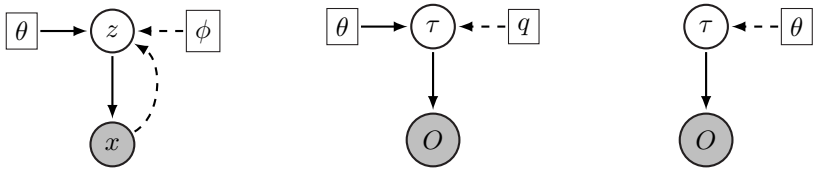
A Details on Graphical Models for Reinforcement Learning

In this section, we review details of the RL *as inference* framework [53, 21] and highlight its critical differences from *Variational* RL.

The graphical model for RL *as inference* is shown in Figure 7(c). The framework also assumes a trajectory variable $\tau \equiv (s_t, a_t)_{t=0}^{T-1}$ which encompasses the state and action sequences. Conditional on the trajectory variable τ, the optimality variable is defined as $p(O = 1 \mid \tau) \propto \exp(\sum_{t=0}^{T-1} r(s_t, a_t)/\alpha)$ for $\alpha > 0$. Under this framework, the trajectory variable has a prior $a_t \sim p(\cdot)$ where $p(\cdot)$ is usually set to be a uniform distribution over the action space $\mathcal{A}$.

The policy parameter θ comes into play with the inference model. The framework asks the question: what is the posterior distribution $p(\tau \mid O = 1)$. To approximate this intractable posterior distribution, consider the variational distribution $q(\tau) \coloneqq \Pi_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$. Searching for the best approximate posterior by minimizing the KL-divergence $\mathbb{KL}[q(\tau) \parallel p(\tau \mid O = 1)]$, it can be shown that this is equivalent to maximum-entropy RL [54, 55, 56]. It is important to note that RL *as inference* does not contain trainable parameters for the generative model.

Contrasting this to *Variational* RL and the graphical model for goal-conditioned RL in Figure 2: the policy dependent parameter θ is part of a generative model. The variational distribution $q(\tau, g)$, defined separately from θ, is the inference model. In such cases, the variational distribution $q(\tau, g)$ is an auxiliary distribution which aids in the optimization of θ by performing partial E-steps.



(a) Probabilistic inference **(b)** Variational RL **(c)** RL as inference

Figure 7: Plot (c) shows the graphical model for RL *as inference* [53, 21]. Solid lines represent generative models and dashed lines represent inference models. Circles represent random variables and squares represent parameters. Filled circles represent observed random variables. This graphical model does not have trainable parameters for the generative model. The policy dependent parameter θ is in the inference model.

B Details on proof

B.1 Proof of Proposition Proposition 1

The proof follows from the observation that $p(O = 1) = \mathbb{E}_{g \sim p(\cdot), \pi}[p(O = 1 \mid \tau, g)] = J(\pi_\theta)$, and taking the log does not change the optimal solution.

B.2 Proof of Theorem 1

Recall that we have a one-step MDP setup where $\mathcal{A} = \mathcal{G}$ and $|\mathcal{A}| = k$. The policy $\pi(a \mid s, g) = \text{softmax}(L_{a,g})$ is parameterized by logits $L_{a,g}$. When the policy is initialized randomly, we have $L_{a,g} \equiv L$ for some L and $\pi(a \mid s, g) = 1/k$ for all a, g. Assume also $p(g) = 1/k, \forall g$.

The one-sample REINFORCE gradient estimator for the component $L_{a,g}$ is $\eta_{a,g} = r(s, b, g') \log_{L_{a,g}} \pi(b \mid s, g')$ with $g' \sim p(\cdot)$ and $b \sim \pi(\cdot \mid s, g')$. Further, we can show

$$\mathbb{E}[\eta_{a,g}] = \frac{1}{k^2}\delta_{a,g} - \frac{1}{k^3}, \ \mathbb{V}[\eta_{a,g}] = (\frac{1}{k^2} + \frac{2}{k^5} - \frac{2}{k^3} - \frac{1}{k^4})\delta_{a,g} + \frac{1}{k^4} - \frac{1}{k^6},$$

where $\delta_{a,b}$ are dirac-delta functions, which mean $\delta_{a,b} = 1$ if $a = b$ and $\delta_{a,b} = 0$ otherwise. Taking the ratio, we have the squared relative error (note that the estimator is unbiased and MSE consists purely of the variance)

$$\frac{\text{MSE}[\eta_{a,g}]}{\mathbb{E}[\eta_{a,g}]^2} = \frac{(k^4 + o(k^4))\delta_{a,g} + (k^2 + o(k^2))}{(k^2 + o(k^2))\delta_{a,g} + 1}$$

The expression takes different forms based on the delta-function $\delta_{a,g}$. However, in either case (either $\delta_{a,g} = 1$ or $\delta_{a,g} = 0$), it is clear that $\frac{\text{MSE}[\eta_{a,g}]}{\mathbb{E}[\eta_{a,g}]^2} = k^2(1 + o(1))$, which directly reduces to the result of the theorem.

Comment on the control variates. We also briefly study the effect of control variates. Let X, Y be two random variables and assume $\mathbb{E}[Y] = 0$. Then compare the variance of $\mathbb{V}[X]$ and $\mathbb{V}[X + \alpha Y]$ where α is chosen optimally to minimize the variance of the second estimator. It can be shown that with the best α^*, the ratio of variance reduction is $(\mathbb{V}[X] - \mathbb{V}[X + \alpha^* Y])/\mathbb{V}[X] = \rho^2 \coloneqq \text{Cov}^2[X, Y]/\mathbb{V}[X]\mathbb{V}[Y]$. Consider the state-based control variate for the REINFORCE gradient estimator, in this case $-\alpha \cdot \nabla_{L_{a,g}} \pi(b \mid s, g')$ where α is chosen to minimize the variance of the following aggregate estimator

$$\eta_{a,g}(\alpha) = r(s, b, g') \log_{L_{a,g}} \pi(b \mid s, g') - \alpha \log_{L_{a,g}} \pi(b \mid s, g').$$

Note that in practice, α is chosen to be state-dependent for REINFORCE gradient estimator of general MDPs and is set to be the value function $\alpha \coloneqq V^\pi(s)$. Such a choice is not optimal [57] but is conveniently adopted in practice. Here, we consider an optimal α^* for the one-step MDP. The central quantity is the squared correlation ρ^2 between $r(s, b, g') \log_{L_{a,g}} \pi(b \mid s, g')$ and $\log_{L_{a,g}} \pi(b \mid s, g')$. With similar computations as above, it can be shown that $\rho^2 \approx 1$ for $b \neq g'$ and $\rho^2 \approx \frac{1}{k^2}$ otherwise. This implies that for k out of k^2 logits parameters, the variance reduction is significant; yet for the rest of the $k^2 - k$ parameters, the variance reduction is negligible. Overall, the analysis reflects that conventional control variantes do not address the issue of sup-linear growth of relative errors as a result of *sparse gradients*.

B.3 Proof of Theorem 2

The key to the proof is the same as the proof of theorem 1: we analytically compute $\mathbb{E}[\eta_{a,g}]$ and $\mathbb{V}[\eta_{a,g}]$. We can show

$$\mathbb{E}[\eta_{a,g}] = \frac{1}{k^2}\delta_{a,g} + o(\frac{1}{k^2}),\ \mathbb{V}[\eta_{a,g}] = \frac{1}{k^3}\delta_{a,g} + o(\frac{1}{k^3}),$$

More specifically, it is possible to show that the normalized one-step REINFORCE gradient estimator $\eta^h_{a,g} = r(s, b, g')\nabla_{L_{a,g}} \log \pi(b \mid s, g')/k$ with $(b, g') \sim q_h(\tau, g)$ has the following property

$$\frac{\text{MSE}[\eta_{a,g}]}{\mathbb{E}[\eta_{a,g}]^2} = \frac{(k^3 + o(k^3))\delta_{a,g} + (k + o(k))}{(k^2 + o(k))\delta_{a,g} + 1}.$$

The above equality implies the result of the theorem. Indeed, the above implies regardless of whether $\delta_{a,g} = 0$ or $\delta_{a,g} = 1$: $\frac{\text{MSE}[\eta_{a,g}]}{\mathbb{E}[\eta_{a,g}]^2} = k(1 + o(1))$.

Remark. Contrast this with the result from Theorem 1, where the result is $k^2(1 + o(1))$. The main difference stems from the variance: in Theorem 1 the variance is of order $\frac{1}{k^2}$ while here the variance is of order $\frac{1}{k^3}$. The variance reduction leads to significant improvements of the sample efficiency of the estimation.

B.4 Proof of Theorem 3

Without loss of generality we assume $p(g)$ is a uniform measure, i.e. $p(g) = 1/|\mathcal{G}|$. If not, we could always find a transformation $g = f(g')$ such that g' takes a uniform measure [6] and treat g' as the goal to condition on.

Let $|\mathcal{G}| < \infty$ and recall $\text{supp}(\tilde{p}(g))$ to be the support of $\tilde{p}(g)$. The uniform distribution assumption deduces that $|\text{supp}(\tilde{p}(g))| = \int_{g \in \text{supp}(\tilde{p}(g))} dg$. At iteration t, under tabular representation, the M-step update implies that π_θ learns the optimal policy for all g that could be sampled from $q(\tau, g)$, whhich effectively corresponds to the support of $\tilde{p}(g)$. Formally, this implies $\mathbb{E}_{p(\tau \mid \theta_{t+1}, g)}[R(\tau, g)] = 1$ for $\forall g \in \text{supp}(\tilde{p}_t(g))$. This further implies

$$J(\pi_{\theta_{t+1}}) \coloneqq \int \mathbb{E}_{p(\tau \mid \theta_{t+1}, g)}[R(\tau, g)]p(g)dg \geq \int_{g \in \text{supp}(\tilde{p}_t(g))} 1 \cdot p(g)dg = |\text{supp}(\tilde{p}_t(g))|/|\mathcal{G}|.$$

C Additional Experiment Results

C.1 Details on Benchmark tasks

All reaching tasks are built with physics simulation engine MuJoCo [58]. We build customized point mass environment; the Reacher and Fetch environment is partly based on OpenAI gym environment [59]; the Sawyer environment is based on the multiworld open source code https://github.com/vitchyr/multiworld.

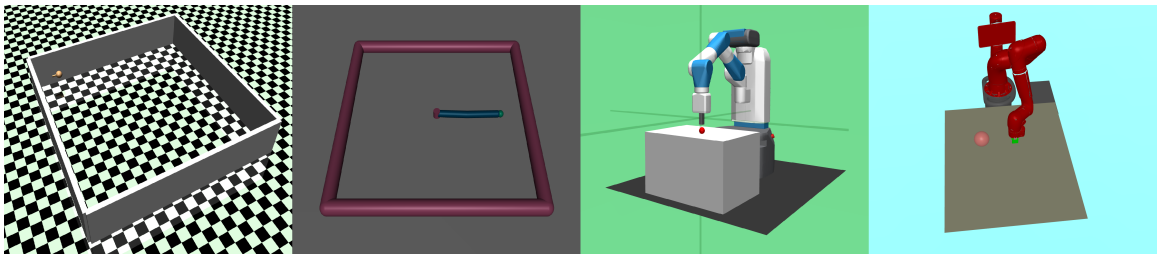


Figure 8: Illustration of tasks. From left to right: Point mass, Reacher, Fetch robot and Sawyer Robot. On the right is the image-based input for Fetch robot. For additional information on the tasks, see Appendix C.

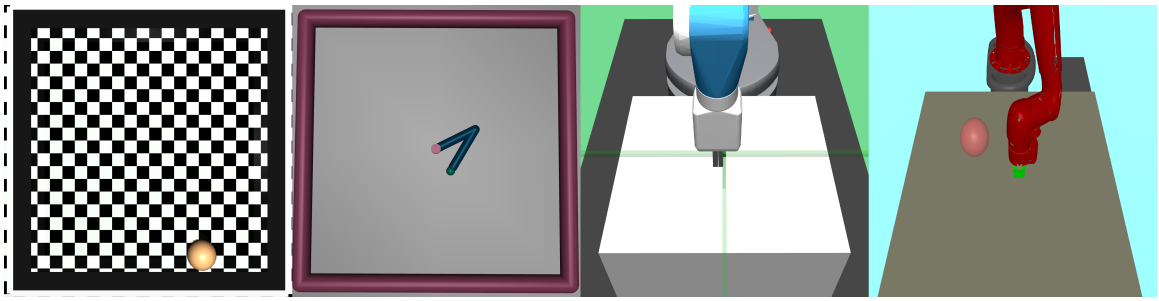


Figure 9: Illustration of image-based inputs for different reaching tasks in the main paper. Images are down-sampled to be of size $w \times w \times 3$ as inputs, where $w \in \{48, 84\}$.

All simulation tasks below have a maximum episode length of $T = 50$. The episode terminates early if the goal is achieved at a certain step. The sparse binary reward function is $r(s, a, g) = \mathbb{I}[\text{success}]$, which indicates the success of the transitioned state $s' = f(s, a)$[1]. Below we describe in details the setup of each task, in particular the success criterion.

- **Point mass [32].** The objective is to navigate a point mass through a 2-D room with obstacles to the target location. $|\mathcal{S}| = 4, |\mathcal{G}| = 2$ and $|\mathcal{A}| = 2$. The goals $g \in \mathbb{R}^2$ are specified as a 2-D point on the plane. Included in the state s are the 2-D coordinates of the point mass, denoted as $s_{xy} \in \mathbb{R}^2$. The success is defined as $d(z(s_{xy}), z(g)) \leq d_0$ where $d(\cdot, \cdot)$ is the Euclidean distance, $z(g)$ is a element-wise normalization function $z(x) \coloneqq (x - x_{\min})/(x_{\max} - x_{\min})$ where $x_{\max}, x_{\min}$ are the boundaries of the wall. The normalized threshold is $d_0 = 0.02 \cdot \sqrt{2}$.

- **Reacher [59].** The objective is to move via joint motors the finger tip of a 2-D Reacher robot to reach a target goal location. $|\mathcal{S}| = 11, |\mathcal{G}| = 2$ and $|\mathcal{A}| = 2$. As with the above point mass environment, the goals $g \in \mathbb{R}^2$ are locations of a point at the 2-D plane. Included in the state s are 2-D coordinates of the finger tip location of the Reacher robot s_{xy}. The success criterion is defined identically as the point mass environment.

- **Fetch robot [59, 2].** The objective is to move via position controls the end effector of a fetch robot, to reach a target location in the 3-D space. $|\mathcal{S}| = 10, |\mathcal{G}| = 3$ and $|\mathcal{A}| = 3$. This task belongs to the standard environment in OpenAI gym [59] and we leave the details to the code base and [2].

[1] For such simulation environments, the transition $s' \sim p(\cdot \mid s, a)$ is deterministic so we equivalently write $s' = f(s, a)$ for some deterministic function f.

- **Sawyer robot [49, 50].** The objective is to move via motor controls of the end effector of a sawyer robot, to reach a target location in the 3-D space. $|\mathcal{S}| = |\mathcal{G}| = |\mathcal{A}| = 3$. This task belongs to the multiworld code base.

Details on image inputs. For the customized point mass and Reacher environments, the image inputs are taken by cameras which look vertically down at the systems For the Fetch robot and Sawyer robot environment, the images are taken by cameras mounted to the robotic systems. See Figure 9 for an illustration of the image inputs.

C.2 Details on Algorithms and Hyper-parameters

Hindsight Expectation Maximization. For hEM on domains with discrete actions, the policy is a categorical distribution $\pi_\theta(a \mid s, g) = \text{Cat}(\phi_\theta(s, g))$ with parameterized logits $\phi_\theta(s, g)$; on domains with continuous actions, the policy network is a state-goal conditional Gaussian distribution $\pi_\theta(a \mid s, g) = \mathcal{N}(\mu_\theta(s, g), \sigma^2)$ with a parameterized mean $\mu_\theta(s, g)$ and a global standard deviation σ^2. The mean is takes the concatenated vector $[x, g]$ as inputs, has 5 hidden layers each with 5 hidden units interleaved with relu(x) non-linear activation functions, and outputs a vector $\mu_\theta(s, g) \in \mathbb{R}^{|\mathcal{A}|}$.

hEM alternates between data collection using the policy and policy optimization with the EM-algorithms. During data collection, the output action is perturbed by a Gaussian noise $a' = \mathcal{N}(0, \sigma_a^2) + a, a \sim \pi_\theta(\cdot \mid s, g)$ where the scale is $\sigma_a = 0.5$. Note that injecting noise to actions is a common practice in off-policy RL algorithms to ensure sufficient exploration [3, 4]; for tasks with a discrete action space, the agent samples actions $a \sim \pi_\theta(\cdot \mid s, g)$ with probability $1 - \epsilon$ and uniformly random with probability $\epsilon \in [0.2, 0.5]$. The baseline hEM collects data with $N = 30$ parallel MPI actors, each with $k = 20$ trajectories. When sampling the hindsight goal given trajectories, we adopt the *future* strategy specified in HER [2]: in particular, at state s, future achieved goals are uniformly sampled at trainig time as $q_h(\tau, g)$. All parameters are optimized with Adam optimizer [31] with learning rate $\alpha = 10^{-3}$ and batch size $B = 64$. By default, we run $M = 30$ parallel MPI workers for data collection and training, at each iteration hEM collects $N = 20$ trajectories from the environment. For image-based reacher and Fetch robot, hEM collects $N = 80$ trajectories.

Hindsight Experience Replay. By design in [2], HER is combined with off-policy learning algorithms such as DQN or DDPG [3, 4]. We describe the details of DDPG. The algorithm maintains a Q-function $Q_\theta(s, a, g)$ parameterized similarly as a universal value function [60]: the network takes as inputs the concatenated vector $[x, a, g]$, has 5 hidden layers with $h = 256$ hidden units per layer interleaved with relu(x) non-linear activation functions, and outputs a single scalar. The policy network $\pi_\theta(s, g)$ takes the concatenated vector $[x, g]$ as inputs, has the same intermediate architecture as the Q-function network and outputs the action vector $\pi_\theta(s, g) \in \mathbb{R}^{|\mathcal{A}|}$. We take the implementation from OpenAI baseline [61], all missing hyper-parameters are the default hyper-parameters in the code base. Across all tasks, HER is run with $M = 20$ parallel MPI workers as specified in [61].

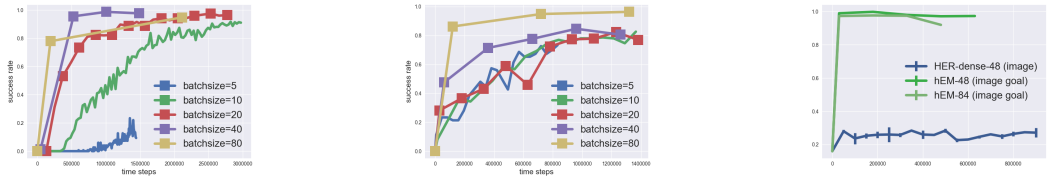
Image-based architecture. When state or goal are image-based, the Q-function network/policy network applies a convolutional network to extract features. For example, let $s, g \in \mathbb{R}^{w \times w \times 3}$ where $w \in \{48, 84\}$ be raw images, and let $f_\theta(s), f_\theta(g)$ be the features output by the convolutional network. These features are concatenated before passing through the fully-connected networks described above. The convolutional network has the following architecture: $[32, 8, 4] \to$ relu $\to [64, 4, 2] \to$ relu $\to [64, 3, 2] \to$ relu, where $[n_f, r_f, s_f]$ refers to: n_f number of feature maps, r_f feature patch dimension and s_f the stride.

C.3 Ablation study

Ablation study on the effect of N. hEM collects N trajectories at each training iteration. We vary $N \in \{5, 10, 20, 40, 80\}$ on two challenging domains: Flip bit $K = 50$ and Fetch robot (image-based) and evaluate the corresponding performance. See Figure 10. We see that in general, large N tends to lead to better performance. For example, when $N = 5$, hEM learns slowly on Flip bit; when $N = 80$, hEM generally achieves faster convergence and better asymptotic performance across both tasks. We speculate that this is partly because with large N the algorithm can have a larger coverage over goals (larger support over goals in the language of Theorem 3). With small N, the policy might converge prematurely and hence learn slowly. Similar observations have been made for

HER, where they find that the algorithm performs better with a large number of MPI workers (effectively large N).

Ablation on image-based goals. To further assess the robustness of hEM against image-based inputs, we consider Sawyer robot where both states and goals are image-based. This differs from experiments shown in Figure 5 where only states are image-based. In Figure 10(c), we see that the performance of hEM does not degrade even when goals are image-based and is roughly agnostic to the size of the image. Contrast this with HER, which does not make significant progress even when only states are image-based.



(a) Flip bit **(b) Fetch robot** **(c) Sawyer robot**

Figure 10: Ablation study. Plot (a) and (b): The effect of the data collection size N. Plot (c): The effect of image-based inputs for both states and goals. 'hEM-48' refers to image-based inputs with size $48 \times 48 \times 3$. Note also 'HER-dense' is equivalent to 'HER'$(-1/0)$ - though the reward function does not provide additional information compared to $(0/1)$, in practice this transformation makes the learning much more stable.

C.4 Comparison between hEM and HPG

We do not list HPG as a major baseline for comparison in the main paper, primarily due to a few reasons: by design, the HPG agent tackles discrete action space (see the author code base https://github.com/paulorauber/hpg), while many goal-conditioned baselines of interest [2, 49, 50] are continuous action space. Also, in [23] the author did not report comparison to traditional baselines such as HER and only report cumulative rewards instead of success rate as evaluation criterion. Here, we compare hEM with HPG on a few discrete benchmarks provided in [23] to assess their performance.

Details on HPG. The HPG is based on the author code base. [23] proposes several HPG variants with different policy gradient variance reduction techniques [25] and we take the HPG variant with the highest performance as reported in the paper. Throughout the experiment we set the learning rate to be 10^{-3} and other hyper-parameters take default values.

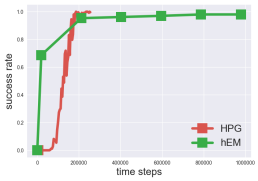
Benchmarks. We compare hEM and HPG on Flip bit $K = 25, 50$ and the four room environment. The details of the Flip bit environment could be found in the main paper. The four room environment is used as a benchmark task in [23], where the agent navigates a grid world with four rooms to reach a target location within episodic time limit. The agent has access to four actions, which moves the agent in four directions. The trial is successful only if the agent reaches the goal in time.

Results. We show results in Figure 11. For the Flip bit $K = 25$, HPG and hEM behave similarly: both algorithms reach the near-optimal performance quickly and has similar convergence speed; when the state space increases to $K = 50$, HPG does not make any progress while the performance of hEM steadily improves. Finally, for the four room environment, we see that though the performance of HPG initially increases quickly as hEM, its success rate quickly saturates to a level significantly below the asymtotpic performance of hEM. These observations show that hEM performs much more robustly and significantly better than HPG, especially in challenging environments.

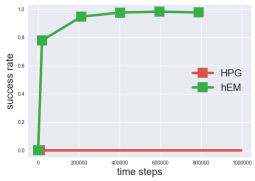
C.5 Additional Results on Baseline Fetch Tasks

Details on Fetch tasks. Fetch tasks are introduced in [2]: **Reach**, **Slide**, **Push** and **Pick and Place**. We have already evaluated on the Reach task in our prior experiments. Here, we focus on the other three experiments.

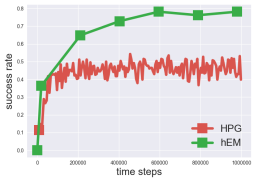
Issues with exploration. As we have alluded to in the main paper, exploration is a critical issue in goal-conditioned RL, as exemplified through the Fetch tasks. Compared to the reaching tasks we considered, Fetch



(a) Flip bit $K = 25$



(b) Flip bit $K = 50$



(c) Four room

Figure 11: Comparison between hEM and HPG. HPG performs well on Flip bit MDP with $K = 25$, but when $K = 50$ its performance drops drastically. HPG also underperforms hEM on the four room environment where it makes fast progress initially but saturates to a low sub-optimal level.

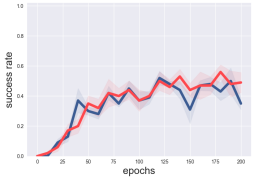
tasks in general have more complicated goal space - purely random exploration might not cover the entire goal space fully. This observation has been corroborated in recent work [51, 52].

Details on algorithms. To solve these tasks, we build on the built-in exploration mechanism of HER implementations [61], which have been shown to work in certain setups. Though hEM does not utilize any value function critics, it shares the policy network with HER. We implement the aggregate loss function as

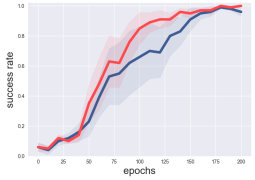
$$L(\theta) = L_{\text{her}} + \eta \cdot L_{\text{hem}} = \mathbb{E}_{(s,g)\sim\mathcal{D}}\left[Q_\phi\left(s, \pi_\theta(s), g\right)\right] + \eta \cdot \mathbb{E}_{q(\tau,g)}\left[\sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t, g)\right].$$

for the policy network. Here $\eta \in \{0.1, 0.2, 0.5\}$ is a hyper-parameter we selected manually to determine the trade-offs between two loss functions. Intuitively, when the policy cannot benefit from the learning signals of HER via L_{her}, it is still able to learn via the supervised learning update through L_{hem}. In practice, we find $\eta = 0.1$ works uniformly well.

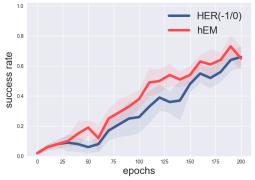
Results. See Figure 12 for the full results. Note that hEM provides marginal speed up on the learning on Push and Pick & Place. On Slide, both algorithms get stuck at local optimal (similar observations were made in [2]). We speculate that improvements in the exploration literature would bring further consistent performance gains.



(a) Slide



(b) Push



(c) Pick and Place

Figure 12: Fetch tasks. Training curves of hEM and HER on two standard Fetch tasks. hEM provides marginal speed up compared to HER. All curves are calculated based on 5 random seeds.