Reinforcement Learning
Lectures 7

Gillian Hayes

29th January 2007

Algorithms for Solving RL: Dynamic Programming

- Policy Evaluation
- Iterative Policy Evaluation
- Policy Improvement
- Policy Evaluation + Policy Improvement = Policy Iteration
- Value Iteration
- Asynchronous Dynamic Programming
- Generalised Policy Iteration

Dynamic Programming

Needs perfect model $P^a_{ss'}$ and $R^a_{ss'}$.

We want to compute V^*, Q^*, the optimal value and action-value functions

POLICY EVALUATION

Suppose we have some policy π which tells us what action a to choose in state s. Find the value function $V^\pi(s)$ of this policy, i.e. eVALUate this policy.

Bellman Equation for $V^\pi(s)$:

$$\begin{aligned} V^\pi(s) &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \\ &= \sum_a \pi(s, a,) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

Soluble, but BIG – $\mid S \mid$ equations in $\mid S \mid$ unknowns. So we need to iterate....

Iterative Policy Evaluation 1

We apply a "sweep", i.e. a **backup operation** to **each** state to compute the value at that state. Each sweep ($\rightarrow$) updates our current estimate of the value function.

$$V_0 \rightarrow V_1 \rightarrow \ldots V_k \rightarrow V_{k+1} \ldots \rightarrow V^\pi$$

Update the value at state s thus:

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

In other words, we're using the Bellman equation as an iterative update equation

Update the V's for *all states* iteratively. This is called **a full policy evaluation backup**. After many sweeps we'll converge to $V^\pi(s)$

Iterative Policy Evaluation 2

1. Start with arbitrary V values
2. Iterate/update

$\Rightarrow$ get V for that policy

V^π is a fixed point – it solves the Bellman equation

So: GIVEN π
WE NOW HAVE V^π

Is it possible to improve policy π?
Yes, we can do **policy improvement**

Policy Improvement 1

We have V^π for our policy π. Can we choose a better action than that stipulated by π? In other words, an $a \neq \pi(s)$.

The value of choosing action a in state s is the Q-value:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

If $Q^\pi(s,a)$ is greater than our current estimate $V^\pi(s)$ then we should choose a.

Do this for each state: **policy improvement** - because we're changing to a policy that gets us more return.

Policy Improvement 2

Improve the policy at each state. We get a new policy π' that's greedy wrt V^π.

$$\begin{aligned}\pi'(s) &= \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]\end{aligned}$$

In this case the new policy is better than the old: $V^{\pi'}(s) > V^\pi(s)$

"Greedification"

If $V^{\pi'}(s) = V^\pi(s)$, then

$$V^{\pi'}(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$$

This is the Bellman Optimality Equation, and the value function and policies are optimal:

$V^{\pi} = V^*$ and $\pi = \pi' = \pi^*$

So...

Alternate Policy Evaluation and Policy Improvement

Evaluate – Improve – Evaluate – Improve – Evaluate ...

$\pi_0 \rightarrow_{PE} V^{\pi^0} \rightarrow_{PI} \pi_1 \rightarrow_{PE} V^{\pi^1} \rightarrow_{PI} \pi_2 \ldots\ldots\ldots\ldots \rightarrow_{PI} \pi^* \rightarrow_{PE} V^*$

- Start with a policy
- EVALUate to give V, the value function
- Improve policy
- EVALUate to get new V for improved policy
- Improve policy

- etc.
- Get optimal policy
- Get optimal value function

This process is called **Policy Iteration**

Policy Iteration

1. Initialise
   - $\pi =$ arbitrary deterministic policy
   - $V =$ arbitrary value function
   - $\theta =$ small positive number

2. Policy Evaluation
   - For each state
   - New $V = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$ where $a = \pi(s)$
   - Repeat until no V changes by more than θ

3. Policy Improvement
   - For each state

- Get $b = \pi(s)$
- New $\pi = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
- If policy changed, i.e. new $\pi(s) \neq b$, goto 2

4. Stop

Issues, Improvements

In **policy evaluation** new values V_{k+1} are calculated in terms of V_k, so need *two* arrays.

Could update "in place", overwriting *one* array of Vs as soon as new value is calculated. So some updates use already updated V_k values – uses new data as it becomes available.

In-place converges faster than two-array version

Jargon "Sweep" through state space – updating values as you go

Problem with policy iteration: each iteration needs a policy evaluation – takes a long time, possibly many sweeps through the state space

So . . . **Value Iteration**

Value Iteration

Just update the values for *one* iteration and then improve the policy.

Update rule:

$$V = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$$

This combines the one-iteration update plus the policy improvement (greedification wrt V)

Value Iteration Algorithm

1. Initialise
   - $V,\ \pi =$ arbitrary

2. Repeat
   - For each state
   - Update $V(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
   - Until no V changes by more than some small amount

3. Policy is
   - $\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

Asynchronous Dynamic Programming

If methods require many sweeps through the state space this can take prohibitively long, e.g. Backgammon has 10^{20} states

Asynchronous DP: • Update arrays in-place AND

• No particular order on which V must be updated when – but must do all eventually, you can't ignore states. Gives us the freedom to choose the order in which to backup states.

Example: Asynchronous Value Iteration

• Use value iteration backup

$V(s_k) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

but only backup the value for one state s_k on each step

Converges to V^* if all states backed up infinitely many times and $0 \leq \gamma < 1$

Good Points About Asynchronous DP

- Saves iterating through whole state space on any given timestep (but must backup them all eventually)

- Can save memory (small advantage)

- Faster convergence – it takes fewer state updates to convergence

- Can prioritise sweeps – update those which have some reason to be updated, e.g. new experience of that bit of state space (so focus on relevant states), or their value functions are changing a lot

- Updated value function used immediately in estimates of other states' value function

- We may not care about some states – maybe we never expect to visit them – so make their backup priority very low

We can consider interleaving the policy evaluation and policy improvements steps at many granularities.... This is called **Generalised Policy Iteration**

Summary

- Policy Evaluation: backups without a max, find the value function for a given policy
- Policy Improvement: make policy greedy wrt value function (if only locally)
- Policy Iteration = Policy Evaluation + Policy Improvement
- Value Iteration: backups with a max, i.e. Bellman optimality equation
- Asynchronous Dynamic Programming: avoids exhaustive sweeps through state space when updating V
- Generalised Policy Iteration: Interleaving policy evaluation and improvement at any granularity

- **Bootstrapping**: updating estimates based on other estimates

- **Full** backups: each backup takes into account **all** the states one can reach from the current state in calculating the backup

Read Sutton and Barto Chapter 4