COMPOSITIONAL TASK GENERALIZATION WITH MODULAR SUCCESSOR FEATURE APPROXIMATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, the Successor Features and Generalized Policy Improvement (SF&GPI) framework has been proposed as a method for learning, composing and transferring predictive knowledge and behavior. SF&GPI works by having an agent learn predictive representations (SFs) that can be combined for transfer to new tasks with GPI. However, to be effective this approach requires state features that are useful to predict, and these state-features are typically hand-designed. In this work, we present a novel neural network architecture, "Modular Successor Feature Approximators" (MSFA), where modules both discover what is useful to predict, and learn their own predictive representations. We show that MSFA is able to better generalize compared to baseline architectures for learning SFs and a modular network that discovers factored state representations.

1 INTRODUCTION

Consider a household robot that needs to learn tasks including picking up dirty dishes and cleaning up spills. Now consider that the robot is deployed and encounters a table with both a spill and set of dirty dishes. Ideally this robot can *combine* its training behaviors to both clean up the spill and pickup the dirty dishes. We study this aspect of generalization: combining knowledge from multiple tasks.

Combining knowledge from multiple tasks is challenging because it is not clear how to synthesize either the behavioral policies or the value functions learned during training. This challenge is exacerbated when an agent also needs to generalize to novel appearances and environment configurations. Returning to our example, our robot might need to additionally generalize to both novel dirty dishes on the table and to novel arrangements of the chairs around the table.

Successor features (SFs) and Generalized Policy Improvement (GPI) provide a mechanism to combine knowledge from multiple training tasks (Barreto et al., 2017; 2020). SFs are predictive representations that estimate how much state-features (known as "cumulants") will be experienced given a behavior. By assuming that reward has a linear relationship between cumulants and a task vector, an agent can efficiently *compute* how much reward it can expect to obtain from a given behavior. If the agent knows multiple behaviors, it can leverage GPI to compute which behavior would provide the most reward (see Figure 2 for an example). However, SF&GPI commonly assume hand-designed cumulants and don't have a mechanism for generalizing to novel environment configurations.

Modular architectures are a promising method for generalizing to distributions outside of the training distribution (Goyal et al., 2019; Madan et al., 2021). Recently, Carvalho et al. (2021a) presented "FARM" and showed that learning multiple state modules enabled generalization to environments with unseen environment parameters (e.g. to larger maps with more objects). In this work, we hypothesize that modules can further be leveraged to discover state-features that are useful to predict.

We present "Modular Successor Feature Approximators" (MSFA), a novel neural network for discovering, composing, and transferring predictive knowledge and behavior via SF&GPI. MSFA is composed of a set of modules, which each learn their own state-features and corresponding predictive representations. We show that an architectural bias of having state-features be produced by their own modules is useful inductive bias for reward-driven discovery of useful features to use with SF&GPI.

Our core contribution is showing that an inductive bias for modularity can enable reward-driven discovery of state-features that are useful for zero-shot transfer with SF&GPI. We exemplify this

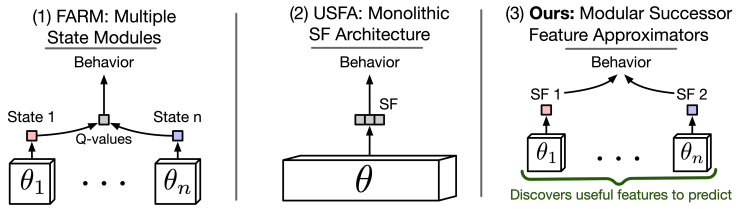

Figure 1: (1) FARM learns multiple state modules. This promotes generalization to novel environments. However, it has no mechanism for combining task solutions. (2) USFA learns a single monolothic architecture for predicing SFs and can combine task solutions. However, it relies on hand-designed state features and has no mechanism for generalization to novel environment configurations. (3) We combine the benefits of both. By leveraging modules, we enable reward-driven discovery of state features that are useful to predict. These form the basis of their own predictive representations (SFs) and enables combining task solutions in novel environments.

with a simple state-feature discovery method presented in Barreto et al. (2018) where the dot-product between state-features and a task vector is regressed to environment reward. This method enabled transfer with SF&GPI in a continual learning setting but had limited success in the zero-shot transfer settings we study. While there are other methods for state-feature discovery, they add training complexity with mutual information objectives (Hansen et al., 2019) or meta-gradients (Veeriah et al., 2019). With MSFA, by adding *only an architectural bias for modularity*, we discover state-features that (1) support zero-shot transfer competitive with hand-designed features, and (2) enable zero-shot transfer in visually diverse, procedurally generated environments. We are hopeful that our architectural bias can be leveraged with other discovery methods in future work.

2 Related Work on Generalization in RL

Hierarchical RL (HRL) is one dominant approach for tackling compositional generalization. The basic idea is that one can sequentially compose policies in time by having a "meta-policy" that sequentially activates "low-level" policies for protracted periods of time. By leveraging hand-designed or pre-trained low-level policies, one can generalize to longer instructions (Oh et al., 2017; Corona et al., 2020), to new instruction orders (Brooks et al., 2021), and to novel subtask graphs (Sohn et al., 2020; 2022). We differ in that we focus on combining policies *concurrently* in time as opposed to sequentially in time. To do so, we develop a modular neural network for the SF&GPI framework.

SFs are predictive representations that represent the current state as a summary of the *successive* features to follow (see §3 for a formal definition). By combining them with Generalized Policy Improvement, researchers have shown that they can transfer behaviors across object navigation tasks (Borsa et al., 2019; Zhang et al., 2017; Zhu et al., 2017), across continuous control tasks (Hunt et al., 2019), and within an HRL framework (Barreto et al., 2019). However, these works tend to require hand-designed cumulants which are cumbersome to design for every new environment. In our work, we integrate successor features with Modular RL to facilitate reward-driven discovery of cumulants and improve successor feature learning.

Modular RL (MRL) (Russell & Zimdars, 2003) is a framework that achieves compositional generalization through the composition of value functions. Early work dates back to (Singh, 1992), who had a mixture-of-experts system select between separately trained value functions. Since then, MRL has been applied to generalize across robotic morphologies (Huang et al., 2020), to novel task-robot combinations (Devin et al., 2017; Haarnoja et al., 2018), and to novel language instructions (Logeswaran et al., 2021). Our architecture, MSFA, is the first to integrate MRL with SF&GPI. This integration enables composing task solutions in novel environment configurations.

Generalizing to novel environment configurations is typically studied with procedurally generated environments (Cobbe et al., 2020; Chevalier-Boisvert et al., 2018; Samvelyan et al., 2021). Some approaches to this problem leverage techniques from supervised learning such as regularization, curriculum strategies, hyper-parameter tuning, and using self-supervised objectives (Cobbe et al., 2020; Laskin et al., 2020; Igl et al., 2019; Anand et al., 2021). Most relevant to us, Carvalho et al.

(2021a) recently showed that a modular architecture for learning a factored state representation, Feature-Attending Recurrent Modules (FARM), could improve generalization in procedurally generated object-centric environments. FARM exploited modularity to combine state representations. We instead exploit modularity to combine value functions and show that this improves an agent's ability to both combine task solutions and generalize to novel environment configurations.

3 Problem Setting and Background

We study a reinforcement learning agent's ability to transfer knowledge between tasks in an environment. During training, the experiences $n_{\texttt{train}}$ tasks $\mathbb{M}_{\texttt{train}} = \{\mathcal{M}_i\}_{i=1}^{n_{\texttt{train}}}$, sampled from a training distribution $p_{\texttt{train}}(\mathcal{M})$. During testing, the agent is evaluated on $n_{\texttt{test}}$ tasks, $\{\mathcal{M}_i\}_{i=1}^{n_{\texttt{test}}}$, sampled from a testing distribution $p_{\texttt{test}}(\mathcal{M})$. Each task $\mathcal{M}_i$ is specified as Partially Observable Markov Decision Process (POMDP), $\mathcal{M}_i = \langle \mathcal{S}, \mathcal{A}, \mathcal{X}, R, p, f_x \rangle$. Here, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{X}$ are the state, action, and observation spaces. $p(\cdot|s_t, a_t)$ specifies the next-state distribution based on taking action a_t in state s_t, and $f_x(s_t)$ maps the underlying environment state to an observation x_t. We focus on tasks where rewards are parameterized by a task vector w, i.e. $r_t^w = R(s_t, a_t, s_{t+1}, w)$ is the reward obtained for transition (s_t, a_t, s_{t+1}) given task vector w. We want to obtain a behavioral policy $\pi(s)$ that best maximises the expected reward it will obtain when taking an action a_t at a state s_t: $Q_t^{\pi,w} = Q^{\pi,w}(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t^w\right]$.

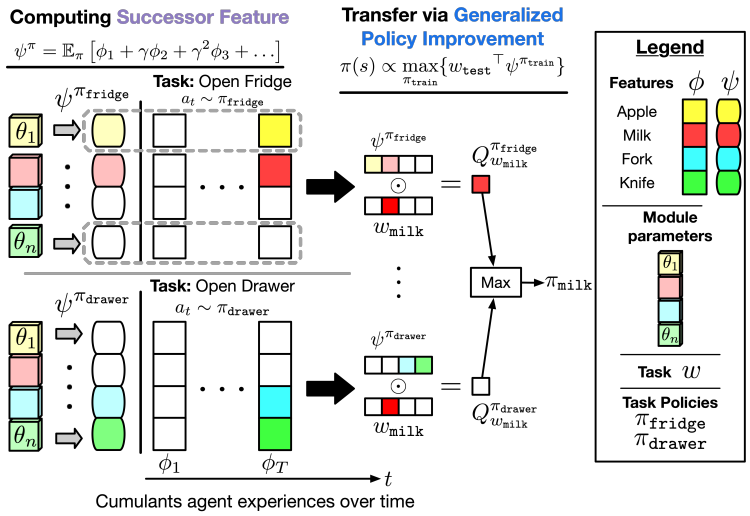

Figure 2: **High-level diagram of how Modular Successor Feature Approximators can be leveraged for transfer with SF&GPI**. During training, the agent learns policies for two tasks: "open drawer" and "open fridge". Each task leads the agent to experience different "cumulants" ϕ—e.g., "fork" during "open drawer" and "apple" during "open fridge". The agent leverages different modules to learn these ϕ and to estimate SFs, ψ. For example, across tasks, module 1 (θ_1) can estimate SFs for apple cumulants. Module SFs are combined to form the SF for a policy. When the agent wants to transfer its knowledge to a test task—e.g., "get milk"—it can compute Q-values for that task as a dot-product with the SFs of each training task. The highest Q-value is then used to select actions.

Transfer with SF&GPI. In order to leverage successor features (SFs) (Barreto et al., 2017), one assumes an agent has access to state features known as "cumulants", $\phi_t = \phi(s_t, a_t, s_{t+1})$. Given a behavioral policy $\pi(a|s)$, SFs are a type of value function that use cumulants as pseudo-rewards:

$$\psi_t^\pi = \psi^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i \phi_{t+i}\right] \tag{1}$$

If reward is (approximately) $r_t^w = \phi_t^\top w$, then action-values can be decomposed as $Q_t^{\pi,w} = \psi_t^{\pi\top} w$. This is interesting because it provides an easy way to *reuse* task-agnostic features ψ_t^π for new tasks.

We can re-use the SFs we've learned from training tasks $\mathbb{M}_{\texttt{train}}$ for transfer with GPI. Assume we have learned (potentially optimal) policies $\{\pi_i\}_{i=1}^{n_{\texttt{train}}}$ and their corresponding SFs $\{\psi^{\pi_i}(s,a)\}_{i=1}^{n_{\texttt{train}}}$.

Given a test task $w_{\texttt{test}}$, we can obtain a new policy with GPI in two steps: (1) compute Q-values using the training task SFs (2) select actions using the highest Q-value. This operation is summarized as follows:

$$\pi(s_t; w_{\texttt{test}}) \in \arg\max_{a \in \mathcal{A}} \max_{i \in \{1,\ldots,n_{\texttt{train}}\}} \{Q_t^{\pi_i, w_{\texttt{test}}}\} = \arg\max_{a \in \mathcal{A}} \max_{i \in \{1,\ldots,n_{\texttt{train}}\}} \{\psi_t^{\pi_i \top} w_{\texttt{test}}\} \quad (2)$$

This is useful because the GPI theorem states that π will perform as well as all of the training policies, i.e. that $Q^{\pi, w_{\texttt{test}}}(s,a) \geq \max_i Q^{\pi_i, w_i}(s,a) \forall (s,a) \in (\mathcal{S} \times \mathcal{A})$ (Barreto et al., 2017).

SF&GPI enable transfer by exploiting structure in the RL problem: a policy that maximizes a value function is guaranteed to perform at least as well as the policy that defined that value function. However, SF&GPI relies on combining a fixed set of SFs. Another form of transfer comes from "Universal Value Function Approximators" (UVFAs) (Schaul et al., 2015), which add the task-vector w as a parameter to a Q-approximator parameterized by θ, $Q_\theta(s,a,w)$. If Q_θ is smooth with respect to w, then Q_θ should generalize to test tasks nearby to train tasks in task space. Borsa et al. (2019) showed that one could combine the benefits of both with "Universal Successor Feature Approximators". Since rewards r^w, and therefore task vectors w, reference deterministic task policies π_w, one can parameterize successor feature approximators with task-vectors $\tilde{\psi}^{\pi_w} = \tilde{\psi}^w \approx \psi_\theta(s,a,w)$. However, USFA assumed hand-designed cumulants. We introduce an architecture for reward-driven discovery of cumulants and improved function approximation of universal successor features.

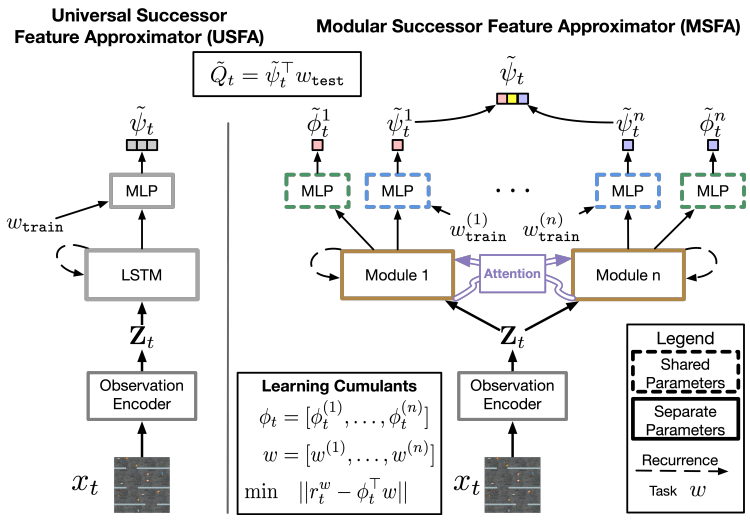

Figure 3: **Left: Universal Successor Feature Approximator (USFA)** learns a single, monolithic successor feature estimator that uses **hand-designed** cumulants. **Right: Modular Successor Feature Approximator (MSFA)** learns a set of successor feature modules, each with their own functions for (a) updating module-state, (b) computing cumulants, and (c) estimating successor features. Modules then share information with an attention mechanism. We hypothesize that isolated module computations facilitate learning cumulants that suppot generalization with GPI.

4 Modular Successor Feature Approximators

We propose a new architecture *Modular Successor Feature Approximators* (MSFA) for approximating SFs, shown in Figure 3. Our hypothesis is that learning disentangled cumulants and SFs will improve compositional generalization with SF&GPI. We accomplish this by learning n disentangled state modules $\{s_{\theta_k}\}_{k=1}^n$ that evolve with independent parameters θ_k and have sparse inter-module information flow. We produce disentangled cumulants $\{\tilde{\phi}_t^{(k)}\}_{k=1}^n$ and SFs $\{\tilde{\psi}_t^{\pi,k}\}_{k=1}^n$ by having their computations depend **only** on individual module-states. For example, a cumulant may correspond to information about apples, and would be a function **only** of the module representing state information related to apples. This is in contrast to prior work, which learns a single monolithic prediction module for computing cumulants and SFs.

The rest of section is structured as follows. In section §4.1, we derive Modular Successor Feature Learning within the Modular RL framework. We then describe our architecture, MSFA, for learning modular successor features in §4.2. In §4.3, we describe how to generate behavior with MSFA. Finally, we describe the learning algorithm for MSFA in section §4.4.

4.1 Modular Successor Feature Learning

Following the Modular RL framework (Russell & Zimdars, 2003), we assume that reward has an additive structure $R(s_t, a_t, s_{t+1}) = \sum_k R^k(s_t, a_t, s_{t+1}) = \sum_k R_t^{(k)}$, where $R_t^{(k)}$ is the reward of the k-th module. We enforce that every module decomposes reward into an inner-product between its own task-description $w^{(k)}$ and task-agnostic cumulants $\phi^{(k)} \in \mathbb{R}$: $R_t^{(k)} = \phi_t^{(k)} \cdot w^{(k)}$. Here, we simply break up the task vector into n pieces so individual modules are responsible for subsets of the task vector. This allows us to decompose the action-value function as

$$Q^\pi(s_t, a_t, w) = \sum_{k=1}^{n} Q^{\pi,k}(s_t, a_t, w^{(k)}) = \sum_{k=1}^{n} \psi^{\pi,k}(s_t, a_t) \cdot w^{(k)} \tag{3}$$

where we now have *modular SFs* $\{\psi^{\pi,k}(s,a)\}_{k=1}^n$ (see the Appendix for a derivation). Rather than hand-designing modules or cumulants, we aim to discover them from the environment reward signal.

4.2 Architecture

We learn a set of modules with states $\mathbb{S}_t = \{s_t^{(k)}\}_{k=1}^n$. They update at each time-step t with the observation x_t, the previous module-state $s_{t-1}^{(k)}$, and information from other modules $A_\theta(s_t^{(k)}, \mathbb{S}_{t-1})$. Following prior work (Santoro et al., 2018; Goyal et al., 2019; Carvalho et al., 2021a), we have A_θ combine transformer-style attention (Vaswani et al., 2017) with a gating mechanism (Parisotto et al., 2020) to enforce that inter-module interactions are sparse. Since A_θ is not the main contribution of this paper, we describe these computations in more detail with our notation in the Appendix. We summarize the high-level update below.

$$s_t^{(k)} = s_{\theta_k}(x_t, s_{t-1}^{(k)}, A_\theta(s_{t-1}^{(k)}, \mathbb{S}_{t-1})) \tag{4}$$

We learn disentangled cumulants and SFs by having sets of cumulants and SFs depend on individual module-states. Module cumulants depend on the module-state from the current and next time-step. Module SFs depend on the current module-state and on their subset of the task description. We summarize this below:

$$\tilde{\phi}_t^{(k)} = \phi_\theta(s_t^{(k)}, a_t, s_{t+1}^{(k)}) \qquad \tilde{\psi}_t^{w,k} = \psi_\theta(s_t^{(k)}, a_t, w^{(k)}) \tag{5}$$

We highlight that cumulants share parameters but differ in their input. This suggests that the key is not having cumulants and SFs with separate parameters but that they are functions of sparse subsets of state (rather than all state information). We show evidence for this hypothesis in Figure 6.

We concatenate module-specific cumulants and SFs to form the final outputs: $\tilde{\phi}_t = \left[\tilde{\phi}_t^{(1)}, \ldots, \tilde{\phi}_t^{(n)}\right]$ and $\tilde{\psi}_t^w = \psi_\theta(s_t, a_t, w) = \left[\tilde{\psi}_t^{w,1}, \ldots, \tilde{\psi}_t^{w,n}\right]$. Note that cumulants update with the module-state from the next time-step. This is fine since cumulants are only used during learning.

4.3 Behavior

During **training**, actions are selected in proportion to Q-values computed using task SFs as $\pi(s_t, w) \propto \tilde{Q}(s_t, a, w) = \psi_\theta(s_t, a, w)^\top w$. In practice we use an epsilon-greedy policy, though one can use other choices. During **testing**, we compute policies with GPI as $\pi(s_t, w_{\texttt{test}}) \in \arg\max_a \max_{z \in \mathbb{M}_{\texttt{train}}} \{\psi_\theta(s_t, a, z)^\top w_{\texttt{test}}\}$, where $\mathbb{M}_{\texttt{train}}$ are train task vectors.

4.4 Learning Algorithm

MSFA relies on three losses. The first loss, $\mathcal{L}_Q$, is a standard Q-learning loss, which MSFA uses to learn optimal policies for the training tasks. The main difference here is that MSFA uses a particular

parameterization of the Q-function $Q^{\pi_w,w}(s,a) = \psi^{\pi_w}(s,a)^\top w$. The second loss, $\mathcal{L}_\psi$, is an SF learning loss, which we use as a regularizer to enforce that the Q-values follow the structure in the reward function $r_t^w = \phi_t^\top w$. For this, we again apply standard Q-learning but using SFs as value functions and cumulants as pseudo-rewards. The final loss, $\mathcal{L}_\phi$ is a loss for learning cumulants that grounds them in the environment reward signal. The losses are summarised as follows

$$\mathcal{L}_Q = ||r_t + \gamma\psi_\theta(s_{t+1}, a', w)^\top w - \psi_\theta(s_t, a_t, w)^\top w||^2 \tag{6}$$

$$\mathcal{L}_\psi = ||\tilde{\phi}_t + \gamma\psi_\theta(s_{t+1}, a', w) - \psi_\theta(s_t, a_t, w)||^2 \tag{7}$$

$$\mathcal{L}_\phi = ||r_t^w - \tilde{\phi}_t^\top w||^2 \tag{8}$$

where $a' = \arg\max_a \psi_\theta(s_{t+1}, a, w)^\top w$. Selecting the next action via the combination of all modules ensures they individually convergence to optimal values (Russell & Zimdars, 2003). The final loss is $\mathcal{L} = \alpha_Q\mathcal{L}_Q + \alpha_\psi\mathcal{L}_\psi + \alpha_\phi\mathcal{L}_\phi$. We found $\alpha_\psi = \alpha_\phi = 1.0$ and $\alpha_Q = 0.5$ to generally work well.

5 Experiments

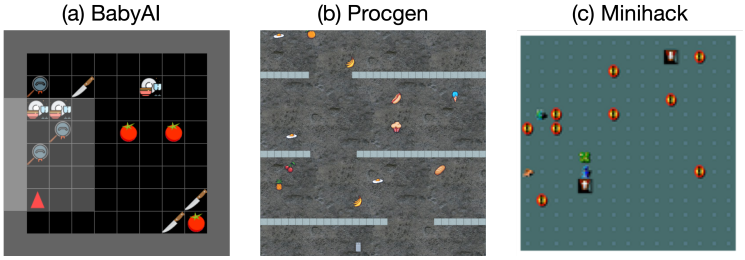

Figure 4: **We study compositional generalization in three environments.** (a) In BabyAI, an agent learns to pick up one object type at a time during training. During testing, the agent must pickup combinations of object types while avoiding other object types. This is the setting used by USFA which assumed **hand-designed** cumulants. (b) In Procgen, we study extending this form of generalization to a visually diverse, procedurally generated environment. (c) In Minihack, we go beyond combining object navigation skills. Here, there are three training settings: (1) avoiding teleportation traps, (2) avoiding monsters, and (3) partial visibility around the agent. We study generalization to the combination of these settings.

We study generalization when training behaviors must be combined concurrently in time in the presence of novel object appearances and layouts. The need to combine training behaviors tests how well MSFA can leverage SF&GPI. Generalization to novel object appearances and layouts tests how well MSFA's modular construction supports generalization to novel environment configurations.

Baselines. (1) **Universal Value Function Approximator (UVFA)** (Schaul et al., 2015), which takes the task as input: $Q_\theta(s, w)$. Comparing against UVFA enables us to measure the transfer benefits of SF&GPI. (2) **UVFA with Feature-Attending Recurrent Modules (UVFA+FARM)** instead takes state-factors as input $Q_\theta(s^{(1)}, \ldots, s^{(n)}, w)$. Each state-factor $s^{(k)}$ is the output of a FARM module. Comparing against UVFA+FARM enables us to probe the importance of having modules learn cumulants and successor features. (3) **Universal Successor Function Approximator (USFA)** (Borsa et al., 2019) leverages a single monolithic function for successor features with **hand-designed cumulants**. USFA is an upper-bound baseline that allows us to test the quality of cumulants and successor features that MSFA learns. We also test a variant of USFA with learned cumulants, **USFA-Learned-ϕ**, which shows how the architecture degrades without oracle cumulants.

Implementation. We implement the state modules of MSFA with FARM modules (Carvalho et al., 2021b). For UVFA and USFA, we learn a state representation with an LSTM (Hochreiter & Schmidhuber, 1997). We implement all ϕ, ψ, and Q functions with Mutli-layer Perceptrons. We train UVFA and UVFA+FARM with n-step Q-learning (Watkins & Dayan, 1992). When learning cumulants, USFA and MSFA have the exact same losses and learning alogirthm. They both learn Q-values and SFs with n-step Q-learning. We use $n = 5$. When not learning cumulants, following (Borsa et al., 2019), USFA only learns SFs with n-step Q-learning. All agents are built with JAX (Bradbury et al., 2018) using the open-source ACME codebase (Hoffman et al., 2020) for reinforcement learning.

5.1 COMPOSITIONAL GENERALIZATION ACROSS OBJECT NAVIGATION TASKS

MSFA learns disentangled functions for computing ϕ and estimating ψ. We hypothesize that this facilitates learning cumulants that respond to different aspects of the environment (e.g. to different object categories). This leads to the following research questions. **R1**: Can we recover prior compositional generalization results that relied on hand-designed cumulants for different object categories? **R2**: How important is it to learn disentangled functions for ϕ and ψ? **R3**: Without GPI, do learning disentangled functions for ϕ and ψ still aid in generalization?

Setup. We implement a simplified version of the object navigation task of (Borsa et al., 2019) in the (Chevalier-Boisvert et al., 2019) grid-world. The environment consists of 3 instances of 4 object categories. **Observations** are partial and egocentric. **Actions**: the agent can rotate left or right, move forward, or pickup an object. When it picks up an object, following (Borsa et al., 2019), the object is respawned somewhere on the grid. **Task** vectors lie in $w \in \mathbb{R}^4$ with training tasks being the standard unit vectors. For example, $[0, 1, 0, 0]$ specifies the agent must obtain objects of category 2. **Generalization** tasks are linear combinations of training tasks. For example, $[-1, 1, -1, 1]$ specifies the agent must obtain categories 2 and 4 while *avoiding* categories 1 and 3. This is challenging because the agent has *never* learned to avoid objects. Borsa et al. (2019) showed that USFA could generalize with *hand-designed* cumulants that described whether an object was picked up.

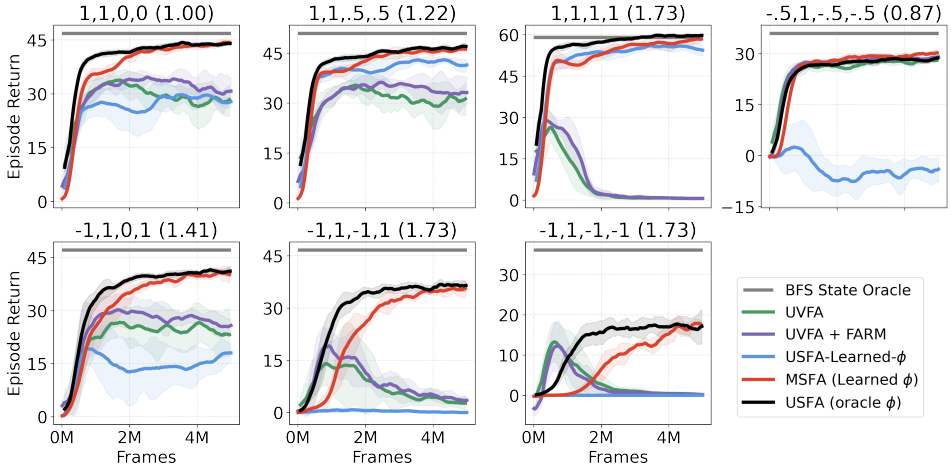

Figure 5: **MSFA matches USFA, which has hand-designed cumulants**. We show mean and standard error generalization episode return across 4 runs. We put a task's L2 distance to the closest train task in parenthesis. USFA best generalizes to novel combinations of picking up and avoiding objects. Once USFA learns cumulants, its performance degrades significantly. UVFA-based methods struggle as more objects should be avoided or tasks are further in distance to train tasks.

R1: MSFA is competitive with USFA, which uses oracle ϕ. Figure 5 shows USFA with a similar generalization trend to (Borsa et al., 2019). Tasks get more challenging as they are further from train tasks or involve avoiding more objects. For simply going to combinations of objects, USFA-Learned-ϕ does slightly worse than MSFA. However, with more objects to avoid, all methods except MSFA (including USFA-Learned-ϕ) degrades significantly. For comparison, we show performance by an oracle bread-first-search policy with access to ground-truth state (**BFS State Oracle**). All methods have room for improvement when objects must be avoided. In the appendix, we present heat-maps for how often object categories were picked up during different tasks. We find that MSFA most matches USFA, while USFA-Learned-ϕ commonly picks up all objects regardless of task.

R2: Learning disentangled ϕ and ψ functions is critical for generalization. Learning an entangled function corresponds to learning a monolothic function for ψ or ϕ where we concatenate module-states, e.g. $\tilde{\psi}_t^w = \psi_\theta(s_t^{(1)}, \ldots, s_t^{(n)}, a, w)$. Disentangled functions correspond to equation 5. Figure 6 (a) shows that without disentangled functions for ϕ **and** ψ, performance severely degrades. This also highlights that a naive combination of USFA+FARM—with entangled functions for ϕ and ψ—does not recover our generalization performance.

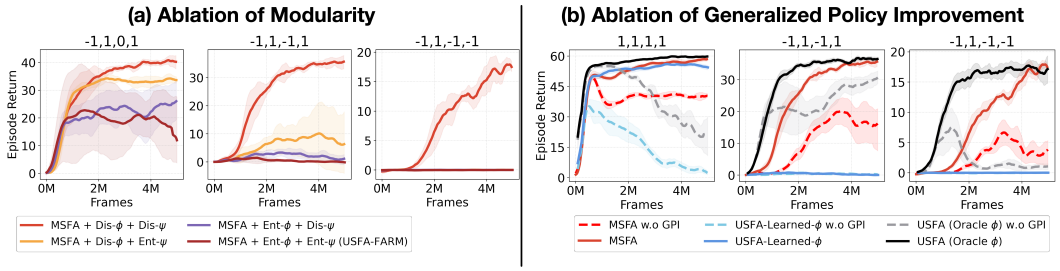

Figure 6: **Learning disentangled ϕ and ψ is key to generalization and improves generalization even without GPI.** We show mean and standard error generalization episode return across 4 runs. (a) We ablate having disentangled functions for ϕ_θ and ψ_θ. Generalization results degrade significantly. (b) We ablate leveraging GPI for generalization from all SF-based methods. MSFA without GPI can outperform both USFA-Learned-ϕ with GPI and USFA without GPI. This shows the utility of modularity for generalization.

R3: Modularity alone improves generalization. For all SF-based methods, we remove GPI and select actions with a greedy policy: $\pi(s) = \arg\max_a \psi_\theta(s_t, a, w)^\top w$. Figure 6 (b) shows that GPI is critical for generalization with USFA as expected. USFA-Learned-ϕ benefits less from GPI (presumably because of challenges in learning ϕ). Interestingly, MSFA can generalize relatively well without GPI, sometimes doing better than USFA without GPI.

5.2 Compositional generalization across object navigation tasks with novel visual appearances and environment configurations

Beyond generalizing to combinations of tasks, agents will need to generalize to different layouts and appearances of objects. Here we ask, **R4**: Can MSFA enable compositional generalization in a visually diverse, procedurally generated environment?

Setup. We leverage the "Fruitbot" environment within ProcGen (Cobbe et al., 2020). Here, an agent controls a paddle that tries to obtain certain categories of objects while avoiding others. When the agent hits a wall or fence, it dies and the episode terminates. If the agent collects a non-task object, nothing happens. **Observations** are partial. **Actions**: At each time-step the agent moves one step forward and can move left or right or shoot pellets to open fences. **Training and generalization tasks** follow the same setup as §5.1.

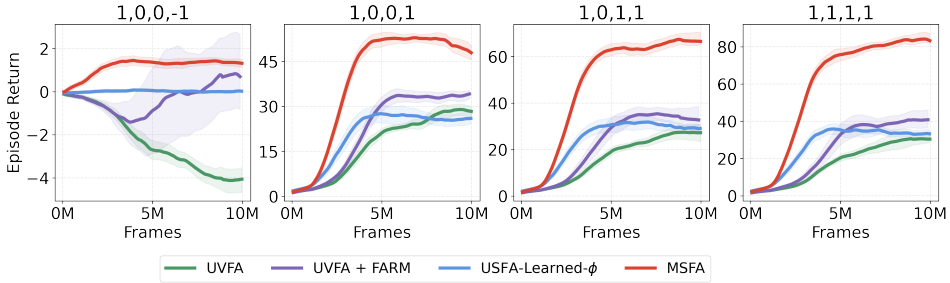

Figure 7: **MSFA is able to extend its compositional generalization abilities to the visually diverse, procedurally generated ProcGen environment.** We find that no method is able to do well when there are objects to avoid ($w = [1, 0, 0, -1]$) in this setting (see text for more). However, as more objects need to be collected MSFA best generalizes (4 runs).

R4: MSFA improves compositional generalization of object navigation tasks in a visually diverse, procedurally generated environment. Figure 7 shows that when an agent has to generalize to collecting more objects, modular architectures generalize best, with MSFA doing best. When objects have to be avoided, we see that no architecture does well, though MSFA tends to do better. We

observe that avoiding objects leads agents to hit walls. Since the agent always moves forward at each time-step in Fruibot, this makes avoiding objects a particularly challenging type of generalization.

5.3 Compositional generalization of heterogeneous tasks

Aside from navigation tasks, researchers and practitioners will be interested in generally combining solutions to different tasks. We ask **R5**: Can MSFA enable combining solutions to heterogeneous tasks?

Setup. We leverage the "Minihack" environment (Samvelyan et al., 2021). Here, an agent is spawned at the top of a map with obstacles. It must navigate to the a ladder at the bottom of the map. The agent experiences partial **observations** of the environment. **Actions**: At each time-step the agent can select from 8 compass directions. **Training tasks**: The agent experiences three training tasks (1) avoiding monsters which kill the agent, (2) avoiding traps which teleport the agent away from the goal, and (3) experiencing a small local view of the environment with everything else black. **Generalization** requires that the agent generalize to (a) new configurations of objects and (b) to the *combination* of all three training conditions.

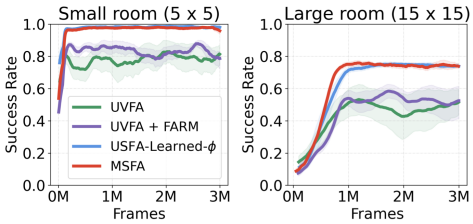

Figure 8: Here, we study whether MSFA can train on three separate settings (1) limited visibility (2) the presence of monsters (3) the presence of teleportation traps (see text for more). All successor feature based methods best enable combining training tasks for generalization in this setting.

R5: MSFA enables combining solutions to heterogeneous tasks. Figure 8 shows that successor feature based methods are best able to successfully combine solutions to different tasks. In a small room, both SF-based methods get 100% success rate. In a larger room, no method generalizes perfectly: SF-based methods do best at 75%, followed by UVFA-FARM and UVFA with 60%.

6 Discussion and Conclusion

We have presented "Modular Successor Feature Approximators", a modular neural network for learning cumulants and SFs produced by their own modules. We first showed that MSFA is competitive with prior object navigation generalization results that relied on hand-designed cumulants (§5.1). Afterwards, we showed that MSFA improves compositional generalization in a visually diverse, procedurally generated environment (§5.2). Finally, we showed that MSFA can combine solutions of heterogeneous tasks (§5.3). Our ablations show that learning disentangled cumulants and SFs is critical for generalization with GPI. Interestingly, without GPI, disentangled cumulants and SFs improve generalization over a monolithic architecture (Figure 6).

We compared MSFA to two baselines: (1) USFA, a monolithic architecture for learning cumulants and SFs; (2) FARM, an architecture which learns multiple state-modules but combines them with a monolithic Q-value function. Our results show that when learning cumulants, MSFA improves generalization with SF&GPI compared to USFA. Additionally, without GPI, MSFA as an architecture improves compositional generalization as compared to both FARM and USFA.

Limitations. While we demonstrated reward-driven discovery of cumulants for transfer with SF&GPI, we focused on relatively simple task encodings. Future work can extend this to more expressive encodings such as language embeddings. Another limitation is that we did not explore more sophisticated state-feature discovery methods such as meta-gradients (Veeriah et al., 2019). Nonetheless, we think that MSFA provides an important insight for future work: that modularity is a simple but powerful inductive bias for discovering state-features that enable zero-shot transfer with SF&GPI.

Future directions. SFs are useful for exploration (Janz et al., 2019; Machado et al., 2020); for discovering and combining options (Barreto et al., 2019; Hansen et al., 2019); for transferring policies across environments (Zhang et al., 2017); for improving importance sampling (Fujimoto et al., 2021); and for learning policies from other agents (Filos et al., 2021). We hope that future work can leverage MSFA for improved state-feature discovery and SF-learning in all of these settings.

REFERENCES

Ankesh Anand, Jacob Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Théophane Weber, and Jessica B Hamrick. Procedural generalization by planning with self-supervised world models. *arXiv preprint arXiv:2111.01587*, 2021.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *NIPS*, 30, 2017.

André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, pp. 501–510. PMLR, 2018.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *PNAS*, 117(48):30079–30087, 2020.

Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado Van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. *ICLR*, 2019.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Ethan Brooks, Janarthanan Rajendran, Richard L Lewis, and Satinder Singh. Reinforcement learning of implicit and explicit control flow instructions. In *International Conference on Machine Learning*, pp. 1082–1091. PMLR, 2021.

Wilka Carvalho, Andrew Lampinen, Kyriacos Nikiforou, Felix Hill, and Murray Shanahan. Feature-attending recurrent modules for generalization in reinforcement learning. *arXiv*, 2021a.

Wilka Carvalho, Anthony Liang, Kimin Lee, Sungryull Sohn, Honglak Lee, Richard L Lewis, and Satinder Singh. Reinforcement learning for sparse-reward object-interaction tasks in a first-person simulated 3d environment. *IJCAI*, 2021b.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJeXCo0cYX`.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. Modular networks for compositional instruction following. *arXiv preprint arXiv:2010.12764*, 2020.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 2169–2176. IEEE, 2017.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Angelos Filos, Clare Lyle, Yarin Gal, Sergey Levine, Natasha Jaques, and Gregory Farquhar. Psiphi-learning: Reinforcement learning with demonstrations using successor features and inverse temporal difference learning. In *International Conference on Machine Learning*, pp. 3305–3317. PMLR, 2021.

Scott Fujimoto, David Meger, and Doina Precup. A deep reinforcement learning approach to marginalized importance sampling with the successor representation. In *International Conference on Machine Learning*, pp. 3518–3529. PMLR, 2021.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6244–6251. IEEE, 2018.

Steven Hansen, Will Dabney, Andre Barreto, Tom Van de Wiele, David Warde-Farley, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pp. 4455–4464. PMLR, 2020.

Jonathan Hunt, Andre Barreto, Timothy Lillicrap, and Nicolas Heess. Composing entropic policies using divergence correction. In *International Conference on Machine Learning*, pp. 2911–2920. PMLR, 2019.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in neural information processing systems*, 32, 2019.

David Janz, Jiri Hron, Przemysław Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: exploration and uncertainty in temporal difference learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33: 19884–19895, 2020.

Lajanugen Logeswaran, Wilka Torrico Carvalho, and Honglak Lee. Learning compositional tasks from language instructions. In *Deep RL Workshop NeurIPS 2021*, 2021.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 04, pp. 5125–5133, 2020.

Kanika Madan, Nan Rosemary Ke, Anirudh Goyal, Bernhard Schölkopf, and Yoshua Bengio. Fast and slow learning of recurrent independent mechanisms. *arXiv preprint arXiv:2105.08710*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pp. 2661–2670. PMLR, 2017.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Stuart J Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 656–663, 2003.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *Advances in neural information processing systems*, 31, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3):323–339, 1992.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. *arXiv preprint arXiv:2001.00248*, 2020.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, Lyubing Qiang, Izzeddin Gur, Aleksandra Faust, and Honglak Lee. Fast inference and transfer of compositional task structures for few-shot task generalization. In *Uncertainty in Artificial Intelligence*, pp. 1857–1865. PMLR, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv*, 2017.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2371–2378. IEEE, 2017.

Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE international conference on computer vision*, pp. 483–492, 2017.

A DERIVATION OF MODULAR SUCCESSOR FEATURES

$$Q^\pi(s,a,w) = \mathbb{E}_\pi\left[\sum_{j=0}^{\infty}\gamma^j R_t | S_t = s, A_t = a\right] \tag{9}$$

$$= \mathbb{E}_\pi\left[\sum_{j=0}^{\infty}\gamma^j \left(\sum_{k=1}^{n}{\phi_{t+j}^{(k)}}^\top w^{(k)}\right) | S_t = s, A_t = a\right] \tag{10}$$

$$= \sum_{k=1}^{n}\left(\mathbb{E}_\pi\left[\sum_{j=0}^{\infty}\gamma^j {\phi_{t+j}^{(k)}}^\top w^{(k)} | S_t = s, A_t = a\right]\right) \tag{11}$$

$$= \sum_{k=1}^{n}\left(\mathbb{E}_\pi\left[\sum_{j=0}^{\infty}\gamma^j \phi_{t+j}^{(k)} | S_t = s, A_t = a\right]^\top w^{(k)}\right) \tag{12}$$

$$= \sum_{k=1}^{n}\psi^{\pi,k}(s,a)^\top w^{(k)} \tag{13}$$

$$= \sum_{k=1}^{n}Q^{\pi,k}(s,a,w^{(k)}) \tag{14}$$

B INTER-MODULE ATTENTION AND GATING

At each time-step t, each module updates with both observation features $z_t = f_z^\theta(x_t)$ and with information from the previous module-states $\{s_{t-1}^{(k)}\}$. MSFA shares information between modules with a "query-key" system. Each module has a "query" representation of its module-state that it uses to select from "key" representations of the other module-states. MSFA scores how well a query matches a key by computing the dot-product between the two representations. A module then updates with the module-state corresponding to the key with the highest dot-product. To enable flexible updating, each module uses a gating mechanism to decide the degree to which information should be incorporated into an update.

More technically, the query vector is computed using the previous-module state and action: $q_t^{(k)} = W^{\texttt{query}}[s_{t-1}^{(k)}, a_{t-1}] \in \mathbb{R}^{d_q}$. Keys and values are computed as $K_t = W^{\texttt{key}}[s_{t-1}^1; \ldots; s_{t-1}^n; 0] \in \mathbb{R}^{n+1\times d_q}$ and $V_t = W^{\texttt{value}}[s_{t-1}^1; \ldots; s_{t-1}^n; 0] \in \mathbb{R}^{n+1\times d_q}$, respectively. Note that we add a zero-vector key and value to allow a query to select "no information". Each module selects "values" $v_t^{(k)}$ to update using dot-product attention (Vaswani et al., 2017):

$$v_t^{(k)} = \text{softmax}\left(\frac{q_t^{(k)}K_t^\top}{d_q}\right)V_t \tag{15}$$

Gating mechanism has been shown important for leveraging transformer-style attention when updating state in RL agents (Parisotto et al., 2020). Thus, each module uses a sigtanh gate when updating:

$$u_t^{(k)} = q_t^{(k)} + \tanh(W^{g_1}v_t^{(k)}) \odot \sigma(W^{g_2}v_t^{(k)} - b_g). \tag{16}$$

MSFA then updates its module-stateas follows:

$$s_t^{(k)} = s_{\theta_k}(u_t^{(k)}, z_t) \tag{17}$$

C ADDITIONAL ANALYSIS

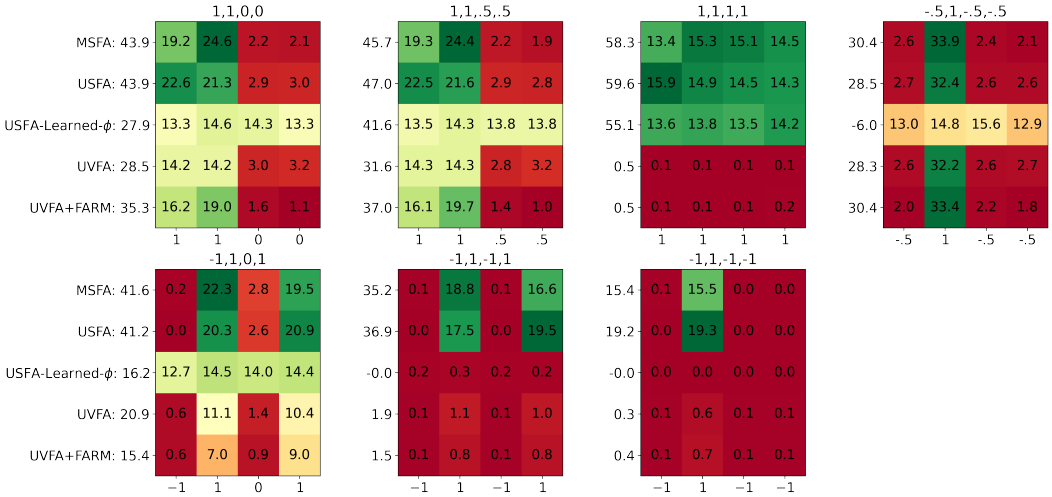

Figure 9: **MSFA best matches the behavior of USFA across generalization tasks**. We present additional analysis for the experiments in §5.1. Each heatmap displays how often object categories were picked up by each method for a given generalization task. Rows correspond to different methods. We show the final episode return for each method along the y-axis. Columns describe the transition-reward when an object category is picked up. USFA has access to hand-designed cumulants that described whether an object category was picked up. Despite learning cumulants, MSFA best matches USFA's object collection dynamics. USFA-Learned-ϕ equally collects all objects regardless of task, except for when $r \leq -1$ when an object is picked up. In this setting, no method except USFA or MSFA collect objects. As a reminder, negative rewards are never experienced during training and these tasks are the furthest away from training tasks in task-space. UVFA-based methods tend to collect rewarding objects but not as effectively as MSFA or USFA.

D FULL RESULTS

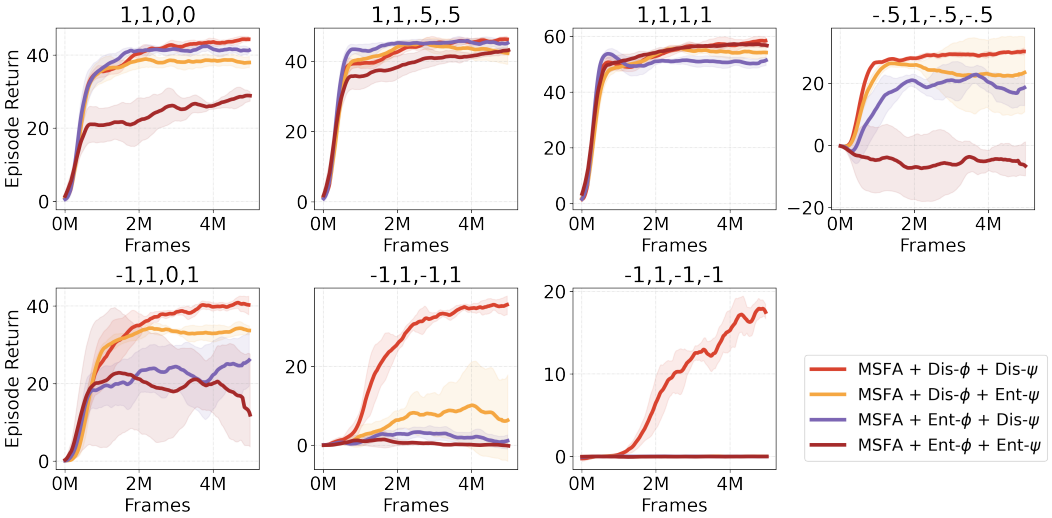

Figure 10: All results for ablating disentanglement of ϕ and ψ. We consistently find that disentangled functions for ϕ and ψ have the best generalization performance.

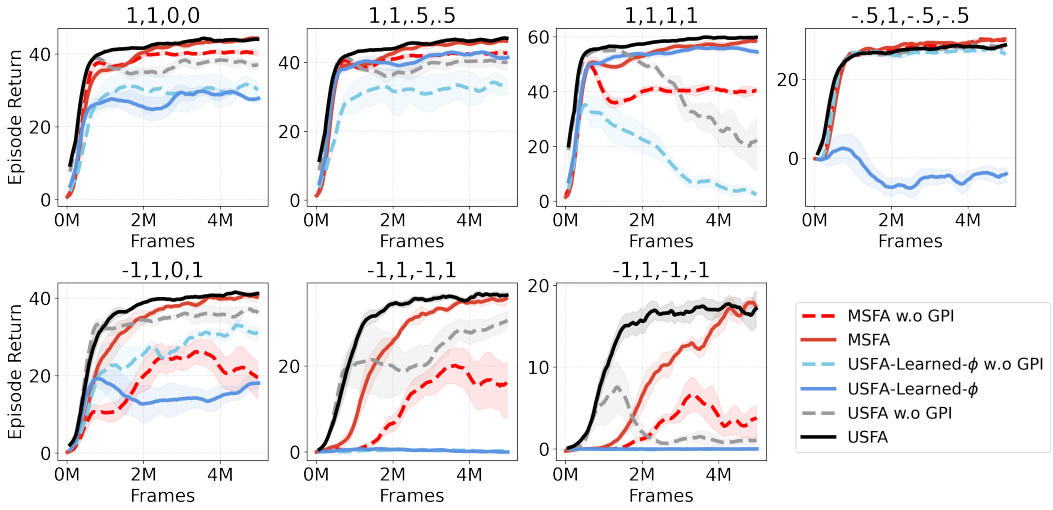

Figure 11: All results for ablating GPI. First, we compare MSFA to USFA (which has hand-designed cumulants). We can see that the two perform comparably across the board. Sometimes (e.g. $[1, 1, 1, 1]$ and $[-1, 1, -1, -1]$), MSFA outperforms USFA. Next we compare MSFA to USFA-Learned-ϕ. We see that MSFA performs as well as USFA-Learned-ϕ or better in all settings except $[-1, 1, 0, 1]$. Interestingly, MSFA without GPI outperforms USFA-Learned-ϕ with GPI in some generalization settings (e.g. $[1, 1, 0, 0]$ and $[-1, 1, -1, 1]$).

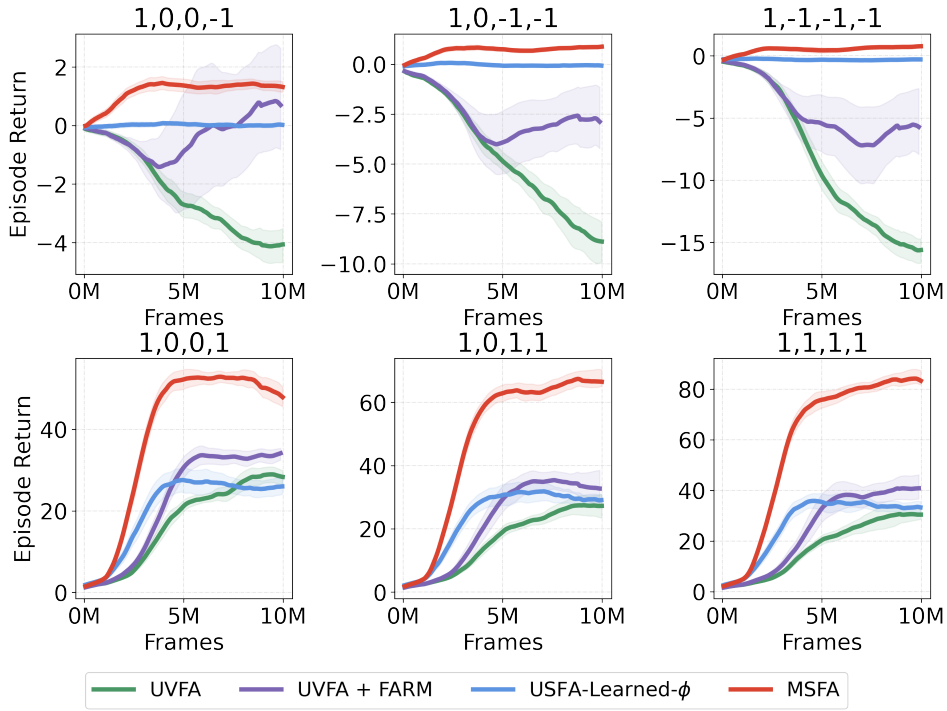

Figure 12: All results for the Fruitbot environment. When objects must be avoided, we see that no method does well, though MSFA generalizes best and can pick up a couple of objects. When combinations of objects must be collected, MSFA outperforms baselines by a large margin.

E HYPERPARAMETERS

We shown hyperparameters in Tables 1 and 2. Tables 1 describes hyperparameters that were shared across algorithms and Tables 2 describes hyperparameters that were different across algorithms. We developed the algorithms using the Hoffman et al. (2020) reinforcement learning codebase and unless stated otherwise using default "R2D2" (Kapturowski et al., 2018) config values in the codebase. We describe our development and hyperparameter search in more detail below.

E.1 DEVELOPMENT AND HYPERPARAMETER SEARCH

The first thing we did was replicate the generalization gap between USFA and UVFA that Borsa et al. (2019) produced in their paper using the BabyAI gridworld (Chevalier-Boisvert et al., 2018). As a result, most hyperparameters were tuned for either USFA or UVFA. We used the Atari Convolutional network used by "Deep Q-Networks" (DQN) in Mnih et al. (2015) as our visual encoder and an LSTM (Hochreiter & Schmidhuber, 1997) as our state representation function.

Once we were able to replicate their generalization performance, we implemented USFA-Learned-ϕ and MSFA. In this setting, we are learning Q-values, successor features, and predicting rewards from cumulants. This leads to the following coefficients for losses: α_Q, α_ψ, and α_ϕ respectivey. We fixed $\alpha_\psi = 1$ and searched $\alpha_\phi \in [.01, .1, 1, 10, 100]$ and $\alpha_Q \in [.01, .05, .1, .5, 1.0]$ for both USFA-Learned-ϕ and MSFA. For MSFA and UVFA-FARM, we chose the number of modules to be the number of training tasks. We set each module size so that the number of parameters by all algorithms was approximately the same (with USFA as a reference). We found that FARM hyperparameter (Carvalho et al., 2021a) worked well for both UVFA-FARM and for MSFA (FARM modules are the basis of MSFA state-modules). We highlight that properly masking all losses was absolutely critical to good generalization performance. Without proper masking, out-of-episode data is used for value estimation of in-episode data, which can lead to inaccurate value estimates. We suspect that this hampers GPI.

When doing experiments for Fruitbot, we took all implementation details from (Cobbe et al., 2020). We first replicated the DQN performance from Cobbe et al. (2020) with UVFA. Some important changes that we made included (1) adding prioritized experience replay (2) leveraging the Impala ResNet vision torso (Espeholt et al., 2018) (3) lowering the memory size to match the paper (4) using the `SIGNED_HYPERBOLIC_PAIR` value transformation for UVFA-based algorithms (5) increasing the capacity of the Q-value estimation MLPs. We applied these settings to other baselines and redid our search over α_ϕ and α_Q for both MSFA and USFA-Learned-ϕ. For each method, we changed the size of the networks so they were approximately the same (with UVFA as a reference). Since MSFA uses disentangled functions for ϕ and ψ, it has fewer parameters.

When doing experiments in Minihack, we found that the Fruitbot changes were important for good performance in this domain. All hyperparameters led to strong training performance but poor generalization for UVFA. We found that increasing the batch size was important for improving generalization of USFA-Learned-ϕ and MSFA but was detrimental to UVFA-based methods.

Table 1: Hyperparameters shared across all algorithms.

Algorithm			
Loss Hyperparameters	BabyAI	Fruibot	Minihack
discount	0.99	0.99	0.99
burn_in_length	0	0	0
trace_length	40	40	40
importance_sampling_exponent	0	0.6	0.6
max replay size	100,000	70,000	70,000
max gradient norm	80	80	80
Shared Network Components			
Vision torso	DQN ConvNet	Impala ResNet	Impala ResNet

Table 2: Hyperparameters for individual algorithms. Note: T_1 = `IDENTITY_PAIR` and T_2 = `SIGNED_HYPERBOLIC_PAIR`.

	BabyAI	Fruibot	Minihack
MSFA			
Parameters (millions)	2.1M	1.66M	1.7M
α_ϕ	1	1	1
α_ψ	1	1	1
α_Q	0.5	0.5	0.5
module_size	150	60	80
nmodules	4	4	3
attention heads	2	2	1
batch size	32	32	64
ϕ MLP hidden sizes	[256]	[256]	[256]
ψ MLP hidden sizes	[128]	[512, 512]	[512, 512]
projection dim	16	16	16
USFA			
Parameters (millions)	2.35M	1.98M	1.95M
lstm size	512	300	300
α_ϕ	1	1	1
α_ψ	1	1	1
α_Q	0.5	0.5	0.5
batch size	32	32	128
ϕ MLP hidden sizes	[256]	[256]	[256]
ψ MLP hidden sizes	[128]	[512, 512]	[512, 512]
UVFA			
Parameters (millions)	2.09M	1.96M	1.95M
lstm size	512	256	256
rlax.TxPair	T_1	T_2	T_2
batch size	32	32	32
Q MLP hidden sizes	[128]	[512, 512]	[512, 512]
UVFA-FARM			
Parameters (millions)	2.17M	2.15M	2.06M
module_size	150	64	80
nmodules	4	4	3
attention heads	2	2	1
rlax.TxPair	T_1	T_2	T_2
batch size	32	32	32
projection dim	16	16	16
Q MLP hidden sizes	[128]	[512, 512]	[512, 512]

F ENVIRONMENTS

We presented most environment information in the main text. Here, we specify which levels we used from the corresponding environments or how the changed environments for our experiments.

F.1 PROCGEN

We used the Fruitbot environment in Procgen. We made the following changes. We set the maximum number of levels that an agent could complete within a lifetime to 4. We divided the 12 object types $\{O_1 \dots O_{12}\}$ into 4 categories: $C_1 = \{O_1, O_2, O_3\}, \dots, C_4 = \{O_{10}, O_{11}, O_{12}\}$, where each category provided postive reward for picking up any of its 3 category types.

F.2 MINIHACK

We did not make any changes to the environment. Train tasks were: "MiniHack-Room-Monster-Y-v0", "MiniHack-Room-Trap-Y-v0", and "MiniHack-Room-Dark-Y-v0". Test tasks were "MiniHack-Room-Ultimate-Y-v0". We used $Y = 5 \times 5$ for the "small room" experiments and $Y = 15 \times 15$ for the "large room" experiments.