KNOWLEDGE REPRESENTATION FOR REINFORCEMENT LEARNING USING GENERAL VALUE FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) is a very powerful approach for learning good control strategies from data. Value functions are a key concept for reinforcement learning, as they guide the search for good policies. A lot of effort has been devoted to designing and improving algorithms for learning value functions. In this paper, we argue that value functions are also a very natural way of providing a framework for knowledge representation for reinforcement learning agents. We show that generalized value functions provide a unifying lens for many algorithms, including policy gradient, successor features, option models and policies, and other forms of hierarchical reinforcement learning. We also demonstrate the potential of this representation to provide new, useful algorithms.

1 INTRODUCTION

Value functions are at the heart of reinforcement learning algorithms (RL) as a very useful tool in guiding the progress towards good policies. Usually, a value function estimate, representing the expected discounted return of a policy, is maintained in order to guide policy improvement. However, the goal of having realistic AI agents requires us to consider how they may go beyond the goal of solving one task, to a setting in which such agents can acquire and maintain diverse knowledge about the world. This brings about the question of how such knowledge should be expressed and maintained in an RL agent.

In this paper, we highlight the fact that value functions can in fact be used as the main building block for knowledge representation in RL agents. We build on the framework of General Value Functions (GVFs) (White, 2015; Sutton & Tanner, 2005; Sutton et al., 2011; Schaul & Ring, 2013), which have been proposed in prior work as a way to capture knowledge about the world in a flexible fashion. GVFs represent long-term predictions about different aspects of an agent's observations. For example, an RL agent interested in hockey can ask questions like "what is the expected time until I hit the puck, given my current strategy?" or "what are the chances of my team winning this game?" and represent the answers to these questions as GVFs. This form of knowledge representation provides two clear benefits. First, since it is grounded in actual observations, it can be learned incrementally from a single stream of experience, using off-policy learning methods. Second, as illustrated by the example questions above, it can be used to model the world at different time scales (Modayil et al., 2014). The Horde architecture (Sutton et al., 2011) illustrates in fact the successful large-scale learning of GVFs about a diversity of signals and at many different time scales, in parallel, from a single stream of data. This aspect of GVFs is very appealing, as it allows using the experience (which may be difficult to acquire) in order to learn many different things. GVFs also have the advantage of being able to profit from a long history of RL algorithm development for value function learning (Sutton & Barto, 1998).

Our main contribution in this paper is to unify and interpret several different RL algorithms, including policy gradients (Sutton et al., 1999a) and different ways of achieving temporal abstraction, such as successor features (Barreto et al., 2017), option models and value functions (Sutton et al., 1999b), feudal networks (Vezhnevets et al., 2017) and universal value functions (Schaul et al., 2015) through the lens of GVFs. We also explore how GVFs can be used to facilitate the development of new algorithmic ideas in RL, and we illustrate the benefits of this approach through two algorithms: a new approach to policy gradient, which produces much more stable and data-efficient convergence, and a combination of feudal networks and successor features.

2 BACKGROUND

Consider a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition function $P(s'|s,a)$. A task is specified using *an extrinsic reward* function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$, and a discount value $\gamma \in [0,1]$, which define the expected discounted return: $E_\pi[\sum_{t=0}^{\infty} \gamma^t R_{t+1}] = E_\pi[\sum_{t=0}^{\infty}(\prod_{i=1}^{t}\gamma)R_{t+1}]$, also known as the *value function* of a given policy π.

General Value Functions (GVFs) Sutton et al. (2011) provide a unified way of expressing predictions about signals other than extrinsic rewards, under policies that are different from the behavior policy, and under flexible, state-dependent discount schemes. Given a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, a (possibly multi-dimensional) *cumulant* function $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^K$ and a continuation function $\gamma : \mathcal{S} \to [0,1]$, the associated *generalized value function (GVF)* $v_{C,\gamma,\pi} : \mathcal{S} \to \mathbb{R}^K$ predicts the expected cumulant-based return, discounted by the continuation function:

$$v_{C,\gamma,\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty}\left(\prod_{i=1}^{t}\gamma(S_i)\right)C_t \;\middle|\; S_0 = s, A_{0:\infty} \sim \pi\right].$$

Note that this definition extends easily to predictions associated with different initial conditions (e.g. for pairs (s,a), rather than just states).

Usual value functions are trivially GVFs $v_{R,\gamma,\pi}$, where the cumulant is simply the MDP reward function, and the discount factor is constant across the state space. Similarly, when the cumulant C depends on state-action pairs, the corresponding GVF is equivalent to the state action Q-values.

In the following sections, we will explain how we can use GVFs to express different RL algorithms, gaining insight both into their behavior, as well as into paths for their possible improvement.

3 GVFS FOR SUCCESSOR FEATURES

Successor representation. Under any distribution over the state space, as induced by a fixed policy, one can define a prediction for the *discounted* number of times a state s is visited. That is, we predict the *one-hot* cumulant $\mathbb{I}(s)$ such that $C_t = 1$ if $S_t = s$ and 0 otherwise[1]. For a fixed starting state s_0, the associated prediction defines a measure over the state space $\mathcal{S}$: $\mu(s|s_0,\gamma,\pi)$. Dayan (1993) introduced the idea of using this representation of states through their possible futures, called *successor representation*, as a way to provide ideal features This becomes useful in describing predictions about other cumulants C: $v_{C,\gamma,\pi}(s_0) = \int C d\mu(\cdot; s_0,\gamma,\pi)$ (see appendix for proof).

Successor features Successor features (SF) (Barreto et al., 2017) generalize the idea of the successor representation to the case of function approximation in a natural way. Given a function $\phi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^K$ defining feature representations for $s \to a \to s'$ transitions, the SF vector is defined as

$$\psi(s,a;\pi) = E\left[\sum_{t=0}^{\infty}\gamma^t\phi(S_t, A_{t+1}, S_{t+1})|S_0 = s, A_1 = a, \pi\right].$$

If the reward function is linear in the features (i.e. $R(s,a,s') = \phi(s,a,s')^T\mathbf{w}$ for some weight vector $\mathbf{w}$), SFs become very useful in the context of transfer learning, where different tasks correspond to different $\mathbf{w}$. Once SFs for a given policy are computed, they can be used for zero-shot policy evaluation, and in *generalized policy improvement*.

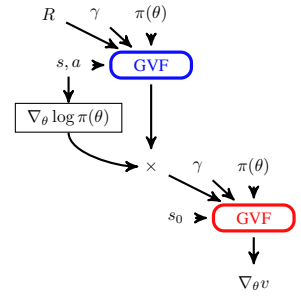
SF (Barreto et al., 2017) are naturally formulated as a GVF $v_{\phi,\gamma,\pi}$, where ϕ is a multi-dimensional cumulant and γ is constant A key result on SFs Barreto et al. (2017) can then be expressed and generalised immediately in the language of GVFs as follows:

Proposition 1. *For a fixed pair (π,γ), given a cumulant $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^k$, and $\mathbf{w} \in \mathbb{R}^k$, $v_{\mathbf{w}^T C,\gamma,\pi}(s) = \mathbf{w}^T v_{C,\gamma,\pi}(s)$.*

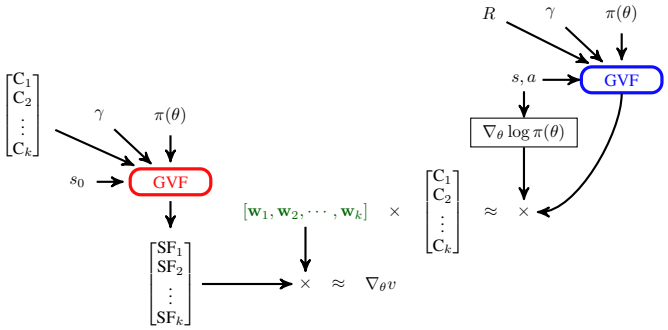
This property is exploited in Barreto et al. (2017) for transfer learning. Given a new task corresponding to scalar cumulant $C' : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the value of any given policy can be evaluated on this new task by approximating C' with the L_2 projection on the space spanned by a multi-dimensional cumulant C. That is, given $\mathbf{w}_{C'} := \arg\min_{\mathbf{w}}(\mathbf{w}^T C - C')^2$, one can estimate $v_{C',\gamma,\pi}(s) \approx \mathbf{w}_{C'}^T v_{C,\gamma,\pi}(s)$.

We will use the GVF version of SF in our experiments.

[1]Similar one-hot cumulants can be defined for (s,a) pairs or $s \to a \to s'$ transitions and derive the corresponding measure over spaces $\mathcal{S} \times \mathcal{A}$ and $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$, respectively.



(a) The policy gradient algorithm describes the interdependence between the GVFs corresponding to the evaluation of a *policy* of a specific task (i.e. R) and the gradient of that evaluation with respect to the *policy* parameters (i.e. $\nabla_\theta v$)



(b) The GVFs involved in the computation of policy gradients can be naturally decoupled by using successor features to approximate the second GVF. The *policy* is evaluated on a set of cumulants (i.e. features) to compute **successor features**, which are then used to approximate the gradient $\nabla_\theta v$ (see Proposition 1).

Figure 1: GVFs for Policy Gradient Methods

4 GVFs FOR POLICY GRADIENT METHODS

Performance gradient estimation techniques of *score function* and *likelihood ratio*, as introduced by Aleksandrov et al. (1968) and Rubinstein (1969), are based on sampling trajectories to estimate both the value of a policy and the gradient used for improving the policy. Given a score function f that depends on a random variable sampled from a parameterised distribution $q(\theta)$, the gradient with respect to the parameters is given by $\nabla_\theta \mathbb{E}_{X\sim q(\theta)}[f(X)] = \mathbb{E}_{X\sim q(\theta)}[f(X)\nabla_\theta \log q(X;\theta)]$. This principle is used in several popular RL algorithms, including REINFORCE (Williams, 1992), actor-critic (Sutton, 1984), policy gradient, eg. (Sutton et al., 1999a; Konda, 2000; Bhatnagar et al., 2009; Lillicrap et al.; Schulman et al., 2015) and other algorithms that use an estimate of the value function to compute the gradient of the policy parameters. The *policy gradient theorem* (Sutton et al., 1999a) provides a formal description of the procedure used in this broad class of algorithms.

We are now going to re-express this result in the language of GVFs, by re-casting the gradient computation in terms of an explicit cumulant.

Theorem 1. *Let π_θ be a policy parameterised by a vector θ, γ be a constant continuation function, and $C : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ be a one-dimensional cumulant. The gradient of the general value function $v_{C,\gamma,\pi_\theta}(s)$ with respect to θ is itself a general value function that depends on the cumulant:*

$$\hat{C}(s,a) := v_{C,\gamma,\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s).$$

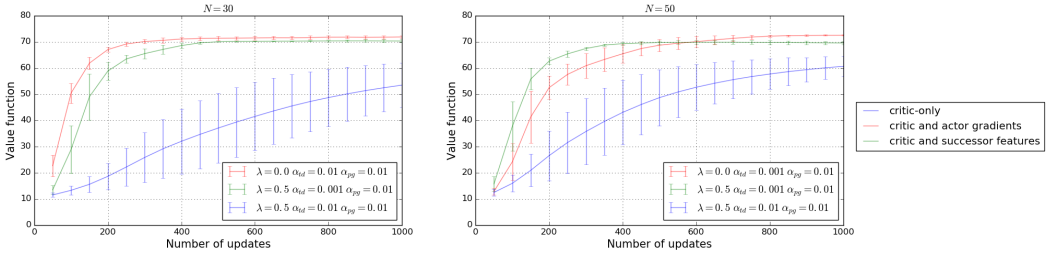
That is, $\nabla_\theta v_{C,\gamma,\pi_\theta}(s) = v_{\hat{C},\gamma,\pi_\theta}(s)$.

Figure 1a provides a visual depiction of this result and the appendix contains the proof.

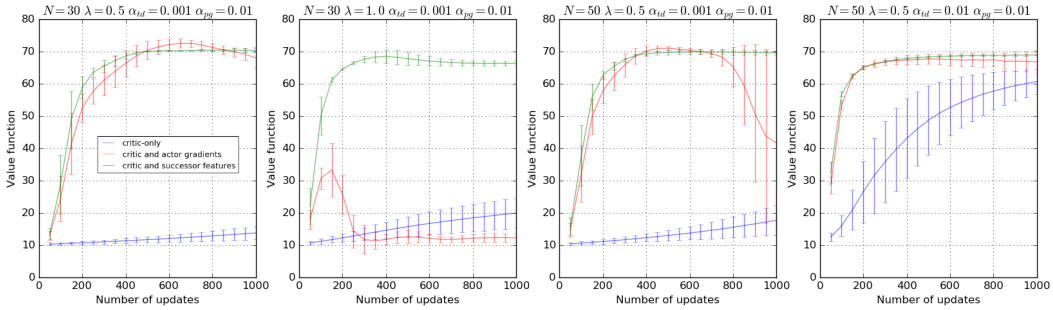
The policy gradient theorem shows that these widely used methods are based on the estimation of two inter-dependent GVFs: the cumulant for one GVF is obtained as the output of another GVF. Note that this separation means we can easily consider other ways of estimating these cumulants.

4.1 BOOTSTRAPPED POLICY GRADIENT

To illustrate the intuition that we gain from Theorem 1, we first run a procedure that is designed to follow the analytic description of the policy gradient: given policy parameters θ, (1) generate a number of rollouts from random initial states (uniformly sampled over the entire state space); (2) use TD(λ) on the sampled data to compute an estimate for the value function v_{R,γ,π_θ}; (3) compute cumulants $\hat{C}$ in Theorem 1 based on the estimates in (1); and (4) generate a separate set of rollouts to estimate the GVF v_{R,γ,π_θ}, using the same TD(λ) procedure. We use TD(λ) to estimate GVFs since it has the flexibility of reusing previous estimates (i.e., bootstrapping).



(a) This plot compares the best combination of hyper-parameters for each algorithm – each plot corresponds to a fixed number N of rollouts for estimating GVFs.



(b) Plots corresponding to individual hyper-parameter configurations illustrate the increased stability of the bootstrapped policy gradient approach when gradients are approximated with successor features.

Figure 2: **Policy gradient updates with TD(λ) estimates.** *L_1 norm of the exact value function vs. number of updates to policy parameters*. We ran all versions of the algorithm over different eligibility trace parameters λ, constant learning rates for the value function α_{td}, and constant learning rates α_{pg} for policy parameters. For performance on various hyper-parameter settings, see Figure 6 (in Appendix). Mean and variance for the value of the policy as a function of policy updates are shown in *blue* when only the critic value is bootstrapped, in *red* when all GVF estimates are bootstrapped, and *green* when successor features are also used.

4.2 Using successor features for policy gradient

Theorem 1 provides a natural linear approximation to the gradient $\nabla_\theta v$. Given a finite number of auxiliary predictions (i.e. cumulants $\{C_i\}_{i=1}^n$) and a linear approximation to an input cumulant $\hat{C} \approx \sum w_i C_i$, we can compute $\nabla_\theta v_{\hat{C},\gamma,\pi} \approx \sum w_i v_{C_i,\gamma,\pi}$, using the successor features idea (see Figure 1b for a corresponding illustration). The intuition is that successor features would help in tracking the continual change in cumulants that is a natural by-product of policy gradient. This should help when the gradient estimates are unstable; sampled data would be used to estimate the long-term accumulation of a fixed set of features instead of a set of parameter gradients.

4.3 Empirical illustration

In Figure 2a, we illustrate the importance of bootstrapping not only the *critic value* v_{C,γ,π_θ} but also in any General Value Functions involved in the computation of the gradient with respect to the policy parameters. All algorithms were run on a simplified version of the four-room environment described in (Sutton et al., 1999a), with discount factor set to 0.9, and the feature map used for linear value function approximation was computed using tile-coding (Sutton & Barto, 1998) (exact details of the experiments are in the appendix). Using small domains allows us to compute the exact value functions, and hence to compare the algorithms against ground truth. As can be seen, the proposed bootstrapped policy gradient approach provides significant improvement in data efficiency. While the advantage of bootstrapping compared to Monte Carlo estimation has been observed before, the effect is magnified through the use of multiple GVFs.

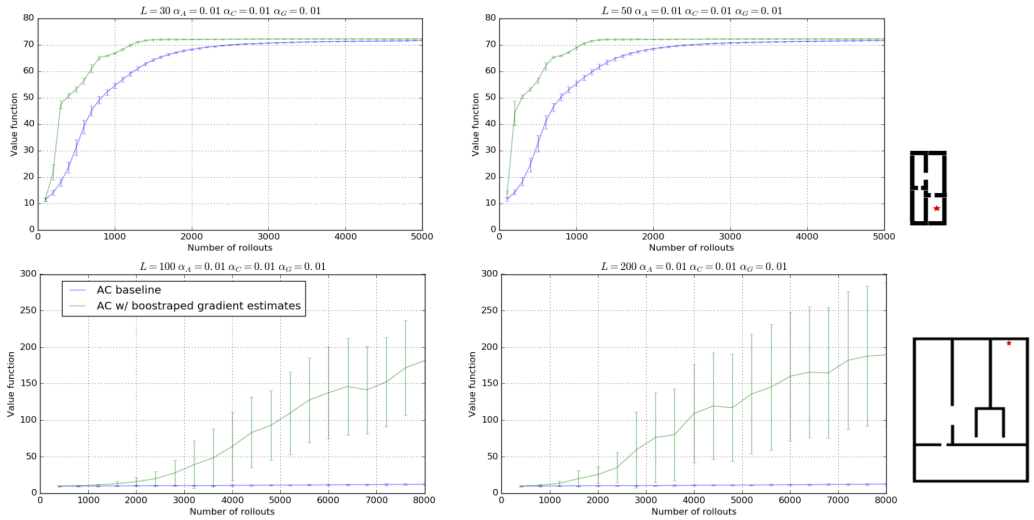


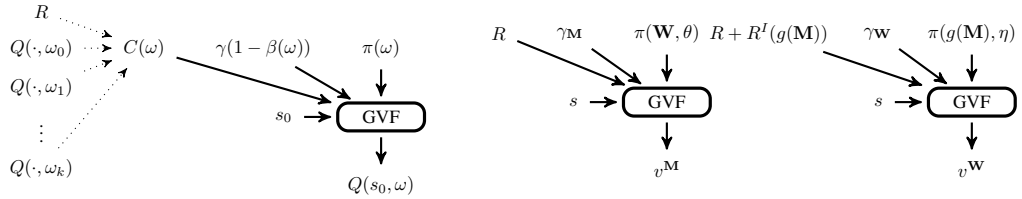
Figure 3: **Actor-Critic with bootstrapped gradient estimates.** L_1 *norm of the exact value function vs. number of rollouts used for policy updates*. *Top:* four-room maze introduced in (Sutton et al., 1999a))($\approx$ 60 states and $\approx$ 15 total features). *Bottom:* large maze grid-world with sparse reward ($\approx$ 3000 states and $\approx$ 90 total features). Each plot corresponds to a fixed number of steps L taken by the agent for every rollout, The plots displayed here correspond to the highest performing hyperparameter setting: α_A is the learning rate for updates to the policy parameters, α_C is the learning rate for updates to the critic component, and α_G is the learning rate for updates to the gradient estimates. For performance on specific hyperparameter setting, see Figure 7 and 8 (in Appendix). Mean and variance for the value of the policy as a function of policy updates are shown in *blue* for baseline actor-critic, in *green* when actor-critic maintains bootstrapped estimates for parameter updates.

We tested the approach with the same tile-coding features as used for value function approximation. The results shown in Figure 2b demonstrate that estimating policy gradients using bootstrapped successor features exhibits much stronger robustness to changes in hyper-parameters.

Algorithm 1: Actor-Critic with GVFs for gradient estimation.

Input: policy $\pi(a|s, \boldsymbol{\theta})$; critic state-value fn. $\hat{v}(s, \mathbf{w})$; general-value fn. $\hat{g}(s, \boldsymbol{\eta})$.
Algorithm parameters: step size $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$, and $\alpha^{\boldsymbol{\eta}} > 0$.
repeat
  Initialize S (first state of episode); $I \leftarrow 1$
  repeat
    $A \sim \pi(a|s, \boldsymbol{\theta})$ and take action A, observe S', R
    $\delta_{\text{crit}} \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta_{\text{crit}} \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$
    $\delta_{\text{grad}} \leftarrow \delta_{\text{crit}} \nabla_{\theta} \log \pi(A|S, \boldsymbol{\theta}) + \gamma \hat{g}(S', \boldsymbol{\eta}) - \hat{g}(S, \boldsymbol{\eta})$
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \alpha^{\boldsymbol{\eta}} I \delta_{\text{grad}} \nabla_{\boldsymbol{\eta}} \hat{g}(S, \boldsymbol{\eta})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \hat{g}(S, \boldsymbol{\eta})$
    $I \leftarrow \gamma I$ and $S \leftarrow \gamma S'$
  until *end of episode*
until *convergence*

Next, we tested the approach in the context of Actor-Critic (Konda, 2000), an approach commonly used to scale the policy gradient algorithm to large or continuous environments. We modified the actor-critic algorithm in (Konda, 2000) to *both* the value function corresponding to the MDP reward function (i.e. the critic) and the gradients w.r.t. to the policy parameters. That is, just as the original actor-critic algorithm uses parameters $\mathbf{w}$ to approximate bootstrap values for the critic function (i.e. $v_{R,\gamma,\pi}(s) \approx \hat{v}_{\mathbf{w}}(s)$, we use a similar approximation for the GVF corresponding to policy gradient



(a) GVFs for Option-value function

(b) GVFs for Feudal Networks.

Figure 4: *Left*: Every option-value corresponds to a separate GVF, concerned both with external reward R, as well as all other GVFs. *Right*: In FuN, manager ($\mathbf{M}$) and worker ($\mathbf{W}$) are trained using separate GVFs: $v^{\mathbf{M}}$, concerned with external return as corresponding to $\mathbf{W}$'s policy; $v^{\mathbf{W}}$, conditioned on goals specified by $\mathbf{M}$.

estimates (i.e. $v_{\hat{C},\gamma,\pi}(s) \approx \hat{g}(s, \boldsymbol{\eta})$). By doing so, we replace updates to the policy parameters that consider only $\nabla_\theta \log \pi \cdot Q$ with long term estimates of the GVFs derived from the analytic form of the policy gradient. These estimates contain gradient information from past gradient computations (i.e. they are bootstrapped). See Algorithm 2 in the appendix for details on the proposed procedure and other implementation details. See Figure 3 for a comparative analysis of the baseline actor-critic algorithm and the aforementioned modification; the results clearly demonstrate the benefit of a principled approach in estimating GVFs corresponding to policy gradients in an end-to-end algorithmic framework.

5 GVFs for Hierarchical Reinforcement Learning

Hierarchical reinforcement learning refers to a collection of methods which aim to structure the policy of an agent into "chunks" which have a temporal extent and can be re-used for different tasks. A large variety of methods have been proposed, including *options* (Sutton et al., 1999b; Precup, 2000), MAXQ (Dietterich, 2000), *feudal architectures* (Dayan & Hinton, 1993; Vezhnevets et al., 2017) and others. We will now illustrate how GVFs can be used in a natural way to analyze and extend such approaches. We will focus on the options framework and on feudal architectures, since these two approaches have recently been shown to yield algorithms that can learn the hierarchical structure from scratch (Bacon et al., 2017; Vezhnevets et al., 2017).

GVFs for options An option ω is defined as a tuple containing a set of states in which the option can be initiated, an internal policy $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which is used to pick actions when the option is executing, and a termination function $\beta_\omega : \mathcal{S} \to [0, 1]$. As in other recent work (Bacon et al., 2017), we ignore the initiation set in this paper. The typical execution model for options uses a policy over options, π_Ω, which chooses options from a set Ω. Whenever an option is picked, its policy chooses actions until the termination condition is met. Note that at a state s, an option ω will continue with probability $1 - \beta_\omega(s)$.

Options also have associated reward and transitions models (Sutton et al., 1999b). For example, the reward model of an option is given by:

$$\begin{aligned} r(s, \omega) &= E[\textstyle\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_t = s, \omega_t = \omega] \\ &= E[R(s, A_{t+1}) + \gamma(1 - \beta(S_{t+1}))r(S_{t+1}, \omega) | S_t = s, A_{t+1} \sim \pi_\omega(s, \cdot), S_{t+1} \sim p(\cdot|s, A_{t+1})] \end{aligned}$$

It is easy to see then that $r(s, \omega)$ can be written as a GVF which has as cumulant the environment reward R, the continuation function is $\gamma(1 - \beta_{\omega(s)})$ and the policy is π_ω. A similar development can be done for option transition models.

Another important quantity in the context of options is the option-value function, $Q_{\pi_\Omega}(s, \omega)$, which generalizes the action-value function, and is used to drive option construction algorithms such as option-critic. Indeed, the option-critic architecture (Bacon et al., 2017) uses a policy gradient algorithm to learn the internal policies and termination functions for a fixed number of options $\Omega = \{\omega_1, \cdots, \omega_k\}$. We now show that the option-value function can be expressed as a GVF.

Proposition 2. *Given a set of options Ω and a fixed policy over options π_Ω, define the cumulant C_ω for option ω as $C_\omega(s,a,s') = R(s,a) + \gamma\beta_\omega(s')E\left[Q(s',\omega') \mid \omega' \sim \pi_\Omega(\cdot\,|s')\right]$. Then:*

$$Q_{\pi_\Omega}(s,\omega) = v_{C_\omega,\gamma(1-\beta_\omega),\pi_\omega}(s,\omega).$$

The proof is in the appendix. Figure 4a illustrates the statement. The GVF view highlights the fact that the option-value function depends both on a short term signal coming from the reward and a long-term signal summarizing the performance of other options. This leads to a very hard task, since each option has to be aware of the performance of the rest of the options. In the option-critic framework, this can lead to degenerate solutions, e.g. when each option considers itself less qualified than the others and wishes to cede control immediately (which leads to all options collapsing to single-step actions), or when one option decides it is better than the others, and hence never terminates and tries to learn the optimal policy internally. Note that the option models, on the other hand, achieve a separation of concerns naturally. Hence, in the context of transfer learning, option models may be more useful than option-value functions.

GVFs for feudal networks Feudal Networks (FuN) (Vezhnevets et al., 2017) is an HRL approach designed to provide an explicit *separation of concerns* between levels of hierarchy. It uses a network which consists of two parts. The top level, or the Manager $\mathbf{M}$, chooses goals $g(\mathbf{M})$ at a slower time scale in a latent state-space that is itself *learned*. The lower level, or the Worker $\mathbf{W}$, operates at a faster time scale and produces primitive actions, conditioned on the goals received from the Manager. The Worker is motivated to follow the goals by a combination of intrinsic and extrinsic reward. No gradients are propagated between $\mathbf{W}$ and $\mathbf{M}$; $\mathbf{M}$ receives its learning signal from the environment alone, adjusting $g(\mathbf{M})$ to maximize the extrinsic reward using *transition policy gradient* (Vezhnevets et al., 2017).

We now show how GVFs can be used to provide a general version of the *transition policy gradient* introduced in Vezhnevets et al. (2017).

Proposition 3. *Let $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^k$ represent some latent variables used to model a hierarchical policy parameterised as: $\pi(a|s) = \int \pi(a|s,\mathbf{z};\theta_2)d\pi(\mathbf{z}|s;\theta_1)$. Let $\mathbf{v}$ be a set of random variables corresponding to predictions about π. Given continuation function γ and cumulant $C : \mathcal{S}\times\mathcal{A}\times\mathcal{Z} \to \mathbb{R}$, $\nabla_{\theta_1} v_{C,\gamma,\pi}(s) = v_{\hat{C}_1,\gamma,\pi}(s)$ and $\nabla_{\theta_2} v_{C,\gamma,\pi}(s) = v_{\hat{C}_2,\gamma,\pi}(s)$ with*

$$\hat{C}_{1,t} = v_{C,\gamma,\pi}(s_t,\mathbf{z}_t,\mathbf{v}_t)\nabla_{\theta_1}\log\pi(\mathbf{z}_t,\mathbf{v}_t|s_t;\theta_1),\ \textit{and}\ \hat{C}_{2,t} = v_{C,\gamma,\pi}(s_t,a_t,\mathbf{z}_t)\nabla_{\theta_2}\log\pi(a_t|s_t,\mathbf{z}_t;\theta_2).$$

The proof is in the appendix. This result allows the manager in FuN to move from fixed temporal difference in state representations (e.g. $\mathbf{v}_t := s_{t+c}$), to any general prediction conditioned on π. It also shows that the transition policy gradient can be used for broader choices of latent attributes than those used in Vezhnevets et al. (2017), where $\log\pi(\mathbf{z}_t,\mathbf{v}_t)$ was proportional to the cosine distance between $(s_{t+c} - s_t)$ and $\mathbf{z}_t$.

When $v_{C,\gamma,\pi}(s,a)$ in Prop. 3 is independent of $\mathbf{z}$, low-level action choices that ignore $\mathbf{z}$ (i.e. $\pi(a|s,\mathbf{z};\theta_2) = \pi(a|s;\theta_2)$) end up as valid local minima for the corresponding optimisation problem. One way to avoid this kind of degeneracy is to use intrinsic rewards corresponding to some notion of alignment between the observed trajectory and $\mathbf{z}$ (Dayan & Hinton, 1993; Barto et al., 2004; Kulkarni et al., 2016; Baranes & Oudeyer, 2013). Alternatively, one could use Universal Value Functions as GVFs, consider latents $\mathbf{z}$ that determine specific goals (i.e. (C,γ) pairs) and train low-level controllers to achieve those goals (Schaul et al., 2015).

5.1 Empirical illustration

To illustrate the ideas above, we have used the following GVFs: $\mathbf{v} := v_{C,\hat{\gamma},\pi}$, with $C_t = s_{t+c} - s_t$ and some discount factor $\hat{\gamma}$ possibly different from the discount of the GVF $v_{R,\gamma,\pi}$ used for evaluating the agent on the given task. We assumed a von Mises-Fisher distribution for the higher-level policy: $\pi(\mathbf{z}_t,\mathbf{v}_t|s_t) \propto \exp(d_{\cos}(\mathbf{z}_t,\mathbf{v}_t))$. The intrinsic reward for the low-level component was chosen as : $r_t^I = \sum_{j=0}^{t-c}\left(\frac{\hat{\gamma}}{\gamma}\right)^j \frac{(s_t - s_{t-c})^T\mathbf{z}_{t-c-j}}{\|\mathbf{v}_{t-c-j}\|\|\mathbf{z}_{t-c-j}\|}$. See Appendix for details on an efficient recursive computation of this reward and a proof that this corresponds to an intrinsic return $R^I = \sum_{i=0}^{\infty}\gamma^i\log\pi(\mathbf{v}_i,\mathbf{z}_i|s_i)$. We also use successor features (SF) for GVFs. We conducted a number of experiments in the ATARI games (Bellemare et al., 2013) for which the original FuN

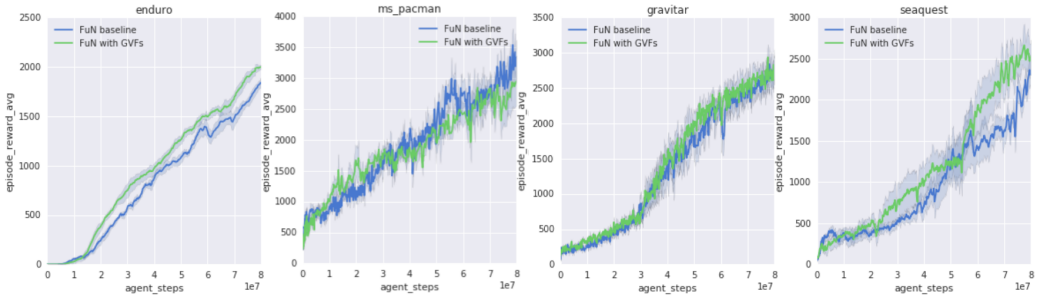


Figure 5: Training curves for *FuN with high-level choices over GVFs in ATARI games (Bellemare et al., 2013)*. In Seaquest and Enduro, agents for which the manager uses successor features consistently outperform the baseline algorithm. In Gravitar and Ms. Pacman, the proposed modifications have no impact on performance. See appendix for more results.

architecture was shown to exhibit substantial empirical improvements over corresponding network structures (Vezhnevets et al., 2017). The experimental setup and all the hyper-parameters were set to those used in the original paper on FuN (Vezhnevets et al., 2017). The results in Figure 5 show that using GVFs either reproduces the results in baseline FuN or improves the learning speed.

6 Other related work

Schaul et al. (2015) address the issue of learning a large number of GVFs individually, which is not scalable and misses out on learning shared feature structure among different GVFs. They propose that an agent may want to focus on learning a special type of GVF, which is optimal with respect to a specific minimum-time-to-goal based task. Such a task can be specified using a (C, γ) pair, in which C is positive at the goal state(s) and 0 otherwise, and $\gamma < 1$ is constant at all states except the goal, where it is 0. In general, however, we could have GVFs that are optimal with respect to any cumulant and continuation, which we can be denoted as $v^*_{C,\gamma}$. Note that, if C is multidimensional, additional requirements such as having a bounded output and an ordering on the co-domain of C are necessary in this case. In Universal Value Functions (UVFs), the task (C, γ) can be summarized through a goal state g, which can then be presented as an input, to allow an agent to generalize between tasks. Hence, in this approach, a shared approximator $v(s, g)$ is estimated from data.

Synthetic gradients (Jaderberg et al., 2016) allow a learner to estimate directly a gradient for certain input data points, without backpropagation. The idea is to use a secondary function approximator for this estimation. The policy gradient approach that we presented is similar in spirit, as we use a separate GVF as a gradient estimator. Further connections are left for future investigation. Auxiliary tasks Jaderberg et al. (2017) are an approach which uses secondary reward functions in order to learn a better internal representation for an RL agent. The idea of using many GVFs, corresponding to many different cumulants, is very much in the same spirit.

7 Conclusion and future work

We proposed to use GVFs as a unifying language for describing several popular reinforcement learning algorithms. This approach provides interesting connections between algorithms and helps us understand certain pathological behaviors (such as the collapse encountered in certain temporal abstraction algorithms). Moreover, we can use the GVF framework in order to generate new algorithms, by making different choices for how these functions are encoded. We illustrated this ability by generating new versions of policy gradient and of feudal networks. In the case of policy gradient, the proposed approach, which uses bootstrapping as well as successor features, greatly improves learning speed and stability. However, more experience is needed with this algorithm, especially in conjunction with function approximation.

GVFs can enable a natural injection of domain knowledge, as the designer of the system can specify cumulants and continuation functions, then let the system learn about all of them. We will investigate this approach in future work. GVFs promise to enable life-long, incremental learning, as an agent could add new GVFs to its repertoire over time, perhaps following a meta-learning or curriculum learning approach.

REFERENCES

V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva. Stochastic optimization. In *Engineering Cybernetics*, 1968.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

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

Andrew G. Barto, Satinder Singh, and Nuttapong Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Development and Learning*, pp. 112–119, 2004.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural Actor-Critic Algorithms. *Automatica*, 45(11), 2009.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NIPS*, 1993.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 2000.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. 2016. URL `http://arxiv.org/abs/1608.05343`.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

Vijaymohan Konda. Actor-critic algorithms. In *NIPS*, 2000.

Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016.

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. URL `http://arxiv.org/abs/1509.02971`.

Joseph Modayil, Adam White, and Richard S. Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 2014.

Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts, 2000.

R. Y. Rubinstein. *Some Problems in Monte Carlo Optimization*. Ph.D. thesis, 1969.

Tom Schaul and Mark B Ring. Better Generalization with Forecasts. In *IJCAI*, 2013.

Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML*, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Richard S Sutton and Brian Tanner. Temporal-difference networks. In *NIPS*, pp. 1377–1384, 2005.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999b.

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 *AAMAS*, 2011.

Richard Stuart Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.

Adam White. *Developing a predictive approach to knowledge*. PhD thesis, University of Alberta, 2015.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

A PROOFS

Proposition. *Let μ be the measure corresponding to predictions on one-hot cumulants, as defined in Section 3. For every General Value Function, $v(s_0; C, \gamma, \pi) = \int C d\mu(\cdot; s_0, \gamma, \pi)$.*

Proof. WLOG, assume C is defined over state-action pairs, i.e $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and fix a starting state s_0. Let $S_0, A_0, S_1, A_1, \cdot, S_n, A_n, \cdots$ be the random trajectory generated when following policy π from initial state $S_0 = s_0$ and using $\gamma(S_i)$ as a Bernoulli process that determines the termination of a trajectory. Following the definition of general value functions (see Section 2),

$$\begin{aligned}
\int C d\mu(\cdot; s_0, \gamma, \pi) &= \sum_{s,a} C(s,a) v_{\mathbb{I}(s,a),\gamma,\pi}(s_0) \\
&= \sum_{s,a} C(s,a) \mathbb{E}\left[\sum_{t=0}^{\infty} \left(\prod_{i=1}^{t} \gamma(S_i) \right) \mathbb{I}((S_t, A_t) = (s,a)) \;\middle|\; S_0 = s_0, A_{0:\infty} \sim \pi \right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \left(\prod_{i=1}^{t} \gamma(S_i) \right) \sum_{s,a} C(s,a) \mathbb{I}((S_t, A_t) = (s,a)) \;\middle|\; S_0 = s_0, A_{0:\infty} \sim \pi \right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \left(\prod_{i=1}^{t} \gamma(S_i) \right) C_t \;\middle|\; S_0 = s_0, A_{0:\infty} \sim \pi \right] = v(s_0; C, \gamma, \pi)
\end{aligned}$$

□

Theorem. *Sutton et al. (1999a). (Policy gradient theorem) Let π_θ be a policy parameterised by a vector θ, γ be a constant continuation function, and $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a one-dimensional cumulant. The gradient of the general value function $v_{C,\gamma,\pi_\theta}(s)$ with respect to θ is itself a general value function that depends on the cumulant:*

$$\hat{C}(s,a) := v_{C,\gamma,\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s).$$

Proof. (sketch) Recall from Section 3 that $v(s; C, \gamma, \pi) = \int C d\mu(\cdot; s, \gamma, \pi)$ and $v_{C,\gamma,\pi}(s, a) = \int C d\mu(\cdot; s, a, \gamma, \pi)$. The derivation below follows,

$$\begin{aligned}\nabla_\theta v_{C,\gamma,\pi(\theta)}(s_0) &= \int \textstyle\sum_{a\in\mathcal{A}} v_{C,\gamma,\pi}(s,a)\nabla_\theta \pi(a|s;\theta) d\mu(s; s_0, \gamma, \pi) \\ &= \int \textstyle\sum_{a\in\mathcal{A}} \pi(a|s;\theta)\left(v_{C,\gamma,\pi}(s,a)\frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)}\right) d\mu(s; s_0, \gamma, \pi) \\ &= \int v_{C,\gamma,\pi}(s,a)\nabla_\theta \log \pi(a|s;\theta) d\mu(s, a; s_0, \pi(\theta), \gamma).\end{aligned}$$

□

Proposition. *Fix a set of options Ω and a meta-policy π^Ω and define the cumulant $C(\omega)$ for every option ω as $C_t(\omega) = R(S_t, A_t) + \gamma\beta(S_{t+1};\omega)E\left[Q(S_{t+1}, \omega') \mid \omega' \sim \pi^\Omega(S_{t+1})\right]$. Then, $Q(s_0, \omega) = v(s_0; C(\omega), \gamma(1-\beta(\omega)), \pi(\omega))$.*

Proof. Let $V(s) := E[Q(s,\omega)|\omega \sim \pi^\Omega]$. Based on the definition provided in Bacon et al. (2017),

$$\begin{aligned}Q(s_0,\omega) &= \sum_{a\in\mathcal{A}} \pi(a|s_0)\left[R(s_0,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s_0,a)\left[(1-\beta(s';\omega)Q(s',\omega) + \beta(s';\omega)V(s')\right]\right] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s_0)\left[R(s_0,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s_0,a)\beta(s';\omega)V(s')\right] + \\ &\qquad + \sum_{a\in\mathcal{A}} \pi(a|s_0)\gamma \sum_{s'\in\mathcal{S}} P(s'|s_0,a)(1-\beta(s';\omega)Q(s',\omega) \\ &= E\left[C(s_0, A, S';\omega) \mid A \sim \pi(s_0); S' \sim P(\cdot|s_0, A)\right] \\ &\qquad + E\left[\gamma(1-\beta(S';\omega))Q(S',\omega) \mid A \sim \pi(s_0); S' \sim P(\cdot|s_0, A)\right].\end{aligned}$$

Note that the statement above is equivalent to the Bellman equation for General Value Functions, where the cumulant is $C(\omega)$ and continuation function is $\gamma(1-\beta(\omega))$. Since the Bellman equation is known to have a unique solution, the statement of the proposition follows as a direct consequence. □

Proposition. *Assume latent attributes $\mathbf{z}$ are elements of $\mathcal{Z} \subseteq \mathbb{R}^k$. Let θ_1 and θ_2 be two sets of parameters used to model distributions $\pi(\mathbf{z}|s;\theta_1)$ and $\pi(a|s,\mathbf{z};\theta_2)$, correspondingly. These models describe the low-level action choice $\pi(a|s) = \int \pi(a|s,\mathbf{z})d\pi(\mathbf{z}|s)$. Let γ be a constant continuation function, and C be any cumulant defined on $\mathcal{S}\times\mathcal{A}\times\mathcal{Z} \to \mathbb{R}$. Additionally, let $\mathbf{v}$ be a set of random variables corresponding to predictions about the policy π. Then $\nabla_{\theta_1} v(s_0; C, \gamma, \pi) = v(s_0; \hat{C}_1, \gamma, \pi)$ and $\nabla_{\theta_2} v(s_0; C, \gamma, \pi) = v(s_0; \hat{C}_2, \gamma, \pi)$ with*

$$\begin{aligned}\hat{C}_{1,t} &= v(s_t, \mathbf{z}_t, \mathbf{v}_t; C, \gamma, \pi)\nabla_{\theta_1} \log \pi(\mathbf{z}_t, \mathbf{v}_t|s_t;\theta_1), and \\ \hat{C}_{2,t} &= v(s_t, a_t, \mathbf{z}_t; C, \gamma, \pi)\nabla_{\theta_2} \log \pi(a_t|s_t, \mathbf{z}_t;\theta_2).\end{aligned}$$

Proof. The statement is a corollary to Theorem 1, where the gradient with respect to the flat policy is:

$$\begin{aligned}\nabla_{\theta_1}\pi(a,\mathbf{z},\mathbf{v}|s;\theta_1,\theta_2) &= \nabla_{\theta_1}\pi(a|\mathbf{z},s;\theta_2)\pi(\mathbf{z},\mathbf{v}|s;\theta_1) = \pi(a|\mathbf{z},s;\theta_2)\nabla_{\theta_1}\pi(\mathbf{z},\mathbf{v}|s;\theta_1) \\ &= \pi(a,\mathbf{z},\mathbf{v}|s;\theta_1,\theta_2)\nabla_{\theta_1}\log\pi(\mathbf{z},\mathbf{v}|s;\theta_1) \\ \nabla_{\theta_2}\pi(a|s;\theta_1,\theta_2) &= \nabla_{\theta_2}\pi(a|\mathbf{z},s;\theta_2)\pi(\mathbf{z},\mathbf{v}|s;\theta_1) = \pi(\mathbf{z},\mathbf{v}|s;\theta_1)\nabla_{\theta_2}\pi(a|\mathbf{z},s;\theta_2) \\ &= \pi(a,\mathbf{z},\mathbf{v}|s;\theta_1,\theta_2)\nabla_{\theta_2}\log\pi(a|\mathbf{z},s;\theta_2)\end{aligned}$$

Now, when computing gradients,

$$\begin{aligned}
\nabla_{\theta_2} v(s_0; C, \gamma, \pi) &= \int \sum_{a,\mathbf{z},\mathbf{v}} v(s, a, \mathbf{z}, \mathbf{v}; C, \gamma, \pi) \nabla_{\theta_2} \pi(a, \mathbf{z}, \mathbf{v}|s; \theta_1, \theta_2) d\mu(s; s_0, \gamma, \pi) \\
&= \int \sum_{a,\mathbf{z},\mathbf{v}} v(s, a, \mathbf{z}, \mathbf{v}; C, \gamma, \pi) \pi(a, \mathbf{z}, \mathbf{v}|s; \theta_1, \theta_2) \nabla_{\theta_2} \log \pi(a|\mathbf{z}, s; \theta_2) d\mu(s; s_0, \gamma, \pi) \\
&= \int \sum_{a,\mathbf{z}} v(s, a, \mathbf{z}; C, \gamma, \pi) \pi(a, \mathbf{z}|s; \theta_1, \theta_2) \nabla_{\theta_2} \log \pi(a|\mathbf{z}, s; \theta_2) d\mu(s; s_0, \gamma, \pi) \\
&= \int v(s, a, \mathbf{z}; C, \gamma, \pi) \nabla_{\theta_2} \log \pi(a|\mathbf{z}, s; \theta_2) d\mu(s, a, \mathbf{z}; s_0, \gamma, \pi) \\
&= v(s_0; \hat{C}_2, \gamma, \pi)
\end{aligned}$$

A similar results can easily be derived for θ_1 to obtain the gradient updates described in the Proposition. □

Proposition 4. *Let $s_t \in \mathbb{R}^K$ represent the observation at time t and $\mathbf{z}_t \in \mathbb{R}^K$ be the latent attributes used to model the hierarchical policy described in Section* **??**. *Define the intrinsic reward r_t^I at time t to be*

$$r_t^I = \textstyle\sum_{j=0}^{t-c} \left(\frac{\hat{\gamma}}{\gamma}\right)^j \frac{(s_t - s_{t-c})^T \mathbf{z}_{t-c-j}}{\|\mathbf{v}_{t-c-j}\| \|\mathbf{z}_{t-c-j}\|}. \tag{1}$$

Given a sequence $s_0, \mathbf{z}_0, s_1, \mathbf{z}_1, \cdots, s_T, \mathbf{z}_T$, the intrinsic rewards r_t^I can be computed iteratively, and $R^I = \sum_{t=0}^{\infty} \gamma^t r_t^I = \sum_{i=0}^{\infty} \gamma^i \log \pi(\mathbf{v}_i, \mathbf{z}_i|s_i)$, where $\log \pi(\mathbf{v}_i, \mathbf{z}_i|s_i)$ is assumed to follow von Mises-Fisher distribution.

Proof. Note that this reward value can be efficiently computed using the concept of a running sum. Define the vector $\mathbf{m}_t$ at every timestep to be $\mathbf{m}_t := \sum_{j=0}^{t-c} \left(\frac{\hat{\gamma}}{\gamma}\right)^j \frac{\mathbf{z}_{t-c-j}}{\|\mathbf{v}_{t-c-j}\| \|\mathbf{z}_{t-c-j}\|}$; it is not hard to check that this can be computed iterative using the following update: $\mathbf{m}_t := \frac{\mathbf{z}_{t-c}}{\|\mathbf{v}_{t-c}\| \|\mathbf{z}_{t-c}\|} + \frac{\hat{\gamma}}{\gamma} \mathbf{m}_{t-1}$. From this, computing the intrinsic reward amounts to $r_t^I = (s_t - s_{t-c})^T \mathbf{m}_t$.

Apart from being computationally convenient, the intrinsic reward in Equation 1 has a corresponding return which is maximized when the latent values $\mathbf{z}$ align with predictions $\mathbf{v}$. That is,

$$\begin{aligned}
R^I = \sum_{t=0}^{\infty} \gamma^t r_t^I &= \sum_{t=0}^{\infty} \gamma^t \sum_{j=0}^{t-c} \left(\frac{\hat{\gamma}}{\gamma}\right)^j \frac{(s_t - s_{t-c})^T \mathbf{z}_{t-c-j}}{\|\mathbf{v}_{t-c-j}\| \|\mathbf{z}_{t-c-j}\|} \\
&= \sum_{t=0}^{\infty} \sum_{j=0}^{t-c} \gamma^{t-j} \hat{\gamma}^j \frac{(s_t - s_{t-c})^T \mathbf{z}_{t-c-j}}{\|\mathbf{v}_{t-c-j}\| \|\mathbf{z}_{t-c-j}\|} \\
&= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \gamma^{i+c} \hat{\gamma}^j \frac{(s_{i+j+c} - s_{i+j})^T \mathbf{z}_i}{\|\mathbf{v}_i\| \|\mathbf{z}_i\|} = \gamma^c \sum_{i=0}^{\infty} \gamma^i \log \pi(\mathbf{v}_i, \mathbf{z}_i|s_i).
\end{aligned}$$

□

B ACTOR-CRITIC WITH GVFS FOR GRADIENT ESTIMATION

Algorithm 2 describes the modifications we hade made to the Actor-Critic algorithm (Konda, 2000) in order to illustrate the benefits of bootstraping not only the values of the critic function, but the general value functions involved in the computation of policy gradients.

To generate the results presented in Figure 3, we first defined a tile-coding feature map $\phi : \mathcal{S} \to \mathbb{R}^K$ (Sutton & Barto, 1998). The policy is parameterized with a vector $\boldsymbol{\theta} \in \mathbb{R}^{K \times |\mathcal{A}|}$ and the critic values with $\mathbf{w} \in \mathbb{R}^K$ using the following functional forms: $\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \phi(s)$ and $\pi(a|s, \boldsymbol{\theta}) = \text{Softmax}(\boldsymbol{\theta}_a^T \phi(s))$. For pol-

icy gradients estimation, we used parameters $\boldsymbol{\eta} \in \mathbb{R}^{K \times |\mathcal{A}|}$ and $\hat{g}(s, \boldsymbol{\eta}) = \boldsymbol{\eta}^T \phi(s)$.

Algorithm 2: Actor-Critic with GVFs for gradient estimation.

Input: policy $\pi(a|s, \boldsymbol{\theta})$; critic state-value fn. $\hat{v}(s, \mathbf{w})$; general-value fn. $\hat{g}(s, \boldsymbol{\eta})$.
Algorithm parameters: step size $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$, and $\alpha^{\boldsymbol{\eta}} > 0$.
repeat
  Initialize S (first state of episode); $I \leftarrow 1$
  repeat
    $A \sim \pi(a|s, \boldsymbol{\theta})$ and take action A, observe S', R
    $\delta_{\text{crit}} \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta_{\text{crit}} \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$
    $\delta_{\text{grad}} \leftarrow \delta_{\text{crit}} \nabla_{\theta} \log \pi(A|S, \boldsymbol{\theta}) + \gamma \hat{g}(S', \boldsymbol{\eta}) - \hat{g}(S, \boldsymbol{\eta})$
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \alpha^{\boldsymbol{\eta}} I \delta_{\text{grad}} \nabla_{\boldsymbol{\eta}} \hat{g}(S, \boldsymbol{\eta})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \hat{g}(S, \boldsymbol{\eta})$
    $I \leftarrow \gamma I$ and $S \leftarrow \gamma S'$
  until *end of episode*
until *convergence*

C Details of GVF-FuN Experiments

We follow the same experimental procedure described in (Vezhnevets et al., 2017), where hyper-parameters are tuned using randomized search on the following domain: the learning rate for the policy gradient updates is sampled from Log-Uniform$(10^{-4.5}, 10^{-3.5})$; the entropy penalty was sampled from Log-Uniform$(10^{-4}, 10^{-3})$; the weight of the intrinsic reward was sampled from Uniform$(0, 1)$. Moreover, to increase stability, the reward is clipped in $[-1, 1]$.

For the purpose of illustrating the concept of separation of concerns, the worker and the manager were trained with respect to GVFs of different discount values: manager's discount $\gamma_{\mathbf{M}} = 0.99$ and worker's discount $\gamma_{\mathbf{W}} = 0.95$.

We compare baseline FuN with a modified network where the goal g picked by the manager corresponds to the latent $\mathbf{z}$ in Proposition 3 and the manager is trained with transition policy gradient $\nabla g_t = A_t^M d_{\cos}(\mathbf{v}_t, g_t(\theta))$. Since the prediction horizon of the baseline FuN agent is set to $c = 10$, we compute successor features v_t using a discount value $\gamma = 0.8$ to make sure that the corresponding temporal resolution is essentially the same[2]. The intrinsic reward corresponding to specific goals set by the manager was computed using Equation 1. Since our intent was to test the hypothesis that a Feudal Network would be capable of reasoning over successor features, all other components were kept invariant.

See Figure 9 for results for all the games for which we have ran the experiments. Specifically, the plots in the figure provide the average and standard deviations over 5 different runs of the algorithm, where each run was based on a randomized search picking the top performing hyper-parameter setting among 20 samples, and the policy gradient algorithm is performed over 80 epochs[3].

[2] When a GVF is using a constant discount value γ and no explicit horizon length is used, rewards obtained after $\log_{\gamma} \epsilon$ become numerically insignificant (i.e. $\leq \epsilon R_{max}$).

[3] A training epoch is equivalent to 10^6 steps.

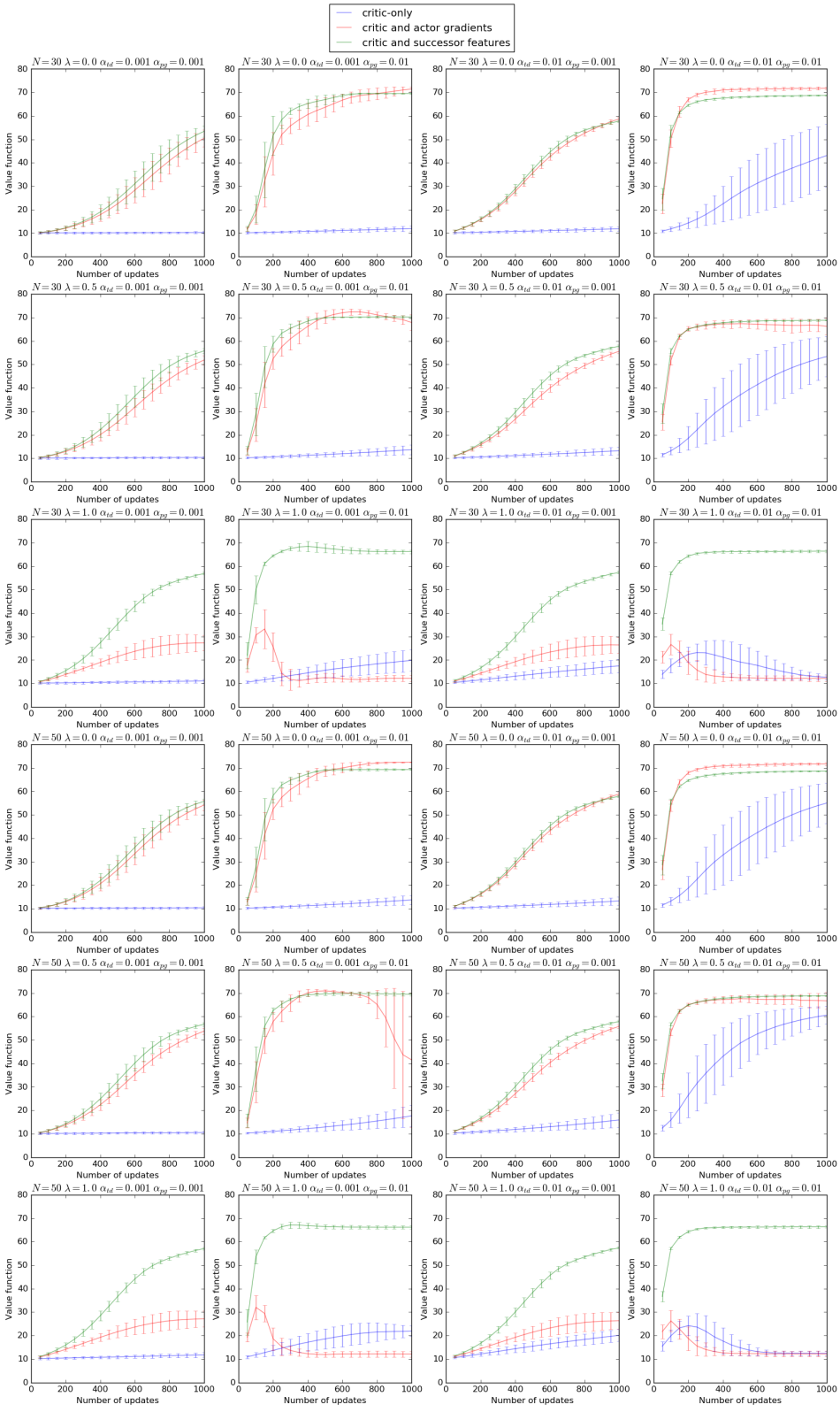


Figure 6: **Policy gradient updates with TD(λ) estimates.** *L_1 norm of the exact value function vs. number of updates to policy parameters*. Each plot corresponds to a fixed number N of rollouts for estimating GVFs, a fixed λ as eligibility for TD(λ), fixed learning rate for α_{td}, and fixed learning rate α_{pg} for policy parameters. Mean and variance for the value of the policy as a function of policy updates are shown in (blue) when only the critic value is bootstrapped, in (red) when all GVF estimates are bootstrapped, and (green) when successor features are used to stabalized the learning algorithm.

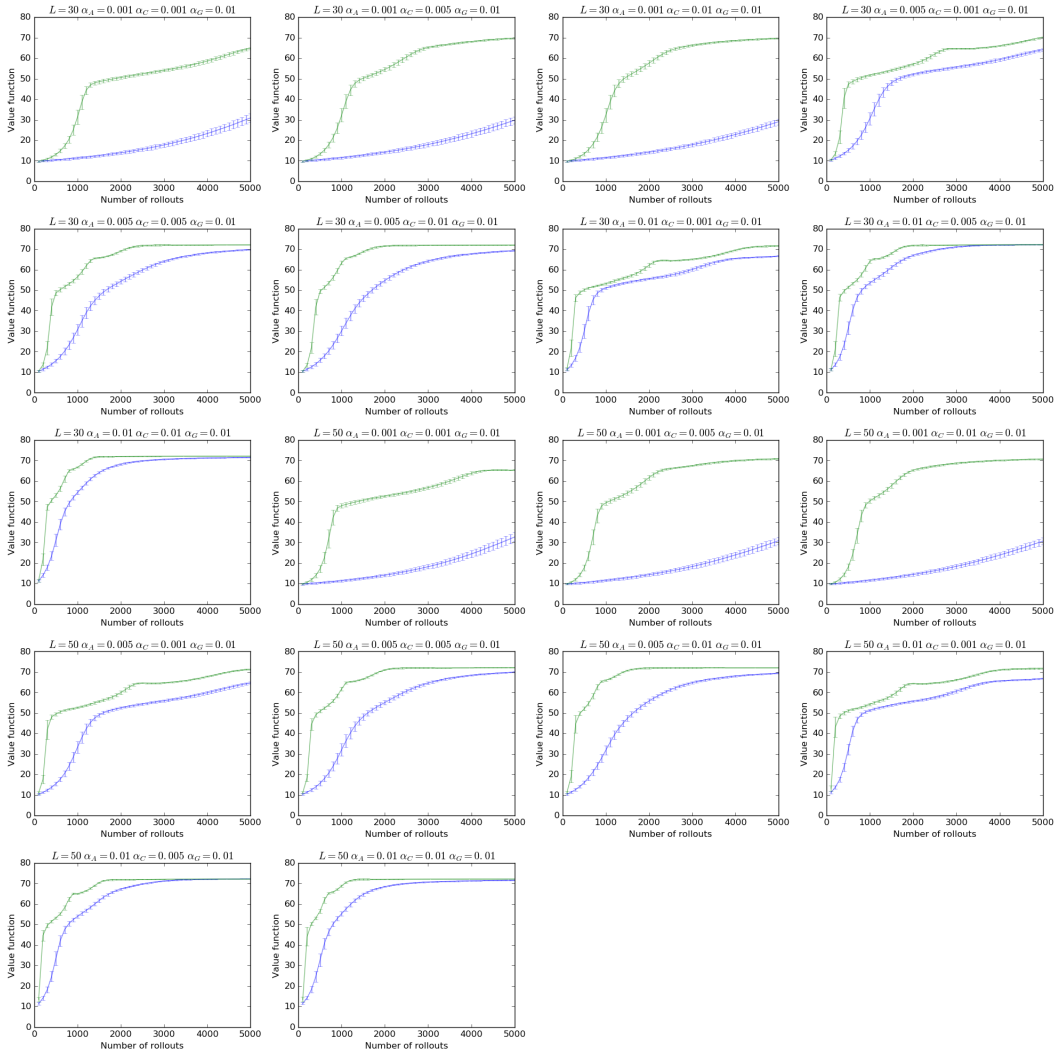


Figure 7: **Actor-Critic with bootstrapped gradient estimates – results on four-room maze (introduced in (Sutton et al., 1999a))** *L_1 norm of the exact value function vs. number of rollouts used for policy updates*. Each plot corresponds to a fixed number of steps L taken by the agent for every rollout, the learning rate α_A for updates to the policy parameters, a learning rate α_C for updates to the critic component, and a learning rate α_G for updates to the gradient estimates. Mean and variance for the value of the policy as a function of policy updates are shown in (blue) for baseline actor-critic, in (green) when actor-critic maintains bootstrapped estimates for parameter updates.

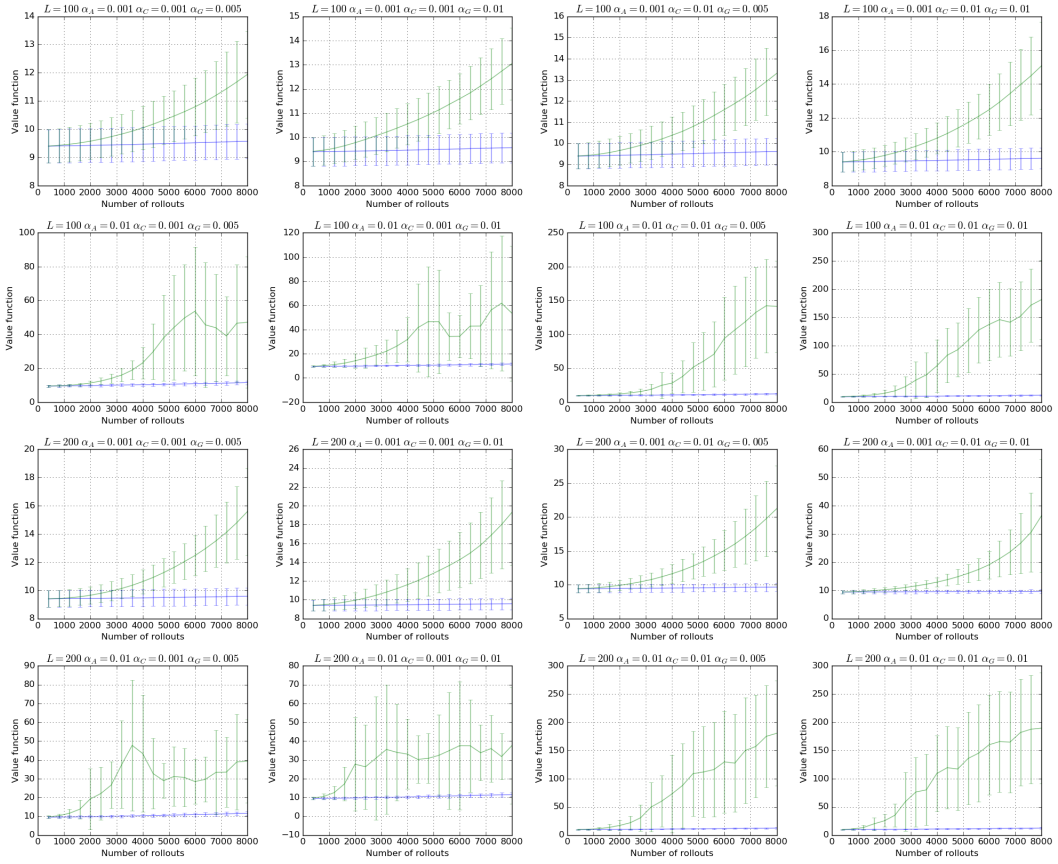


Figure 8: **Actor-Critic with bootstrapped gradient estimates – results on large maze grid-world with sparse reward.** *L_1 norm of the exact value function vs. number of rollouts used for policy updates*. Each plot corresponds to a fixed number of steps L taken by the agent for every rollout, the learning rate α_A for updates to the policy parameters, a learning rate α_C for updates to the critic component, and a learning rate α_G for updates to the gradient estimates. Mean and variance for the value of the policy as a function of policy updates are shown in (blue) for baseline actor-critic, in (green) when actor-critic maintains bootstrapped estimates for parameter updates.

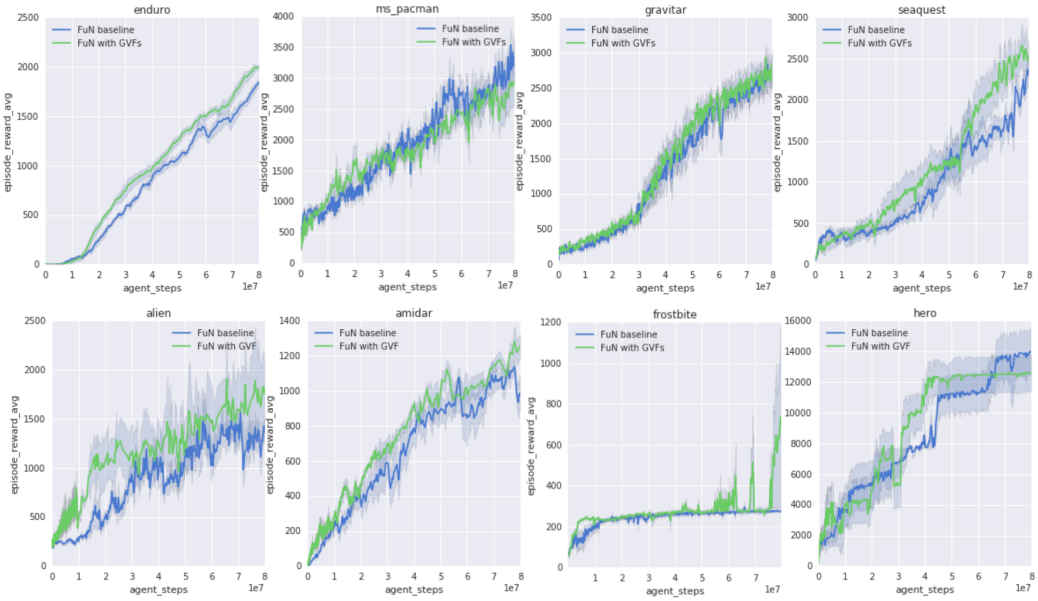


Figure 9: Training curves for *FuN with high-level choices over GVFs*. Results on ATARI games (Bellemare et al., 2013) demonstrate gains in data-efficiency for some ATARI games. In games such as Seaquest, Alien, and Enduro, agents for which the manager uses successor features consistently outperform the baseline algorithm. In Gravitar, Amidar and Ms. Pacman, the proposed modifications have no impact on performance. In Frostbite, both versions of the algorithm are usually stuck in a local minimum (average reward ≈ 250), and the noticeable improvement is due to one of the agents luckily getting out of the local minimum for a limited number of steps.