Beyond Model-Free Reinforcement Learning

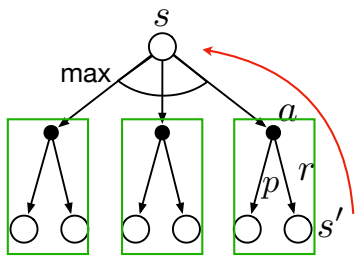
Why Going Beyond Model-Free RL?

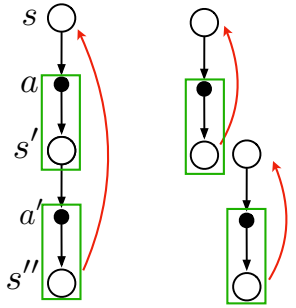
- Models provide "understanding" of the world (cf physics, causality...)
- Even if some parts of the problem change, others stay the same, which can help with faster learning

  Eg. Reward may change but the layout and dynamics of thee world may be thee same
- Models can be used to "dream" up new experiences, and use them to update the value / policy

Dynamic programming-style planning

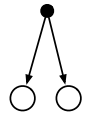
- Planning proceeds by using the model to look ahead from states, imagining something about the future
  - Each imagining from a state-action pair is called a lookahead
- Then, after one or more lookaheads, something computed at the leaves is passed back to the starting state to update its stored policy or value estimate
  - This is called a backup
- Backups continue forever
  ⇒ general planning





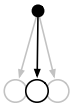
What should the model predict?

- Clearly we need the *reward*: easy problem, solved by regression
- What about the prediction of the *next state*?
  1. *Distribution model*: construct a distribution over next states / features



  2. *Sample model*: have the ability to generate sampled next states / features

  3. *Expectation model*: predict the expected next state / feature



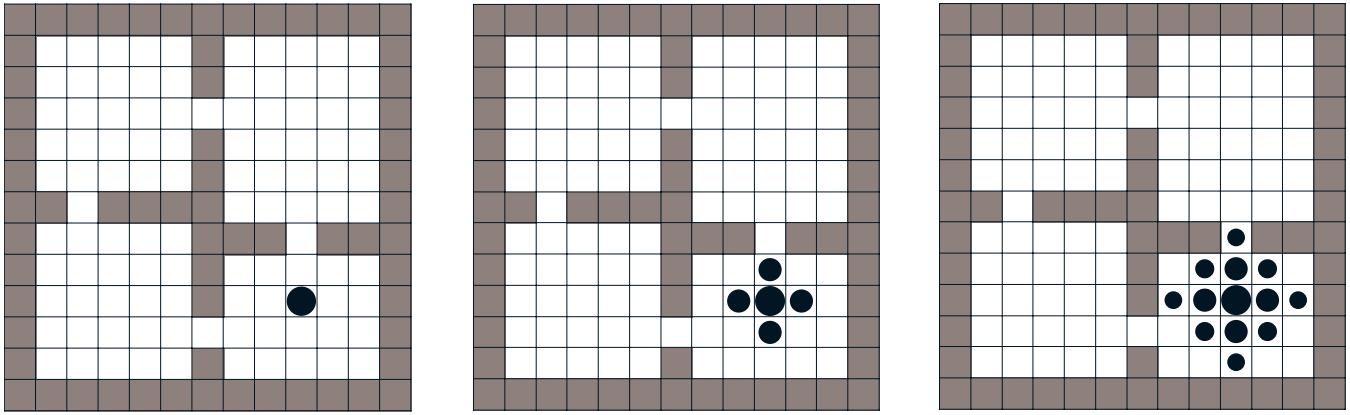
Using a Distribution Model: Value Iteration

- Approximates the optimal value function by doing repeated sweeps through the states:
  1. Start with some initial guess, e.g. V_0
  2. Repeat:

$$V_{k+1}(s) \leftarrow \max_{a \in A} \left(r_s^a + \gamma \sum_{s' \in S} p_{ss'}^a V_k(s') \right), \forall s \in S$$

  3. Stop when the maximum change between two iterations is smaller than a desired threshold (the values stop changing)
- One can prove that the error $||V_k - V^*||_\infty$ decreases as γ^k

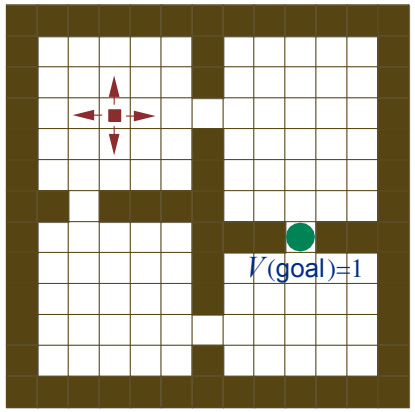
Illustration



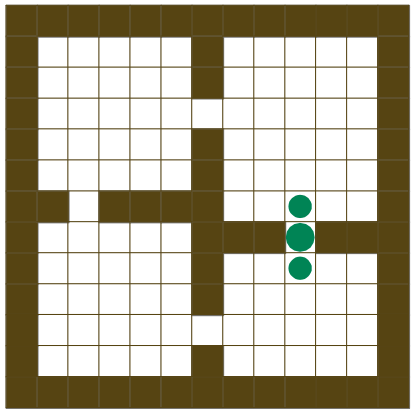
- Values propagate backwards from rewarding states
- May take a long time if rewards are sparse, unless we are smart about sampling states
- *Prioritized sweeping* (Moore & Atkeson, 1992 and much follow-up work): sample preferentially predecessors of states that had a large value change

Speeding up: Jumpy (temporally extended) models

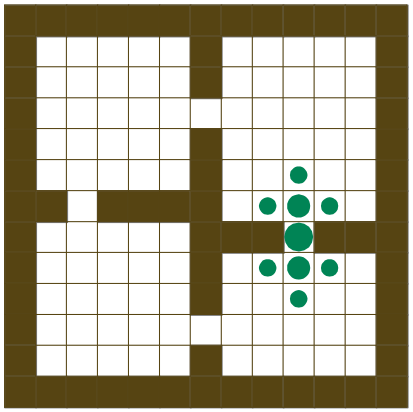
with cell-to-cell primitive actions



Iteration #0

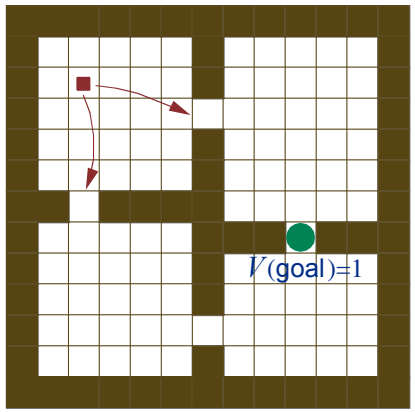


Iteration #1

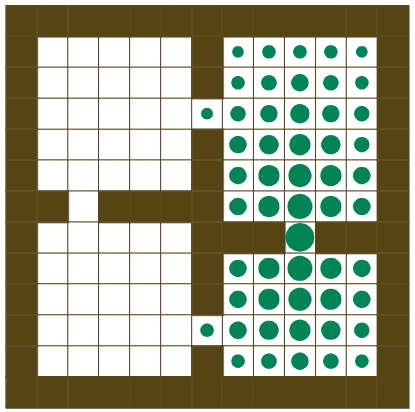


Iteration #2

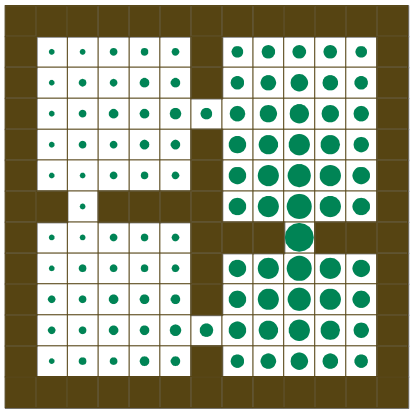
with room-to-room options



Iteration #0



Iteration #1



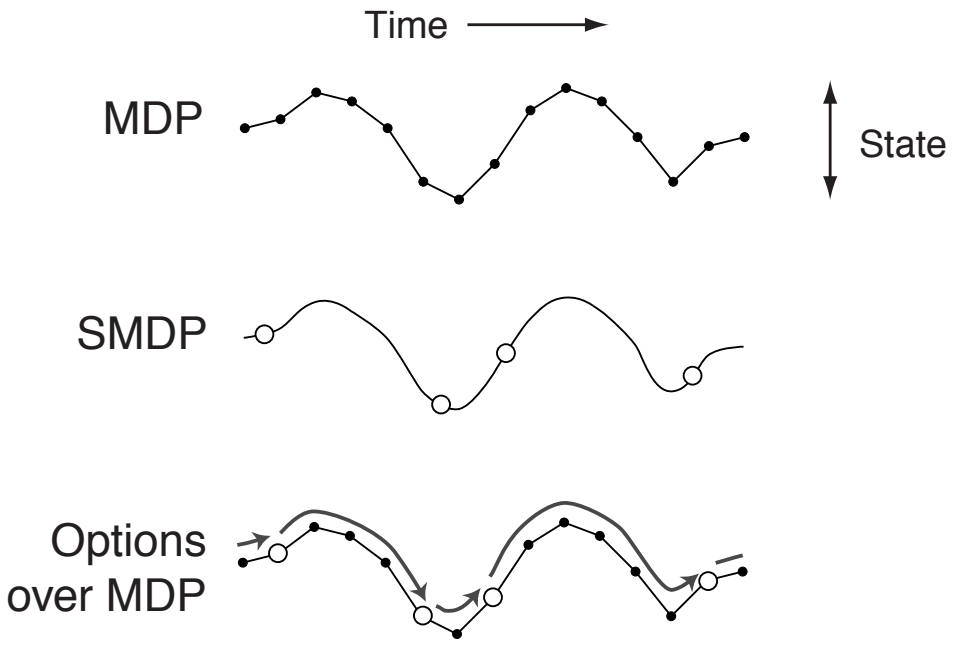
Iteration #2

Options framework

- An *option* o consists of 3 components
  - An *initiation set* of states $I_o \subseteq S$ (aka precondition)
  - A *policy* $\pi_o : S \times A \rightarrow [0, 1]$
    $\pi_o(s, a)$ is the probability of taking a in s when following the option
  - A termination condition $\beta_o : S \rightarrow [0, 1]$:
    $\beta(s)$ is the probability of terminating the option at s
- Eg., robot navigation: if there is no obstacle in front (I_o), go forward (π_o) until you get too close to another object (β_o)
- Other representations of the termination condition are possible (cf. Comanici and Precup, 2010)

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

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

Options framework

- *Option model* has two parts:
  - Expected reward $\mathbf{r}^o$: for every state, it gives the expected return during o's execution
  - Transition model $\mathbf{p}^o$: gives a sub-probability distribution over next states (reflecting the discount factor γ and the option duration)
- Models are *predictions* about the future, conditioned on the option being executed, i.e. *generalized value functions*
- 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*

What is needed to do planning with options

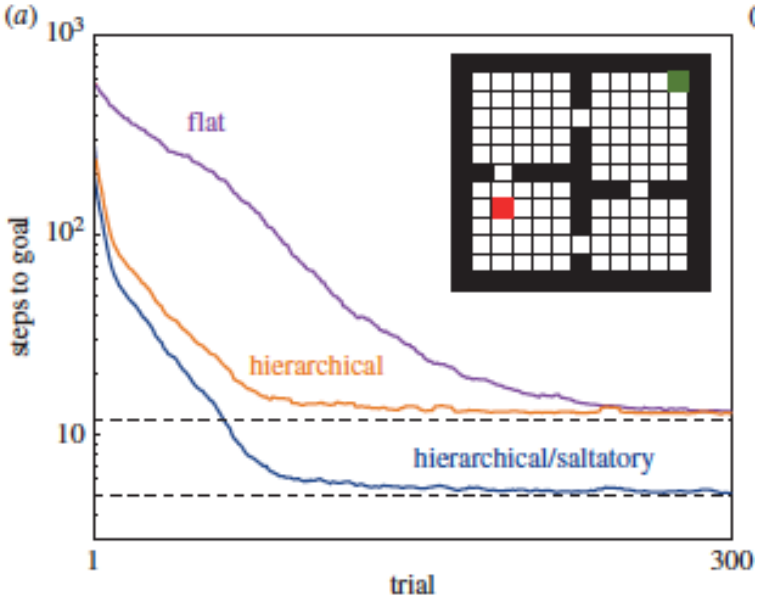
- *Compositionality*: putting models together into larger models

$$\begin{aligned}\mathbf{r}^{o_1 o_2} &= \mathbf{r}^{o_1} + \mathbf{p}^{o_1}\mathbf{r}^{o_2} \\ \mathbf{p}^{o_1 o_2} &= \mathbf{p}^{o_1}\mathbf{p}^{o_2}\end{aligned}$$

  If models are compositional, we can reason about the effect of sequential decisions
- Linear options (Sorg & Singh, 2010): using options with linear expectation models that respect these conditions
- 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

Benefits of Options Models

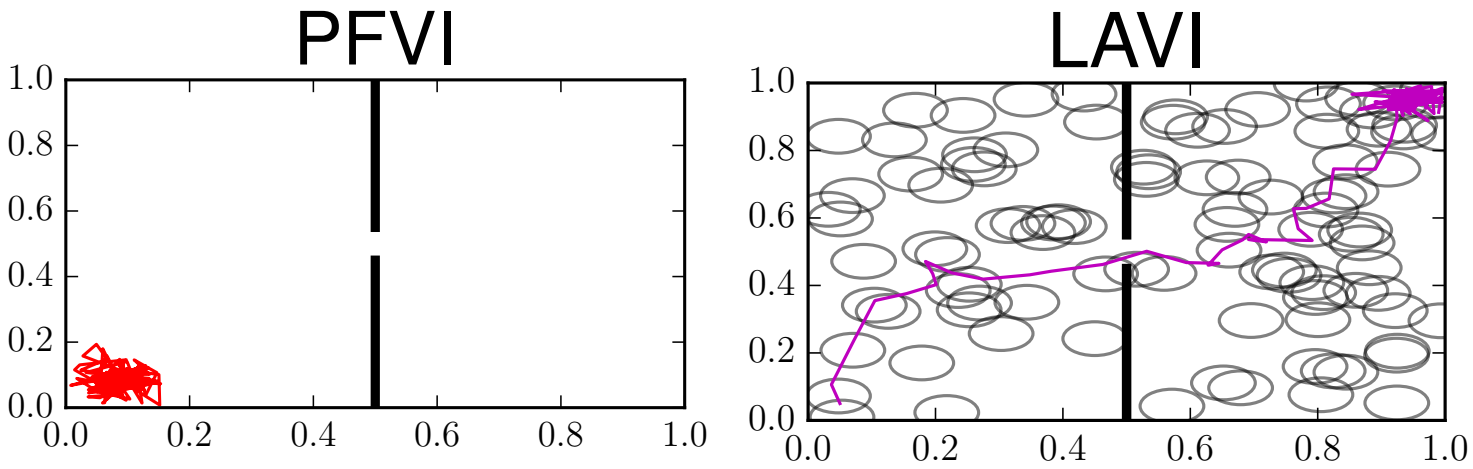
- Planning with option models provides benefits beyond using options to bias behavior (cf Botvinick & Weinstein, 2014)



How can we generate useful options / jumpy models?

- *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, JAIR'2015)
- 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



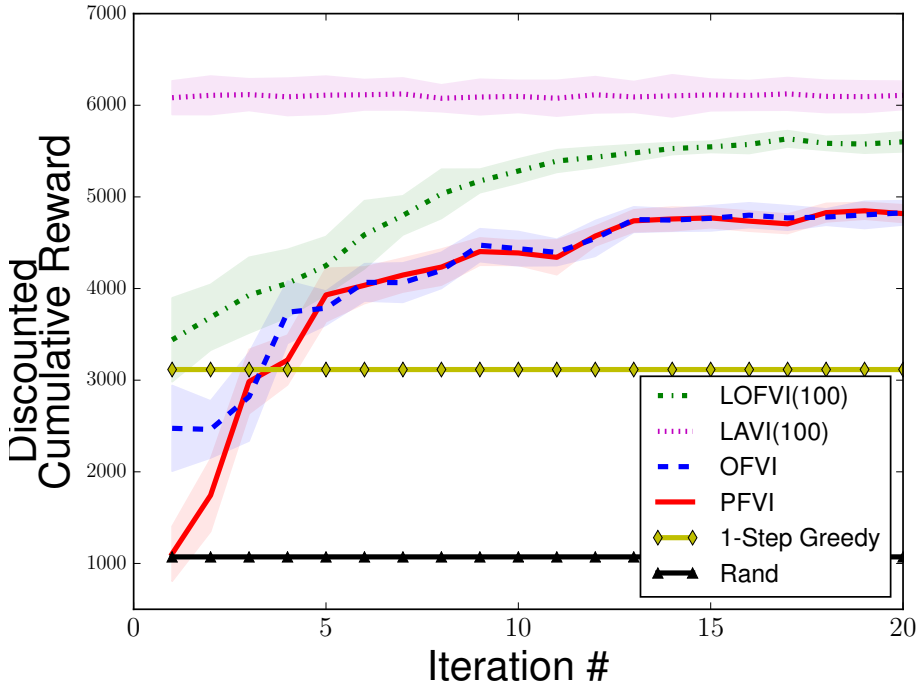
Landmark-based approximate value iteration gets a good solution much faster!

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

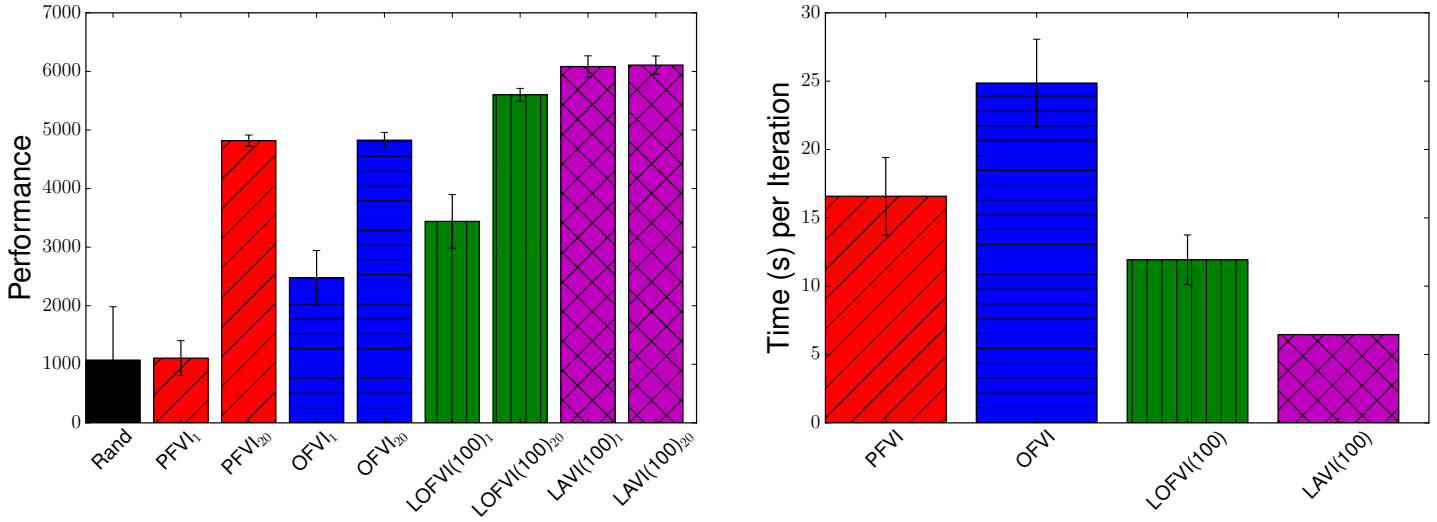
- Comparing a random policy and a 1-step greedy choice with using just primitives (PFVI) using primitives and hand-crafted options (OFVI), using "landmarks" (LOFVI) and using landmarks and only computing values for landmarks states (LAVI)



- *Randomly generated landmarks perform much better*

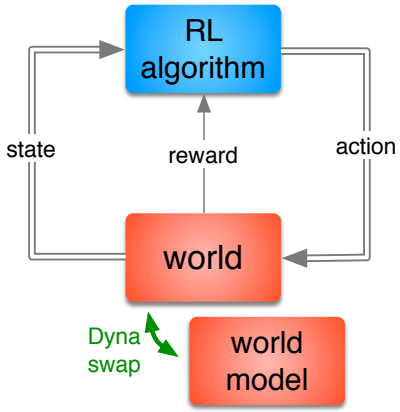
Performance and time evaluation

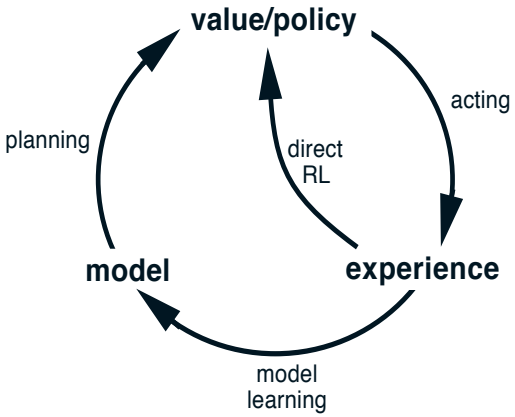
- Performance of initial and final policy (left) and running time (right) averaged offer 20 independent runs



- 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*

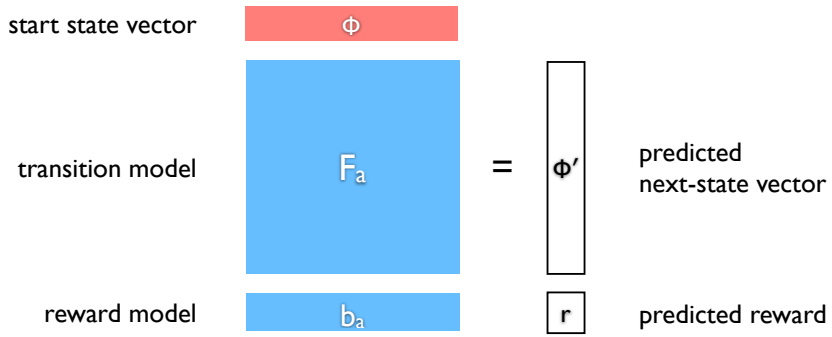
Dyna (Sutton, 1990)





- Learn a sampling model of the world
- Same RL algorithm used for both learning and planning (TD(0), Q-learning...)
- Planning, learning and execution are all done simultaneously and asynchronously

Special case: Linear Expectation Models



- states are represented by feature vectors

$$s \longrightarrow \phi_s \qquad s_t \longrightarrow \phi_t \qquad \in \Re^n$$

- the model is a set of matrix-vector pairs

$$M = \{F_a, b_a\}_{a \in Actions}$$

expected transition matrix

$$E\{\phi_{t+1} | \phi_t = \phi, a_t = a\} = F_a \phi$$

$$E\{r_{t+1} | \phi_t = \phi, a_t = a\} = b_a^\top \phi$$

expected reward vector

Linear Dyna

- Use a linear model and a linear parametrization of the value function
- Note that the features ϕ could be non-linear (eg coming from a convnet) but they must be fixed
- In this case, value iteration using an expectation model is the same as using a full model

Generalizing Expectation Models: 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)

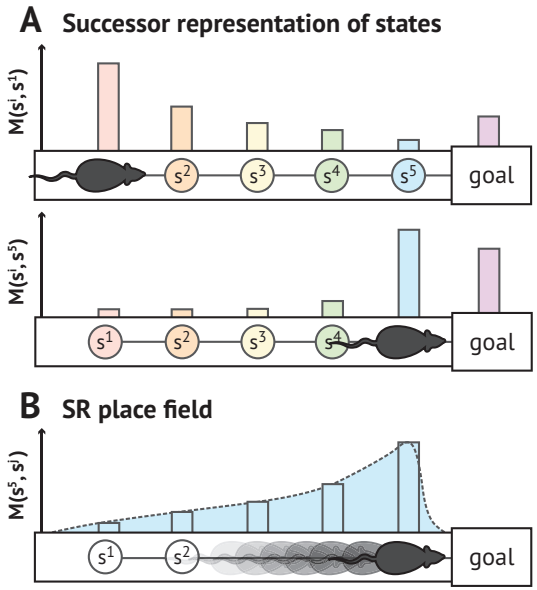
Expectation Models of Options are GVFs

- 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 $\gamma(1 - \beta_\omega)$
- Expected option transition model can be similarly written as a GVF

Successor Representations



- Successor representation (Dayan, 1992): special case when ϕ is a 1-hot encoding of states
- Stachenfeld et al (Nat. neurosci, 2017): Successor representation is linked to place cells in hippocampus

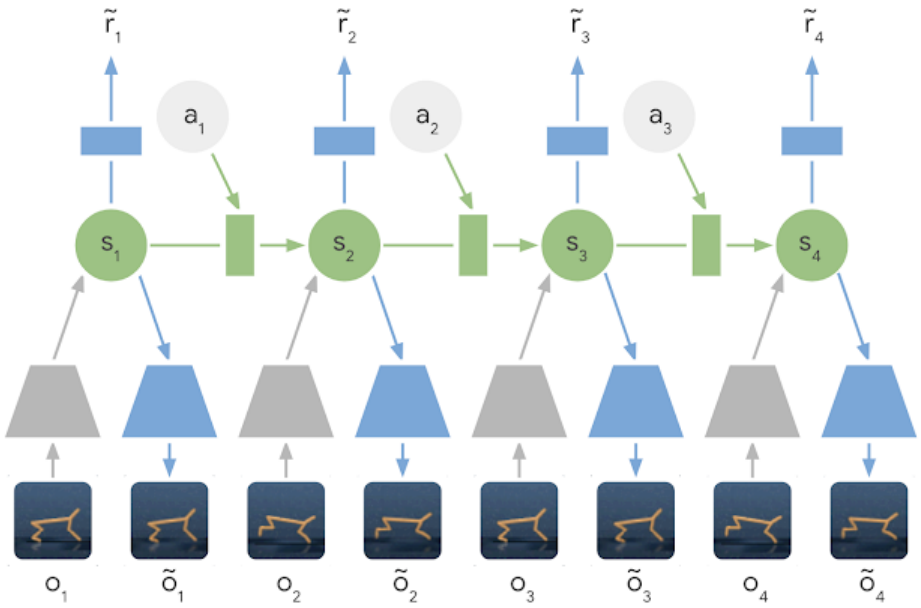
Successor states and successor features are GVFs

- *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:

$$v_{\pi,\mathbf{w}^T c,\gamma}(s) = \mathbf{w}^T v_{\pi,c,\gamma}(s)$$

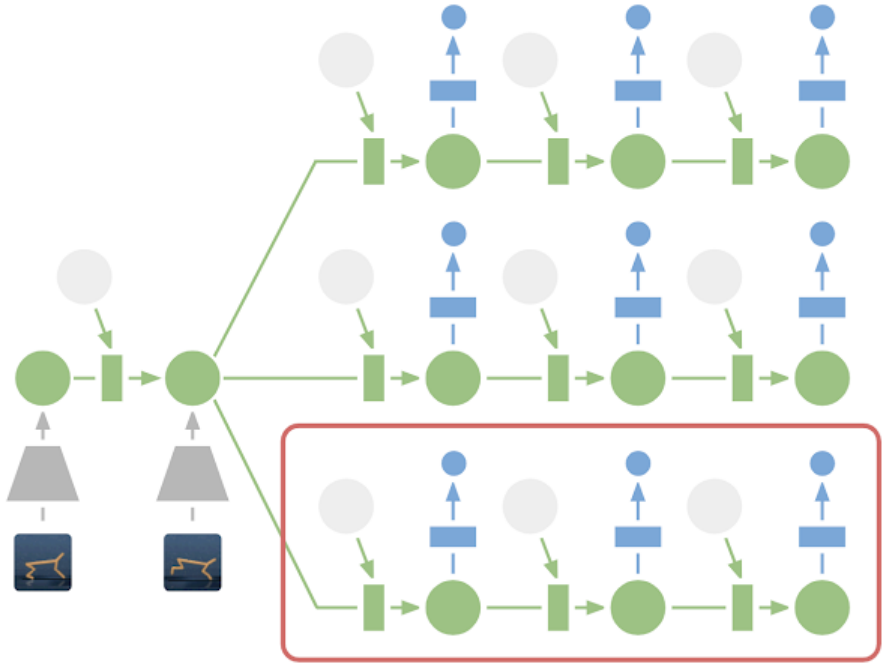
which leads to one-shot computation of new GVFs

Using Approximate Models: PlaNet (Hafner et al, 2019)



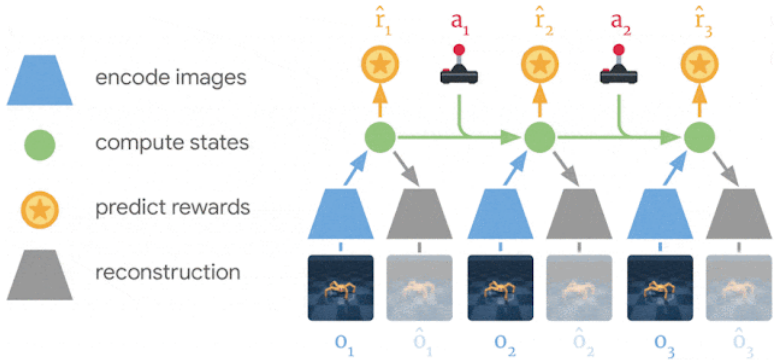
- Building on world models work by Ha and Schmidhuber (2017)
- Learn a model that tries to fit the observations (using a loss function)

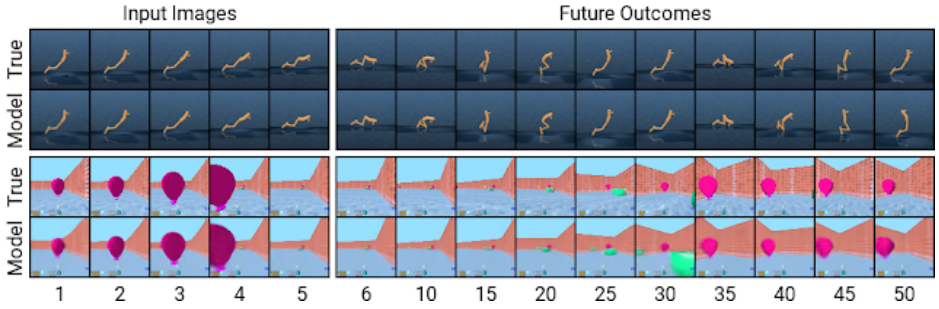
PlaNet Planning Process



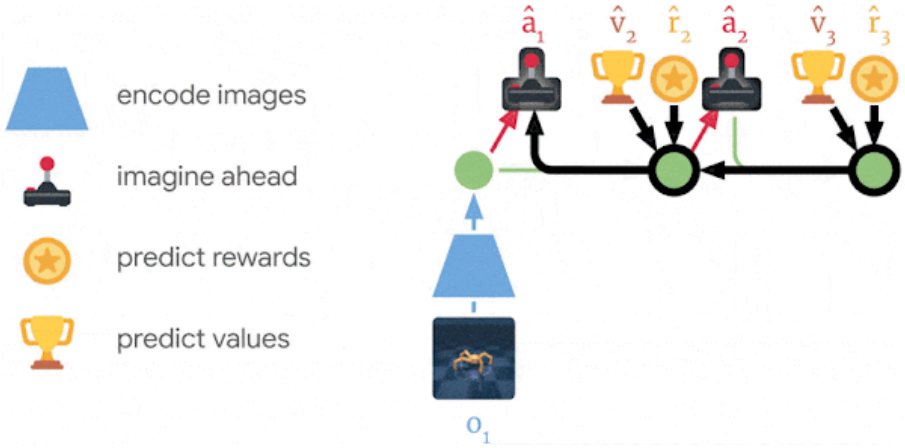
- At planning time, only abstract states are generated

Dreamer (Hafner et al, 2020)

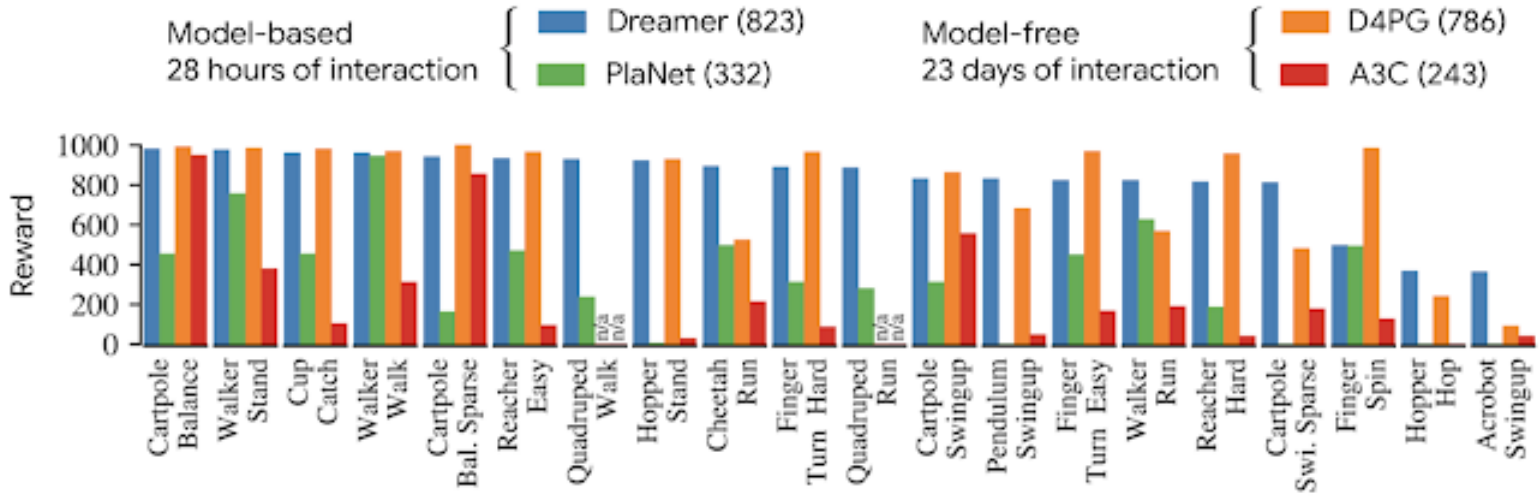




Value Propagation in Dreamer

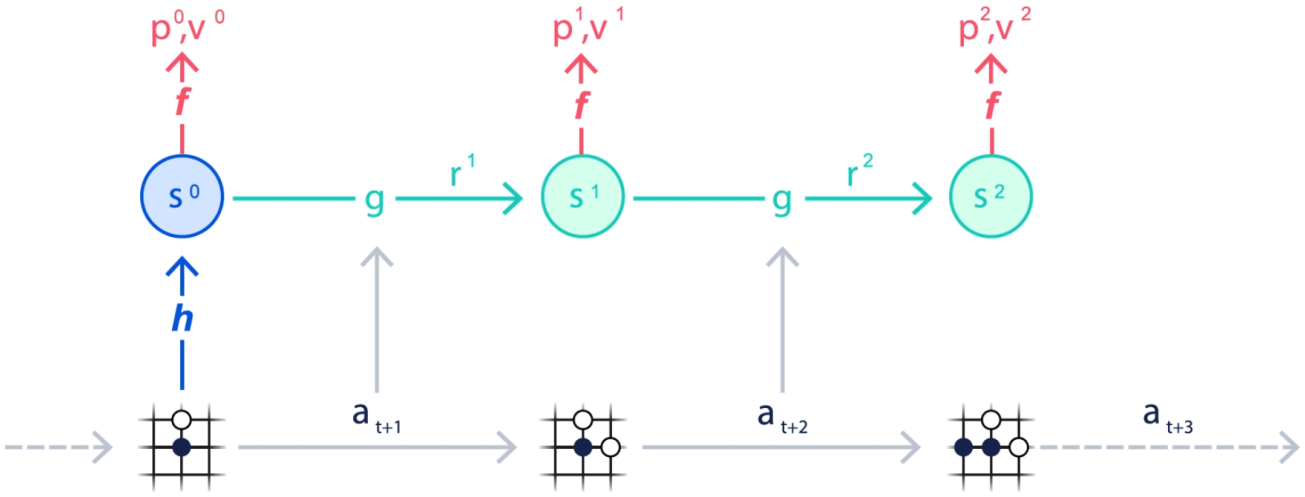


Dreamer and Planet Results



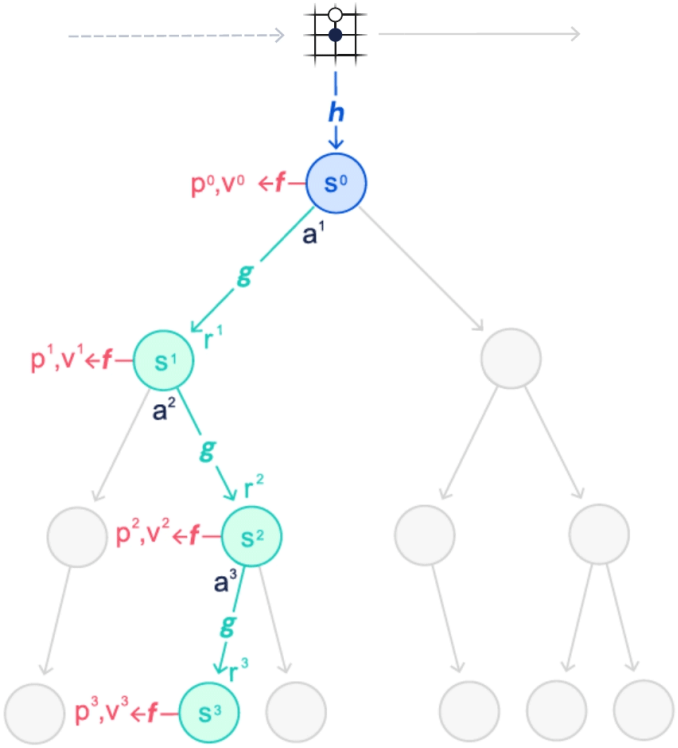
Model-based methods achieve comparable results to model-free with much less data

Using Approximate Models: MuZero (Schrittwieser et al, Nature, 2020)



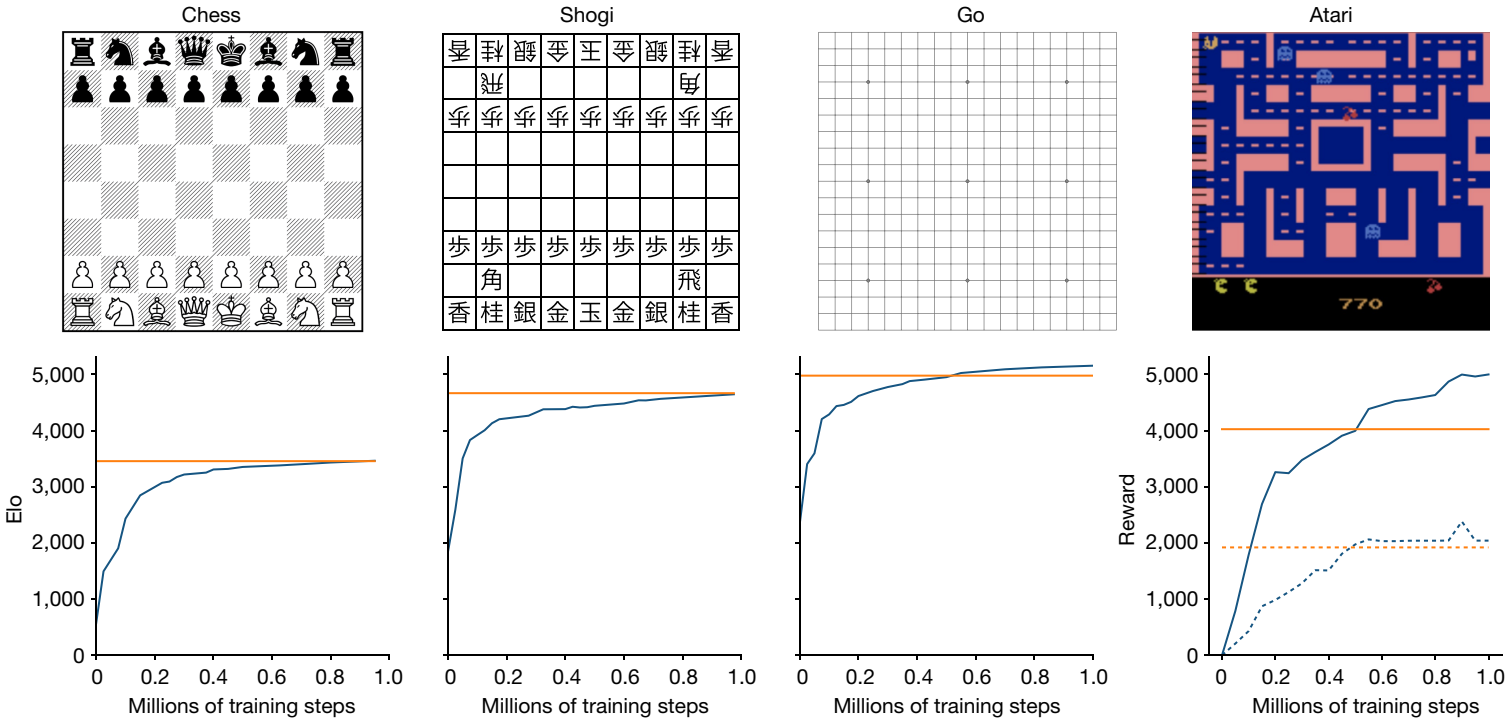
- Rather than predict the entire environment, make sure predictions are accurate for values, rewards and actions
- Values are trained with observed returns, actions to mimic the policy obtained through search

Execution in MuZero



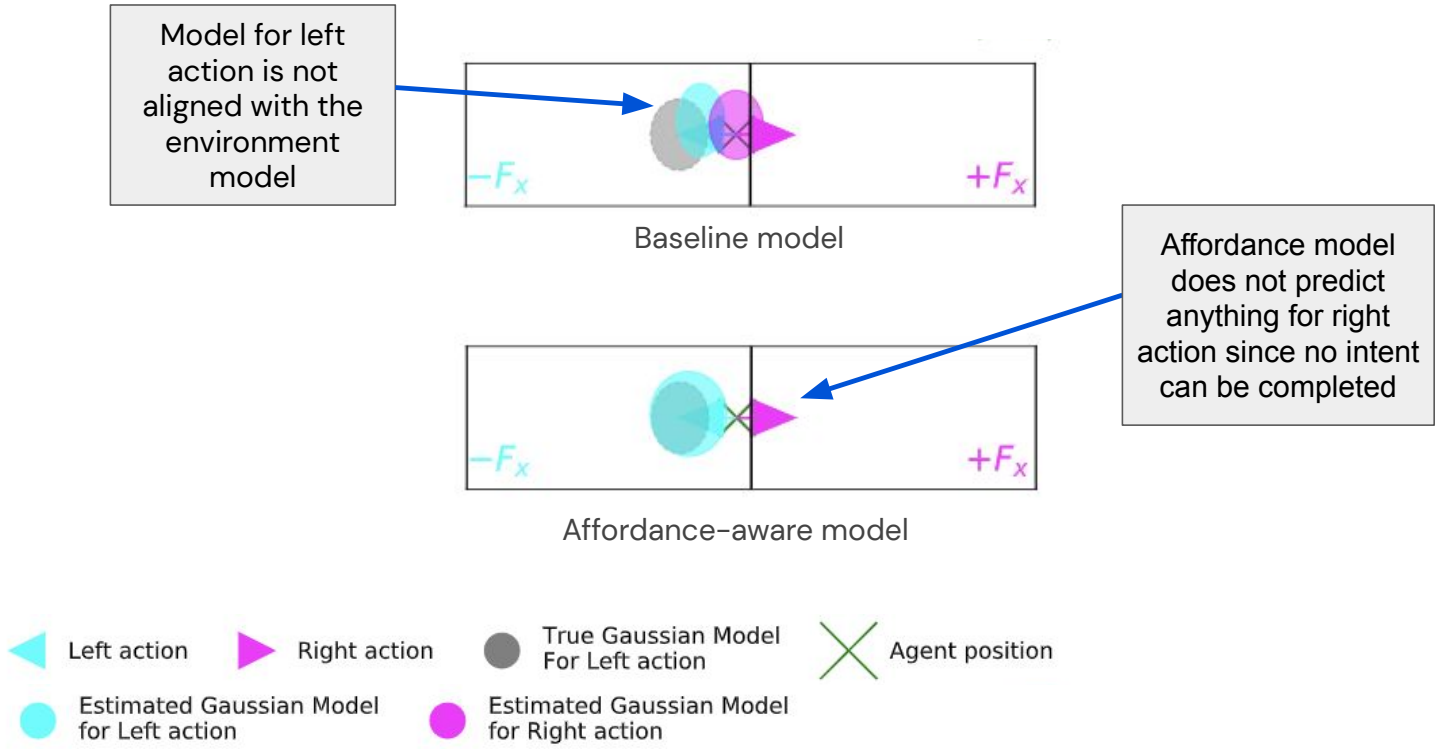
- Model is rolled forward in Monte Carlo Tree Search-style

MuZero Results



MuZero outperforms R2D2 (best model-free agent at the time)

Partial Model Learning (Khetarpal et al, ICML'2020)



- Talvitie & Singh, 2010-2018: build models only in certain parts of the environment, and predict only certain aspects
- GVFs predict only certain aspects, so do abstract models
- This work: predict only in certain parts of the environment

Conclusions

- Model-based RL is conceptually appealing as an approach to building general AI agents
- Some model-based RL agents are starting to deliver on the promise of comparable or better performance than model-free, with better sample complexity
- Theoretical results on model-based vs model-free are still largely inconclusive
- Intuitively, abstract, jumpy and partial models and planning should be even better
- Lots of open questions in this area of research!
- Using models for continual learning is an important open avenue