

FOURTH APPLICATION UNIT OF Q520: MARKOV DECISION PROCESSES

Larry Moss

Indiana University

Lecture notes for Q520

MARKOV DECISION PROCESSES

An MDP comes to us with
a **state set** S , whose elements are s_1, \dots, s_n
an **action set** Act , whose elements are α, β, \dots
a **probability transition function** $go(s, \alpha, t)$,
such that for each s and α , $\sum_t go(s, \alpha, t) = 1$.
an **immediate reward function** $reward : S \rightarrow \mathbb{R}$.

MARKOV DECISION PROCESSES

A **policy** is a function $\pi : S \rightarrow Act$.

A policy π then gives us a **transition matrix** P^π ,
whose i, j entry is $go(s, \pi(s), t)$,
the probability that if we start in the i th state
and do the action $\pi(s)$,
we then go to the j th state.

MARKOV DECISION PROCESSES

A **policy** is a function $\pi : S \rightarrow Act$.

A policy π then gives us a **transition matrix** P^π ,

whose i, j entry is $go(s, \pi(s), t)$,

the probability that if we start in the i th state

and do the action $\pi(s)$,

we then go to the j th state.

Since we are using matrices to make more elegant

and readable calculations,

we also organize the rewards into a **reward vector**.

It is a column vector called R .

MDP NOTATIONS

We then get an **expected long-term value** function

$$V^\pi : \mathcal{S} \rightarrow \mathbb{R}$$

defined in terms of itself by the matrix equation

$$V^\pi = R + \gamma P^\pi V^\pi$$

This is just a succinct way of putting down a big system of equations, one for each state s :

$$V^\pi(s) = \text{reward}(s) + \gamma \sum_t \text{go}(s, \pi(s), t) V^\pi(t).$$

Think of this system of equations as one where the $V^\pi(s)$ are **variables whose values we want to know**.

MDP NOTATIONS

$$V^\pi = R + \gamma P^\pi V^\pi$$

The system has a **unique solution** given by

$$V^\pi = (I - \gamma P^\pi)^{-1} R$$

For large MDPs, one cannot solve the system explicitly, and so must turn to other methods to find or approximate V^π .

A RUNNING EXAMPLE MDP: THE LITTLE PRINCE

-1 <i>a</i> ↑	-1 <i>b</i> ↑	great food 10 <i>c</i> ↑
-1 <i>d</i> ↑	sand -5 <i>e</i> ↑	monster -4 <i>f</i> ↑
avg food 5 <i>g</i> ↑	-1 <i>h</i> ↑	-1 <i>i</i> ↑

The set of states is $\{a, \dots, i\}$. The rewards are shown.

The possible actions are \uparrow , \downarrow , \leftarrow , and \Rightarrow .

So one policy π_0 is shown.

The go function works as described in the handout.

For example, $go(b, \uparrow, h) = .8$, $go(b, \uparrow, a) = .1$, and $go(b, \uparrow, c) = .1$.

A RUNNING EXAMPLE MDP: THE LITTLE PRINCE

$$R = \begin{bmatrix} -1 \\ -1 \\ 10 \\ -1 \\ -5 \\ -4 \\ 5 \\ -1 \\ -1 \end{bmatrix} \quad V^{\pi_0} = \begin{bmatrix} 3.7 \\ -3.7 \\ 11.1 \\ 1.3 \\ -7.2 \\ 3.4 \\ 5.6 \\ -5.6 \\ 1.5 \end{bmatrix}$$

I have rounded the values of V^{π_0} .

THE BELLMAN EQUATION

For every MDP, we also have an **absolute max long-term value** function

$$V^* : \mathcal{S} \rightarrow \mathbb{R}$$

“defined” by the **Bellman equation**.

$$V^*(s) = \text{reward}(s) + \gamma \max_{\alpha} \sum_t \text{go}(s, \alpha, t) V^*(t)$$

A subtle but important point is that at first glance, it is not obvious that the Bellman equation actually has a solution at all! For that matter, it is not obvious that it has a unique solution, either. We'll return to these points later, and in the homework.

COMPARING VECTORS AND POLICIES

If we have two vectors of the same length, say A and B , we say that $A \geq B$ if for all i , $A_i \geq B_i$.

We compare two policies π_1 and π_2 by looking at their long-term expected values.

So we define

$$\pi_1 \geq \pi_2 \quad \text{if} \quad V^{\pi_1} \geq V^{\pi_2}.$$

This just means that for all states s , $V^{\pi_1}(s) \geq V^{\pi_2}(s)$.

We would also write

$$\pi_1 \equiv \pi_2 \quad \text{if both} \quad \pi_2 \geq \pi_1 \text{ and } \pi_1 \geq \pi_2.$$

That is, for all states s , $V^{\pi_1}(s) = V^{\pi_2}(s)$.

Two policies could be equivalent (\equiv) and yet different (not literally the same function).

POLICY IMPROVEMENT

Take a policy π .

Compute its expected long-term value function V^π .

Then, defined an “improved” policy π' as follows:

for each state s , let

$$\pi'(s) = \operatorname{argmax}_\alpha \sum_t \operatorname{go}(s, \alpha, t) V^\pi(t).$$

($\operatorname{argmax}_\alpha \dots$ means “give me the α that maximizes \dots ”).

In case of a tie, pick any action that gives a maximal value.

The idea is that π' asks about the best **single-step action** that could be taken **before acting just like π forevermore**.

We say “improved” with quotes because it’s not obvious at all that π' is better than π .

The proof of this is one of the main theoretical points in this lecture.

POLICY IMPROVEMENT

We have defined π' by

$$\operatorname{argmax}_{\alpha} \sum_t \operatorname{go}(s, \alpha, t) V^{\pi}(t).$$

And so π' is a policy; that is, a function from states to actions. Since $\pi(s)$ is one of the actions α considered in the argmax ,

$$\sum_t \operatorname{go}(s, \pi'(s), t) V^{\pi}(t) \geq \sum_t \operatorname{go}(s, \pi(s), t) V^{\pi}(t)$$

POLICY IMPROVEMENT IN OUR EXAMPLE

$$V^{\pi_0} = \begin{bmatrix} 3.7 \\ -3.7 \\ 11.1 \\ 1.3 \\ -7.2 \\ 3.4 \\ 5.6 \\ -5.6 \\ 1.5 \end{bmatrix}$$

For example, to decide on $\pi'(a)$, we compare four things.

$$(\text{go}(a, \uparrow, g) \cdot V^{\pi_0}(g)) + (\text{go}(a, \uparrow, b) \cdot V^{\pi_0}(b)) + (\text{go}(a, \uparrow, c) \cdot V^{\pi_0}(c)) =$$

$$(.8)(5.6) + (.1)(-3.7) + (.1)(11.1) = 5.2,$$

...

$$(\text{go}(a, \leftarrow, c) \cdot V^{\pi_0}(c)) + (\text{go}(a, \leftarrow, d) \cdot V^{\pi_0}(d)) + (\text{go}(a, \leftarrow, g) \cdot V^{\pi_0}(g)) =$$

$$(.8)(11.1) + (.1)(1.3) + (.1)(5.6) = 9.5.$$

The winner for these is \leftarrow . So we set $\pi'(a) = \leftarrow$.

POLICY IMPROVEMENT

To summarize:

Given π , we used its associated long-term value function V^π to define a new policy π' .

For each state s ,

$$\sum_t \text{go}(s, \pi'(s), t) V^\pi(t) \geq \sum_t \text{go}(s, \pi(s), t) V^\pi(t)$$

This is a little cleaner if we write it as a matrix inequality

$$P^{\pi'} V^\pi \geq P^\pi V^\pi.$$

THE POLICY IMPROVEMENT THEOREM

Let π and π' be **any two policies**, and assume that

$$P^{\pi'} V^{\pi} \geq P^{\pi} V^{\pi}.$$

Then

$$V^{\pi'} \geq V^{\pi}.$$

That is, π' is at least as good a policy as π :

$$\pi' \geq \pi.$$

THE POLICY IMPROVEMENT THEOREM: PROOF

To prove this, our hypothesis $P^{\pi'} V^{\pi} \geq P^{\pi} V^{\pi}$ implies that

$$R + \gamma P^{\pi'} V^{\pi} \geq R + \gamma P^{\pi} V^{\pi} = V^{\pi}$$

This means that

$$R \geq V^{\pi} - \gamma P^{\pi'} V^{\pi} = (I - \gamma P^{\pi'}) V^{\pi}.$$

THE POLICY IMPROVEMENT THEOREM: PROOF

To prove this, our hypothesis $P^{\pi'} V^{\pi} \geq P^{\pi} V^{\pi}$ implies that

$$R + \gamma P^{\pi'} V^{\pi} \geq R + \gamma P^{\pi} V^{\pi} = V^{\pi}$$

This means that

$$R \geq V^{\pi} - \gamma P^{\pi'} V^{\pi} = (I - \gamma P^{\pi'}) V^{\pi}.$$

It then follows that

$$\begin{aligned} V^{\pi'} &= (I - \gamma P^{\pi'})^{-1} R && \text{this is true for any policy} \\ &\geq (I - \gamma P^{\pi'})^{-1} (I - \gamma P^{\pi'}) V^{\pi} && \text{see the next slide} \\ &= V^{\pi} && \text{because inverses cancel} \end{aligned}$$

And we're done!

This beautiful argument uses the matrix inverse to shorten an otherwise-complicated appeal to an infinite sum.

A BACKGROUND FACT

We used an important but fairly basic fact:

If x and y are column vectors,

P is any **stochastic** matrix (**the columns add to 1**)

$0 < \gamma < 1$,

and $x \geq y$,

then

$$(I - \gamma P)^{-1}x \geq (I - \gamma P)^{-1}y.$$

A BACKGROUND FACT

We used an important but fairly basic fact:

If x and y are column vectors,

P is any **stochastic** matrix (**the columns add to 1**)

$0 < \gamma < 1$,

and $x \geq y$,

then

$$(I - \gamma P)^{-1}x \geq (I - \gamma P)^{-1}y.$$

To see this, we'll show that

$$(I - \gamma P)^{-1}(x - y) \geq 0.$$

Write z for $x - y$. All entries in z are ≥ 0 .

$$(I - \gamma P)^{-1}z = z + \gamma Pz + \cdots + \gamma^n P^n z + \cdots$$

and all the entries in all the vectors on the right are non-negative.

THE POLICY ITERATION ALGORITHM

Start with any policy π_0 .

Define π_1 to be the improvement of π_0 as we have seen it:

$$\pi_1(s) = \operatorname{argmax}_{\alpha} \sum_t g_{\alpha}(s, \alpha, t) V^{\pi_0}(t).$$

If $\pi_0 \equiv \pi_1$, then **stop**: π_0 is **unimprovable**.

(Recall that $\pi_0 \equiv \pi_1$ means that for all s , $V^{\pi_0}(s) = V^{\pi_1}(s)$.)

THE POLICY ITERATION ALGORITHM

Start with any policy π_0 .

Define π_1 to be the improvement of π_0 as we have seen it:

$$\pi_1(s) = \operatorname{argmax}_{\alpha} \sum_t g_{\alpha}(s, \alpha, t) V^{\pi_0}(t).$$

If $\pi_0 \equiv \pi_1$, then **stop**: π_0 is **unimprovable**.

Otherwise, **by the Policy Improvement Theorem** we have $\pi_0 < \pi_1$.

Then define π_2 from π_1 the same way as above.

If $\pi_2 \equiv \pi_1$, then **stop**: π_1 is **unimprovable**.

THE POLICY ITERATION ALGORITHM

Start with any policy π_0 .

Define π_1 to be the improvement of π_0 as we have seen it:

$$\pi_1(s) = \operatorname{argmax}_{\alpha} \sum_t g_{\alpha}(s, \alpha, t) V^{\pi_0}(t).$$

If $\pi_0 \equiv \pi_1$, then **stop**: π_0 is **unimprovable**.

Otherwise, **by the Policy Improvement Theorem** we have $\pi_0 < \pi_1$.

Then define π_2 from π_1 the same way as above.

If $\pi_2 \equiv \pi_1$, then **stop**: π_1 is **unimprovable**.

Repeat the step above, getting

$$\pi_0 < \pi_1 < \pi_2 < \dots$$

Now, there are only finitely many policies: an upper bound is $|S|^{|Act|}$.

So we cannot go on improving forever!

At some point, we have $\pi_{n+1} \equiv \pi_n$, and then this one is **unimprovable**.

THE POLICY ITERATION ALGORITHM: EXAMPLE

In the Little Prince example, we started with π_0 “go north”.

Then we improved to π_1 , and π_2 .

π_2 turned out to be unimprovable.

It is a good idea to check that $\pi_0 < \pi_1 < \pi_2$, just as our results predict.

A RUNNING EXAMPLE MDP: THE LITTLE PRINCE

-1 a \Leftarrow 37.8	-1 b \Rightarrow 32.5	great food 10 c \Uparrow 38.8
-1 d \Downarrow 30.8	sand -5 e \Uparrow 29.5	monster -4 f \Uparrow 26.6
avg food 5 g \Downarrow 33.4	-1 h \Leftarrow 33.8	-1 i \Downarrow 30.5

An optimal policy π is shown, along with the value of V^π in each state.

THE POLICY ITERATION ALGORITHM

There still is a loose end:

True enough, the algorithm gives us an “unimprovable” policy.

But how do we know that “unimprovable” policies are any good?

And do they really help us solve the Bellman equation for V^* ?

THE POLICY ITERATION ALGORITHM

There still is a loose end:

True enough, the algorithm gives us an “unimprovable” policy.

But how do we know that “unimprovable” policies are any good?

And do they really help us solve the Bellman equation for V^* ?

Much of the results from a homework problem:

If π_1 and π_2 are **any** two policies,
 then there is a **joint improvement** σ such that for all states s ,
 $V^\sigma(s) \geq V^{\pi_1}(s)$ and $V^\sigma(s) \geq V^{\pi_2}(s)$.

So you will get to reason for yourselves some of the wonderful facts about unimprovable policies.

CONSEQUENCES

If π_1 and π_2 are **any** two policies,
then there is a **joint improvement** σ such that for all states s ,
 $V^\sigma(s) \geq V^{\pi_1}(s)$ and $V^\sigma(s) \geq V^{\pi_2}(s)$.

Every unimprovable policy π has the property that **no**
policy does strictly better than π on *any* state.

If π_1 and π_2 are unimprovable, then $\pi_1 \equiv \pi_2$.

If π is unimprovable, then V^π satisfies the Bellman equation.

For any MDP, the Bellman equation has a unique solution.

VALUE ITERATION

We have already seen **policy iteration**:

Given π , **compute** V^π and then let π' be defined by

$$\pi'(s) = \operatorname{argmax}_\alpha \sum_t \operatorname{go}(s, \alpha, t) V^\pi(t).$$

This computation step could be expensive.

And anyways, the whole set-up is not consonant with many of our goals in AI or Cog Sci modeling.

Before we turn to **reinforcement learning**, I want to (re)turn to an alternative to policy iteration.

VALUE ITERATION

Let $V_0(s) = \text{reward}(s)$.

Given $V_n(s)$, let

$$V_{n+1}(s) = R(s) + \gamma \left(\operatorname{argmax}_{\alpha} \sum_t \text{go}(s, \alpha, t) V_n(t) \right)$$

This generates an infinite sequence of **value functions**

$$V_0, V_1, V_2, \dots, V_n, \dots$$

Unlike the ones in policy iteration, these in general **do not converge**.

VALUE ITERATION

However, the iterates

$$V_0, V_1, V_2, \dots, V_n, \dots$$

get closer and closer to V^* , the unique solution of the Bellman equation

$$V^*(s) = \text{reward}(s) + \gamma \max_{\alpha} \sum_t \text{go}(s, \alpha, t) V^*(t)$$

And in fact, if

$$|V_{n+1} - V_n(s)| < \epsilon \frac{(1 - \gamma)}{\gamma},$$

then $|V_{n+1} - V^*| < \epsilon$.

This means that eventually, V_n is going to be “good enough”.

And we can use *any* value function V go generate a policy:

$$\pi^V(s) = \text{argmax}_{\alpha} \sum_t \text{go}(s, \alpha, t) V(t)$$

If V_n is “close to V^* ”, then π^V will be “close to optimal.”

Q-LEARNING

But all of this is still far from the situation that we really want to explore:

an agent who **does not know** the model!

To explore this, we need a way of estimating V^* **by sampling as we go**.

There are several general ways of doing this;

they all involve tradeoffs between **exploration** and **exploitation**.

Perhaps the easiest to get into is **Q-learning**.

A QUIOTE

The basic paradigm of reinforcement learning is as follows: The learning agent observes an input state or input pattern, it produces an output signal (most commonly thought of as an "action" or "control signal"), and then it receives a scalar "reward" or "reinforcement" feedback signal from the environment indicating how good or bad its output was. The goal of learning is to generate the optimal actions leading to maximal reward. In many cases the reward is also delayed (i.e., is given at the end of a long sequence of inputs and outputs). In this case the learner has to solve what is known as the "temporal credit assignment" problem (i.e., it must figure out how to apportion credit and blame to each of the various inputs and outputs leading to the ultimate final reward signal).

The reinforcement learning paradigm has held great intuitive appeal and has attracted considerable interest for many years because of the notion of the learner being able to learn on its own, without the aid of an intelligent "teacher," from its own experience at attempting to perform a task. In contrast, in the more commonly employed paradigm of supervised learning, a "teacher signal" is required that explicitly tells the learner what the correct output is for every input pattern.

Q-LEARNING

In our setting, rewards are given in the states, not on the actions. So these definitions will differ from what we find in Mitchell's book, for example.

The idea is that $Q(s, \alpha)$ is the maximum cumulative reward one could get after starting in state s and doing α as the first action. So

$$Q(s, \alpha) = \gamma \sum_t \text{go}(s, \alpha, t) V^*(t)$$

But

$$V^*(s) = \text{reward}(s) + \max_{\alpha} Q(s, \alpha).$$

And so

$$Q(s, \alpha) = \gamma \left(\sum_t \text{go}(s, \alpha, t) (\text{reward}(t) + \max_{\alpha'} Q(t, \alpha')) \right)$$

Q-LEARNING

Now the basic idea is to turn this equation

$$Q(s, \alpha) = \gamma \left(\sum_t \text{go}(s, \alpha, t) (\text{reward}(t) + \max_{\alpha'} Q(t, \alpha')) \right)$$

into an **update rule**, so that we can have functions

$$Q_0, Q_1, \dots, Q_n, \dots$$

Then there should be some theoretical results telling us something like

After we have found Q_n for some large enough n , it gives us something close to an optimal policy.

In this setting, one must update **all** of the values $Q(s, \alpha)$, even the ones that (from the outside) look silly.

Q-LEARNING

Further one must update things in a wise way, generalizing what's done on the k -arm bandit problem.

The text chapter by Mitchell has a discussion of the main results here.