General Value Functions, General Policy Evaluation and General Policy Improvement

With thanks to Rich Sutton, Satinder Singh, Gheorghe Comanici, Anna Harutyunyan, Andre Barreto, David Silver, Pierre-Luc Bacon, Jean Harb, Shibl Mourad, Khimya Khetarpal, Zafarali Ahmed, David Abel, Sasha Vezhnevets, Shaobo Hou, Philippe Hamel, Eser Aygun, Diana Borsa, Justin Novosad, Will Dabney, Nicholas Heess, Remi Munos

COMP579 Lecture 20, 2024

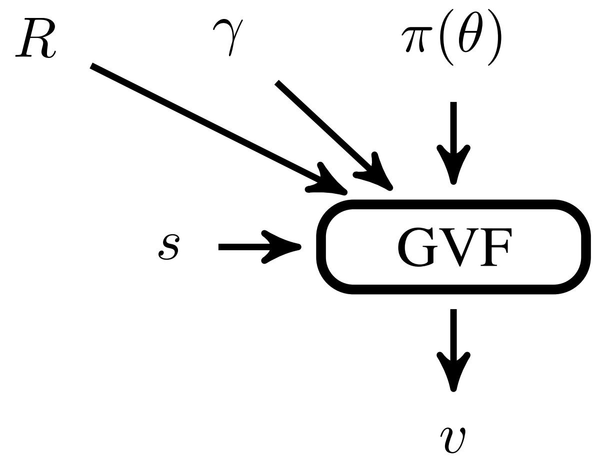
Recall: General Value Functions (GVFs)

- Given a cumulant function c, state-dependent continuation function γ and policy π, the General Value Function $v_{\pi,\gamma,c}$ is defined as:

$$v_{\pi,c,\gamma}(s) = \mathbf{E}\left[\sum_{k=t}^{\infty} c(S_k, A_k, S_{k+1}) \prod_{i=t+1}^{k} \gamma(S_i) | S_t = s, A_{t:\infty} \sim \pi\right]$$

- *Cumulant* c can output a vector (even a matrix)
- *Continuation function* γ maps states to [0,1] (further generalizations are possible)
- Cf. Horde architecture (Sutton et al, 2011); Adam White's thesis; inspiration from Pandemonium architecture
- Special case: policy is optimal wrt c, γ, $v^*_{c,\gamma}$ - Universal Value Function approximation (UVFA) (Schaul et al, 2015)
- No single task is required, just a multitude of cumulants and time scales!

GVFs as building blocks of knowledge



- Note that one can take the output of a GVF and make it an input to another GVF
- Or, the output of a GVF could become part of the "state" for another GVF

Option models are GVFs

- The reward model for an option ω is defined as:

$$r_\omega(s) = \mathbb{E}_\omega[r(S_t, A_t) + \gamma(1 - \beta_\omega(S_{t+1}))r_\omega(S_{t+1})|S_t = s]$$

- This means the option reward model is a GVF:
  - policy is π_ω
  - *cumulant* is the environment reward r
  - *continuation function* is $\gamma(1 - \beta_\omega)$
- Option transition model can be similarly written as a GVF

Many other approaches that can be expressed as GVFs

- Option-value functions (Precup, 2000; Sutton, Precup & Singh, 1999)
- Feudal networks (Dayan, 1994; Vezhnevets et al, 2017)
- Value transport (Hung et al, 2018)
- Auxilliary tasks (Jaderberg et al, 2016)
- *Are GVFs just an interesting insight or can they be useful?*

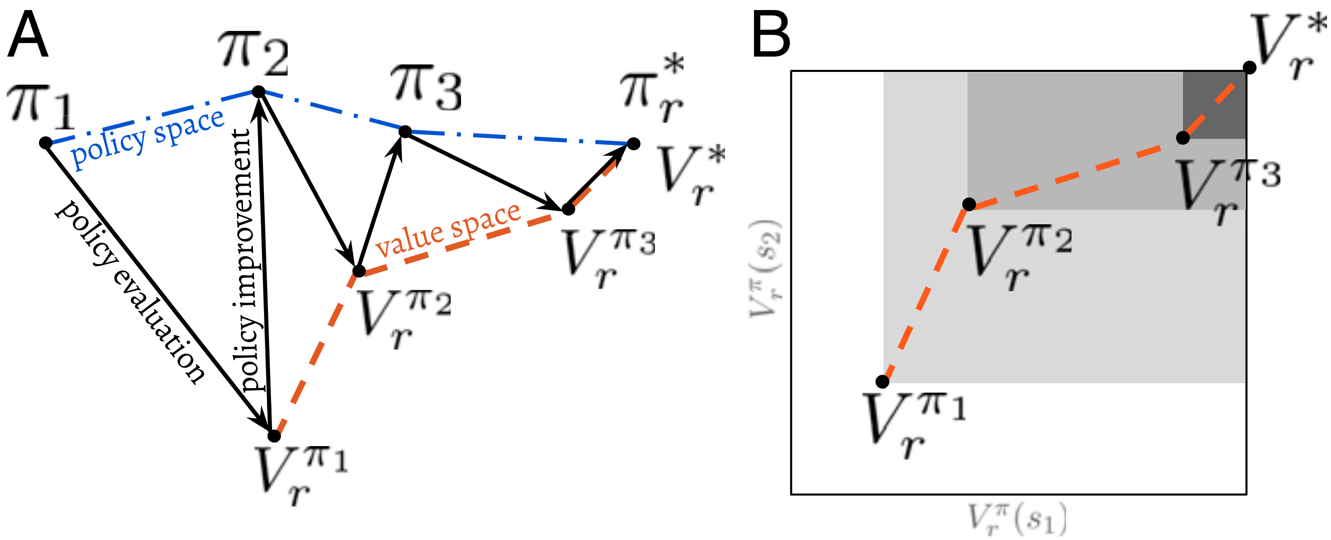
Policy Evaluation and Policy Improvement

- Consider a Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ and a policy $\pi : \mathcal{S} \to Dist(\mathcal{A})$
- Classic dynamic programming relies on two basic operations:
  - *Policy evaluation*: given policy π, compute the value function V_r^π and/or Q_r^π
  - *Policy improvement*: given value function Q_r^π, compute an improved policy: $\pi'(s) = \arg\max_{a' \in \mathcal{A}} Q_r^\pi(s, a')$
- Policy improvement guarantee:

$$Q_r^{\pi'}(s, a) \geq Q_r^\pi(s, a), \, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

- Dynamic programming: interleave these steps (executed exactly)
- Reinforcement learning: carry out these steps approximately

Visualizing Policy Evaluation and Policy Improvement



- Generalize this process to *multiple reward functions (ie tasks)* $r \in \mathcal{R}$ and *multiple policies* $\pi \in \Pi$

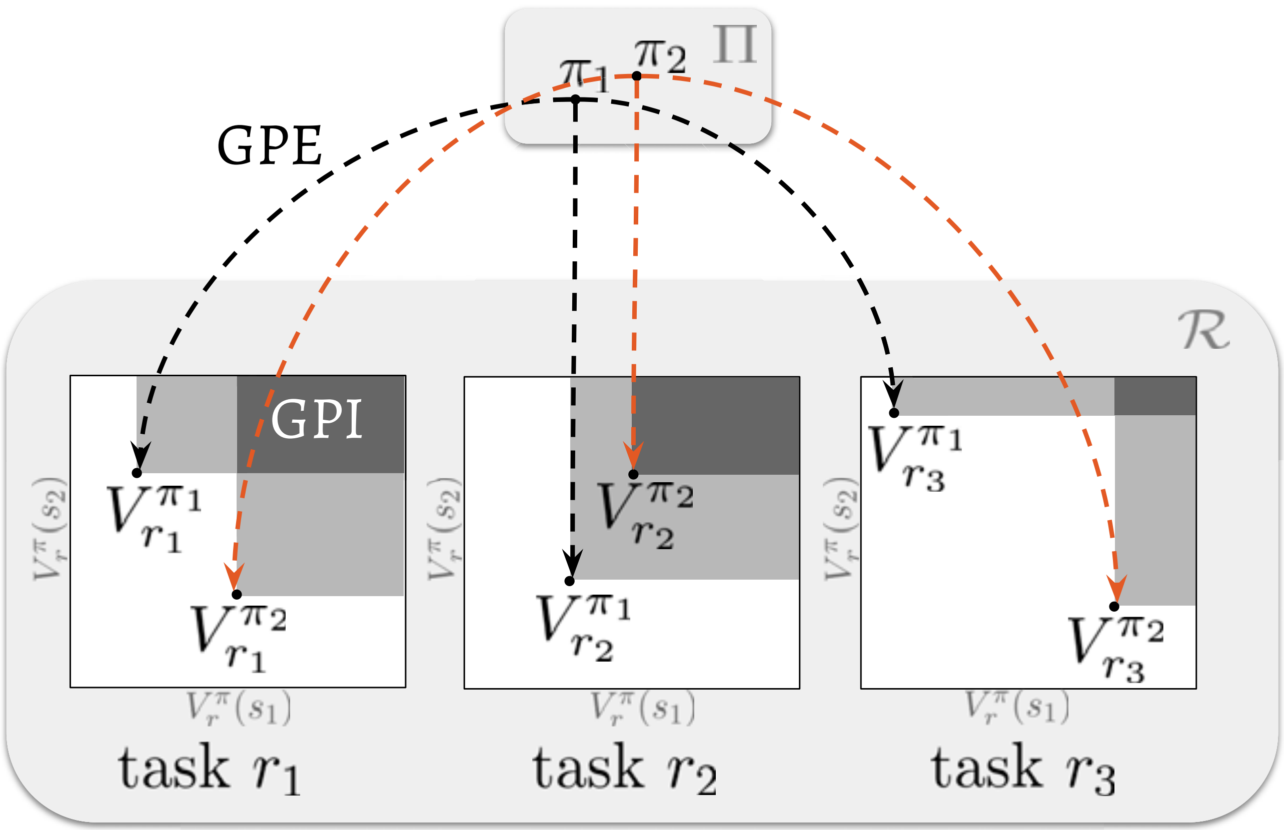
Generalized Policy Updates

- *Generalized policy evaluation (GPE)*: compute the value of a policy π on a set of reward functions $\mathcal{R}$
- *Generalized policy improvement (GPI)*: given a set of policies Π and a reward function r, compute a new policy such that:

$$Q_r^{\pi'}(s,a) \geq \sup_{\pi \in \Pi} Q_r^{\pi}(s,a),\ \forall s \in \mathcal{S} \forall a \in \mathcal{A}$$

- If we have only one r and one π, we recover usual policy evaluation and policy improvement

Visualizing Generalized Policy Updates



Fast Generalized Policy Evaluation

- If we had a nice map from r to Q_r^π, GPE could be efficient
- Consider the class of reward functions that are linear in some feature space $\phi(s,a)$:

$$r_{\mathbf{w}}(s,a) = \mathbf{w}^T \phi(s,a) \text{ and } \mathcal{R}_\phi = \{r_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^d\}$$

  Note that ϕ can be learned and non-linear
- *Successor features*: $\psi^\pi(s,a) = \mathbf{E}_\pi[\sum_{t=1}^\infty \gamma^t \phi(s_t,a_t) | s_0 = s, a_0 = a]$
- Then the value function for a specified reward function can be easily computed as a function of the successor features:

$$Q_{\mathbf{w}}^\pi(s,a) = \mathbf{w}^T \psi^\pi(s,a)$$

- *Successor features can be pre-computed for π once and re-used thereafter (a form of model!)*
- Connections to hippocampus representations

Successor states and successor features are GVFs

- *Successor features* (Barreto et al, 2017, 2018) are a natural extension of successor states (Dayan, 1992)
- Successor states give the expected occupancy of future states
- If states are defined by a feature vector $\phi(s)$, successor features give the expected, discounted sum of future feature vectors from a state.
- In GVF terms, the *cumulant is $c = \phi$*, and there is a fixed policy and discount
- Interesting property highlighted in Barreto et al:

$$v_{\pi,\mathbf{w}^T c,\gamma}(s) = \mathbf{w}^T v_{\pi,c,\gamma}(s)$$

which leads to one-shot computation of new GVFs

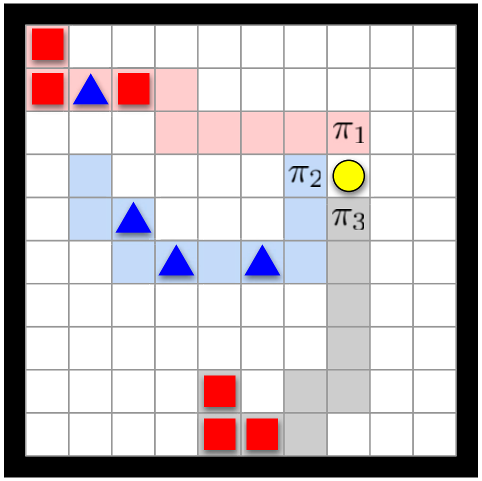
Fast Generalized Policy Improvement

- Compute the improved policy as:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} Q_r^{\pi}(s,a)$$

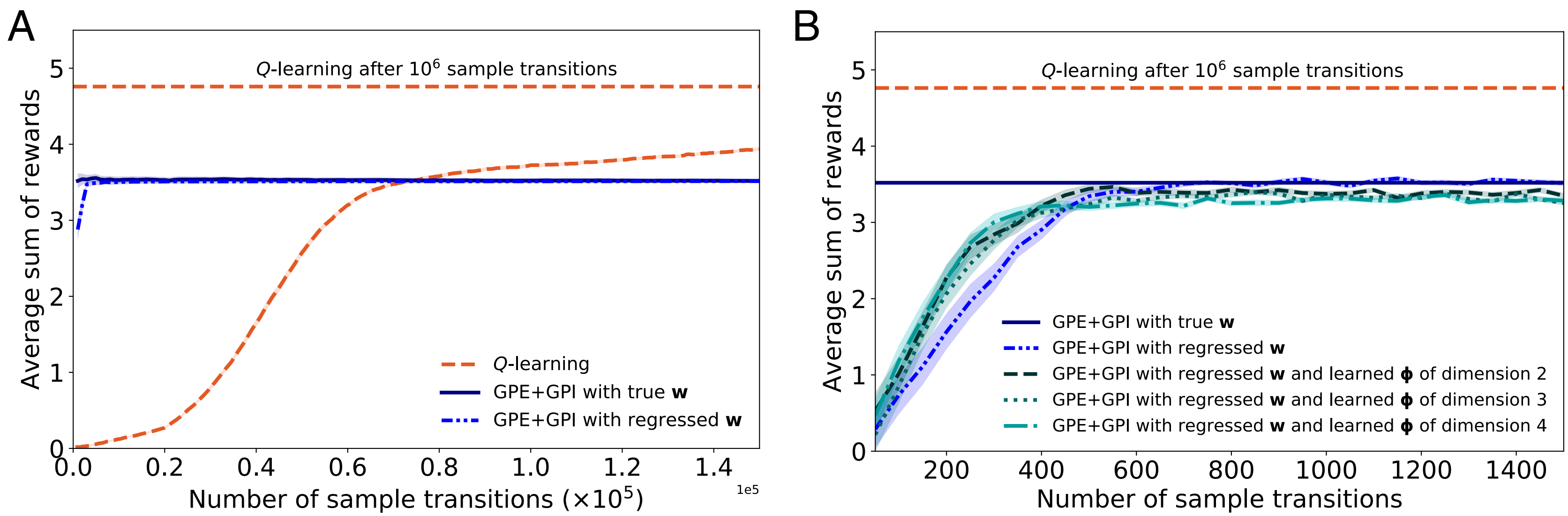
- Note that π' *could choose actions that are not chosen by any of the* π
- The process takes only *one iteration*, after which no further change to the policy π' would happen
- In contrast with iterative policy improvement...

Illustration



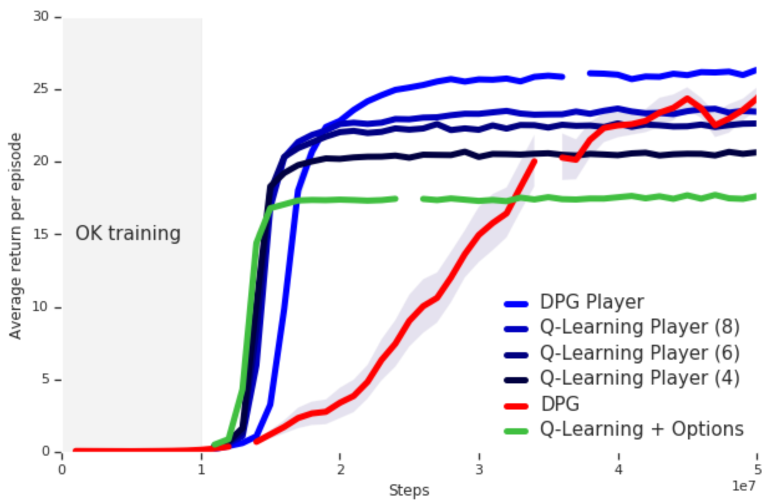
- The three policies correspond to three weight vectors: like red ($\mathbf{w}_1 = [1, 0]^T$), like blue ($\mathbf{w}_2 = [0, 1]^T$) and like red not blue ($\mathbf{w}_3 = [1, -1]^T$)
- *Note that* $\mathbf{w}$ *can be viewed as a preference function over features!*
- We can pre-train the policies that optimize for each preference, and train their successor features as well
- Then just do GPE/GPI!

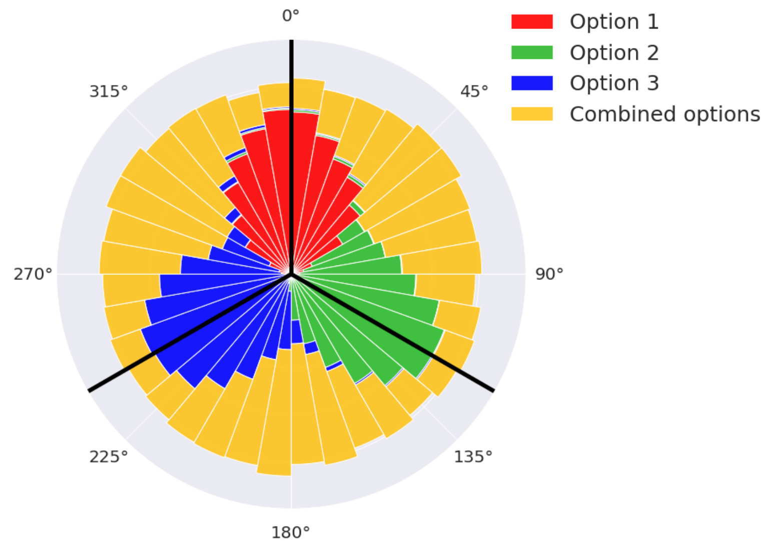
Illustration: Results



- Training the successor features for $\mathbf{w}_1$, $\mathbf{w}_2$ over 5×10^5 samples then GPE/GPI for $\mathbf{w}_3$
- GPE/GPI with successor features achieves *75x improvement in sample size* compared to Q-learning
- Obtaining $\mathbf{w}$, ϕ by learning almost as good as knowing these in advance

Synthesizing new behavior: Moving Target Arena





General way to synthesize quickly new behavior for combinations of reward functions!