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:

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

The optimal policy π then takes the form:

$$a^* = \frac{U^T g}{\|g\|_2} \tag{24}$$

(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

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

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:

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

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

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

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:

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

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 $s^1 \neq 0$. In this case, the immediate reward is always zero, and we also know that $s^{1\prime} = \mathbf{0}$ so the "standard" TD update is:

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

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 $s^1 \neq 0$. For all of these, the immediate reward is zero, so we can use the gradient-based update for all of them.

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

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:

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

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:

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

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

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

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*:

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

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

- Soft target for *gradient loss only*:

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

  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*:

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

- 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.

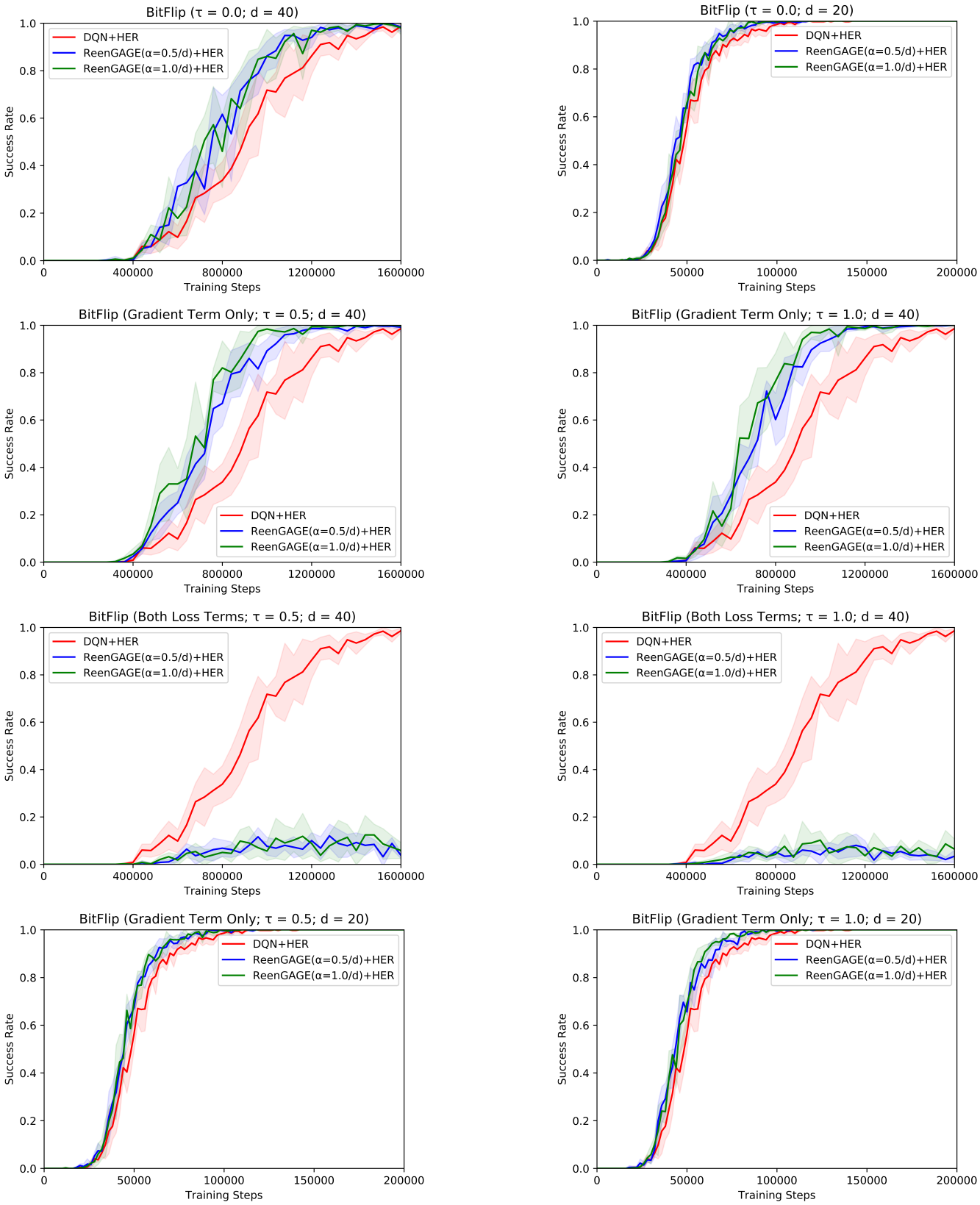


Figure 7: Results for the Bit-Flipping environment.

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.

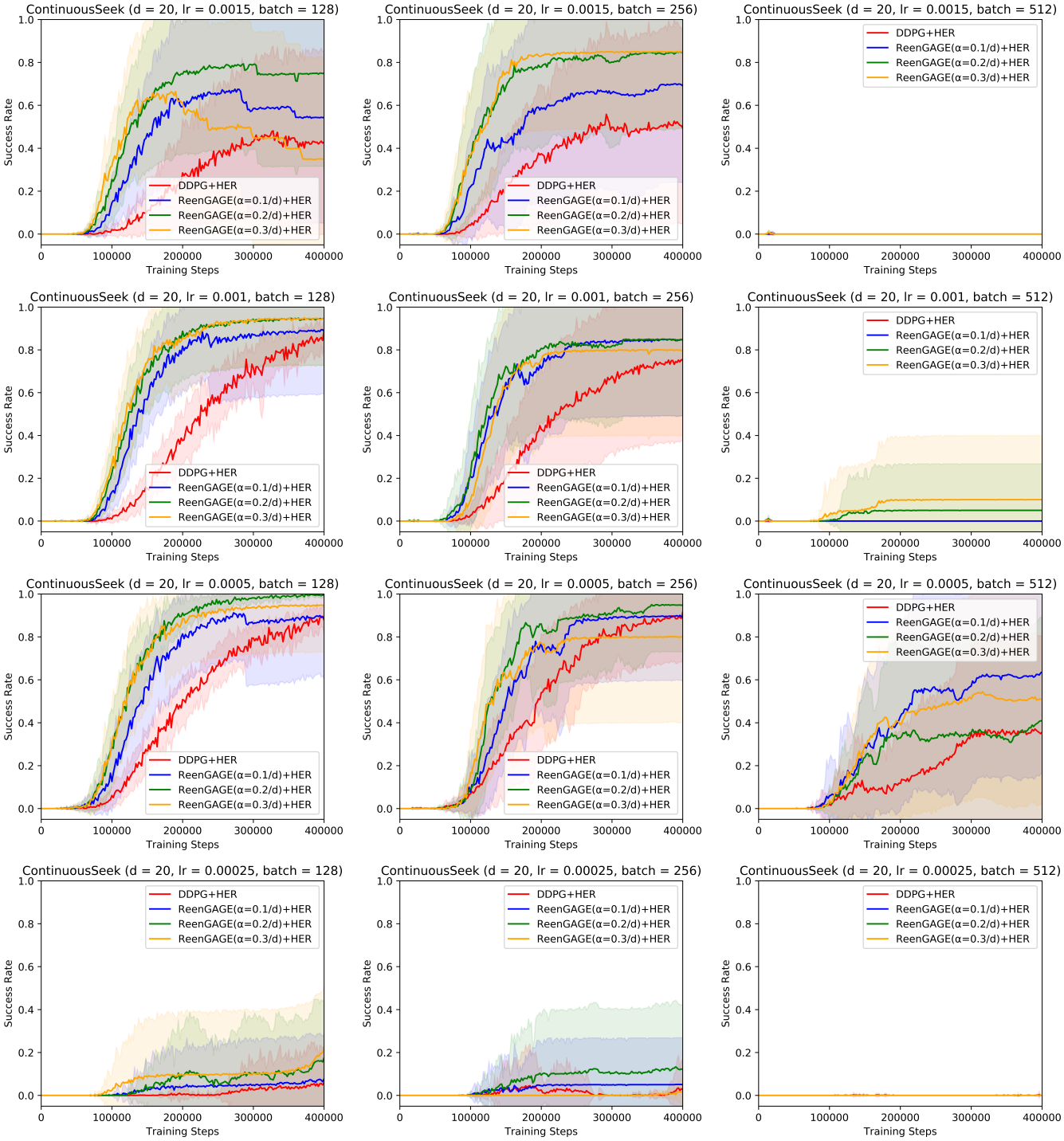


Figure 8: Complete ContinuousSeek Results for $d = 20$.

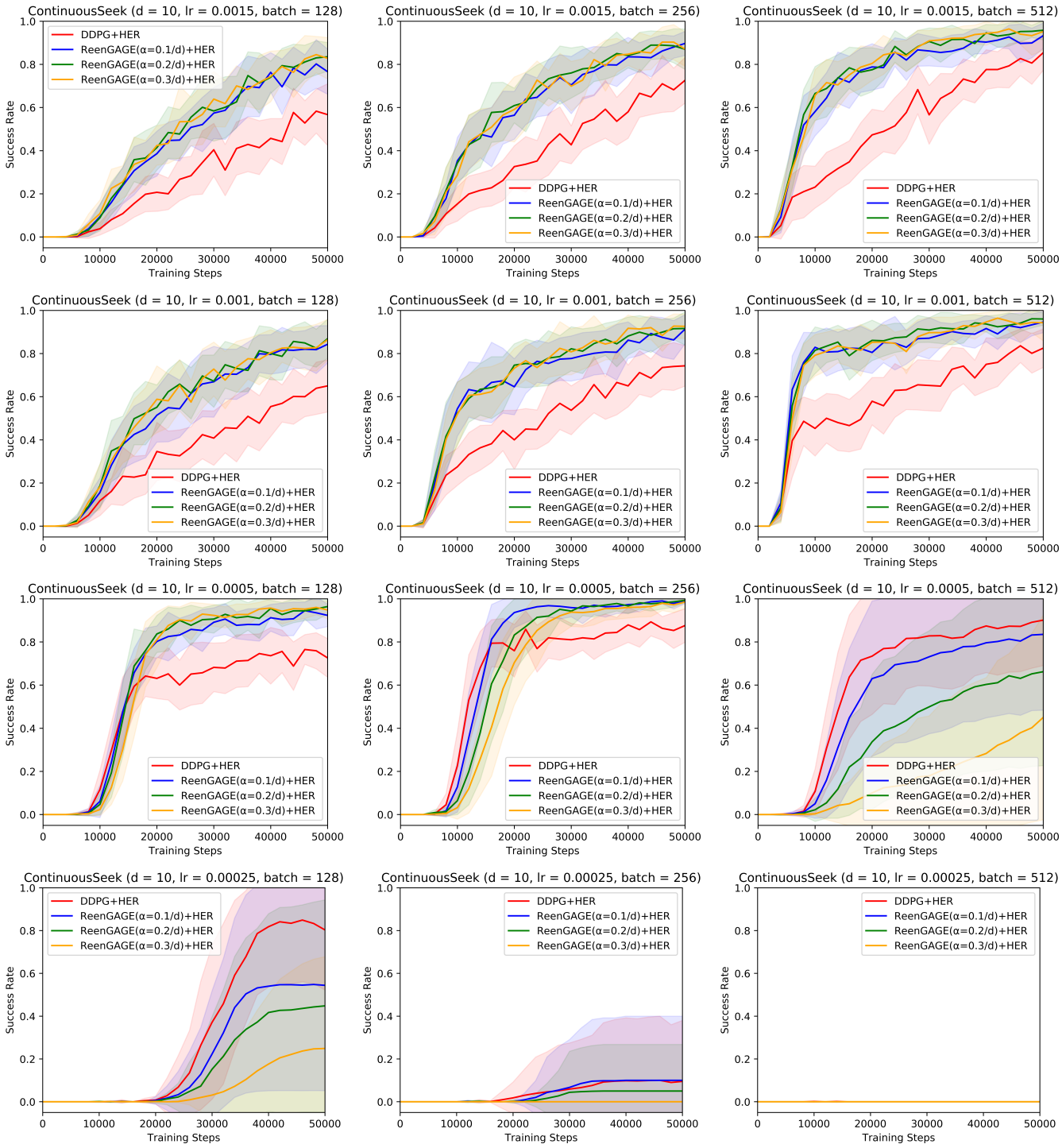


Figure 9: Complete ContinuousSeek Results for $d = 10$.

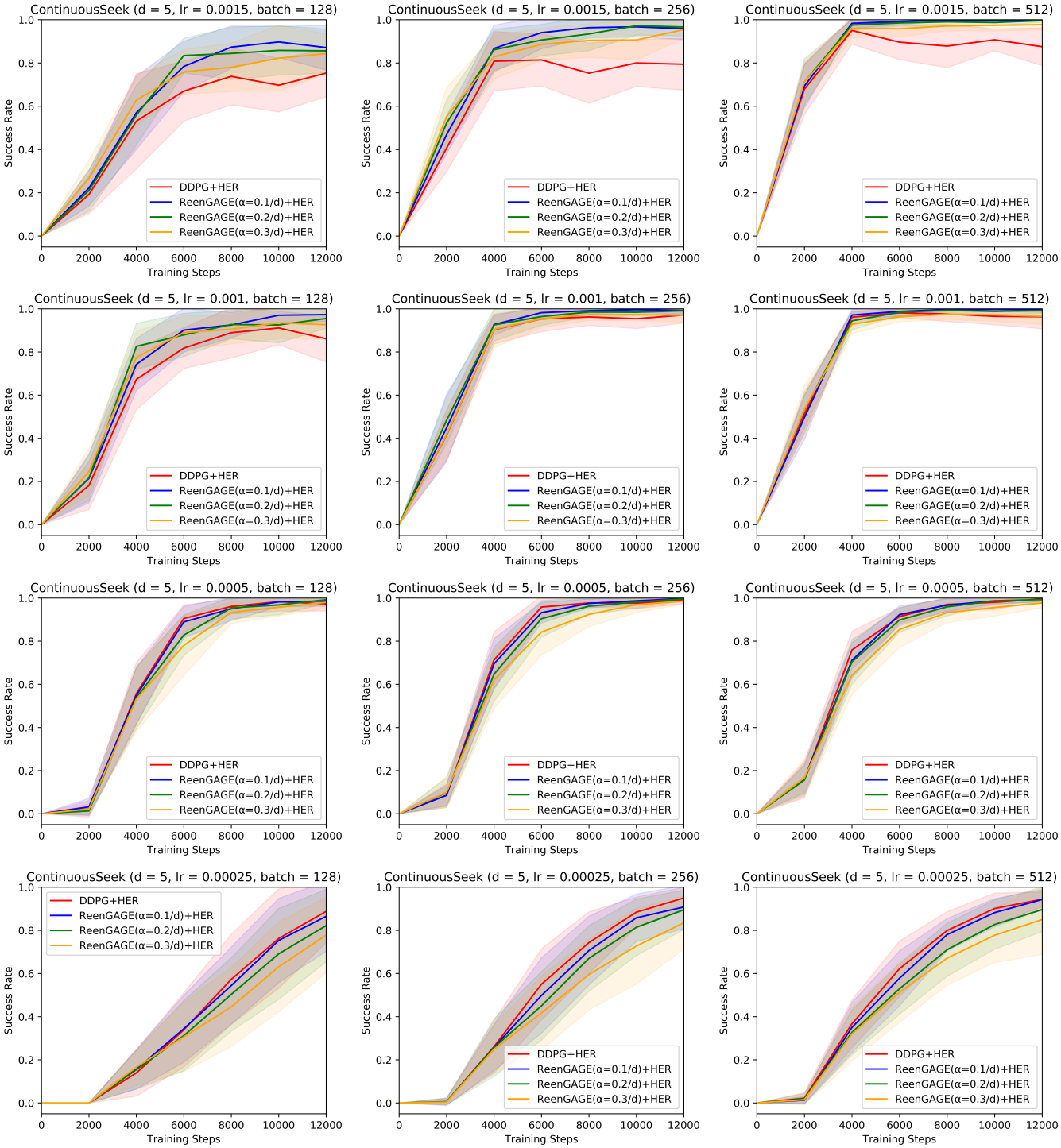


Figure 10: Complete ContinuousSeek Results for $d = 5$.

Additional Results for Robotics Experiments

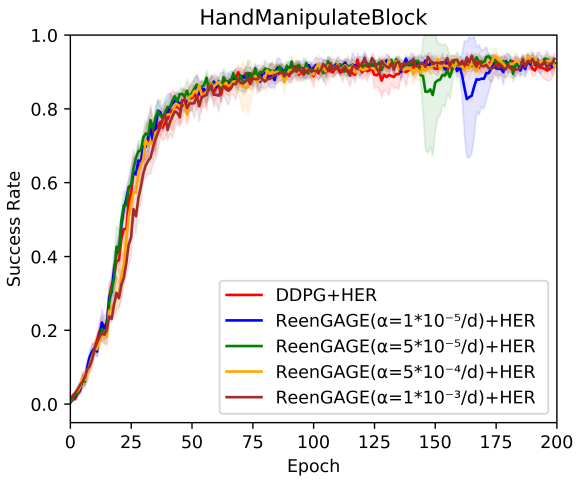
In Figure 11, we provide additional results for the robotics experiments. In Figure 11-(a) and (b), we give results for the **HandManipulateBlock** environment: as mentioned in the text, we did not see a significant advantage to using ReenGAGE for this environment.

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.

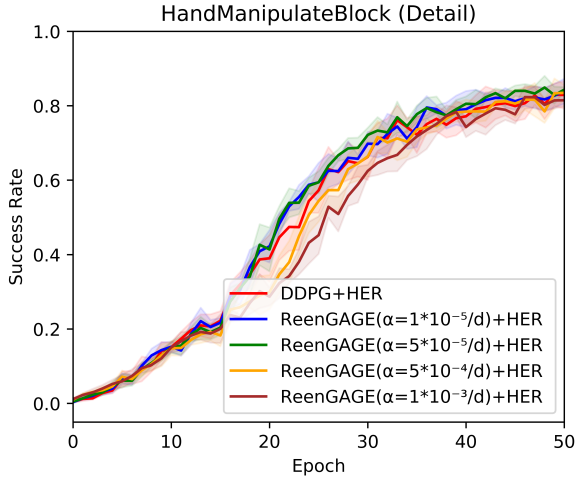
In order to handle this, we attempted normalizing each dimension of the ReenGAGE loss term. We accomplish this by multiplying the MSE loss for each coordinate i by σ_i^2, where σ_i is the running average standard deviation of dimension i of the goals. ((Plappert et al. 2018)'s implementation already 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)).

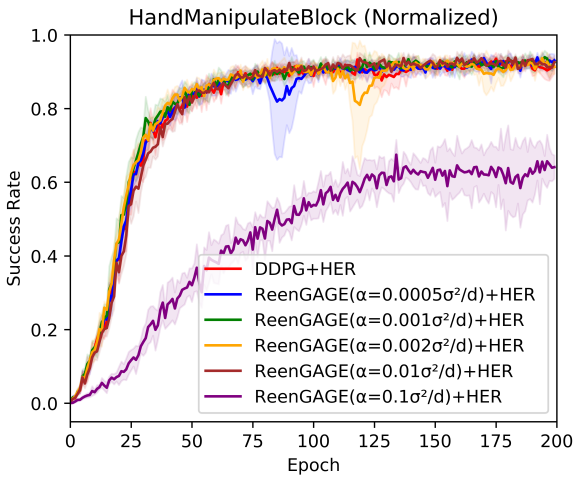
Finally, in Figure 11-(f), we show some additional experiments on HandReach. Specifically, we show results using a single random seed for a wider range of α than shown in the main text: this was performed as a first pass to find the appropriate range of the hyperparameter α to use for the complete experiments.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



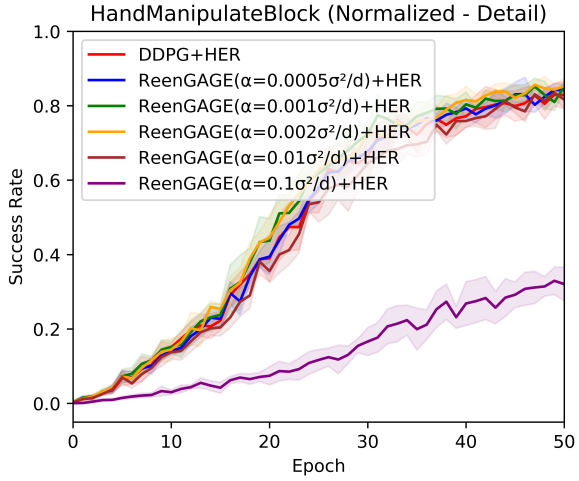
(a)



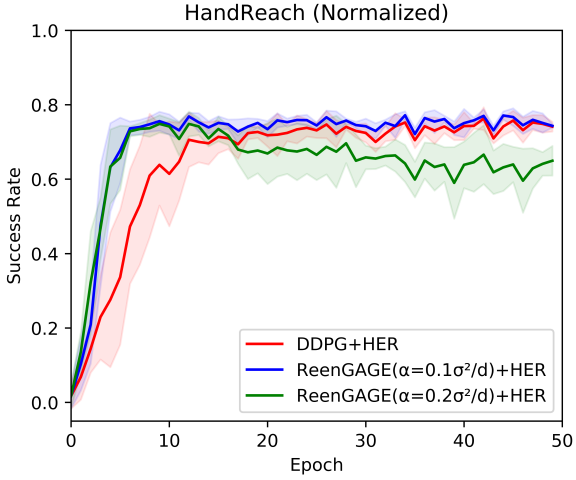
(b)



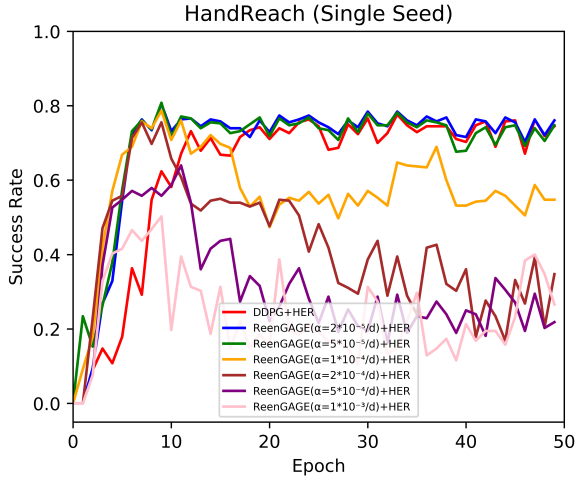
(c)



(d)



(e)



(f)

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

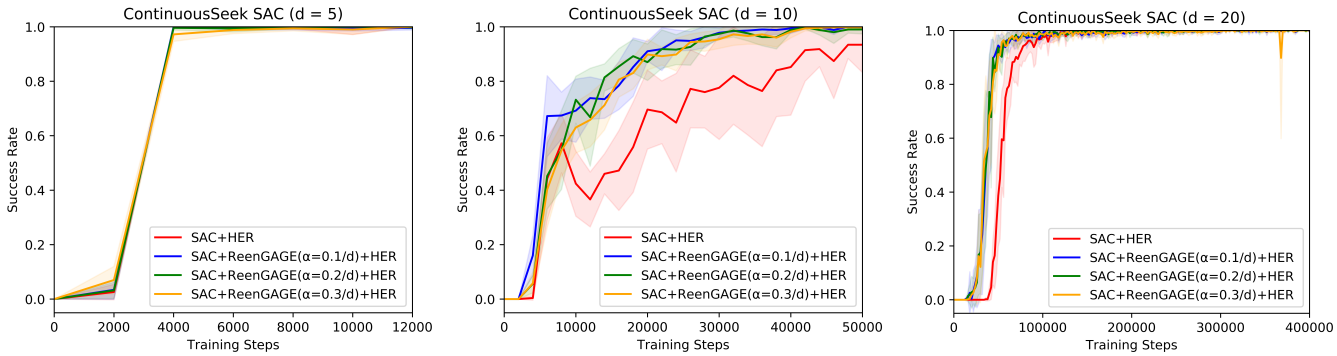


Figure 12: ContinuousSeek results with SAC. Lines show the mean and standard deviation over 10 random seeds (kept the same for all experiments.) The Y-axis represents the success rate, defined as the fraction of test episodes for which the goal is ever reached.

	SAC+HER		SAC+ReenGAGE(α=.1)+HER		SAC+ReenGAGE(α=.2)+HER		SAC+ReenGAGE(α=.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 polices, 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.

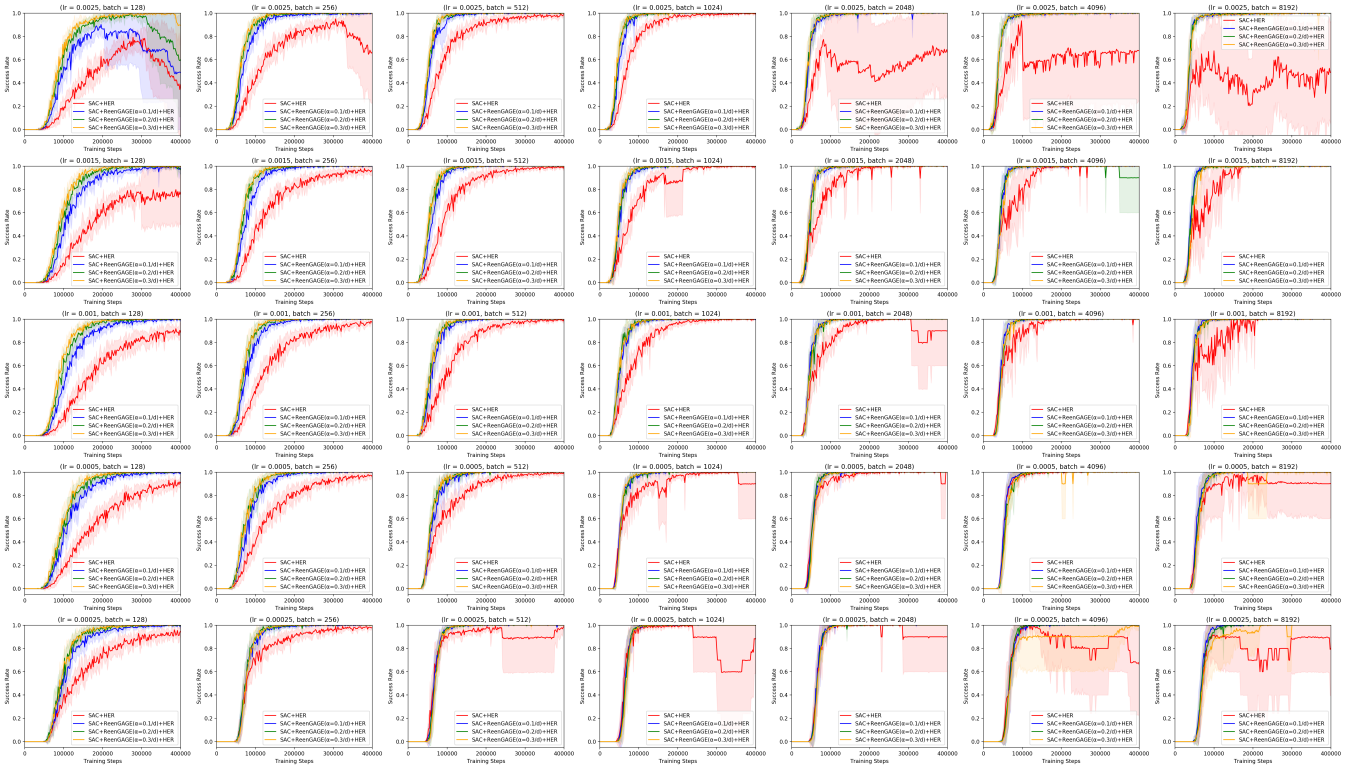


Figure 13: Complete SAC ContinuousSeek results for $d = 20$.

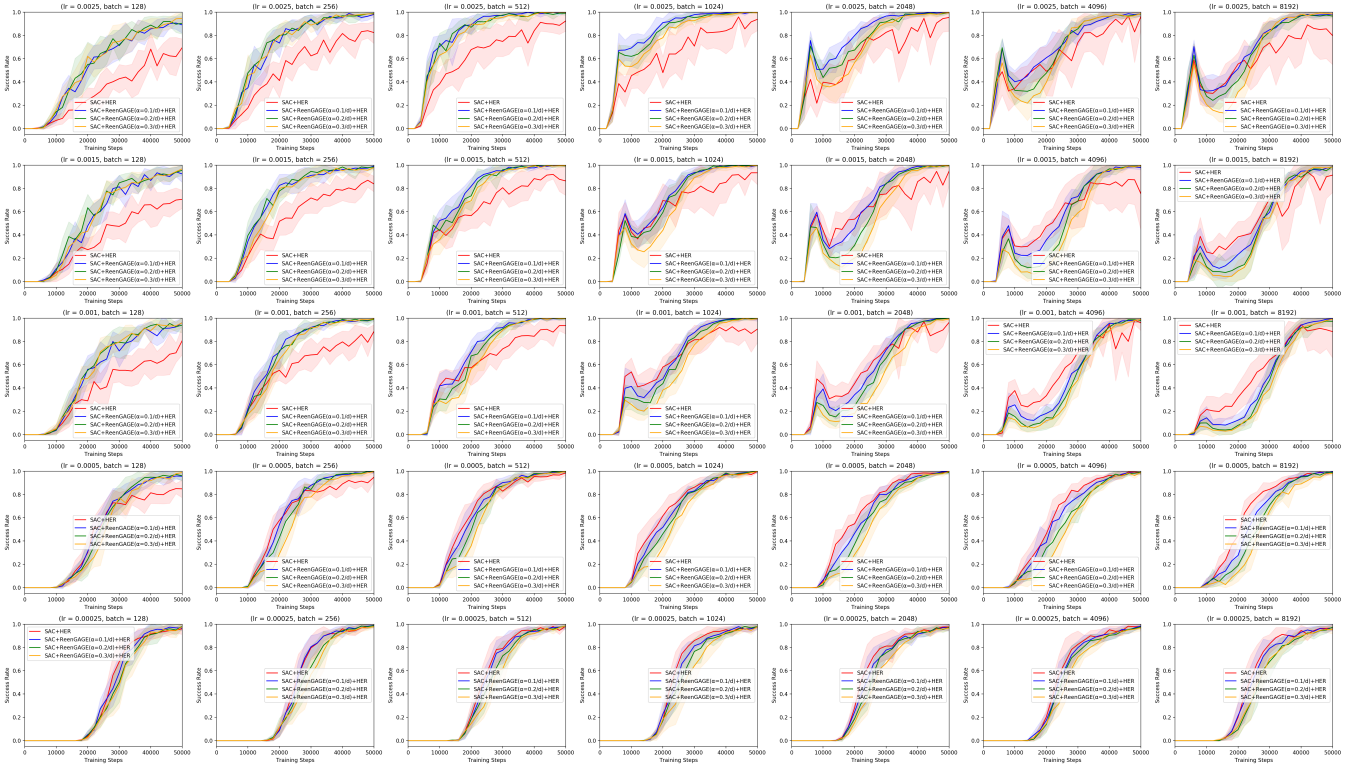


Figure 14: Complete SAC ContinuousSeek results for $d = 10$.

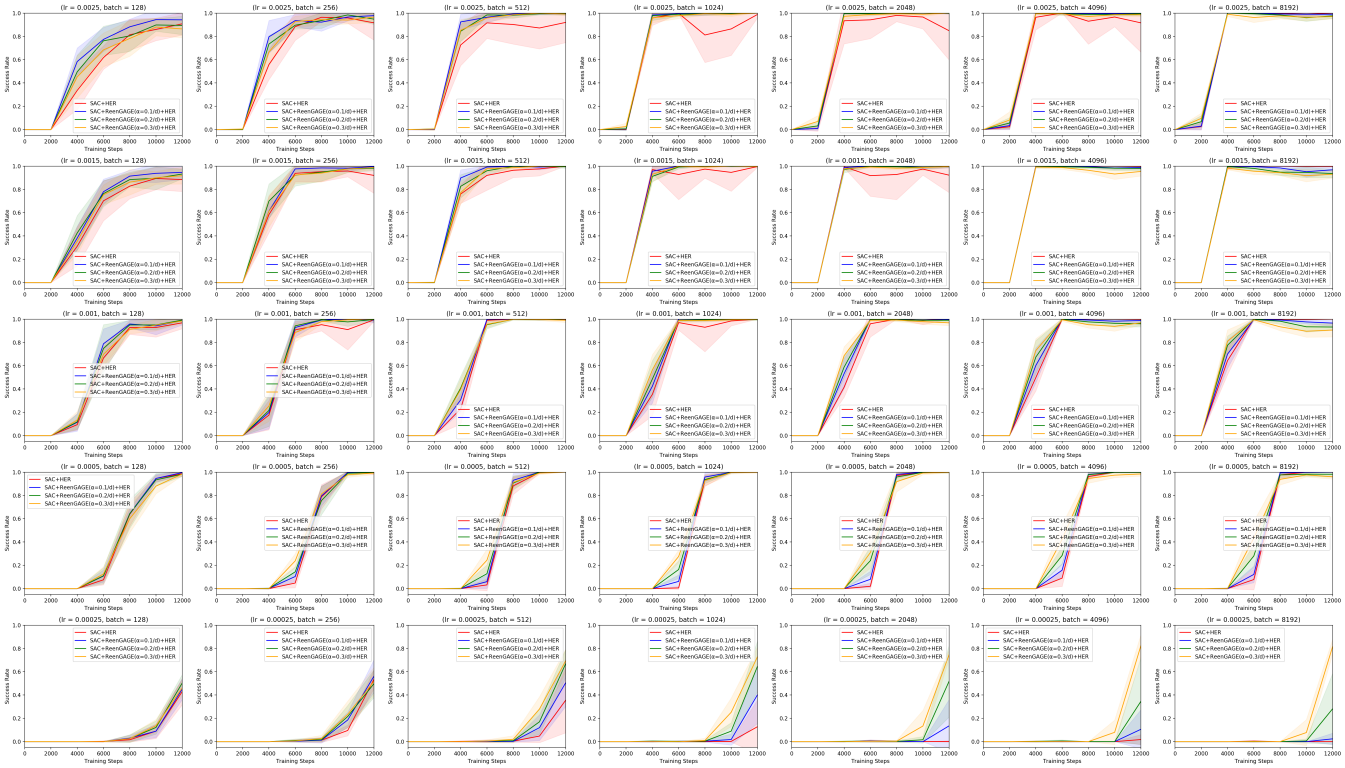


Figure 15: Complete SAC ContinuousSeek results for $d = 5$.

Multi-ReenGAGE 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 $g = \{g_1, ..., g_n\}$. Specifically, we let $g = \{g_1, ..., g_{n_{\max}}\}$, where $\{g_{n+1}, ..., g_{n_{\max}}\}$ are set to a dummy value (practically, $\mathbf{0}$), and the gate variables $\{b_{n+1}, ..., b_{n_{\max}}\}$ are all set to zero. Then:

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

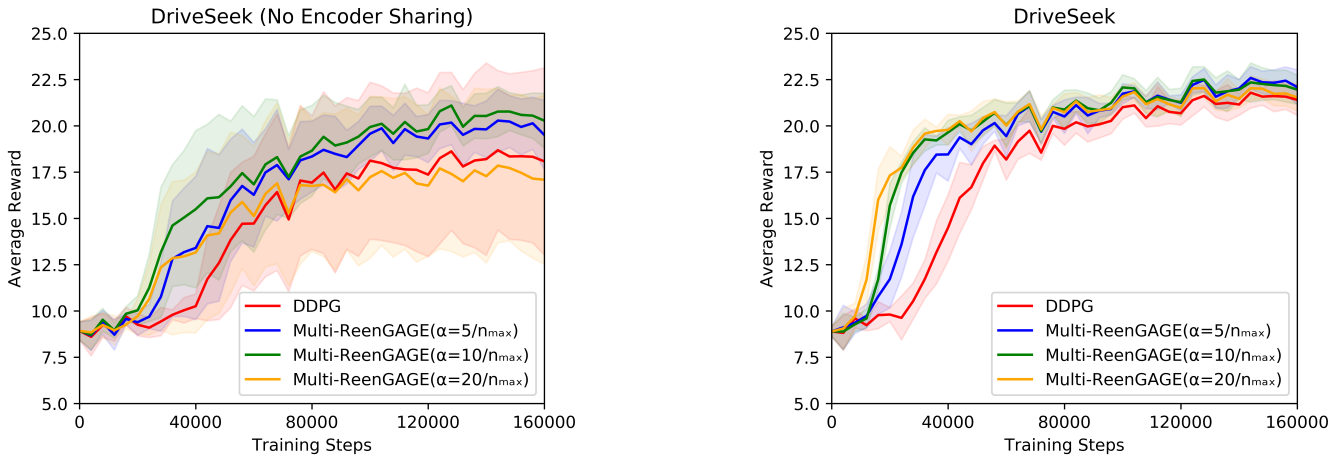
However, while this is equal to the intended form of the Q-function without the dummy inputs, the gradient with respect to the full vector b differs from the intended form. Specifically, note that:

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

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

$$Q^{\text{head}}_{\theta^{\text{h.}}}(s, a, \sum_{i=1}^{n_{\max}}[b_i^2 Q^{\text{encoder}}_{\theta^{\text{e.}}}(s, g_i)]). \tag{36}$$

(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:



(a) (b)

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

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

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:

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

So that the gradient is:

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

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:

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

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 $\{1, ..., 200\}$; 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 $s_{\text{pos.}}$, the *rounded* version of $s_{\text{pos.}}$ (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 $\{1, ..., 200\}$. 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 $\alpha_1, ..., \alpha_K = 1$, 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

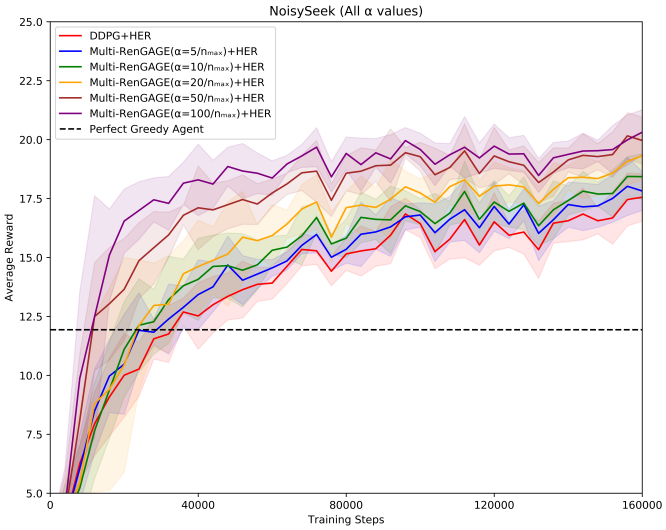


Figure 17: Results for NoisySeek Environment including additional values of α.

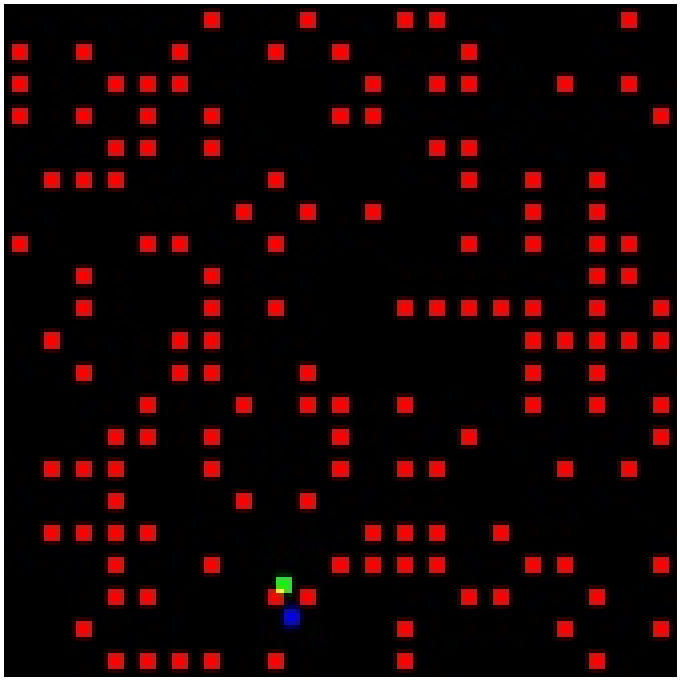


Figure 18: Example input to CNN architecture for DriveSeek. We use one channel (red) to represent the goals (note that these pixels directly correspond to the differentiable indicator variables b_i^2 for each possible goal), another channel (green) to represent the current position, and a third channel (blue) to represent the next position if the action a is zero: in other words, it indicates $\mathbf{s}_{\text{vel.}}$. 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.

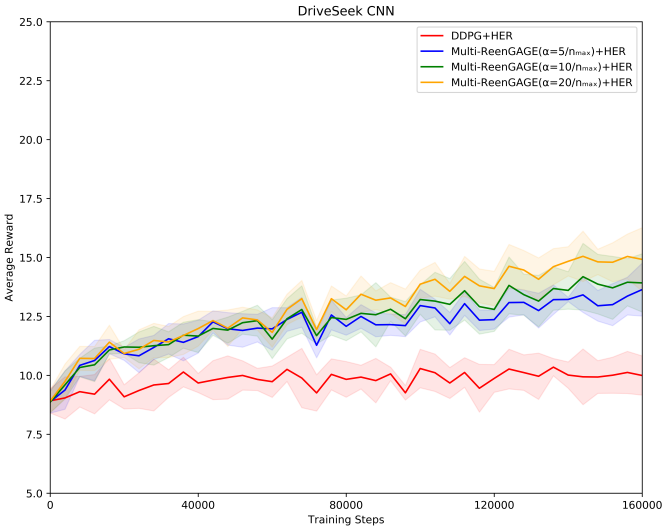


Figure 19: Results for DriveSeek with CNN architecture.

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.

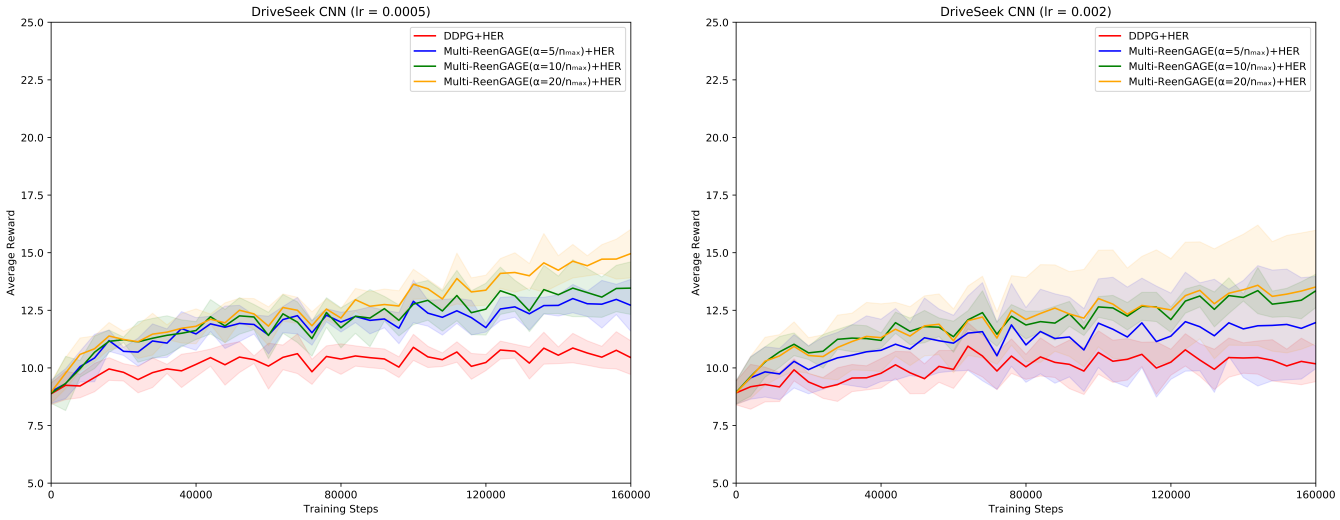
For an intuition as to why this is the case, recall the well-known fact that for a differentiable scalar-output function $f(\mathbf{x}, \theta)$ represented by some computational graph which requires n operations to compute, standard backpropagation can compute the gradient $\nabla_{\mathbf{x}} f(\mathbf{x}, \theta)$ (or the gradient $\nabla_{\theta} f(\mathbf{x}, \theta)$) using only cn operations, for a constant c. However, now note that $\nabla_x f(\mathbf{x}, \theta)$ can *itself* be though 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 $c(cn + k) = c^2 n + ck$; 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 $c(cn + c(n + m) + k) = 2c^2 n + c^2 m + ck$, 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%.



(a) (b)

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.