BI-LINEAR VALUE NETWORKS FOR MULTI-GOAL REINFORCEMENT LEARNING

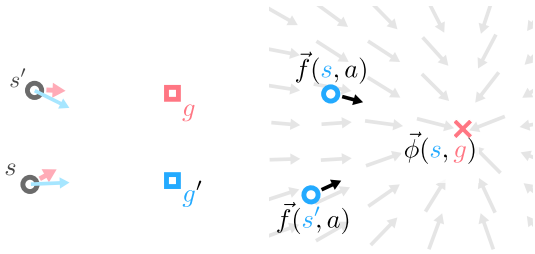
Anonymous authors
Paper under double-blind review

ABSTRACT

Universal value functions are used to score the long-term utility of actions to achieve a goal from the current state. In contrast to prior methods that learn a monolithic function to approximate the value, we propose a bi-linear decomposed value function. The first component, akin to a *global* plan models how the state should be changed to reach the goal. The second component, akin to a *local* controller selects the optimal action to actualize the desired change in state. We simultaneously learn both the components. Such decomposition enables both the global and local components to make efficient use of interaction data and independently generalize. The consequence is superior overall generalization and performance of our system on a wide range of challenging goal conditioned tasks in comparison to the current state-of-the-art.

1 INTRODUCTION

Constructing agents capable of accomplishing a diverse set of goals is a long standing challenge in reinforcement learning (Kaelbling, 1993). A major issue is the high sample complexity resulting from interference caused by learning a single policy or value function for a multitude of goals. The sample complexity can be partially mitigated by re-using interaction data such as in off-policy learning along with hindsight data re-labelling (HER, see Andrychowicz et al. 2017). A complementary approach improves data efficiency by encoding inductive biases in the neural network, such as the graph structure (Li et al., 2019). Such a structure enables sharing information across goals and thereby mitigates interference and results in better generalization.



(a) Move towards goals (b) Bilinear Decomposition

Figure 1: (a) The value towards a goal depends on both the current state s and the goal g which is far away. (b) we can capture this relationship by decomposing the value function into a bi-linear product between a *local* vector field $\vec{f}(s, a)$ and a *global* vector field, $\vec{\phi}(s, g)$. Gray arrows indicate the vector field $\vec{\phi}(s, g)$ evaluated at all states. The black arrow indicates the best alignment with $\vec{f}(s, a)$.

In off-policy goal-conditioned setting it is typical to estimate the *universal value function* (UVFA Schaul et al. 2015) that approximates the long-term return for a set of goals. One possibility is to model the UVFA as a single neural network, $Q_\theta(s, a, g)$, where s, a, g denote state, action and goal and θ are the model parameters. In such a form, there is no explicit incentive for the network to take advantage of the shared structure between goals, such that such structure can be leveraged when the agent is tasked with a new goal (Sutton et al., 2011). Transfer to new goals type of transfer to new goals is especially important during the early stage of learning, when the ability for the policy to perform given limited data has a significant compounding effect through the active collection of interactive data.

To exploit the shared structure between goals, we decompose the value prediction into two disentangled concerns *"where to go"* and *"what can I do"*. *"Where to go"* models how should the state be changed to get closer to the goal, while *"what I can do"* captures where the agent can move at the

current state. The knowledge of *"what I can do"* is independent of *"where to go"* and thus can be re-used across new goals. We illustrate this intuition in figure 1a. As an analogy to *"what I can do"*, the red arrow denotes the succeeding state after taking an action at state s, which is goal-agnostic. Independent of action, the blue arrow represents *"where to go"* that can be viewed as a direction to the goal g from the current state s. Adhering to this intuition, we thus treat value predictions as the interactions between two vectors. Consider *"what I can do"* and *"where to go"* as two vector fields $\vec{f}(s, a)$ and $\vec{\varphi}(s, g)$ in figure 1b. It is natural to define the value predictions as the level of alignment between two vector field $\vec{f}$ and $\vec{\varphi}$. The maximum value implicitly emerges when *"what I can do"* ($\vec{f}(s, a)$) corresponding to taking an action a aligns *"where to go"* ($\vec{\varphi}(s, g)$). In fact, our intuition resembles bilinear mapping in vector analysis, that parametrizes a scalar function by two vector spaces.

Our principal contribution is bi-linear value network (BVN), a parameterization scheme that decomposes the goal-conditioned value function into a bilinear map between two distinct vector representations. The first component $\phi(s, g)$ represents the long-distance relationship between the current state and a distant goal, whereas the second component $f(s, a)$ is more *local*, representing the immediate action at the current state. To understand the learning dynamics, we evaluate the agent on goals that never appeared during training. Surprisingly, we found that the bi-linear decomposition allows the critic to generalize to unseen goals (see section 4.3), which in turn improves the overall sample complexity and policy performance. We evaluate our approach on the challenging task of dexterous manipulation with a simulated shadow hand and other object manipulation tasks. Our method, BVN, achieves state-of-the-art sample efficiency *and* asymptotic performance on the majority of tasks. Apart from comparing with prior works, we also produce detailed analysis on the influence of the dimensionality of the two latent spaces, and how different constructions of the binary mapping affects performance.

2 Preliminaries

We consider a special case of the Markov decision process (MDP), the multi-goal reinforcement learning problem (Kaelbling, 1993). At the beginning of an episode, an initial state s_0 and a *behavior* goal g_b is sampled according to the distribution $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{G}}$. The behavior goal g_b is fixed throughout the episode during roll-out by the policy π. For each time step t, the agent takes action $a_t = \pi(s_t, g_b)$, receives reward $r_{t,g_b} = R(s_t, a_t, g_b)$, and moves to the next state s_{t+1}. We consider an infinite horizon MDP problem with the discount factor γ. The goal of multi-goal RL is to produce a policy π^* that maximizes the expected return J

$$\pi^* = \arg\max_{\pi} J(\pi) \text{ where } J(\pi) = \mathbb{E}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau,g_b} | s_t = s, g = g_b\right]. \tag{1}$$

2.1 Off-policy RL with Actor-critic Methods

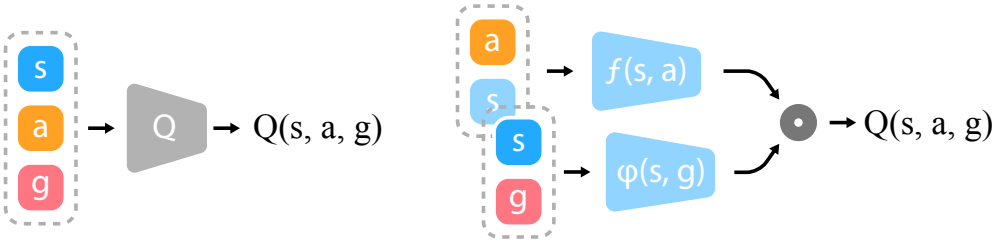
As we consider multi-goal continuous control tasks, we follow (Andrychowicz et al., 2017) and use deterministic policy gradient algorithm (DDPG) (Silver et al., 2014) to learn a π^*. DDPG is an actor-critic method. Note that a more advanced actor-critic method (Fujimoto et al., 2018) can be used, but we empirically found that DDPG is better than that in the tasks we consider. In actor-critic methods the value function, referred to as the critic, is responsible for generating the learning signal of π. The critic is trained by minimizing the loss function

$$L(Q) = \mathbb{E}_{(s_t,a_t)\sim\text{Uniform}(\mathcal{Z})}(r_{t,g} + \gamma Q(s_{t+1}, \pi(s_{t+1}, g), g) - Q(s_t, a_t, g))^2, \tag{2}$$

where $\mathcal{Z}$ denotes a replay buffer (Mnih et al., 2015) that stores multiple state transitions tuples $(s_t, a_t, r_t, s_{t+1}, g_b)$. The actor π is trained to maximize the critic outputs by the gradient

$$\nabla_\pi J(\pi) = \mathbb{E}[\nabla_a Q^\pi(s, a, g)|_{a=\pi(s,g)}]. \tag{3}$$

In multi-goal RL settings, the value function Q is often referred as *universal value function approximator* (Schaul et al., 2015) since Q encompasses the values of all goals in the MDP. As we are particularly interested in sample efficiency of policy learning, we highlight the importance of the critic for the sample efficiency of training the actor. A critic that provides better estimate of values for previously unseen states (i.e., better generalization of the critic) will lead to faster policy improvement via optimization in Equation 3.



(a) UVFA with a monolithic network

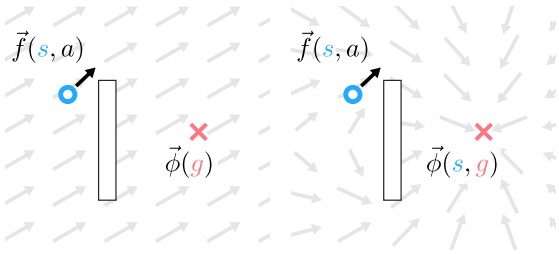
(b) Bilinear Value Network

Figure 3: The illustrations of value function architectures.

3 Bi-linear Value Decomposition for Multi-goal RL

The key insight behind the bilinear value network architecture, as illustrated in figure 2, is that we can often decompose value estimate into the alignment between the concern of "what I can do" and "where I want to go." When these two concerns are implicitly aligned, we consider the action to be good for reaching that goal. Whereas if these two are misaligned, the action is considered bad. These two concerns interact with each other when we try to find the solution to what I can do, but they are also disentangled because they take in different inputs. "What I can do," for example, only concerns the current state of affair and the possible set of actions. When a new goal appears the agent would not have to reconsider the relationship between its current state and the available actions, therefore lowers the burden to learn. It does however need to update what it knows about getting to that goal from where it is, which is inescapable.

To bring this insight into reality, we design a drop-in replacement of the standard *universal value function approximator*, as illustrated in figure 3, which we refer to as the *bilinear value network*. For context we present the standard architecture of a universal value function approximator (UVFA) on the left (see figure 3a). The UVFA is a single, monolithic neural network which is entangled in the way it processes the inputs. It first concatenates the state, action and the goal together into a single vector, and does not impose any inductive bias into the way these inputs interact.



(a) Vector Field $\vec{\phi}(g)$

(b) $\vec{\phi}(s,g)$

Figure 2: We compare two different parameterizations. The gray arrows indicate the vector field $\vec{\phi}$ evaluated at a particular goal g, for all states $s \in \mathcal{S}$. In (a), the vector field $\phi(g)$ is a constant and does not depend on s. (b) parameterizes the vector field via both s and g, so that the *local* field, $\vec{f}(s,a)$ only needs to align to it.

With bilinear value networks (see figure 3b), we partition the three input vectors into two overlapping groups. In the first group that we consider to be *local*, the neural network $\vec{f}(s,a)$ is only concerned with the current state and the immediate actions the agent can take. This is the component that is relatively easier to learn, because it does not have to integrate over long horizons. The second group is more *global*. The neural network $\vec{\phi}(s,g)$ captures the long-horizon relationship between the current state and the goal which can be quite far apart.

To bring the information that are processed separately by these two components, we borrow the concept of bilinear map from vector analysis, and define the Q function as a dot product between the two distinct vector representations

$$Q : Z_f \times Z_\phi \mapsto \mathbb{R}. \tag{4}$$

A consequence of this treatment is that the *value* becomes the result of an *alignment* between the two vector fields:

$$Q(s,a,g) = \vec{f}(s,a)^\top \vec{\phi}(s,g). \tag{5}$$

In figure 2, we show that passing the current state s into the global stream $\phi(s, g)$ is important, because to find the solution to the optimal action, the global field needs to vary depending on the current state. Figure 2a shows a goal stream that does not take into the current state – the vector field becomes a constant $\forall s \in \mathcal{S}$, making it impossible to accommodate the complex behavior needed to succeed from the entire state space.

An important issue to keep in mind is that Equation 5 is *unidentifiable* in the statistical sense because we can insert an arbitrary vector $\boldsymbol{\lambda}$ into the vector space and produce the same value estimate, as long as $\boldsymbol{\lambda}$ contains no zeros. In statistics, *identifiability* is the condition under which we can recover the true parameter of the model given infinite samples.

$$Q(s, a, g) = f(s, a)^\top \phi(g) \iff Q(s, a, g) = (\boldsymbol{\lambda} f)^\top (\boldsymbol{\lambda}^{-1} \phi) \quad \forall \quad \boldsymbol{\lambda} \in \mathbb{R}^n. \tag{6}$$

What this means is that under end-to-end training, the particular semantics of the two vector fields becomes particularly irrelevant. It is the grouping of the input variables and the bilinear structure that is critical.

It is also important to note that the bilinear value networks is implemented as a drop-in replacement of the universal value function approximator, as opposed to any additional algorithmic step. The training procedure we adopt in this paper follows those of deep deterministic policy gradient and hindsight experience replay, but the bilinear value networks is equally applicable to other off-policy learning algorithms under goal-conditioned scenario that uses either a value function, or a critic.

4 Experiment

Our experiments are aimed to answer the following two questions: (1) Does our value function parametrization improve sample efficiency? (2) Does the sample efficiency gain result from better value function generalizability? Moreover, we investigated the influence of each of our design choices in section 4.4.

4.1 Setup

Tasks We evaluate our method in well-known multi-goal RL benchmarks, *Fetch* robot and *Shadow hand* manipulation tasks, as shown in figure 4. For *Fetch* robot domain, the robot has to interact with the object and move the object to the target positions. In *Shadow hand* domain, the agent controls the joints of the hand's fingers and manipulates the objects in hand to a target pose. The details of these tasks can be found in the the section A.1.

Baselines and our method We compare with two baselines: *UVFA* (Andrychowicz et al., 2017) and *Bi-linear (lowrank)* (Schaul et al., 2015). *UVFA* is the most common multi-goal value function architecture that parametrizes the Q-function by a monolithic neural network. *Bi-linear (lowrank)* proposed in (Schaul et al., 2015) is a bi-linear value function parameterization similar to our method. *Bi-linear (lowrank)* decomposes the Q-function into $Q(s, a, g) = f(s, a)^T \varphi(g)$, where f and φ denote two neural networks respectively. We term *Bi-linear (lowrank)* as *lowrank* since their parametrization $\varphi(g)$ is incapable of having sufficient expressivity to model states' variations. In the following sections, we will refer our method to as *Bi-linear (ours)*.

Implementation The baselines and our method are trained by deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) with hindsight experience replay (HER) (Andrychowicz et al., 2017). Following the implementation in (Andrychowicz et al., 2017), we train each agent in parallel with multiple workers (varies across tasks). For *Bi-linear (ours)* and *Bi-linear (lowrank)*, the output dimensions f, $\varphi(s, g)$, and $\varphi(g)$ are set as 16. The rest of hyperparameters for training are left in section A.2.

Evaluation metric We ran each experiment with 5 different random seeds and reported the mean (solid or dashed line) and 95%-confidence interval on the learning curves. The learning curves indicate the average success rate over 15 testing rollouts in a varying number of training epoch. One epoch corresponds to the number of workers times two rollouts.

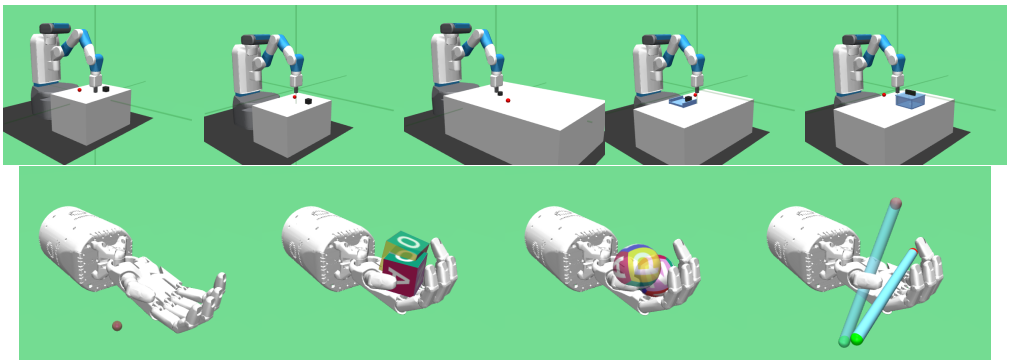


Figure 4: The illustrations of each task in Fetch and ShadowHand domains.

4.2 Sample Efficiency and Asymptotic Performance

To examine if our bi-linear value function parametrization improves sample efficiency, we evaluate each method in *Fetch* and *Shadow hand* domains. For *Fetch* domain, figure 5 shows that our method exhibits higher sample efficiency in 4 out of 8 tasks and matches the baselines in the rest of tasks. Remarkably, our method even attains higher asymptotic performance in *Bin-place*. Figure 5 shows that in *Shadow hand*, our method substantially improves the sample efficiency in 6 out of 8 tasks and is on par with the baselines for the rest of tasks. These results suggest that our bi-linear value network advances the sample efficiency. One explanation is that better Q-function's generalizability across goals helps the agent quickly learn a useful critic that will improve the policy's performance. As a result, the exploration will be more focused on high-return regime. The collected high-return experience in turn improves the policy.

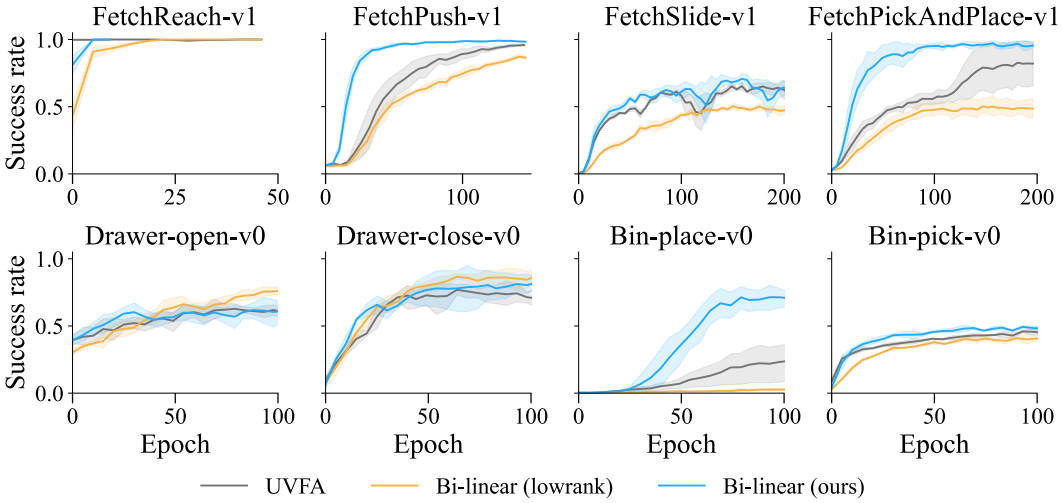


Figure 5: Learning curves on the *Fetch* gym environments, and the *Fetch-extension* (see section A.3) task suit.

4.3 Generalization to Unseen Goals

To verify our hypothesis that generalizability improves sample efficiency, we examine bi-linear value network 's performance on unseen goals. A Q-function that generalizes well should result in a policy that obtains high rewards on unseen goals since the policy's training signals is solely from the Q-function in actor-critic methods. To quantify the generalizability of the Q-function we split the goal distributions at training and testing time. We consider two scenarios in the *Fetch* robot domain: (1) near-to-far and (2) left-to-right. In near-to-far case, we specify a threshold on the goals' distance to the agent's initial state, and sample goals with euclidean distance under and above the threshold for

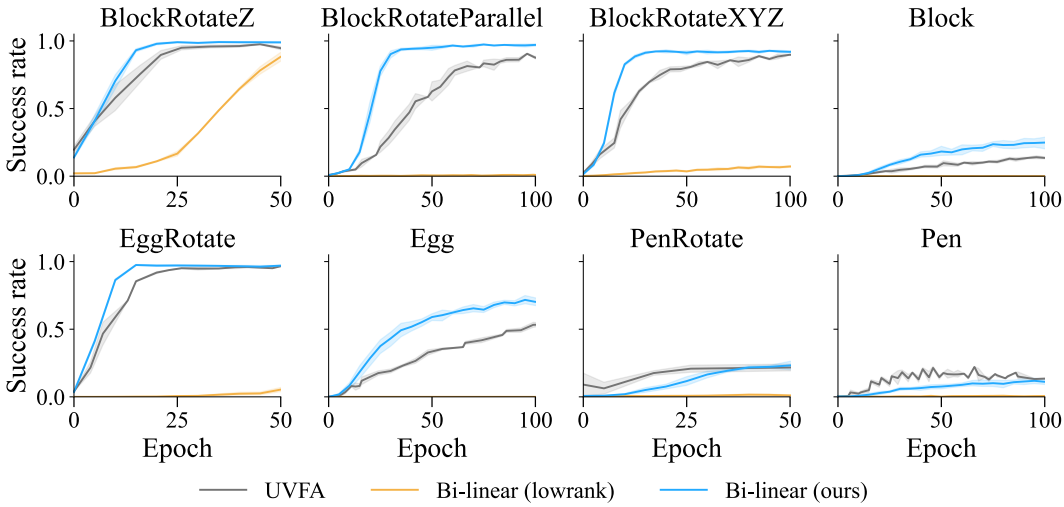


Figure 6: Learning curves on the *Shadow hand* dexterous manipulation suite.

training and testing respectively. For left-to-right scenario, we split the goal space by an axis on the table (see figure 7) into the left and right half-spaces.

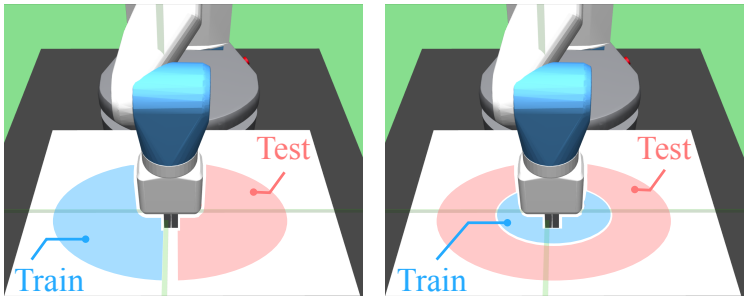


Figure 7: Example showing how (a) we split the goal distribution into left and right and (b) according to the radius to the center (near versus far). We train with the blue regions, then evaluate on the red regions. The state distribution remain identical to the vanilla *Fetch* environment.

The goals from the left space are sampled at training time, whereas others at test time. We present the rest of implementation details on these scenarios in section A.3. Figure 8 shows both the training and testing success rate of the policies trained by each method. It can be seen that our method outperforms the baselines in terms of testing performance, showing that our method generalizes better to unseen goals. Moreover, it is shown that our method's gap between training and testing success rate are smaller than that of the baselines in *Push* and *Slide* in both near-to-far and left-to-right scenarios. Notably, figure 8 shows that *Bi-linear (lowrank)* underperforms the *UVFA* in most tasks, suggesting that taking states as inputs at both branches (i.e., f and φ) is crucial to the success of the bi-linear decomposed value function.

4.4 Ablation Studies

We investigate the influence of each component in our method to further our understanding of the bi-linear value function. First, as the output dimension of f and φ could affect the expressivity of our bi-linear Q-function, we are interested in what extent this dimension influences the performance. Also, we investigate if dot product is a good way to combine f and φ by comparing ways to combine them. Lastly, we investigate if there is other viable ways to decompose the Q-function as well as to group the inputs s, a, g.

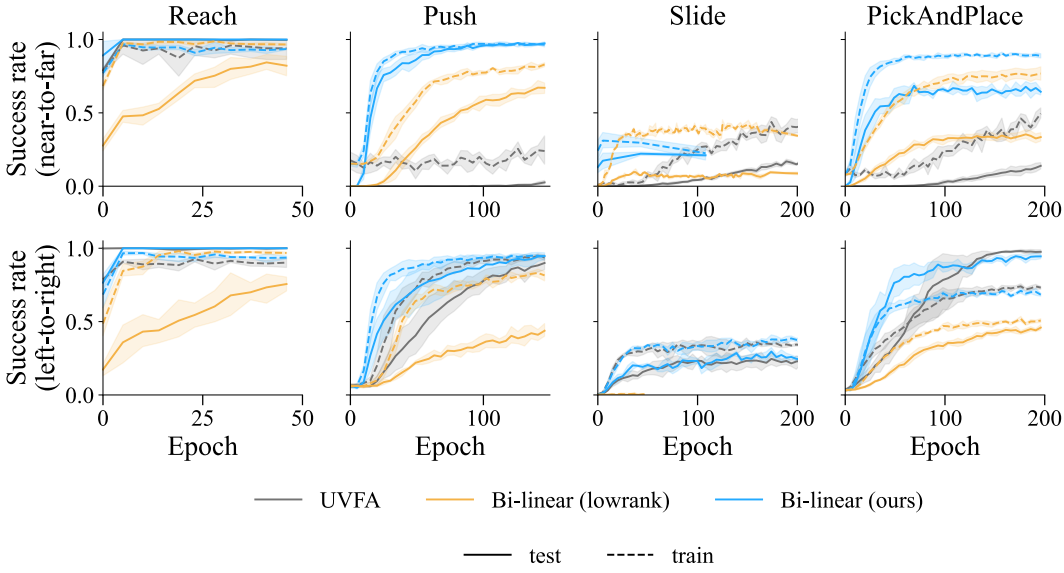


Figure 8: Our method attains higher testing performance than the baselines. This result shows that bi-linear value network improves the Q-function's and the policy's generalizability.

How does latent dimension affect our method? We made several variants of our method with different latent dimension of the outputs of f and φ and present their learning curves in figure 9. It can be seen that the performance consistently improves when we increase the latent dimension. Though the performance drops when the dimension decreases, our method with dimension 3 still outperforms the *UVFA* baseline. These results suggest that our method benefits from increasing latent dimension but is insensitive to the choice of the dimension.

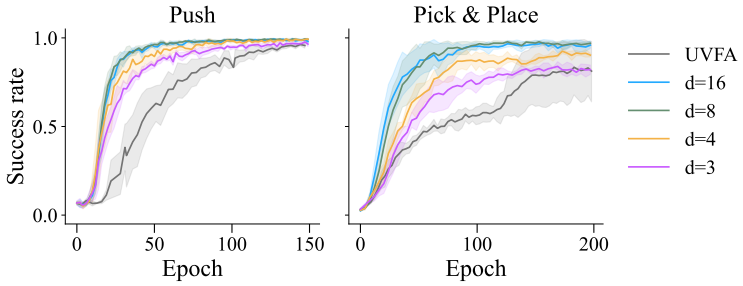


Figure 9: Increasing the latent dimension of f and φ improves our method's performance. Despite that, the result shows that a low dimensional latent space ($d = 3$) is sufficient to enable our method to outperform the baseline *UVFA*.

Is combining f and φ by dot product crucial? We compared the variant that reduces f and φ by 2-norm. In 2-norm variant, we parametrize the Q-function as $Q(s, a, g) = -||f(s, a) - \varphi(s, g)||_2$. The negative sign is added because multi-goal RL tasks typically assume the reward is within the range $[-1, 0]$. For the 2-norm variant, we set its latent dimension as the best one found in our hyperparameter search (see section A.3). The results shown in figure 10 demonstrate that *Bi-linear-full (2-norm)* approximately matches the performance of *Bi-linear-full (dot)*, suggesting that dot product can be replaced with other functions (e.g., norm).

Does the performance gain result from a larger model? To answer this question, we compared a variant that concatenates f and φ at the last layer of the value network. For concatenation variant, the Q-function is formulated as $Q(s, a, g) = \text{MLP}([f(s, a), \varphi(s, g)], 1)$, where we feed the concatenated $f(s, a)$ and $\varphi(s, g)$ to a multi-layer perceptron (MLP) network that outputs a one-dimensional value. This concatenated variant has slightly more parameters than our method since

the last layer introduces a few more parameters. The implementation details of these variants are left at section A.3. We use the same latent dimension as we use in section 4.2. Figure 10 shows that concatenation ruins the performance, suggesting a larger model size that results from decomposition is not the reason for our sample efficiency gain.

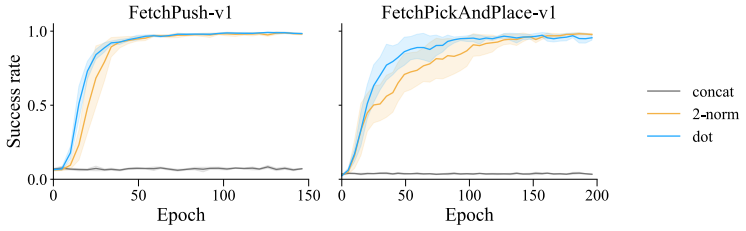


Figure 10: The variant that combines f and φ by 2-norm approximately matches the performance of the variant of using dot product. Concatenating f and φ in an MLP ruins the performance, which suggests that decoupling f and φ is crucial.

Does other inputs arguments grouping strategy work as well? One hypothesis is that any inputs arguments (i.e., s, a, g) grouping can enhance sample efficiency as long as the parametrization does not have low-rank issues in *Bi-linear (lowrank)*. Thus we compare with an alternative inputs grouping scheme that groups s, a, g into f component and g into φ. The Q-function is then formulated as $Q(s, a, g) = f(s, a, g)^T \varphi(g)$. This parametrization does not suffer from low rank issues since $f(s, a, g)$ itself is already sufficient to express the Q-value well, unlike *Bi-linear (lowrank)*. Figure 11 demonstrates that $f(s, a, g)^T \varphi(g)$ grouping cannot match the performance of our proposed $f(s, a)^T \varphi(s, g)$, suggesting that not only the decomposition is important but a carefully designed grouping scheme is also crucial.

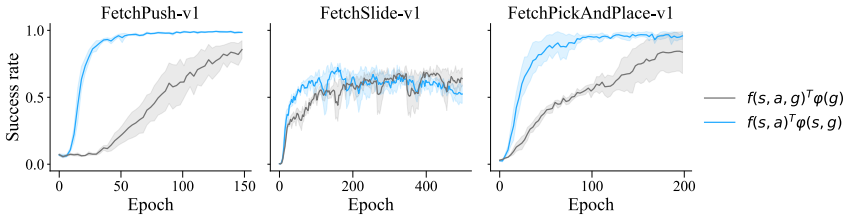


Figure 11: Our $f(s, a)^T \varphi(s, g)$ grouping outperforms $f(s, a, g)^T \varphi(g)$ even though the issues of lack expressivity of *Bi-linear (lowrank)* does not exist in $f(s, a, g)^T \varphi(g)$. It suggests that inputs grouping is crucial for value function decomposition.

5 Related Works

Our work is closely related to the literature on value function approximation in the sense that both seek for a lower dimensional space to represent the value function in a more compact way. Most early works in RL approximate the value function by the linear combination of basis functions (Sutton & Barto, 2018) (e.g., polynomial basis, radial basis, fourier basis (Konidaris et al., 2011)). Instead of arbitrarily picking basis functions, proto-value function framework (Mahadevan, 2007; Mahadevan & Maggioni, 2006) compose the value function in the linear combination of task-independent basis vectors constructed by the spectral of the state transition function of the environment. The works on successor representations (Dayan, 1993; Kulkarni et al., 2016; Borsa et al., 2018; Barreto et al., 2017; 2019) seek for a value function representation that help the value function transfer to new tasks. This line of works bi-linearly compose the value function by a reward-agnostic and reward-specific parts, so the reward-agnostic part can be reused in a new task where the environment's transition function is the same. The idea of successor representations was introduced to multi-goal RL in (Schaul et al., 2015) for decomposing the goal-conditioned value function. Its difference to prior works on successor representations is that the reward-specific component is a function conditioned on goals. Despite the appealing performance shown in toy gridworld, the

follow-up works in multi-goal RL rarely use the decomposed architecture proposed in (Schaul et al., 2015). Andrychowicz et al. (2017); Pong et al. (2018) all parameterize the Q-function as a monolithic neural network. Section 4.2 empirically shows that the low-rank bi-linear decomposed value function proposed in (Schaul et al., 2015) cannot extend its advantages to more complex tasks. As a result, positioned in the literature of multi-goal value function approximation, our principal contribution lies in solving the bottleneck that hinders the advantages of value function decomposition in multi-goal RL.

In the abstract sense, the spirit of our method is to model the goal-agnostic (state-action) and goal-specific (state-goal) information separately in different components. A proper decomposition is known to make function approximation easier due to modularity. Rusu et al. (2016) demonstrates the modularity in a network facilitates continual learning in new tasks, as the previously learned sub-networks can be reused in new tasks Wang et al. (2018) learns a modular policy where each network controls a joint of a robot. This modular policy is shown to be able to better transfer to new robot morphology. Galashov et al. (2019) train a specialized policy and a reusable policy restricted from the reward information for encouraging the reusability in other tasks. Different from these works, our work imposes modularity in the parametrization of value functions and focuses on improving genearlizability and sample efficiency of value functions.

6 Discussion

Our experimental results demonstrated that our BVN significantly improves sample efficiency in multi-goal RL tasks. Also, we show that this sample efficiency gain could result from better generalizability of the value function. In addition to the above empirical results, we discuss the design choice of each component and the potential extensions below.

Why does other parameter grouping schemes fail? In section 4.4, we showed that $f(s, a, g)^T \varphi(g)$ inputs grouping fails to improve sample efficiency. One explanation to this result is that $f(s, a, g)$ cannot prevent negative interference from new goals encountered during training since goals g couple with states-actions s, a. As a result, the merit of bi-linear value function decomposition vanishes and even causes extra burden on learning because the number of model parameters is larger than *UVFA*. This preliminary finding shows that to benefit from decomposition, the guideline is to only expose necessary inputs to each component. Redundant inputs in either component could break the decomposition.

How is 2-norm compared with dot product? Section 4.4 demonstrated that using 2-norm to reduce $f(s, a)$ and $\varphi(s, g)$ into Q-value matches the performance of our BVN with even lower dimensions of Z_f and Z_φ. However, 2-norm relies on an assumption that the reward function must be non-positive or non-negative since norm is defined as a non-negative value. The $\|f(s, a) - \varphi(s, g)\|$ can only fit to either non-positive or non-negative Q-value by manipulating the sign of norm. On the other hand, dot product is more expressive and free from the reward function assumption.

Can our method facilitate policy transfer? Since our method decomposes the value function into a local (i.e., goal-agnostic) and a global (i.e., goal-specific) component, our BVN might facilitate policy transfer by resetting either one of components and fine-tuning another one. As multi-goal tasks considered in (Plappert et al., 2018) have the same objective in nature but different dynamics, one potential advantage of our method is to reuse $\varphi(s, g)$ in the new task and re-train $f(s, a)$. The results in section A.4 showed that the agent transferred by BVN converges to the the performance of training from scratch in the target task slightly faster than the baselines. One reason for the limited policy transferring performance is that $\varphi(s, g)$ conditions on states s. Consequently, environment dynamics shifts could affect the representations of φ. We, nevertheless, showed that states variables are crucial in φ. One exciting avenue is to disentangle the representations for keeping the rich expressivity as well as desired modularity for transferring across tasks.

7 Reproducibility Statement

Please refer to section A.3.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. July 2017. URL http://arxiv.org/abs/1707.01495. 1, 2, 4, 9

A Barreto, W Dabney, R Munos, J J Hunt, and others. Successor features for transfer in reinforcement learning. *Adv. Neural Inf. Process. Syst.*, 2017. ISSN 1049-5258. URL http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning. 8

A Barreto, D Borsa, J Quan, T Schaul, D Silver, and others. Transfer in deep reinforcement learning using successor features and generalised policy improvement. *arXiv preprint arXiv*, 2019. URL https://arxiv.org/abs/1901.10964. 8

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. December 2018. URL http://arxiv.org/abs/1812.07626. 8

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Comput.*, 5(4):613–624, July 1993. ISSN 0899-7667. doi: 10.1162/neco.1993.5.4.613. URL http://dx.doi.org/10.1162/neco.1993.5.4.613. 8

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. February 2018. URL http://arxiv.org/abs/1802.09477. 2

Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. *arXiv preprint arXiv:1905.01240*, 2019. 9

L P Kaelbling. Learning to achieve goals. *IJCAI*, 1993. ISSN 1045-0823. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.3077&rep=rep1&type=pdf. 1, 2

George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. *AAAI*, 25(1):380–385, August 2011. ISSN 2374-3468, 2374-3468. URL https://ojs.aaai.org/index.php/AAAI/article/view/7903. 8

Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *ArXiv*, 2016. URL https://www.semanticscholar.org/paper/10a4992ece5baea79326a8878a6244eeacbc6af5. 8

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pp. 11674–11685, 2019. 1

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 4

Sridhar Mahadevan. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. https://www.jmlr.org/papers/volume8/mahadevan07a/mahadevan07a.pdf, 2007. URL https://www.jmlr.org/papers/volume8/mahadevan07a/mahadevan07a.pdf. Accessed: 2021-9-23. 8

Sridhar Mahadevan and Mauro Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. *Adv. Neural Inf. Process. Syst.*, 18:843, 2006. ISSN 1049-5258. URL http://all.cs.umass.edu/pubs/2005/mahadevan_m_TECH05.pdf. 8

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. 2

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018. 9, 12

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018. 9

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 9

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 2015. PMLR. URL http://proceedings.mlr.press/v37/schaul15.html. 1, 2, 4, 8, 9

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pp. 387–395, January 2014. URL http://www.jmlr.org/proceedings/papers/v32/silver14.html. 2

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, October 2018. ISBN 9780262352703. URL https://play.google.com/store/books/details?id=uWV0DwAAQBAJ. 8

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011. 1

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018. 9

A APPENDIX

A.1 IMPLEMENTATION DETAILS OF TASKS

The implementation details of all the tasks used in section 4.2 are referred to (Plappert et al., 2018). In addition to (Plappert et al., 2018), we also use the tasks from Fetch extension[1]. For left/right environments used in section 4.3, the goal (x, y, z) with $x < 0$ are left goals, and $x \geq 0$ right goals. In near/far environments, we set the distance thresholds as 0.2 for Slide, 0.2 for PickAndPlace, 0.1 for Reach, and 0.15 for Push. The goals with distance below the threshold are considered as near goals, otherwise far goals.

A.2 TRAINING HYPERPARAMETERS

Fetch We adapted the hyperparameters in (Plappert et al., 2018) for training in a single desktop. The performances of the agent trained by our hyperparameters match that in (Plappert et al., 2018).

- `num_workers`: 2 for Reach, 8 for Push, 16 for Pick & Place, and 20 for Slide.
- `Batch size`: 1024
- `Warm up rollouts`: We collected 100 initial rollouts for prefilling the replay buffer.
- `Training frequency`: We train the agent per 2 environment steps. Each iteration of training uses the `num_workers` batches for training.

The rest of hyperparameters remain the same as that used in (Plappert et al., 2018).

Shadow hand We use the hyperparameters used in (Plappert et al., 2018) and reuse the open source code[2].

A.3 IMPLEMENTATION DETAILS

Dot product This is our proposed method. We set the outputs' dimension of f and φ as 16.

2-norm We parametrize Q-function as $Q(s, a, g) = \|f(s, a) - \varphi(s, g)\|_2$. We use set the dimension of the outputs from f and φ as 3. Figure 12 shows that $d = 3$ works the best for 2-norm.

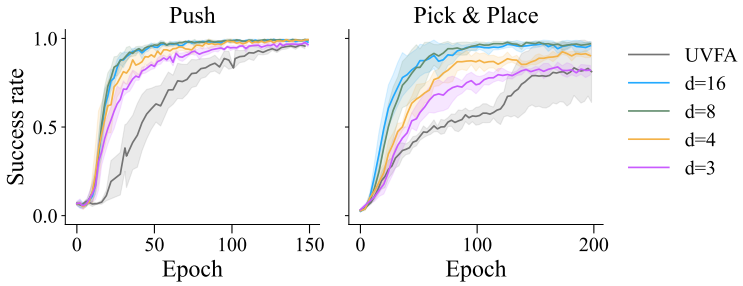


Figure 12: The learning curves for 2-norm variant with different d. We show that $d = 3$ is best for 2-norm.

concat We concatenate $f(s, a)$ and $\varphi(s, g)$, feed the concatenated vectors to a multi-layer perceptron (MLP) layer, and take the outputs of this layer as the Q-value $Q(s, a, g)$. The dimensions of the outputs of f and φ are 16.

[1] https://github.com/geyang/gym-fetch
[2] https://github.com/TianhongDai/hindsight-experience-replay

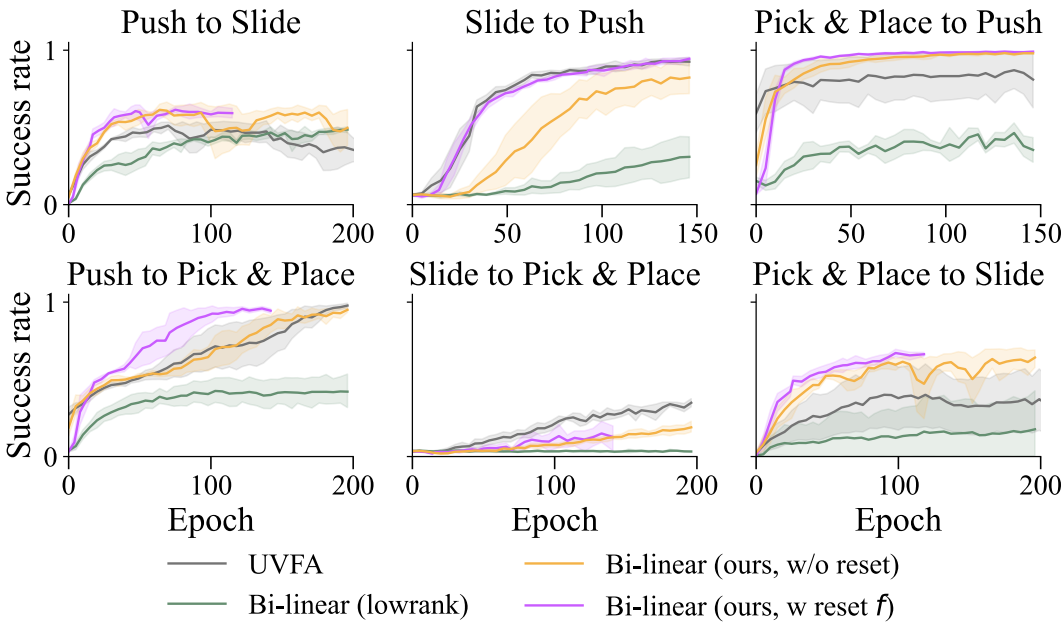


Figure 13: Resetting f and finetuning φ achieves higher success rate than the baselines in 4 out 6 task and mostly matches the baselines' performance in the rest of tasks.

A.4 Transferring Across Tasks

In multi-goal RL tasks, we observe that most of tasks are identical in terms of high-level goals, but they have different environment dynamics. For instance, the objective of "Push" and "Slide" are all to move the object to the destination. They only differ in the friction coefficient of the table where the agent cannot slide objects in "Push" because of high friction. Because we decouple the state-action and goals into two different functions, $f(s, a)$ and $\varphi(s, g)$, one potential benefit on multi-goal RL is reusing φ for transferring to tasks with the same (or similar) task objectives (i.e., reward specification). Reusing $\varphi(s, g)$ exploits the learned reward-specific state-goal relations. Then we simply need to learn a new f to recover the reward-agnostic relations in the new task. To examine this hypothesis, we select three tasks with the same state and action spaces[3], and measure the performance of transferring amongst these tasks. In transferring an agent from a source task to a target task, we initialize *UVFA* agent's policy and Q-function by the neural networks' weights learned in the source task and fine-tune them in the target task. Different from *UVFA*, we reinitialize f and fine-tune φ in *Bi-linear (lowrank)* and *Bi-linear (ours)* when training them in the target task. Figure 13 shows the success rates of transferring from a source task to a target task. For example, "Push to Slide" indicates transferring from "Push" to "Slide". It can be seen that *Bi-linear (ours, w reset f)* attains higher success rates at convergence than the baselines and *Bi-linear (ours, w/o reset)* in 4 out 6 tasks (i.e., "Push to Slide", "Pick & Place to Push", "Push to Pick & Place", and "Pick & Place to Slide"), despite slight improvements.

The limited efficiency gain could result from that φ is dependent on states s and thus φ cannot be effectively reused. Even though the target task's reward function is the same as that in the source task, the relation of states and goals can be affected by the dynamics. For instance, suppose we have a state where the object is near the goal. In "Pick & Place", because the friction coefficients of the table are high, the $\varphi(s, g)$ should direct the agent to lift the object. On the other hand, in "Slide", the $\varphi(s, g)$ is expected to guide the agent to push the object to slide.

[3]If the two tasks have different state/action spaces, we will not be able to use the weights from a task and resume training in another task.