More on Hierarchical Reinforcement Learning

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

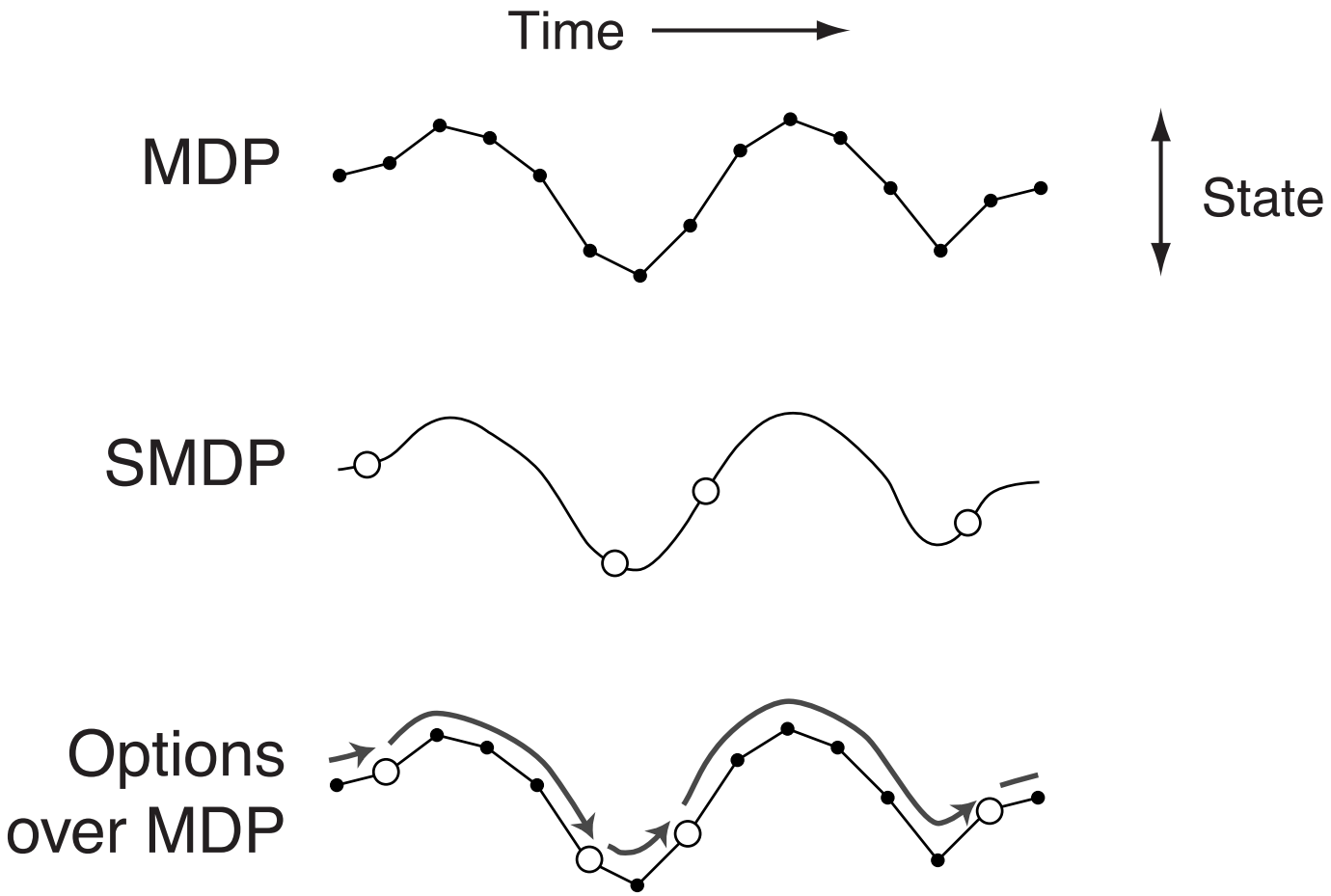
COMP579 Lecture 18, 2025

Recall: Options

- An *option* ω consists of 3 components
  - An *initiation set* $I_\omega \subseteq \mathcal{S}$ (aka precondition)
  - A *policy* $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$
    $\pi_\omega(a|s)$ is the probability of taking a in s when following option ω
  - A termination condition $\beta_\omega : \mathcal{S} \to [0, 1]$:
    $\beta_\omega(s)$ is the probability of terminating the option ω upon entering s
- Eg., robot navigation: if there is no obstacle in front (I_ω), go forward (π_ω) until you get too close to another object (β_ω)
- Inspired from macro-actions / behaviors in robotics / hybrid planning and control

Cf. Sutton, Precup & Singh, 1999; Precup, 2000

Recall: Decision-Making with Options

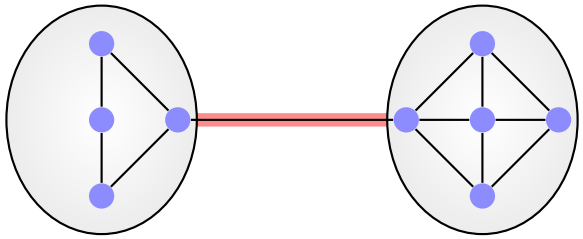


Learning and planning algorithms are the same at all levels of abstraction!

How Should Options Be Created?

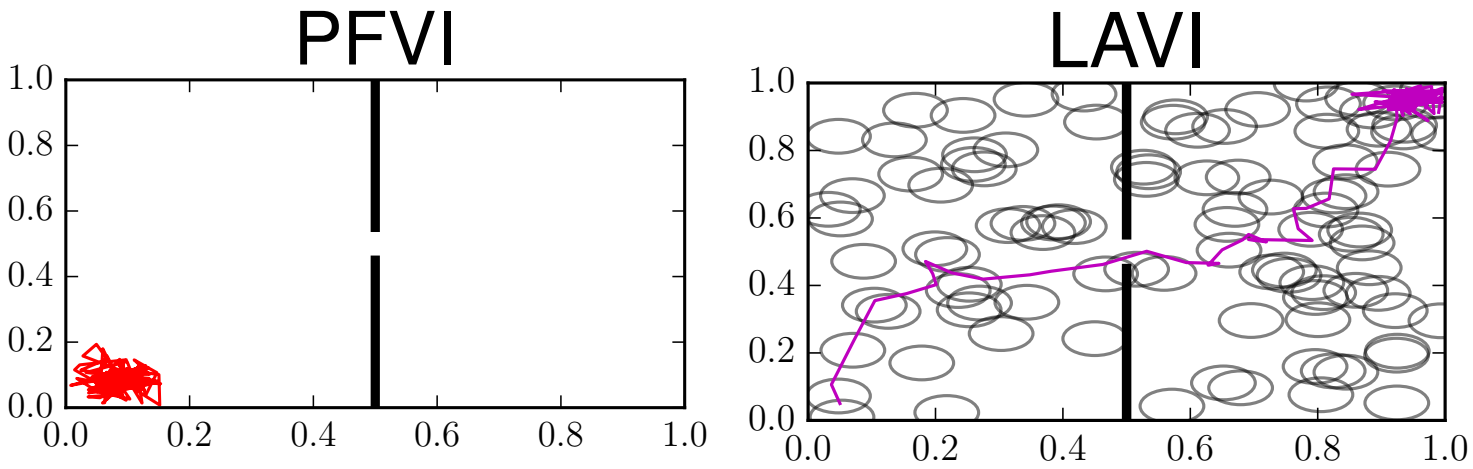
- Options can be given by a system designer (eg robotics)
- If subgoals / secondary reward structure is given, the option policy can be obtained, by solving a smaller planning or learning problem (cf. Precup, 2000)
  - Eg. acquiring certain objects in a game
  - Eg. Intrinsic motivation
- *What is a good set of subgoals / options?*
- This is a *representation discovery* problem
- Studied a lot over the last 20 years
- Bottleneck states and change point detection currently the most successful methods

Bottleneck States



- Perhaps the most explored idea in options construction
- A bottleneck allows "circulating" between many different states
- Lots of different approaches!
  - Frequency of states (McGovern et al, 2001, Stolle & Precup, 2002)
  - Graph partitioning / state graph analysis (Simsek et al, 2004, Menache et al, 2004, Bacon & Precup, 2013) / graph Laplacian (eg Klissarov and Machado, 2023)
  - Information-theoretic ideas (Peters et al., 2010)
- People seem quite good at generating these (cf. Botvinick, 2001, Solway et al, 2014)
- *Main drawback: expensive both in terms of sample size and computation*

Simpler Idea: Random Subgoals



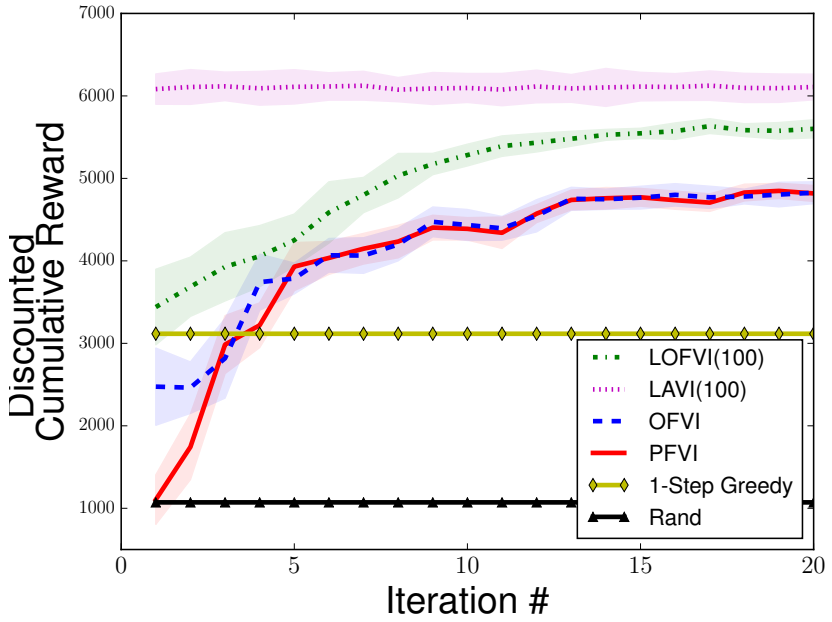
Cf. Mann, Mannor & Precup, 2015

Inventory management application

- Manage a warehouse that can stock 8 different commodities
- At most 500 items can be stored at any given time
- Demand is stochastic and depends on time of year
- Negative rewards are given for unfulfilled orders and for the cost of ordered items
- Hand-crafted options: order nothing until some threshold is crossed
- Primitive actions: specify amount of order for each item

Inventory management results

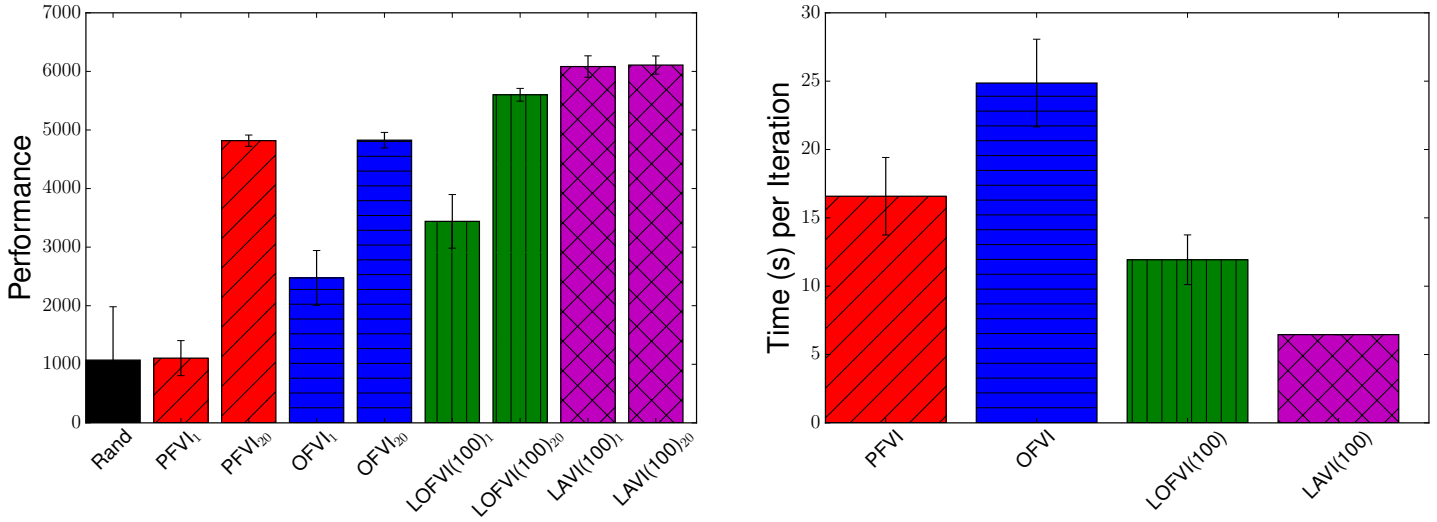
- Comparing a random policy and a 1-step greedy choice with using just primitives (PFVI) using primitives and hand-crafted options (OFVI), using "landmarks" (LOFVI) and using landmarks and only computing values for landmarks states (LAVI)



- *Randomly generated landmarks/subgoals perform much better*

Performance and time evaluation

- Performance of initial and final policy (left) and running time (right) averaged offer 20 independent runs

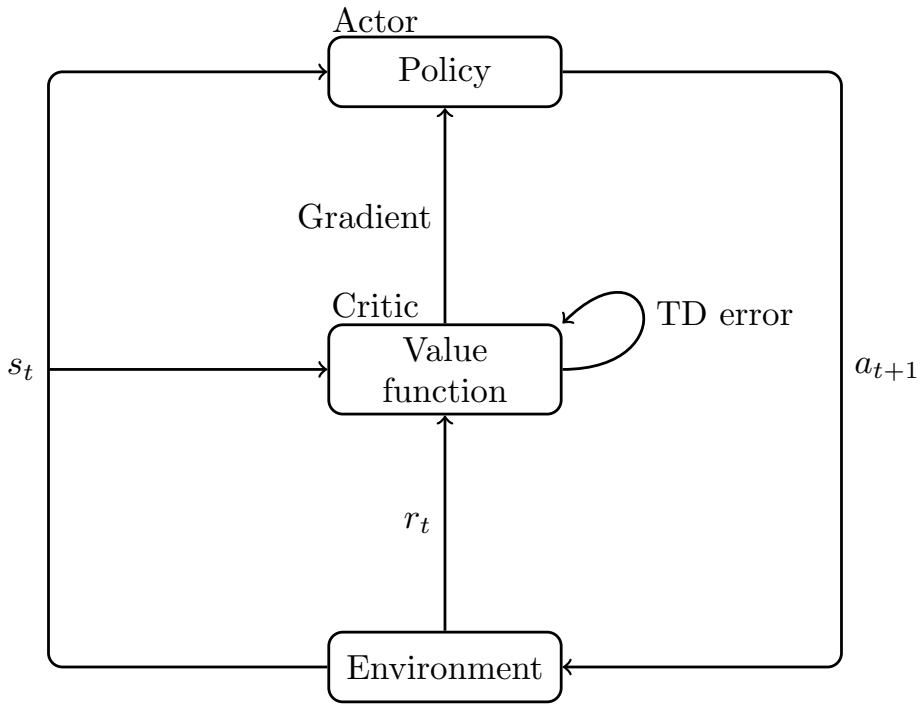


- Computing values only at landmark states yields a good policy almost immediately
- Handcrafted options are better than primitives in the beginning but slightly worse in the long run but *randomly generated landmarks are much better*

Option-Critic: Learn Options that Optimize Return

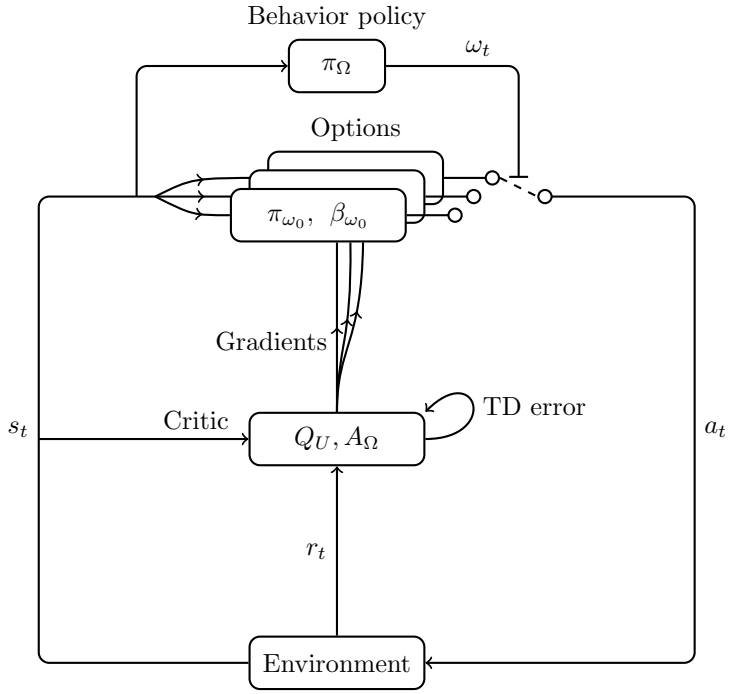
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or at least not cause slow-down...)

Actor-Critic Architecture



- Clear optimization objective: average or discounted return
- Continual learning
- Handles both discrete and continuous states and actions

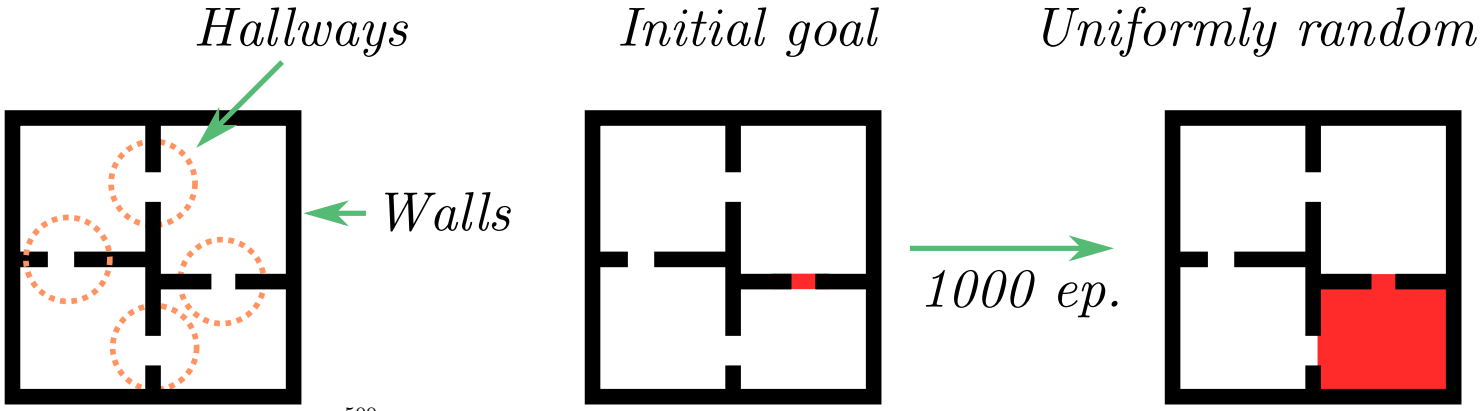
Option-Critic Architecture

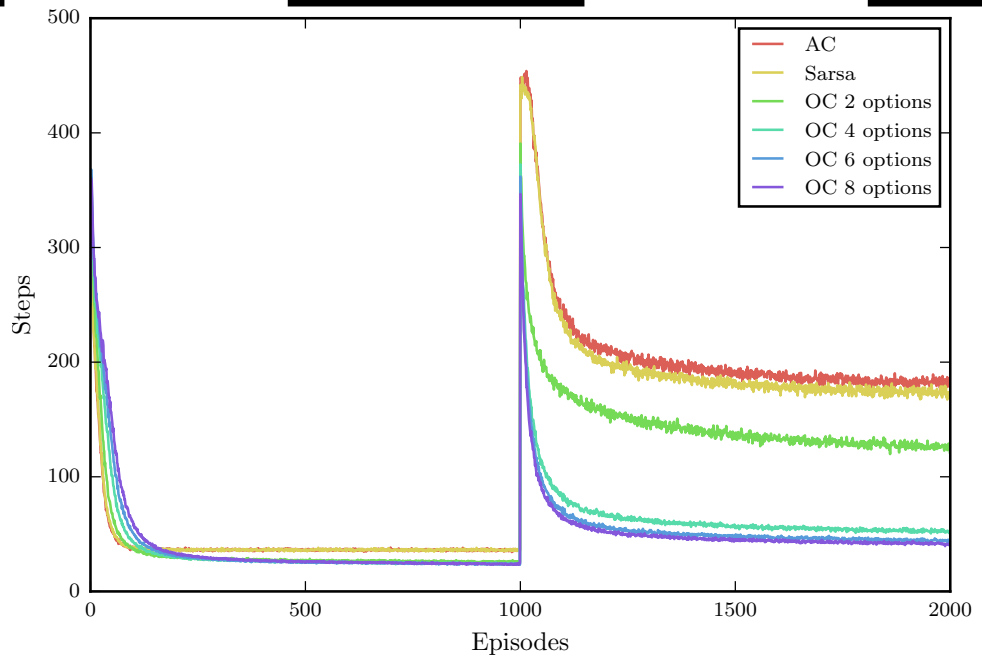


- Given a number of desired options, optimize internal policies and termination conditions using the cumulative reward signal

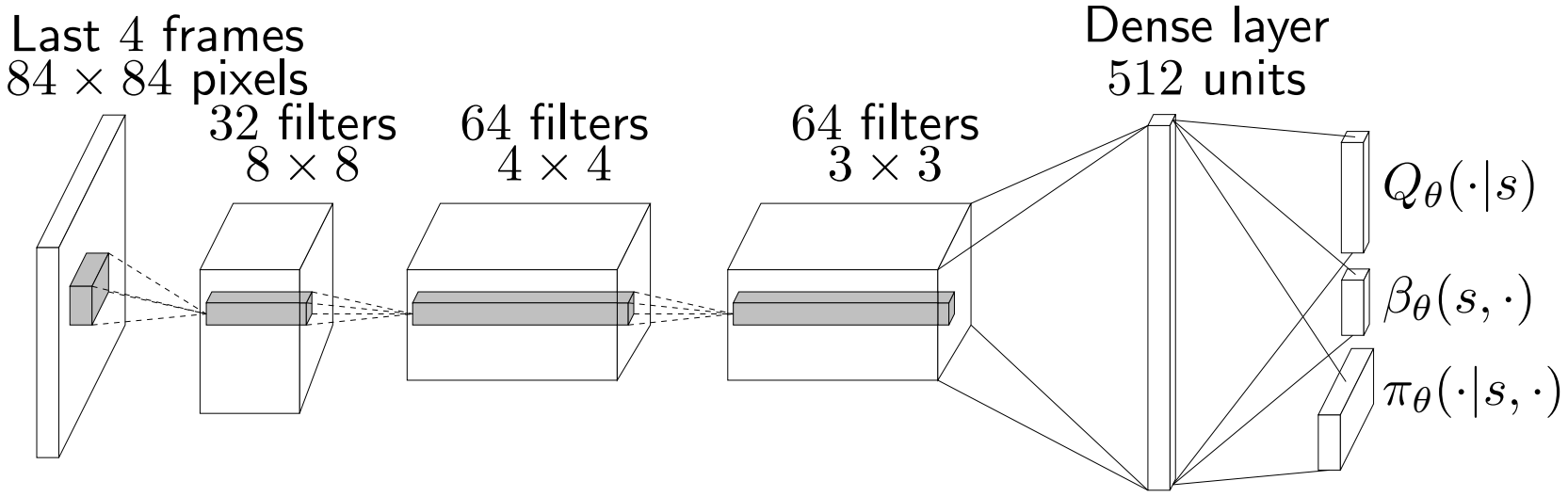
cf. Bacon et al, AAAI'2017

Results: Transfer in Rooms Domain



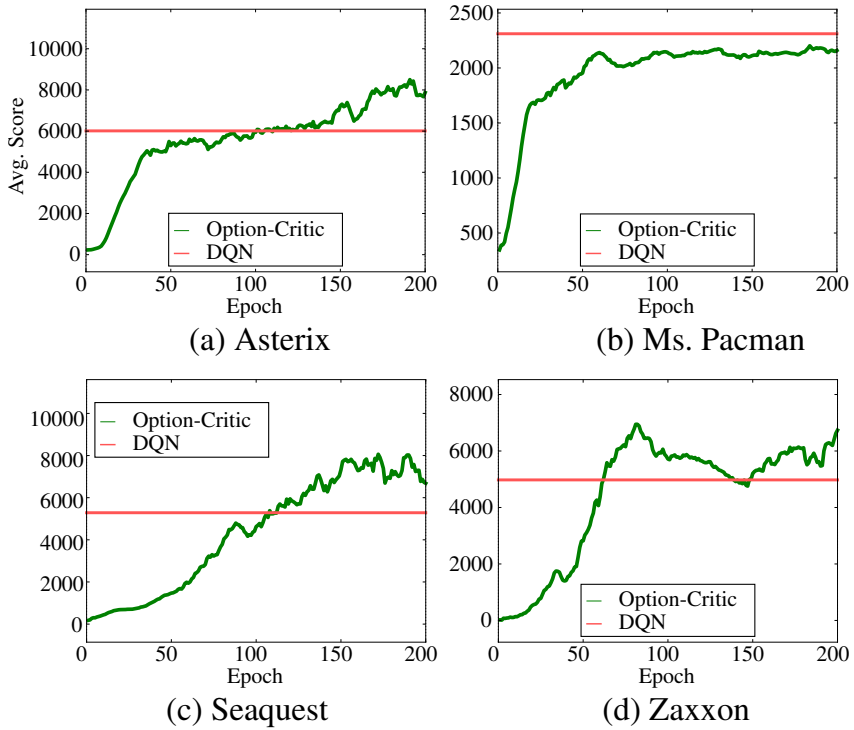


Option-Critic Architecture



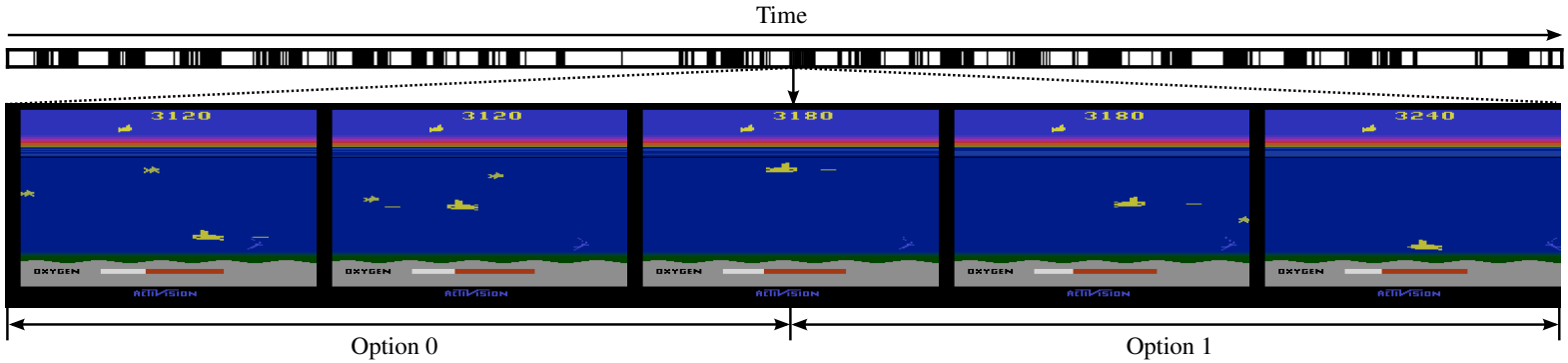
- Given a number of desired options, optimize internal policies and termination conditions using the reward signal
- DQN-style or advantage asynchronous option-critic (A2OC) (other choices possible)

Quantitative results in Atari games



- Performance matching or better than DQN *learning within a single task*
- Out of 8 games tested, option-critic does better that published results in 7, with A3C version superior to DQN - mainly due to exploration

Qualitative results in Atari games



- In Seaquest, separate options are learned to go up and down

Preserving Procedural Knowledge over Time

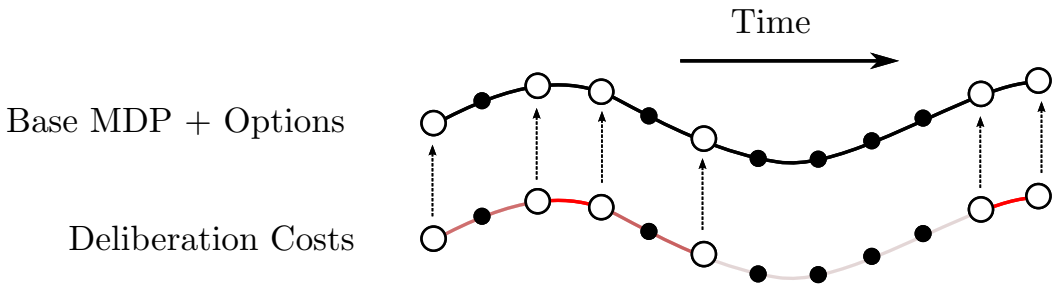
- Successful simultaneous learning of terminations and option policies
- But, as expected, *options shrink over time* unless additional regularization is imposed

  Cf. time-regularized options, Mann et al, (2014)
- Intuitively, using longer options increase the speed of learning and planning (but may lead to a worse result in call-and-return execution)
- Diverse options are useful for exploration in continual learning setting

Bounded Rationality as Regularization

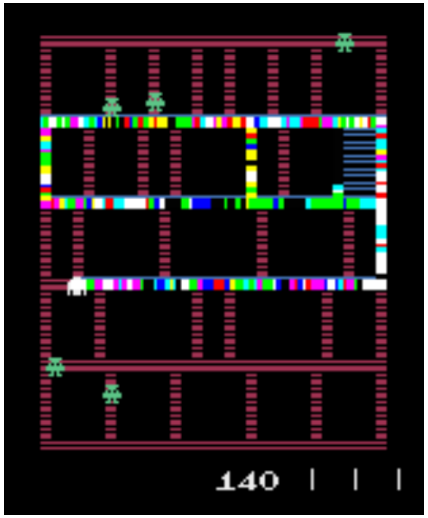
- Problem: optimizing return leads to option collapse (primitive actions are sufficient for optimal behaviour)
- Bounded rationality: reasoning about action choices is expensive (energy consumption and missed-opportunity cost)

  Eg Russell, 1995, Lieder & Griffiths, 2018
- Idea: switching options incurs an additional cost

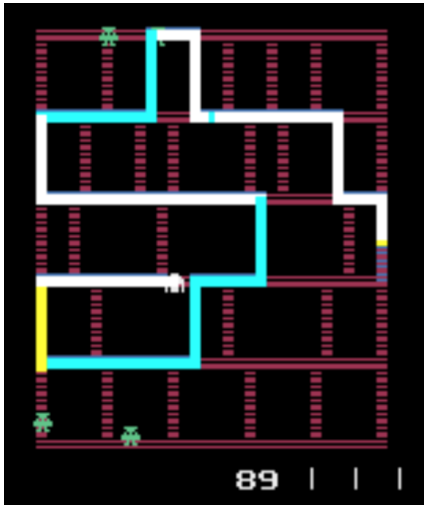


- Can be shown equivalent to requiring that *advantage exceeds a threshold* before switching

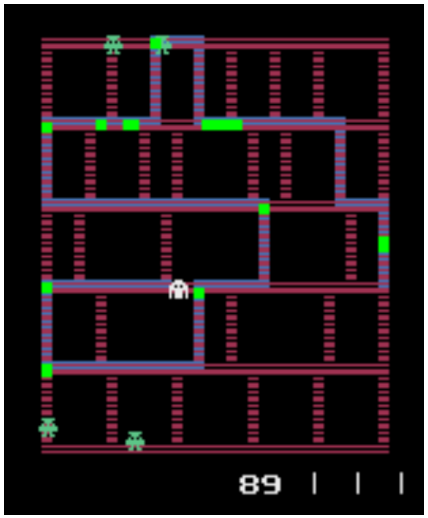
Illustration: Amidar



(a) Without a deliberation cost, options terminate instantly and are used in any scenario without specialization.



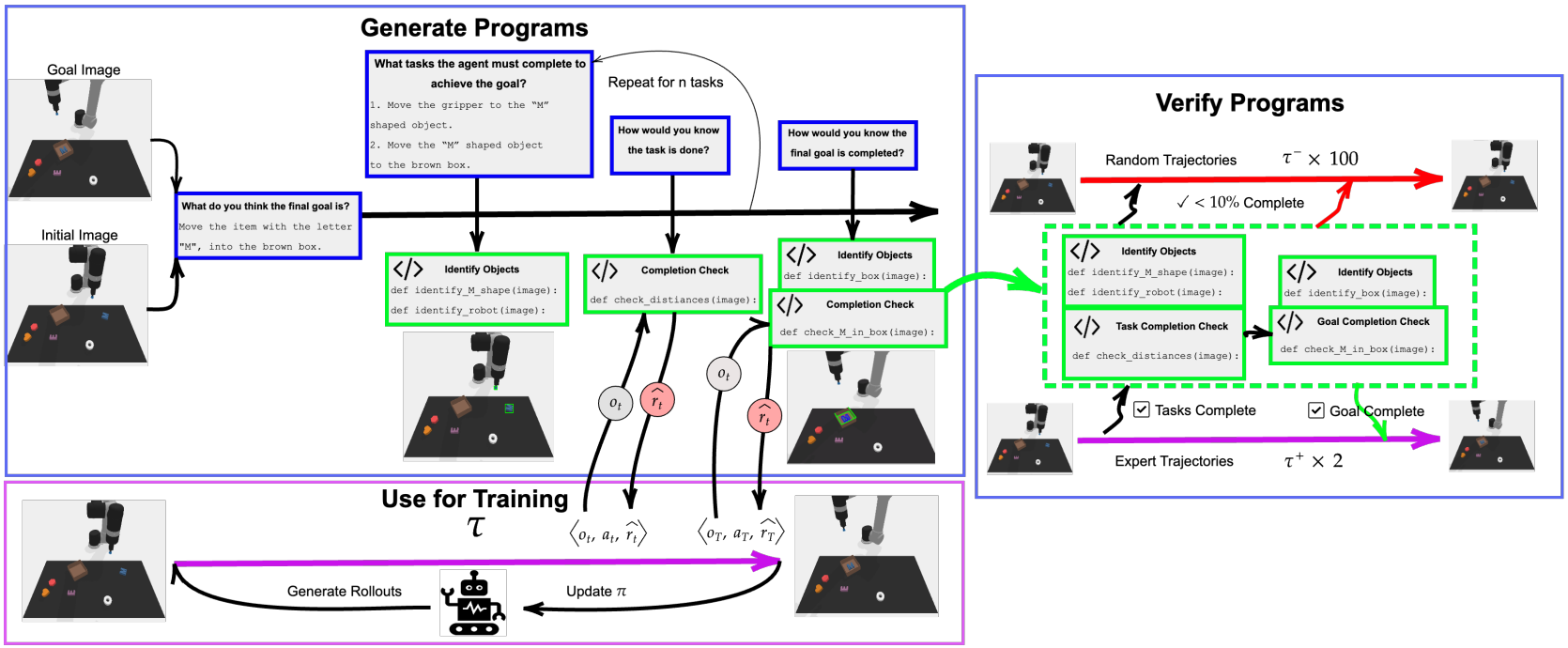
(b) Options are used for extended periods and in specific scenarios through a trajectory, when using a deliberation cost.



(c) Termination is sparse when using the deliberation cost. The agent terminates options at intersections requiring high level decisions.

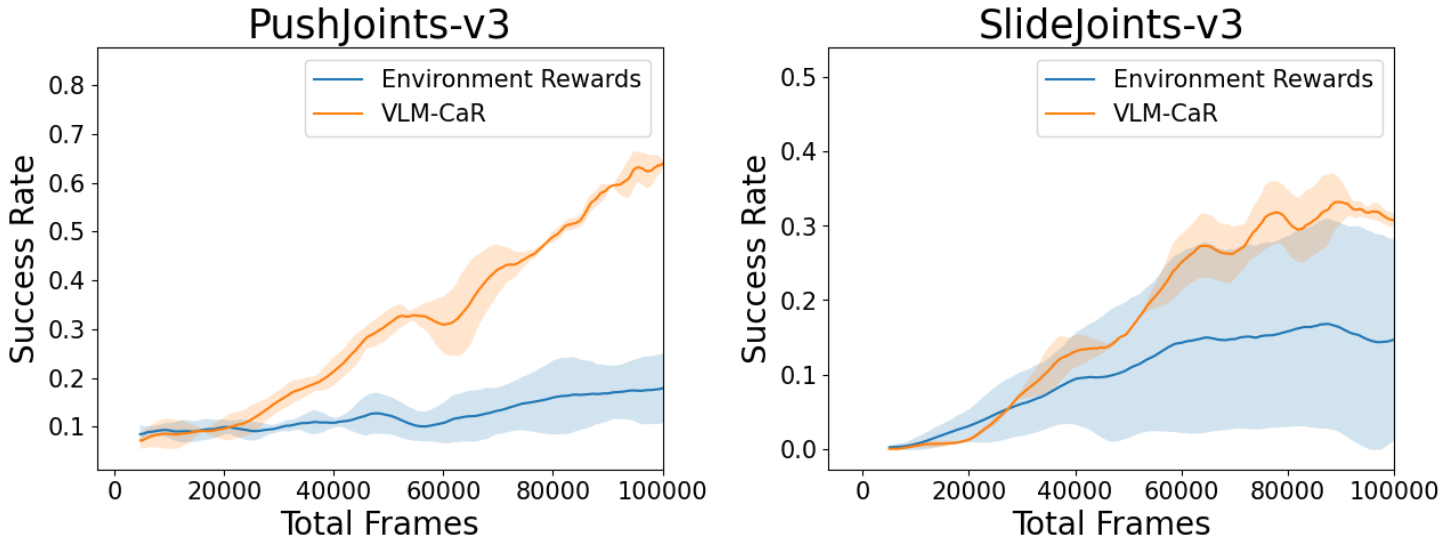
- Deliberation costs prevent options from becoming too short
- Terminations are intuitive

Leveraging Large Models to Construct Options



Cf. Venuto et al, ICML'2024

Illustration: Generated Code Options Improve RL!



Cf. Venuto et al, ICML'2024, Klissarov et al, ICLR'2025

Predictive knowledge: Value Function

- Given a policy π, a discount factor γ and a reward function r, the value function of the policy is given by:

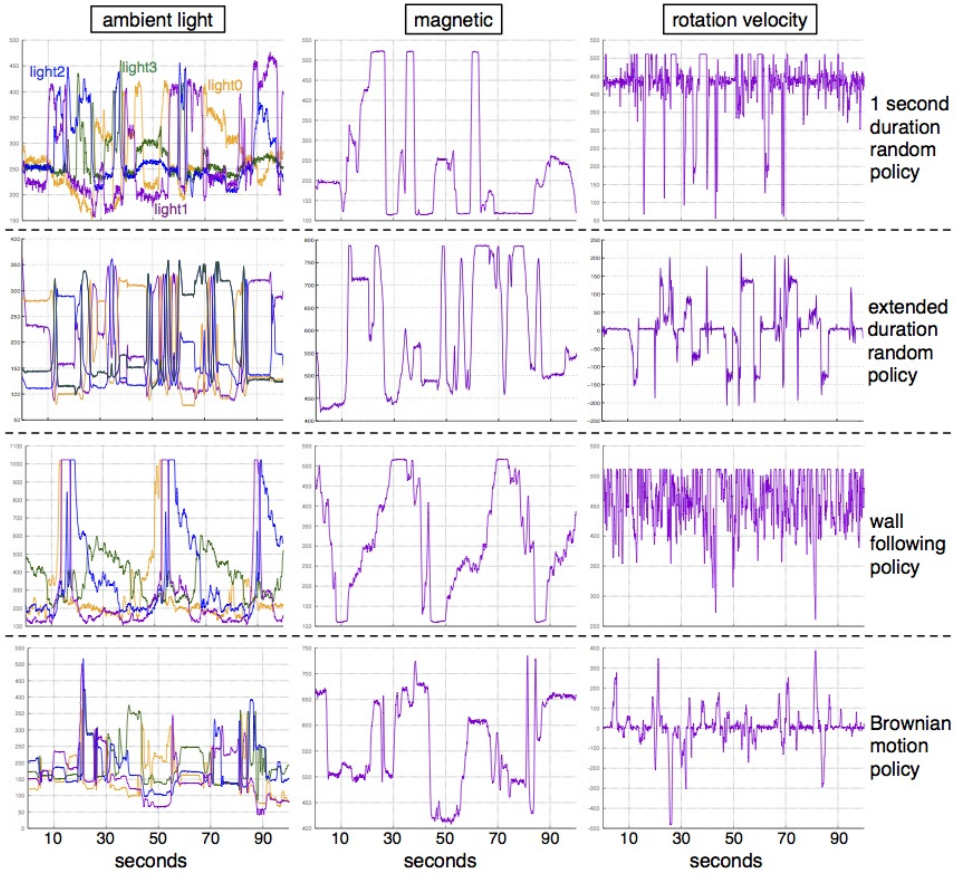
$$\begin{aligned} v_\pi(s) &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k)\gamma^{k-t} | S_t = s, A_{t:\infty} \sim \pi] \\ &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k) \prod_{i=t+1}^{k} \gamma | S_t = s, A_{t:\infty} \sim \pi] \end{aligned}$$

- r is the *signal of interest* for the prediction
- γ defines the time scale over which we want to make the prediction (in a very crude way)
- Optimal value function: given a discount factor γ and a reward function r, compute v_{π^*} and π^*, the optimal policy wrt γ, r

Focusing on value function

- Definition allows us to leverage great tools: *bootstrapping* (as in dynamic programming) and *sampling*
- We have good ideas for how to learn value functions from data using temporal-difference methods, off-policy learning...
- Usual objection: this is restricted to one reward function and usually a fixed time scale (discount)
- An agent may need to make predictions about many different things and at many different time scales

There are many things to learn! (Adam White's thesis)



Sensory stream of Critterbot robot about different sensors for different policies

Can we learn about all these signals in parallel from one stream of data?

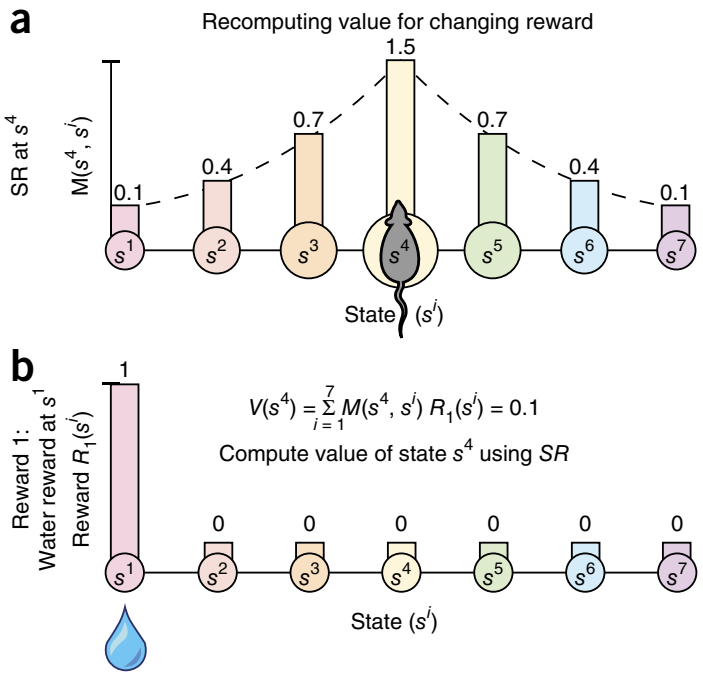
Temporally Abstract Predictions: General Value Functions (GVFs)

- 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)

Special case: Successor States



- Successor states (Dayan, 1992): expected occupancy of future states, for a given policy
- Allow the value function for *any reward* to be quickly computed
- Evidence linking to the hippocampus (Stachenfeld et al, 2018)

Special case: Successor Features

- *Successor features* (Barreto et al, 2017, 2018) are a natural extension of successor states
- If states are defined by a feature vector $\phi(s)$, successor features are GVFs where 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

Cpecial case: Option models

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

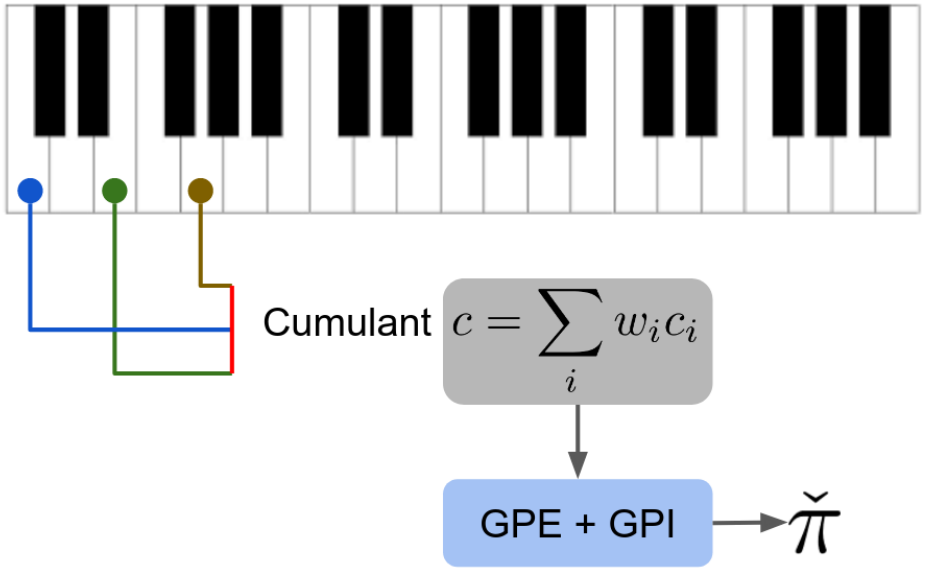
$$r_\omega(s) = \mathbb{E}_\omega[r(S_t, A_t) + \gamma(1 - \beta_\omega(S_{t+1}))r_\omega(S_{t+1})|S_t = s]$$

- 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?*

GVFs for synthesizing new behaviors



Option-keyboard - Barreto et al, 2019, based on ideas of Rich Sutton

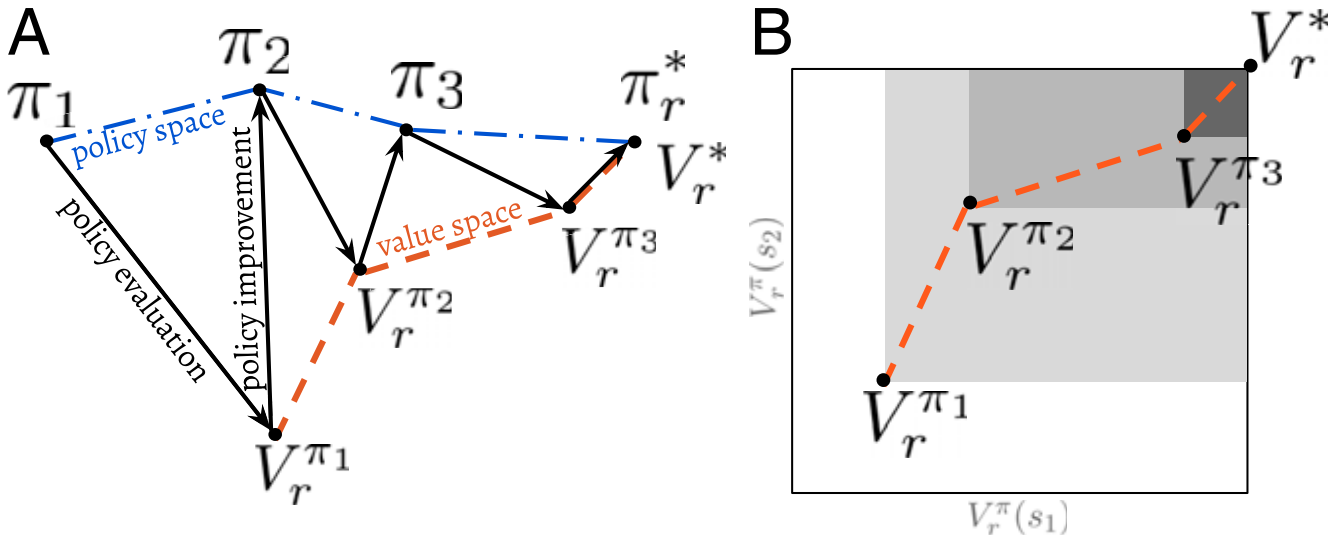
Policy Evaluation and Policy Improvement

- Consider a Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ and a policy $\pi : \mathcal{S} \rightarrow Dist(\mathcal{A})$
- 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* $\pi \in \Pi$

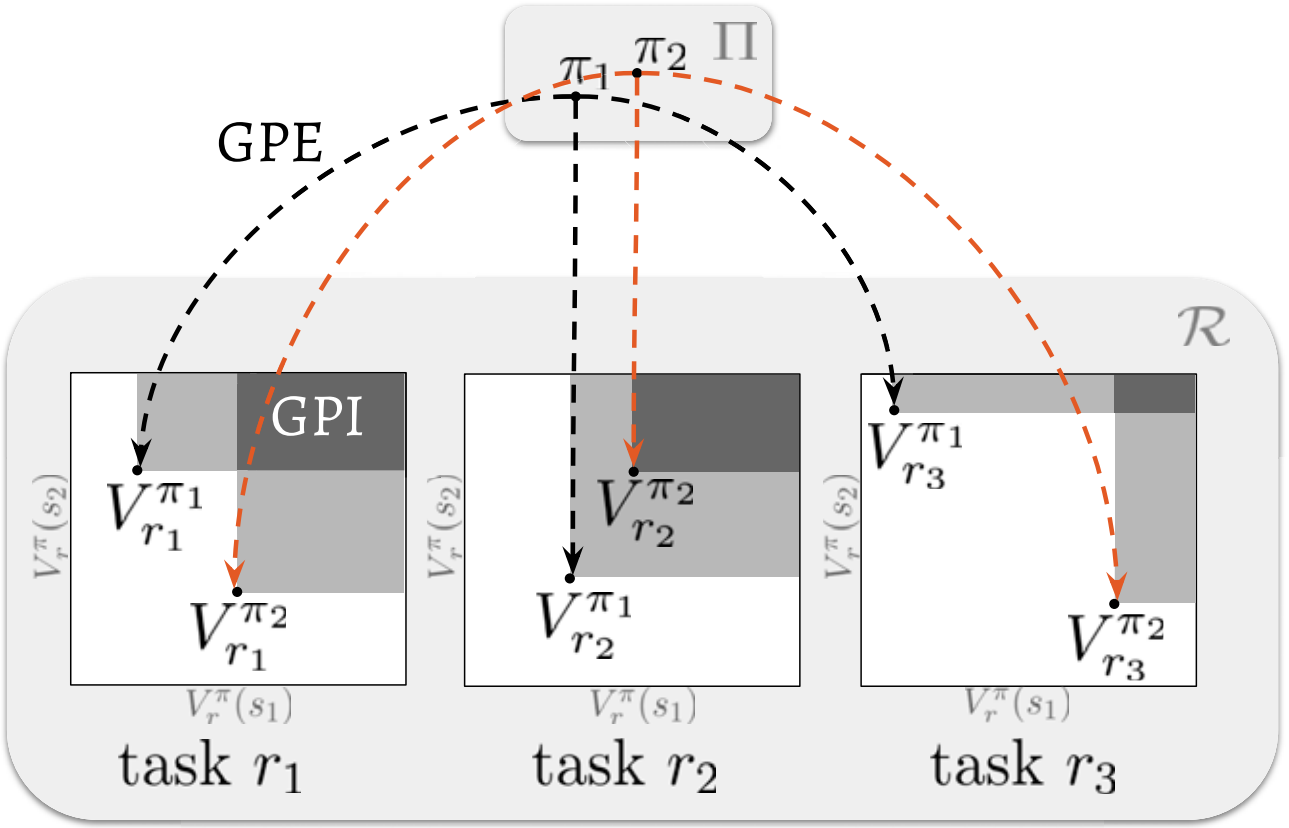
Generalized Policy Updates

- *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^π_r, 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:

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

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

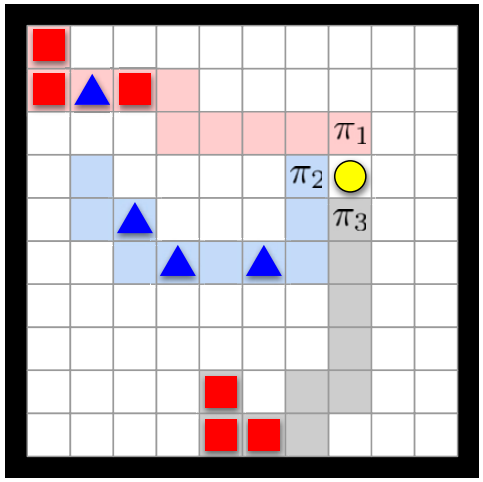
Fast Generalized Policy Improvement

- Compute the improved policy as:

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

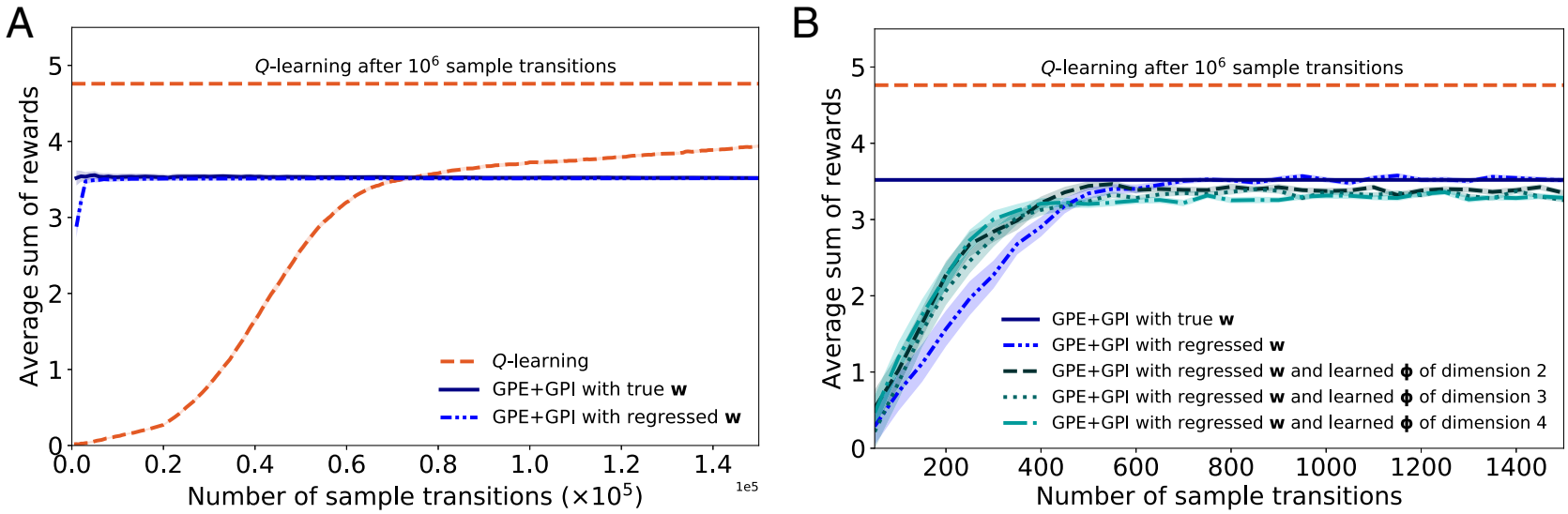
- Note that π' *could choose actions that are not chosen by any of the* π
- The process takes only *one iteration*, after which no further change to the policy π' would happen
- In contrast with iterative policy improvement...

Illustration



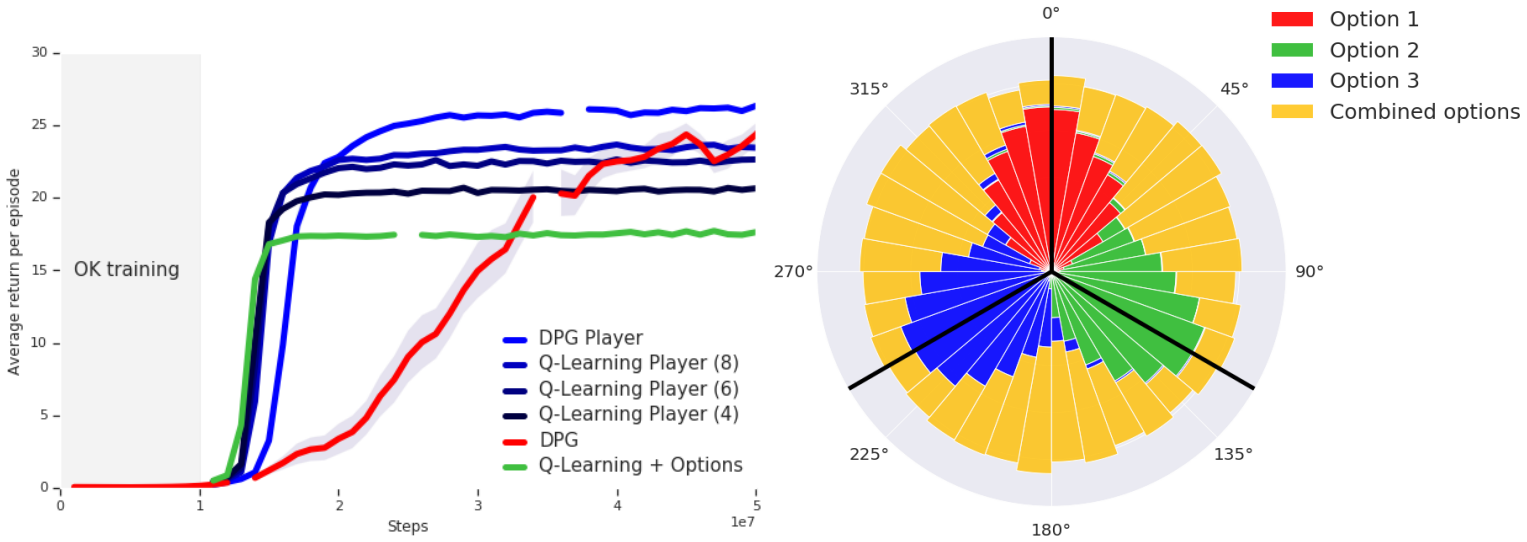
- The three policies correspond to three weight vectors: like red ($\mathbf{w}_1 = [1, 0]^T$), like blue ($\mathbf{w}_2 = [0, 1]^T$) and like red not blue ($\mathbf{w}_3 = [1, -1]^T$)
- *Note that $\mathbf{w}$ can be viewed as a preference function over features!*
- We can pre-train the policies that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

Option-Keyboard for Moving Target Arena



General way to synthesize quickly new behavior for combinations of reward functions!
How to efficiently compute many GVFs/successor features form one stream of experience?