Building Knowledge with Reinforcement Learning

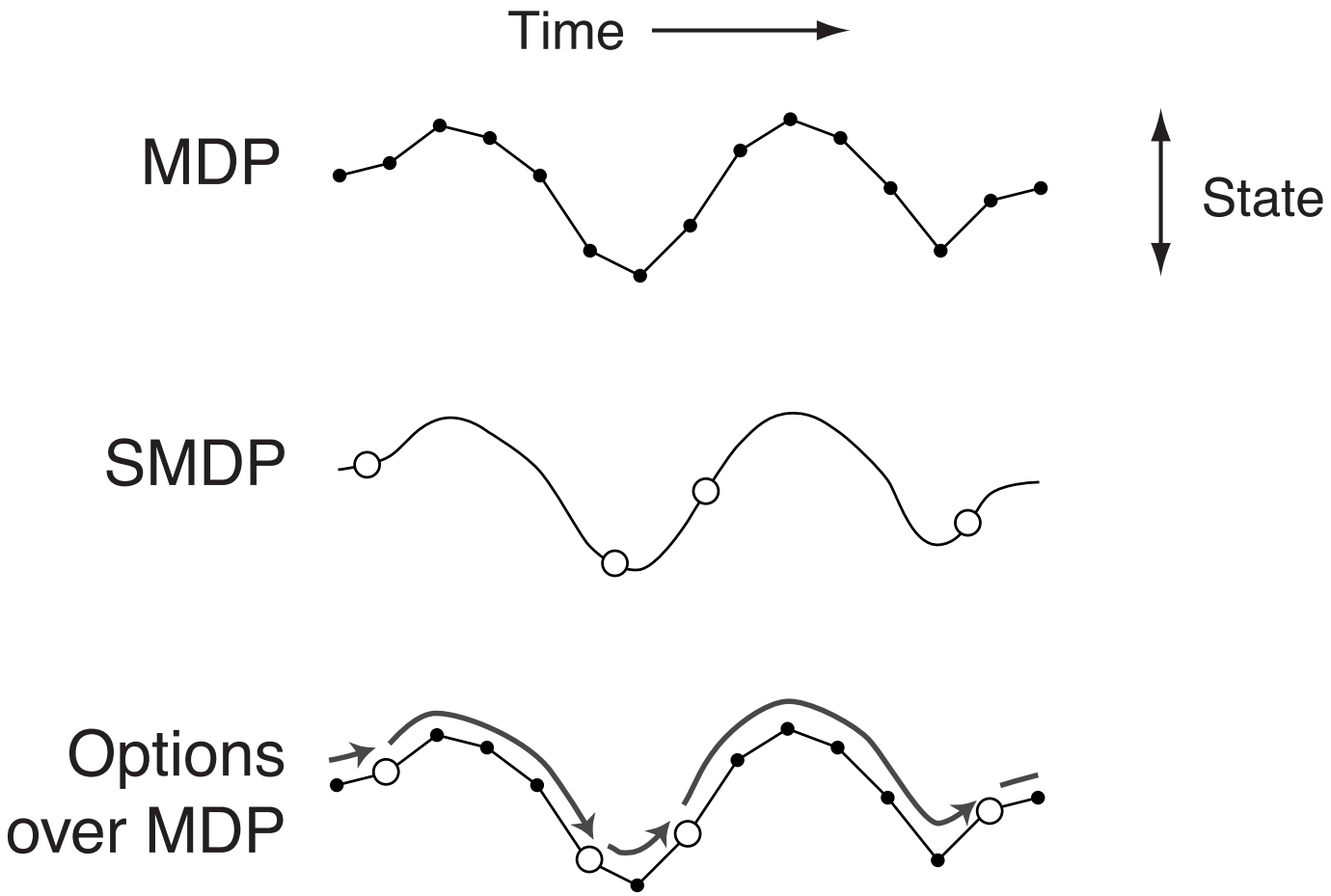
- Focusing on two types of knowledge:
  - *Procedural knowledge*: skills, goal-driven behavior
  - *Predictive, empirical knowledge*: Analogous to the laws of physics, predicting effects of actions
- The knowledge must be:
  - *Expressive*: able to represent many things, including abstractions like objects, space, people, and extended actions
  - *Learnable*: from data without labels or supervision (for scalability)
  - *Composable*: suitable for supporting planning / reasoning by assembling existing pieces

Procedural Knowledge: 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

Decision-Making with Options



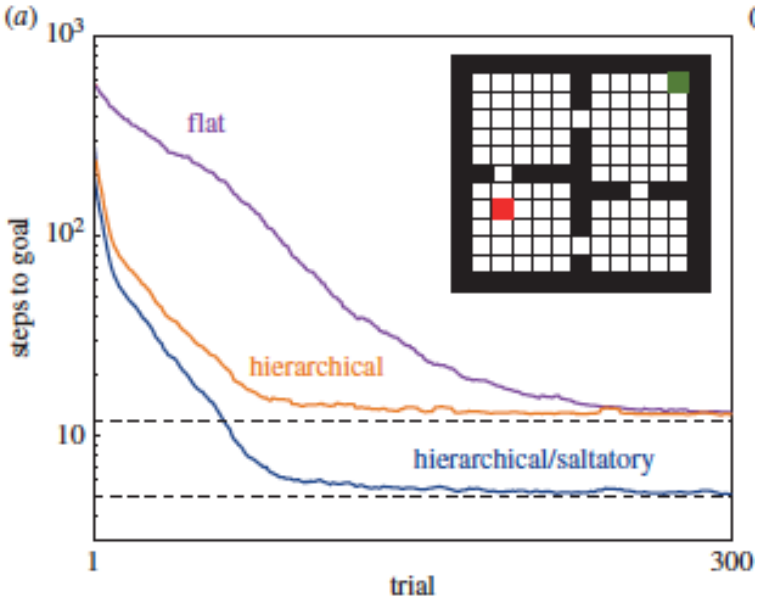
Learning and planning algorithms are the same at all levels of abstraction!

Options as Behavioral Programs

- *Call-and-return execution*
  - When called, option ω is pushed onto the execution stack
  - During the option execution, the program looks at certain *variables (aka state)* and executes an *instruction (aka action)* until a termination condition is reached
  - The option can keep track of additional *local variables*, eg counting number of steps, saturation in certain features (e.g. Comanici, 2010)
  - *Options can invoke other options*
- *Interruption*
  - At each step, one can check if a better alternative has become available
  - If so, the option currently executing is *interrupted* (special form of concurrency)

Option Models Provide Semantics

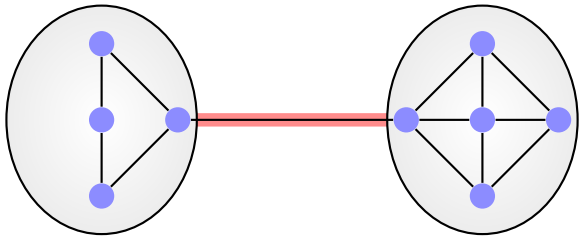
- Models of actions consist of immediate reward and transition probability to next state
- Models of options consist of reward until termination and (discounted) transition to termination state
- Models are *predictions about the future* and provide more *benefits beyond hierarchical behavior* (cf Botvinick & Weinsteein, 2014)



How Should Options Be Created?

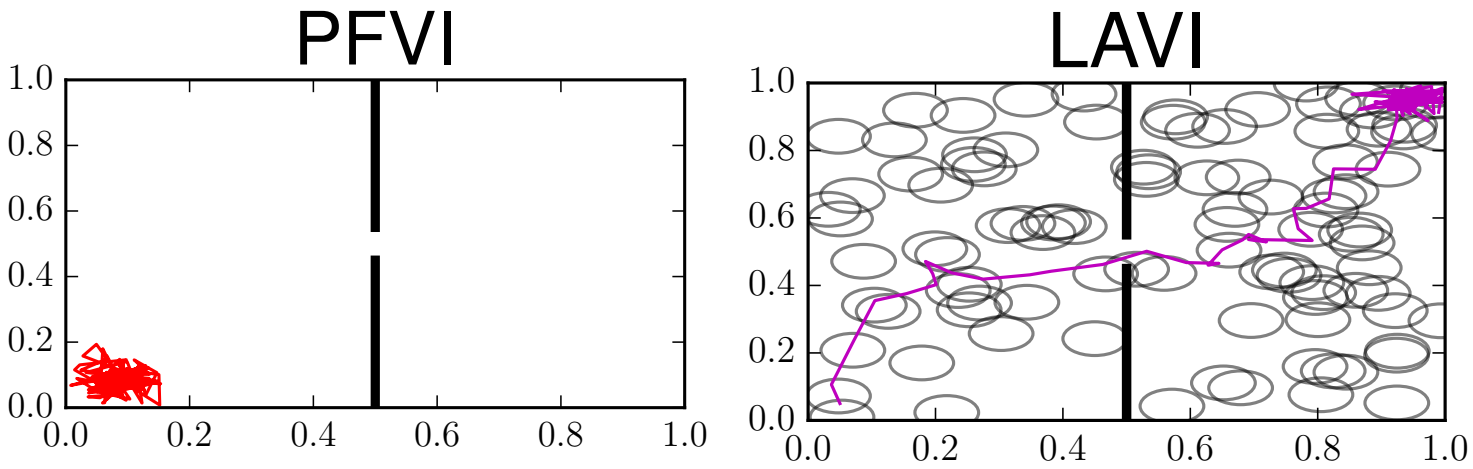
- Options can be given by a system designer (eg robotics)
- If subgoals / secondary reward structure is given, the option policy can be obtained, by solving a smaller planning or learning problem (cf. Precup, 2000)
  - Eg. acquiring certain objects in a game
  - Eg. Intrinsic motivation
- *What is a good set of subgoals / options?*
- This is a *representation discovery* problem
- Studied a lot over the last 15 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 differenet 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)
  - 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*

Random Subgoals Also Help

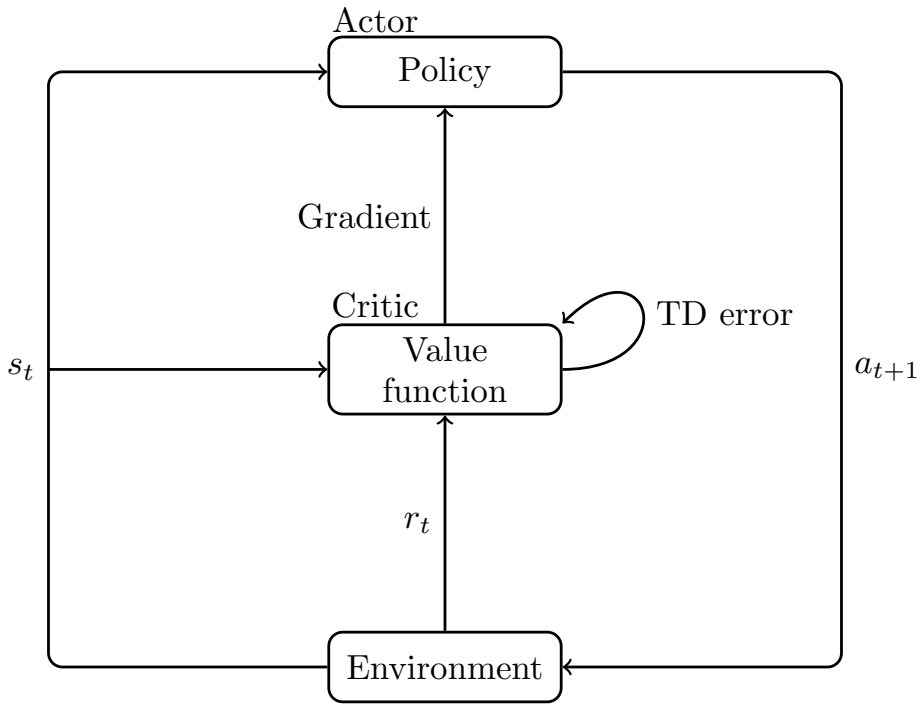


Cf. Mann, Mannor & Precup, 2015

Option-Critic: Learn Options that Optimize Return

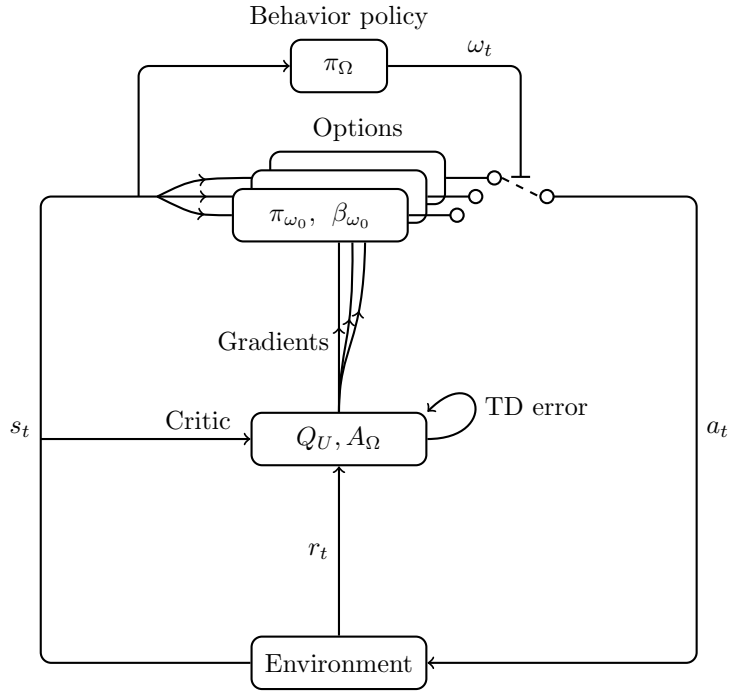
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or 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

Some details

- The action-value over options can be expressed as

$$Q_\Omega(s,\omega) = \sum_a \pi_\omega(a|s) Q_U(s,\omega,a)$$

  where

$$Q_U(s,\omega,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) U(\omega,s')$$

- The last quantity is the utility from s' onwards, *given that we arrive in s' using ω*

$$U(\omega,s') = (1-\beta_\omega(s'))Q_\Omega(s',\omega) + \beta_\omega(s')V_\Omega(s')$$

- We parameterize the internal policies by θ, as $\pi_{\omega,\theta}$, and the termination conditions by ν, as $\beta_{\omega,\nu}$
- Note that θ and ν can be shared over the options!

Main result: Gradient updates

- Suppose we want to optimize the expected return: $\mathbb{E}_{(s,\omega)\sim\mu} Q_\Omega(s,\omega)$
- The *gradient wrt the internal policy parameters* θ is given by:

$$\mathbb{E}_{(s,\omega)\sim\mu}\mathbb{E}_{a\sim\pi_{\omega,\theta}(\cdot|s)}\left\{\frac{\partial \log \pi_{\omega,\theta}(a|s)}{\partial\theta} Q_U(s,\omega,a)\right\}$$

  This has the usual interpretation: *take better primitives more often* inside the option

- The *gradient wrt the termination parameters* ν is given by:

$$\mathbb{E}_{(s,\omega)\sim\mu}\mathbb{E}_{a\sim\pi_{\omega,\theta}(\cdot|s)}\mathbb{E}_{s'\sim P(\cdot|s,a)}\left\{-\frac{\partial \beta_{\omega,\nu}(s')}{\partial\nu} A_\Omega(s',\omega)\right\}$$

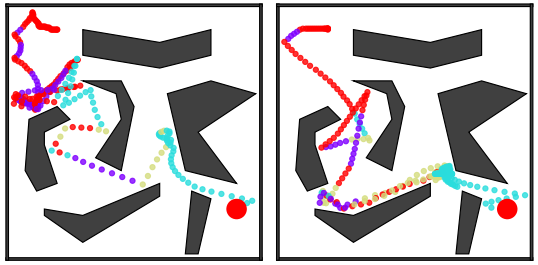
  where A_Ω is the advantage function

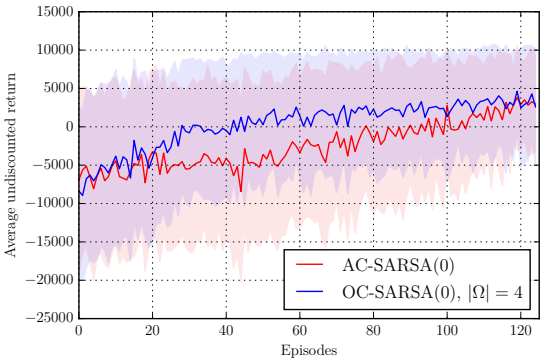
  This means that we want to *lengthen options that have a large advantage*

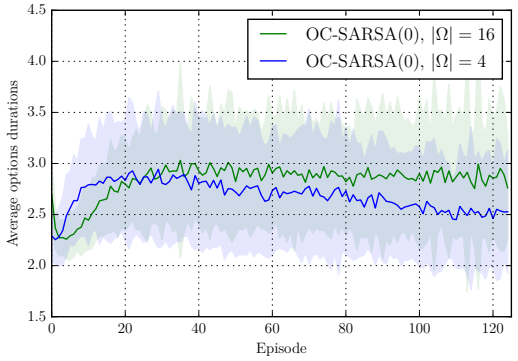
Experimental setup

- Linear function approximation for Q_U, Q_Ω, weights initialized to 0
- Q_Ω learned simultaneously with Sarsa
- Internal policies of options parameterized by softmax
- Termination conditions for options parameterized with a logistic function

Experiments: Pinball

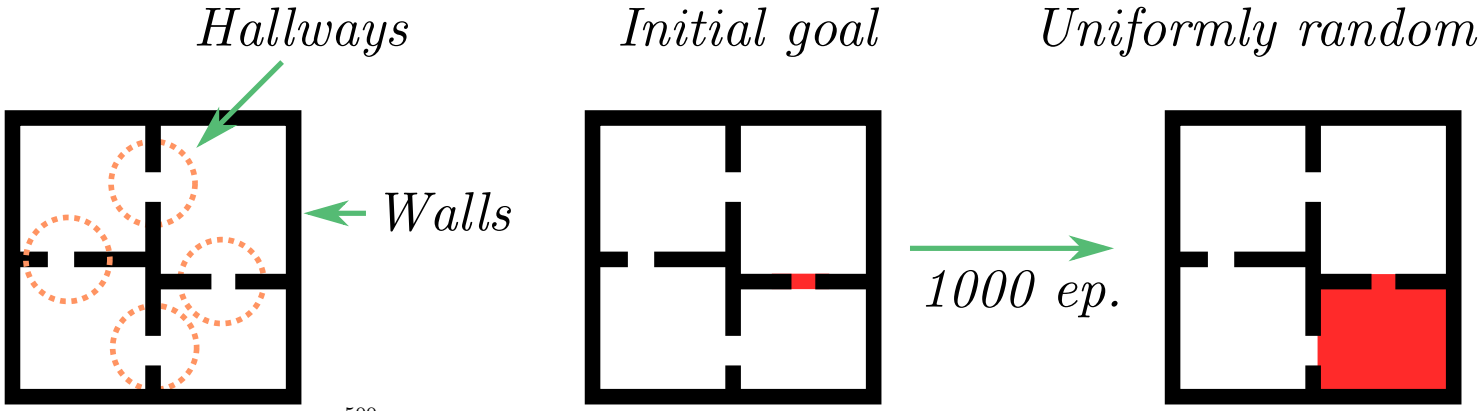


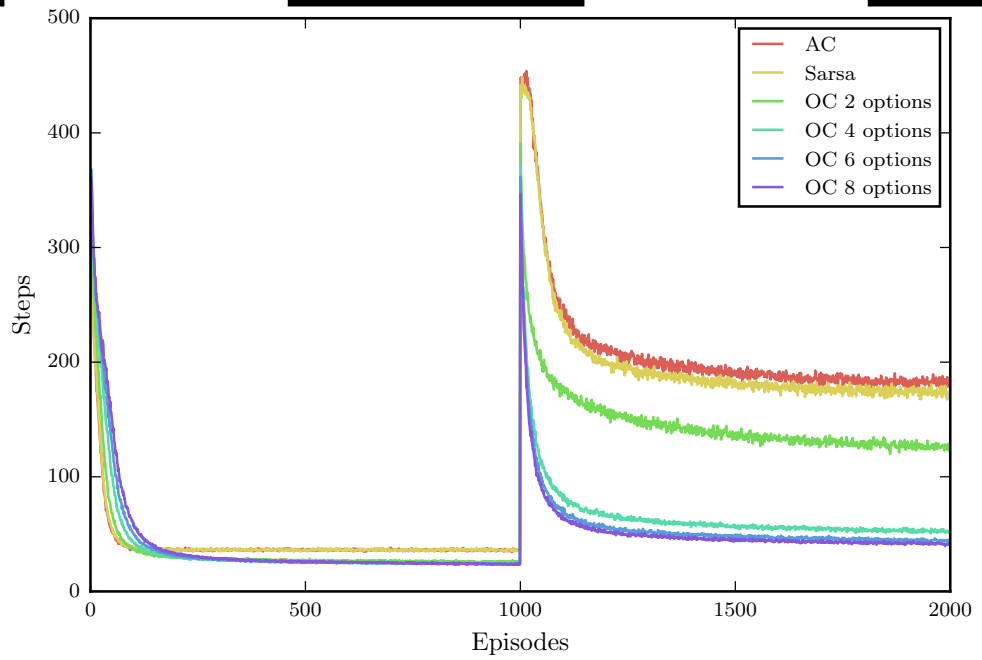




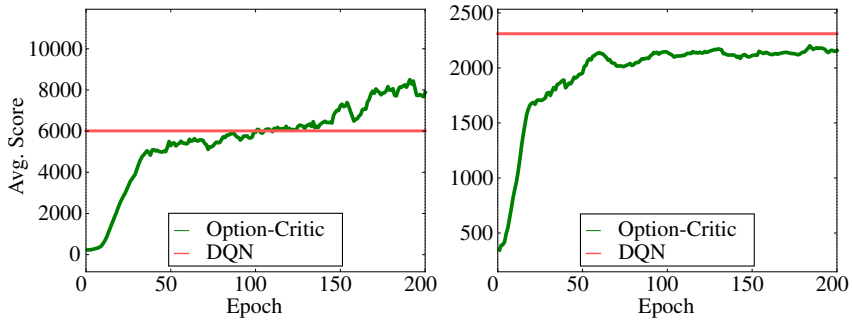
- Option parameters not shared in this case
- Options specialize in different areas (e.g. around the goal)
- In the long run, options get shorter
- This is expected: optimal policy is made of primitive actions

Results: Transfer in Rooms Domain



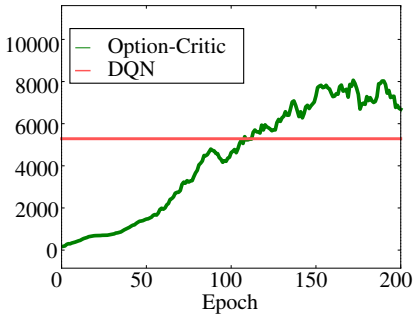


Quantitative and qualitative results in Atari games

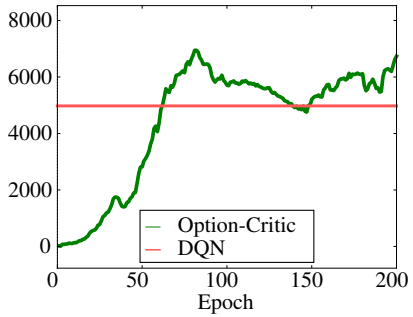


(a) Asterix

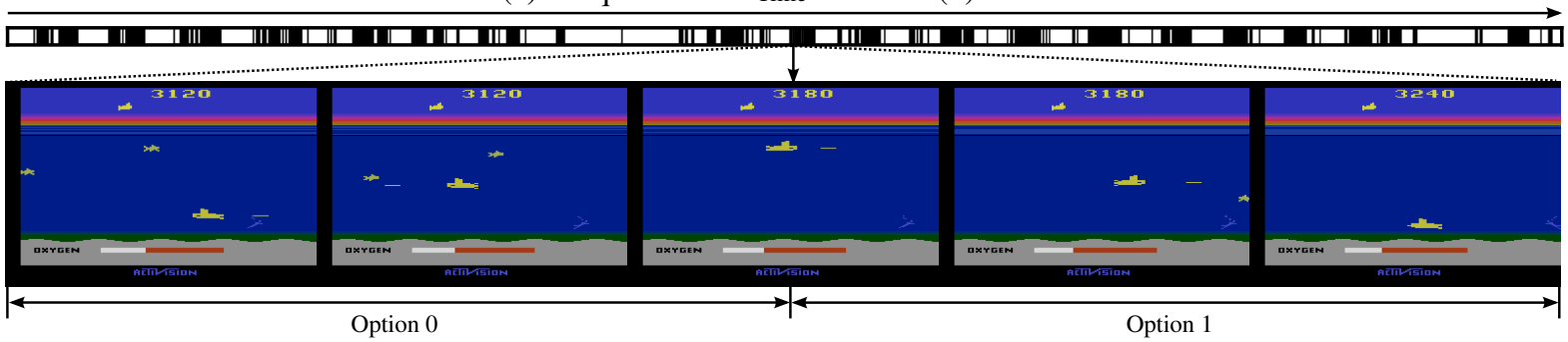
(b) Ms. Pacman



(c) Seaquest



(d) Zaxxon



Preserving Procedural Knowledge over Time

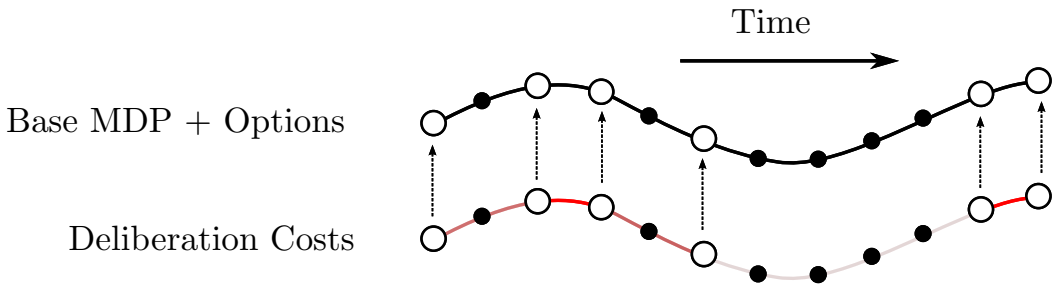
- Successful simultaneous learning of terminations and option policies
- But, as expected, *options shrink over time* unless additional regularization is imposed

  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

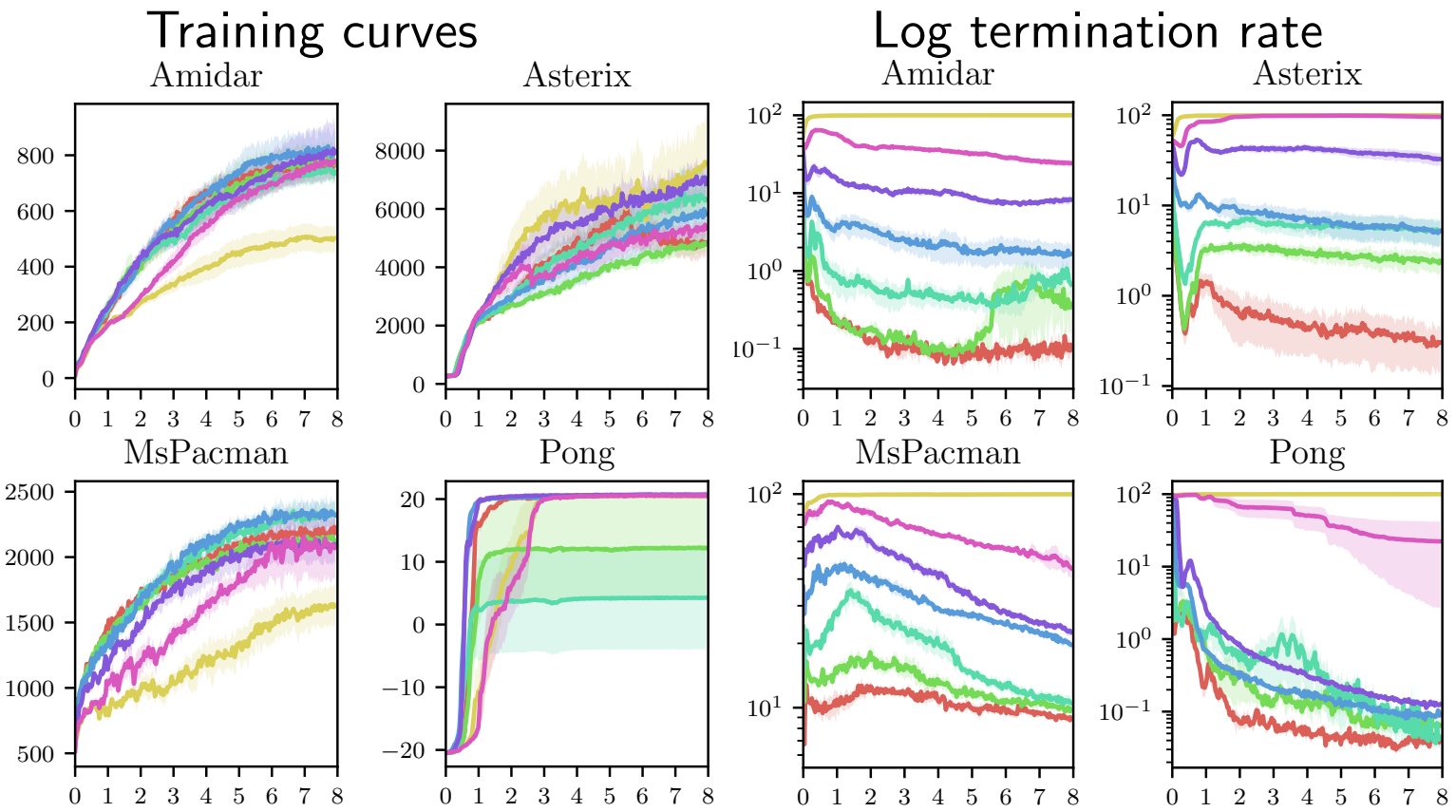
- Problem: optimizing return leads to option collapse (primitive actions are sufficient for optimal behaviour)
- Bounded rationality: reasoning about action choices is expensive (energy consumption and missed-opportunity cost)

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



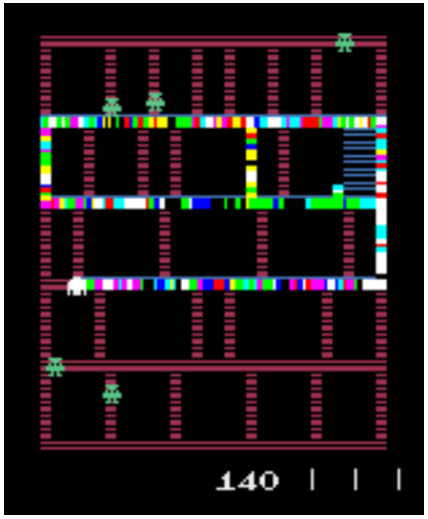
- Can be shown equivalent to requiring that *advantage exceeds a threshold* before switching

Effect of Deliberation Cost Regularization

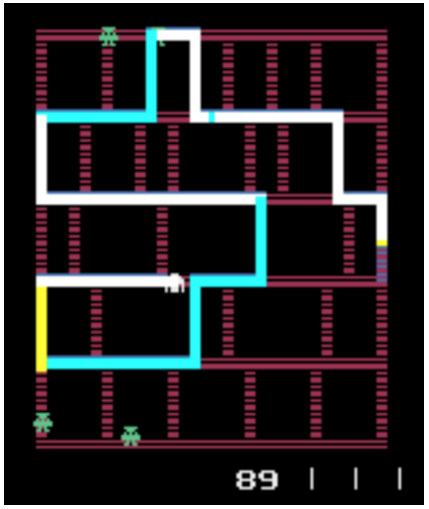


Yellow: no regularization; red: most regularization

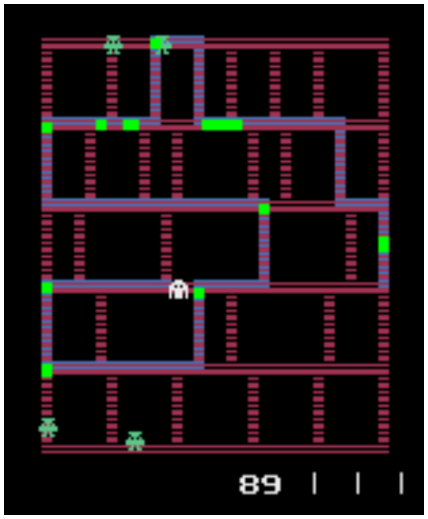
Illustration: Amidar



(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

Should All Option Components Optimize the Same Thing?

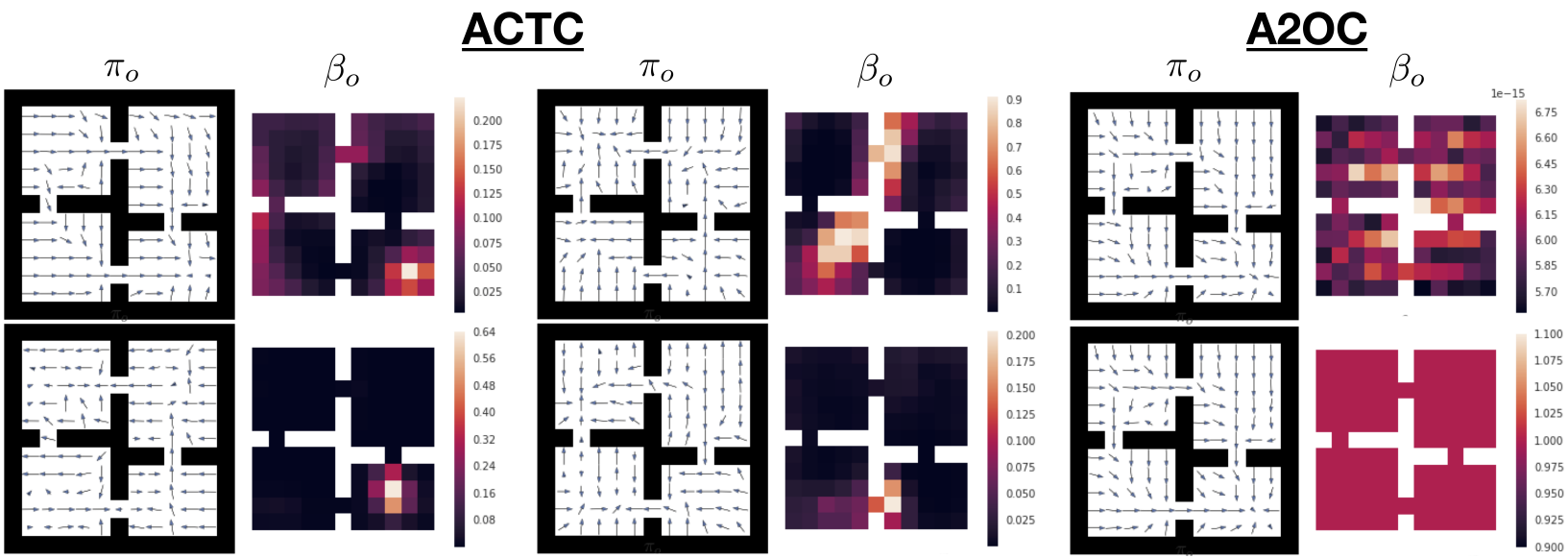
- Deliberation cost can be viewed as associated specifically with termination
- Rewards could be optimized mainly by the internal policy of the option
- Can we generalize this idea to other optimization criteria?

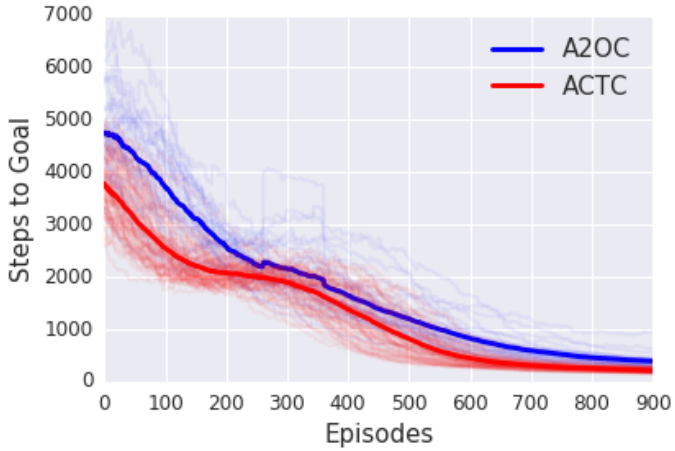
Termination-Critic

- *Optimize the termination condition independently of the policy inside the option*
- Option termination should focus on *predictability* ie finding "funnelling states"
- Interesting side effect: if each option ended at a funelling state, expectation and distribution model would be almost identical and the option would be almost deterministic
- Implementation: minimize the entropy of the option transition model P_ω

cf. Harutyunyan et al, AISTATS'2019

Illustration: Rooms environment





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:

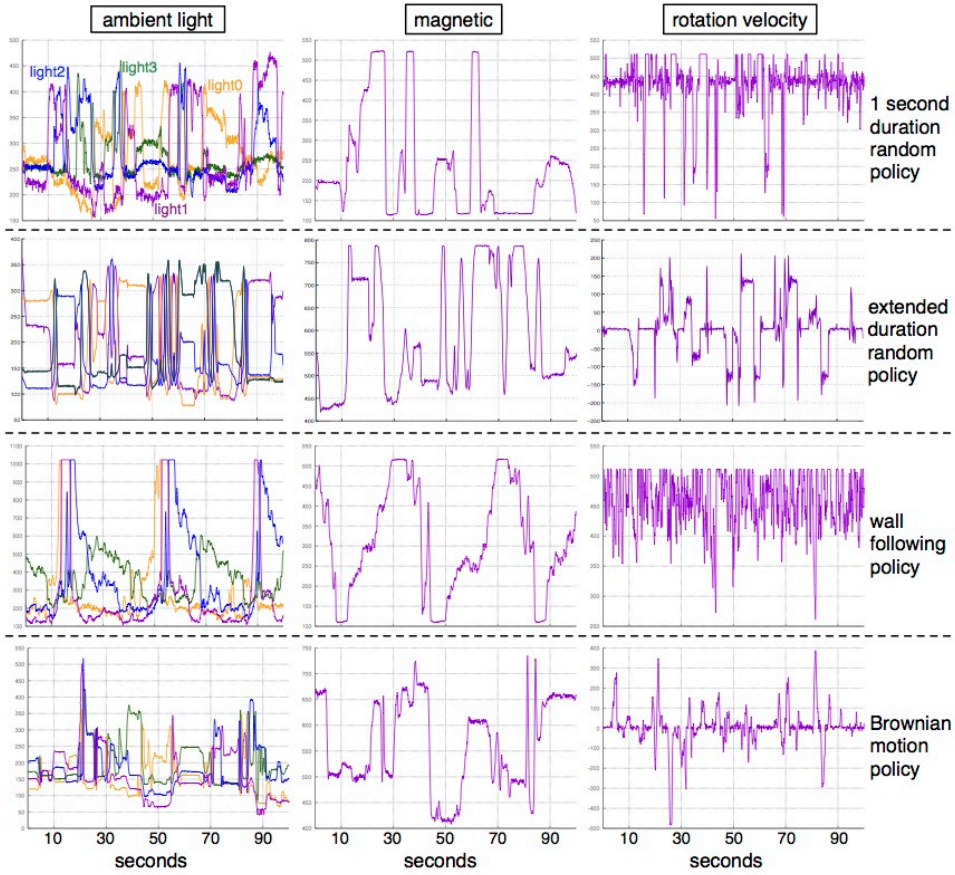
$$\begin{aligned} v_\pi(s) &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k)\gamma^{k-t} | S_t = s, A_{t:\infty} \sim \pi] \\ &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k) \prod_{i=t+1}^{k} \gamma | S_t = s, A_{t:\infty} \sim \pi] \end{aligned}$$

- 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?

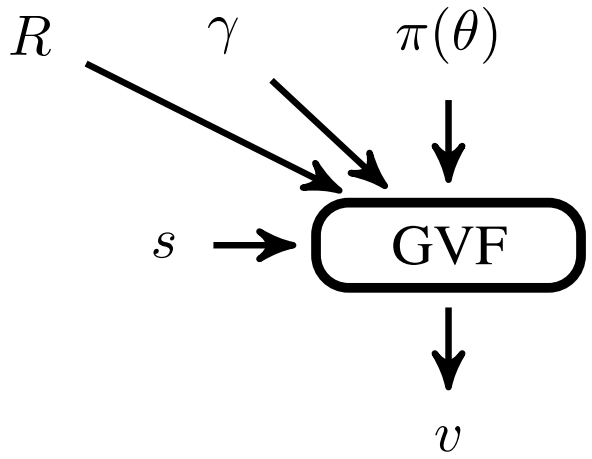
Generalized Value Functions (GVFs)

- Given a cumulant function c, state-dependent continuation function γ and policy π, the Generalized 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

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

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?*

GVF formulation of policy gradient

- 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.
- Let $v_{c,\gamma,\pi_\theta}(s)$ be the corresponding general value function
- The gradient of $v_{c,\gamma,\pi_\theta}(s)$ with respect to θ is itself a general value function that depends on the cumulant:

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

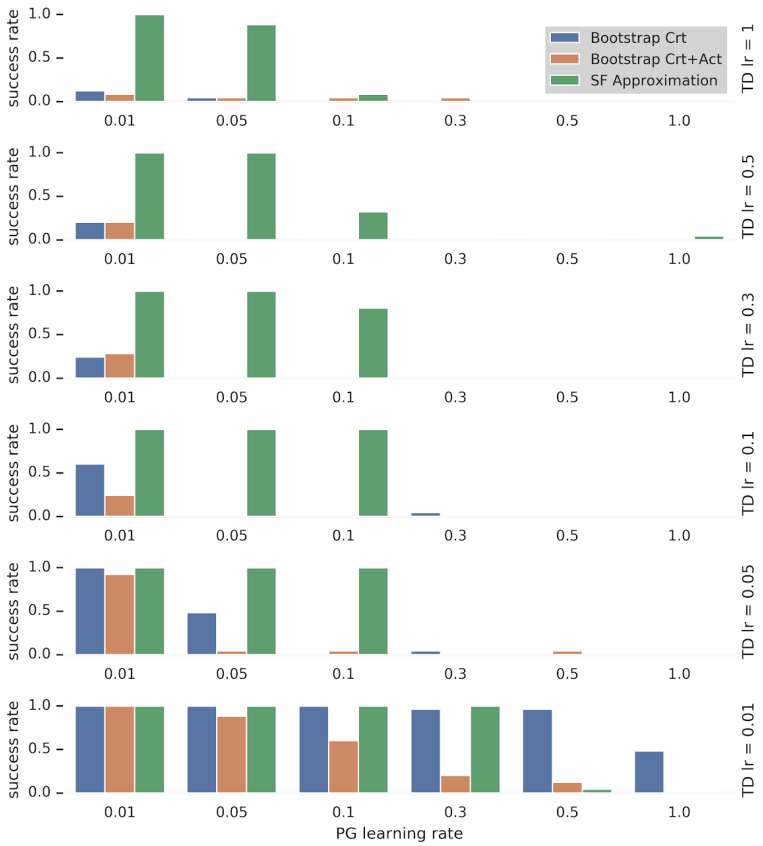
- In other words:

$$\nabla_\theta v_{c,\gamma,\pi_\theta}(s) = v_{\hat{c},\gamma,\pi_\theta}(s)$$

- A case in which *a GVF builds on another GVF and they can be modelled separately*

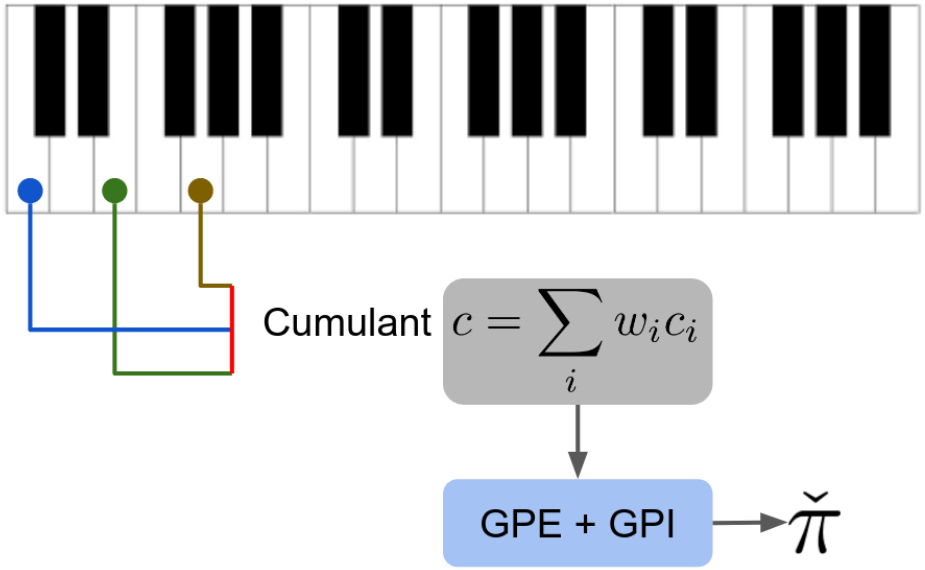
cf. Comanici et al, 2018

Empirical example (gridworld)



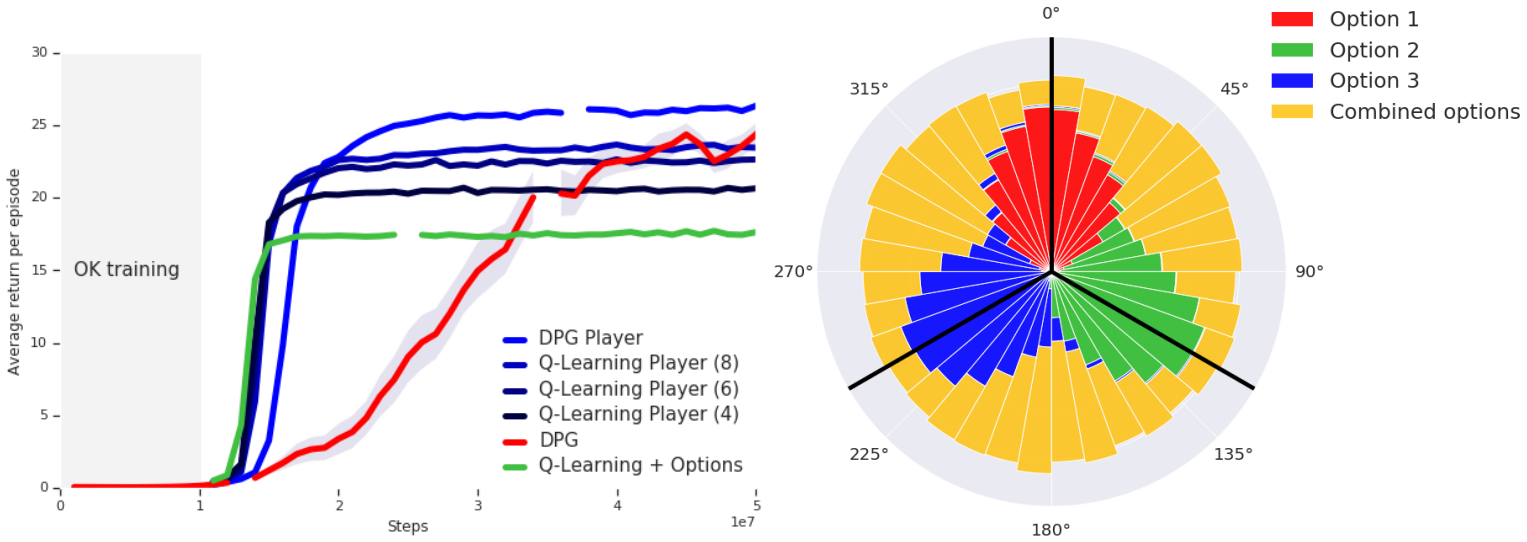
- GVF representation leads to more stable learning across parameter settings (green bars)
- *GVFs allow us to combine algorithms very easily!*

GVFs for synthesizing new behaviors



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

Option-Keyboard for Moving Target Arena



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

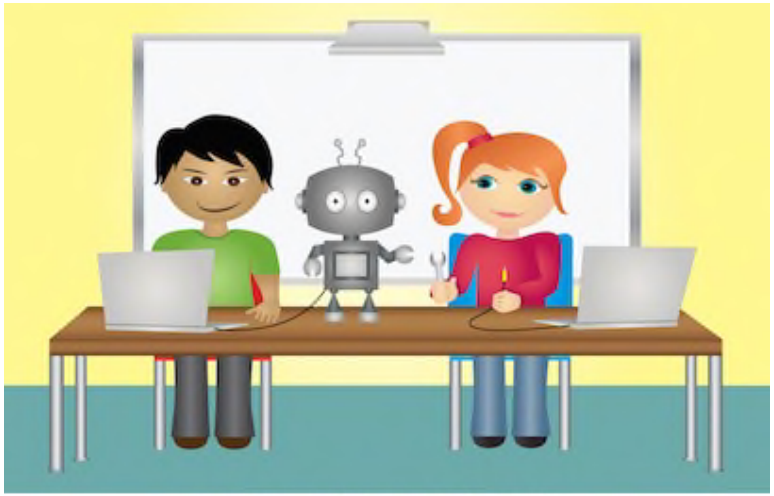
Discussion

- *Reinforcement learning suggests very powerful tools for knowledge representation*
- *Options* are a way to encode procedural knowledge
- We have made great progress in learning options through gradients
- Priors can be easily built into the option construction process through the optimization criterion
- Nice features of *generalized value functions*:
  - versatile
  - incorporate many existing algorithms as special cases
  - can be combined as building blocks
  - can be useful both for representation shaping and planning
- Open questions: how to generate behavior with/for such representations, how to do discovery
- *Bigger open question: how to evaluate empirically lifelong learning AI systems*

Assessing the capability of a life-long learning agent

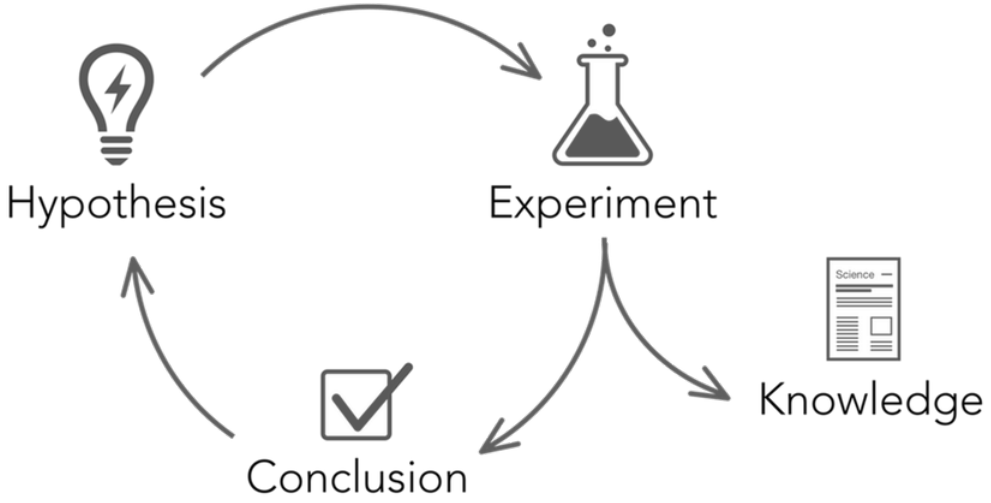
- There is no longer just a single task!
- Returns are important, but too simplistic
- Qualitative analysis of behavior interesting but difficult for drawing conclusions
- How well is the agent preserving and enhancing its knowledge?
- Problem of *methodology* not solved simply by open-sourcing or reproducibility

Testing the agent in multiple ways



- Take inspiration from school - ask multiple questions
- Try to devise a "certification" system for AI agents
- Maybe more important than current definitions of interpretability (which are akin to brain scans...)

Hypothesis-driven evaluation of continual learning systems



- Formulate a hypothesis about what the agent should know or how it should behave given certain knowledge
- Design an experiment to test this hypothesis
- Be patient and let the agent continue training without tinkering with the task or the algorithm!