

# Markov Decision Processes

Noel Welsh

11 November 2010

- Applicant visitor day seeks robot demonstrators for exciting half hour of robotic demonstrations.

- We have studied the simplest decision making problem, the bandit problem.
- In the bandit problem is very restrictive.
- Today we're going to look at partially relaxing the restrictions.

# Extending the Bandit Problem

- Two extensions to the bandit problem:
  - State
  - Observations (in a restricted sense)
- The model we'll arrive at is called a Markov decision process (MDP).

- What does state mean?
- Informally, actions have consequences.
- In (the MDP extension to) the bandit problem, this means rewards for an action are not always given by the same distribution.
- Rewards are a function of *state* and action.

- The process does not transition randomly between states.
  - If it did, this would be (like) the *non-stochastic bandit* I mentioned briefly last lecture.
- Rather, the robot's actions influence the choice of next state.
  - So, the robot has some control over its environment.
  - In the general case, actions do not decide the next state, but influence its probability.

# The MDP Formalism

- Two additions (so far) to the bandit model:
  - Rewards a function of action *and* state
  - Next state a function of action and current state.
- From this we get the Markov decision process (MDP):
  - $\mathcal{S}$  is the set of states that the environment may be in.
  - $\mathcal{A}$  is the set of actions that the agent can perform in the environment.
  - $\mathcal{T}$  is the transition function, which determines the next state given the current state, and the chosen action.  $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  defines the probability  $\mathcal{T}(s, a, s') = P(s'|a, s)$
  - $\rho$  is the reward function that maps state and action pairs to reward:  
 $\rho : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
  - $\mathcal{I}$  is the initial distribution over states  $\mathcal{I} : \mathcal{S} \rightarrow [0, 1]$  defining the probability  $\mathcal{I}(s) = P(s)$

# The Reward Model

- In the bandit model there was a distribution over rewards.
- Here we've said that rewards are a deterministic function of state and action ( $\rho : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ).
- What's up?
- Can view this as a simplification. Rewards are a deterministic function of state and action *in expectation* (or average, or mean).
- So long as we have a model (we'll talk about this later) this doesn't make any difference.

# The Markov Property

- Distribution over next state determined only by the current state and action.
- Past irrelevant given the current state.
- This is the *Markov property*. The future is independent of the past given the present.
- Makes computation easier.

# What is a State?

- What exactly is a state?
- Can reverse Markov property to define it.
- A state is sufficient information about the world such that the Markov property holds.
- Hence, a state is a *sufficient statistic*. It summarises everything we need to know.
- Applications in inferring models from data.

# What is State Take 2

- If the set of states  $\mathcal{S}$  has no structure we call it *enumerated*.
- If states are represented as tuples of values it is called a *factored* representation.
- If the states are represented by a graph of relations the model is a *relational* MDP.
- Basic theory deals with enumerated representations.
- Specialised algorithms apply to factored and relational representations, but you can still use the fundamental enumerated algorithms with suitable modifications.

- In the MDP observations uniquely identify the state. This is a *fully observable* model.
  - So often just remove observations from the model. Consider states to be observations.
- If this isn't true, the world is *partially observable*.
- The corresponding model is the partially observable Markov decision process (POMDP).
- Do robots operate in fully or partially observable worlds?

- Why use MDPs in robotics. Two good reasons:
- Engineering trade-off. MDPs are much easier to work with. For modelling purposes a MDP may be sufficient.
- Sensors (and processing) actually getting very good.
- E.g. FAB-MAP [Cummins and Newman(2008)], given thousands of images can match that were taken at the same spot with very little error.
- Maybe an assumption of full observability isn't so bad?

# Other Limitations of MDPs

- We only consider *discrete* state and action spaces. *Continuous* spaces more appropriate for low-level control (but you said you don't want to learn about this).
- Not considering more than one agent. These models known as *Markov games*.
- Time is somewhat problematic.

# Solving a MDP

- What do we want to do with a MDP?
- Find an (near-)optimal way to act.
- What does that mean? We need a measure of quality – that is, an *objective function*.
- In the following, we assume we have a model available. Known as *planning*. If we get time we'll look at what we do when we don't have a model available, which is called *reinforcement learning*.

- In the bandit problem we used regret as our objective function.
- In MDPs the traditional objective function is *value*.
- More recent work tends to use regret, but we're kicking it old skool today.

- A *policy* is a way of acting.
- Always consider the value of some policy.
- By the Markov property the current state and action completely determine the probability of the next state and reward.
- Thus the current state encapsulates everything necessary to choose the best action.
- Thus we can assume a *stationary* or *reactive policy*.
- Formally a stationary policy  $\pi$  is a function from states to actions;  
 $\pi : \mathcal{S} \rightarrow \mathcal{A}$ .

# Value, Again

- Consider only a single time step.
- The environment is in state  $s$
- The *one-step value* for a policy  $\pi$  is the reward it receives for taking a single action.
- We write it as  $V_1^\pi(s)$ .
- What is it?

# Optimal One-step Value

- We can easily work out the best one-step value  $V_1^*(\mathbf{s})$ .
- What is it?
- From this we can derive a one-step optimal policy

$$\pi_1^*(\mathbf{s}) = \operatorname{argmax}_{a \in \mathcal{A}} \rho(\mathbf{s}, a) \quad (1)$$

# Exploration and Exploitation

- One-step value is not the true value.
- Why not?

# The True Value

- Given a policy  $\pi$  we can write the *value* of the policy as:

$$\begin{aligned} V^\pi(\mathbf{s}) = & \rho(\mathbf{s}, \pi(\mathbf{s})) + \left( \gamma \sum_{\mathbf{s}' \in \mathcal{S}} T(\mathbf{s}, \pi(\mathbf{s}), \mathbf{s}') \rho(\mathbf{s}', \pi(\mathbf{s}')) \right) \\ & + \left( \gamma^2 \sum_{\mathbf{s}' \in \mathcal{S}} \sum_{\mathbf{s}'' \in \mathcal{S}} T(\mathbf{s}, \pi(\mathbf{s}), \mathbf{s}') T(\mathbf{s}', \pi(\mathbf{s}'), \mathbf{s}'') \rho(\mathbf{s}'', \pi(\mathbf{s}'')) \right) \\ & + \dots \end{aligned} \tag{2}$$

- Confused?

# Understanding Value

- We include the transition function  $\mathcal{T}$  to weight the future rewards by the probability of achieving them.
- The discount factor  $\gamma$  serves two purposes: it keeps the sum of rewards bounded, and it determines how we weight future rewards relative to immediate rewards.
  - If the discount factor is zero we completely ignore future rewards.
  - If the discount factor is one, future rewards, no matter how far away they are, are considered equal to immediate rewards.
  - Values between zero and one given proportionally more weight to future rewards.
  - Values greater than one give more weight to rewards the further away they are – a nonsensical choice in any realistic setting.

- Note that the definition of value has a recursive structure. This is more apparent in the standard definition of value, which is:

$$V^\pi(\mathbf{s}) = \rho(\mathbf{s}, \pi(\mathbf{s})) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} (\mathcal{I}(\mathbf{s}, \pi(\mathbf{s}), \mathbf{s}') V^\pi(\mathbf{s}')) \quad (3)$$

- The previous definition gives the value of a given policy. We can instead maximise value, yielding:

$$V^*(s) = \max_{a \in \mathcal{A}} \left( \rho(s, a) + \gamma \sum_{s' \in \mathcal{S}} (T(s, a, s') V^*(s')) \right) \quad (4)$$

- Given  $V^*$  we can define the optimal policy as that which chooses the action giving the highest value:

$$\pi^*(\mathbf{s}) = \operatorname{argmax}_{a \in \mathcal{A}} \left( \rho(\mathbf{s}, a) + \gamma \sum_{s' \in \mathcal{S}} (\mathcal{T}(\mathbf{s}, a, s') V^*(s')) \right) \quad (5)$$

# Bellman Equations

- The optimal value (or policy) equation is known as a *Bellman equation*.
- It is a set of simultaneous equations.
- We solved simultaneous equations in high school, so what's the big deal?
- The maximisation operator means the equations are not linear so Gaussian elimination cannot be used to solve them.
- Luckily, there are simple algorithms for solving MDPs.

# Value Iteration Overview

- *Value iteration* is a classic algorithm for solving MDPs.
- The inner loop of value iteration takes a  $t - 1$  step estimate of the value function and extends it a single time step into the future.
- The outer loop just repeats the inner loop till the maximum change in value is less than some threshold.

# Value Iteration Inner Loop

---

**Function** oneValueIteration( $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{I} \rangle, V_{t-1}$ ) Inner loop of value iteration for an MDP

---

```
for  $s \in \mathcal{S}$  do  
  for  $a \in \mathcal{A}$  do  
     $Q_t(s, a) \leftarrow \rho(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_{t-1}(s')$ ;  
  end  
   $\pi_t(s) \leftarrow \operatorname{argmax}_a Q_t(s, a)$ ;  
   $V_t(s) \leftarrow Q_t(s, \pi_t(s))$ ;  
end  
return ( $V_t, \pi_t$ );
```

---

# Value Iteration Outer Loop

---

**Function** `valueIteration( $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{I} \rangle, \epsilon$ )` Value iteration for an MDP

---

$t \leftarrow 0$  ;

$V_0 \leftarrow 0$  ;

**repeat**

$t \leftarrow t + 1$  ;

$(V_t, \pi_t) \leftarrow \text{oneValueIteration}(V_{t-1})$  ;

**until**  $\max_{\mathcal{S}} |V_t(\mathbf{s}) - V_{t-1}(\mathbf{s})| < \epsilon$  ;

**return**  $(V_t, \pi_t)$

---



Mark Cummins and Paul Newman.

FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance.

*The International Journal of Robotics Research*, 27(6):647–665, 2008.