# BILINEAR VALUE NETWORKS

**Zhang-Wei Hong**[*], **Ge Yang**[*] **& Pulkit Agrawal**
Improbable AI Lab
Massachusetts Institute Technology

## ABSTRACT

Universal value functions are a core component of off-policy multi-goal reinforcement learning. The de-facto paradigm is to approximate $Q(s, a, g)$ using monolithic neural networks which lack inductive biases to produce complex interactions between the state $s$ and the goal $g$. In this work, we propose a bilinear decomposition that represents the Q-value via a low-rank approximation in the form of a dot product between two vector fields. The first vector field, $f(s, a)$, captures the environment's local dynamics at the state $s$; whereas the second component, $\phi(s, g)$, captures the global relationship between the current state and the goal. We show that our bilinear decomposition scheme improves sample efficiency over the original monolithic value approximators, and transfer better to unseen goals. We demonstrate significant learning speed-up over a variety of tasks on a simulated robot arm, and the challenging task of dexterous manipulation with a Shadow hand.

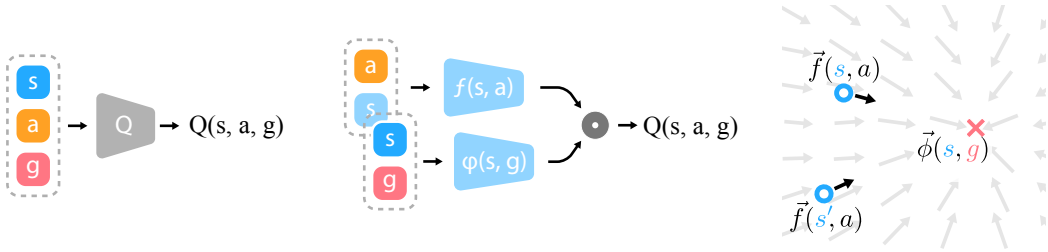
## 1 INTRODUCTION

Learning goal conditioned policies (Kaelbling, 1993; Jordan & Rumelhart, 1992; Wolpert et al., 1995; Schaul et al., 2015; Agrawal et al., 2016; Nair et al., 2017; Pathak et al., 2018; Andrychowicz et al., 2017; Paster et al., 2021) is a central problem in decision making. Reinforcement learning (RL) is a popular approach for learning goal-conditioned policies. Learning a single policy for many goals using RL is challenging because the data requirement is substantially higher than single task learning because the agent needs to collect data for multiple goals. To reduce the amount of data needed for learning, it is common to leverage off-policy methods to learn a goal-conditioned Q-value function $Q(s, a, g)$ (or, *Universal Value Function Approximator (UVFA)* (Schaul et al., 2015)), where $s$, $a$, and $g$ denote states, actions, and goals, respectively.

To efficiently learn $Q(s, a, g)$ for a large set of goals, it is important for the UVFA to generalize to new goals, as suggested in Schaul et al. (2015). The number of combinations in the inputs (i.e., $s, a$ and $g$) grows significantly than single-task learning problem since single-task problem does not require accounting for variations of goals. The increased number of input patterns impedes the value function learning progress. The speed of value function learning influences the progress of policy learning, hence impacting the data collection process that is crucial for sample efficiency. Under-performing data cannot effectively improve the agent's performance. As a result, the ability to generalize value predictions to unseen combinations state-actions and goals is critical for multi-goal RL. A generalizable Q-value function can make approximately accurate value predictions to new goals seen in the middle of training via the knowledge learned early during training.

A common approach to improve generalizability is using hindsight experience replay (HER) (Andrychowicz et al., 2017) training the UVFA with additional data produced by relabeling, such that UVFA is exposed to more diverse combinations of goals. An alternative strategy to achieve generalizability is disentangling the problem so that parts of the knowledge can be reused on new instances of the problem. For example, Barreto et al. (2016) decouple the representations of environment dynamics and rewards using successor features (Dayan, 1993), transferring the value function to new reward functions without re-training the dynamics representations. In the context of multi-goal RL, the goal-conditioned value function can be disentangled as the interaction between two concerns *"where to go"* and *"what happens next"*. *"Where to go"* models how should the state

[*]equal contribution, order is determined by rolling a dice




(a) A Monolithic UVFA (b) Bilinear Value Network (c) Bilinear Decomposition

Figure 1: Bilinear value decomposition decomposes the UVFA into a dot product between the neural representation of two vector fields: $f(s, a)$ and $\phi(s, g)$. (a) A monolithic UVFA that concatenates the inputs. (b) Bilinear Value Networks (BVN) that combines the two vector fields via a bilinear map. (c) Illustration of the bilinear decomposition into two vector fields. Gray arrows indicate the vector field $\vec{\phi}(s, g)$ evaluated at all states. The black arrow indicates the best alignment with $f(s, a)$.

be changed to get closer to the goal, while *"what happens next"* captures where the agent can move at the current state. Thus, one plausible way to factorize the value function is separating the *local* knowledge of the dynamics, and the *global* topology of the domain. The knowledge of *"what happens next"* is independent of *"where to go"* and thus can be re-used across new goals. We illustrate this intuition in Figure 1c. As an analogy to *"what I can do"*, the black arrow denotes the succeeding state after taking an action at state $s$, which is goal-agnostic. Independent of action, the grey 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 vector fields $f(s, a)$ and $\phi(s, g)$. The value predictions can be interpreted as the degree of alignment between two vector field $f$ and $\phi$. The maximum value implicitly emerges when *"what I can do"* ($f(s, a)$) corresponding to taking an action $a$ aligns *"where to go"* ($\phi(s, g)$). Our principal contribution is *bilinear value network* (BVN), a parameterization scheme, shown in Figure 1b, that increases the sample efficiency of the value function by decomposing the goal-conditioned value function into a bilinear map from two distinct vector fields into a scalar. The first vector field, a neural network $\phi(s, g)$ captures the long-range structure between the current state and a distant goal, whereas the second vector field, network $f(s, a)$ is *local*, which represents taking immediate action at the current state.

We build upon state-of-the-art off-policy learning algorithms such as DDPG (Lillicrap et al., 2016) and TD3 (Fujimoto et al., 2018) where the policy is supervised entirely by the value function on off-policy interaction data. We show empirically that bilinear value decomposition enables the value function to learn significantly faster than the commonly used monolithic architecture shown in Figure 1a (Section 5.1). Further, BVN also exhibits superior speed when adapting to new goals (Section 5.2). Results are presented on two standard benchmark task suites in simulation: (i) dexterous manipulation with a twenty degrees-of-freedom (DOF) shadow hand and (ii) object pick-place tasks using the Fetch manipulator.

## 2 Preliminaries

We consider the multi-goal reinforcement learning problem (Kaelbling, 1993; Plappert et al., 2018), formulated as a goal-conditioned Markov Decision Process (MDP). We assume infinite horizon with the discount factor $\gamma$, and use $\mathcal{S}, \mathcal{A}, \mathcal{G}$ for the state, action, and goal spaces. At the beginning of each episode, an initial state $s_0$ and a goal $g$ is sampled according to the distribution $\rho_{\mathcal{S}}$ and $\rho_{\mathcal{G}}$. The goal $g$ is fixed throughout the episode. At every time step $t$, the agent takes an action according to $a_t \sim \pi(s_t, g)$, and receives the reward $r_{t,g} = R(s_t, a_t, g)$ while transitioning to the next state $s_{t+1}$. The objective of multi-goal RL is to produce a policy $\pi^*$ that maximizes the expected return $J$

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

where the reward $r_{\tau,g}$ is often structured as the follows:

$$R(s_\tau, a_\tau, g) := \begin{cases} 0, & \text{goal reached} \\ -1, & \text{otherwise.} \end{cases} \quad \forall \tau \in \mathbb{N} \tag{2}$$

Note that our proposed parameterization scheme, BVN, does not rely on this reward structure in general. All of the baselines use a discount factor $\gamma = 0.98$.

### 2.1 Off-policy RL with Actor-critic Methods

The multi-goal continuous control tasks considered in this paper are typically solved via actor-critic algorithms such as *deep deterministic policy gradient* (DDPG) (Silver et al., 2014), *twin-delayed DDPG* (TD3) (Fujimoto et al., 2018) and *soft actor-critic* (SAC) (Haarnoja et al., 2018), to name a few. In DDPG, the critic $Q$ is learned by minimizing the temporal-difference (TD) loss:

$$\mathcal{L}(Q) = \mathbb{E}_{(s_t,a_t)\sim\text{Uniform}(\mathcal{D})}(r_{t,g} + \gamma Q(s_{t+1}, \pi(s_{t+1}, g), g) - Q(s_t, a_t, g))^2, \tag{3}$$

where $\mathcal{D}$ is the the replay buffer (Mnih et al., 2015) containing state transition tuples $(s_t, a_t, r_t, s_{t+1}, g)$. The actor $\pi$ is trained to maximize the score from the critic, using the deterministic policy gradient:

$$\nabla_\pi J(\pi) = \mathbb{E}[\nabla_a Q(s, a, g)|_{a=\pi(s,g)}]. \tag{4}$$

In multi-goal RL settings, we refer to the goal-conditioned critic as the *universal value function approximator (UVFA)* (Schaul et al., 2015) because $Q(s, a, g)$ approximates the expected utility for all goals $\in \mathcal{G}$. Because the actor relies on the critic for supervision, the ability for the critic to quickly learn and adapt to unseen goals is a key contributing factor to the overall sample efficiency of the learning procedure. A critic that provides better value estimates using less sample and optimization gradient steps, will lead to faster policy improvement via the gradient in Equation 4. We refer the reader to the original papers for the algorithmic details on SAC and TD3 because we found empirically that DDPG works better than TD3 and SAC on these robot manipulation tasks (see Section B).

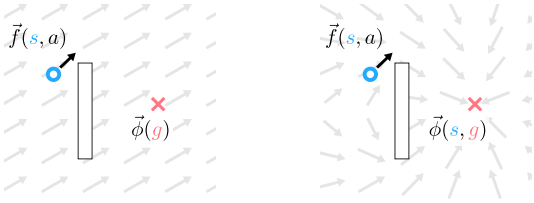
## 3 Bilinear Value Networks

Bilinear value network (BVN) factorizes the value function approximator into a bilinear form from two vector representations into a scalar:

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

where $f : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^d$ and $\phi : \mathcal{S} \times \mathcal{G} \mapsto \mathbb{R}^d$. The first component $f(s, a)$ is a *local* field that only concerns the immediate action at the current state $s$. The second component is more *global* in the sense that $\phi(s, g)$ captures long-horizon relationship between the current state and the goal.

A close bilinear decomposition was explored in Schaul et al. (2015), though local-global disentanglement was not the motivation of Schaul et al. (2015). However, such a bilinear decomposition was not widely used afterwards, despite being potentially beneficial for generalization. The majority of multi-goal RL algorithms employ a monolithic network shown in Figure 1a to model a value function. We argue that the critical missing piece for bilinear decomposition is a proper inputs factorization.




(a) $f(s,a)^\top \phi(g)$ (b) $f(s,a)^\top \phi(s,g)$

Figure 2: Comparison between two value decomposition schemes. Gray arrows indicate the vector field $\phi$ 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 field $f(s, a)$ is more expressive while benefits from disentanglement.

To motivate the design of our factorization, we present a vector analysis perspective on previous value decomposition Schaul et al. (2015) and our method in Figure 2, where each point on the 2D plane is a state $s$. In Figure 2a, we illustrate the vector field of low-rank bilinear

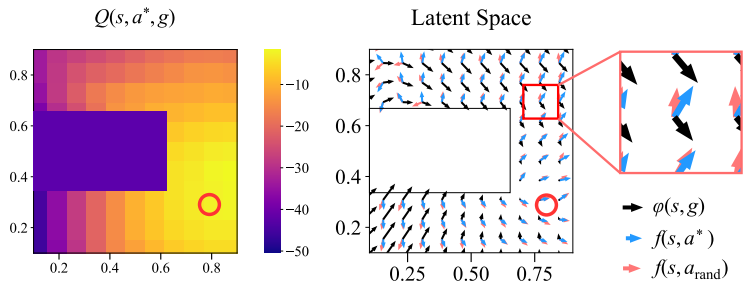



Figure 3: **Left:** Showing the learned value function $Q(s, a^*, g)$ over the 2-dimensional state space for a goal indicated by the red circle. Showing overall decrease in value as the agent gets further and further away from the goal. **Right:** Showing the latent vectors produced by the agent, projected to a 2D vector space. There is an overall trend for $\vec{\phi}$ to decrease in magnitude as one gets closer to the goal. **Right (Inset):** Compares the $f$ of a random action versus an optimal action from the policy $\pi$. The latter consistently produces higher value, and is better-aligned to $\phi$ with a smaller angle. The original vector space was $\mathbb{R}^{16}$.

UVFA (Schaul et al., 2015), where $f : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^d$ and $\phi : \mathcal{G} \mapsto \mathbb{R}^d$. It can be seen that $\phi(g)$ is a constant across states since $\phi(g)$ does not depend on the state $s$. This makes learning more challenging due to restricted expressivity. The expressivity of low-rank bilinear UVFA is restricted since the joint relation between state $s$ and goal $g$ only occurs in the dot product at the last layer. Consequently, the burden of modeling different Q-values for different states falls completely on $f$. In contrast, in Figure 2b, the field $\phi(s, g)$ produced by BVN is also a function of the state $s$. This makes $\phi$ in BVN strictly more expressive than low-rank bilinear UVFA that do not depend on $s$.

Notably, bilinear value networks is a drop-in replacement for the monolithic universal value function approximator. Our implementation is not specific to DDPG, SAC or TD3, but can be applied to any algorithms that computes a goal-conditioned value function. Bilinear value network can be trained by any value function learning objective such as TD objective introduced in Section 2.1.

### 3.1 A Toy Example: Visualizing $f(s, a)$ And $\phi(s, g)$

To examine whether the learned bilinear value decomposition reflect our intuition mentioned in Section 1, we trained a deep deterministic policy gradient (DDPG) (Lillicrap et al., 2016) with hindsight experience replay (HER) (Andrychowicz et al., 2017) agent using BVN in a 2D maze where $s \in \mathbb{R}^2$, $a \in \mathbb{R}^2$, $g \in \mathbb{R}^2$, and the reward is the negative euclidean distance between states and goals. The output dimensions of $f$ and $\phi$ are both 16. We collected the output vectors of $f(s, a^*)$ and $\phi(s, g)$ with a grid of states $(x, y)$, a fixed goal $g$, and $a^* = \pi(s, g)$. For visualization, we use principal component analysis (PCA) to project $\phi(s, g)$ to a 2-dimensional vector space, and plot them along with $f(s, a^*)$ in Figure 3 (right). We plot $f(s, a^*)$ as $\phi(s, g)$ with $\angle(f(s, a^*), \phi(s, g))$ rotation. In Figure 3 (right), we found that the lengths of the vectors reflect the distances to the goal at a state, which validates our hypothesis that $\phi$ captures the global topology to the goal (i.e., proximity to the goal). Moreover, we further compare the the angle $\angle_{f,\phi}$ between that of the optimal action (from the policy $\pi$) $f(s, a^*)$ and a sub-optimal, random action $f(s, a_{\text{rand}})$. We show that on this simple domain, the $f(s, a^*)$ better aligns with $\phi(s, g)$, validating our intuition that the policy finds the best-aligned $f(s, a)$ Note that there is an offset of 90 degrees in the relative angle between $f$ and $\phi$, that exists. This is because in this domain, the maximum value the agent can get is not 1, but zero. As a result, $f$ for the optimal action will not be perfectly aligned with $\phi$.

## 4 Experimental Setup

**Tasks** We evaluate our method in all the multi-goal RL benchmarks in Plappert et al. (2018), *Fetch* robot and *Shadow hand* manipulation tasks, as shown in Figure 9. For *Fetch* robot domain, the robot is required to move the object to a target position. In *Shadow hand* domain, the agent controls the joints of the hand's fingers and manipulates the objects in hand to a target pose. For all

tasks, we set discount factor $\gamma = 0.98$ based on Plappert et al. (2018). The details, including the reward functions, of these tasks can be found in the Section A.2.

**Baselines** We compare with two baselines, *UVFA (Monlithic)* (Andrychowicz et al., 2017; Schaul et al., 2015) and *UVFA (Bilinear)* proposed in Schaul et al. (2015). *UVFA (Monlithic)* is the most common multi-goal value function architecture that parametrizes the Q-function by a monolithic neural network. *UVFA (Bilinear)* is a bi-linear value function that decomposes the Q-function as $Q(s, a, g) = f(s, a)^\top \phi(g)$, where $f$ and $\phi$ denote two neural networks respectively. Our method uses a different bilinear decomposition and we refer to it as *BVN (Ours)*.

**Implementation** The baselines and our method are trained by deep deterministic policy gradient (DDPG) (Lillicrap et al., 2016) with hindsight experience replay (HER) (Andrychowicz et al., 2017). We use DDPG with HER as the base algorithm for our method and the baselines because it outperforms other recent Q-learning algorithms (Haarnoja et al., 2018; Fujimoto et al., 2018) on the task of learning a goal-conditioned Q-function. Comparison of DDPG against SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) is provided in Section B. For all baselines, the Q-function is implemented as a three layer neural network with 256 neurons per layer. In our method, $f$ and $\phi$ are both implemented as a 3-layer neural network with 176 neurons to match the total number of parameters of the baselines' Q-function. For *BVN (Ours)* and *UVFA (Bilinear)*, the output dimensions $f$, $\phi(s, g)$, and $\phi(g)$ are set as 16. The rest of hyperparameters for training are left in Section A.3.

**Evaluation metric** We ran each experiment with 5 different random seeds and report the mean (solid or dashed line) and 95%-confidence interval (shaded region) using bootstrapping method (DiCiccio & Efron, 1996). Each training plot reports the average success rate over 15 testing rollouts as a function of training epochs, where the goal is sampled in the beginning of each testing rollout. Each epoch consists of a fixed amount of data (see Section A.3 for details).

## 5 Results

Our experiments answer whether BVN improves sample efficiency and generalizes better to unseen goals. In Section 5.3, we also provide ablation studies that investigate the importance of several design decisions.

### 5.1 Sample Efficiency and Asymptotic Performance

Figure 4 compares the sample efficiency of our method, *BVN (Ours)*, against baselines on the *Fetch* tasks. The results show that our method has higher data efficiency in 4 out of 8 tasks and matches the baselines in other tasks. Our method also attains higher asymptotic performance in *Bin-place*. Figure 5 shows that in *Shadow hand*, our method improves the sample efficiency in 6 out of 8 tasks and is on par with the baselines for the other two tasks. These results suggest that BVN consistently improves sample efficiency of learning goal-conditioned policies.

### 5.2 Generalization to Unseen Goals

Since learning values of multiple goals is one of the challenges in multi-goal RL, we hypothesized that the improvements in data efficiency result from the fast adaptation of Q-function to new state-goal pairs encountered during the training time. Fast adaptation implies that the Q-function learned in the early stage of training requires fewer data to learn the values of new state-goal pairs appearing in the later stage. The resulting better critic improves the policy's performance faster, thus improving data-efficiency. To examine this hypothesis, we split the goal distributions at pre-training and fine-tuning time. We consider two scenarios in the *Fetch* robot domain: (1) left-to-right and (2) near-to-far. In near-to-far scenario, 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 pre-training and fine-tuning respectively (see Figure 6b). For left-to-right scenario, we split the goal space by an axis on the table (see Figure 6a) into the left and right half-spaces. At pre-training time, we disable HER in order to prevent the Q-function from receiving goals in fine-tuning goals by relabeling. We pre-train the baselines and our method until they have similar success rates (see Section A.3 for details).

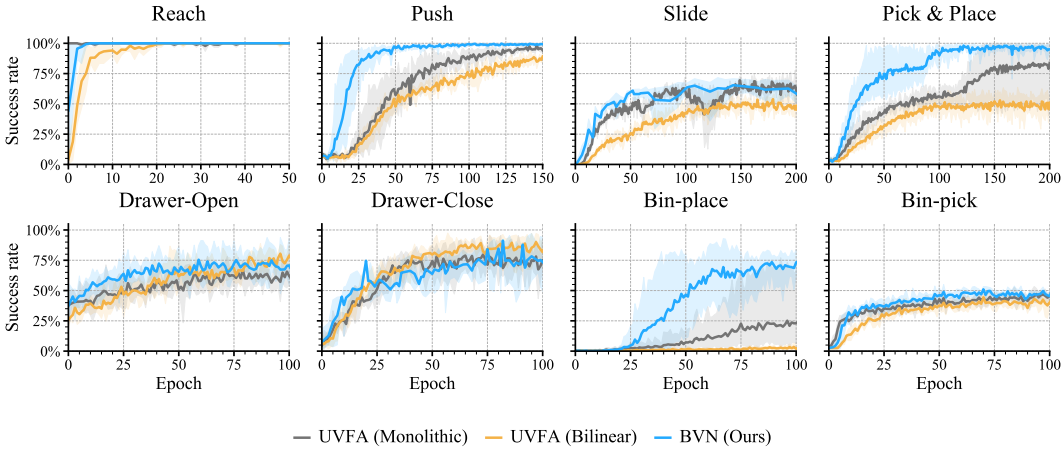



Figure 4: Learning curves on the *Fetch* gym environments, and the *Fetch-extension* (see Section A.4) task suit.

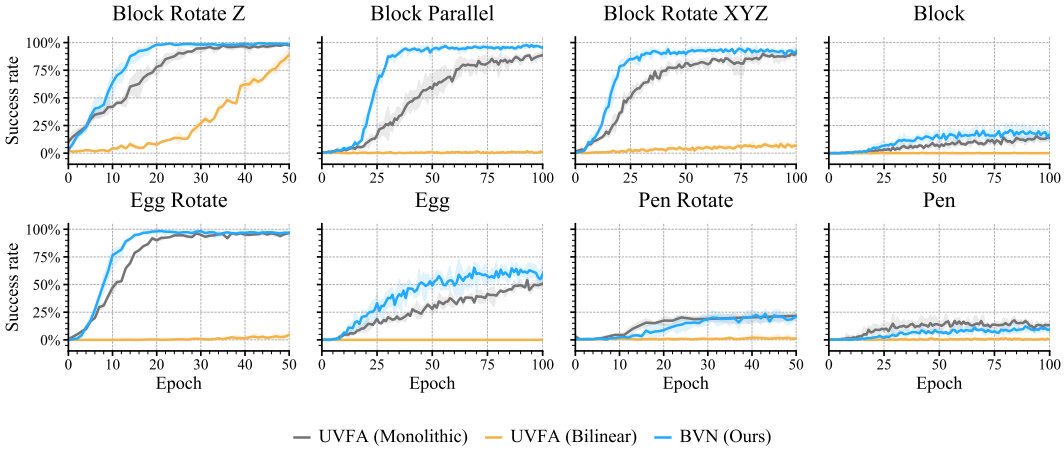



Figure 5: Learning curves on the *Shadow hand* dexterous manipulation suite. BVN learns quicker on six out of eight domains.

During fine-tuning, we initialize the Q-function and the policy with the best pre-trained models over 3 random seeds of each task at the fine-tuning time, and train the Q-function and the policy with HER.

We compared the fine-tuning performance of the baselines and BVN in Figure 6c. As $f$ network in BVN is goal-agnostic, we also compared the performance of BVN with freezing $f$ at the fine-tuning time. It can be seen that both *BVN (finetune $f$)* and *BVN (freeze $f$)* outperform *UVFA (Monlithic)*. Also, it is observed that *BVN (freeze $f$)* causes BVN performance to plateau, indicating both $f$ and $\phi$ are pivotal for approximating the Q-values.

### 5.3 Ablation Studies

**How does latent dimension affect our method?** We investigate the effect of the dimensionality of the latent space on the agent performance. Figure 7a shows the learning curve for BVN with 3, 4, 8 and 16 dimensional latent spaces. The performance consistently improves as latent dimensions increase, and even a BVN with just a 3-dimensional latent space still outperforms the monolithic UVFA baseline. Hence bilinear value decomposition produces consistent gain with respect to the monolithic UVFA baseline regardless of the latent dimensions, and benefits performance-wise from a larger latent space.

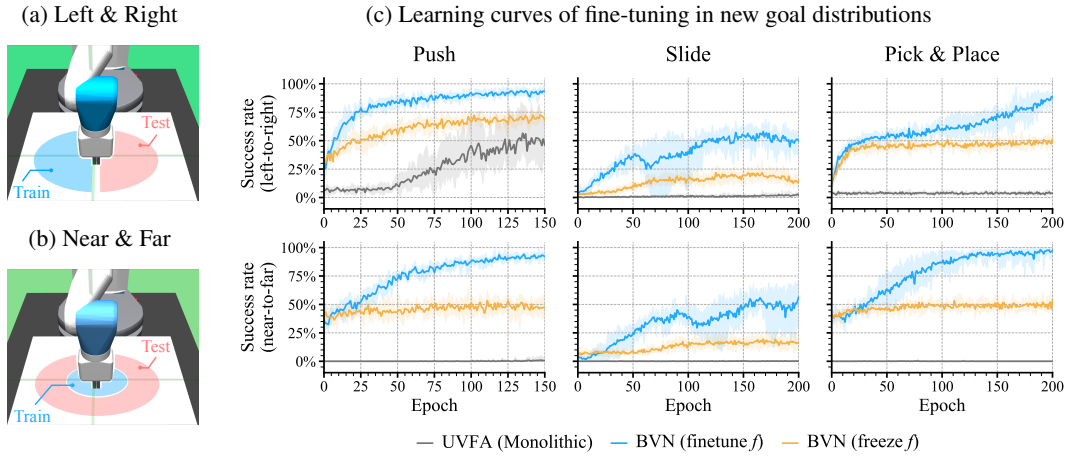



Figure 6: **(a & b)** Examples showing (a) how we split the goal distribution into left and right and (b) according to the radius to the center (near versus far). We pretrain with the blue regions, then finetune on the red regions. The state distribution remain identical to the vanilla *Fetch* environment. **(c)** Fine-tuning curves on unseen goals. BVN achieves better performance than the *UVFA (Monlithic)* on all tasks in both left-to-right and near-to-far adaptation scenarios. Freezing $f$ causes BVN performance to plateau, indicating that both $\phi$ and $f$ are needed for approximating the multi-goal value function.

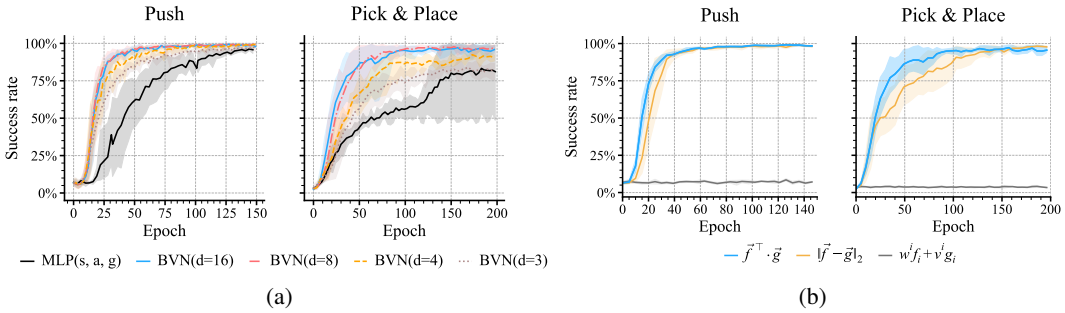



Figure 7: **(a)** Increasing the latent dimension of $f$ and $\phi$ improves our method's performance. Despite that, the result shows that a low dimensional latent space ($d = 3$) is sufficient to enable our method to outperform the baseline *UVFA (Monlithic)*. **(b)** Replacing the dot-product with an $\ell^2$ norm does not affect the final performance or the sample complexity. The initial performance is affected marginally. Whereas the linear combination is too restrictive.

**Dot product "$\cdot$" vs $\ell^2$ Metric $\|\cdot\|_2$** We compare two alternative ways to decompose the value function using the same variable grouping as the BVN. In the first variant, we replace the dot-product with an $\ell^2$ metric $\|\cdot\|_2$, and parameterize the Q function via $Q(s, a, g) = -\|f(s, a) - \vec{\phi}(s, g)\|_2$. The negative sign is added because multi-goal RL tasks typically assume the reward is within the range $[-1, 0]$. In the second variant we simply combine the two latent vectors linearly. The Q-function is formulated as a linear combination $Q(s, a, g) = w^i f_i(s, a) + v^i \phi_i(s, g)$ [1]

We set the latent dimension of the $\ell^2$-norm variant to 3 as it is the best one found through hyperparameter search (see Section A.4). The results shown in Figure 7b shows that both the bilinear and the metric decomposition attain good performance. The $\ell^2$ metric decomposition is marginally slower during the initial learning phase. The linear combination works poorly, likely due to the restricted expressivity.

[1] $x^i y_i$ implies a sum over the $i$ indices implicitly. This is the Einstein notation, or "einsum".

Table 1: Comparison between linear value approximators.

| **Reference** | **Parameterization** |
|---|---|
| Linear basis (Sutton & Barto; Littman et al.; Singh et al.) | $Q(s,a) = \sum_i c^i Q_i(s,a)$ |
| Proto-Value Function (Mahadevan & Maggioni, 2006; 2007) | $V(s) = \sum_i \lambda^i V_i(s)$ |
| General Value Approximator (GVF/Horde, Sutton et al. 2011) | $Q(s,a,\theta) = \theta^\top \phi(s,a)$ |
| Low-rank Bilinear UVFA (Schaul et al., 2015) | $Q(s,a,g) = f(s,a)^\top w(g)$ |
| Bilinear Value Network (**ours**) | $Q(s,a,g) = f(s,a)^\top \phi(s,g)$ |

**Does other inputs arguments grouping strategy work as well?** Lastly, we investigate if there are other viable ways to decompose the Q-function based on grouping the inputs $s, a, g$. An alternative hypothesis is that any grouping of the inputs arguments (i.e., $s, a, g$) could improve the sample efficiency. To test if this is the case, we compare our bilinear value decomposition scheme against two alternatives: $f(s,a)^\top \phi(a,g)$ and $f(s,a,g)^T \phi(g)$. The results in Figure 8 show that neither matches the performance of the bilinear value network, which suggests correct grouping does play a crucial role. It is important to process $s$ and $g$ together and duplicate the state input $s$ on both branches.

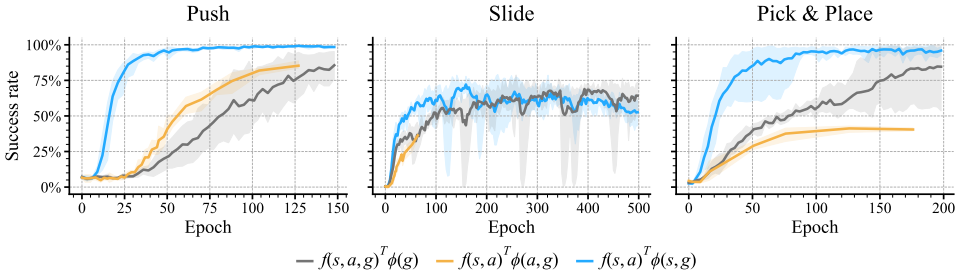



Figure 8: Comparison between alternative factorizations show that our bilinear value decomposition scheme, $f(s,a)^\top \phi(s,g)$, outperforms these alternatives. This suggests that this particular input grouping is crucial.

## 6 Discussion

A large number of prior works have explored ways to improve value function approximation by decomposing the function approximator into linear superposition of basis functions. Classic works (Sutton & Barto, 2018; Littman et al., 2001; Singh et al., 2003) approximate the value function in a set of linear basis. Horde (Sutton et al., 2011) uses a collection of value functions, each learned for a different task, as a representation of the state. Successor features (Dayan, 1993; Kulkarni et al., 2016; Borsa et al., 2019) factorize the value function as the dot product between state-action feature expectation and a vector of reward coefficients to the features. Laplacian reinforcement learning (Mahadevan & Maggioni, 2006; 2007) cast learning a set of basis value functions as a geometric problem, where one generalizes Fourier basis functions into higher dimension and non-euclidean manifolds via the diffusion operator $\nabla^2$. In the original universal value function approximator (UVFA, see Schaul et al. 2015), the authors included a linearized UVFA $Q(s,a,g) = f(s,a)^\top w_g$ where $w_g$ is a set of mixing coefficients that depends on the goal. As we will demonstrate in this work, this naïve linear decomposition reduces the approximator's expressivity and lowers the agent's performance. Therefore the more thoughtful approach is needed. To summarize prior works and their choice of parameterization, we compile key publications in Table 1, ordered by the year of publication. A more comprehensive list is available in Table 2 in the appendix.

Bilinear value networks take its name from the bilinear interaction between the two vector representations that resembles a bilinear form. Such exploitation of linearity goes back as early as 1989, with Ullman & Basri exploiting the linear basis of the special orthogonal group $SO(2)$ for pattern recognition, a scheme further extended to bilinearity by Freeman & Tenenbaum 1997; Tenenbaum & Freeman 2000. In research that naturally span multiple modalities such as visual question answering (VQA), the bilinear form is used as a means to "align" the visual representations with those

of language (Fukui et al., 2016; Yu et al., 2017; Hudson & Manning, 2018). When the input is just images, techniques such as *bilinear-pooling* induce a similar two-stream architecture (Kong & Fowlkes, 2017; Lin et al., 2018). Self-attention layers used in generative models (Zhang et al., 2019) and transformers Vaswani et al. (2017) are also bilinear forms, and in the latter case involve non-trivial processing through multiple layers. Moreover, multiplicative interactions are known to improve generalizability of neural networks and increase the expressivity of the network (Jayakumar et al., 2020). When combined with sinusoidal activation functions, even a shallow, multiplicative network can express an exponential number of basis functions (Fathony et al., 2020).

## 7 Conclusion

Our empirical evaluation shows significant sample efficiency gain when bilinear value networks is used with multi-goal RL tasks. Our ablation study on adaptation to unseen goals demonstrates that such gains likely result from faster adaptation of the value function. Exciting future work is to leverage the bilinear decomposition to facilitate transferring across different robots' morphology and goal spaces. One could keep the state and goal spaces fixed and train a single $\phi(s, g)$ that is shared between the different morphologies. The difference between morphologies could be accommodated by different $f$ (e.g., different manipulators for the same task). Also, one can pre-train a shared $f$ for the same robot and train $\phi$ for different tasks (i.e., goal spaces). In addition, we further discuss the additional implications of our experimental results in Section C of appendix.

### Acknowledgments

The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC and consultation resources that have contributed to the research results reported within this paper.

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-21-1-0328. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This work is also supported in by the NSF AI Institute for Artificial Intelligence and Fundamental Interactions and MIT-IBM Research Grant.

## 8 Reproducibility Statement

Please refer to Section A.4.

## References

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016. 1

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 2017. 1, 4, 5, 13

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. June 2016. URL http://arxiv.org/abs/1606.05312. 1

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. In *Proceedings of the International Conference on Learning Representations*, 2019. 8, 13

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Comput.*, 5(4):613–624, 1993. ISSN 0899-7667. 1, 8

Thomas J DiCiccio and Bradley Efron. Bootstrap confidence intervals. *Statistical science*, 11(3): 189–228, 1996. 5

Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *Proceeding of The International Conference on Learning and Representation*, September 2020. URL https://openreview.net/pdf?id=OmtmcPkkhT. 9

W T Freeman and J B Tenenbaum. Learning bilinear models for two-factor problems in vision. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 554–560, June 1997. doi: 10.1109/CVPR.1997.609380. URL http://dx.doi.org/10.1109/CVPR.1997.609380. 8

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *Proceedings of the International Conference on Machine Learning*, 2018. 2, 3, 5, 16

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of The Conference on Empirical Methods in Natural Language Processing*, June 2016. 9

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 2018. 3, 5, 16

Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *Proceedings of The International Conference on Learning and Representation*, 2018. 9

Siddhant M Jayakumar, Wojciech M Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osidnero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *Proceedings of The International Conference on Learning and Representation*, 2020. 9

Michael I Jordan and David E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive science*, 1992. 1

Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1993. 1, 2

Shu Kong and Charless Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 365–374, 2017. URL http://openaccess.thecvf.com/content_cvpr_2017/html/Kong_Low-Rank_Bilinear_Pooling_CVPR_2017_paper.html. 9

Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. Deep successor reinforcement learning. *ArXiv*, abs/1606.02396, 2016. 8, 13

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. In *Proceedings of the International Conference on Learning Representations*, 2016. 2, 4, 5

Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear convolutional neural networks for fine-grained visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(6):1309–1322, June 2018. ISSN 0162-8828. doi: 10.1109/TPAMI.2017.2723400. URL http://dx.doi.org/10.1109/TPAMI.2017.2723400. 9

M Littman, R Sutton, and Satinder Singh. Predictive representations of state. 2001. 8, 13

Sridhar Mahadevan and Mauro Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. *Advances in Neural Information Processing Systems*, 2006. 8, 13

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 2007. 8, 13

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *Proceedings of the International Conference on Representation Learning*, 2017. 13

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. 3

Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2146–2153. IEEE, 2017. 1

Keiran Paster, Sheila A. McIlraith, and Jimmy Ba. Planning from pixels using inverse dynamics models. In *International Conference on Learning Representations*, 2021. 1

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *International Conference on Learning Representations*, 2018. 1

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2020. 16

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. 2, 4, 5, 13, 14, 16, 17

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the International Conference on Machine Learning*, 2015. 1, 3, 4, 5, 8, 13, 14

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014. 3

Satinder P Singh, Michael L Littman, Nicholas K Jong, David Pardoe, and Peter Stone. Learning predictive state representations. *Proceedings in the International Conference on Machine Learning*, 2003. 8, 13

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, 13

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 *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2011. 8, 13

J B Tenenbaum and W T Freeman. Separating style and content with bilinear models. *Neural Comput.*, 12(6):1247–1283, June 2000. ISSN 0899-7667, 1530-888X. doi: 10.1162/089976600300015349. URL http://dx.doi.org/10.1162/089976600300015349. 8

Shimon Ullman and Ronen Basri. *Recognition by Linear Combination of Models*. Defense Technical Information Center, 1989. doi: 10.21236/ada224268. URL https://play.google.com/store/books/details?id=r1OJNwAACAAJ. 8

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017. 9

Daniel M Wolpert, Zoubin Ghahramani, and Michael I Jordan. An internal model for sensorimotor integration. *Science-AAAS-Weekly Paper Edition*, 1995. 1

Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 1821–1830, 2017. URL http://openaccess.thecvf.com/content_iccv_2017/html/Yu_Multi-Modal_Factorized_Bilinear_ICCV_2017_paper.html. 9

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7354–7363. PMLR, 2019. URL https://proceedings.mlr.press/v97/zhang19d.html. 9

Table 2: Comprehensive list of linear value decomposition schemes from the literature.

| Reference | Parameterization |
|---|---|
| Linear basis (Sutton & Barto; Littman et al.; Singh et al.) | $Q(s,a) = \sum c_i Q^i(s,a)$ |
| Proto-Value Function (Mahadevan & Maggioni, 2006; 2007) | $V(s) = \sum \lambda_i V^i(s)$ |
| General Value Approximator (GVF/Horde, Sutton et al. 2011) | $Q(s,a,\theta) = \theta^\top \phi(s,a)$ |
| Successor Representation (SR, Kulkarni et al. 2016) | $Q(s,a) = M(s,a,s')R(s')$ |
| Monolithic MLP UVFA (Schaul et al.; Andrychowicz et al.) | $\text{MLP}(s,a,g)$ |
| Recurrent UVFA (Mirowski et al., 2017) | $\text{RNN}(z_s,a,z_g)$, $z_s, z_g$ are embeddings |
| Low-rank Bilinear UVFA (Schaul et al., 2015) | $Q(s,a,g) = f(s,a)^\top w(g)$ |
| USFA (Borsa et al., 2019) | $Q(s,a,\vec{z}) = f(s,a,\vec{z})^\top \vec{w}$ |
| Bilinear Value Network (**ours**) | $Q(s,a,g) = f(s,a)^\top \phi(s,g)$ |

# A Appendix

## A.1 Details On Previous Linear Decomposition Schemes

We present the comparison of parameterizations in Table 2.

## A.2 Implementation Details of Tasks

All experiments in Section 5.1 happen on the standard object and dexterous manipulation tasks from the gym robotics suite (Plappert et al., 2018).

We briefly summarize the specifications of the tasks in the list below and illustrate these tasks in Figure 9:

Table 3: Environment Specifications.

| | Fetch Robot | | | | Shadow Hand | | |
|---|---|---|---|---|---|---|---|
| Task | $\mathcal{S}$ | $\mathcal{A}$ | $\mathcal{G}$ | Task | $\mathcal{S}$ | $\mathcal{A}$ | $\mathcal{G}$ |
| Reach | $\mathbb{R}^{10}$ | $\mathbb{R}^3$ | $\mathbb{R}^3$ | BlockRotateZ | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| Push | $\mathbb{R}^{25}$ | $\mathbb{R}^3$ | $\mathbb{R}^3$ | BlockRotateParallel | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| PickAndPlace | $\mathbb{R}^{25}$ | $\mathbb{R}^4$ | $\mathbb{R}^3$ | BlockRotateXYZ | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| Slide | $\mathbb{R}^{25}$ | $\mathbb{R}^3$ | $\mathbb{R}^3$ | BlockRotate | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| DrawerOpen | $\mathbb{R}^{25}$ | $\mathbb{R}^3$ | $\mathbb{R}^3$ | Egg | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| DrawerClose | $\mathbb{R}^{25}$ | $\mathbb{R}^3$ | $\mathbb{R}^3$ | EggRotate | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| | | | | Pen | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |
| | | | | PenRotate | $\mathbb{R}^{61}$ | $\mathbb{R}^{20}$ | $\mathbb{R}^7$ |

In addition to (Plappert et al., 2018), we also use the tasks from Fetch extension[2]. For left/right environments used in Section 5.2, 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.3 Training Hyperparameters

**Fetch** We adapted the hyperparameters in (Plappert et al., 2018) for training in a single desktop. In Plappert et al. (2018), the agent is trained by multiple workers collecting data in parallel, where each worker collects rollouts using the same policy with different random seeds. 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

[2]https://github.com/geyang/gym-fetch

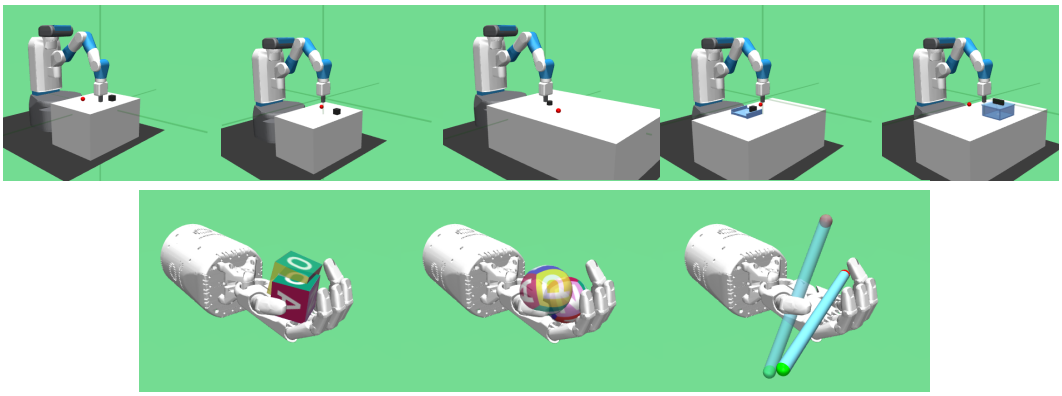


Figure 9: Renderings of the *fetch robot* and *shadow hand* domains considered in this work. Left-to-right: (**top**) Fetch push, pick-and-place, slide, bin-pick, box-open; (**bottom**) Shadow hand block, egg, pen. For Fetch, the goal is to move the gray object to the red dot by controlling the arm. For Shadow hand, the objective is to reorient the object (e.g., block, egg, and pen) to the desired poses rendered by the transparent object.

Table 4: The success rates of each method in the pretrained tasks used in Section 5.2.

| | Left | | | Near | | |
|---|---|---|---|---|---|---|
| | Push | Pick & Place | Slide | Push | Pick & Place | Slide |
| UVFA (Monolithic) | 96% | 50% | 81% | 93% | 91% | 96% |
| BVN | 96% | 59% | 48% | 95% | 96% | 90% |

- `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[3].

**Adaptation experiments (Section 5.2)** We present the success rates after pretraining in Table 4. Note that in the pretraining stage of each method and task, we trained the agents with 5 different random seeds and took the best-performing models as the initialization at the fine-tuning time.

### A.4 Implementation Details

**Q-function architecture** The monolithic network baselines in this paper represents the Q-function as a multi-layer perceptron (MLP) network that takes in $s$, $a$, and $g$ concatenated together as a single vector. The bilinear UVFA (Schaul et al., 2015) uses an MLP $f$ that takes the concatenated $s$ and $a$ as inputs, and an MLP $\phi$ using $g$ as inputs. Our biliniear value network on the other hand, concatenate $[s, a]$ together as the input to $f$, and $[s, g]$ as the inputs of $\phi$.

**Policy architecture** We use an MLP as the policy network for all the methods.

For our method and its variants presented in Section 5.3 , we use the following nomenclature to denote them:

1. **Dot product** This is our proposed method. We set the outputs' dimension of $f$ and $\phi$ as 16.

[3]https://github.com/TianhongDai/hindsight-experience-replay

2. **2-norm** We parametrize Q-function as $Q(s, a, g) = \|f(s, a) - \phi(s, g)\|_2$. We use set the dimension of the outputs from $f$ and $\phi$ as 3. Figure 10 shows that $d = 3$ works the best for 2-norm.

3. **concat** We concatenate $f(s, a)$ and $\phi(s, g)$, feed the concatenated vectors to an 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 $\phi$ are 16.

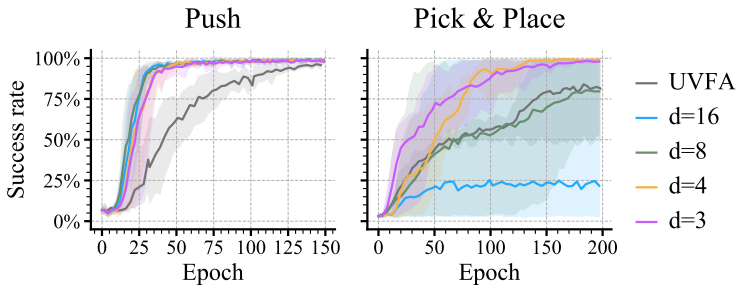



Figure 10: 3-dimensional latent space works the best for 2-norm variant (see Section A.4).

### A.5 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 $\phi(s, g)$, one potential benefit on multi-goal RL is reusing $\phi$ for transferring to tasks with the same (or similar) task objectives (i.e., reward specification). Reusing $\phi(s, g)$ 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[4], and measure the performance of transferring amongst these tasks. In transferring an agent from a source task to a target task, we initialize *UVFA (Monlithic)* and *BVN (no reset)* agent's policy and Q-function by the neural networks' weights learned in the source task and fine-tune them in the target task. For *BVN (reset f)*, we reinitialize $f$ and fine-tune $\phi$ when training them in the target task. Figure 11 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 *BVN (reset f)* attains higher success rates at convergence than the baselines and *BVN (no reset)* in 4 out 6 tasks (i.e., "Push to Slide", "Pick & Place to Push", "Push to Pick & Place", and "Pick & Place to Slide").

The limited efficiency gain could result from that $\phi$ is dependent on states $s$ and thus $\phi$ cannot be effectively reused in new tasks. 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 $\phi(s, g)$ should direct the agent to lift the object. On the other hand, in "Slide", the $\phi(s, g)$ is expected to guide the agent to push the object to slide.

[4]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.

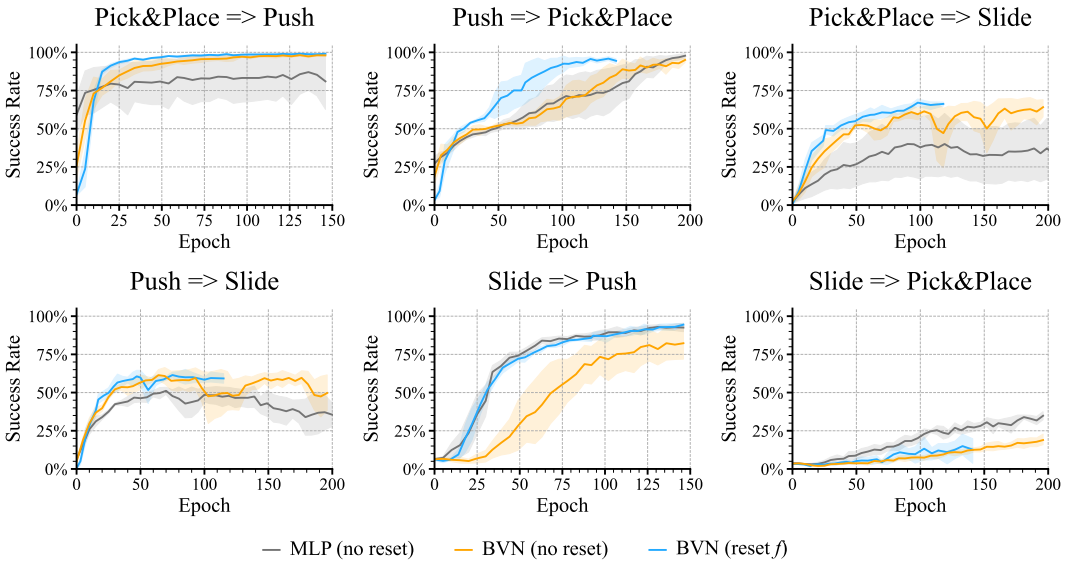



Figure 11: The two-stream architecture of the BVN allows us to improve transfer by resetting $f$. Resetting $f$ and fine-tuning $\phi$ achieve higher success rate than the baselines when transferring between push and pick & place, and to slide.

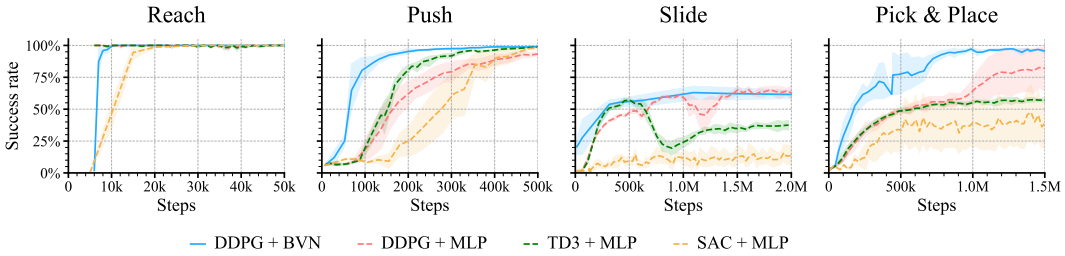



Figure 12: We show that our BVN improves the sample efficiency over the best baseline, *DDPG + MLP*, in 4 commonly used environments in multi-goal RL.

## B Comparison Against SOTA Algorithms

Bilinear value networks achieves State-Of-The-Art (SOTA) performance in common multi-goal reinforcement learning tasks (Plappert et al., 2018). In figure 12, we compare bilinear value networks against two strong baselines: soft actor critic (SAC, see Haarnoja et al. 2018) and twin-delayed deep deterministic policy gradient (TD3, see Fujimoto et al. 2018). We reuse the hyperparameters reported in Fujimoto et al. 2018) and implement TD3 in our codebase. As for SAC, we use the implementation in this codebase[5]. Figure 12 shows the performance of each baseline algorithm and our method (*DDPG + BVN*). We show that our *DDPG + BVN* outperforms the best baseline, and find that *DDPG* works the best among all the *MLP* variants. Our findings align with (Pitis et al., 2020) and the documentation in its released code[6]. Pitis et al. (2020) also found that DDPG works better than TD3 and SAC in multi-goal RL domains.

## C Additional Discussion

**Why does other parameter grouping schemes fail?** In Section 5.3, we showed that $f(s, a, g)^T \phi(g)$ 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

[5]https://github.com/spitis/mrl

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 (Monlithic)*. 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 5.3 demonstrated that using 2-norm to reduce $f(s, a)$ and $\phi(s, g)$ into Q-value matches the performance of our BVN with even lower dimensions of $Z_f$ and $Z_\phi$. 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 $\|f(s, a) - \phi(s, g)\|$ 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 $\phi(s, g)$ in the new task and re-train $f(s, a)$. The results in Section A.5 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 $\phi(s, g)$ conditions on states $s$. Consequently, environment dynamics shifts could affect the representations of $\phi$. We, nevertheless, showed that states variables are crucial in $\phi$. One exciting avenue is to disentangle the representations for keeping the rich expressivity as well as desired modularity for transferring across tasks.