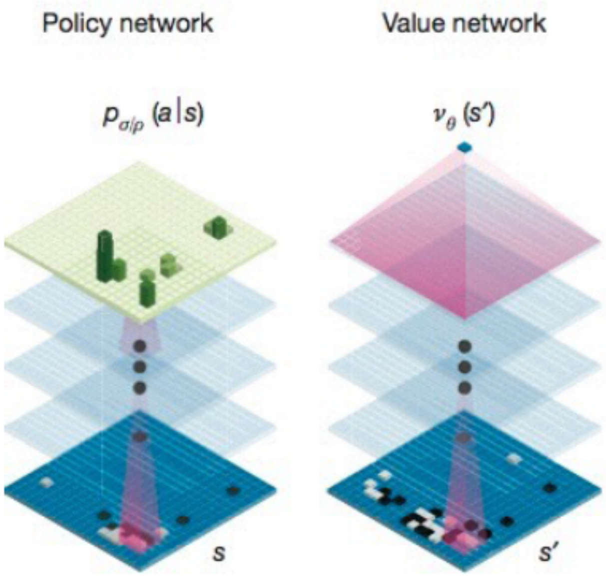
Hierarchical Reinforcement Learning

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

Knowledge in AlphaGo



- *Policy*: what to do (probability of action given current "state") - ie *procedural knowledge*
- *Value function*: estimation of expected long-term return - ie *predictive knowledge*

From Reinforcement Learning to AI

- Growing knowledge and abilities in an environment
- Learning efficiently from one stream of data
- Reasoning at multiple levels of abstraction
- Adapting quickly to new situations

Building Knowledge with Reinforcement Learning

- Focusing on two types of knowledge:
  - *Procedural knowledge*: skills, goal-driven behavior
  - *Predictive, empirical knowledge*: predicting effects of actions

- Knowledge must be:
  - *Expressive*: able to represent many things, including abstractions (objects, places, high-level strategies...)
  - *Learnable*: from data, ideally without supervision (for scalability)
  - *Composable*: suitable for fast planning by assembling existing pieces

Abstraction and generalization

- An *abstract representation* ignores low-level details of the problem, or modifies the problem representation altogether

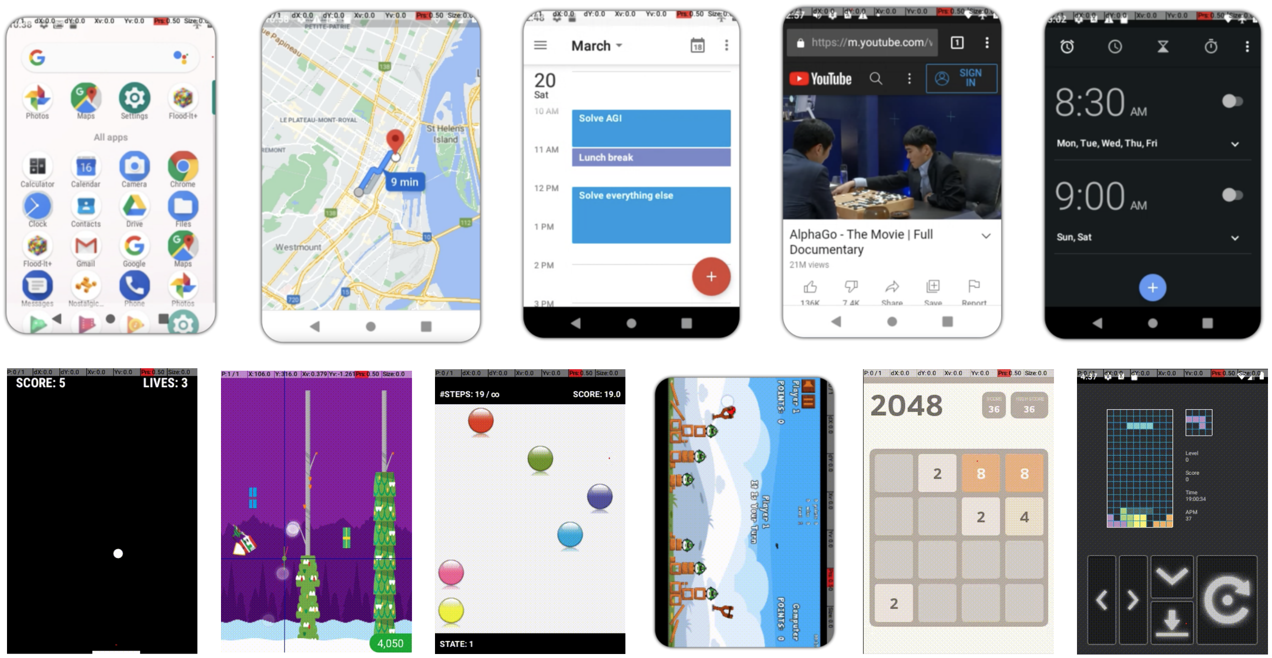
  Eg. addresses vs exact coordinates

- *Generalization* is the ability to take knowledge acquired in some circumstances and applying it in different circumstances

  Eg. Being good at some games helps us learn other games faster

- These two concepts are related but not identical: an abstract representation may helps us to generalize
- Generalization is often achieved in AI/ML by using function approximation (eg deep nets)
- In RL, we have an extra important dimension: *time/action* - can we build abstraction/generalization here too?

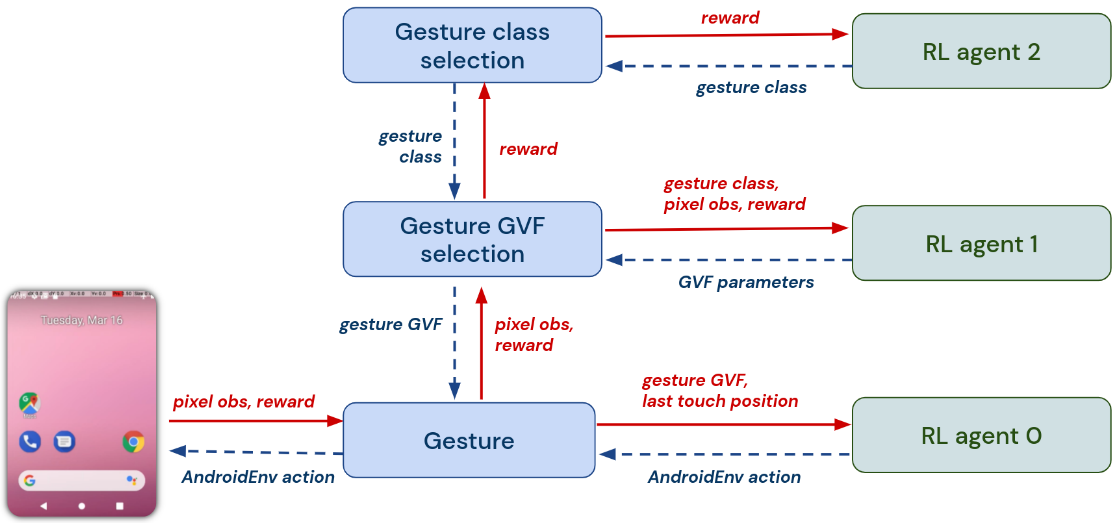
Motivating Example: Learning to Manipulate Complex Interfaces



- Agent interacting with a phone screen, learning how to control apps
- Native action space: touch anywhere on the screen

Toyama, Hamel, Gergely, Comanici et al (2021), https://arxiv.org/pdf/2105.13231.pdf

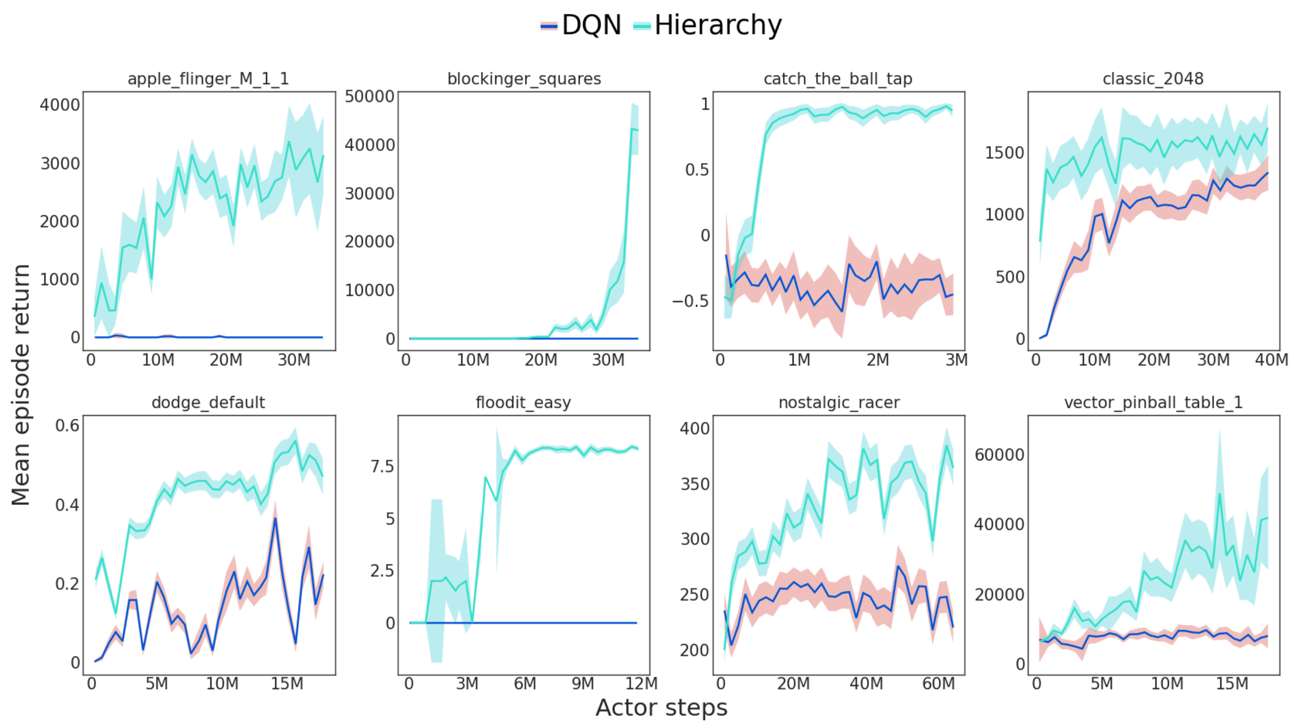
Example: Using Abstraction to Structure Learning



- Instead of primitive actions, learn and use gestures (tap, swipe, fling)
- Value functions predict reward associated with different gesture goals
- *Learning happens in parallel at all levels of abstraction*

Comanici, Glaese, Gergely, Toyama et al (2022) https://arxiv.org/pdf/2204.10374.pdf

Learning knowledge at multiple levels of abstraction drastically improves performance



Comanici, Glaese, Gergely, Toyama et al (2022) https://arxiv.org/pdf/2204.10374.pdf

What is temporal abstraction?

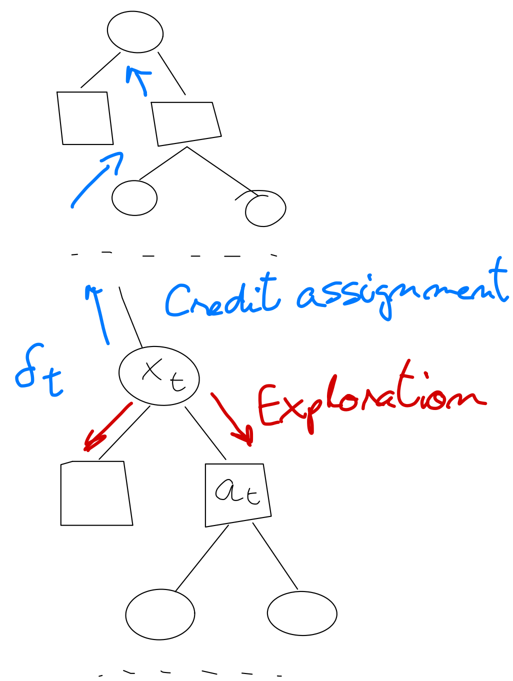
- Consider an activity such as cooking dinner

  - 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

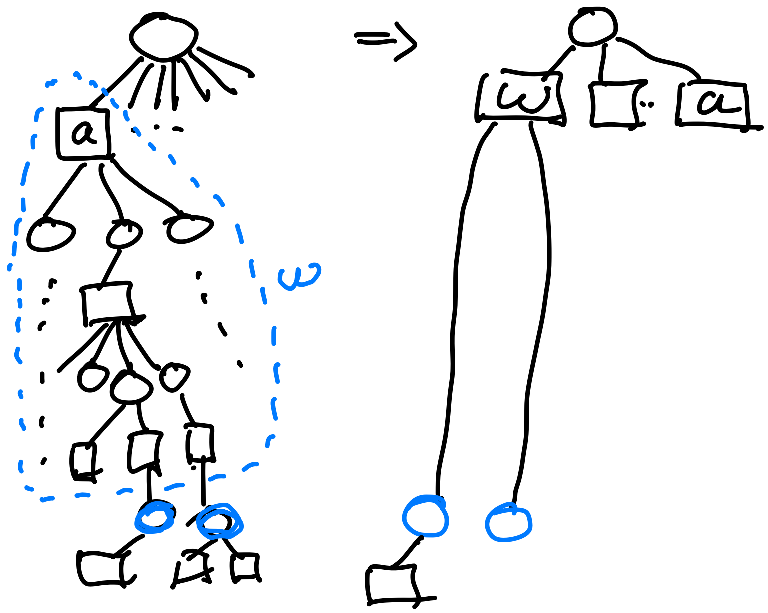
Recall: RL cartoon



Goals of temporal abstraction:

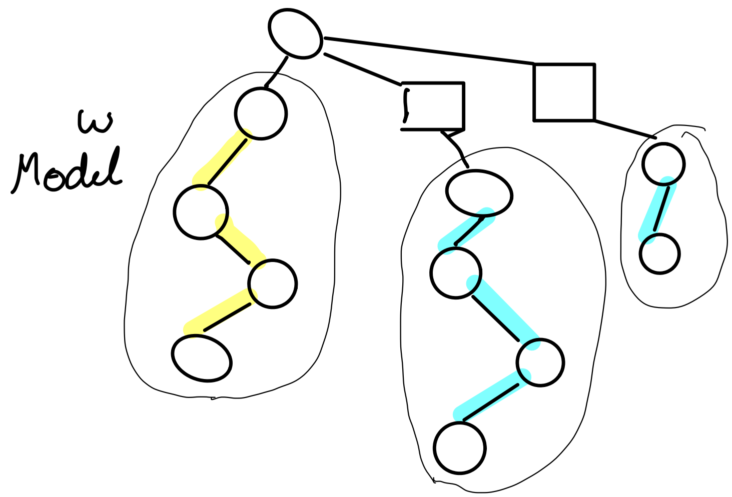
- Reduce tree width - helps exploration!
- Reduce tree depth - helps make planning/reasoning faster
- Generalize between different branches of the tree - improves learning

Options Intuition



- Package a whole sub-tree as an option ω
- Jumps are to the state *at the end* of the sub-tree
- Primitive actions are a special case (one-step tree)
- Two components: (sub)policy and model

Both abstraction and generalization!



Procedural, Temporally Abstract 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

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

- *Option model* has two parts:
  1. *Expected reward* $r_\omega(s)$: the expected return during ω's execution from state s
  2. *Transition model* $P_\omega(s'|s)$: 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
- Programming languages: preconditions (initiation set) and postconditions
- Models of options represent *(probabilistic) post-conditions*
- "Jumpy" planning is better for temporal credit assignment, accurate value estimation

What type of planning?

- *Models that are compositional can be used to plan through value iteration*
- *Sequencing*

$$\mathbf{r}_{\omega_1\omega_2} = \mathbf{r}_{\omega_1} + P_{\omega_1}\mathbf{r}_{\omega_2}$$
$$P_{\omega_1\omega_2} = P_{\omega_1}P_{\omega_2}$$

Cf. Sutton et al, 1999, Sorg & Singh, 2011

- *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
- Model-predictive control (receding horizon planning) is also possible

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*

Illustration: Navigation

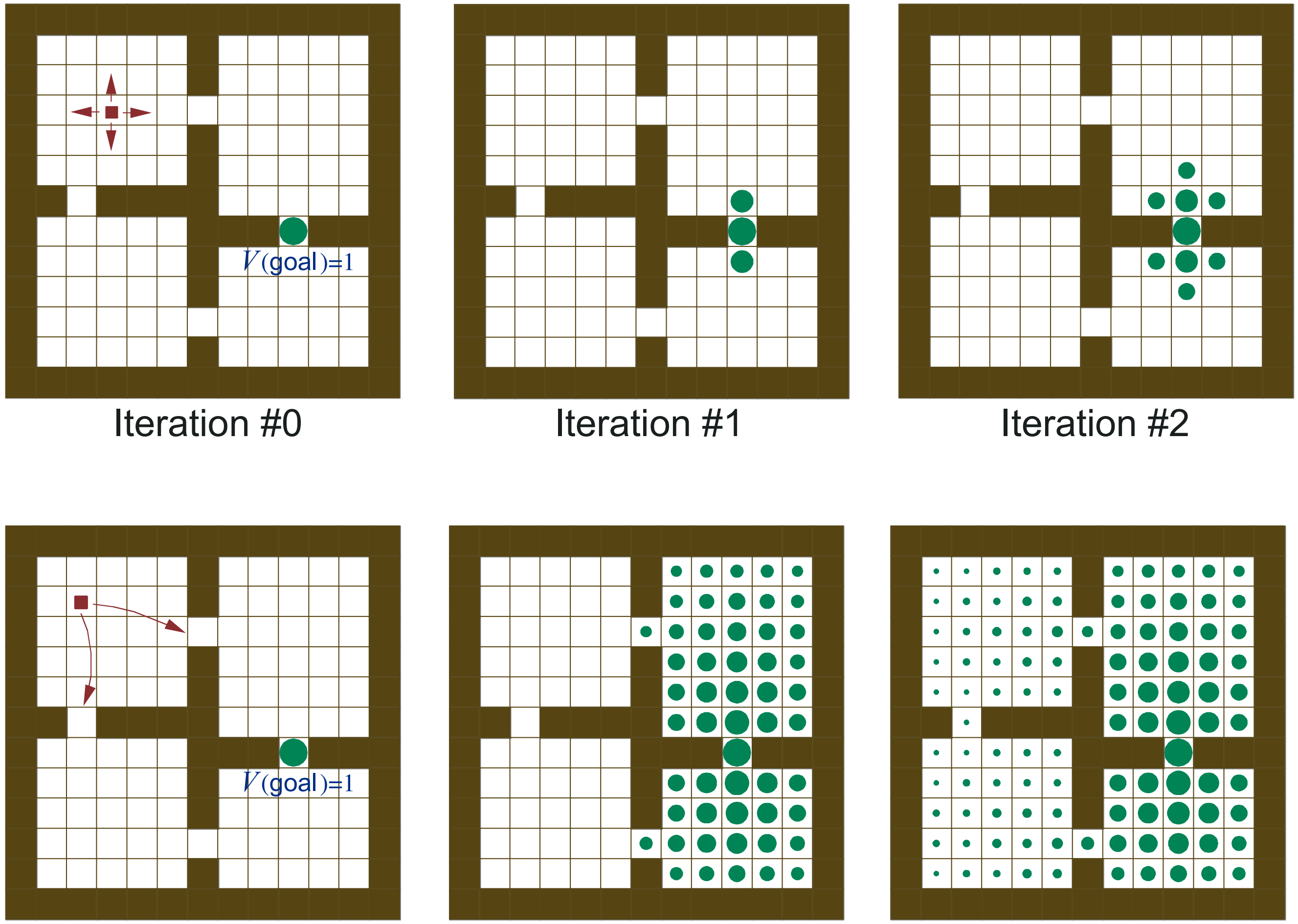
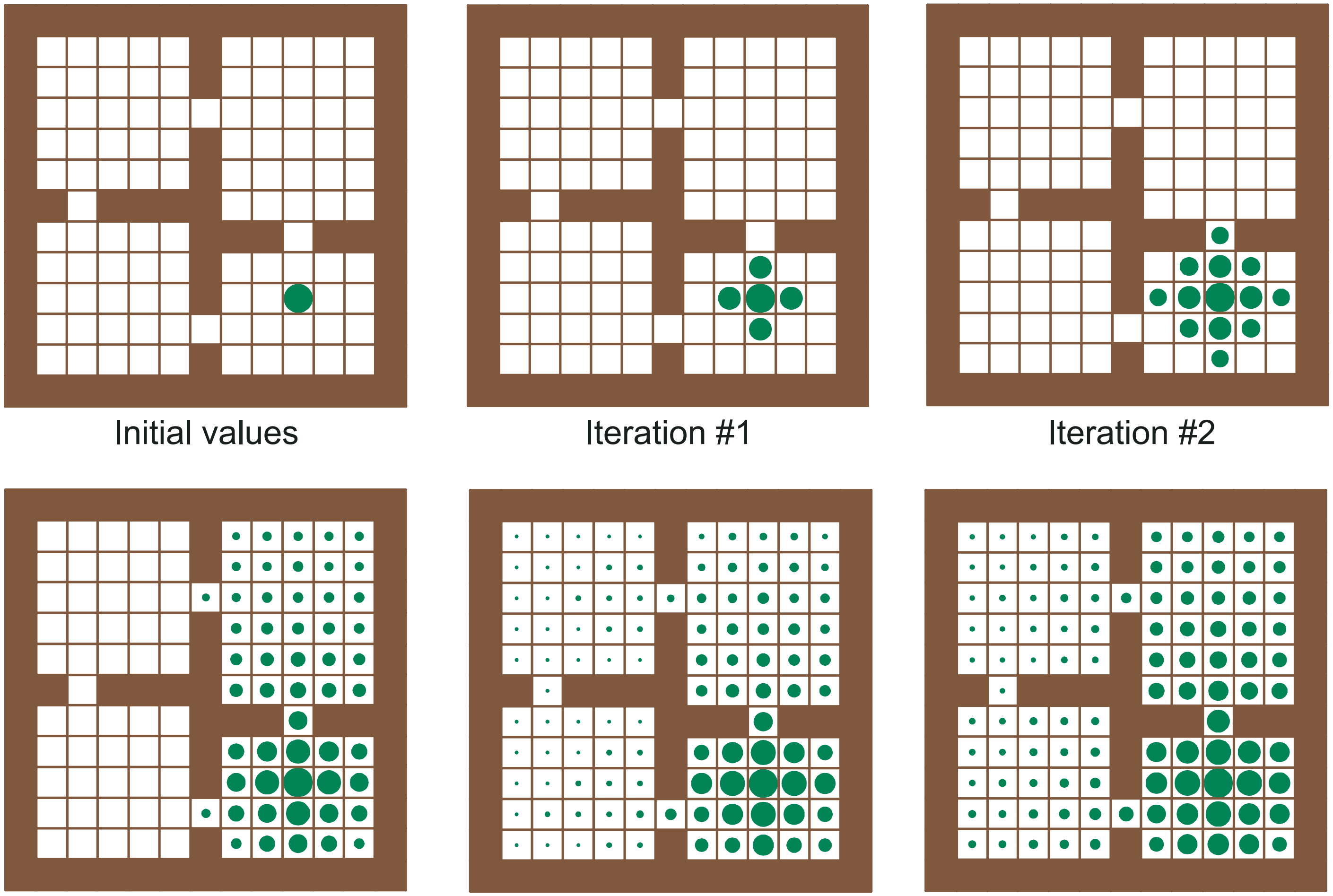
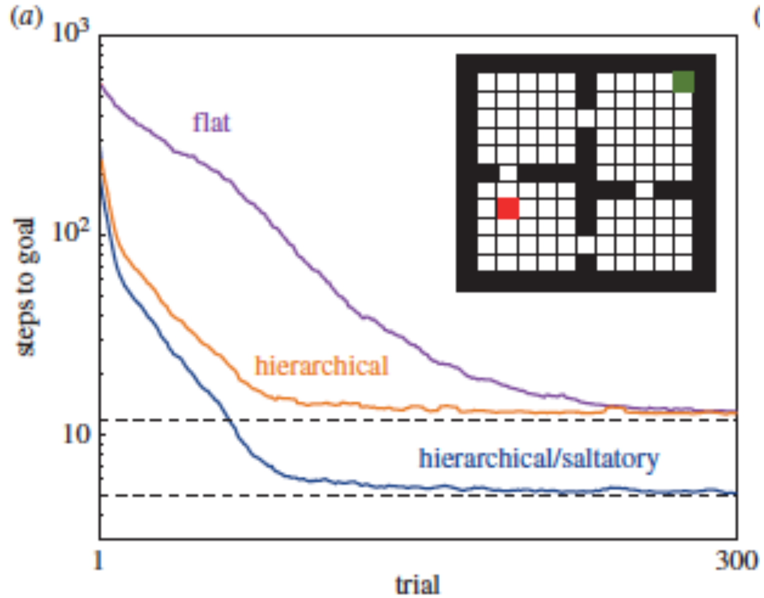


Illustration: Options and Primitives

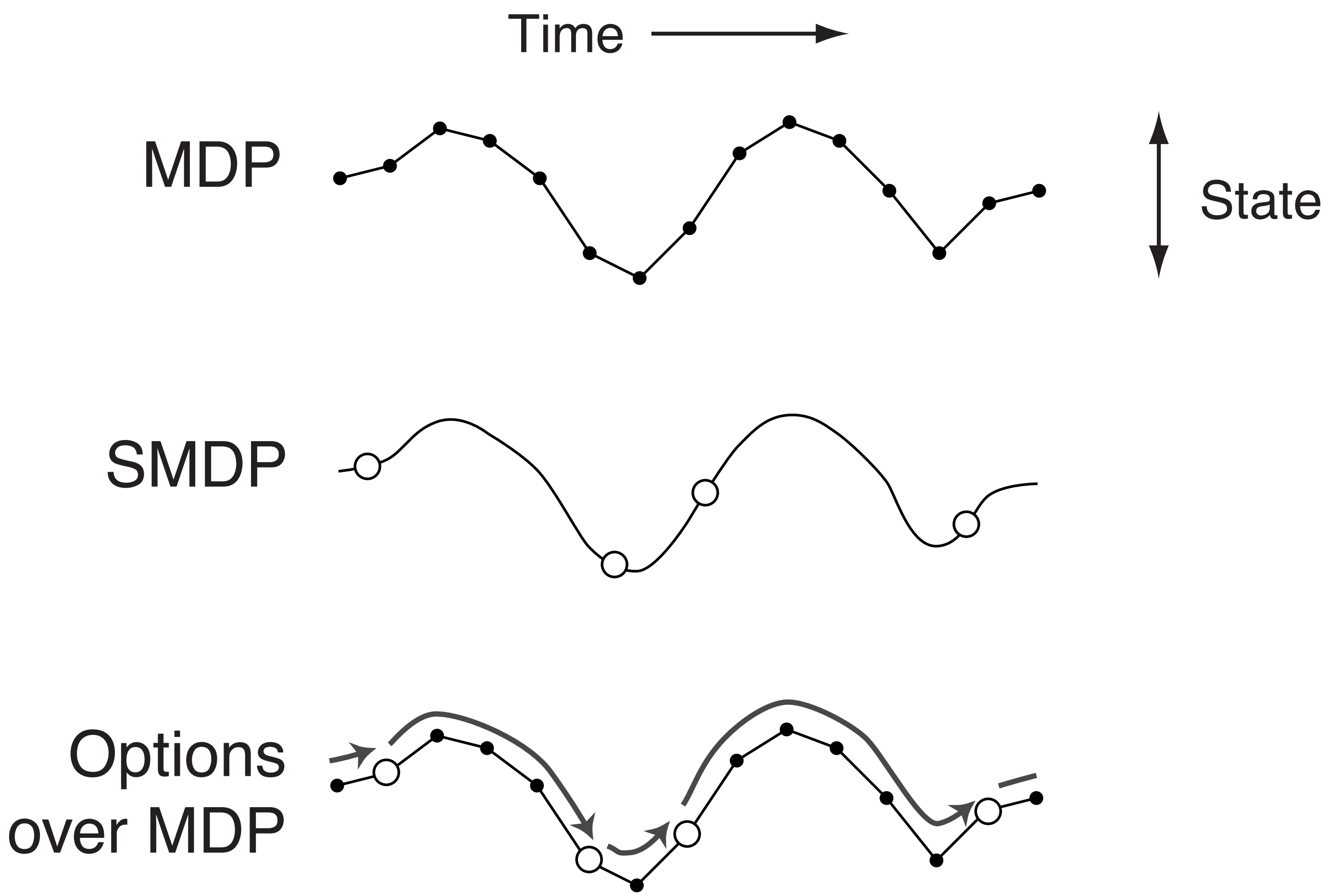


Initial values — Iteration #1 — Iteration #2 — Iteration #3 — Iteration #4 — Iteration #5

Benefits of options (cf Botvinick & Weinstein, 2014)



Decision-Making with Options



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

Option-value function

- The option-value function of a policy over options π_Ω is defined as:

$$q_{\pi_\Omega}(s,\omega) = \mathbf{E}_{\pi_\Omega}\Big[R_{t+1} + \gamma\beta_\omega(S_{t+1})q_{\pi_\Omega}(S_{t+1},\omega_{t+1}))$$
$$+ \gamma((1-\beta_\omega(S_{t+1}))q_{\pi_\Omega}(S_{t+1},\omega)|S_t = s\Big]$$

- One can use eg Q-learning, actor-critic,... to learn this!
- Note that if we learn/plan in an SMDP, the *contraction factor will be lower than γ*
- So fixing a set of options may allow solving the problem faster, but maybe in a slightly sub-optimal way
- Intuitively, models are more self-contained than option-value functions

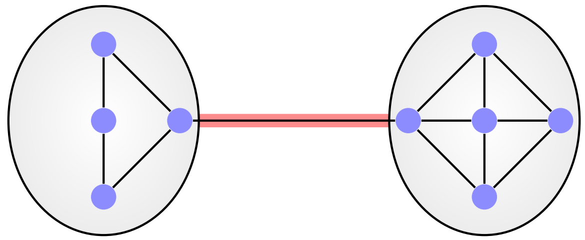
Advantages

- 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, Q_Ω
- *But we can also use the underlying MDP structure to help in learning the options*

How Should Options Be Created?

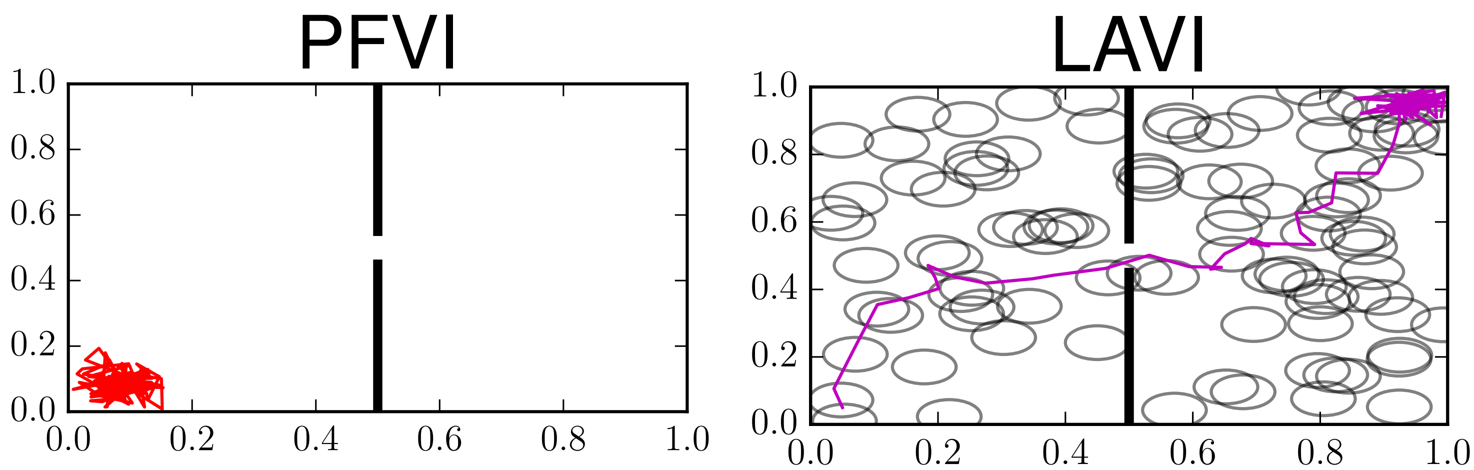
- Options can be given by a system designer (eg robotics)
- If subgoals / secondary reward structure is given, the option policy can be obtained, by solving a smaller planning or learning problem (cf. Precup, 2000)
  - Eg. acquiring certain objects in a game
  - Eg. Intrinsic motivation
- *What is a 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*

Random Subgoals Also Help



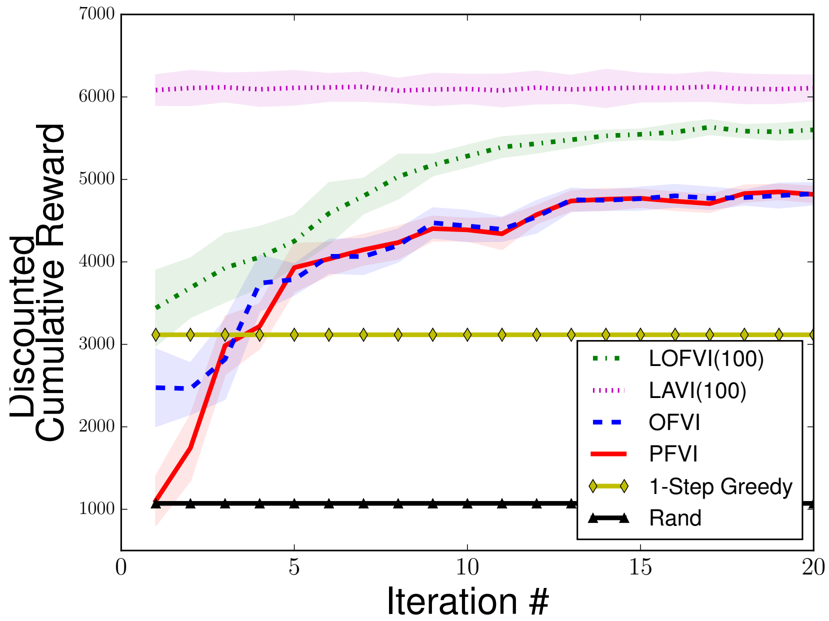
Cf. Mann, Mannor & Precup, 2015

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

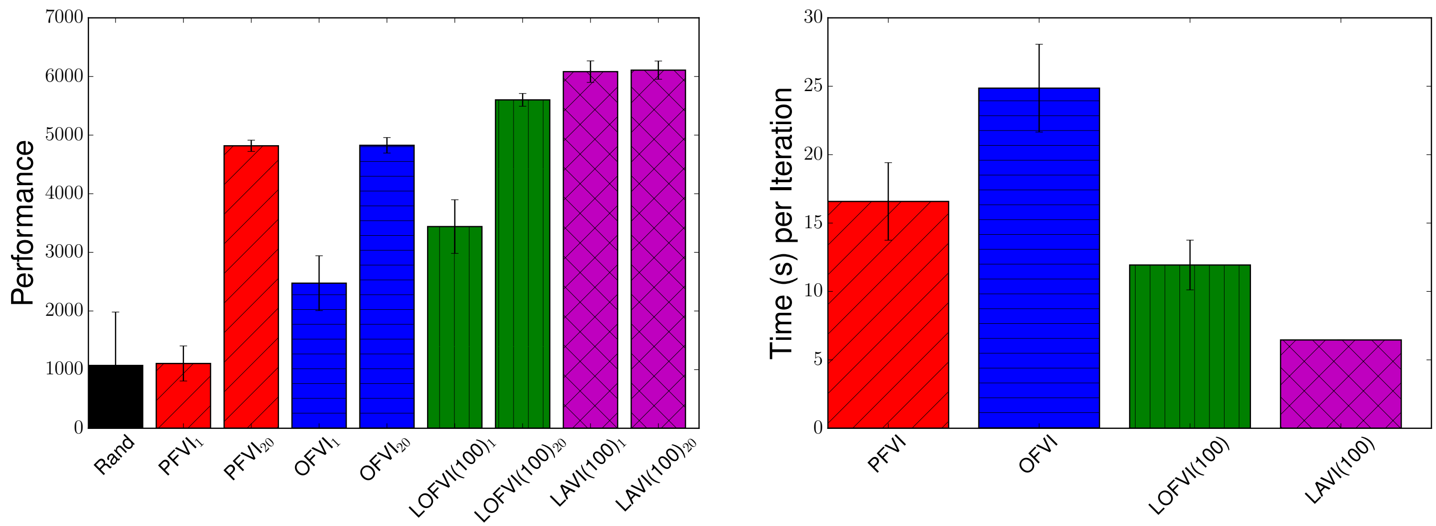
- Comparing a random policy and a 1-step greedy choice with using just primitives (PFVI) using primitives and hand-crafted options (OFVI), using "landmarks" (LOFVI) and using landmarks and only computing values for landmarks states (LAVI)



- *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

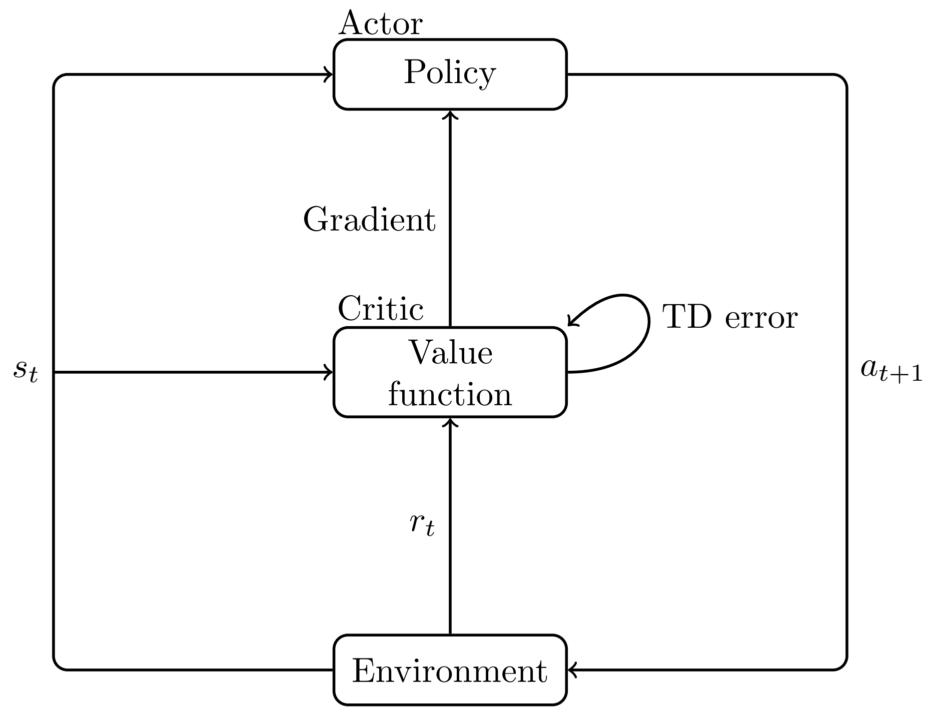


- Computing values only at landmark states yields a good policy almost immediately
- Handcrafted options are better than primitives in the beginning but slightly worse in the long run but *randomly generated landmarks are much better*

Option-Critic: Learn Options that Optimize Return

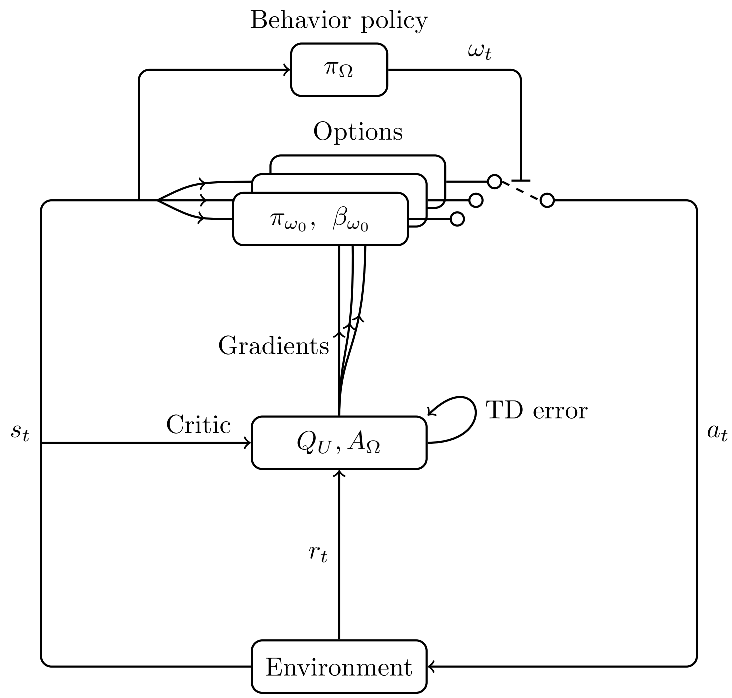
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or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

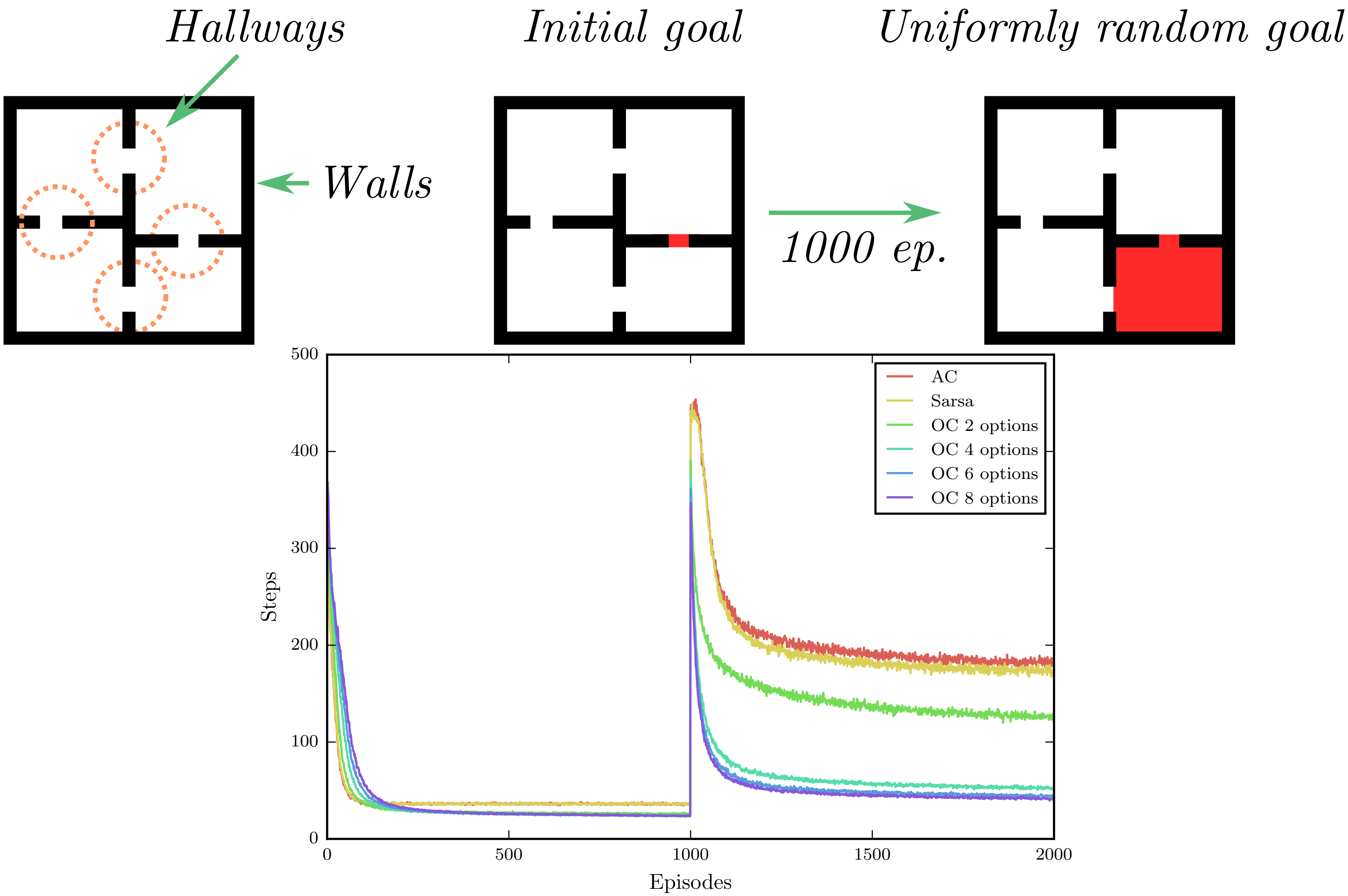
Option-Critic Architecture



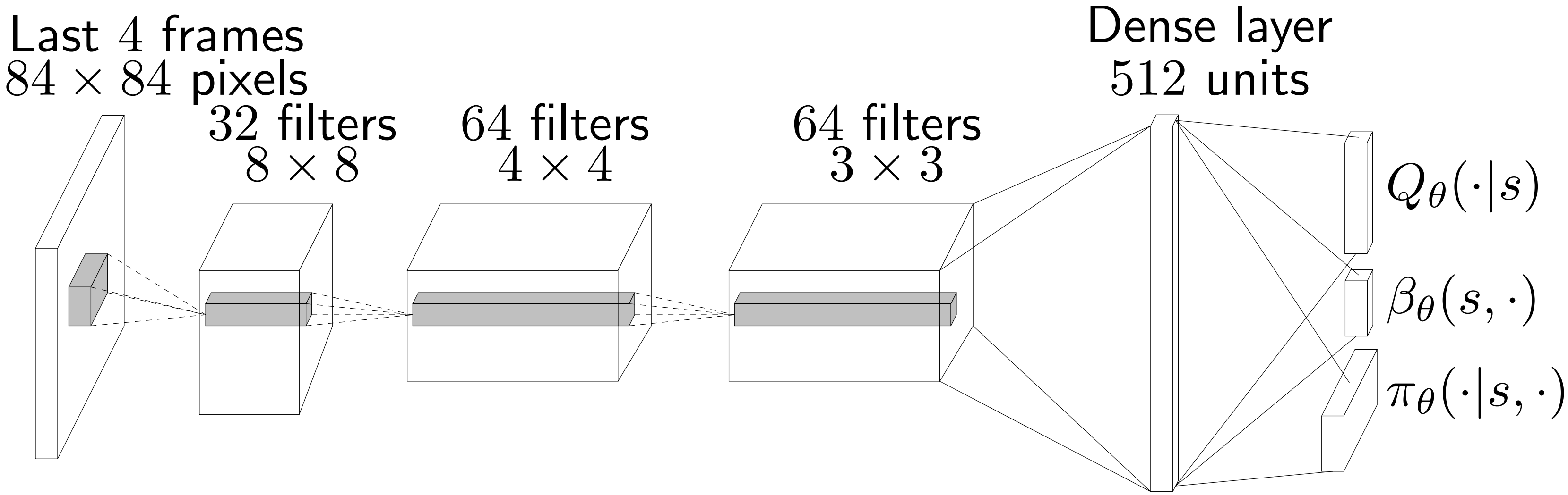
- Given a number of desired options, optimize internal policies and termination conditions using the cumulative reward signal

cf. Bacon et al, AAAI'2017

Results: Transfer in Rooms Domain

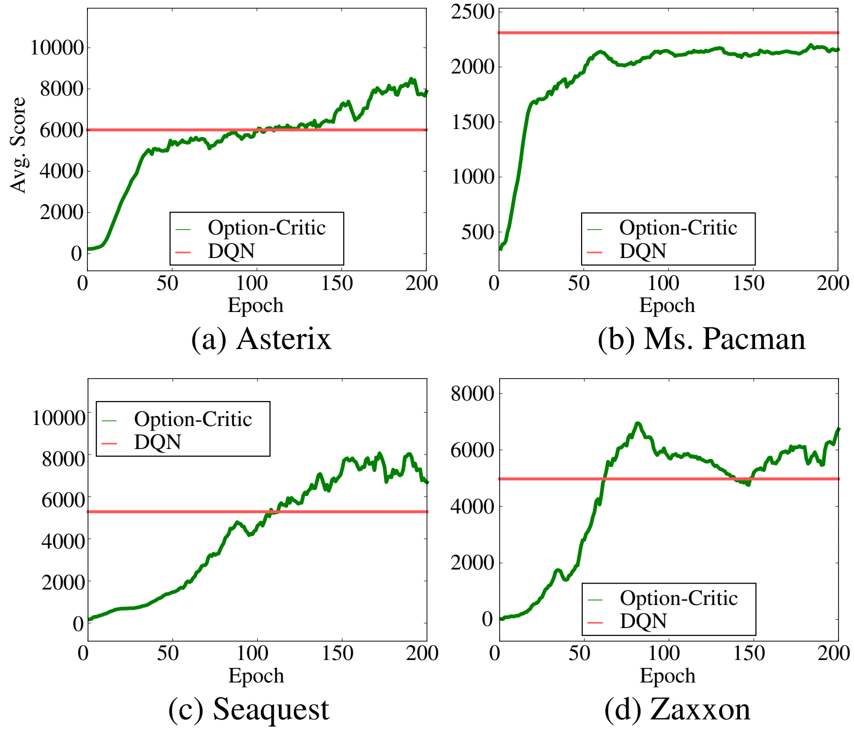


Option-Critic Architecture



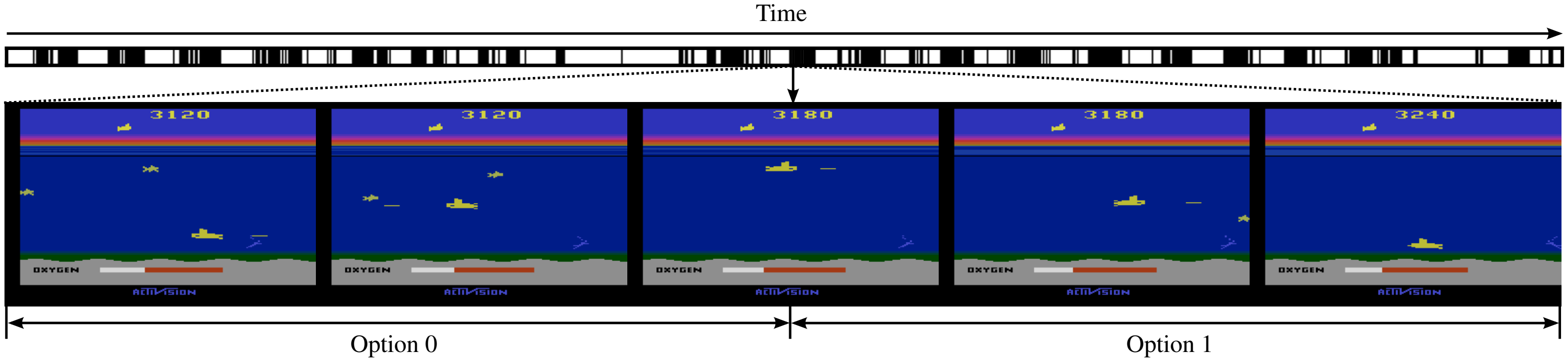
- Given a number of desired options, optimize internal policies and termination conditions using the reward signal
- DQN-style or advantage asynchronous option-critic (A2OC) (other choices possible)

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

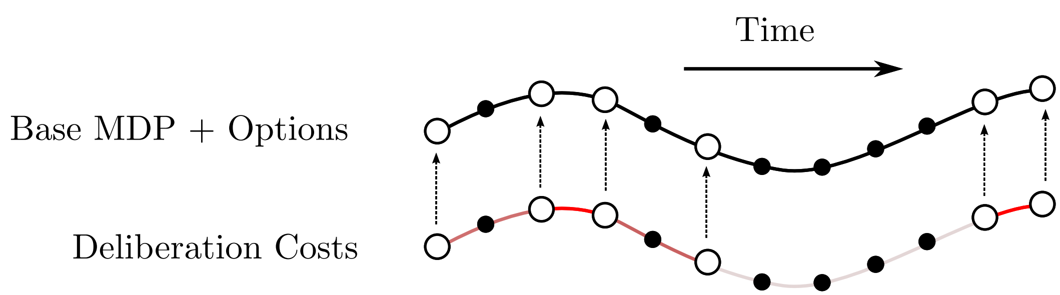
- Successful simultaneous learning of terminations and option policies
- But, as expected, *options shrink over time* unless additional regularization is imposed

  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

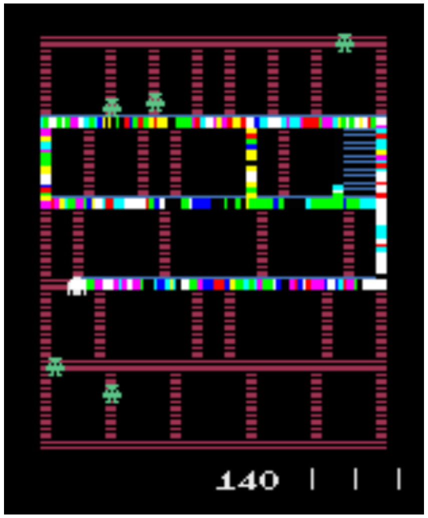
- Problem: optimizing return leads to option collapse (primitive actions are sufficient for optimal behaviour)
- Bounded rationality: reasoning about action choices is expensive (energy consumption and missed-opportunity cost)

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

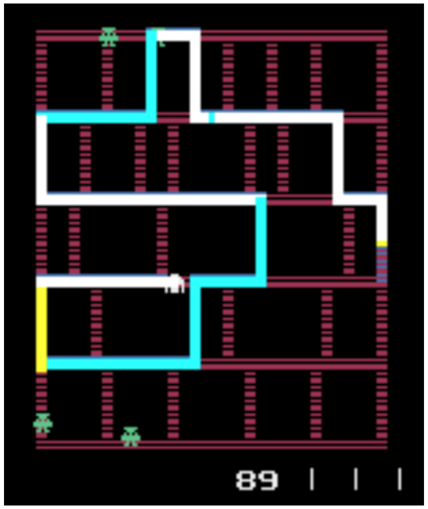


- Can be shown equivalent to requiring that *advantage exceeds a threshold* before switching

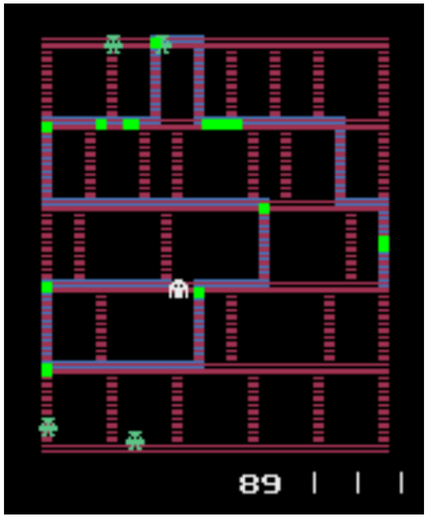
Illustration: Amidar



(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

Should All Option Components Optimize the Same Thing?

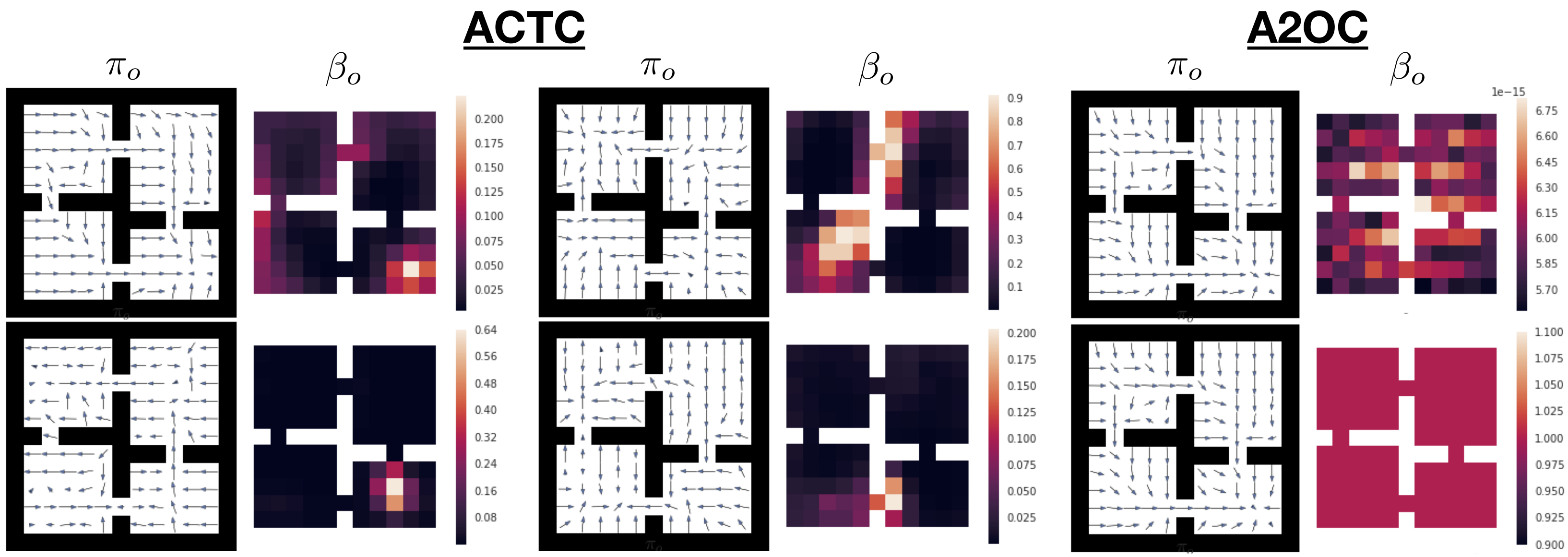
- Deliberation cost can be viewed as associated specifically with termination
- Rewards could be optimized mainly by the internal policy of the option
- Can we generalize this idea to other optimization criteria?

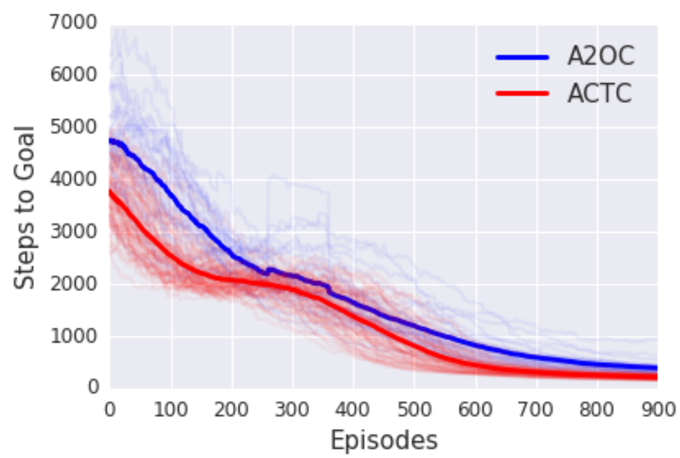
Termination-Critic

- *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





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