Reinforcement Learning: Planning using Dynamic Programming

Earl Wong

Planning

- Planning is defined as the process for computing value functions and policies, using a model of the environment dynamics.

- For example, in an environment involving games, plans are computed using a model of the game.

- The plan describes all actions that can be taken over a fixed window of time from the present time.

- The agent executes the first action of the plan, and (then) discards the rest of the plan.

- The process repeats.

- Chess example: There is a given board state. 1) Every initial, next move by white is examined. 2) Every initial, next move (then) has multiple “follow on” moves by black, by white, by black, by white, etc. One of the initial moves in 1) will yield the largest return. This move is executed. The remainder of the plan is discarded.

Planning

- We will discuss two main algorithms: Value Iteration and Policy Iteration.

- Policy Iteration consists of two components: Policy Evaluation and Policy Improvement

- Value Iteration involves repeatedly updating the value function, without any policy improvement.

- In both cases, dynamic programming is involved.

Dynamic Programming

- What is dynamic programming?
- Dynamic programming solves complex problems by breaking the problem into simpler sub-problems, solving the sub-problems and reusing the solution to the solved sub-problems.

Dynamic Programming

- Dynamic programming is composed of two properties: optimal sub-structure and re-occurring sub-problems.

- If a problem has optimal sub-structure, the optimal solution can be constructed from the the optimal solutions of it's sub-problems.

- Example: Shortest path. Given a midpoint, two sub-problems exist. The optimal solution can be constructed from 1) the shortest path from the start to the midpoint and 2) the shortest path from the midpoint to the end.

- Re-occurring subproblems refer to sub-problems that can benefit from caching.

- Example: Calculating the Fibonacci numbers recursively. Here, numerous re-occurring sub-problems exist.

Dynamic Programming

- The Markov Decision Process encompasses both properties of Dynamic Programming.

- 1) The Bellman Optimality Equation models the optimal sub-structure property.

- This is because the Bellman Optimality Equation can be broken into two components - the optimal behavior for the first step (immediate reward) and the optimal behaviors after the first step (trajectory)

- 2) The value function models the re-occurring sub-problem property since the value function stores and re-uses past solutions.

Policy Iteration

- As stated previously, policy iteration consists of two components: policy evaluation and policy improvement.

- In policy evaluation, the state value function was obtained by evaluating a specific policy.

- This was summarized by the following relation:

$$V_{k+1}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_k(s')]$$

- Repeated iteration of policy evaluation produces a sequence of value functions that converge for the given policy.

$$V_1(s) \rightarrow V_2(s) \rightarrow \ldots \rightarrow V_\pi(s)$$

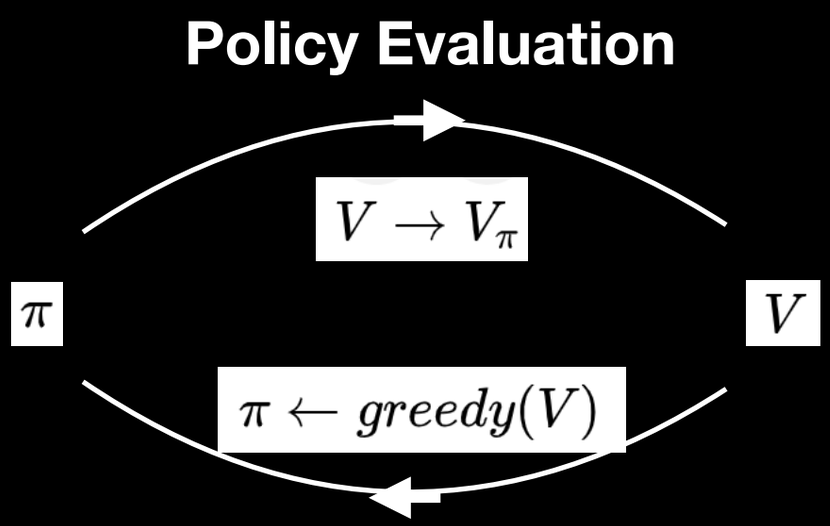
Policy Improvement

- Policy improvement entails choosing a new policy that is better than the existing policy.
- For example, a greedy policy satisfies this requirement:

$$\pi' = greedy(V_\pi)$$

Policy Iteration

The entire algorithm is now summarized (in picture and in words).



$$\pi_* \rightarrow V_*$$

$$\pi_* \leftarrow V_*$$

1) Initialize a state value function V(s) = 0 for all states.

2) Apply a random policy.

3) Continue to apply the policy for all states, until a stopping criteria is satisfied.

4) Update the current policy, by applying a greedy policy improvement, using the current value function.

5) Repeat steps 3 and 4 until convergence / improvement stops.

This produces the optimal policy and the optimal value function.

Note: Convergence is guaranteed by the contraction mapping theorem.

Policy Iteration

- In general, we do not need to begin with a random policy. i.e. We can start with any general policy.

- In general, we do not need to use a greedy policy improvement algorithm. i.e. We only require that

$$\pi' > \pi$$

- By starting with a better policy, “wall clock” convergence times will decrease.

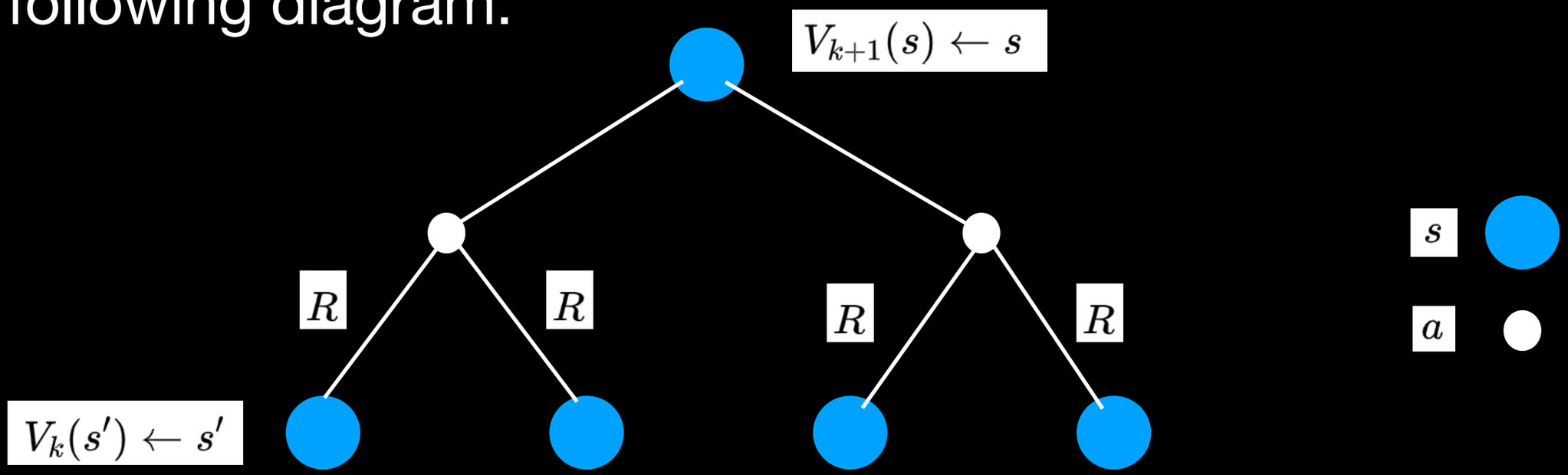
- Regardless of the policy used, the convergence rate remains fixed.

Value Iteration

- Previously, step 3 consisted of applying N iterations, for some integer valued N.
- In value iteration, N=1.
- After each iteration, a new value function is obtained.

Value Iteration

- Value iteration consists of the following repeated operation: $$V_{k+1}(s) = max_a \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_k(s')]$$
- To understand what this formula is doing, consider the following diagram:



For a given state s, we want to find the action a that produces the maximum expected return, using the value function stored at the previous time iteration k.

We then update the value function at state s for time k+1, using the selected action a, that produced the maximum expected return.

This slide was taken from David Silver's lecture notes, and summarizes everything discussed so far.

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2 n^2)$ per iteration

Synchronous versus Asynchronous Updates

- Suppose there are N states in the value function. (In the previous GridWorld, example there were 15).
- If every state is updated at time k before proceeding to time k + 1, synchronous updates are occurring.
- This is what we have been doing so far.
- In practice, there may be many locations in the state space that are of “less” concern, requiring less frequent updates.

Practical Considerations

- In these cases, it would be advantageous to assign the majority of the updates to the “most important” states.

- “Most important” could be defined in several different ways:

- 1) The states that experience the largest change at time k+1, relative to their values at time k.

- 2) The states that have been most frequently visited so far (and their surrounding / closest states).

- In practice, as long as every state continues to be visited, convergence is guaranteed.

Curse of Dimensionality

- As the number of states continue to increase, a computational bound will eventually be reached.
- In the case of GridWorld, for every additional state, four additional choices are added.
- This bound restricts the size of the value functions that can be solved using dynamic programming.
- As a workaround, model free techniques can be used in place of dynamic programming.
- Model free techniques break the curse of dimensionality, since model free techniques only require sample rollouts.

Convergence

- How do we know that policy evaluation, policy iteration and value iteration will converge?
- To answer this question, we need to use the contraction mapping theorem.

  Contraction Mapping Theorem

- Suppose you are given a complete (closed) metric space and an operator T. Let T be a contraction map -> Then, T converges to a unique fixed point.

Convergence

- Let V be the space of all possible value functions and let T be the Bellman expectation equation.
- By construction, V is closed. In addition, the Bellman expectation equation is a contraction map.
- Therefore, a fixed point exists, guaranteeing convergence of policy evaluation to V_π
- Similarly, every policy in policy improvement is a contraction map.
- Hence, policy iteration converges to π_*
- Similarly, the Bellman optimality equation is a contraction map. Hence, value iteration converges to V_*

Convergence

Contraction Map

We are given elements U and W. (For example, U and W could be NxN matrices.)

The operator T is a contraction map, if the following relation holds:

$$||T(U) - T(W)||_\infty \leq ||U - W||_\infty$$

For our specific problems of interest, U and W represent the following value functions:

$$U = V_{t+1}(s),\ \ W = V_t(s)$$