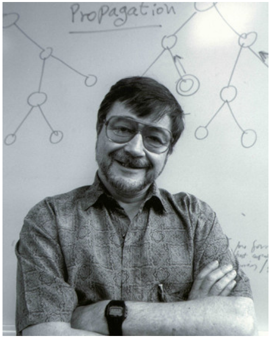
15-780: Markov Decision Processes

J. Zico Kolter

Feburary 29, 2016

Outline

1988



Judea Pearl publishes *Probabilistic Reasoning in Intelligent Systems*, bring probability and Bayesian networks to forefront of AI

Speaking today for the Dickson prize at 12:00, McConomy Auditorium Cohon University Center

Outline

Decision making under uncertainty

Building upon our recent discussions about probabilistic modeling, we want to consider a framework for decision making under uncertainty

Markov decision processes (MDPs) and their extensions provide an extremely general way to think about how we can act optimally under uncertainty

For many medium-sized problems, we can use the techniques from this lecture to compute an optimal decision policy

For large-scale problems, approximate techniques are often needed (more on these in later lectures), but the paradigm often forms the basis for these approximate methods

Markov decision processes

A more formal definition will follow, but at a high level, an MDP is defined by: states, actions, transition probabilities, and rewards

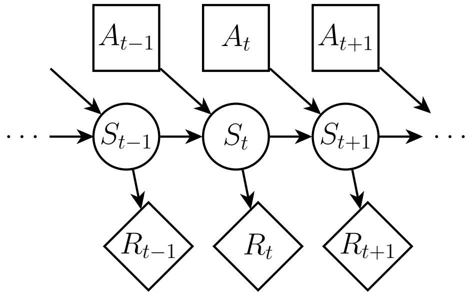
States encode all information of a system needed to determine how it will evolve when taking actions, with system governed by the state transition probabilities

$$P(s_{t+1}|s_t, a_t)$$

note that transitions only depend on current state and action, not past states/actions (Markov assumption)

Goal for an agent is to take actions that maximize expected reward

Graphical model representation of MDP



Applications of MDPs

A huge number of applications of MDPs, using standard solution methods: see e.g. [White, “A survey of applications of Markov decision processes”, 1993]

Survey lists: population harvesting, agriculture, water resources, inspection, purchasing, finance, queues, sales, search, insurance, overbooking, epidemics, credit, sports, patient admission, location, experimental design

But, perhaps more compelling is the number of applications of using *approximate* solutions: self-driving cars, video games, robot soccer, scheduling energy generation, autonomous flight, many many others

In these domains, small components of the problem are still often solved with exact methods

Outline

Formal MDP definition

A Markov decision process is defined by:

- A set of *states* $\mathcal{S}$ (assumed for now to be discrete)

- A set of *actions* $\mathcal{A}$ (also assumed discrete)

- *Transition probabilities* P, which defined the probability distribution over next states given the current state and current action

$$P(S_{t+1}|S_t, A_t)$$

- **Crucial point**: transitions only depend on the *current* state and action (Markov assumption)

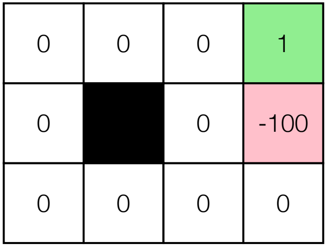
- A *reward function* $R : S \to \mathbb{R}$, mapping states to real numbers (can also define rewards over state/action pairs)

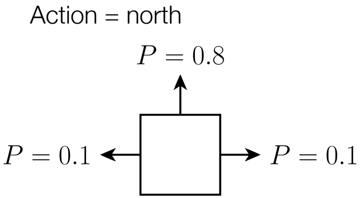
Gridworld domain

Simple grid world with a goal state with reward and a “bad state” with reward -100

Actions move in the desired direction with probably 0.8, in one of the perpendicular directions with

Taking an action that would bump into a wall leaves agent where it is





Policies and value functions

A *policy* is a mapping from states to actions $\pi : S \rightarrow A$ (can also define stochastic policies)

A *value function* for a policy, written $V^\pi : S \rightarrow \mathbb{R}$ gives the expected sum of discounted rewards when acting under that policy

$$V^\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s, a_t = \pi(s_t), s_{t+1}|s_t, a_t \sim P\right]$$

where $\gamma < 1$ is a *discount factor* (also formulations for finite horizon, infinite horizon average reward)

Can also define value function recursively via the *Bellman equation*

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s')$$

Aside: computing the policy value

Let $v^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be a vector of values for each state, $r \in \mathbb{R}^{|\mathcal{S}|}$ be a vector of rewards for each state

Let $P^\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ be a matrix containing probabilities for each transition under policy pi

$$P^\pi_{ij} = P(s_{t+1} = i | s_t = j, a_t = \pi(s_t))$$

Then Bellman equation can be written in vector form as

$$\begin{aligned} v^\pi &= r + \gamma P^\pi v^\pi \\ \Longrightarrow (I - \gamma P^\pi) v^\pi &= r \\ \Longrightarrow v^\pi &= (I - \gamma P^\pi)^{-1} r \end{aligned}$$

i.e., computing value for a policy requires solving a linear system

Optimal policy and value function

The *optimal policy* is the policy that achieves the highest value for every state

$$\pi^\star = \operatorname*{argmax}_\pi V^\pi(s)$$

and it’s value function is written $V^\star = V^{\pi^\star}$ (but there are an exponential number of policies, so this formulation is not very useful)

Instead, we can directly define the optimal value function using the *Bellman optimality equation*

$$V^\star(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V^\star(s')$$

and optimal policy is simply the action that attains this max

$$\pi^\star(s) = \operatorname*{argmax}_a \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V^\star(s')$$

Outline

Computing the optimal policy

How do we compute the optimal policy? (or equivalently, the optimal value function?)

Approach #1: **value iteration**: repeatedly update an estimate of the optimal value function according to Bellman optimality equation

1. Initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0, \;\; \forall s \in \mathcal{S}$$

2. Repeat, update:

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s'), \;\; \forall s \in \mathcal{S}$$

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0	⬛	0	-100
0	0	0	0

Original reward function

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0	0	0.72	1.81
0	⬛	0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268	■	0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$ at five iterations

Illustration of value iteration

Running value iteration with $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021	■	1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$ at 10 iterations

Illustration of value iteration

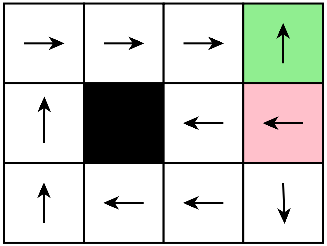
Running value iteration with $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802	■	3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

Illustration of value iteration

Running value iteration with $\gamma = 0.9$



Resulting policy after 1000 iterations

Convergence of value iteration

Theorem: Value iteration converges to optimal value: $\hat{V} \to V^\star$

Proof: For any estimate of the value function $\hat{V}$, we define the Bellman backup operator $B : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$

$$B\,\hat{V}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\,\hat{V}(s')$$

We will show that Bellman operator is a *contraction*, that for any value function estimates $V_1,\ V_2$

$$\max_{s \in \mathcal{S}} |B V_1(s) - B V_2(s)| \leq \gamma \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|$$

Since $BV^\star = V^\star$ (the contraction property also implies existence and uniqueness of this fixed point), we have:

$$\max_{s \in \mathcal{S}} \left| B\,\hat{V}(s) - V^\star(s) \right| \leq \gamma \max_{s \in \mathcal{S}} \left| \hat{V}(s) - V^\star(s) \right| \implies \hat{V} \to V^\star$$

Proof of contraction property:

$$
\begin{aligned}
&|BV_1(s) - BV_2(s)| \\
&= \gamma \left| \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V_1(s') - \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V_2(s') \right| \\
&\le \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V_1(s') - \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V_2(s') \right| \\
&= \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s'|s,a)\, |\, V_1(s') - V_2(s')| \\
&\le \gamma \max_{s\in\mathcal{S}} |\, V_1(s) - V_2(s)|
\end{aligned}
$$

where third line follows from property that

$$|\max_x f(x) - \max_x g(x)| \le \max_x |f(x) - g(x)|$$

and final line because $P(s'|s,a)$ are non-negative and sum to one

Value iteration convergence

How many iterations will it take to find optimal policy?

Assume rewards in $[0, R_{\max}]$, then

$$V^\star(s) \leq \sum_{t=1}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma}$$

Then letting V^k be value after kth iteration

$$\max_{s \in \mathcal{S}} |V^k(s) - V^\star(s)| \leq \frac{\gamma^k R_{\max}}{1-\gamma}$$

i.e., we have linear convergence to optimal value function

But, time to find optimal policy depends on separation between value of optimal and second suboptimal policy, difficult to bound

Asynchronous value iteration

Subtle point, standard value iteration assumes $\hat{V}(s)$ are all updated *synchronously*, i.e. we compute

$$\hat{V}'(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s')$$

and then set $\hat{V}(s) \leftarrow \hat{V}'(s)$

Alternatively, can loop over states $s = 1, \ldots, |\mathcal{S}|$ (or randomize over states), and directly set

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s')$$

Latter is known as *asynchronous* value iteration (also called Gauss-Seidel value iteration given fixed ordering), is also guaranteed to converge, and usually performs *better* in practice

Outline

Policy iteration

Another approach to computing optimal policy / value function

Policy iteration algorithm

1. Initialize policy $\hat{\pi}$ (e.g., randomly)

2. Compute value of policy, V^{π} (e.g., via solving linear system, as discussed previously)

3. Update π to be greedy policy with respect to V^{π}

$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi}(s')$$

4. If policy π changed in last iteration, return to step 2

Convergence property of policy iteration: $\pi \rightarrow \pi^\star$

Proof involves showing that each iteration is also a contraction, and policy must improve each step, or be optimal policy

Interesting theoretical note: since number of policies is finite (though exponentially large), policy iteration converges to *exact* optimal policy

In theory, could require exponential number of iterations to converge (though only for γ very close to 1), but for some problems of interest, converges much faster

Illustration of policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) =$ North

0	0	0	1
0	■	0	-100
0	0	0	0

Original reward function

Illustration of policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

0.418	0.884	2.331	6.367
0.367	■	-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

V^{π} at one iteration

Illustration of policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) =$ North

5.414	6.248	7.116	8.634
4.753	■	2.881	-102.7
2.251	1.977	1.849	-8.701

V^{π} at two iterations

Illustration of policy iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) =$ North

5.470	6.313	7.190	8.669
4.803	■	3.347	-96.67
4.161	3.654	3.222	1.526

V^π at three iterations (converged)

Gridworld results

Approximation of value function

- Policy iteration: *exact* value function after three iterations

- Value iteration: after 100 iterations, $\|V - V^\star\|_2 = 7.1 \times 10^-4$

Calculation of optimal policy

- Policy iteration: three iterations

- Value iteration: 12 iterations

In other words, value iteration converges to optimal policy long before it converges to correct value in this MDP (but, this property is highly MDP-specific)

Policy iteration or value iteration?

Policy iteration requires fewer iterations that value iteration, but each iteration requires solving a linear system instead of just applying Bellman operator

In practice, policy iteration is often faster, especially if the transition probabilities are structured (e.g., sparse) to make solution of linear system efficient

Modified policy iteration (Putterman and Shin, 1978) solves linear system approximately, using backups very similar to value iteration, and often performs better than either value or policy iteration

Outline

Introduction

Formal definition

Value iteration

Policy iteration

Linear programming for MDPs

Linear programming solution methods

A slightly less frequently described method for MDPs: solution via linear programming

Basic idea: we can capture the constraint

$$V(s) \geq R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, |s, a) V(s')$$

via the set of $|\mathcal{A}|$ linear constraints

$$V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s'), \ \ \forall a \in \mathcal{A}$$

Now consider the linear program

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & \sum_s V(s) \\ \text{subject to} \quad & V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s'), \;\; \forall a \in \mathcal{A}, s \in \mathcal{S} \end{aligned}$$

Theorem: the optimal value of this linear program will be $V^\star$

Proof: Suppose there exists some $s \in \mathcal{S}$ with

$$V(s) > R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s')$$

Then we can construct a solution with only $V(s)$ changed to make this an equality: this will have a lower objective value, but be feasible, since it can only decrease right hand side for other constraints

Comments on LP formulation

In objective, we can optimize any positive linear function of $V(s)$ and the result above still holds

If we optimize

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & \sum_{s} d(s) V(s) \\ \text{subject to} \quad & V(s) \geq R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s'), \;\; \forall a \in \mathcal{A}, s \in \mathcal{S} \end{aligned}$$

where $d(s)$ is a distribution over states, then objective is equal to total expected accumluted reward when beginning at a state drawn from this distribution

Adding dual variables $\mu(s,a)$ for each constraint, dual problem is (after some simplification)

$$\begin{array}{ll} \underset{\mu(s,a)}{\text{maximize}} & \sum_{s \in \mathcal{S}} R(s) \sum_{a \in \mathcal{A}} \mu(s,a) \\ \text{subject to} & \sum_{a \in \mathcal{A}} \mu(s',a) = d(s') + \gamma \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} P(s'|s,a)\mu(s,a) \;\; \forall s' \in \mathcal{S} \\ & \mu(s,a) \geq 0 \end{array}$$

These have the interpretation that

$$\mu(s,a) = \sum_{t=0}^{\infty} \gamma^t P(S_t = s, A_t = a)$$

i.e., they are discounted state-action counts, which directly encode the optimal policy

$$\pi^\star(s) = \max_{a \in \mathcal{A}} \mu(s,a)$$

LP versus value/policy iteration

Some surprising connections between LP formulation and standard value and policy iteration algorithms: e.g. a certain form of dual simplex is equivalent to policy iteration

Typically, best specialized MDP algorithms (e.g. modified policy iteration) are faster than general LP algorithms, but the LP formulation provides a number of connections to other methods, and has also been the basis for much work in approximate large-scale MDP solutions (e.g., de Farias and Van Roy, 2003)