Gaps in the Foundations of Planning with Approximation

Rich Sutton

with particular thanks to Joseph Modayil, Yi Wan, Abhishek Naik, M. Zaheer, Katya Kudashkina, Martha Steenstrup

Outline

- "Planning" is AI's way of achieving cognition, reason, thought
- In reinforcement learning, planning is naturally viewed as value iteration with a learned model
- I see 5 big challenges to extending value iteration to the goals of AI (while keeping it simple, general, scalable, and efficient)
    - For 4 of them—*average reward, partial observability, temporal abstraction, and function approximation*—the way forward seems clear
    - For the 5th challenge, *stochastic transitions with approximation*, the way forward remains unclear

Planning is any way of using a model of the world + computation (not experience) to improve a decision-making policy or value function

It’s common to view intelligence as having 2 parts

1. A fast, reactive part giving us (learned) reflexes and intuitions

2. A slower, deliberative part that makes better choices

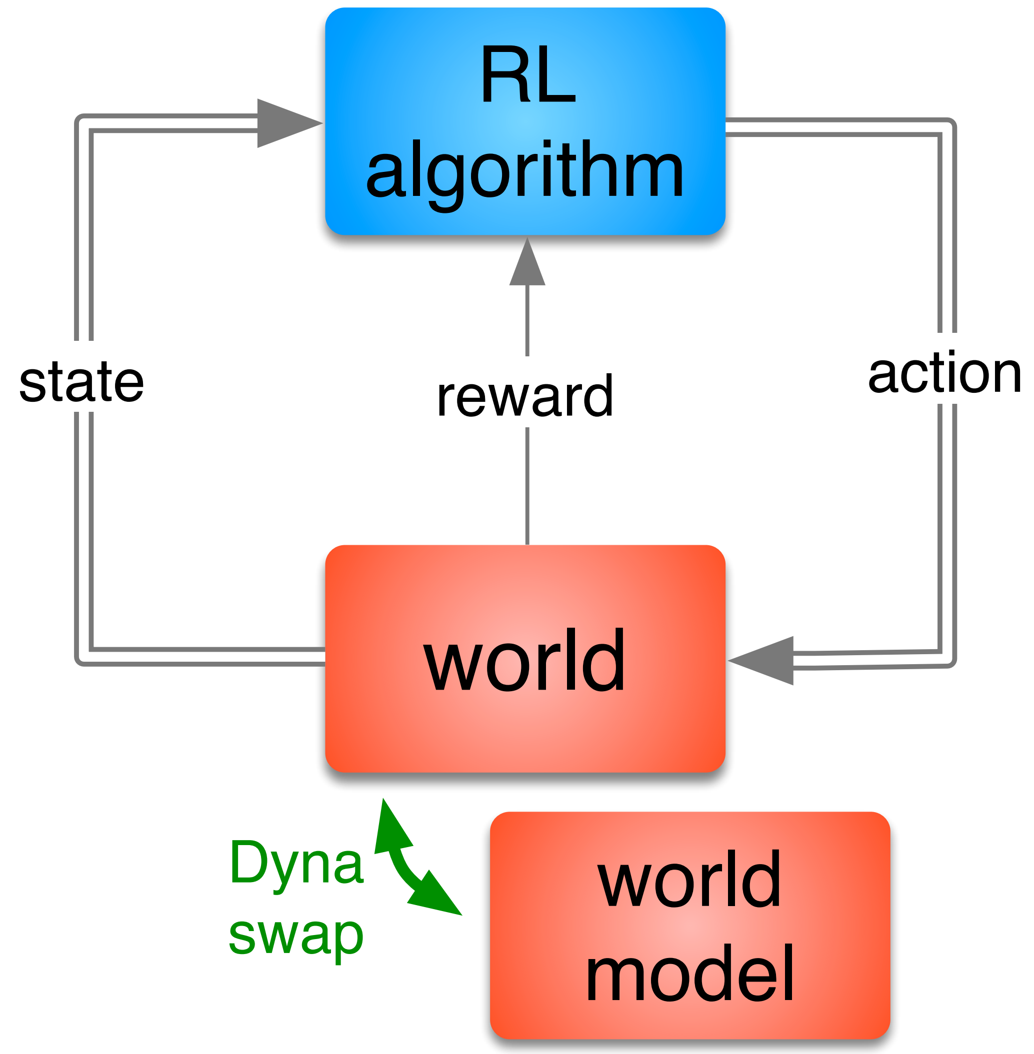
- I think of them as the *reactive foreground* and *deliberative background* of the mind

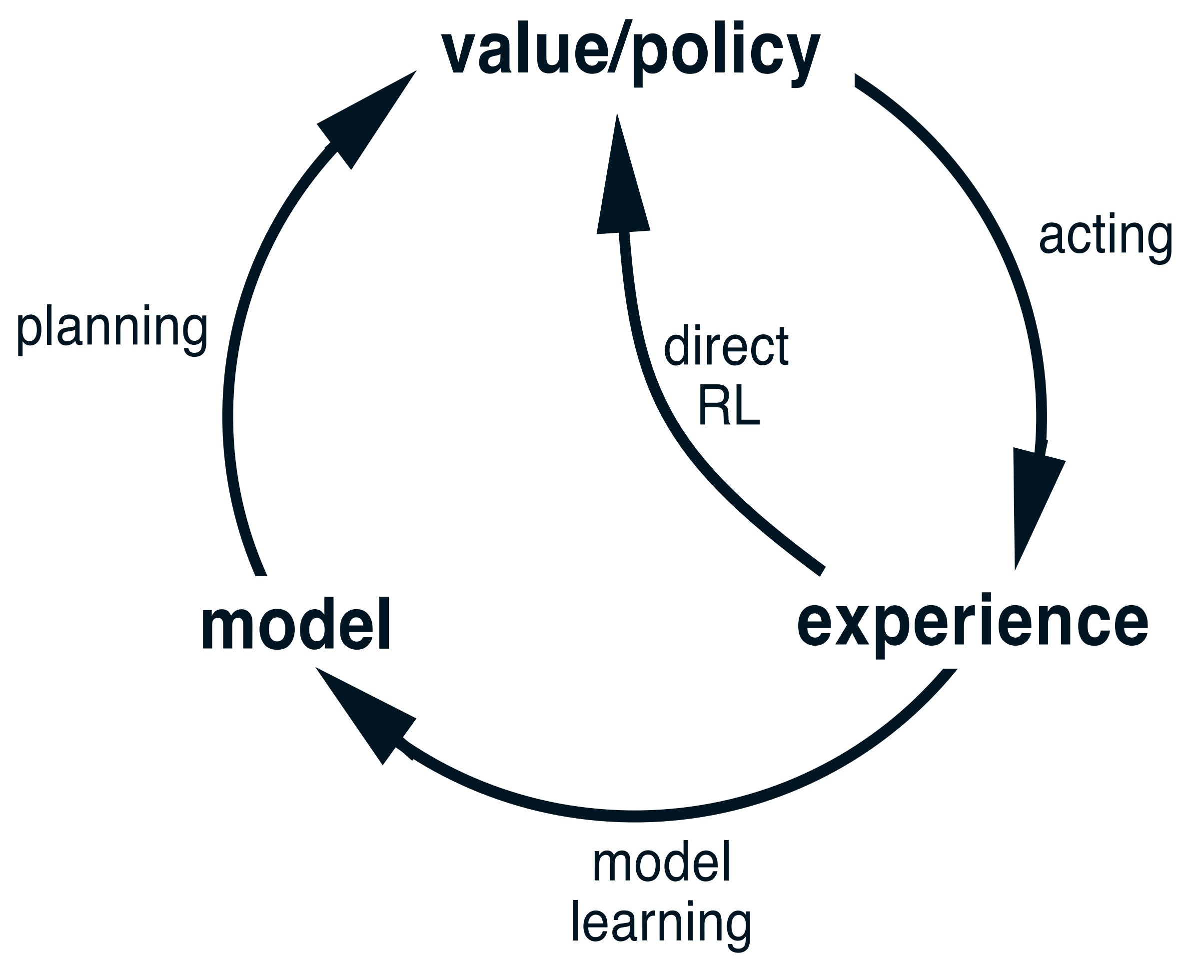
- In psychology, Daniel Kahneman calls them *System 1* and *System 2* in his NYT best-selling book *Thinking Fast and Slow* (2013)

- In robotics, there is a controller and a trajectory planner

Model-based Reinforcement Learning

- **Setting:** An agent interacts with a world in discrete time steps, emitting actions, receiving states and rewards
- Learning: The agent learns, from experience:
    - a reactive policy, mapping states to actions
    - a value function, mapping states to predictions of future reward
    - a model, mapping states and actions to expected rewards and (distributions over) next states
- Planning: The agent uses model and computation (and no new data) to improve the reactive policy and value function

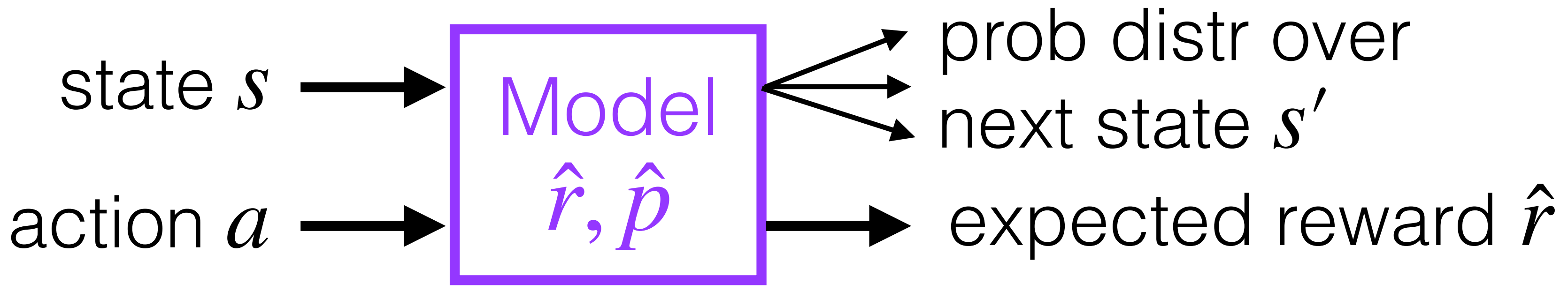




Dyna, Sutton 1990

Planning by value iteration

In which planning is using a model of the world to compute state values (estimates of future total reward from each state)



- All the time, when you have time, select a state $s \in \mathcal{S}$ (search control) and perform a *backup* at s:

$$V(s) \leftarrow \max_{a} \Big[\hat{r}(s,a) + \gamma \sum_{s'} \hat{p}(s' \mid s,a) V(s') \Big]$$

model

estimated value of state s

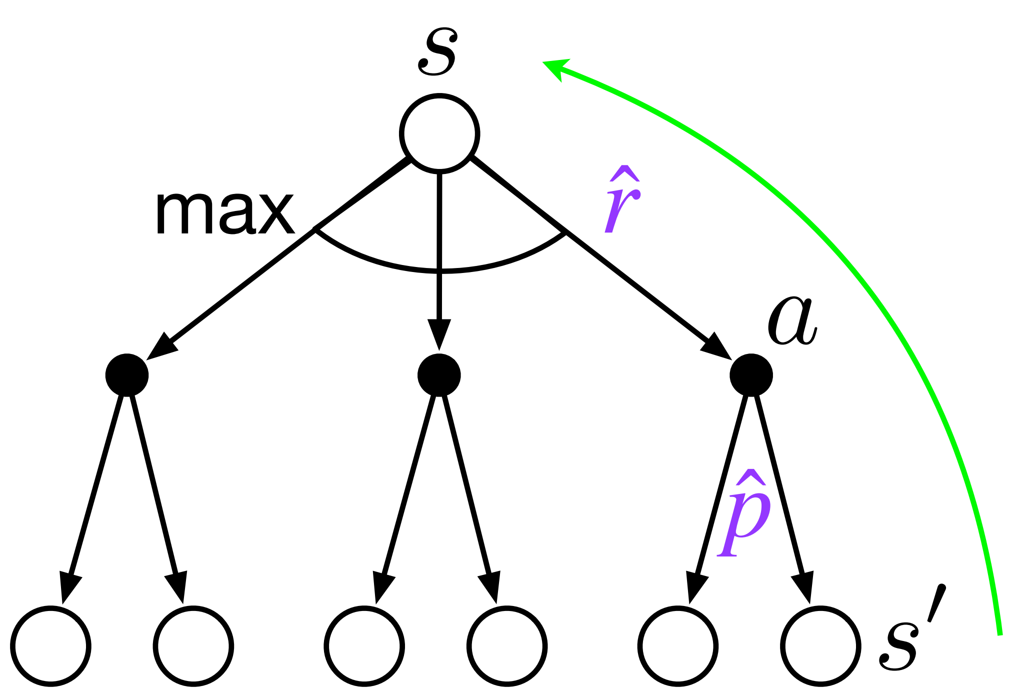
possible action in state s

expected reward when taking a in s

discount rate

possible next state

probability of transition to s' when taking a in s



- Not that different from tree search, MCTS, even A*

- Well suited to RL, and typical of many planning methods

The Five

(outline of the rest of the talk)

5 extensions to make value iteration more realistic and powerful

5 major branch points in research direction

5 challenges

5 tests of your ambition and courage

1. **Average reward**
   moving beyond discounting and episodes
2. **Partial observability**
   moving beyond fully observable Markov state
3. **Temporal abstraction, options**
   resisting the siren call of the one-step trap
4. **Function approximation**
   embracing the demands of the big world perspective
5. **Stochastic transitions**
   moving beyond deterministic worlds

All while remaining simple, scalable, general, and computationally efficient

[Things in red are changed or new]

1. Average reward

moving beyond discounting and episodes

- Discounting is not compatible with approximation and control
- Really, there are no episodes
- The agent should maximize the average reward per-step
  - It is 'easy' to do

Conventional value iteration:

$$V(s) \leftarrow \max_a \Big[\hat{r}(s,a) \; + \; \underbrace{\gamma}_{\text{discount-rate parameter}} \sum_{s'} \hat{p}(s'|s,a)V(s')\Big]$$

Becomes *differential value iteration* (Wan, Naik & Sutton, 2021):

$$V(s) \leftarrow \max_a \Big[\hat{r}(s,a) \underbrace{-\bar{R}}_{\text{reward-rate estimate}} + \sum_{s'} \hat{p}(s'|s,a)V(s')\Big]$$

with $\bar{R} \leftarrow \max_a \Big[\hat{r}(s,a) + \sum_{s'} \hat{p}(s'|s,a)V(s') - V(s)\Big]$

[Things in red are changed or new]

2. Partial observability

moving beyond fully observable Markov state

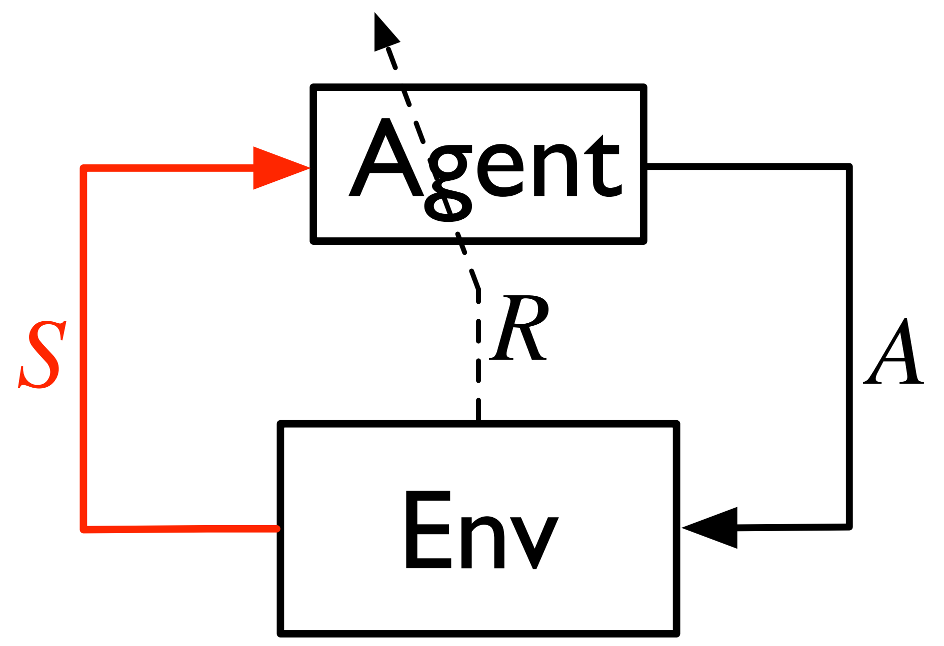
- Really, the state is not given, only an *observation* O_t
- The agent must *create state* from observations and actions

$$S_t = u(S_{t-1}, A_{t-1}, O_t)$$

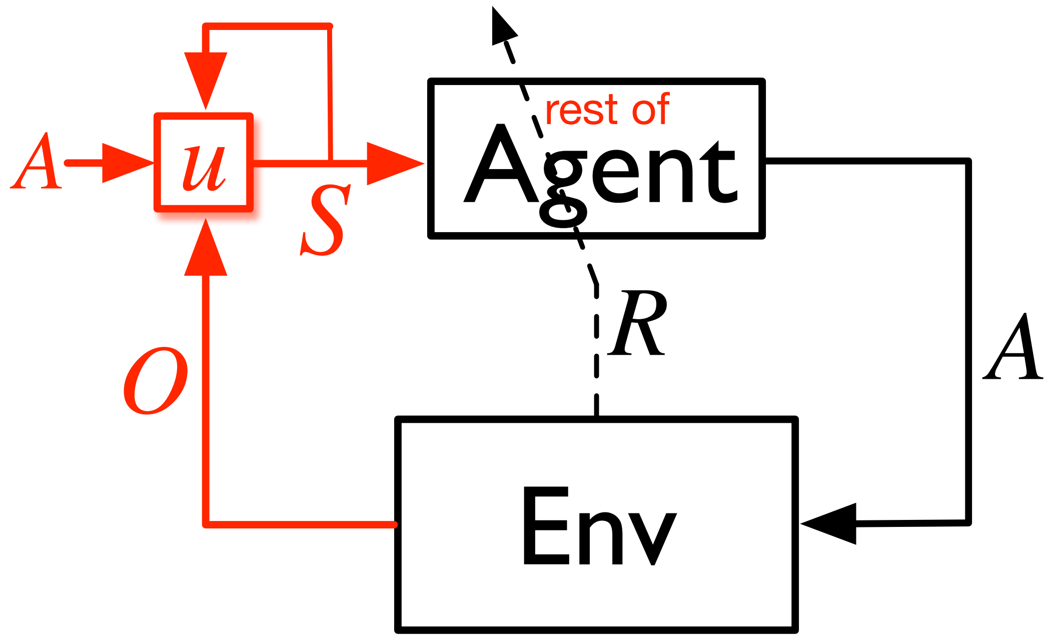
new state | state-update function | last action | new observation

- State update is part of the fast part of the intelligence
- The created state is not Markov for the env, but it is for the model

Conventional agent-environment interaction:



Interaction with observations:



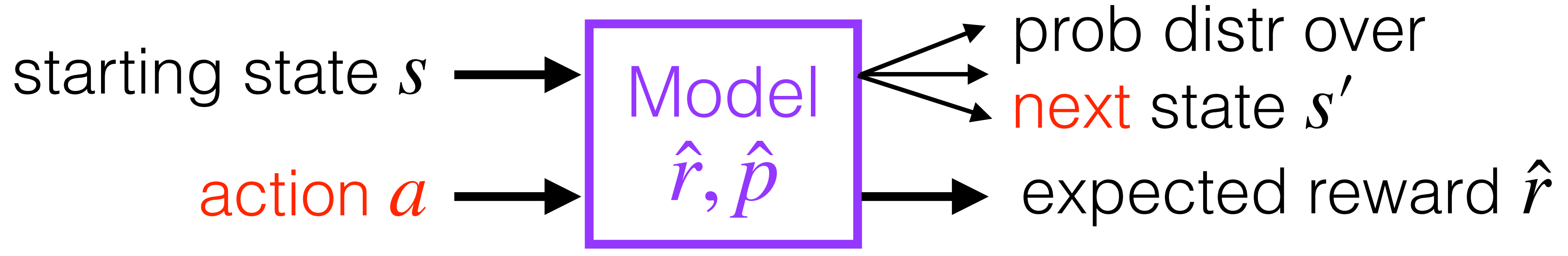
State update reduces the new case to the old!

3. Temporal abstraction, options

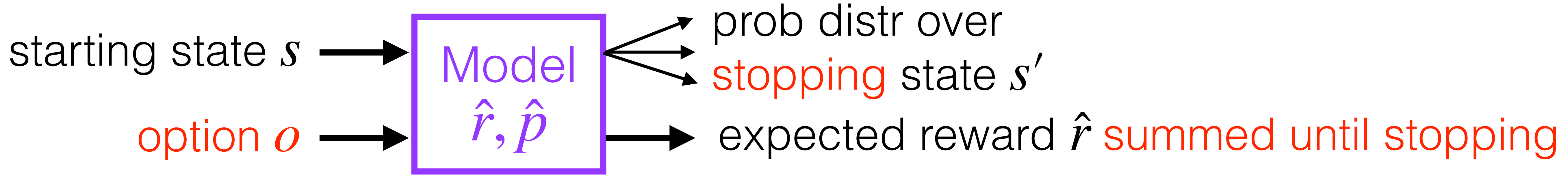
resisting the siren call of the one-step trap

- Life is lived one step at a time
- But it is planned at higher levels
- Our models, our *knowledge*, is about large-scale purposive dynamics
    - conditional not on single actions
    - but on sustained, ways of acting
    - knowledge is about *options*
- Option = policy + stopping condition
- Option model = just like before, but with a temporally abstract semantics

Conventional model:



Becomes an *option model*:



Value iteration is unchanged!

$$V(s) \leftarrow \max_{o} \left[\hat{r}(s, o) + \sum_{s'} \hat{p}(s' | s, o) V(s') \right]$$

possible option in state s

The one-step trap:
Thinking that one-step predictions are sufficient

- Thinking that we can predict the state and observation *one step later*
  - with longer-term predictions made by *iterating* the model at the time the prediction is made

- In theory this works, but not in practice
  - iteration amplifies even small errors in the one-step predictions
  - longer-term predictions are policy dependent, and finding the policies involves branching and concomitant exponential complexity

- We need direct models of many particular policies (options, jumps)

- POMDP and related methods can never escape the trap because they use state update in backups, and state update is inherently one step

Options must still be discovered, their models learned

- One way to discover options is for the agent to pose subproblems for itself
  - and then learn optimal policies and stopping conditions (this is a normal RL learning problem)
- Then, given the options, their *models* can be learned

Self-supervised prediction learning

  - by Monte Carlo methods (from the start and stop states), but only on-policy

Temporal-difference prediction learning

  - by modern TD methods — online, off-policy, and incrementally

4. Function approximation

embracing the demands of the <u>big world perspective</u>

- The environment is huge!
- Much bigger than the agent
- Even a single state is too big
  - the state of the environment includes
    the positions of all atoms…
    the thoughts of all people…
- The dynamics of the world is roughly
  the *square* of the state's complexity!
- Still, we have function approximation…

$V(s)$ becomes $\hat{v}(s, \mathbf{w})$

model $\hat{r}, \hat{p}$ also becomes parametric

4. Function approximation

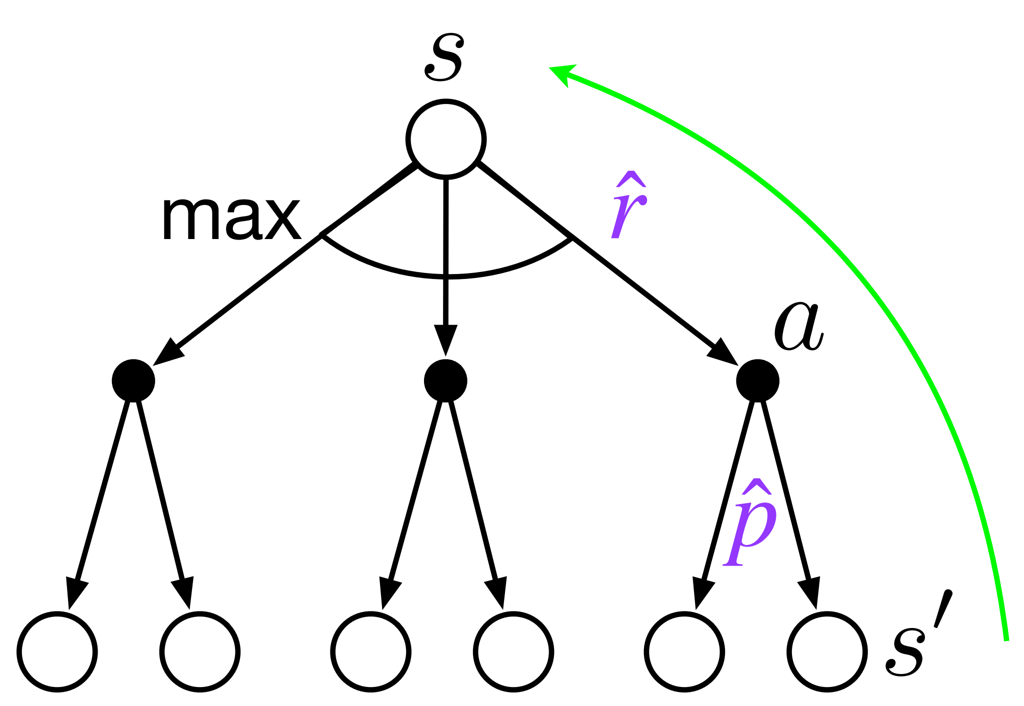
embracing the demands of the big world perspective

Conventional value iteration:

$$V(s) \leftarrow \max_a \left[\hat{r}(s,a) + \gamma \sum_{s'} \hat{p}(s'|s,a) V(s') \right]$$

$V(s)$ becomes $\hat{v}(s, \mathbf{w})$

model $\hat{r}, \hat{p}$ also becomes parametric



Becomes *approximate value iteration*:

$$b(s,a,\mathbf{w}) \;=\; \hat{r}_\theta(s,a) + \gamma \sum_{s'} \hat{p}_\theta(s'|s,a) \hat{v}(s,\mathbf{w})$$

"backed-up value" of state-action pair

model parameter

weight vector of function approximator

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[\max_a b(s,a,\mathbf{w}) - \hat{v}(s,\mathbf{w}) \right] \nabla_{\mathbf{w}} \hat{v}(s,\mathbf{w})$$

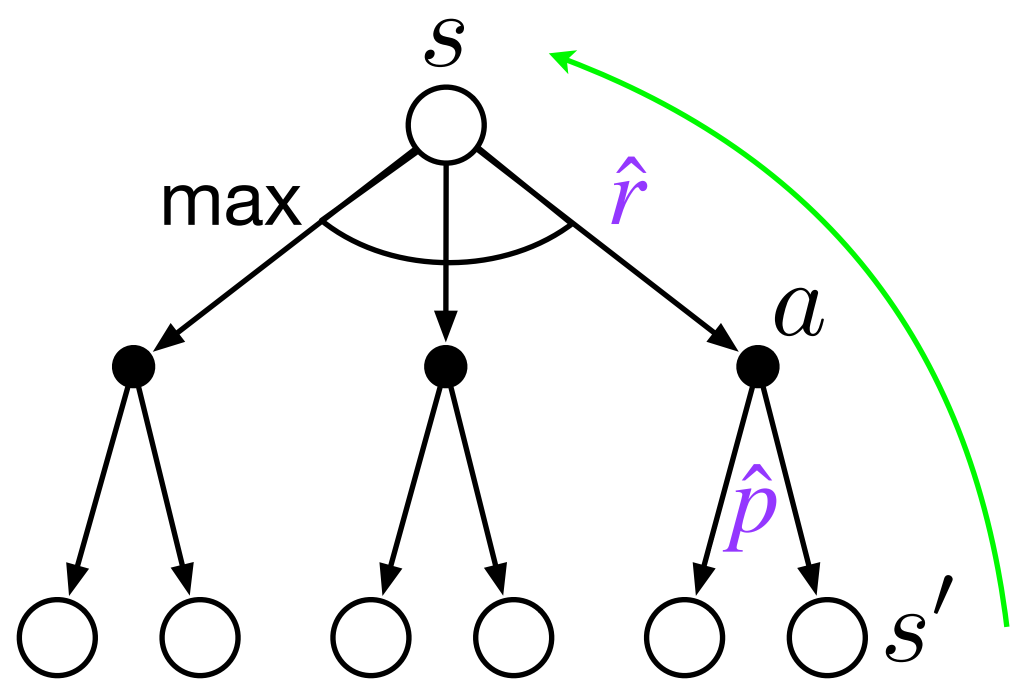
step-size parameter

gradient vector

Consider the computational expense

$$b(s,a,\mathbf{w}) \;=\; \hat{r}_\theta(s,a) + \gamma \boxed{\sum_{s'} \hat{p}_\theta(s' \,|\, s,a)\hat{v}(s,\mathbf{w})}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[\max_a b(s,a,\mathbf{w}) - \hat{v}(s,\mathbf{w})\right] \nabla_{\mathbf{w}} \hat{v}(s,\mathbf{w})$$



- This operation is called a *backup* of state s
- Remember, *many* states must be backed up, perhaps many times (two outer loops in classical VI)
  - To be feasible and effective, the states backed up must be carefully selected (*search control*)
- There are also two loops *inside* each backup
  - The $\max$ is a problem if there are many options, but it can be done incrementally (keep track of best-so-far, check selected new options to see if they are better)
  - The $\sum$ is a problem if the world is stochastic. Can the expected action value be computed efficiently??

5. Stochastic transitions

moving beyond deterministic worlds

- The expected action value must be computed for every backup:

$$\sum_{s'} \hat{p}_\theta(s' | s, a)\hat{v}(s', \mathbf{w})$$

- It is cheap if the world is deterministic
- But really the environment is *very* stochastic; there are lots of next states s'
- If the model returns *samples* of the next state, then we would be pretty good (*sample model*)
- There is a trick if the value function is linear

$$\hat{v}(s, \mathbf{w}) = \mathbf{s}^\top \mathbf{w}$$

state feature vector

Conventional model:

starting state s → Model $\hat{r}, \hat{p}$ → prob distr over next state s'

action a → Model $\hat{r}, \hat{p}$ → expected reward $\hat{r}$

becomes an *expectation model*:

starting state feature vector $\mathbf{s}$ → Model $\hat{r}, \bar{\mathbf{s}}$ → expected next state $\bar{\mathbf{s}}$

action a → Model $\hat{r}, \bar{\mathbf{s}}$ → expected reward $\hat{r}$

The computation of the expected action value is now cheap and exact:

$$\sum_{\mathbf{s}'} \hat{p}_\theta(\mathbf{s}' | \mathbf{s}, a)\hat{v}(\mathbf{s}', \mathbf{w}) = \sum_{\mathbf{s}'} \hat{p}_\theta(\mathbf{s}' | \mathbf{s}, a)\, \mathbf{s}'^\top \mathbf{w}$$

transpose, indicating inner product

$$= \left(\sum_{\mathbf{s}'} \hat{p}_\theta(\mathbf{s}' | \mathbf{s}, a)\, \mathbf{s}'\right)^\top \mathbf{w}$$

$$= \mathbb{E}\left[\mathbf{s}_{t+1} \mid \mathbf{s}_t = \mathbf{s}, A_t = a\right]^\top \mathbf{w}$$

$$\doteq \bar{\mathbf{s}}(\mathbf{s}, a)^\top \mathbf{w}$$

the expectation model gives us this

Questions

Q. Does the trick (expectation models + linear value functions) work with all the other things (options, average reward, state update)?

A. yes, everything goes through

Q. Doesn't the restriction to linear value functions mean that we have failed to achieve generality and scalability?

A. not entirely clear, but i think maybe not;
everything can be linear with the right state-feature representation

Q. Do we know how to learn sample models with options?

A. definitely not

Q. Can we make a sample model work at all?

A. not clear

My best guess at a full solution to planning…

- Expectation model + linear value function + state-feature creation
- The model is heavily annotated with meta data — not just the expectations, but certainties and usefulness-es
    - remember we will need usefulness measures for search control anyway
- Useful option models are rare; the agent searches for them
- Learning a model is not like filling in a table, or estimating known quantities; it is more like searching for rare gold
- Where the gold is options whose approximate models are useful
- Options provide semantics, making the learning problem well defined, but useful options and state features must be found by experimentation

Final perspective

- *Planning is subtle and surprisingly unsolved if there are stochastic dynamics*
- *There are grand research-strategy decision points in planning;*
  these are important and we should be aware of them
    - *You need not choose as I have chosen, but you must choose*
- *This is a grand quest! Automated discovery and learning of <u>knowledge</u>!*
    - *We should be ambitious in our vision, incremental in our progress*

Thank you for your attention

with special thanks to Joseph Modayil, Yi Wan, Abhishek Naik, M. Zaheer, Katya Kudashkina, Martha Steenstrup