Sequential Decision Making
Dynamic programming

Christos Dimitrakakis

Intelligent Autonomous Systems, Ivl, University of Amsterdam, The Netherlands

March 18, 2008
Introduction

Some examples

Dynamic programming

Summary
The purpose of this 'lecture'

Basic concepts

- Refresh memory.
- Present the MDP setting.
- Define optimality.
- Categorize planning tasks

Algorithms

- Introduce basic planning algorithms.
- Promote intuition about their relationships.
- Discuss their applicability.

Ultimate goal

A firm foundation in reasoning and planning under uncertainty.
Preliminaries
Markov decision processes
Value functions and optimality

Introduction

Some examples
Shortest-path problems
Continuing problems
Episodic, finite, infinite?

Dynamic programming
Introduction
Backwards induction
Iterative Methods
Policy evaluation
Value iteration
Policy iteration

Summary
Lessons learnt
Learning from reinforcement...
Bibliography
Preliminaries

Variables

- Environment $\mu \in \mathcal{M}$
- States $s_t \in S$.
- Actions $a_t \in A$.
- A reward $r_t \in \mathbb{R}$.
- A policy $\pi \in \mathcal{P}$.

Notation

- Probabilities $P(x|y, z) \equiv z(x|y)$.
- Expectations $E(x|y, z)$
- Sometimes $P(a_t = a|\cdot)$ will be used for clarity.
- i.e. $\pi_t(a|s) = P(a_t = a|s_t = s, \pi_t)$
Markov decision processes

The setting

We are in some dynamic environment $\mu$, where at each time step $t$ we observe

- States $s_t \in S$.
- Actions $a_t \in A$.
- A reward $r_t \in \mathbb{R}$.

\[ P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = P(s_{t+1}|s_t, a_t, \mu) \]  
\[ p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1}|s_{t+1}, s_t, a_t, \mu) \]
Markov decision processes

The setting
We are in some dynamic environment \( \mu \), where at each time step \( t \) we observe

- States \( s_t \in S \).
- Actions \( a_t \in A \).
- A reward \( r_t \in \mathbb{R} \).

\[
\begin{align*}
P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) &= P(s_{t+1}|s_t, a_t, \mu) \\
p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) &= p(r_{t+1}|s_{t+1}, s_t, a_t, \mu) \\
p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) &= p(r_{t+1}|s_t, a_t, \mu)
\end{align*}
\]
Markov decision processes

The setting

We are in some dynamic environment $\mu$, where at each time step $t$ we observe

- States $s_t \in S$.
- Actions $a_t \in A$.
- A reward $r_t \in \mathbb{R}$.

\[
P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = P(s_{t+1}|s_t, a_t, \mu) \tag{1}
\]
\[
p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1}|s_{t+1}, s_t, a_t, \mu) \tag{2}
\]
\[
p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1}|s_t, a_t, \mu) \tag{3}
\]
\[
p(r_{t+1}|s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1}|s_{t+1}, \mu) \tag{4}
\]
Markov decision processes

Controlling the environment
We wish to control the environment according to some (for now undefined) optimality criterion.

The agent
The agent is fully defined by its policy $\pi$. This induces a probability distribution on actions and states.

\[
P(a_t|s_t, a_{t-2}, s_{t-1}, a_{t-2}, \ldots, \pi, \mu) = P(a_t|s_t, \pi)
\]
Markov decision processes

The induced Markov chain
Together with the policy $\pi$ and the model $\mu$, we induce a Markov chain on states.

\[
P(s_{t+1}|s_t, \pi, \mu) = \sum_{a \in A} P(s_{t+1}|a_t = a, s_t, \pi, \mu) P(a_t = a|s_t, \pi) \quad (6a)
\]

\[
P(s_{t+k}|s_t, \pi, \mu) = \sum_s P(s_{t+k}|s_{t+k-1} = s, \pi, \mu) P(s_{t+k-1}|s_t, \pi, \mu) \quad (6b)
\]

Note: $\lim_{k \to \infty} P(s_{t+k} = s|s_t, \pi, \mu)$ is the stationary distribution.
The induced Markov chain
Together with the policy $\pi$ and the model $\mu$, we induce a Markov chain on states.

\[ P(s_{t+1}|s_t, \pi, \mu) = \sum_{a \in A} P(s_{t+1}|a_t = a, s_t, \pi, \mu) P(a_t = a|s_t, \pi) \]  

\[ P(s_{t+k}|s_t, \pi, \mu) = \sum_{s} P(s_{t+k}|s_{t+k-1} = s, \pi, \mu) P(s_{t+k-1}|s_t, \pi, \mu) \]  

Note: $\lim_{k \to \infty} P(s_{t+k} = s|s_t, \pi, \mu)$ is the stationary distribution.
Planning

The goal in reinforcement learning
To maximise a function of future rewards.

Finite horizon
We are only interested in rewards up to a fixed point in time.

Infinite horizon
We are interested in all rewards.
Value functions

The return / utility

The agent’s goal is to maximize the return (Too many Rs, switching to $U$).

For example the utility given a policy $\pi$ and an MDP $\mu$

$$U_{t,\mu}^\pi(\cdot) \triangleq \mathbb{E}(U|\cdot, \pi, \mu) = \mathbb{E}\left(\sum_{k=1}^{T} \gamma^k r_{t+k} | \cdot, \pi, \mu\right)$$

$$= \sum_{k=1}^{T} \gamma^k \sum_{i\in S} \mathbb{E}[r_{t+k}|s_{t+k}=i, \mu] \mathbb{P}(s_{t+k}=i|\cdot, \pi, \mu)$$

Can in principle be calculated from (6).

The value functions

$$V_t^\pi(s) \triangleq \sum_{a\in A} U_{t,\mu}^\pi(s, a) \pi(a|s)$$

$$Q_t^\pi(s, a) \triangleq U_{t,\mu}^\pi(s, a)$$

Special case: $T \to \infty$, $V_t^\pi(s) = V^\pi(s)$. 
An optimal policy

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The recursion

\[
V^\pi_t(s) = g(t) \mathbb{E}[r_{t+1}|s_t=s, \pi] + \sum_{k=2}^{T-t} g(t + k) \mathbb{E}[r_{t+k}|s_t=s, a_t=a, \pi, \mu] 
\]

\[
= g(t) \mathbb{E}[r_{t+1}|s_t=s, \pi] + \sum_{i \in S} V^\pi_{t+1}(i) \mu(s_{t+1}=i|s_t=s, \pi). 
\]

- The current stage’s value is just the next reward plus the next stage’s value.

- See also the Hamilton-Jacobi-Bellman equation in optimal control.
Greedy policies

The 1-step greedy policy
The 1-step-greedy policy with respect to a given value function can be expressed as

\[
\pi(a|s) = \begin{cases} 
1, & a = \arg \max_{a'} Q(s, a') \\
0, & \text{otherwise}
\end{cases}
\]  

(13)

The optimal policy
The 1-step-greedy policy with respect to the optimal value function is optimal.

Naive solution
Evaluate all policies, select \( \pi^* : V^{\pi^*}(s) \geq V^\pi(s) \ \forall s \in S \).

Clever solutions
- Directly estimate \( V^* \).
- Iteratively improve \( \pi \).
Introduction

Some examples

- Shortest-path problems
- Continuing problems
- Episodic, finite, infinite?

Dynamic programming

- Introduction
- Backwards induction
- Iterative Methods
  - Policy evaluation
  - Value iteration
  - Policy iteration

Summary

- Lessons learnt
- Learning from reinforcement...
- Bibliography
Problem types

Planning with...

- Finite vs Infinite horizon
- Discounted vs Undiscounted rewards
- Certain vs Uncertain knowledge
- Expected vs worst-case utility functions

Environments

- Deterministic ↔ Stochastic
- Episodic ↔ Continuing
- Observable ↔ Hidden state
- Statistical ↔ Adversarial
Deterministic shortest-path problems

Properties

- \( g(t) = 1, \ T \to \infty. \)
- \( r_t = -1 \) unless \( s_t = X \), in which case \( r_t = 0. \)
- \( \mu(s_{t+1} = X|s_t = X) = 1. \)
- \( A = \{\text{North, South, East, West}\} \)
- Transitions are deterministic and walls block.

What is the shortest path to the destination from any point?
Stochastic shortest path problem, with a pit

Properties

- $g(t) = 1$, $T \to \infty$.
- $r_t = -1$, but $r_t = 0$ at $X$ and $-100$ at $O$ and episode ends.
- $\mu(s_{t+1} = X|s_t = X) = 1$.
- $A = \{\text{North, South, East, West}\}$
- Moves to a random direction with probability $\theta$. Walls block.

For what value of $\theta$ is it better to take the dangerous shortcut? (However, if we want to take into account risk explicitly we must modify the agent’s utility function)
Continuing stochastic MDPs

Inventory management

- There are $K$ storage locations.
- Each place can store $n_i$ items.
- At each time-step there is a probability $\phi_i$ that a client try to buy an item from location $i$, $\sum_i \phi_i \leq 1$. If there is an item available, you gain reward 1.
- Action 1: ordering $u$ units of stock, for paying $c(u)$.
- Action 2: move $u$ units of stock from one location $i$ to another, $j$, for a cost $\psi_{ij}(u)$.

An easy special case

- $K = 1$.
- There is one type of item only.
- Orders are placed and received every $n$ timesteps.
Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every $m$ timesteps.
- Each time-step a client arrives with probability $\phi$.

Properties

- The state set .
- The action set .
- The transition probabilities
Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every $m$ timesteps.
- Each time-step a client arrives with probability $\phi$.

Properties

- The state set is the number of items we have: $S = \{0, 1, \ldots, n\}$.
- The action set .
- The transition probabilities
Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every $m$ timesteps.
- Each time-step a client arrives with probability $\phi$.

Properties

- The state set is the number of items we have: $S = \{0, 1, \ldots, n\}$.
- The action set $A = \{0, 1, \ldots, n\}$ since we can order from nothing up to $n$ items.
- The transition probabilities
Inventory management

An easy special case

▶ \( K = 1. \)
▶ Deliveries happen once every \( m \) timesteps.
▶ Each time-step a client arrives with probability \( \phi \).

Properties

▶ The state set is the number of items we have: \( S = \{0, 1, \ldots, n\} \).
▶ The action set \( A = \{0, 1, \ldots, n\} \) since we can order from nothing up to \( n \) items.
▶ The transition probabilities \( P(s'|s, a) = \binom{m}{d} \phi^d (1 - \phi)^{m-d} \), where \( d = s + a - s' \), for \( s + a \leq n \).
Episodic, finite, infinite?

Shortest path problems

- Episodic tasks with infinite horizon, $-1$ reward everywhere, but $0$ in absorbing state.
- Continuing tasks with $0$ reward everywhere, but $>0$ in goal state, $\gamma \in (0, 1)$, state reset after goal.
- Equivalent if optimal policy is the same.
Preliminaries
Markov decision processes
Value functions and optimality

Introduction

Some examples
Shortest-path problems
Continuing problems
Episodic, finite, infinite?

Dynamic programming
Introduction
Backwards induction
Iterative Methods
Policy evaluation
Value iteration
Policy iteration

Summary
Lessons learnt
Learning from reinforcement...
Bibliography
Introduction

Why dynamic programming?

- Programming means finding a solution.
- i.e. linear programming.
- Dynamic because we find solution to dynamical problems.
- Direct relation to control theory.
The shortest-path problem revisited

Properties

- $\gamma = 1, \ T \rightarrow \infty$.
- $r_t = -1$ unless $s_t = X$, in which case $r_t = 0$.
- The length of the shortest path from $s$ equals the negative value of the optimal policy.
- Also called cost-to-go.
- Remember Dijkstra’s algorithm?
Backwards induction I

- If we know the value of the last state, we can calculate the values of its predecessors.
- The value of $s_{T-1}^i$ is the reward obtained by moving from $s_{T-1}^i$ to $s_T$, plus the value of $s_T$. 

![Diagram](image-url)
Backwards induction II

All $w, x, y, z < 0$, and reward $e < 0$ of staying at the same state, apart from $A$.  

All $w, x, y, z$
Backwards induction III

Backwards induction in deterministic environments

Input $\mu$, $S_T$.
Initialise $V_T(s)$, for all $s \in S_T$.

for $n = T - 1, T - 2, \ldots, t$ do
  for $s \in S_n$ do
    $a_n^*(s) = \arg\max_a \mathbb{E}(r|s'_s, a, s, \mu) + V^*_{n+1}(s'_s, a)$
    $V_n^*(s) = \mathbb{E}(r|s'_s, a_n^*(s), s, \mu) + V^*_{n+1}(s'_s, a_n^*(s))$
  end for
end for

Notes

- $s'_s, a$ is the state that occurs if we take $a$ in $s$.
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!
Backwards induction III

Backwards induction in deterministic environments

Input $\mu$, $S_T$.
Initialise $V_T(s)$, for all $s \in S_T$.

for $n = T - 1, T - 2, \ldots, t$ do
  for $s \in S_n$ do
    $a_n^*(s) = \arg\max_a \sum_{s' \in S_{n+1}} \mu(s'|s, a) E(r|s', s, \mu) + V_{n+1}^*(s')$
    $V_n(s)^* = \sum_{s' \in S_{n+1}} \mu(s'|s, a_n^*(s)) E(r|s', s, \mu) + V_{n+1}^*(s')$
  end for
end for

Notes

▶ $\mu(s'|s, a)$ is an indicator function
▶ Because we always know the optimal choice at the last step, we can find the optimal policy directly!
Backwards induction in deterministic environments

Input $\mu$, $S_T$.
Initialise $V_T(s)$, for all $s \in S_T$.

for $n = T - 1, T - 2, \ldots, t$ do
  for $s \in S_n$ do
    $a_n^*(s) = \arg\max_a \sum_{s' \in S_{n+1}} \mu(s'|s, a) \mathbb{E}(r|s', s, \mu) + V_{n+1}^*(s')$
    $V_n(s)^* = \sum_{s' \in S_{n+1}} \mu(s'|s, a_n^*(s)) \mathbb{E}(r|s', s, \mu) + V_{n+1}^*(s')$
  end for
end for

Notes

▶ $\mu(s'|s, a)$ is an indicator function
▶ Nothing apparently stops $\mu(s'|s, a)$ from being a distribution
▶ So, what happens in stochastic environments?
Backwards induction IV: Stochastic problems

Almost as before, but state depends stochastically on actions, i.e. \( \mu(s_{t+1}=A|s_t=B, a_t=a) \)

The backup operators

\[
V^\pi_n(s) = \sum_{s'} [\mu(s'|s, \pi) E(r|s', s) + V^\pi_{n+1}(s')] \\
V^*_n(s) = \max_a \sum_{s'} \mu(s'|s, a) [E(r|s', s) + V^*_n+1(s')] 
\]  

(14)  

(15)
Backwards induction $V$

Policy evaluation with Backwards induction

Input $\pi$, $\mu$, $S_T$.
Initialise $V_T(s)$, for all $s \in S_T$.

for $n = T - 1, T - 2, \ldots, t$ do
  for $s \in S_n$ do
    $V^\pi_n(s) = \sum_{s' \in S_{n+1}} \mu(s'|s, \pi) \left[ E(r|s', s, \mu) + V^\pi_{n+1}(s') \right]$  
  end for
end for

Notes

- $\mu(s'|s, \pi) = \sum_a \mu(s'|s, a) \pi(a|s)$.
- Finite horizon problems only, or approximations to finite horizon (i.e. lookahead in game trees).
- Hey, it works for stochastic problems too! (By marginalizing over states)
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!
- Can be used with estimates of the value function.
Backwards induction V

Finding the optimal policy with Backwards induction

Input $\mu$, $S_T$.
Initialise $V_T(s)$, for all $s \in S_T$.

for $n = T - 1, T - 2, \ldots, t$ do
  for $s \in S_n$ do
    $a_n^*(s) = \arg \max_a \mu(s'|s, a)[E(r|s', s, \mu) + V_{n+1}^*(s')]$
    $V_n(s)^* = \sum_{s' \in S_{n+1}} \mu(s'|s, a_n^*)[E(r|s', s, \mu) + V_{n+1}^*(s')]$
  end for
end for

Notes

▶ Finite horizon problems only, or approximations to finite horizon (i.e. lookahead in game trees).
▶ Hey, it works for stochastic problems too! (By marginalizing over states)
▶ Because we always know the optimal choice at the last step, we can find the optimal policy directly!
▶ Can be used with estimates of the value function.
What happens when the horizon is infinite in stochastic shortest path problems?

▶ Episodic tasks still terminate with probability one for proper policies.
▶ Assumption: there exists at least one proper policy.
▶ Assumption: Every improper policy has negatively infinite value for at least one state.
Introduction

Some examples
- Shortest-path problems
- Continuing problems
- Episodic, finite, infinite?

Dynamic programming
- Introduction
- Backwards induction
  - Iterative Methods
    - Policy evaluation
    - Value iteration
    - Policy iteration

Summary
- Lessons learnt
- Learning from reinforcement...
- Bibliography
Why evaluate a policy?
We can always generate a better policy given the value function of any policy!

**Theorem (Policy improvement)**

Let some policy $\pi \in \mathcal{P}$. If $\pi'(a|s) = 1$ for $a = \arg\max_a Q^\pi(s, a)$ and 0 otherwise, then

$$V^{\pi'}(s) \geq V^\pi(s), \quad \forall s \in S$$
Policy improvement theorem

Theorem (Policy improvement)

Let some policy $\pi \in \mathcal{P}$. If $\pi'(a|s) = 1$ for $a = \arg \max_a Q^\pi(s, a)$ and 0 otherwise, then

$$V^\pi'(s) \geq V^\pi(s), \quad \forall s \in S$$

Proof.

Let $\pi_k$ be the policy which execute $\pi'$ for $k$ steps and then reverts to $\pi$. Then $\pi = \pi_0$, $\pi' = \lim_{k \to \infty} \pi_k$, and we have

$$V^\pi(s_t) = \sum_{a_t} \pi(a_t|s_t)Q^\pi(s, a)$$

$$\leq \max_{a_t} Q^\pi(s, a) = \max_{a_t} \left[ \sum_{s_{t+1}} \mu(s_{t+1}|s_t, a_t) V^\pi(s_{t+1}) \right] = V^{\pi_1}(s_t).$$

Similarly, we show that $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all $s$. Then $V^\pi \leq V^{\pi_1}(s) \leq V^{\pi_k}(s) \leq V^{\pi_{k+1}}(s)$... and so

$V^\pi'(s) = \lim_{k \to \infty} V^{\pi_k}(s) \geq V^\pi(s)$. \qed
Iterative policy evaluation

Policy Evaluation

Input $\pi$, $\mu$ and $\hat{V}_0$.

$n = 0$.

repeat

$n = n + 1$

for $s \in S$ do

$\hat{V}_n(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in S} \mu(s'|s,a)[E(r|s',\mu) + \gamma \hat{V}_{n-1}(s')]$

end for

until $\|\hat{V}_n - \hat{V}_{n-1}\|_\infty < \theta$

Notes

▶ Arbitrary initialization.
▶ $V^\pi$, $\hat{V}_n \in \mathbb{R}^{|S|}$,
▶ $\lim_{n \to \infty} \hat{V}_n = V^\pi$, if the limit exists.
▶ Can be done in-place as well.
Policy evaluation example I

Random policy evaluation.

0 iterations
Policy evaluation example 1

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

1 iteration

Random policy evaluation.
Policy evaluation example I

10 iterations

Random policy evaluation.
Policy evaluation example 1

99 iterations

Random policy evaluation.
Policy evaluation example I

Greedy policy with respect to value function of random policy

Random policy evaluation.
Policy evaluation example II

Random policy evaluation.
Policy evaluation example II

Random policy evaluation.
Policy evaluation example II

Random policy evaluation.
Random policy evaluation.
Value Iteration

Input $\mu$.
$\hat{V}_0(s) = 0$ for all $s \in S$.
$n = 0$.

repeat
  $n = n + 1$
  for $s \in S$ do
    $\hat{V}_n(s) = \max_{a \in A} \sum_{s' \in S} \mu(s'|s,a)[E(r|s',\mu) + \gamma \hat{V}_{n-1}(s')]$
  end for
until $\|\hat{V}_n - \hat{V}_{n-1}\|_{\infty} < \theta$

Notes

- No reason to assume a fixed policy, convergence holds.
- $\lim_{n \to \infty} \hat{V}_n = V^*$.
- Equivalent to backwards induction as horizon $\to \infty$.
- This is because $\lim T \to \infty V_t^\pi(s) = V^\pi(s)$ for all $t$. 
Value iteration example

iter: 0
Value iteration example

![Value iteration example](image-url)
Value iteration example

iter: 10
Value iteration example

<table>
<thead>
<tr>
<th>-1.4</th>
<th>-1.3</th>
<th>-1.2</th>
<th>-1.1</th>
<th>-1.0</th>
<th>-0.9</th>
<th>-0.8</th>
<th>-0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.6</td>
<td>-1.5</td>
<td>-1.4</td>
<td>-0.4</td>
<td>-0.3</td>
<td>-0.4</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>-1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.8</td>
<td>-1.9</td>
<td>-2.0</td>
<td>-0.2</td>
<td>-0.1</td>
<td>-0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

iter: 100

→ → → → → → → ↓
↑ ↑ ↑ ↑ → ↓ ← ←
↑ ↑ → ↑ ↓ ↓ ← ←
↑ ↑ ↑ ↑ ← ↓ ← ←
↑ ↑ ↑ ↑ ← ← ← ←
Introduction

Some examples
  Shortest-path problems
  Continuing problems
  Episodic, finite, infinite?

Dynamic programming
  Introduction
  Backwards induction
    Iterative Methods
      Policy evaluation
      Value iteration
      Policy iteration

Summary
  Lessons learnt
  Learning from reinforcement...
  Bibliography
Policy iteration I

Policy Iteration

Input $\pi$, $\mu$.
repeat
  Evaluate $V^\pi$.
  $\pi' : \pi'(s) = \arg\max_a Q^\pi(s, a)$
until $\arg\max_a Q^{\pi'}(s, a) = V^\pi(s)$ for all $s$

Theorem (Policy iteration)

The policy iteration algorithm generates an improving sequence of proper policies, i.e.

$V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s), \quad \forall k > 0, s \in S$

and terminates with an optimal policy, i.e. $\lim_{k \to \infty} V^{\pi_k} = V^\ast$.

Remark (Policy iteration termination)

If $\pi_k$ is not optimal, then $\exists s \in S$:

$V^{\pi_{k+1}}(s) > V^{\pi_k}(s)$.

Conversely, if no such $s$ exists, $\pi_k$ is optimal and we terminate.
Policy iteration II

The evaluation step

- It can be done exactly by solving the linear equations. (Proper policy iteration)
- We can use a limited number $n$ of policy evaluation iterations (Modified policy iteration algorithm).
- These can be initialised from the last evaluation.
- If we use just $n = 1$, then the method is identical to value iteration.
- If we use $n \to \infty$, then we have proper policy iteration.

Other methods

- Asynchronous policy iteration.
- Multistage lookahead policy iteration.
- See [1], section 2.2 for more details.
- See [3], Chapters 4,5,6 for detailed theory.
Preliminaries
Markov decision processes
Value functions and optimality

Introduction

Some examples
Shortest-path problems
Continuing problems
Episodic, finite, infinite?

Dynamic programming
Introduction
Backwards induction
Iterative Methods
  Policy evaluation
  Value iteration
  Policy iteration

Summary
Lessons learnt
Learning from reinforcement...
Bibliography
Lessons learnt

Planning with a known model

▶ Find the optimal policy given model and objective.
▶ Bellman recursion is the basis of dynamic programming.
▶ Easy to solve for finite-horizon problems or episodic tasks.
▶ Stochasticity does not make the problem significantly harder.
▶ Infinite-horizon continuing problems harder, but tractable.

Things to think about

▶ Would iterative methods be better than backwards induction?
▶ How does it depend on the problem?
▶ Does the discount factor have any effect?
▶ How can backwards induction be applied to iterative problems and vice-versa?
Learning from reinforcement...

Bandit problems

- $\gamma \in [0, 1]$, $T > 0$.
- $|S| = 1$.
- Rewards are random with expectation $E[r_t|a_t, \mu]$.
- If $\mu$ known, trivial: $a^* = \arg \max_a E[r_t|a_t = a, \mu]$, for all $t, \gamma$.
- If $\mu$ is unknown, can be intractable.
- Simplest case of learning from reinforcement.
Further reading


