Temporal Abstraction

Doina Precup
McGill University & DeepMind Montreal
dprecup@cs.mcgill.ca

With thanks to Rich Sutton, Satinder Singh, Pierre-Luc Bacon, Jean Harb, DeepMind Montreal

DLRL Toronto 2018

What is temporal abstraction?

- Consider an activity such as cooking dinner

  - High-level steps: choose a recipe, make a grocery list, get groceries, cook,...
  - Medium-level steps: get a pot, put ingredients in the pot, stir until smooth, check the recipe ...
  - Low-level steps: wrist and arm movement while driving the car, stirring, ...
- All have to be seamlessly integrated!

Temporal abstraction in AI

- A cornerstone of AI planning since the 1970's:
  - Fikes et al. (1972), Newell (1972, Kuipers (1979), Korf (1985), Laird (1986), Iba (1989), Drescher (1991) etc.
- It has been shown to :
  - Generate shorter plans
  - Reduce the complexity of choosing actions
  - Provide robustness against model misspecification
  - Allows taking shortcuts in the environment
- In robotics and hybrid systems, the use of controllers provides similar benefits, and also improves interpretability and allows specifying prior knowledge

Why is temporal abstraction useful for complex RL tasks

- *Advantages to planning*
  - Need to generate shorter plans
  - Improves robustness to model errors
  - Might need to look at fewer states, since the abstract actions have pre-defined termination conditions
  - Discretize the action space in continuous problems
- *Advantages to learning*
  - Improves exploration (can travel in larger leaps)
  - Gives a natural way of using a single stream of data to learn many things (off-policy learning)
- *Advantages to interpretability*:
  - Focusing attention: Sub-plans ignore a lot of information
  - Improves readability of both models and resulting plans
  - Reduces the problem size

Formalization of temporal abstraction in RL

- Hierarchical abstract machines (Parr, 1998)
- MAXQ (Dietterich, 1998)
- Dynamic motion primitives (Schaal et al. 2004)
- Skills (Konidaris et al, 2009)
- Feudal networks (Dayan & Hinton, 1992; Vezhnevets et al, 2016, 2017)
- *Options* (Sutton,Precup & Singh, 1999; Precup, 2000)

Options framework

- An *option* ω consists of 3 components
  - An *initiation set* of states $I_\omega \subseteq S$ (aka precondition)
  - A *policy* $\pi_\omega : S \times A \to [0, 1]$
    $\pi_\omega(s, a)$ is the probability of taking a in s when following the option
  - A termination condition $\beta_\omega : S \to [0, 1]$:
    $\beta_\omega(s)$ is the probability of terminating option ω upon entering state 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)
- Other representations of the termination condition are possible (cf. Comanici and Precup, 2010)

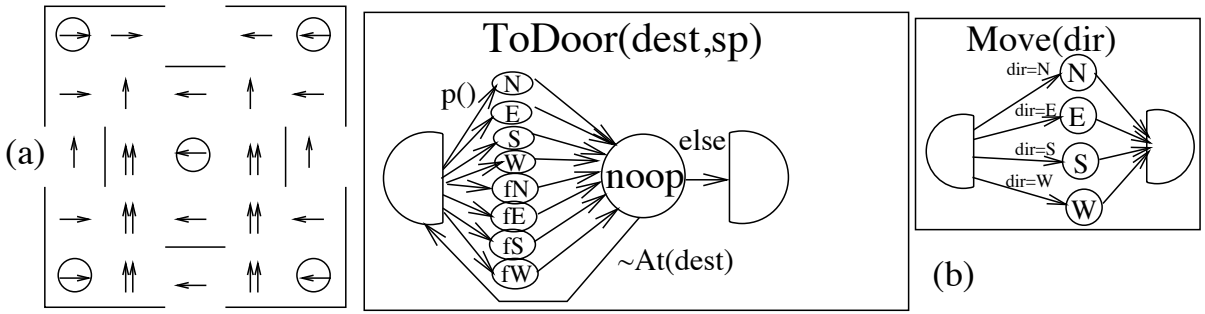
Cf. Sutton, Precup & Singh, 1999; Precup, 2000

Options as behavioral programs

- *Call-and-return execution*
  - Option is a subroutine which gets called by a policy over options π_Ω
  - When called, 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)
- *The option identity is also a form of memory: what is the agent currently trying to achieve?* Cf. Shaul et al, 2014, Kulkarni et al, 2016

Alternative formalisms

- MAXQ (Dietterich, 2000): specify a graph of sub-tasks
- Hierarchies of Abstract Machines (Parr & Russell, 1997): abstractions given by automata
- Andre (2003): programming language (ALISP) which allows specifying HAM programs, learning parts of them



Option models

- *Option model* has two parts:
  1. *Expected reward* $r_\omega(s)$: the expected return during ω's execution from s
     - Needed because it is used to update the agent's internal representations
  2. *Transition model* $P_\omega(s'|s)$: a sub-probability distribution over next states (reflecting the discount factor γ and the option duration) given that ω executes from s
     - Specifies both *where* the agent will end up after the option execution and *when* termination will happen
- Models are *predictions* about the future, conditioned on the option being executed

Option models provide semantics

- Programming languages: preconditions (initiation set) and postconditions
- *Models of options represent (probabilistic) post-conditions*
- *Models that are compositional*, can be used to reason about the policy over options, cf. Sutton et al, 1999, Sorg & Singh, 2010
  - *Sequencing*

$$\begin{aligned} r_{\omega_1\omega_2} &= r_{\omega_1} + P_{\omega_1} r_{\omega_2} \\ P_{\omega_1\omega_2} &= P_{\omega_1} P_{\omega_2} \end{aligned}$$

  - *Stochastic choice*: can take expectations of reward and transition models, eg if we have a policy over options Ω, its immediate reward is obtained as:

$$\sum_{\omega} \pi_{\Omega}(\omega|s) r_{\omega}(s)$$

  Convergence of Bellman equations follows immediately!

Recall: Bellman Equations (model-based RL)

- Given the model r_π and P_π for a policy:

$$v_\pi = r_\pi + \gamma P_\pi v_\pi$$

- Let $b = r_\pi$ and $F = \gamma P_\pi$:

$$v_\pi = b + F v_\pi$$

- We can then solve for v_π with :
  1. DP methods
  2. Sample-based TD methods (including Dyna)

Bellman equations for multi-step models

- Let's expand v once:

$$\begin{aligned} v_\pi &= r_\pi + \gamma P_\pi v_\pi \\ &= r_\pi + \gamma P_\pi (r_\pi + \gamma P_\pi v_\pi) \\ &= r_\pi + \gamma P_\pi r_\pi + \gamma^2 P_\pi^2 v_\pi \end{aligned}$$

- Define the *2-step reward model* as:

$$b^{(2)} \doteq r_\pi + \gamma P_\pi r_\pi$$

  and the *2-step transition model*:

$$F^{(2)} \doteq \gamma^2 P_\pi^2$$

- The Bellman equation can then also be written as:

$$v_\pi = b^{(2)} + F^{(2)} v_\pi$$

  (The gamma term is folded in F. A matter of taste...)

General n-step models

- Reward and transition matrix:

$$b^{(n)} \doteq \sum_{t=0}^{n-1} (\gamma P_\pi)^t r_\pi \qquad\qquad F^{(n)} \doteq (\gamma P_\pi)^n$$

- Bellman equation:

$$v_\pi = b^{(n)} + F^{(n)} v_\pi$$

- Note that The Bellman equation is a contraction with factor γ^n, so this can converge faster than a 1-step model

Fun fact: model composition

- Homogeneous coordinates: familiar from computer graphics, vision:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & t_x \\ u_{21} & u_2 & u_{23} & t_y \\ u_{31} & u_{32} & u_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  where the u_{ij} are elements of a rotation matrix and $[t_x, t_y, t_z]$ is a translation vector.
- Bellman equations in homogeneous form

$$\begin{bmatrix} v_\pi \\ 1 \end{bmatrix} = \begin{bmatrix} P_\pi & r_\pi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_\pi \\ 1 \end{bmatrix} = \begin{bmatrix} P_\pi v_\pi + r_\pi \\ 1 \end{bmatrix}$$

  where P_π is a block matrix of size $n \times n$ (n being the number of states) and r_π is a column vector of size n.

Composing models in homogeneous form

- A 2-step model is:

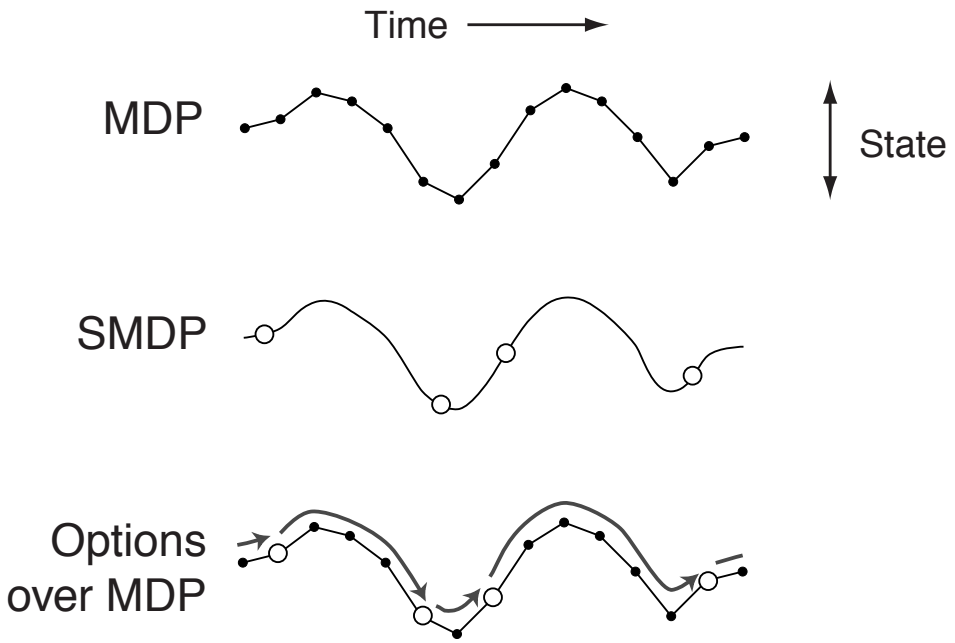
$$M_\pi^{(2)} \doteq M_\pi^2 = \begin{bmatrix} \gamma^2 P_\pi^2 & \gamma P_\pi r_\pi + r_\pi \\ 0 & 1 \end{bmatrix}$$

- Generally, $M_\pi^{(n)} \doteq M_\pi^n$ and $v_\pi = M_\pi^{(n)} v_\pi$
- For linear option models in homogeneous coordinated, sequencing is:

$$M_{\omega_1 \omega_2} = M_{\omega_1} M_{\omega_2}$$

- Stochastic choice is an expectation of M_{ω_i} taken wrt π_Ω
- Compositional planning (Silver & Ciosek, 2012): compute a whole hierarchy of options using this principle, staring with primitive models
  - Solves Towers of Hanoi order of magnitude faster
  - Requires number of iterations grows linearly when adding discs, instead of exponentially

MDP + Options = Semi-Markov Decision Precess



- Introducing options in an MDP induces a related semi-MDP
- Hence *all planning and learning algorithms* from classical MDPs transfer directly to options (Cf. Sutton, Precup & Singh, 1999; Precup, 2000)

TD at the SMDP level

- Let N_t now be a random variable for the duration of an option started at time t:

$$G_t^{(N_t)} = R_t + \gamma R_{t+1} + \ldots + \gamma^{N_t-1} R_{t+N_t-1} + \gamma^{N_t} Q(S_{t+N_t}, W_{t+N_t})$$

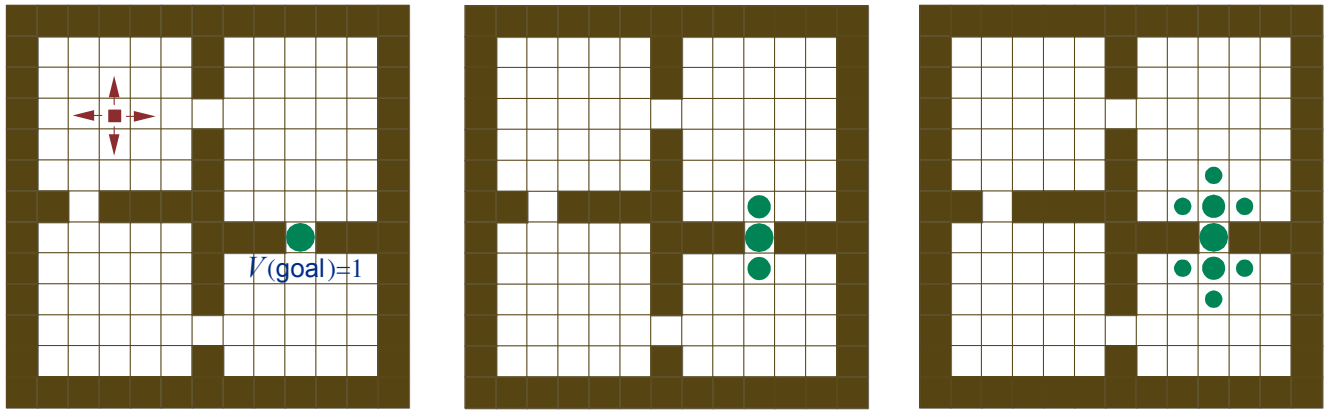
- It's just like using the *n-step return but where n is a random variable rather than being fixed.*
- SMDP TD Prediction (for option values)

$$Q_{t+N_t}(S_t, W_t) = Q_{t+N_t-1}(S_t, W_t) + \alpha \left(G_t^{(N_t)} - Q_{t+N_t-1}(S_t, W_t) \right)$$

- SMDP Q-learning: just take a $\max_\omega$ in the $G_t^{(N_t)}$ update target.
- Models of options can also be viewed as n-step models but with a random n

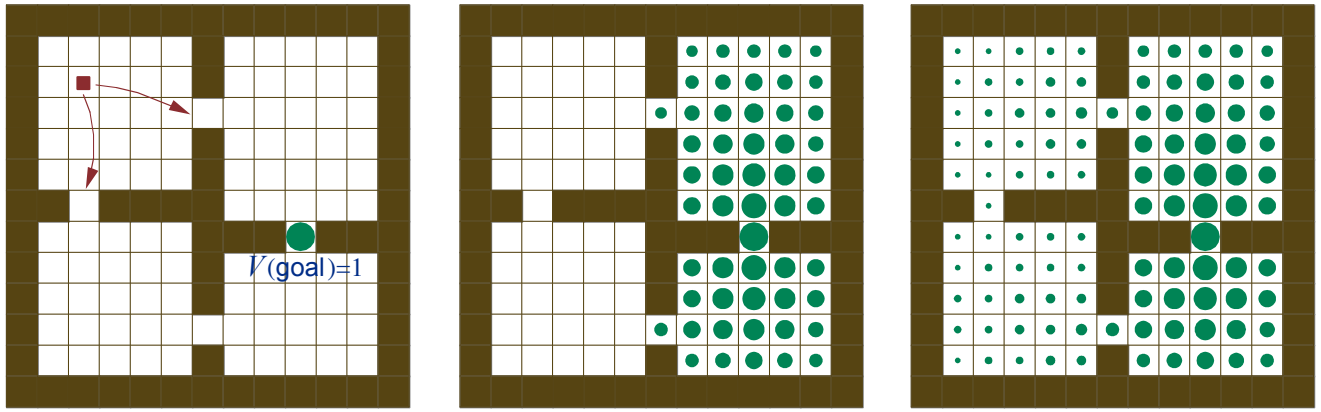
Illustration: Navigation

with cell-to-cell primitive actions



Iteration #0 | Iteration #1 | Iteration #2

with room-to-room options



Iteration #0 | Iteration #1 | Iteration #2

Advantages

- Easy to learn using temporal-difference-style methods, from a single stream of experience
- Planning with option models is done just like planning with primitives - *no explicit hierarchy*
- Result of planning with a set of options Ω is an option-value function, e.g. V_Ω
- *But we can also use the underlying MDP structure to help in learning the options*

The anatomy of the reward option model

- Primitive action model: $r_a(s) = \mathbb{E}[R_t | S_t = s, A_t = a]$
- Option model:

$$r_\omega(s) = \mathbb{E}[R_t + \gamma R_{t+1} + \ldots | S_t = s, \omega_t = \omega]$$

- This expectation indicates a Markov-style property, as it depends only on the identity of the state and the option, not on the time step
- Notice the *model is basically a value function* so we can write Bellman equations for the model:

$$r_\omega(s) = \sum_a \pi_\omega(a|s)[r_a(s) + \sum_{s'} \gamma(1 - \beta_\omega(s'))r_\omega(s')]$$

- This means that we can use RL methods to learn the models of options!
- Very similar equations hold for the transition model

Intra-option algorithms

- Learning about one option at a time is very inefficient
- In fact, we may not want to execute options at all!
- Instead, *learn about all options consistent with the behaviour*
- In some sense, a form of *attention*
- E.g. action-value function, tabular case

  On single-step transition $\langle s, a, r, s' \rangle$, for all ω that could have been executing in s and taken a:

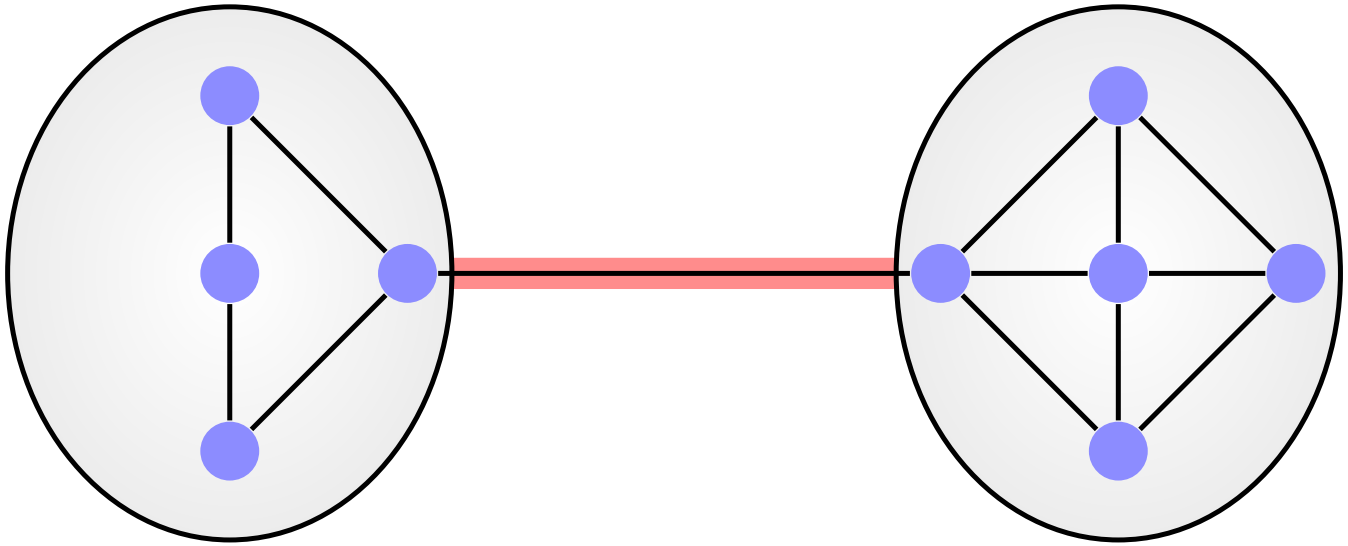
$$\begin{aligned} Q_\Omega(s,\omega) &= Q_\Omega(s,\omega) + \alpha[r_a(s) + \gamma(1-\beta_\omega(s'))Q_\Omega(s',\omega) + \\ &+ \gamma\beta_\omega(s')\sum_{s'}\max_{\omega'} Q_\Omega(s',\omega') - Q_\Omega(s,\omega)] \end{aligned}$$

- In general function approximation, importance sampling will need to be used (several papers on this)

Frontier: Option discovery

- Options can be given by a system designer
- If subgoals / secondary reward structure is given, the option policy can be obtained, by solving a smaller planning or learning problem (cf. Precup, 2000)
- *What is a good set of subgoals / options?*
- This is a *representation discovery* problem

Bottleneck states



[1]

- Perhaps the most explored idea in options construction (e.g. McGovern & Barto, 2002, Simsek & Barto, 2004)
- A bottleneck is a special state, which is visited more often than others, allows "circulating on the graph"

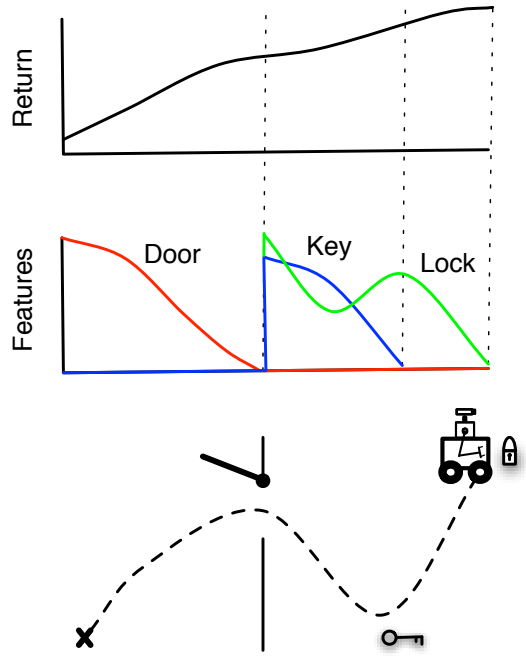
[1] Botvinick, 2011

Finding bottleneck 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. Solway et al, 2014)
- Laplace options (Machado et al, 2017)
- *Main drawback: expensive both in terms of sample size and computation*

Change-point detection

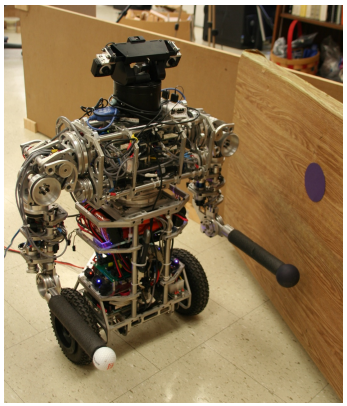
- Main idea: detect statistical changes in the time series generated by the behaviour, use this to segment into trajectories that are assumed to correspond to different options.

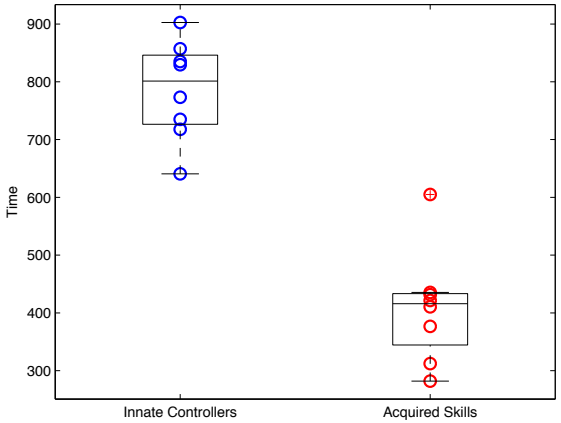


Cf. Konidaris et al., AAAI 2012

Applications: Robot control

- Controllers are learned automatically, at the same time as solving the task, using state-of-art Bayesian change point detection methods (Konidaris et al, 2009, 2012)





- Time to solve the problem is much shorter with the learned controllers (right) than using designed controllers

Applications: Learning from demonstration

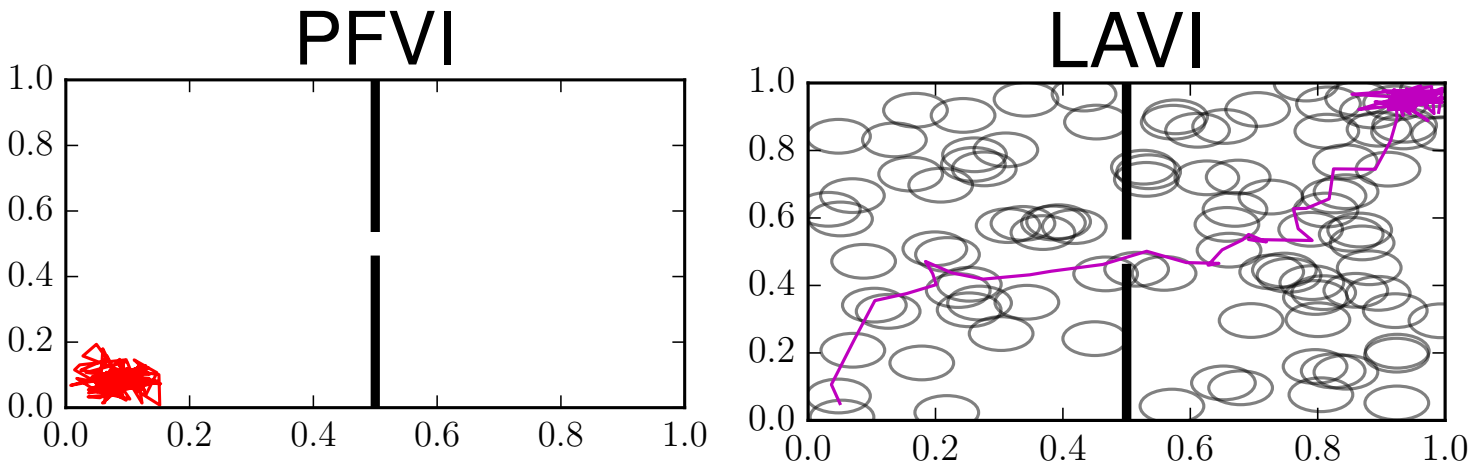
- Very important in human-robot interaction, game playing systems
- Key idea: take human-generated trajectories and decompose them into options
  - Each option policy and model is assumed to rely on a small set of state variables
  - Change-point detection is used to detect option boundaries
  - Once boundaries are detected, feature selection is used to decide which features are important
- Successfully applied to robot block manipulation and to PacMan

Cf. Shim & Thomaz, 2011; Subramanian, Isbell & Thomaz, 2011

Can we do something simple?

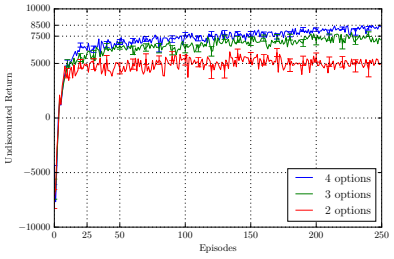
- *Generate a lot of options, then worry about which are useful!*
- More precisely, suppose we have a large set of *landmarks*, i.e. states in the environment, perhaps chosen at random (Mann et al, in press)
- Suppose we have a rough planner which can get to a landmark from its vicinity, by solving a *deterministic relaxation* of the MDP
- We use the landmarks to generate options, then use these in approximate value iteration

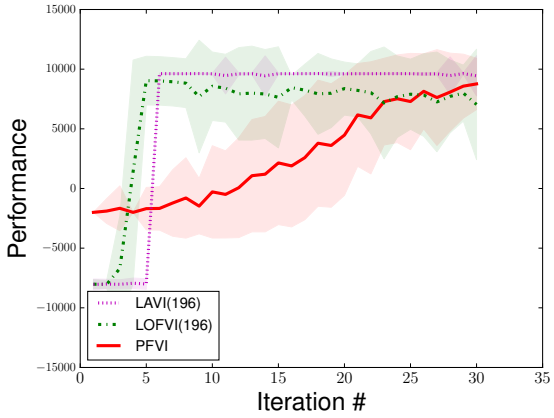
Illustration of random landmarks

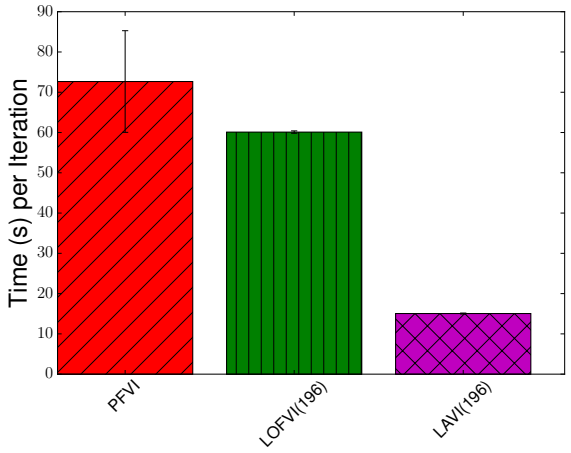


Landmark-based approximate value iteration gets a good solution much faster!

Pinball domain





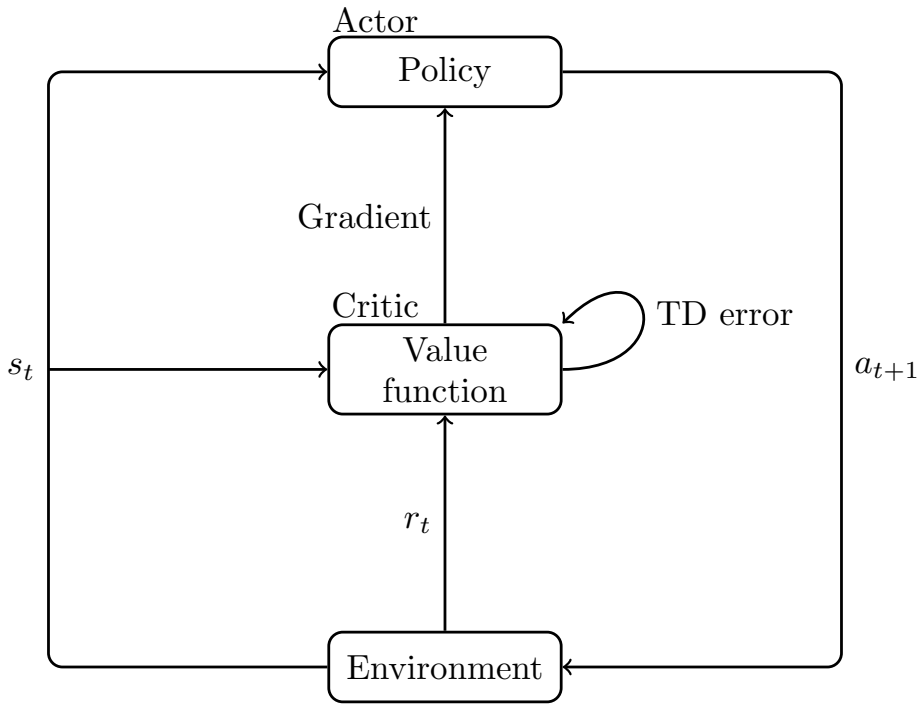


- Random landmarks are very useful
- Representing the value function only at landmark states provides significant speedup
- Consistent with theoretical analysis (in terms of concentrability coefficients)

Goals of our current work

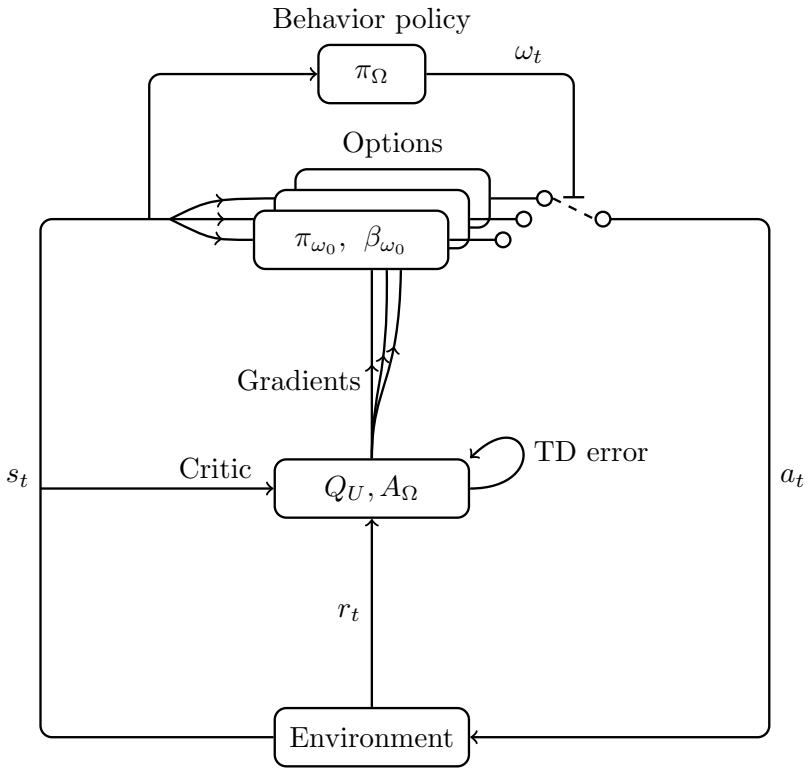
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or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



- Parameterize internal policies and termination conditions
- Policy over options is computed by a separate process (planning, RL, ...)

Formulation

- The option-value function of a policy over options π_Ω is given by

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

  where

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

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

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

- 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: $\mathbb{E}\left\{Q_{\pi_\Omega}(s,\omega)\right\}$
- The *gradient wrt the internal policy parameters* θ is given by:

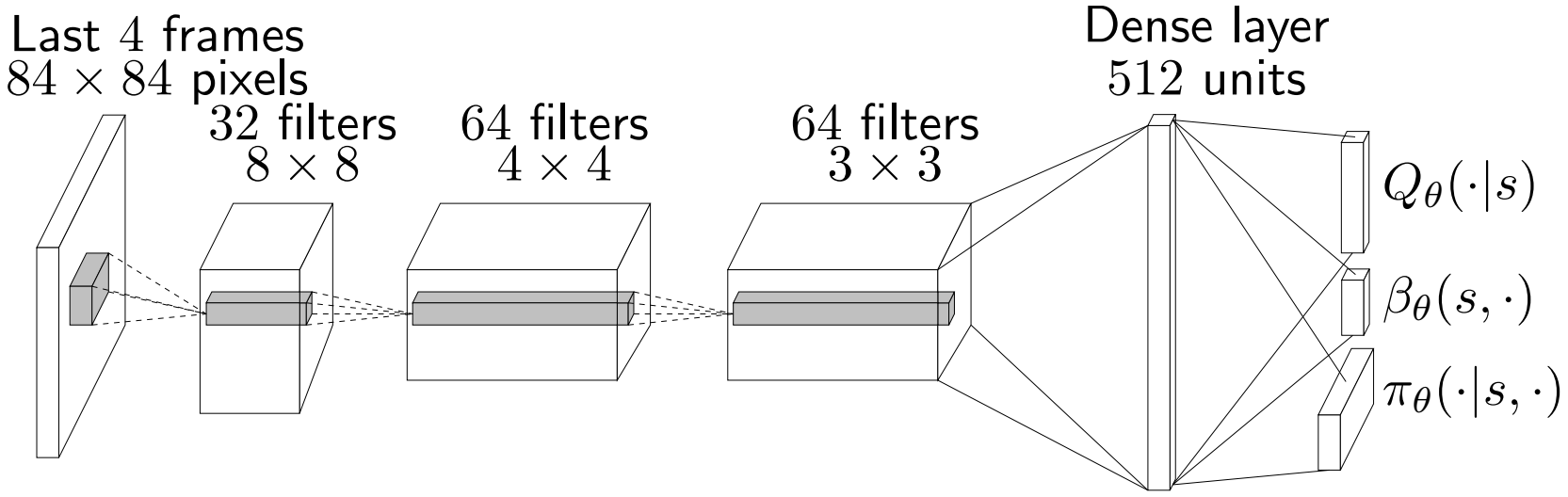
$$\mathbb{E}\left\{\frac{\partial \log \pi_{\omega,\theta}(a|s)}{\partial \theta} Q_U(s,\omega,a)\right\}$$

  This has the usual interpretation: *take better primitives more often* inside the option
- The *gradient wrt the termination parameters* ν is given by:

$$\mathbb{E}\left\{-\frac{\partial \beta_{\omega,\nu}(s')}{\partial \nu} A_{\pi_\Omega}(s',\omega)\right\}$$

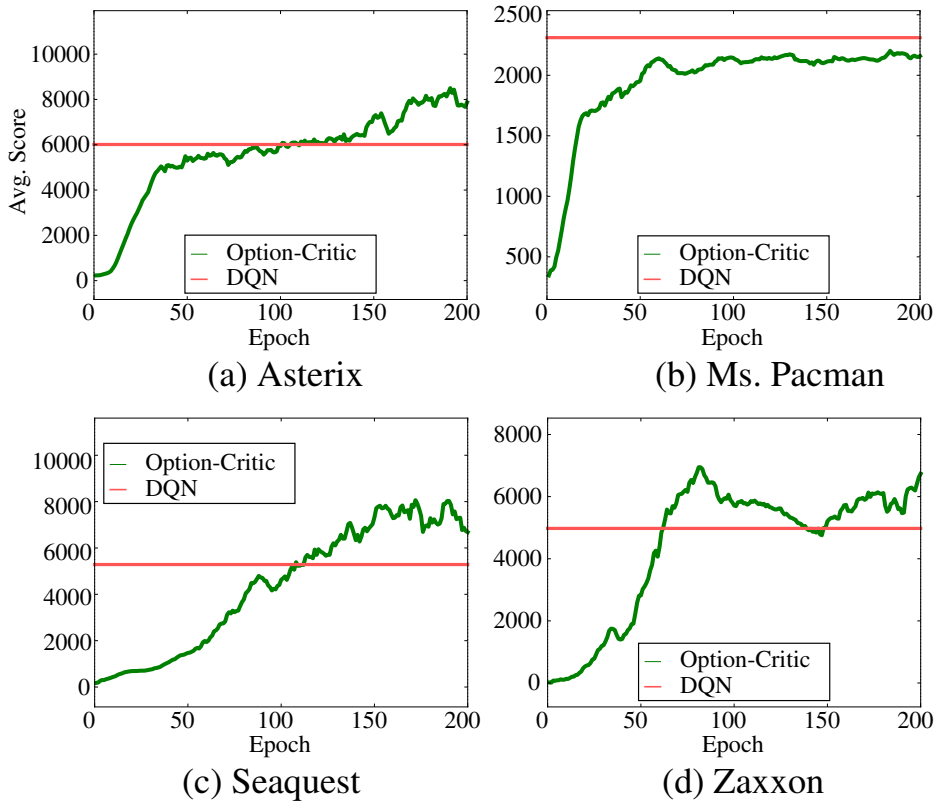
  where $A_{\pi_\Omega} = Q_{\pi_\Omega} - V_{\pi_\Omega}$ is the advantage function
  This means that we want to *lengthen options that have a large advantage*

Option-Critic with Deep Neural Netowrks



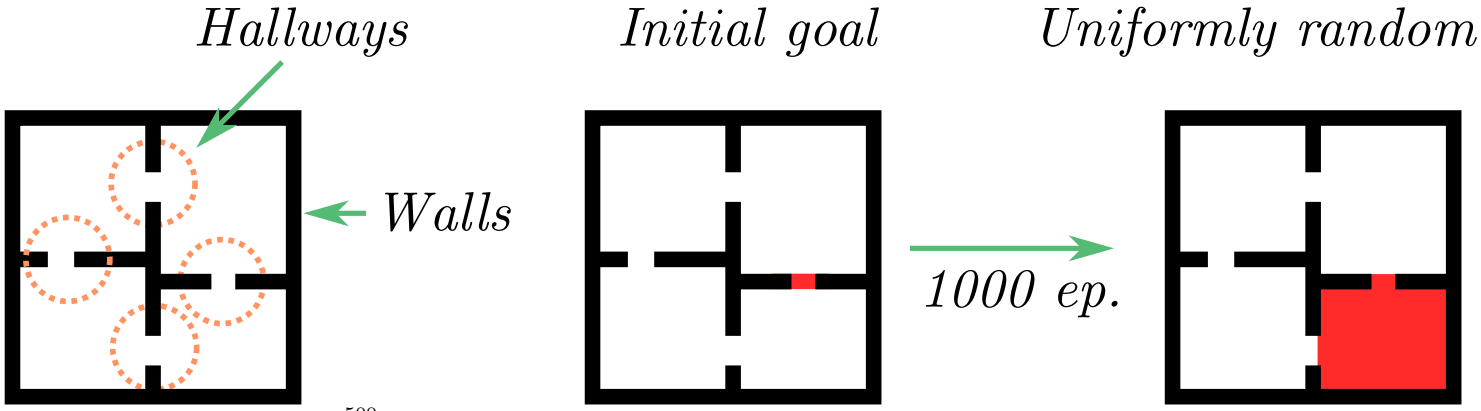
- Atari simulator
- Parameters shared across all options
- Either DQN-style or advantage asynchronous option-critic (A2OC)
- Only number of options needs to be pre-specified

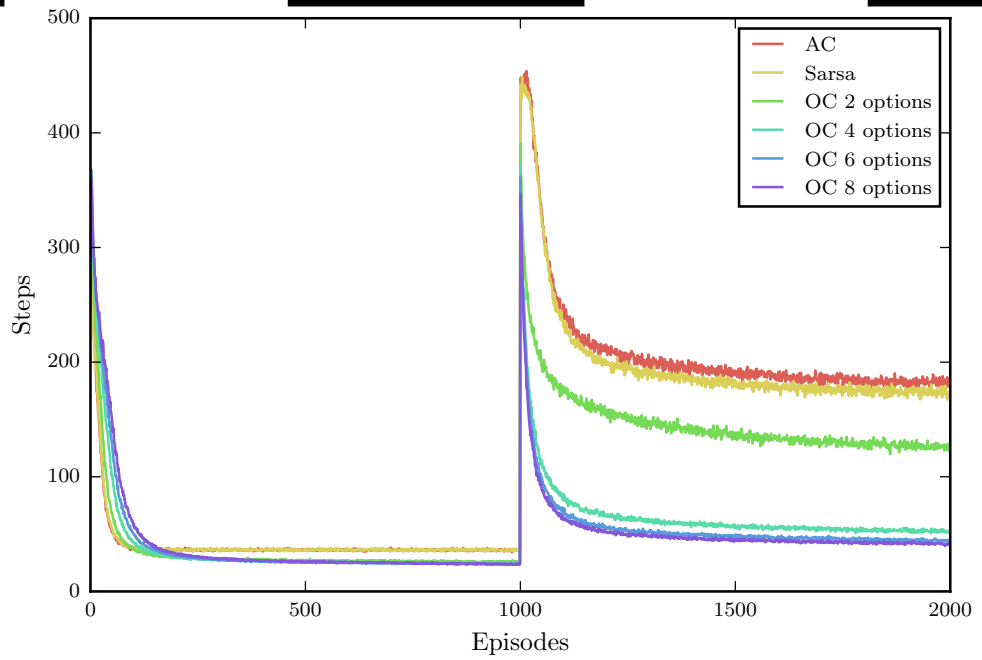
Results: Atari



- 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

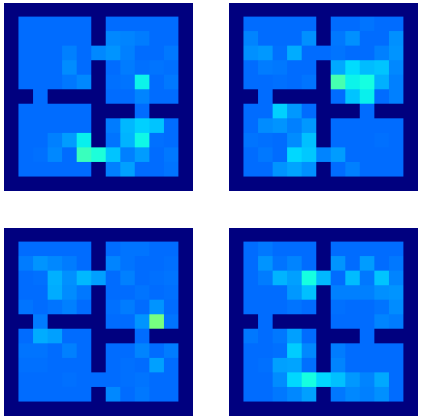
Results: Transfer in Rooms Domain



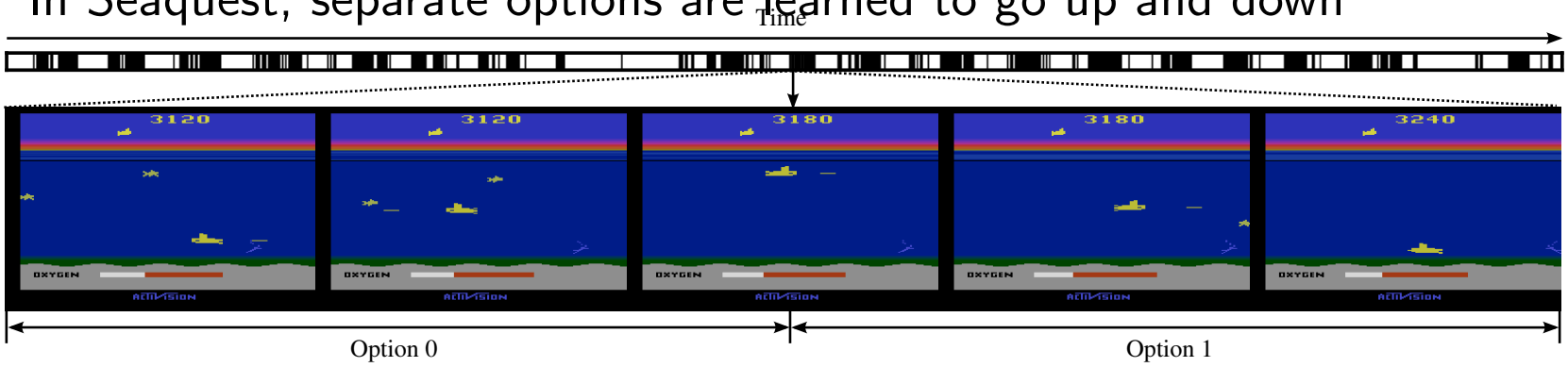


Results: Learned options are intuitive

- In rooms environment, terminations are more likely near hallways (although there are no pseudo-rewards provided)



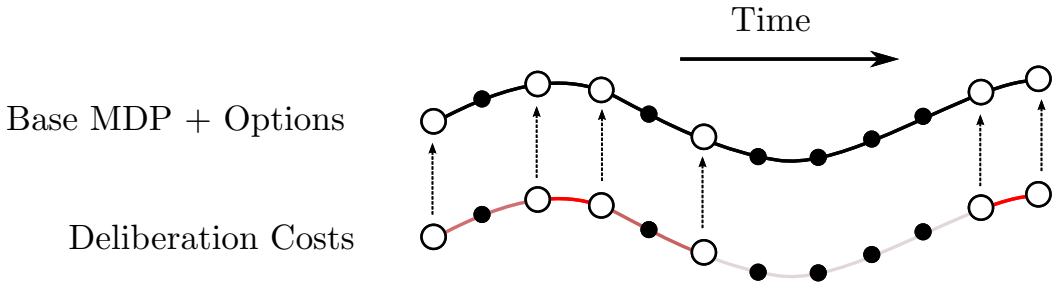
- In Seaquest, separate options are learned to go up and down



Regularization

- Entropy-based regularization is useful to encourage diversity in the options
- *Length collapse:* options "dissolve" into primitive action over time
- Assumption: *executing a policy is cheap, deciding what to do is expensive*
  - Many choices may need to be evaluated (branching factor over actions)
  - In planning, many next states may need to be considered (branching factor over states)
  - Evaluating the function approximator might be expensive (e.g. if it is a deep net)
- Deliberation is also expensive in animals:
  - Energy consumption (to engage higher-level brain function)
  - Missed opportunity cost: thinking too long means action is delayed

Deliberation/Switching Cost



- Let $c(s, \omega) < 0$ be the immediate cost of switching to ω in s
- *New value function expresses total deliberation cost*:

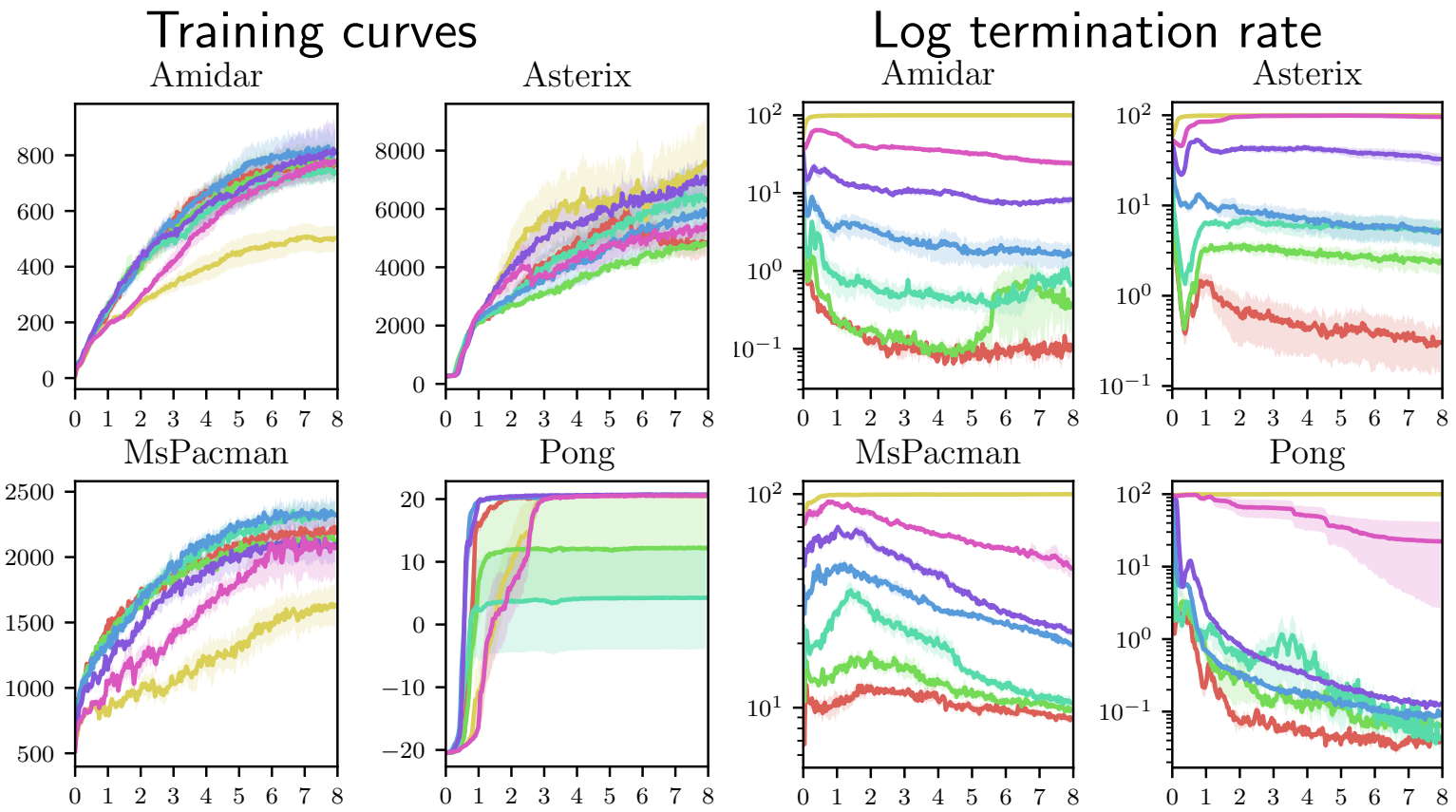
$$Q_c(s, \omega) = c(s, \omega) + \sum_{s'} P(s'|s, \omega) \sum_{\omega'} \mu(\omega'|s') Q_c(s', \omega')$$

- *New objective: maximize reward with reasonable effort*

$$\max_{\Omega} \mathbb{E}\left[Q_\Omega(s, \omega) + \tau Q_c(s, \omega)\right]$$

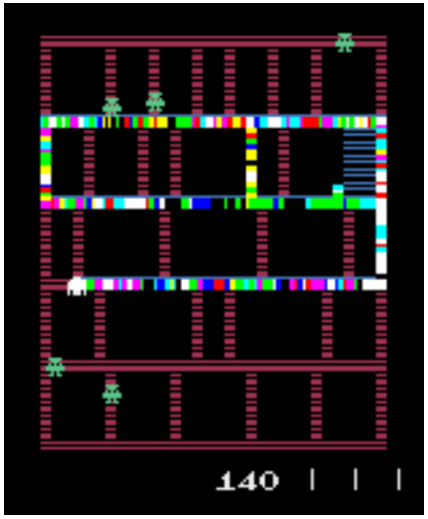
  $\tau \geq 0$ controls the trade-off between value and computation effort
- Can be shown equivalent to requiring that *advantage exceeds a threshold* before switching

Effect of Deliberation Cost Regularization

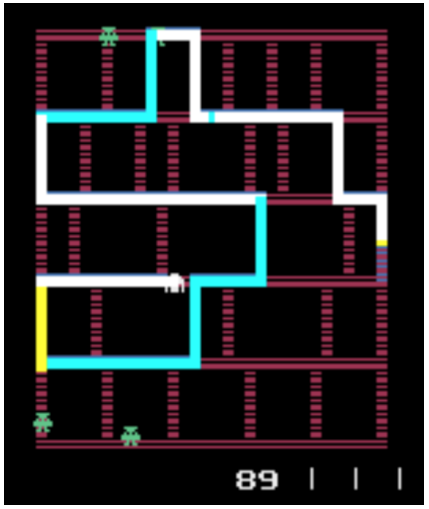


Yellow: no regularization; red: most regularization

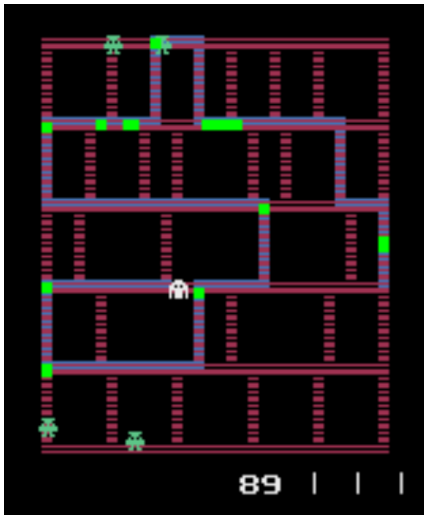
Example: Amidar



(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.

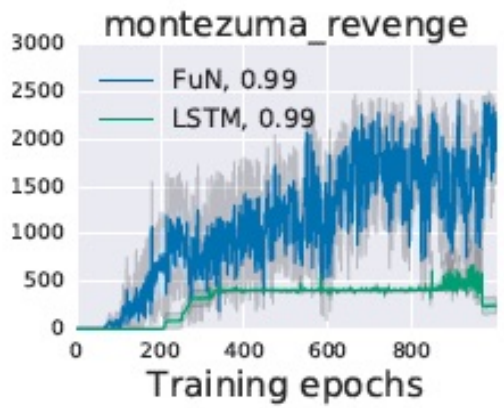
- Deliberation costs prevent options from becoming too short
- Terminations are intuitive

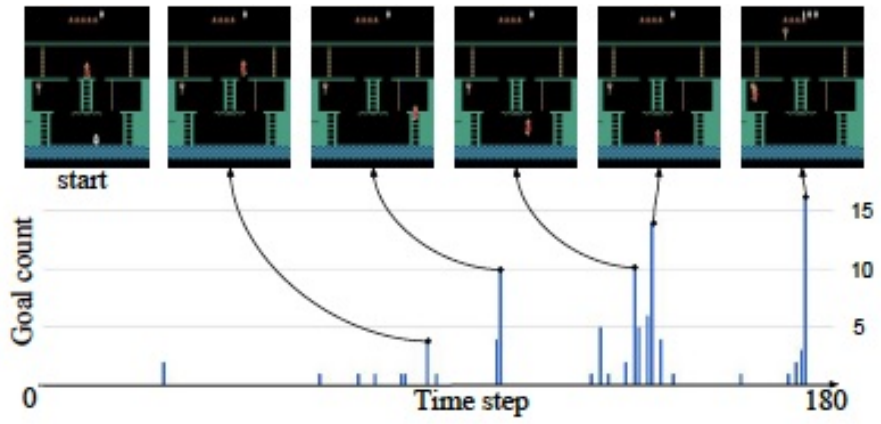
Relationship to other ideas

- Human problem solving: Solway et al (2014) proposed a Bayesian model selection framework to explain subgoal learning by humans trying to navigate an unknown map
- Humans found subgoals "around" bottleneck states
- Inspection of their criterion (Bayesian fit to the data) shows strong similarity with using a cost equal to the branching factor between options, plus the branching factor within the active option
- *Deliberation costs can also explain the value of options for exploration*
  - Travelling quickly around the environment means values will become accurate more quickly
  - The best action becomes clear earlier, which would make it easier to choose

Feudal Networks

- Fully hierarchical model with a manager and a worker (Dayan & Hinton, 1992)
- Manager provides goals to the worker (which determine the worker's reward)
- Manager is rewarded by the environment
- Vezhnevets et al, 2016, 2017: specific implementation using cosine distance to measure goal achievement, separate fixed time scale for manager and worker, state-of-art results on some tasks





Temporally abstract knowledge representation: 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:

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

- *Cumulant* c is a function that can depend on the S_t, or (S_t, A_t) or (S_t, A_t, S_{t+1}) and can output a vector (even a matrix)
  Eg $C_{k+1} = c(S_k, A_k, S_{k+1})$
- *Continuation function* γ maps states to [0,1]
- Cf. Horde architecture (Sutton et al, 2011); Adam White's thesis; inspiration from Pandemonium architecture

Option models are GVFs!

- Option reward model is defined as:

$$r_\omega(s) = \sum_a \pi_\omega(a|s)[r_a(s) + \sum_{s'} \gamma(1 - \beta_\omega(s'))r_\omega(s')]$$

- This means the option reward model is a GVF, $v_{\omega,r}$: cumulant is the environment reward r, policy is π_ω,. continuation function is $\gamma(1 - \beta_\omega)$
- Note that *we could use $\gamma = 1$ for the overall task* and these GVFs would still be well defined
- *Option transition model can similarly be shown to be a GVF with a vector-valued cumulant*

GVFs generalize many other temporal abstraction quantities!

- Universal Value Function Approximators (UVFAs)
- Option value functions
- Feudal Networks
- Successor features
- ....
- *Focus of GVFs is NOT execution but modelling*
- GVFs help us identify the tradeoffs of different HRL models

Summary

- *Temporal abstraction* helps scale up RL algorithms by reducing the branching factor and the number of decisions, and improving temporal credit assignment
- As a result the speed of learning and planning can increase considerably
- Temporal abstraction is *not just about execution/policies but also about modelling*
- *Options and models may not need to be very carefully crafted*
- *Options and models can be learned end-to-end from data*

Research frontier

- What should option sets be optimizing?
- Move away from thinking about execution to thinking about planning
- Decrease variance induced by off-policy learning
- Improve sensitivity to time scales
- Much more practical experience with temporal abstraction!