From temporal abstraction to programs

Doina Precup
McGill University

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

NIPS NAMPI workshop, December 2016

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!
- Cf. macro actions in classical AI, controllers in robotics

Formalization of temporal abstraction

- Hierarchical abstract machines (Parr, 1998)
- MAXQ (Dietterich, 1998)
- Dynamic motion primitives (Schaal et al. 2004)
- Skills (Konidaris et al, 2009)
- *Options* (Sutton, Precup & Singh, 1999; Precup, 2000)

Options framework

- Suppose we have an MDP $\langle \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$
- An *option* ω consists of 3 components
  - An *initiation set* of states $I_\omega \subseteq \mathcal{S}$ (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)

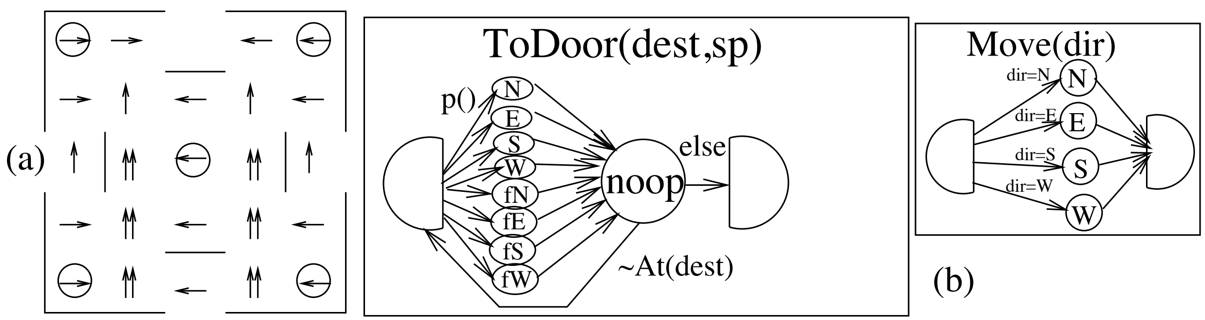
Cf. Sutton, Precup & Singh, 1999; Precup, 2000

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
     - P specifies *where* the agent will end up after the option/program execution and *when* termination will happen
- Models are *predictions* about the future, conditioned on the option being executed

Option models provide semantics

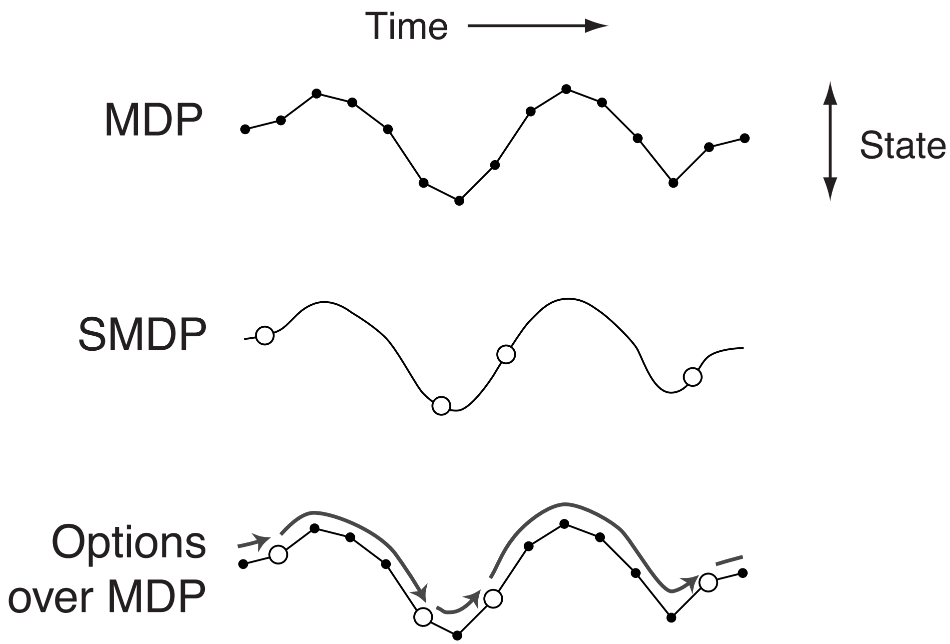
- Programming languages: preconditions (initiation set) and postconditions
- *Models of options represent (probabilistic) post-conditions*
- *Models that are compositional*, can be used to reason about the policy over options
- *Sequencing*

$$
\begin{aligned}
\mathbf{r}_{\omega_1 \omega_2} &= \mathbf{r}_{\omega_1} + P_{\omega_1} \mathbf{r}_{o_2} \\
P_{\omega_1 \omega_2} &= P_{\omega_1} P_{\omega_2}
\end{aligned}
$$

  Cf. Sutton et al, 1999, Sorg & Singh, 2010
- *Stochastic choice*: can take expectations of reward and transition models
- These are sufficient conditions to allow Bellman equations to hold
- Silver & Ciosek (2012): re-write model in one matrix, compose models to construct programs

  Eg. good generalization in Towers of Hanoi

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)
- But planning and learning with options can be much faster!

Illustration: Navigation

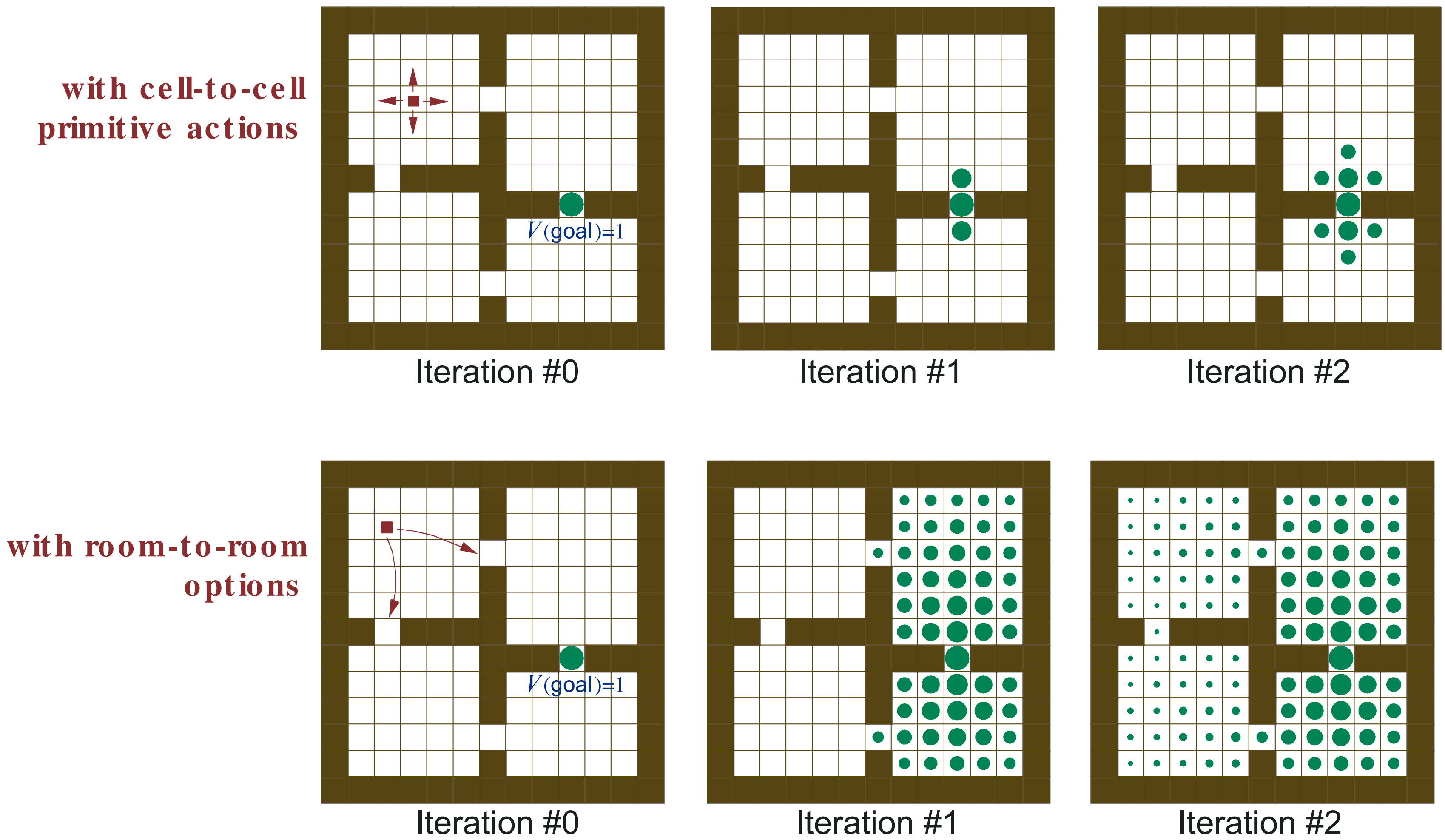
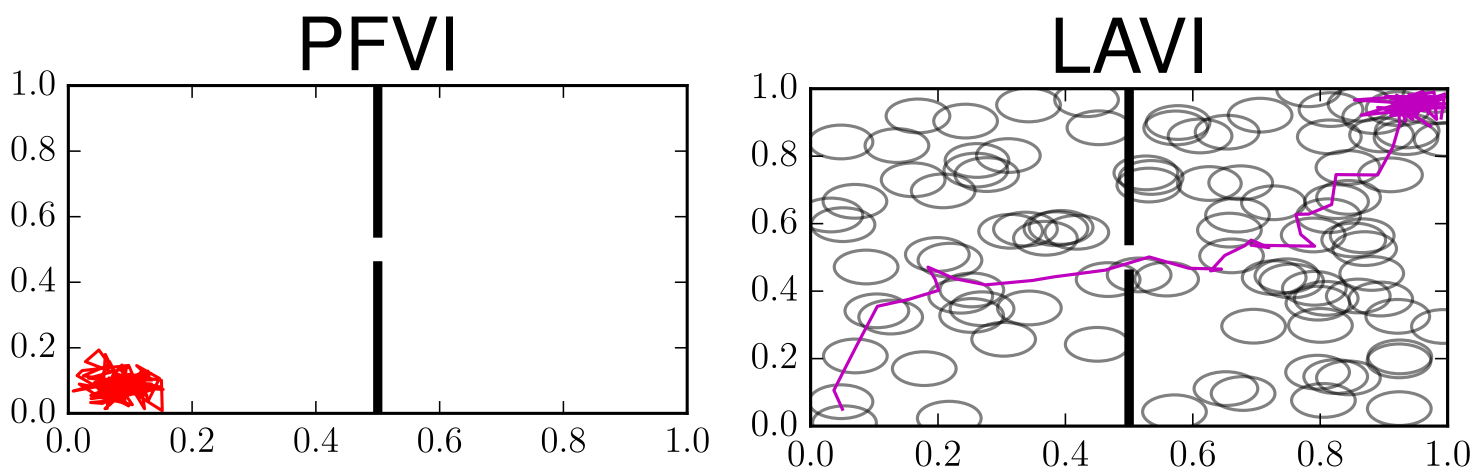


Illustration: Random landmarks

- *Generate a lot of options, then worry about which are useful!*
- Large set of *landmarks*, i.e. states in the environment, chosen at random (Mann, Mannor & Precup, 2015)
- Rough planner which can get to a landmark from its vicinity, by solving a *deterministic relaxation* of the MDP



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

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:

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

- 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:

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

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

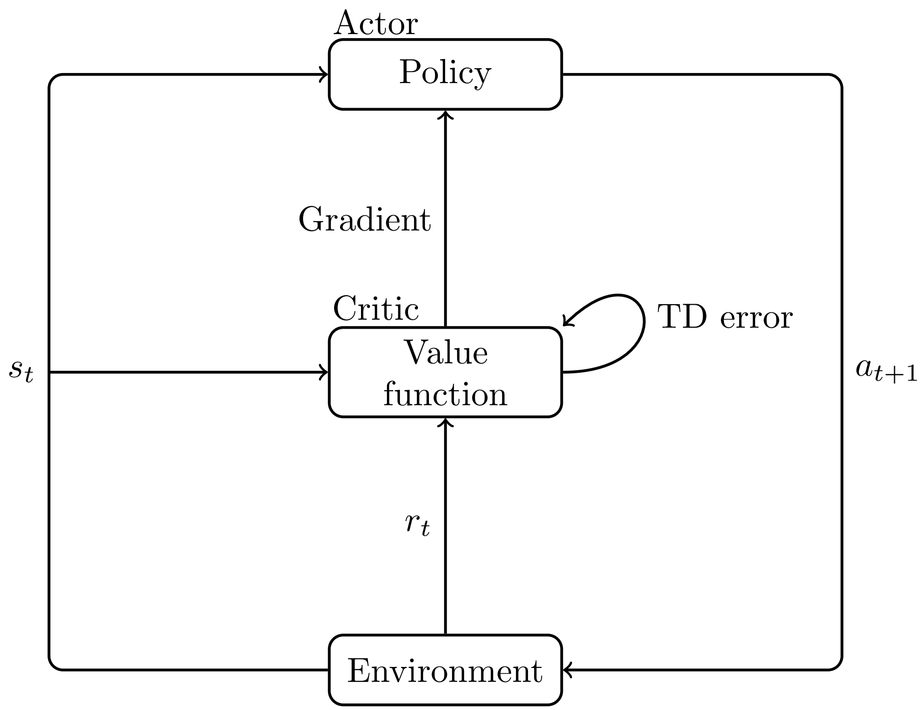
Frontier: Option Discovery

- 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
- Studied a lot over the last 15 years
- Bottleneck states and change point detection currently the most successful methods

Goals of our current work

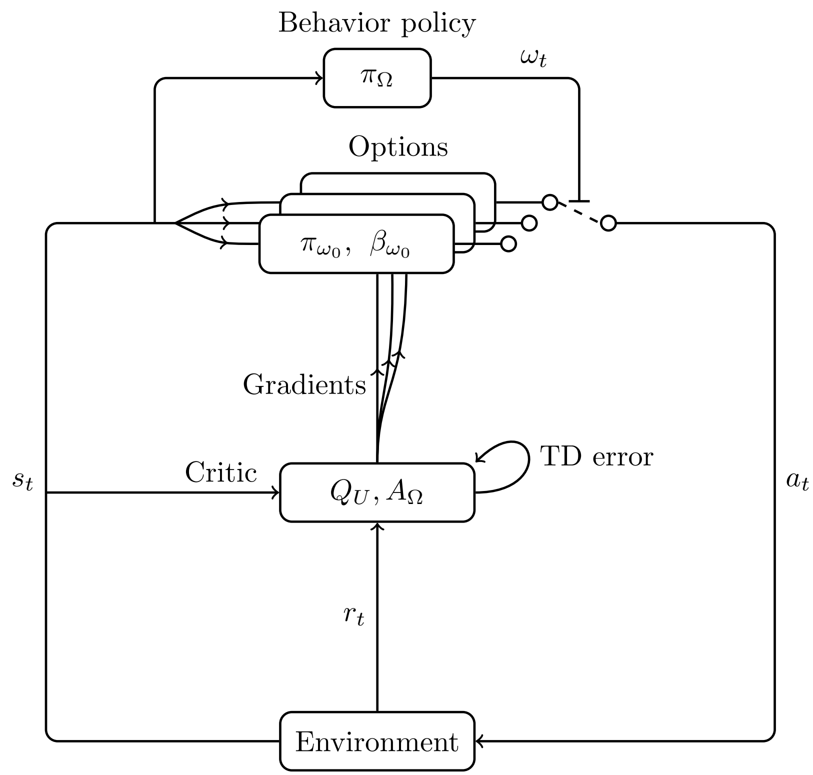
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or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

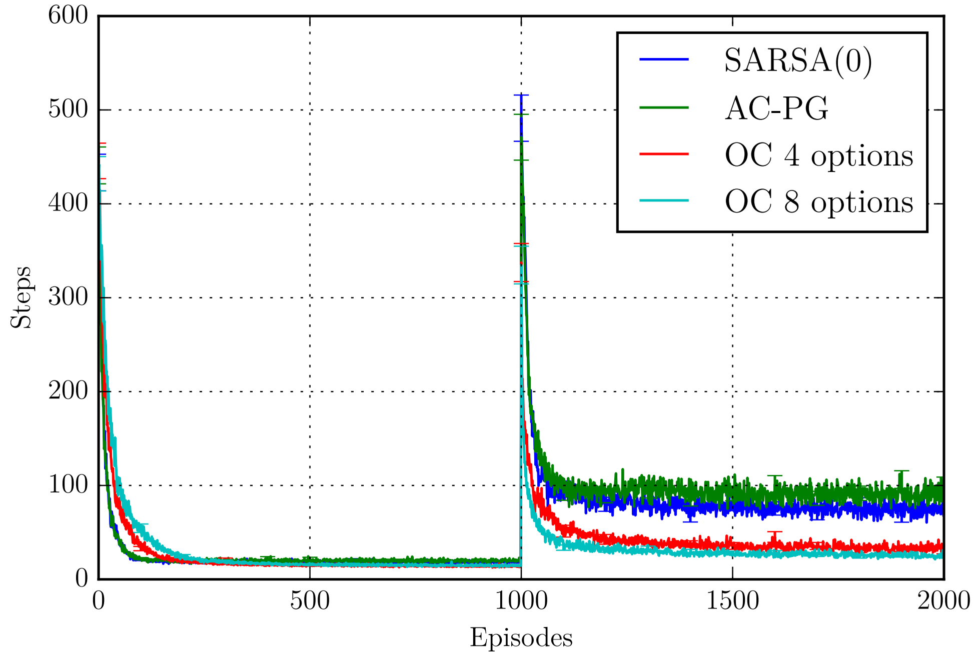
$$\mathbb{E}\left\{\frac{\partial \log \pi_{\omega,\theta}(a|s)}{\partial \theta} Q_U(s,\omega,a)\right\}$$

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

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

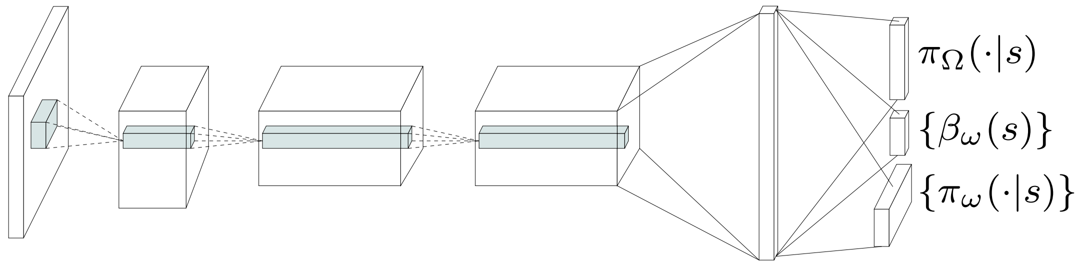
  where $A_{\pi_\Omega} = Q_{\pi_\Omega} - V_{\pi_\Omega}$ is the advantage function
  This means that we want to *lengthen options that have a large advantage*

Results: Options transfer

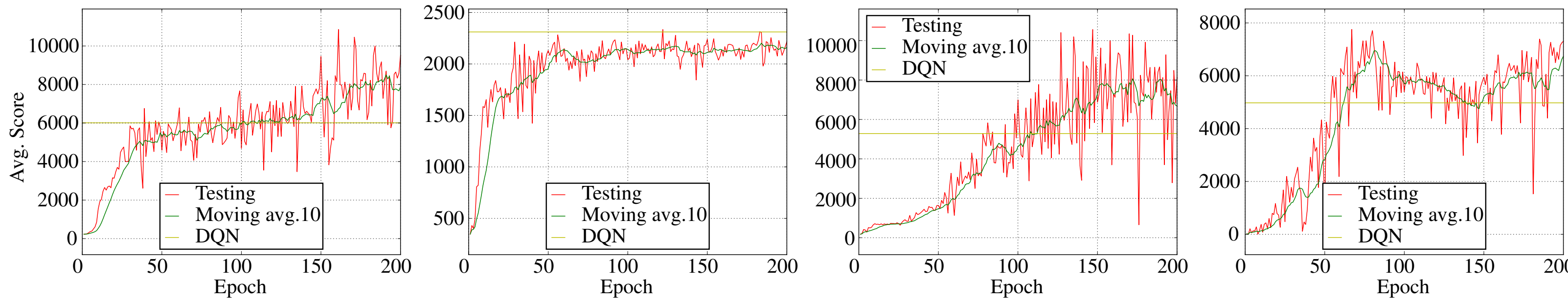


- 4-rooms domain, tabular representations, value functions learned by Sarsa
- Learning in the first task no slower than using primitives
- Learning once the goal is moved faster with the options

Results: Nonlinear function approximation



- Atari simulator, DQN to learn value function over options, actor as above

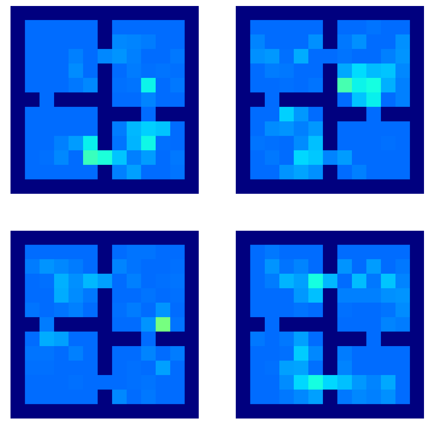


(a) Asterix (b) Ms. Pacman (c) Seaquest (d) Zaxxon

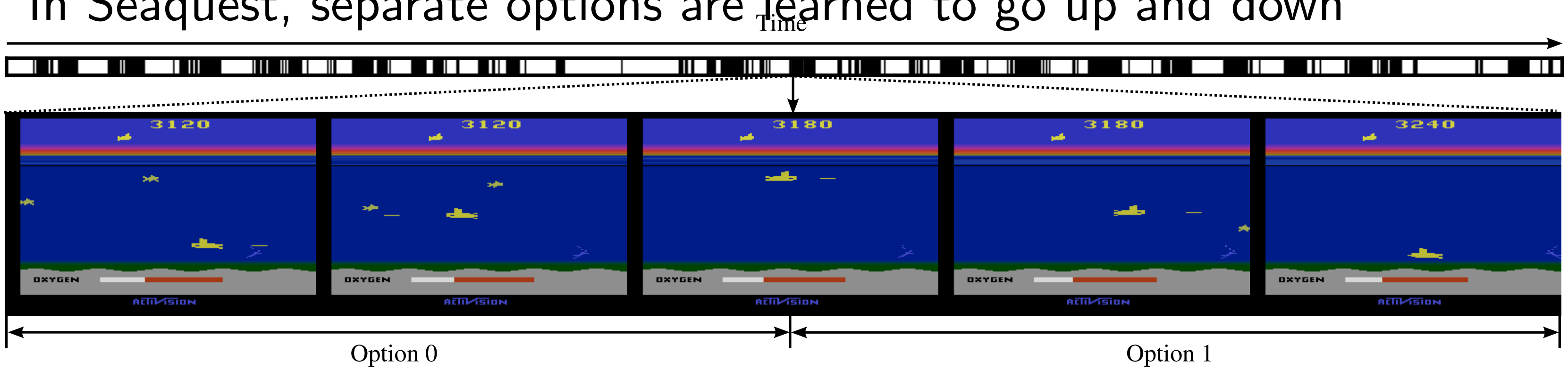
- Performance matching or better than DQN

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



Conclusions and future work

- Temporal abstraction methods developed in reinforcement learning provide syntax and semantics of behavioral programs
- Option-critic allows using policy gradient ideas for continual option construction
- Lots of things to do:
  - More empirical work in option construction
  - Tighter integration with Neural Turing Machines and similar models
  - *Other execution models*