UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3547 Intelligent Agents & Reinforcement Learning

Module 7: Dynamic Programming & Monte Carlo Methods

Course Plan

Module Titles
Module 1 – Introduction to Intelligent Agents
Module 2 – Search
Module 3 – Logical Inference
Module 4 – Planning and Knowledge Representation
Module 5 – Probabilistic Reasoning
Module 6 – Intro to Reinforcement Learning and Finite Markov Decision Processes
Module 7 – **Current Focus: Dynamic Programming and Monte Carlo Methods**
Module 8 – Temporal Difference Learning
Module 9 – Function Approximation for RL
Module 10 – Deep Reinforcement Learning and Policy Gradient Methods
Module 11 – Introduction to Advanced DRL
Module 12 – Presentations (no content)

Learning Outcomes for this Module

- Discuss how an optimum policy can be computed when a model is known
- See how an optimum policy can be learned when the environment dynamics aren’t known in advance

Topics for this Module

- **7.1** Dynamic Programming
- **7.2** Monte Carlo Methods
- **7.3** Resources and Wrap-up

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 7 – Section 1

Dynamic Programming

Review of the Four Elements

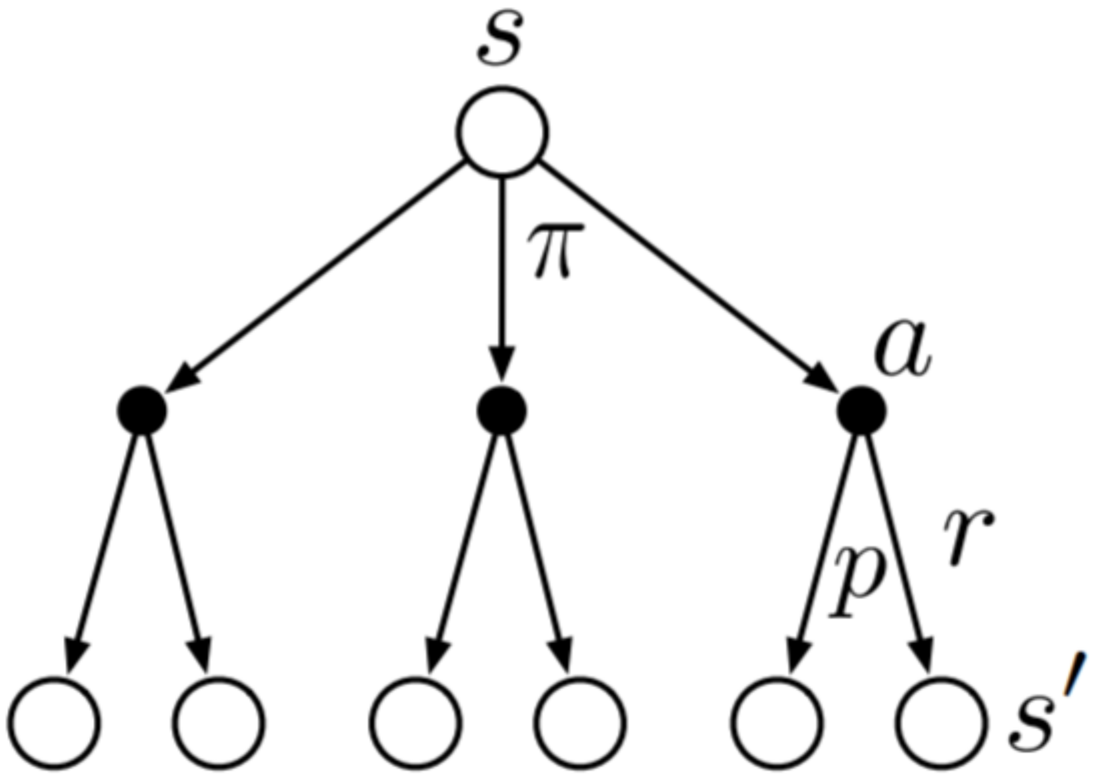
- **Policy**
  - $\pi(a|s)$
- **Reward Signal**
  - R
- **Value Function**
  - Value of a state: $v_\pi(s) = E_\pi(G_t|S_t = s)$
  - Value of a state-action: $q_\pi(s,a) = E_\pi(G_t|S_t = s, A_t = a)$
- **Model of the Environment**
  - $p(s', r|s, a) = \Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$

The Bellman Equation for v_π

- Tells us a recursive condition that must hold for any policy:

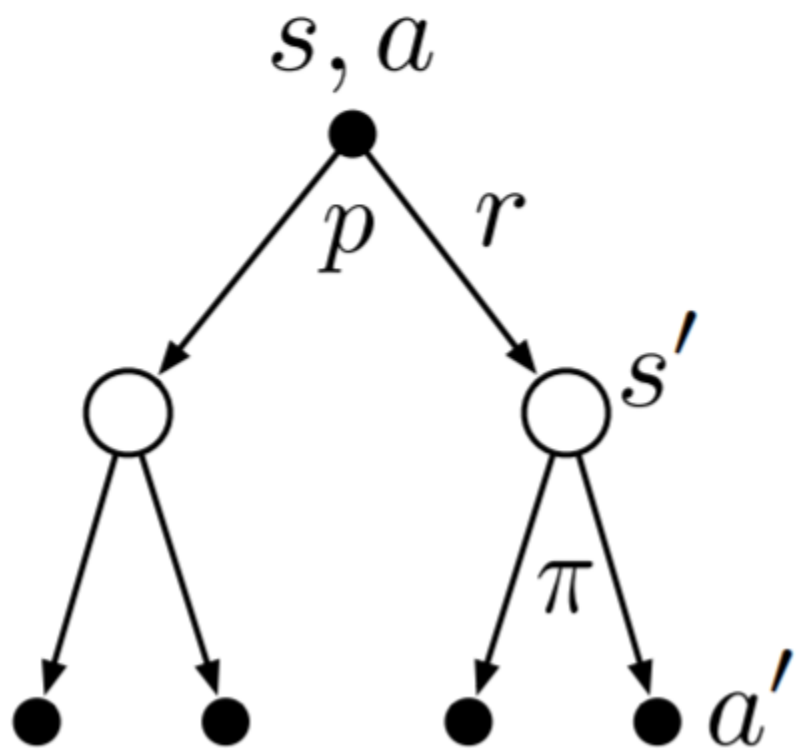
$$
\begin{aligned}
v_\pi(s) &= E_\pi(G_t|S_t = s) \\
&= E_\pi(R_{t+1} + \gamma G_{t+1}|S_t = s) \\
&= \sum_a \pi(a|s) \sum_{s\prime} \sum_r p(s', r|s, a)[r + \gamma E_\pi(G_{t+1}|S_{t+1} = s')] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \text{ for all } s \in S
\end{aligned}
$$

The Backup Diagram for v_π

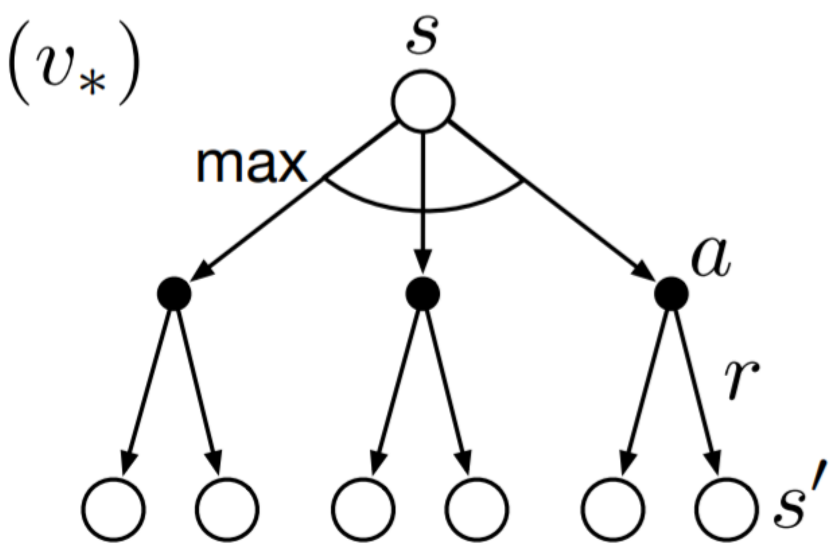


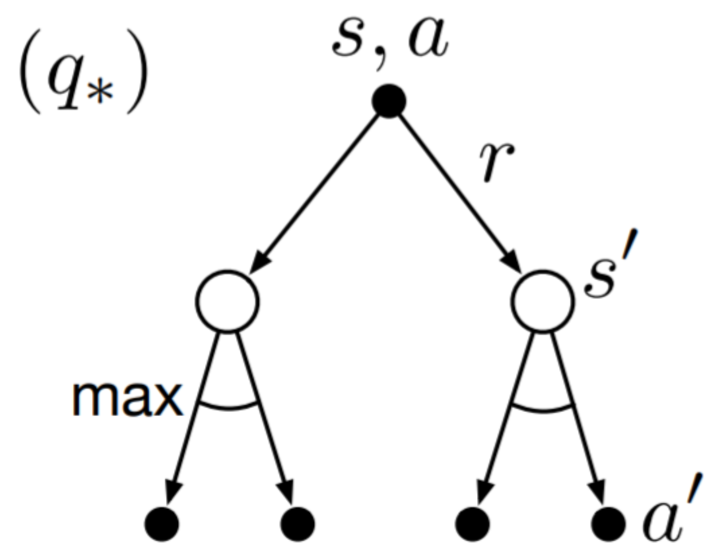
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

The Backup Diagram for q_π



Bellman Optimality





Solving for Optimality

- The objective is to optimize long-run total reward
- We've seen we can do this by always choosing the next action with the largest $q_t(a)$ (or an approximation $Q_t(a)$ if that's all we have)
- For Finite MDP's the Bellman equation is a system of n non-linear equations in n unknowns
- Is there another way we can solve it, given that it is defined recursively?

Policy Evaluation (Prediction)

- Yes! Dynamic Programming!
- We start by solving for any v_π, not just optimal ones
- We convert the Bellman equation into an update rule:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

  and (very) fortunately the sequence $\{v_k\}$ converges to v_π as $k \to \infty$
- Rather than using separate arrays for v_k and v_{k+1} it turns out that this usually converges faster if we use a single array
- We sweep through the entire array v updating all the values on each iteration

Policy Improvement

- If we have any old policy π and its state-value function v_π we can find better policies by being greedy and choosing the action in each state that takes us to a state with the biggest expected value of $v_\pi(s')$

$$\pi'(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- Remember, in general π is actually π(a|s), a probability distribution over actions for each state so if several actions are equally optimal, we can keep them all and select between them with any distribution we like

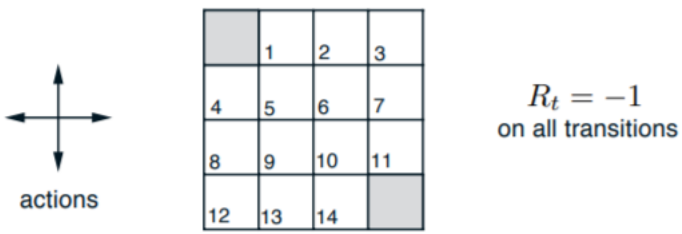
Policy Iteration

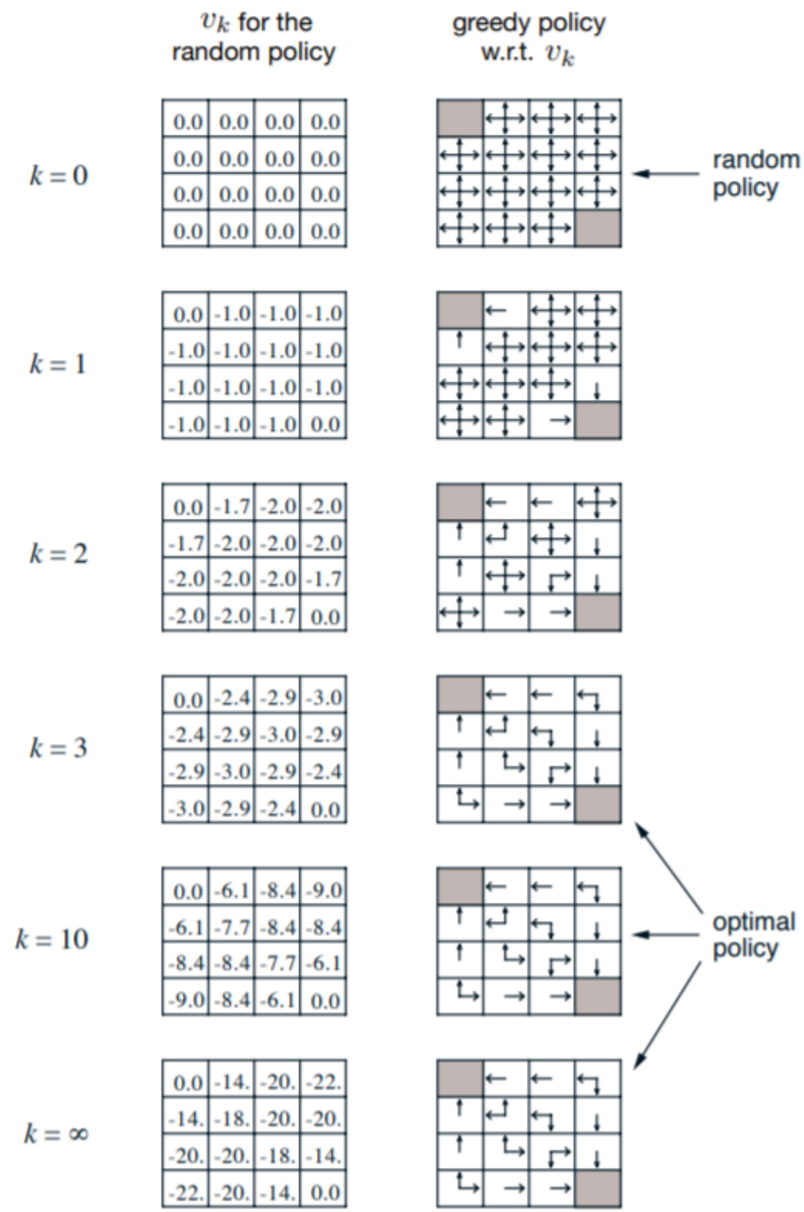
- Now we have a better policy π_{k+1}, but our value function v_k is not going to be still right for it, so we need to re-evaluate it to get v_{k+1}
- But then we can find a better π: π_{k+2}, and so on…
- We can interleave recalculations of v and π
- Repeatedly switching between policy evaluation and improvement will eventually converge to our optimal π_*

Value Iteration

- Each iteration of *policy iteration* requires a *policy evaluation* which means computing successive values of $v(s)$ over all of the states potentially many times before it converges to v_π
- What if we just make $v(s)$ a little closer to v_π (just one sweep through the states) then do a policy improvement?
- Turns out this converges too (sometimes even quicker) and is called **value iteration**

Policy Evaluation & Improvement

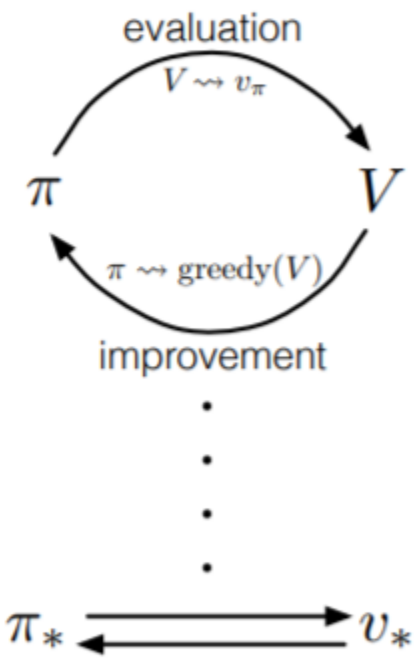


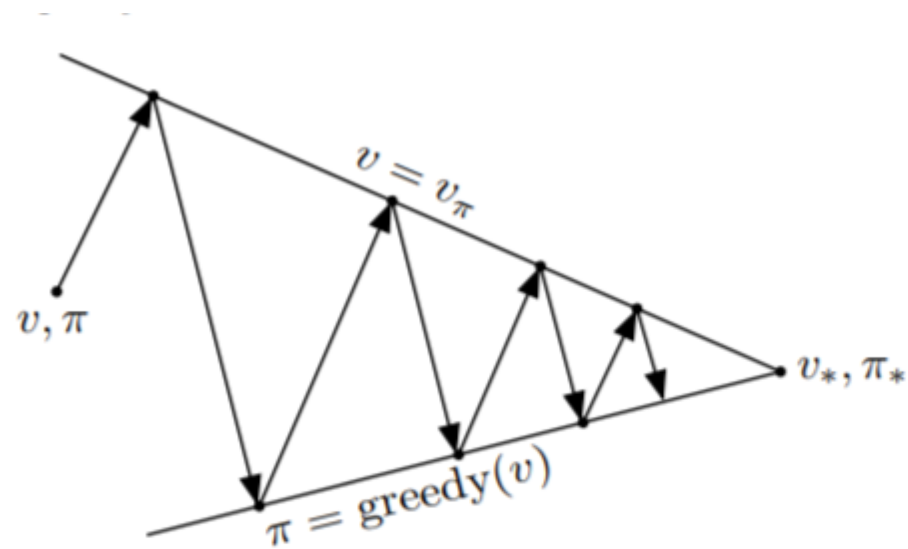


Source: Sutton & Barto

Generalized Policy Iteration

- Sutton and Barto call this notion of interacting policy evaluation and improvement **generalized policy iteration**





Source: Sutton & Barto

Asynchronous Dynamic Programming

- Turns out we don’t even need to do full state sweeps
- We can update states in any order and any number of times as long as all states are eventually updated
- Particularly useful if the agent is interacting with the environment so new information about some states is becoming available and we’d like to use it immediately
- Asynchronous DP algorithms are designed for these situations

Efficiency of Dynamic Programming

- DP is very efficient (polynomial in the number of states and actions)
- Can potentially be used with state spaces with millions of states
- But still not efficient enough for very large state spaces
- Here asynchronous DP is often better, or approximating methods we will look at next

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 7 – Section 2

Monte Carlo Methods

Monte Carlo Prediction

- What if we don't have a complete specification of the environment's dynamics?
- How can we learn from actual or simulated experience?
- Monte Carlo methods apply when we need to roll out to the end of an episode to determine the total reward
- We'll build on the methods we've already discussed

Monte Carlo Estimation of State Values

- When we can't determine $v_\pi(s)$ exactly we need to estimate it: $V(s)$
- The strategy: simulate many entire episodes and average them
- We can update $V(s)$ the first time we visit state s or every time we visit it: they both converge
- Consider how this would work in various games: Blackjack, chess, checkers, Go, WumpusWorld

Monte Carlo Estimation of Action Values

- We can roll out and update estimates $Q(s,a)$ of the expected return of state-action values $q_\pi(s,a)$ in the same way as we did for $V(s)$
- *But* we need to ensure that all valid actions can be tried, either by:
  - Intentionally starting in states chosen at random (**exploring starts**), or
  - Always having some chance of selecting any of the actions available in a state (for example, by being ε-greedy), or
  - By using a separate, exploratory policy for learning and adjusting the expected values of the rewards to more closely match what they should be under the current actual policy (called **off-policy estimation with importance sampling**)

Monte Carlo Control

- Sutton and Barto refer to the process of finding or approximating optimum policies as the **control problem**
- The issue is the huge number of episode rollouts required to fully learn $Q(s, a)$ so we need to compromise in some way:
  - Learn $Q(s, a)$ for many state-actions but not very accurately
  - Learn $Q(s, a)$ only for likely trajectories

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 7 – Section 3

Resources and Wrap-up

Resources

- Sutton & Barto. Reinforcement Learning, 2nd Ed. MIT Press. 2018.
- Richard Sutton's website: http://incompleteideas.net/
- https://gym.openai.com/
- Pumperla & Ferguson. Deep Learning and the Game of Go, Chap. 9. Manning. 2019.
- https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter05/blackjack.py (note: you'll probably need to `pip install tqdm` if you want to try running it yourself)

Summary

- If we have a complete specification of the dynamics of the environment we can directly solve for an optimal set of actions
- If not, we can estimate the value functions probabilistically using Monte Carlo techniques

Next Week

- Temporal Difference Learning

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies