g) = g^T s$
- $\mathcal{T}(s, a) = s + Ua$, where $U \in \mathbb{R}^{d \times d}$ is an unknown orthogonal (rotation) matrix.

This environment class is illustrated in Figure 5. Environments of this class are parameterized by U , so the learning task is to estimate U . We make the following assumptions:

- The “hypothesis class” consists of all environments with dynamics of the type described above. We therefore take as an inductive bias that each model in the considered model class consists of a Q -function $Q_{\tilde{U}}$ and policy $\pi_{\tilde{U}}$ which are in the form of the optimal Q -function and policy for an estimate of U , notated as $\tilde{U} \in \mathbb{R}^{d \times d}$ (constrained to be orthogonal).
- We assume that $Q_{\tilde{U}}$ and $\pi_{\tilde{U}}$ share the same estimated parameter $\tilde{U}$. (This is analogous to – although admittedly stronger than – the parameter sharing we used for Multi-ReenGAGE.) Taken with the above assumption, this implies that $a = \pi_{\tilde{U}}(s, g)$ maximizes $Q_{\tilde{U}}(s, a, g)$, so we do not need to train π separately. Similar parameter sharing occurs between the target policy and Q -function.

- We are comparing our method, minimizing the loss in Equation 8, with minimizing the “vanilla” DDPG loss (Equation 4).
- States and actions in the replay buffer are in general position.

In this case, the following proposition holds:

Proposition 1. *Under the above assumptions, minimizing the ReenGAGE loss can learn U (and therefore learn the optimal policy) using $O(d)$ unique replay buffer transitions. However, minimizing the standard DDPG loss requires at least $O(d^2)$ unique transitions to successfully learn U .*

Proofs are provided in the appendix. This result shows that, in some cases, ReenGAGE requires asymptotically less replay data to successfully learn to perform a task than standard DDPG.

Sparse Reward Case The result shown above might be unsurprising to many readers. Specifically, because the gradient $\nabla_g R(s', g)$ is used by our method and not by standard DDPG, in the dense-reward case, our method is utilizing more information from the environment (to the extent that R , which we assume that agents know *a priori*, is part of the “environment”) than the standard algorithm. However, here we show a class of *sparse reward* environments for which the same result holds, despite $\nabla_g R(s', g)$ being unavailable. The environments are constructed as follows:

- $g, a \in \mathbb{R}^d$; $\|a\|_2 \leq 1$; $s \in \mathbb{R}^{2d}$; the state vector consists of two halves, denoted s^1, s^2 ; we write s as $(s^1; s^2)$.
- $R(s, g) = g^T s^1$
- $\mathcal{T}(s, a) = \begin{cases} (0; s^1 + Ua) & \text{if } s^1 \neq 0 \\ (s^2; 0) & \text{if } s^1 = 0 \end{cases}$
- $U \in \mathbb{R}^{d \times d}$ is an unknown orthogonal (rotation) matrix.
- We define $c_{\text{low}} = 0$.

See Figure 6 for an illustration. Note that this satisfies sparseness properties: namely, $R(s', g) = 0 = c_{\text{low}}$ at least every other step; and, when $R(s', g) = 0$, then $\nabla_g R(s', g) = 0$ (assuming general position). It is also deterministic, so we do not need to worry about the bias discussed in the previous section. We can therefore apply the sparse version of our method (Equation 9), which does *not* use gradient feedback from the reward:

Proposition 2. *Under the same assumptions as Proposition 1 (replacing Equation 8 with Equation 9), minimizing the ReenGAGE loss can learn U in the sparse environment class using $O(d)$ unique replay buffer transitions. However, minimizing the standard DDPG loss requires at least $O(d^2)$ unique transitions to successfully learn U .*

This example is admittedly a bit contrived: the single-step reward can always be computed without knowledge of the parameter U . However, it may still give insight about real-world scenarios in which predicting immediate reward is much easier than understanding long-term dynamics.

Note that these two scaling results apply to the number of *replay buffer transitions*. In particular, if a goal relabeling algorithm is used on top of DDPG, then $O(d^2)$ replay buffer

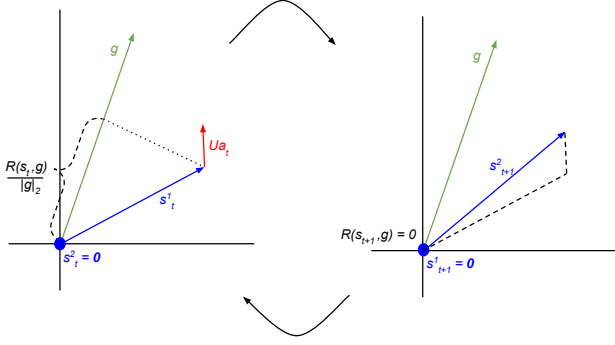


Figure 6: Example sparse reward environment class. If s^1 is initially nonzero, then the (rotated) action Ua is added to it, as in the dense case. However, the resulting vector is immediately “stored” in s^2 , and s^1 is zeroed: this means that no immediate reward is obtained. In the next step, with s^1 zero, the action is ignored and s^2 is “reloaded” into s^1 , resulting in a reward that depends on the *previous* action.

transitions may be able to be constructed from $O(d)$ observed training rollout transitions, so standard DDPG *combined with goal relabeling* might only require $O(d)$ training rollout transitions. However, this would be computationally expensive, and may not work in practice for particular goal relabeling algorithms. (HER, for instance, only relabels using achieved states from the same episode: if the episode length is $O(1)$ in d , then $O(d^2)$ observed training rollout transitions would still be required for DDPG+HER.) Also, goal relabeling techniques require *a priori* knowledge of the function R , while in the sparse example, ReenGAGE does not (although in the case of this example, we assume that we are using the correct “hypothesis class”, i.e., the functional form of $Q_{\bar{U}}$: constructing this in practice would likely require knowing R).

Related Works

Many prior approaches have been taken to the goal-conditioned reinforcement learning problem (Schaal et al. 2015). See (Liu, Zhu, and Zhang 2022) for a recent survey of this area. One line of work for this problem involves *automated curriculum generation*: here, the idea is to select goals during training that are dynamically chosen to be the most informative (Florensa et al. 2018; Sukhbaatar et al. 2018; Zhang, Abbeel, and Pinto 2020). In the off-policy reinforcement learning setting, a related technique becomes a possibility: one can re-label past experiences with counter-factual goals. This allows a single experienced transition to be used to train for multiple goals, and the re-labeled goals can be chosen using various heuristics to improve training (Andrychowicz et al. 2017; Nair et al. 2018; Yang et al. 2021; Fang et al. 2019). Note that our proposed method can be combined with any of these off-policy techniques. (Schroecker and Isbell 2021) discusses bias that can result from some goal relabeling techniques. (Eysenbach, Salakhutdinov, and Levine 2021) proposes a method based

on recursive classification which is in practice similar to hindsight relabeling, but requires less parameter tuning.

In alternative approaches to goal-conditioned RL, (Eysenbach et al. 2022) has proposed using an on-policy goal-conditioned reinforcement learning technique, using contrastive learning, while (Janner et al. 2022) and (Janner, Li, and Levine 2021) propose model-based techniques.

Note that our proposed method is distinct from *policy distillation* (Rusu et al. 2016): the goal of policy distillation is to consolidate one or more *already trained* policy networks into a smaller network; whereas our method is intended to improve initial training. Some prior (Manchin, Abbasnejad, and Hengel 2019; Choi, Lee, and Zhang 2017; Wu, Khetarpal, and Precup 2021) and concurrent (Bertoin et al. 2022) works have focused on using attention-based mechanisms to improve either the performance or interpretability of reinforcement learning algorithms. However, to our knowledge, ours is the first to apply gradient-based attention transfer to the critic update to enhance goal-conditioned off-policy reinforcement learning.

Some prior works have been proposed for goal-seeking with structured, complex goals made up of sub-goals, similar to (and in some cases more general than) the multi-goal setting that Multi-ReenGAGE is designed for. Some of these works (Oh et al. 2017; Sohn, Oh, and Lee 2018) use a hierarchical policy; however, such a structure may be unable to represent the true optimal policy (Vaezipoor et al. 2021). (Vaezipoor et al. 2021) proposes a method without this limitation; although the setting considered (Linear Temporal Logic) is different from the multi-goal setting considered here, in that a reward is achieved at most once per episode. (Touati and Ollivier 2021) proposes a method for arbitrary reward functions specified at test time, under discrete action spaces; in concurrent work (Touati, Rapin, and Ollivier 2023), this is generalized to continuous actions. (Janner et al. 2022), a model-based technique mentioned above, can use reward function gradient information to adapt to an arbitrary *shaped* (i.e., non-sparse) reward function at test time.

Limitations and Conclusion

ReenGAGE has some important limitations. For example, we have seen that the hyper-parameter α requires tuning and can vary greatly (likely due to diverse scales in goal coordinates and rewards); and the benefits of ReenGAGE seem limited to tasks with high goal dimension.

Still, ReenGAGE represents a novel approach to goal-conditioned RL, with benefits demonstrated both empirically and theoretically. In future work, we are particularly interested in exploring the use of Multi-ReenGAGE in safety and robustness applications. In particular, the ability to encode many simultaneous goals at test time could allow the agent to consider many “backup” goals, all of which are acceptable, rather than forcing the agent to focus only a single goal (resulting in total failure if that goal is unreachable.)

Acknowledgements

This project was supported in part by NSF CAREER AWARD 1942230 and ONR YIP award N00014-22-1-2271.

References

- Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.
- Bertoin, D.; Zouitine, A.; Zouitine, M.; and Rachelson, E. 2022. Look where you look! Saliency-guided Q-networks for generalization in visual Reinforcement Learning. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *Advances in Neural Information Processing Systems*.
- Choi, J.; Lee, B.-J.; and Zhang, B.-T. 2017. Multi-Focus Attention Network for Efficient Deep Reinforcement Learning. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.
- Etmann, C. 2019. A Closer Look at Double Backpropagation. *ArXiv*, abs/1906.06637.
- Eysenbach, B.; Salakhutdinov, R.; and Levine, S. 2021. C-Learning: Learning to Achieve Goals via Recursive Classification. In *International Conference on Learning Representations*.
- Eysenbach, B.; Zhang, T.; Salakhutdinov, R.; and Levine, S. 2022. Contrastive Learning as Goal-Conditioned Reinforcement Learning. *arXiv preprint arXiv:2206.07568*.
- Fang, M.; Zhou, T.; Du, Y.; Han, L.; and Zhang, Z. 2019. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, 32.
- Florensa, C.; Held, D.; Geng, X.; and Abbeel, P. 2018. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, 1515–1528. PMLR.
- Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, 1587–1596. PMLR.
- Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, 2052–2062. PMLR.
- Gou, J.; Yu, B.; Maybank, S. J.; and Tao, D. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6): 1789–1819.
- Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.
- Hinton, G. E.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. *ArXiv*, abs/1503.02531.
- Hsu, Y.-C.; Smith, J.; Shen, Y.; Kira, Z.; and Jin, H. 2022. A Closer Look at Knowledge Distillation with Features, Logs, and Gradients. *arXiv preprint arXiv:2203.10163*.
- Janner, M.; Du, Y.; Tenenbaum, J. B.; and Levine, S. 2022. Planning with Diffusion for Flexible Behavior Synthesis. *arXiv preprint arXiv:2205.09991*.
- Janner, M.; Li, Q.; and Levine, S. 2021. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*.
- Lee, J.; Lee, Y.; Kim, J.; Kosior, A.; Choi, S.; and Teh, Y. W. 2019. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 3744–3753. PMLR.
- Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.
- Liu, M.; Zhu, M.; and Zhang, W. 2022. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 5502–5511. International Joint Conferences on Artificial Intelligence Organization. Survey Track.
- Manchin, A.; Abbasnejad, E.; and Hengel, A. v. d. 2019. Reinforcement learning with attention that works: A self-supervised approach. In *International conference on neural information processing*, 223–230. Springer.
- Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Hiedmiller, M.; Fiedland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.
- Nair, A. V.; Pong, V.; Dalal, M.; Bahl, S.; Lin, S.; and Levine, S. 2018. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31.
- Oh, J.; Singh, S.; Lee, H.; and Kohli, P. 2017. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, 2661–2670. PMLR.
- Plappert, M.; Andrychowicz, M.; Ray, A.; McGrew, B.; Baker, B.; Powell, G.; Schneider, J.; Tobin, J.; Chociej, M.; Welinder, P.; Kumar, V.; and Zaremba, W. 2018. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research.
- Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.
- Rusu, A. A.; Colmenarejo, S. G.; Gülçehre, Ç.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2016. Policy Distillation. In *ICLR (Poster)*.
- Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In Bach, F.; and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1312–1320. Lille, France: PMLR.
- Schroecker, Y.; and Isbell, C. L. 2021. Universal Value Density Estimation for Imitation Learning and Goal-Conditioned Reinforcement Learning.

- Sohn, S.; Oh, J.; and Lee, H. 2018. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in neural information processing systems*, 31.
- Sukhbaatar, S.; Lin, Z.; Kostrikov, I.; Synnaeve, G.; Szlam, A.; and Fergus, R. 2018. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *International Conference on Learning Representations*.
- Touati, A.; and Ollivier, Y. 2021. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34: 13–23.
- Touati, A.; Rapin, J.; and Ollivier, Y. 2023. Does Zero-Shot Reinforcement Learning Exist? In *The Eleventh International Conference on Learning Representations*.
- Vaezipoor, P.; Li, A. C.; Icarte, R. A. T.; and McIlraith, S. A. 2021. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, 10497–10508. PMLR.
- Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8(3): 279–292.
- Wu, H.; Khetarpal, K.; and Precup, D. 2021. Self-Supervised Attention-Aware Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12): 10311–10319.
- Yang, R.; Fang, M.; Han, L.; Du, Y.; Luo, F.; and Li, X. 2021. MHER: Model-based Hindsight Experience Replay. In *Deep RL Workshop NeurIPS 2021*.
- Zagoruyko, S.; and Komodakis, N. 2017. Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer. In *International Conference on Learning Representations*.
- Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R. R.; and Smola, A. J. 2017. Deep Sets. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Zhang, Y.; Abbeel, P.; and Pinto, L. 2020. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33: 7648–7659.

Proofs

Proposition 1 (Gradient reward feedback example)

Problem setting

- $g, s, a \in \mathbb{R}^n$; $\|a\|_2 \leq 1$
- Single step reward $R(s, g) = g^T s$
- Transition function: $s_{t+1} := s_t + U a_t$, where $U \in \mathbb{R}^{n \times n}$ is an unknown orthogonal (rotation) matrix
- The “hypothesis class” consists of all environments with dynamics of this form: the learning task is to learn the unknown rotation matrix U . We therefore take as an “inductive bias” that the model class consists of a Q -function $Q_{\tilde{U}}$ and policy $\pi_{\tilde{U}}$ which are in the form of the optimal Q -function and policy for an estimate $\tilde{U} \in \mathbb{R}^{n \times n}$ of U (constrained to be orthogonal). The task is to learn this parameter. We assume that $Q_{\tilde{U}}$ and $\pi_{\tilde{U}}$ share the parameter estimate.
- Let $\tilde{U}$ be the “current” parameter estimate and $\bar{U}$ be the “target” parameter estimate for TD updates.
- We assume actions in the replay buffer are in general position.

Analysis The optimal Q-value function is of the form:

$$\begin{aligned} Q^*(s, a, g) &= \sum_{n=0}^{\infty} \gamma^n (g^T (s + U a) + n \|g\|_2) \\ &= \frac{g^T (s + U a)}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} \end{aligned} \quad (23)$$

The optimal policy π then takes the form:

$$a^* = \frac{U^T g}{\|g\|_2} \quad (24)$$

(Note that, because we are sharing the parameter $\tilde{U}$ between $Q_{\tilde{U}}$ and $\pi_{\tilde{U}}$, and because the optimal action for $Q_{\tilde{U}}$ can be written in closed form, we do not require a training step for $\pi_{\tilde{U}}$.)

The “standard” MSE Bellman error for a tuple (s, a, g, s') is

$$\begin{aligned} &[Q_{\tilde{U}}(s, a, g) - R(s', g) + \gamma Q_{\bar{U}}(s', \pi_{\bar{U}}(s, g), g)]^2 = \\ &\left[\frac{g^T (s + \tilde{U} a)}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} - g^T s' + \gamma \left(\frac{g^T (s' + \bar{U} \frac{\bar{U}^T g}{\|g\|_2})}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} \right) \right]^2 = \\ &\left[\frac{g^T (s + \tilde{U} a)}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} - g^T s' + \frac{\gamma}{1 - \gamma} g^T s' + \frac{\gamma}{1 - \gamma} \|g\|_2 + \frac{\gamma^2 \|g\|_2}{(1 - \gamma)^2} \right]^2 = (\text{using } \bar{U} \bar{U}^T = I) \\ &\left[\frac{g^T (s + \tilde{U} a)}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} - \frac{g^T s'}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} \right]^2 = \\ &\left[\frac{g^T (s + \tilde{U} a)}{1 - \gamma} - \frac{g^T s'}{1 - \gamma} \right]^2 = \\ &\left[g^T (s + \tilde{U} a) - g^T s' \right]^2 = (\text{dropping constant factor}) \\ &\left[g^T \tilde{U} a - g^T (s' - s) \right]^2 = \\ &\left[g^T \tilde{U} a - g^T U a \right]^2 \end{aligned}$$

This is equivalent to fitting the bilinear form $g^T \tilde{U} a$ using the observed scalar $g^T U a$ ($= g^T (s' - s)$). Noting that orthogonal matrices have $\sim n^2/2$ degrees of freedom, **this will require at least $O(n^2)$ samples to learn.**

Now consider our gradient-based Bellman error:

$$\begin{aligned}
& \|\nabla_g Q_{\tilde{U}}(s, a, g) - \nabla_g [R(s', g) + \gamma Q_{\tilde{U}}(s', \pi_{\tilde{U}}(s, g), g)]\|_2^2 = \\
& \left\| \nabla_g \left[\frac{g^T(s + \tilde{U}a)}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} \right] - \nabla_g \left[g^T s' + \gamma \left(\frac{g^T(s' + \tilde{U} \frac{\tilde{U}^T g}{\|g\|_2})}{1 - \gamma} + \frac{\gamma \|g\|_2}{(1 - \gamma)^2} \right) \right] \right\|_2^2 = \\
& \left\| \nabla_g \left[\frac{g^T(s + \tilde{U}a)}{1 - \gamma} \right] - \nabla_g \left[\frac{g^T s'}{1 - \gamma} \right] \right\|_2^2 = \\
& \left\| \frac{(s + \tilde{U}a)}{1 - \gamma} - \frac{s'}{1 - \gamma} \right\|_2^2 = \\
& \left\| (s + \tilde{U}a) - s' \right\|_2^2 = (\text{dropping constant factor}) \\
& \left\| \tilde{U}a - (s' - s) \right\|_2^2 = \\
& \left\| \tilde{U}a - Ua \right\|_2^2
\end{aligned}$$

Here, we are fitting the linear form $\tilde{U}a$ to the observed vector $(s' - s)$. Assuming general position, **this can be solved with $O(n)$ samples!**

Proposition 2 (No reward gradient case example)

Problem setting

- $g, a \in \mathbb{R}^n; \|a\|_2 \leq 1; s \in \mathbb{R}^{2n}$; the state vector consists of two halves, which we denote s^1 and s^2 .
 - Single step reward $R(s, g) = g^T s^1$
 - Transition function:
 - If $s_t^1 \neq \mathbf{0}$, then $s_{t+1}^1 := \mathbf{0}$, $s_{t+1}^2 := s_t^1 + Ua_t$. (Note that the reward is always zero here.)
 - If $s_t^1 = \mathbf{0}$, then $s_{t+1}^1 := s_t^2$, $s_{t+1}^2 := \mathbf{0}$,
- where $U \in \mathbb{R}^{n \times n}$ is an unknown orthogonal (rotation) matrix
- We make the same assumptions about the hypothesis class and inductive bias as in the last example, and also assume general position.

Analysis

Optimal Q-function

$$\begin{aligned}
Q^*(s, a, g) &= \begin{cases} \sum_{n \text{ odd}}^{\infty} \gamma^n (g^T(s^1 + Ua) + \frac{n-1}{2} \|g\|_2) & \text{if } s^1 \neq \mathbf{0} \\ \sum_{n \text{ even}}^{\infty} \gamma^n (g^T s^2 + \frac{n}{2} \|g\|_2) & \text{if } s^1 = \mathbf{0} \end{cases} \\
&= \begin{cases} \sum_{m=0}^{\infty} \gamma^{2m+1} (g^T(s^1 + Ua) + m \|g\|_2) & \text{if } s^1 \neq \mathbf{0} \\ \sum_{m=0}^{\infty} \gamma^{2m} (g^T s^2 + m \|g\|_2) & \text{if } s^1 = \mathbf{0} \end{cases} \\
&= \begin{cases} \frac{\gamma g^T(s^1 + Ua)}{1 - \gamma^2} + \frac{\gamma^3 \|g\|_2}{(1 - \gamma^2)^2} & \text{if } s^1 \neq \mathbf{0} \\ \frac{g^T s^2}{1 - \gamma^2} + \frac{\gamma^2 \|g\|_2}{(1 - \gamma^2)^2} & \text{if } s^1 = \mathbf{0} \end{cases} \tag{25}
\end{aligned}$$

The optimal policy π then takes the form:

$$a^* = \frac{U^T g}{\|g\|_2} \tag{26}$$

(Note that if $s^1 = \mathbf{0}$, then the action does not appear in the Q function, so any action is optimal.)

We now consider the standard TD error:

$$[Q_{\tilde{U}}(s, a, g) - R(s', g) + \gamma Q_{\tilde{U}}(s', \pi_{\tilde{U}}(s, g), g)]^2 \quad (27)$$

Note that if $s^1 = \mathbf{0}$, then the trainable parameter $\tilde{U}$ does not appear in the expression of $Q_{\tilde{U}}(s, a, g)$. Then no learning can occur from these tuples; we can instead only consider the case where $s^1 \neq \mathbf{0}$. In this case, the immediate reward is always zero, and we also know that $s^{1'} = \mathbf{0}$ so the “standard” TD update is:

$$\begin{aligned} & [Q_{\tilde{U}}(s, a, g) - \gamma Q_{\tilde{U}}(s', \pi_{\tilde{U}}(s, g), g)]^2 = \\ & \left[\frac{\gamma g^T (s^1 + \tilde{U}a)}{1 - \gamma^2} + \frac{\gamma^3 \|g\|_2}{(1 - \gamma^2)^2} - \gamma \left(\frac{g^T s^{2'}}{1 - \gamma^2} + \frac{\gamma^2 \|g\|_2}{(1 - \gamma^2)^2} \right) \right]^2 = \\ & \left[\frac{\gamma g^T (s^1 + \tilde{U}a)}{1 - \gamma^2} - \frac{\gamma g^T s^{2'}}{1 - \gamma^2} \right]^2 = \\ & \left[g^T (s^1 + \tilde{U}a) - g^T s^{2'} \right]^2 = (\text{dropping constant factor}) \\ & \left[g^T \tilde{U}a - g^T (s^{2'} - s^1) \right]^2 = \\ & \left[g^T \tilde{U}a - g^T Ua \right]^2 \end{aligned}$$

This is the same update as in the previous example, and again **we need $O(n^2)$ samples**. (Note that there is a constant factor of 2 increase in the number of needed samples, due to the wasted samples in which $s^1 = \mathbf{0}$. However, assuming general position, this will only account for half of the replay buffer.)

We now consider the case in which we use our gradient TD update. Again we only can use the tuples where $s^1 \neq \mathbf{0}$. For all of these, the immediate reward is zero, so we can use the gradient-based update for all of them.

$$\begin{aligned} & \|\nabla_g Q_{\tilde{U}}(s, a, g) - \gamma \nabla_g Q_{\tilde{U}}(s', \pi_{\tilde{U}}(s, g), g)\|_2^2 = \\ & \left\| \nabla_g \left(\frac{\gamma g^T (s^1 + \tilde{U}a)}{1 - \gamma^2} + \frac{\gamma^3 \|g\|_2}{(1 - \gamma^2)^2} \right) - \gamma \nabla_g \left(\frac{g^T s^{2'}}{1 - \gamma^2} + \frac{\gamma^2 \|g\|_2}{(1 - \gamma^2)^2} \right) \right\|_2^2 = \\ & \left\| \frac{\gamma (s^1 + \tilde{U}a)}{1 - \gamma^2} - \frac{\gamma s^{2'}}{1 - \gamma^2} \right\|_2^2 = \\ & \left\| (s^1 + \tilde{U}a) - s^{2'} \right\|_2^2 = (\text{dropping constant factor}) \\ & \left\| \tilde{U}a - (s^{2'} - s^1) \right\|_2^2 = \\ & \left\| \tilde{U}a - Ua \right\|_2^2 \end{aligned}$$

As in the previous gradient feedback case, assuming general position, **this can be solved with $O(n)$ samples!**

ReenGAGE for Discrete Actions

Note that Equation 8 requires that the gradient:

$$\nabla_g Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \quad (28)$$

be computable. This means that $\pi_{\phi'}(s', g)$ must be continuous and differentiable, which implies a continuous action space. To extend our method to discrete action spaces, we instead consider the target Q-value of DQN (Mnih et al. 2015), the standard baseline method for discrete Q-learning, in a goal-conditioned setting:

$$R(s', g) + \gamma \max_a Q_{\theta', a}(s', g), \quad (29)$$

where $Q_{\theta} \in S \times G \rightarrow \mathbb{R}^{|A|}$. This is not differentiable everywhere with respect to g : in particular, at points where, for some pair of actions a', a'' , we have:

$$\max_a Q_{\theta', a}(s', g) = Q_{\theta', a'}(s', g) = Q_{\theta', a''}(s', g), \quad (30)$$

the gradient with respect to g is not necessarily defined. In practice, this means that naively using auto-differentiation to take the gradient of Equation 29 produces the gradient of the target with respect to the goal *assuming the optimal action remains constant*. To overcome this, we consider instead using a *soft target*:

$$\begin{aligned} \text{SoftQTarget}(s', g) &:= R(s', g) + \gamma \text{SoftMax}[Q_{\theta'}(s', g)/\tau] \cdot Q_{\theta'}(s', g) \\ &= R(s', g) + \gamma \frac{\sum_{a \in A} Q_{\theta', a}(s', g) e^{Q_{\theta', a}(s', g)/\tau}}{\sum_{a' \in A} e^{Q_{\theta', a'}(s', g)/\tau}}, \end{aligned} \quad (31)$$

Where τ is the temperature hyperparameter. (Note that this approaches the standard DQN target as $\tau \rightarrow 0$.) We tested three uses of this soft target:

- Soft target for *gradient loss only*:

$$\begin{aligned} \mathcal{L} = & \mathbb{E}_{(s, a, s', g) \sim \text{Buffer}} \left[\mathcal{L}_{\text{Huber}} \left[Q_{\theta, a}(s, g), R(s', g) + \gamma \max_a Q_{\theta', a}(s', g) \right] \right. \\ & \left. + \alpha \mathcal{L}_{\text{mse}} \left[\nabla_g Q_{\theta, a}(s, g), \gamma \nabla_g \text{SoftQTarget}(s', g) \right] \right] \end{aligned} \quad (32)$$

Following (Mnih et al. 2015), we use the Huber loss for the main loss term, although we use MSE for the gradient loss.

- Soft target for *both loss terms*:

$$\begin{aligned} \mathcal{L} = & \mathbb{E}_{(s, a, s', g) \sim \text{Buffer}} \left[\mathcal{L}_{\text{Huber}} \left[Q_{\theta, a}(s, g), \text{SoftQTarget}(s', g) \right] \right. \\ & \left. + \alpha \mathcal{L}_{\text{mse}} \left[\nabla_g Q_{\theta, a}(s, g), \gamma \nabla_g \text{SoftQTarget}(s', g) \right] \right] \end{aligned} \quad (33)$$

- Soft target for both loss terms, *and* take actions nondeterministically, with a probability distribution given by $\text{SoftMax}[Q_{\theta'}(s', g)/\tau]$.

We test on an implementation of the “Bit-Flipping” sparse-reward environment from (Andrychowicz et al. 2017), with dimensionality $d = 40$ bits using DQN with HER as the baseline. Because code for this experiment is not provided by (Andrychowicz et al. 2017), we re-implemented the experiment using the Stable-Baselines3 package (Raffin et al. 2021). We used the hyperparameters specified by (Andrychowicz et al. 2017), with the following minor modifications (note that for some of these specifications, (Andrychowicz et al. 2017) is unclear about whether the specification was applied for the Bit-Flipping environment, or only in the continuous control, DDPG, environments tested in that work):

- We train on a single GPU, rather than averaging gradient updates from 8 workers.
- While we use Adam optimizer with learning rate of 0.001 as specified, we use PyTorch defaults for other Adam hyperparameters, rather than TensorFlow defaults.
- We do not clip the target Q-values (this feature is not part of the DQN implementation of (Raffin et al. 2021)).
- We evaluate using the current, rather than target, Q-value function.
- We do not normalize observations (which are already $\{0, 1\}$).

The baselines we present are from our re-implementation, to provide a fair comparison. For our method, we performed a grid search on $\alpha \in \{0.5, 1.0\}$ and $\tau \in \{0.0, 0.5, 1.0\}$. Results are presented in Figure 7. We observed that the “gradient loss only” method was effective at improving performance, both over standard DQN and over ReenGAGE with standard DQN targets ($\tau = 0$). By contrast, using soft targets for both loss terms made the performance worse, and using nondeterministic actions with soft targets for both loss terms brought the success rate to zero (and hence is not shown). We tested all three methods with $d = 40$ bits, and then applied the successful method (“gradient loss only”) to $d = 20$ as well; improvements over baseline for $d = 20$ were minor.

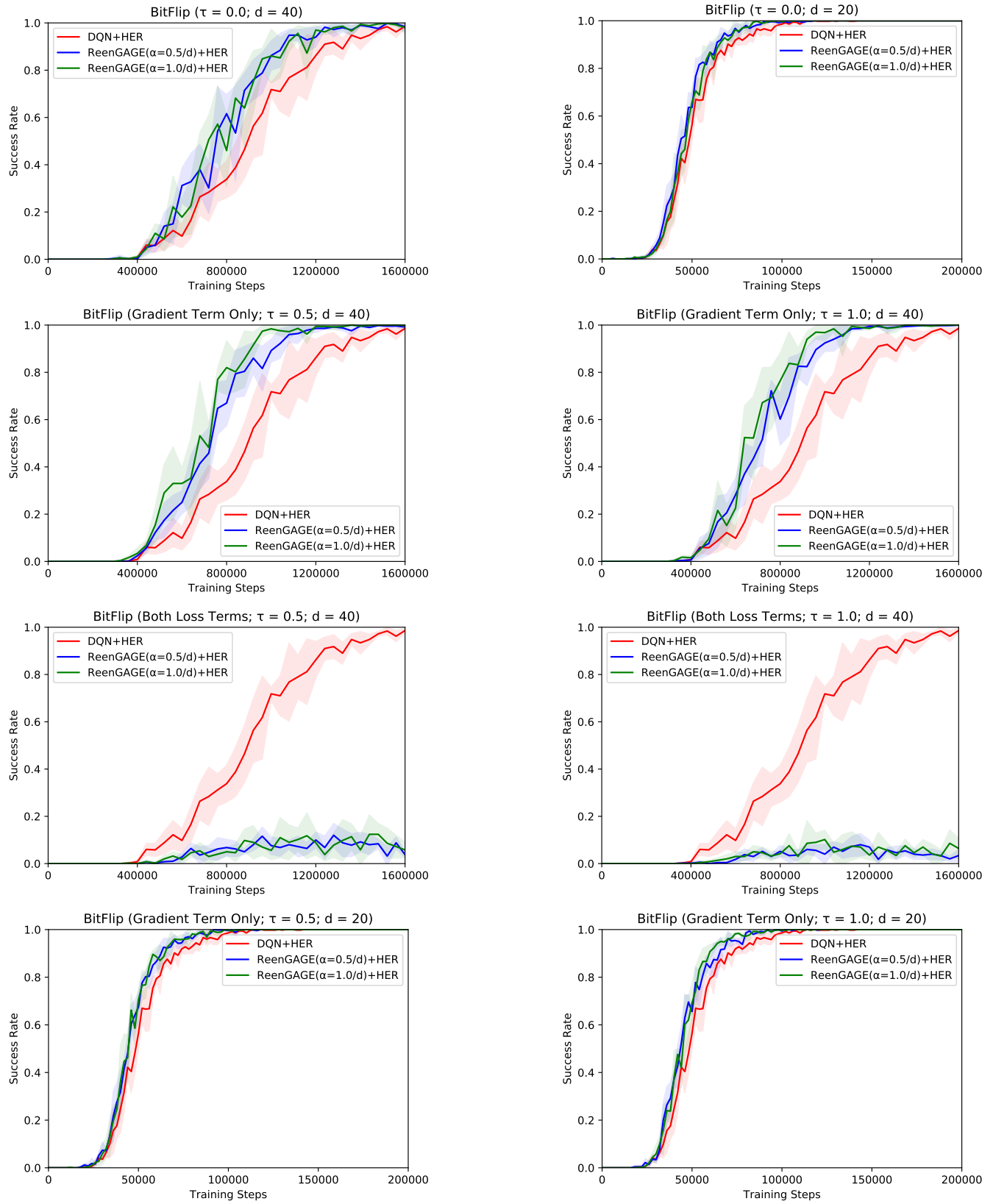


Figure 7: Results for the Bit-Flipping environment.

Replay Buffer Size	1000000
Frequency of Training	Every 1 environment step
Gradient Descent Steps per Training	1
Initial Steps before Training	1000
Discount γ	0.95
Polyak Update τ	0.005
Normal action noise for training σ	0.03
Architecture (both actor and critic)	Fully Connected; 2 hidden layers of width 256; ReLU activations
HER number relabeled goals k	4
HER relabeling strategy	‘Future’
Evaluation episodes	50
Evaluation Frequency	Every 2000 environment steps

Table 1: Hyperparameters for ContinuousSeek

Hyperparameters and Full Results for ContinuousSeek

As mentioned in the main text, we performed a full hyperparameter search over the batch size (in $\{128, 256, 512\}$) and learning rate (in $\{0.00025, 0.0005, 0.001, 0.0015\}$). The results in the main paper represent the best single curve (as defined by area under the curve, or in other words best average score over time) for the baseline and each value of the ReenGAGE α term. Complete results are presented in Figures 8, 9, and 10. The values of these parameters which yielded the best average performance, and were therefore reported in the main text, were as follows (Table 2):

	DDPG+HER		ReenGAGE($\alpha = 0.1$)+HER		ReenGAGE($\alpha = 0.2$)+HER		ReenGAGE($\alpha = 0.3$)+HER	
	Batch	LR	Batch	LR	Batch	LR	Batch	LR
$d = 5$	512	0.001	512	0.0015	512	0.0015	512	0.0015
$d = 10$	256	0.0005	512	0.001	512	0.001	512	0.001
$d = 20$	256	0.0005	128	0.0005	128	0.0005	128	0.0005

Table 2: “Best” batch sizes and learning rates for ContinuousSeek for DDPG and ReenGAGE.

Note that for the larger-scale experiments ($d = 10$ and $d = 20$) the “best” hyperparameters for the baseline DDPG+HER model lie in the interior of the search space of both the batch size and learning rate: this would seem to imply (assuming concavity) that increasing the range of the search space of these parameters would likely not improve the baseline. However, this is not the case for ReenGAGE: we could perhaps achieve even better performance for ReenGAGE by conducting a larger search, specifically in the batch size dimension.

Other hyperparameters were fixed, and are listed in Table 1. We use the implementation of HER with DDPG from Stable-Baselines3 (Raffin et al. 2021) as our baseline; any unlisted hyperparameters are the default from this package. Note that we use the “online” variant of HER provided in the Stable-Baselines3 package.

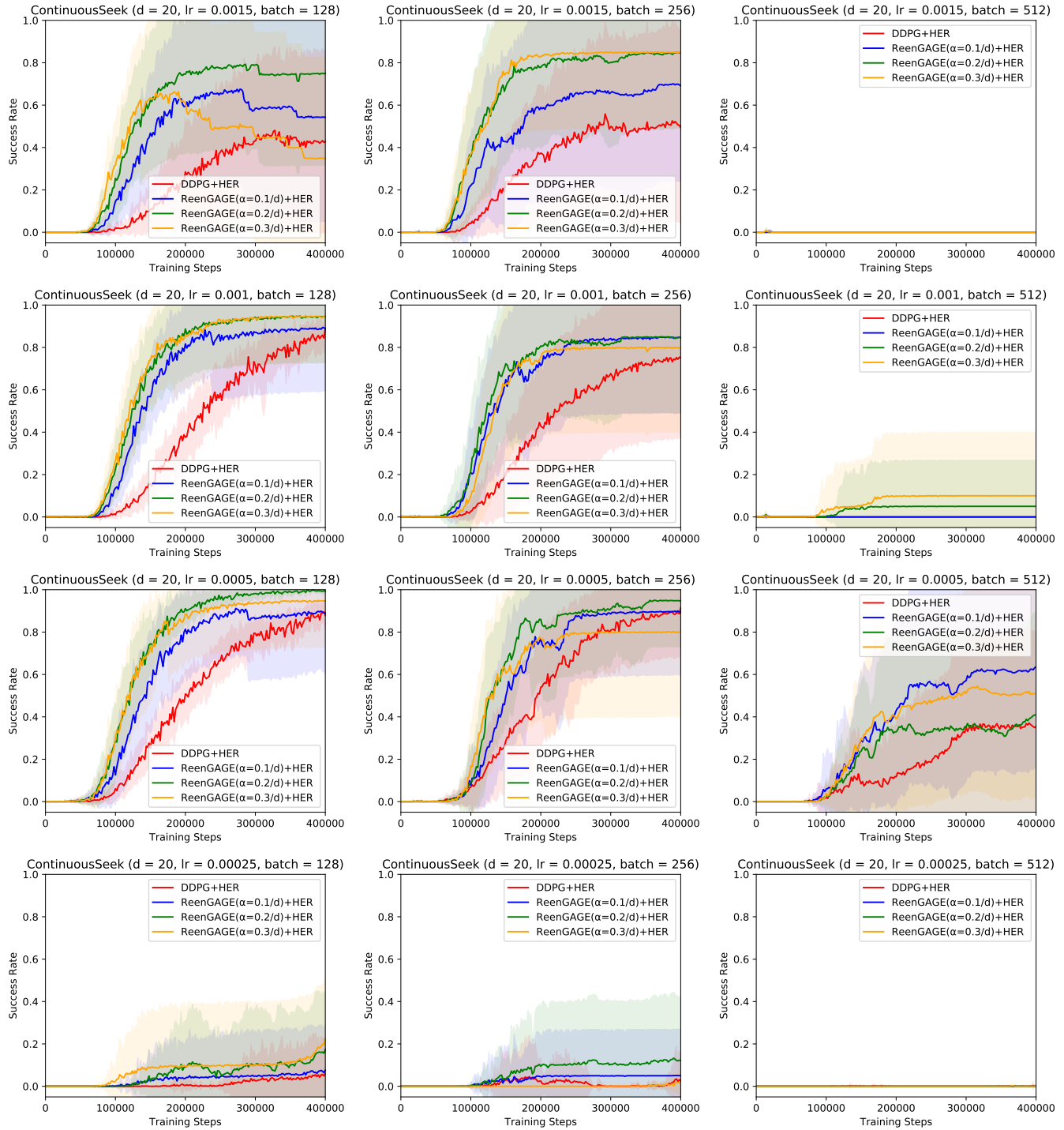


Figure 8: Complete ContinuousSeek Results for $d = 20$.

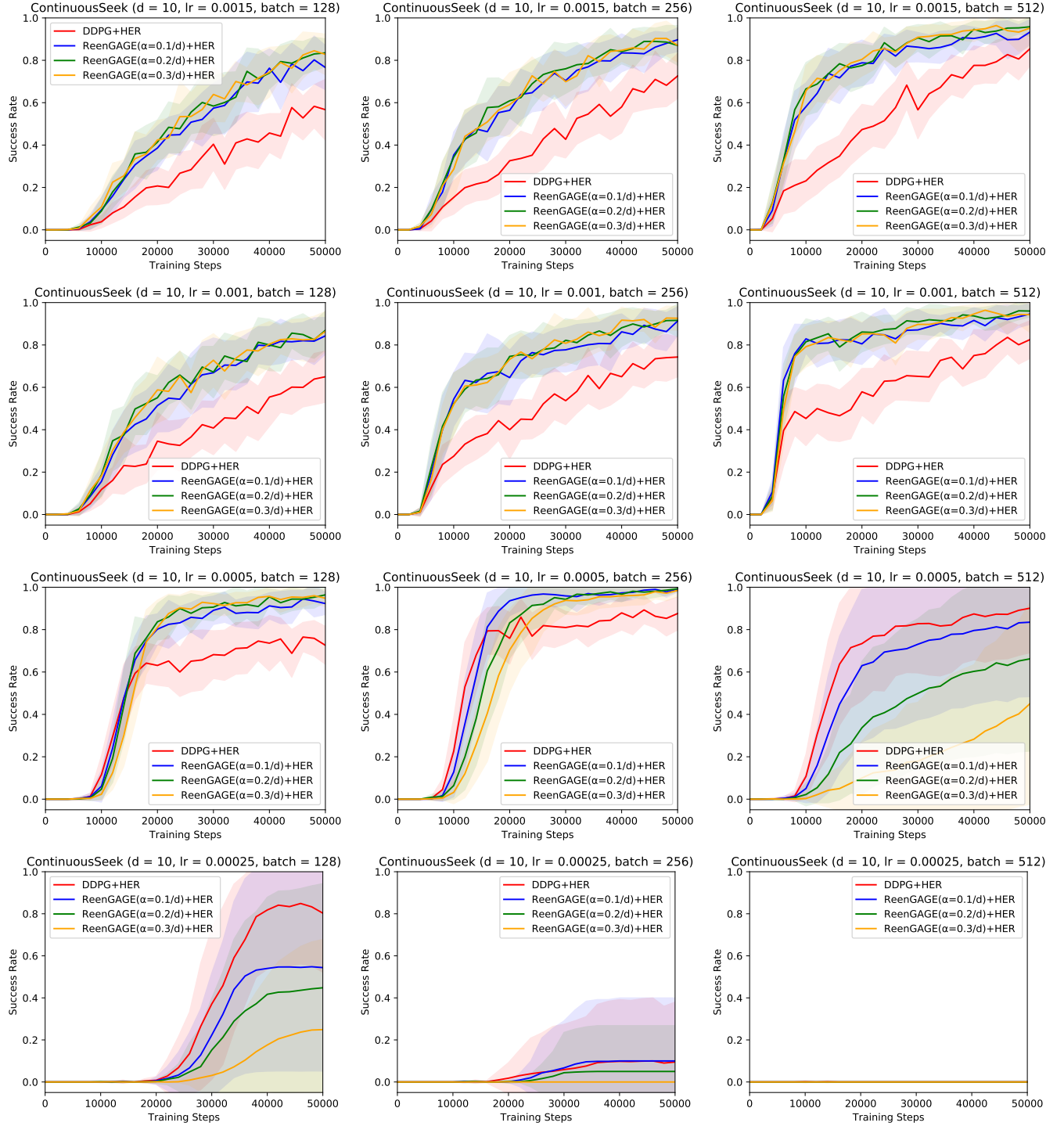


Figure 9: Complete ContinuousSeek Results for $d = 10$.

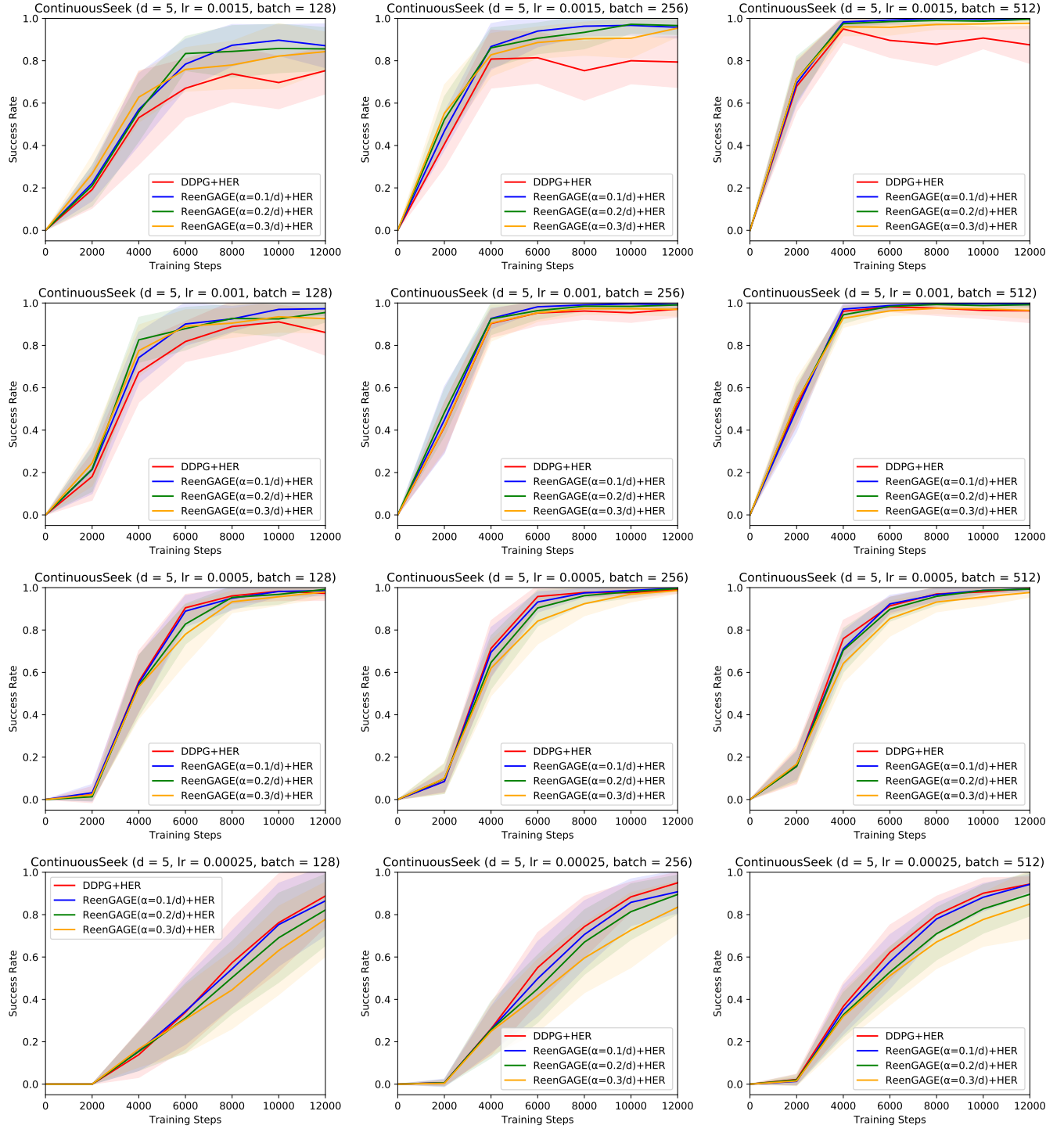


Figure 10: Complete ContinuousSeek Results for $d = 5$.

Additional Results for Robotics Experiments

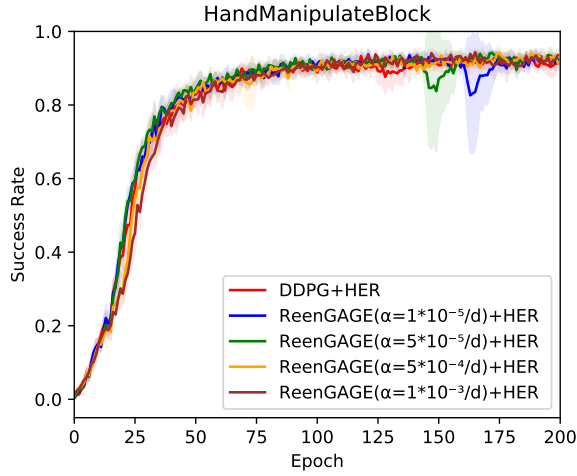
In Figure 11, we provide additional results for the robotics experiments. In Figure 11-(a) and (b), we give results for the **HandManipulateBlock** environment: as mentioned in the text, we did not see a significant advantage to using ReenGAGE for this environment.

In addition to the lower-dimensional goal space of this problem ($d = 7$, versus $d = 15$ for HandReach), one additional possible reason why ReenGAGE may underperform in this setting is that, unlike in the HandReach case, the dimensions of the goal vector represent diverse quantities of significantly varying scales: some represent angular measurements, while others represent position measurements.

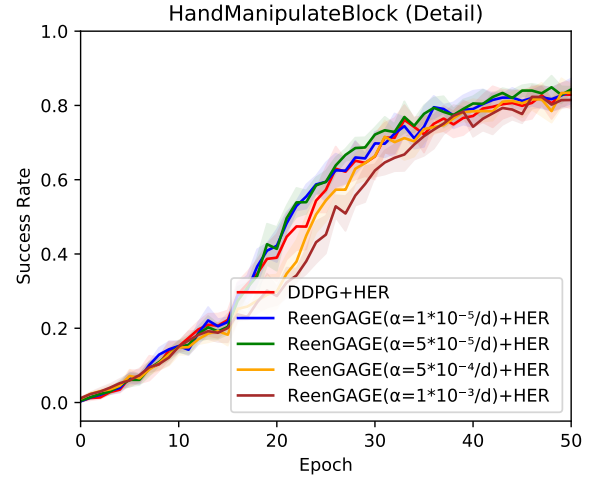
In order to handle this, we attempted normalizing each dimension of the ReenGAGE loss term. We accomplish this by multiplying the MSE loss for each coordinate i by σ_i^2 , where σ_i is the running average standard deviation of dimension i of the goals. ((Plappert et al. 2018)’s implementation already computes these averages in order to normalize inputs to neural networks.) Intuitively, we do this because the derivative in a given coordinate is in general inversely proportional to the scale of that coordinate (i.e., if $y = 2x$, then $\frac{df}{dy} = 0.5 \frac{df}{dx}$), and because the ReenGAGE MSE loss in each dimension is proportional to the square of the derivative.

However, unfortunately, this did not result in superior performance for HandManipulateBlock: see Figure 11-(c) and (d). As a sanity check, we confirmed that this method performed similarly to the non-normalized ReenGAGE method on HandReach (11-(e)).

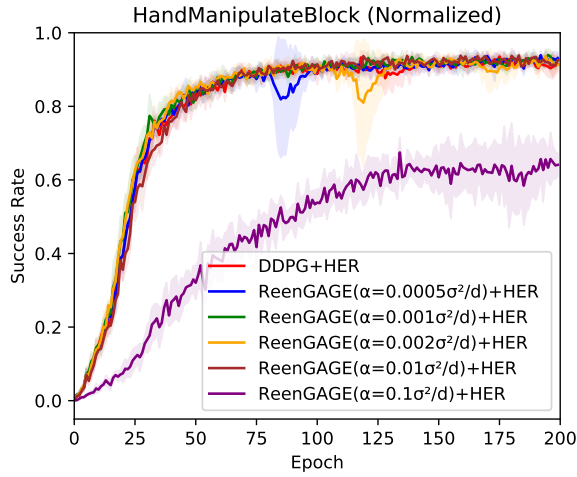
Finally, in Figure 11-(f), we show some additional experiments on HandReach. Specifically, we show results using a single random seed for a wider range of α than shown in the main text: this was performed as a first pass to find the appropriate range of the hyperparameter α to use for the complete experiments. As shown in the main text results, ReenGAGE becomes unstable if too-high values of α are used.



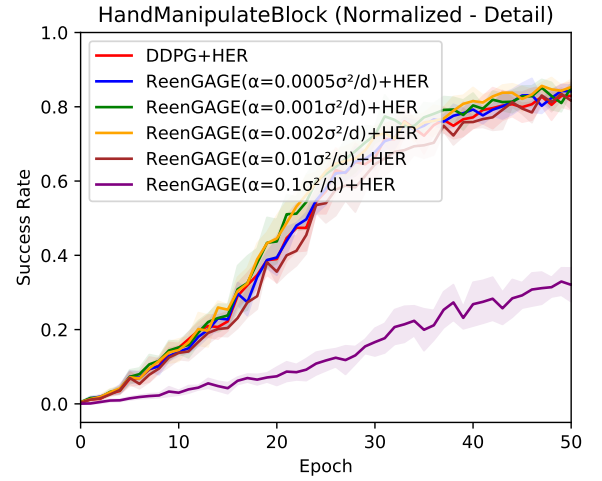
(a)



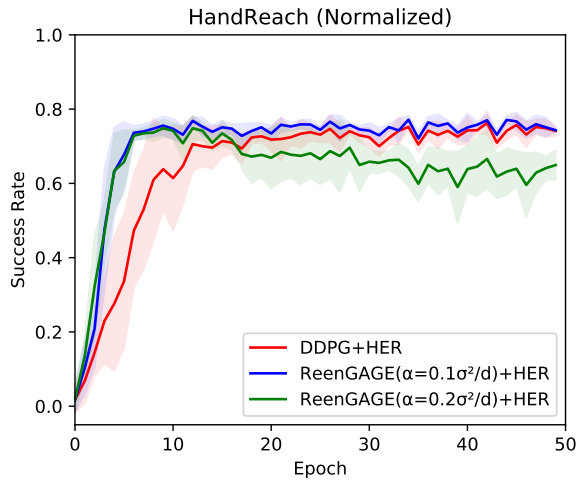
(b)



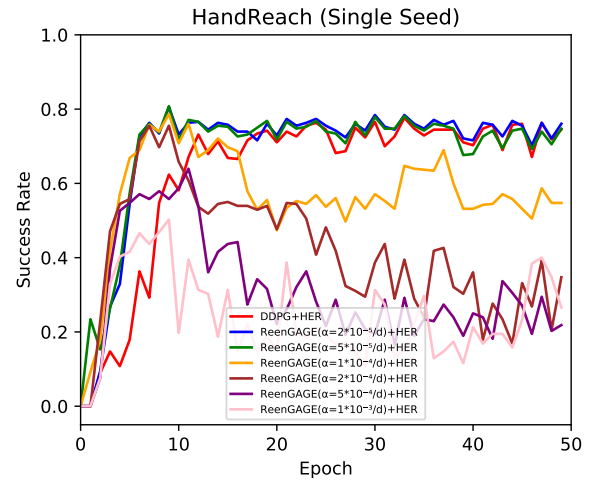
(c)



(d)



(e)



(f)

Figure 11: Additional Results from Robotics experiments. See text for details.

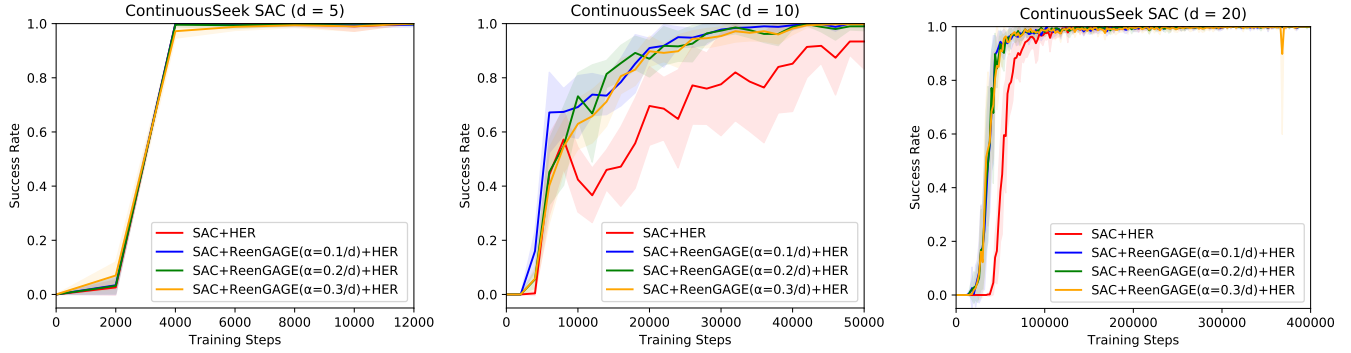


Figure 12: ContinuousSeek results with SAC. Lines show the mean and standard deviation over 10 random seeds (kept the same for all experiments.) The Y-axis represents the success rate, defined as the fraction of test episodes for which the goal is ever reached.

	SAC+HER		SAC+ReenGAGE($\alpha=.1$)+HER		SAC+ReenGAGE($\alpha=.2$)+HER		SAC+ReenGAGE($\alpha=.3$)+HER	
	Batch	LR	Batch	LR	Batch	LR	Batch	LR
d=5	8192	0.0025	4096	0.0025	2048	0.0025	2048	0.0025
d=10	1024	0.0015	1024	0.0025	512	0.0025	512	0.0025
d=20	4096	0.0005	8192	0.0025	8192	0.0025	8192	0.0025

Table 3: “Best” batch sizes and learning rates for ContinuousSeek for SAC and SAC+ReenGAGE.

ReenGAGE with SAC

In this section, we explore applying ReenGAGE on top of SAC (Haarnoja et al. 2018), a variant of DDPG which uses a stochastic policy and rewards for high-entropy policies, uses the current policy to compute targets, and also uses an ensemble critic architecture. The application of ReenGAGE to SAC is straightforward: when computing estimates for $\nabla_g \text{Targ.}(g; s, a)$, we use the well-known “reparameterization trick” to differentiate through the stochastic policy (which depends on the goal) and we also differentiate through the entropy reward (which depends on policy’s action distribution, and therefore on the goal).

In our experiments, we combine ReenGAGE with SAC and HER, test on our ContinuousSeek environment, and compare to an SAC+HER baseline. Hyperparameter-optimized “best” results for each value of α are shown in Figure 12. As in our DDPG experiments, we performed a grid search over batch size and learning rate hyperparameters. However, using the range of these parameters we used for DDPG led to all best baselines and ReenGAGE models having values of the hyperparameters lying at the “edges” and “corners” of the hyperparameter grid search space. We therefore increased the search space size, testing all learning rates in $\{0.00025, 0.0005, 0.001, 0.0015, 0.0025\}$ and batch sizes in $\{128, 256, 512, 1024, 2048, 4096, 8192\}$. This led to optimal hyperparameters for the baseline in the interior of the search space for the larger scale experiments ($d = 10$ and $d = 20$); see Table 3. As with the DDPG ContinuousSeek experiments, the ReenGAGE models still lie on edges/corners of the hyperparameter search space, so it may be possible to get even better ReenGAGE performance by increasing the search space further. Full results are presented in Figures 13, 14, and 15. Note that due to computational limits, we only use 10 random seeds in these experiments, as opposed to 20 for the DDPG experiments.

Other, non-optimized hyperparameters were fixed, and are generally the same as those for DDPG listed in Table 1. As with DDPG, we use the implementation of SAC from Stable-Baselines3 (Raffin et al. 2021) as our baseline; any unlisted hyperparameters are the default from this package.

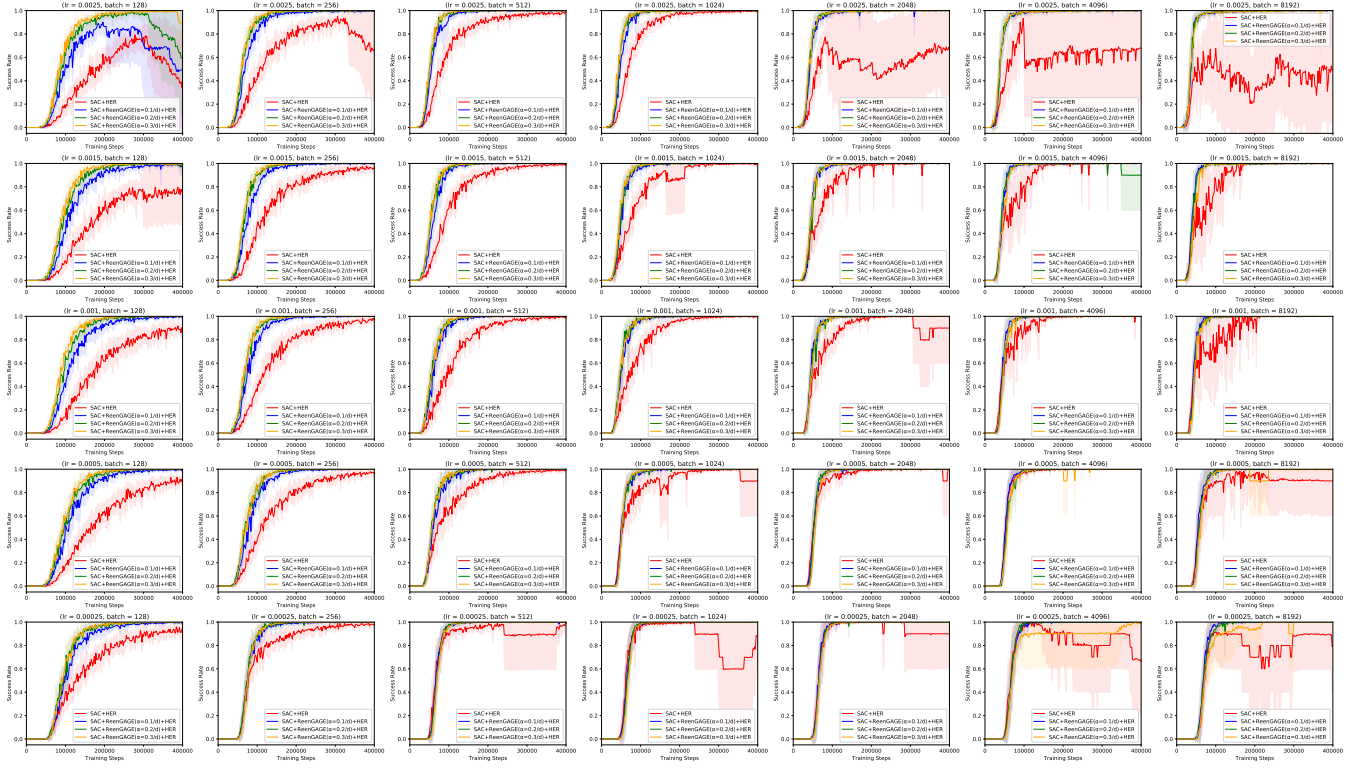


Figure 13: Complete SAC ContinuousSeek results for $d = 20$.

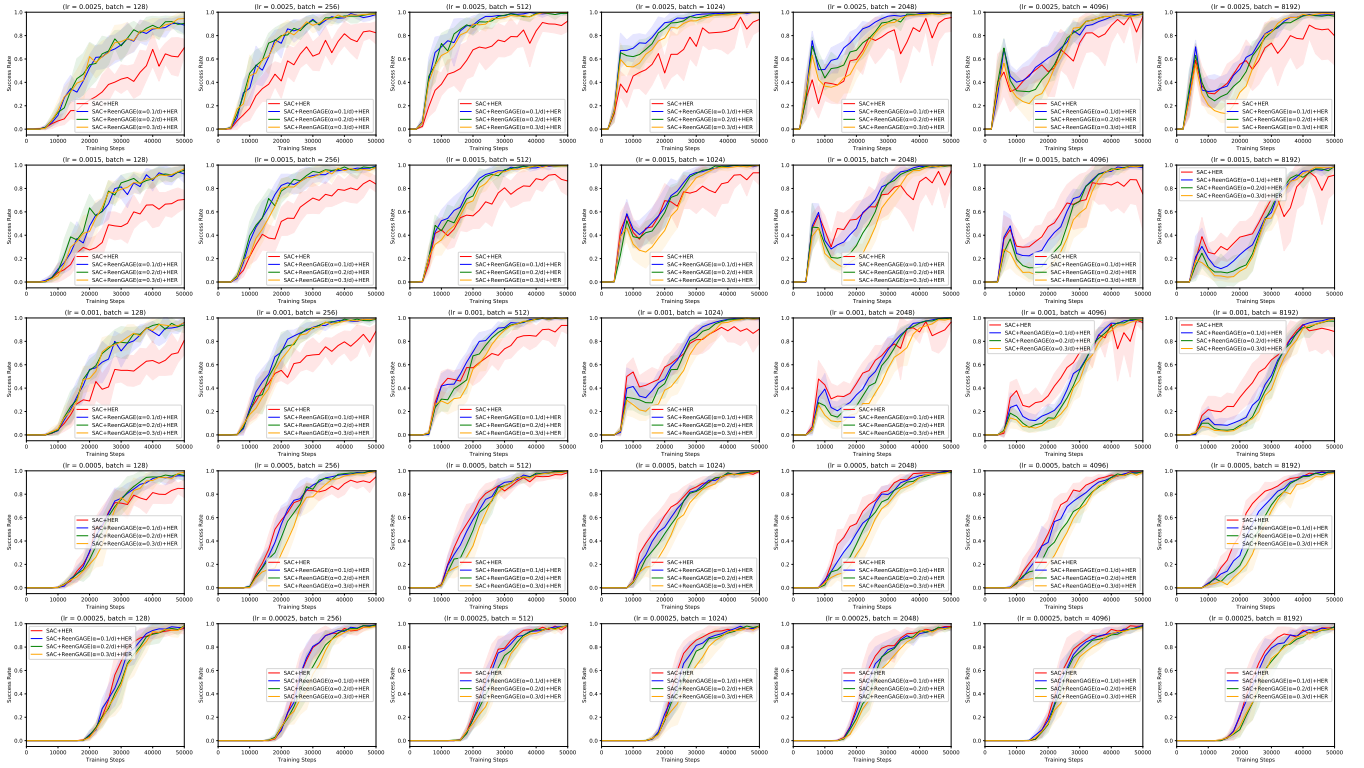


Figure 14: Complete SAC ContinuousSeek results for $d = 10$.

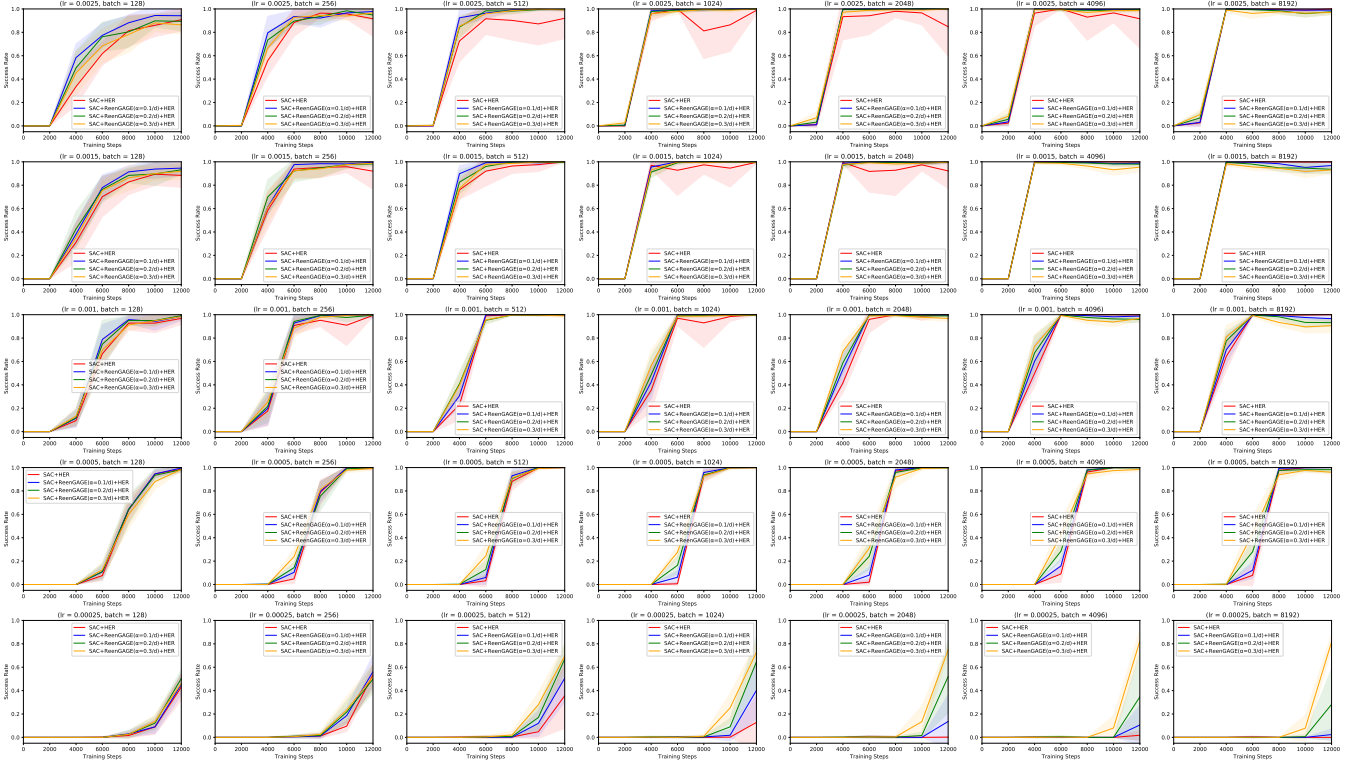


Figure 15: Complete SAC ContinuousSeek results for $d = 5$.

Multi-ReinGAGE Implementation Details

Batch Implementation

To efficiently implement the multi-goal Q-function in Equation 15 as a batch equation, we use a constant-size representation of the goal set $g = \{g_1, \dots, g_n\}$. Specifically, we let $g = \{g_1, \dots, g_{n_{\max}}\}$, where $\{g_{n+1}, \dots, g_{n_{\max}}\}$ are set to a dummy value (practically, 0), and the gate variables $\{b_{n+1}, \dots, b_{n_{\max}}\}$ are all set to zero. Then:

$$Q_{\theta}(s, a, g) := Q_{\theta}^{\text{head}}(s, a, \sum_{i=1}^n [b_i Q_{\theta}^{\text{encoder}}(s, g_i)]) = Q_{\theta}^{\text{head}}(s, a, \sum_{i=1}^{n_{\max}} [b_i Q_{\theta}^{\text{encoder}}(s, g_i)]). \quad (34)$$

However, while this is equal to the intended form of the Q-function without the dummy inputs, the gradient with respect to the full vector b differs from the intended form. Specifically, note that:

$$\frac{\partial Q_{\theta}(s, a, g)}{\partial b_i} = Q_{\theta}^{\text{encoder}}(s, g_i) \cdot (\nabla Q_{\theta}^{\text{head}})(s, a, \sum_{i=1}^{n_{\max}} [b_i Q_{\theta}^{\text{encoder}}(s, g_i)]) \quad (35)$$

In particular, this is nonzero even when b_i is zero, so the gradient will depend on the **dummy** value $Q_{\theta}^{\text{encoder}}(s, 0)$. This is clearly not intended. To prevent this, we instead use the form:

$$Q_{\theta}^{\text{head}}(s, a, \sum_{i=1}^{n_{\max}} [b_i^2 Q_{\theta}^{\text{encoder}}(s, g_i)]). \quad (36)$$

(For the target, we construct the policy network similarly using b_i^2 's). Note that the above form of the Q-function is equal to the intended form of the Q-function, because $0^2 = 0$ and $1^2 = 1$. However, in this case:

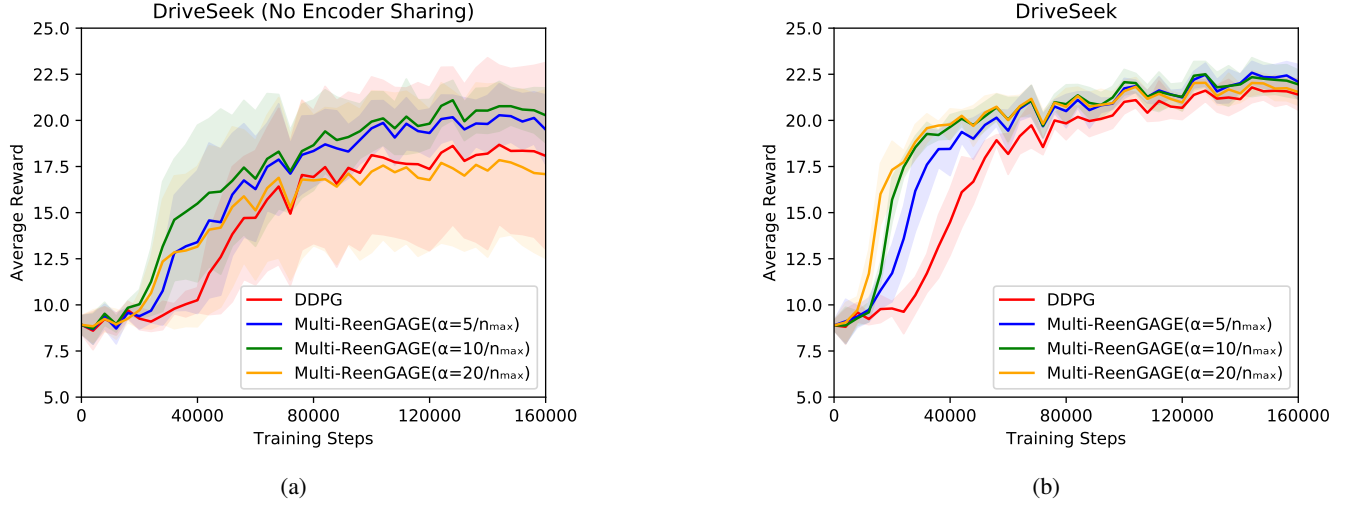


Figure 16: Ablation study of encoder sharing for Multi-ReenGAGE on DriveSeek. (a) Without Encoder Sharing; (b) With Encoder Sharing.

$$\begin{aligned}
\frac{\partial Q_{\theta}(s, a, g)}{\partial b_i} &= \\
2b_i Q_{\theta^e}^{\text{encoder}}(s, g_i) \cdot (\nabla Q_{\theta^h}^{\text{head}})(s, a, \sum_{i=1}^{n_{\max}} [b_i Q_{\theta^e}^{\text{encoder}}(s, g_i)]) &= \\
\begin{cases} 2Q_{\theta^e}^{\text{encoder}}(s, g_i) \cdot (\nabla Q_{\theta^h}^{\text{head}})(s, a, \sum_{i=1}^{n_{\max}} [b_i Q_{\theta^e}^{\text{encoder}}(s, g_i)]) & \text{if } b_i = 1 \\ 0 & \text{if } b_i = 0 \end{cases} & \quad (37)
\end{aligned}$$

Which is the intended gradient of the Q-function without the dummy inputs, times a constant factor of two. For the reward, we also use:

$$R(s', g) = \sum_{g_i \in g} b_i^2 R_{\text{item}}(s', g_i) \quad (38)$$

So that the gradient is:

$$\frac{\partial R}{\partial b_i} = b_i R_{\text{item}}(s', g_i) = \begin{cases} 2R_{\text{item}}(s', g_i) & \text{if } b_i = 1 \\ 0 & \text{if } b_i = 0 \end{cases} \quad (39)$$

Which again the intended gradient of the Q-function without the dummy inputs, times a constant factor of two. Putting everything together, the final gradient loss term is in the form:

$$\mathcal{L}_{\text{Multi-ReenGAGE}} = \mathcal{L}_{\text{DDPG-Critic}} + \begin{cases} \alpha \mathcal{L}_{\text{mse}} \left[2 \nabla_b Q_{\theta}(s, a, g), 2R_{\text{item}}(s', g) + 2\gamma \nabla_b Q_{\theta'}(s', \pi_{\phi'}(s', g), g) \right] & \text{if } b_i = 1 \\ \alpha \mathcal{L}_{\text{mse}} [0, 0] (= 0) & \text{if } b_i = 0 \end{cases} \quad (40)$$

Which is the desired loss, up to a constant factor.

Shared Encoder Ablation Study

We performed an ablation study on sharing the encoder between the Q-value function and the policy on DriveSeek when using Multi-ReenGAGE. Results are presented in Figure 16. We see that sharing the encoder improves the performance of both Multi-ReenGAGE and the DDPG baseline.

Training Hyperparameters for Experiments

The hyperparameters used for the Multi-ReenGAGE experiments are presented in Table 4.

Replay Buffer Size	1000000
Frequency of Training	Every 1 environment step
Gradient Descent Steps per Training	1
Initial Steps before Training	1000
Discount γ	0.95
Batch Size	256
Learning Rate	0.001
Polyak Update τ	0.005
Normal action noise for training σ	0.05 for DriveSeek; 0.1 for NoisySeek
Embedding Dimension	20
Extractor Architecture	Fully Connected; 2 hidden layers of width 400; ReLU activations
Head Architecture (both actor and critic)	Fully Connected; 1 hidden layer of width 400; ReLU activation
Evaluation episodes	100
Evaluation Frequency	Every 4000 environment steps

Table 4: Hyperparameters for Multi-ReenGAGE Experiments

Additional Details about Environments

In this section, we provide additional details about the DriveSeek and NoisySeek environments not included in the main text, in order to more completely describe them.

DriveSeek In the DriveSeek environment, the initial position is always fixed at $(0, 0)$, and the initial velocity vector is always at 0 radians. Episodes last 40 time steps. Goals are sampled by the following procedure: first, a number of goals n is chosen uniformly at random from $\{1, \dots, 200\}$; then n goals are chosen uniformly without replacement from the integer coordinates in $[-10, 10]^2$. In addition to the goals, the observation s that the agent and policy receives is 6 dimensional: it consists of the current position s_{pos} , the *rounded* version of s_{pos} . (this is the “achieved goal”: a reward is obtained if this matches one of the input goals), and the sine and cosine of the velocity vector.

NoisySeek In the NoisySeek Environment, the initial position is fixed a $(0, 0)$. Episodes last 40 time steps. Goals are sampled by the following procedure: first, a number of clusters n_c is chosen from a geometric distribution with parameter $p = 0.15$, and a maximum number of goals n' is chosen uniformly at random from $\{1, \dots, 200\}$. Then, cluster centers are chosen from a Gaussian distribution with mean 0 and standard deviation 10 in both dimensions. Next, a Dirichlet distribution of order $K = n_c$, with $\alpha_1, \dots, \alpha_K = 1$, is used to assign a probability p_j to each cluster. Next, each of the n' goals are assigned to a cluster, with probability p_j of being assigned to cluster j . Then, each goal is determined by adding Gaussian noise with mean 0 and standard deviation 2 in both dimensions to the cluster center assigned to that goal, and then rounding to the nearest integer coordinates. Finally, goals are de-duplicated.

In addition to the goals, the observation s that the agent and policy receives is 4 dimensional: it consists of the current position s , and the *rounded* version of s (this is the “achieved goal”: a reward is obtained if this matches one of the input goals).

NoisySeek results for additional α values

We tested NoisySeek with additional values of α , which we did not include in the main text to avoid cluttered presentation; the trend is generally the same as shown in the main text. Results are shown in Figure 17.

DriveSeek with CNN Architecture

Because the positions in the DriveSeek environment are bounded, we attempted using a CNN architecture to interpret the state and goals. In particular, because all possible goals appear at unique locations in two-dimensional space, rather than using a DeepSets (Zaheer et al. 2017)–style architecture, we can directly surface the goal “gates” b_i as part of the input image: the location in the image corresponding to g_i is blank if b_i is zero (the goal is absent) and colored in if b_i is one (the goal is present). See Figure 18. Note that, as in the DeepSet implementation, we use b_i^2 in the representation, so that goals which are absent have zero associated attention. We used the standard “Nature CNN” architecture from (Mnih et al. 2015) with otherwise the same training hyperparameters as used in the main-text experiment. The CNN was shared between the actor and the critic networks, and trained only using the critic loss; its output was then fed into separate actor and critic heads, consisting of a single hidden layer of width 400, as in the DeepSets-based architecture. Results are shown in Figure 19.

In general, the performance was worse than using the DeepSets-style architecture; however, the models using Multi-ReenGAGE still outperform the standard DDPG models. One possible explanation for this is that the hyperparameters are poorly-tuned for the CNN architecture. Doubling and halving the learning rate (to 0.002 and 0.0005, respectively) did not seem to affect the results much (Figure 20), but it is possible that other hyperparameter adjustment may lead to better performance. Another possible explanation for the poor performance is that the pixel-resolution of the images (each pixel has width of 0.125

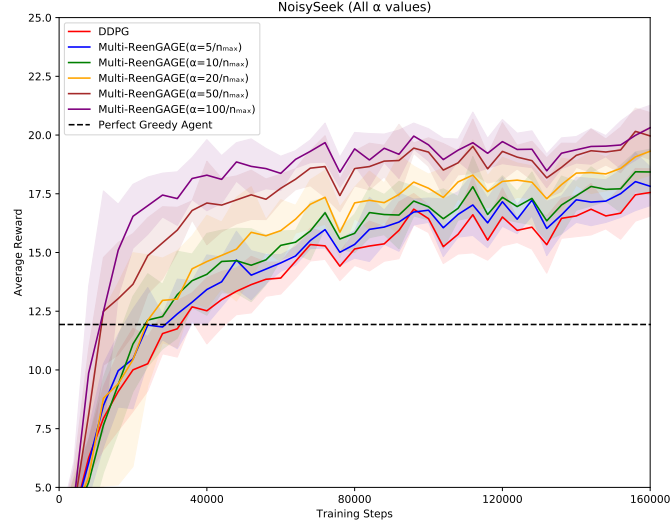


Figure 17: Results for NoisySeek Environment including additional values of α .

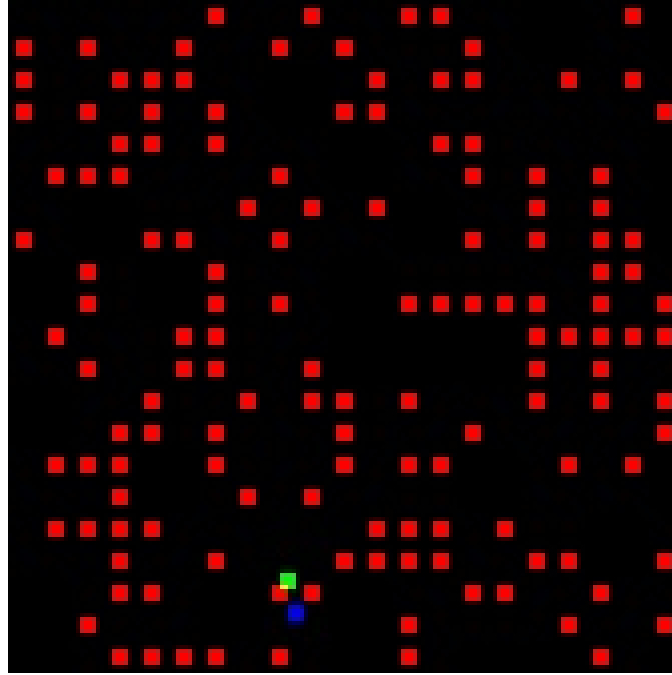


Figure 18: Example input to CNN architecture for DriveSeek. We use one channel (red) to represent the goals (note that these pixels directly correspond to the differentiable indicator variables b_i^2 for each possible goal), another channel (green) to represent the current position, and a third channel (blue) to represent the next position if the action a is zero: in other words, it indicates $s_{vel.}$. Goals are spaced out so that (approximately, up to single-pixel rounding) a goal is achieved if the current position indicator (green) overlaps with the (red) goal. The exact position and velocity vectors are also provided as a separate input, apart from the CNN.

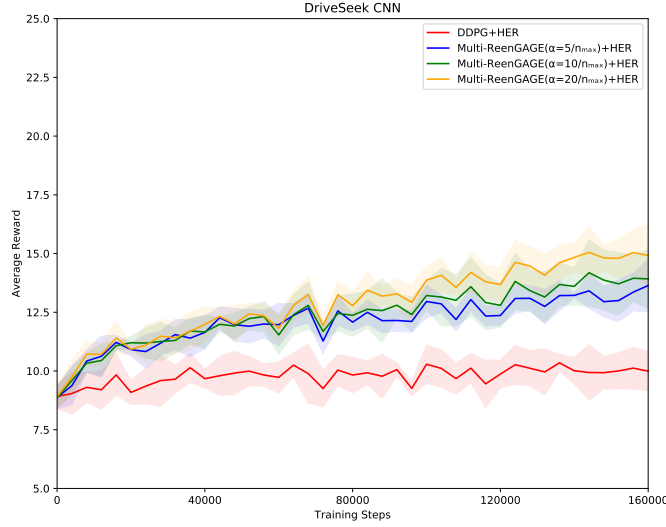


Figure 19: Results for DriveSeek with CNN architecture.

units) was not sufficient to capture the real-valued dynamics of the environment (although we did also include the real-valued state position and velocity vectors [and rounded state position] as inputs concatenated to the CNN output).

Runtime Discussion and Empirical Comparisons

As noted in (Etmann 2019), loss terms involving gradients with respect to inputs, such as the ReenGAGE loss, should only scale the computational cost of computing the parameter gradient update by a constant factor.

For an intuition as to why this is the case, recall the well-known fact that for a differentiable scalar-output function $f(\mathbf{x}, \theta)$ represented by some computational graph which requires n operations to compute, standard backpropagation can compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x}, \theta)$ (or the gradient $\nabla_{\theta} f(\mathbf{x}, \theta)$) using only cn operations, for a constant c . However, now note that $\nabla_{\mathbf{x}} f(\mathbf{x}, \theta)$ can *itself* be thought of as a cn -operation component of a larger computational graph. Thus, if we consider the scalar-valued function $h(\nabla_{\mathbf{x}} f(\mathbf{x}, \theta))$, where h itself takes k additional operations to compute, then we can upper-bound the total number of operations required to compute the gradient $\nabla_{\theta} h(\nabla_{\mathbf{x}} f(\mathbf{x}, \theta))$ by $c(cn + k) = c^2n + ck$; in other words, a constant factor c^2 of n , plus some overhead. In the particular case of the ReenGAGE loss term (in the sparse case, for simplicity), x corresponds to the goal g and f corresponds to $Q_{\theta}(s, a, g)$, while h corresponds to $\|\nabla_g f(g, \theta) - \nabla_g \gamma Q_{\theta'}(s', \pi_{\phi'}(s', g), g)\|_2^2$. If the Q function requires n operations to evaluate and the policy π requires m operations, then the overall computational cost can therefore be upper-bounded by $c(cn + c(n + m) + k) = 2c^2n + c^2m + ck$, where k is the (trivial) amount of computation required to compute the norm. This is therefore a constant factor of the time needed to compute the value of the Q function and its target (with some trivial overhead due to the norm computation).

(Etmann 2019) provides explicit algorithms for computations of gradients of forms similar to the form $\nabla_{\theta} h(\nabla_{\mathbf{x}} f(\mathbf{x}, \theta))$ discussed above and confirms the constant-factor increase in computational complexity; in particular, $h(\cdot)$ corresponds to $p(\cdot)$ in Equation 10 of (Etmann 2019). (Note that (Etmann 2019)’s analysis is somewhat more general, allowing for a vector-valued f , with $p(\cdot)$ a function of a Jacobian-vector product of f rather than simply the gradient).

To provide empirical support for this, we provide runtime comparisons for training with and without the ReenGAGE loss term, for the experiments in the main text. Note that we are comparing total runtimes, so these times include the environment simulation; however this should be relatively minor for all experiments (because the environments themselves are relatively simple) except possibly the robotics experiments.

For ContinuousSeek and Multi-ReenGAGE experiments, all tests were run on a single GPU. We used a pool of shared computational resource, so the GPU models may have varied between runs; GPUs possibly used were NVIDIA RTX A4000, RTX A5000, and RTX A6000 models. (This adds some uncertainty to our runtime comparisons.) Robotics experiments were run on 20 CPUs each, as described by (Plappert et al. 2018).

Runtime comparison results are given in Table 5. For ContinuousSeek, we consider only experiments with $d = 20$, batch size = 256; for others, we include all experiments shown in the main text. For runtimes with ReenGAGE, we average over all values of α included in the main text. In general, runtime increases ranged from 34%-60%.

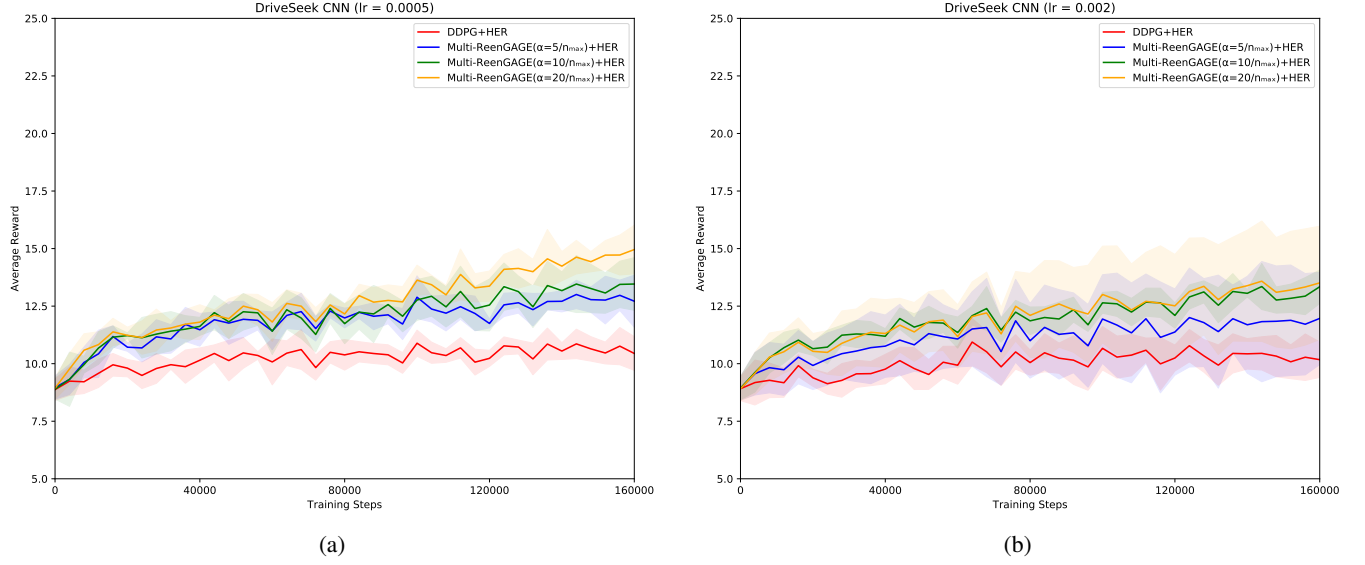


Figure 20: DriveSeek with CNN architecture with varied values of the learning rate.

Environment	Runtime without ReenGAGE (s)	Runtime with ReenGAGE (s)	Mean Percent Increase
ContinuousSeek	2549 ± 203	3591 ± 248	40.9%
HandReach	2533 ± 2	3395 ± 208	34.0%
DriveSeek	8931 ± 2037	12467 ± 1354	39.6%
NoisySeek	7385 ± 503	11814 ± 1519	60.0%

Table 5: Effect of ReenGAGE on runtimes. Error values shown are standard deviations over all runs.