Lecture 9: Reinforcement Learning

Simon Parsons

School of Computer Science
University of Lincoln

Version 1.1

Cover


FACE COVERINGS
HELP KEEP OUR
COMMUNITY SAFE
#LoveLincoln
ONE
COMMUNITY
UNIVERSITY OF
LINCOLN

Test


TAKE REGULAR
LATERAL FLOW TESTS
AND PROTECT YOUR
FRIENDS
#LoveLincoln
ONE
COMMUNITY
UNIVERSITY OF
LINCOLN

Vaccinate

UNIVERSITY OF
LINCOLN

Isolate


IF YOU HAVE SYMPTOMS, SELF ISOLATE
#LoveLincoln
ONE COMMUNITY
UNIVERSITY OF LINCOLN

Today

- Quantifying uncertainty
  - Intro to probability, etc.
- Probabilistic reasoning
  - Bayesian networks, causal inference, etc.
- Making complex decisions
  - Intro to complex decision making
  - Markov Decision Processes
  - Reinforcement learning
- Reasoning over time
  - Hidden Markov Models, etc.
- Strategic reasoning
  - Game theory

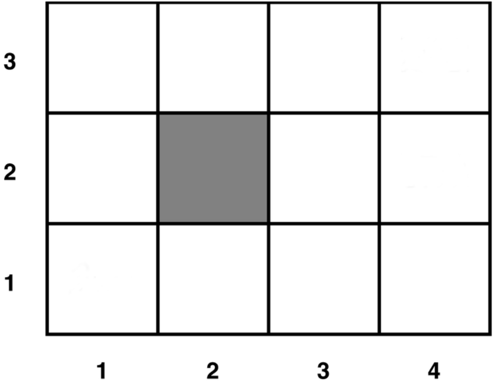
UNIVERSITY OF LINCOLN

Recap

- Markov Decision Processes (MDPs) is/are a method that can be used to make decisions when actions are non-deterministic.
- Decisions about what to do.
- Sequential decision problems
  Sequences of decisions.
- Use probability and expectation.

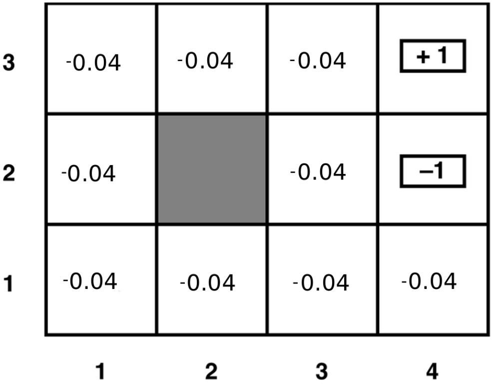
Recap II

- MDP components.
- State space:



Recap III

- MDP components.
- Reward:

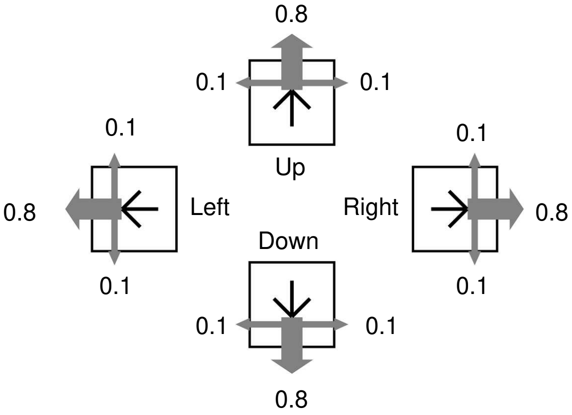


- Intrinsic value of each state.

UNIVERSITY OF LINCOLN

Recap IV

- MDP components.
- Motion model:



- For every state/action combination, how likely are we to get to every other state.

Recap IV

- From these we compute *utilities*, *value iteration*.

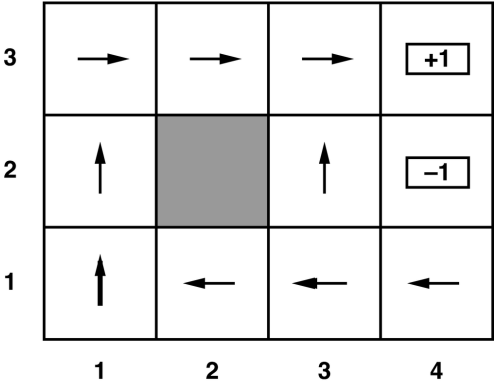
3	**0.812**	**0.868**	**0.918**	**+ 1**
2	**0.762**		**0.660**	**–1**
1	**0.705**	**0.655**	**0.611**	**0.388**
	1	**2**	**3**	**4**

- How good each state is in the context of the whole world.

UNIVERSITY OF LINCOLN

Recap V

- From utilities we extract the *policy*

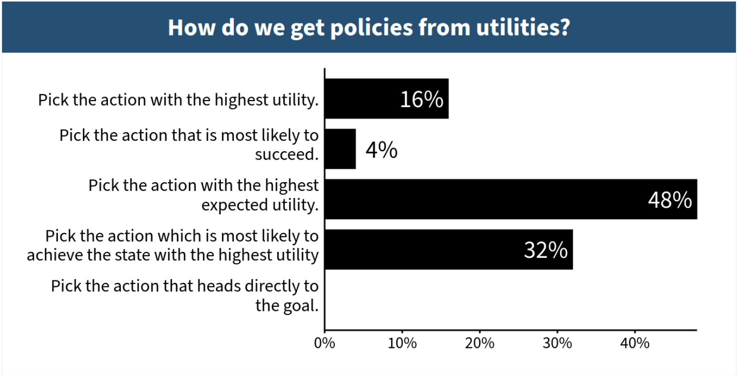


- What to do in each state.

Recap VI: Utility to policy?

- How do we get policies from utilities?

Recap VII: Utility to policy?



- The right answer is by picking the action with the highest expected utility.

Recap VIII: Policy iteration

- The utilities are only helpful in getting us to the policy
- We can also go direct — *policy iteration*.

Limitations of MDPs?

(Pendleton Ward/Cartoon Network)

UNIVERSITY OF LINCOLN

Partially observable MDPs

- MDPs made the assumption that the environment was fully observable.
    - Agent always knows what state it is in.
- The optimal policy only depends on the current state.
- Not the case in the real world.
    - We only have a belief about the current state.
- POMDPs extend the model to deal with partial observability.

UNIVERSITY OF LINCOLN

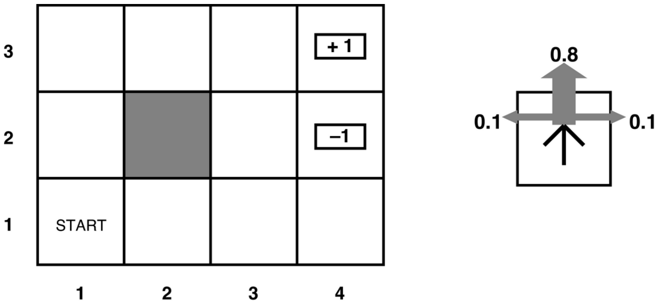
Partially observable MDPs

- Basic addition to the MDP model is the sensor model:

$$P(e|s)$$

  probability of perceiving evidence e in state s.
- As a result of noise in the sensor model, the agent only has a belief about which state it is in.
- Probability distribution over the possible states.
- “The world is a POMDP”

Partially observable MDPs



(a) (b)

$\mathbf{P}(S) : P(s_{1,1}) = 0.05, P(s_{1,2}) = 0.01, \ldots$

Partially observable MDPs

- The agent can compute its current belief as the conditional probability distribution over the states given the sequence of actions and percepts so far.

Partially observable MDPs

- The agent can compute its current belief as the conditional probability distribution over the states given the sequence of actions and percepts so far.
- We will come across this task again in temporal probabilistic reasoning.
- Filtering.
- Computing the state that matches best with a stream of evidence.

Partially observable MDPs

- If $b(s)$ was the distribution before an action and an observation, then afterwards the distribution is:

$$b'(s') = \alpha P(e|s') \sum_{s} P(s'|s,a)b(s)$$

- Everything in a POMDP hinges on the belief state b.
  - Including the optimal action.
- Indeed, the optimal policy is a mapping $\pi^*(b)$ from beliefs to actions.
  "If you think you are next to the wall, turn left"
- The agent executes the optimal action given its beliefs, receives a percept e and then recomputes the belief state.

UNIVERSITY OF LINCOLN

Partially observable MDPs

- The big issue in solving POMDPs is that beliefs are continuous.
- When we solved MDPs, we could search through the set of possible actions in each state to find the best.
- To solve a POMDP, we need to look through the possible actions for each belief state.
  But belief is continuous, so there are a lot of belief states.
- Exact solutions to POMDPs are intractable for even small problems (like the example we have been using).
- Need (once again) to use approximate techniques.

UNIVERSITY OF LINCOLN

Reinforcement Learning

- What happens if we don't know the components of the (PO)MDP?
  - State space
  - Motion model
  - Rewards and utilities.
- We learn them.
- This is the domain of *reinforcement learning* (RL).
- Learning through trial and error.

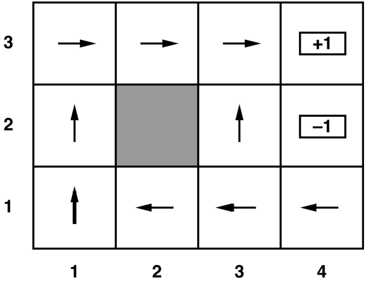
UNIVERSITY OF LINCOLN

RL in a nutshell

(Pendleton Ward/Cartoon Network)

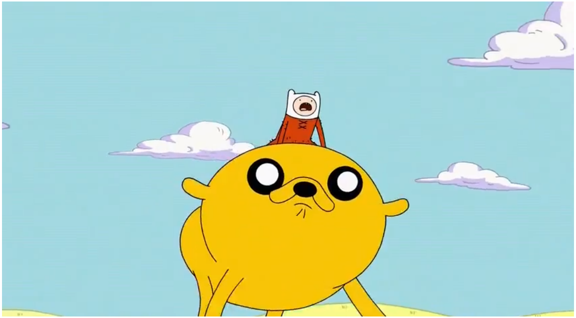
UNIVERSITY OF LINCOLN

Passive learning



- Agent learns utility $U^{\pi}(s)$ by carrying out runs through the environment, following some policy π.

Passive learning

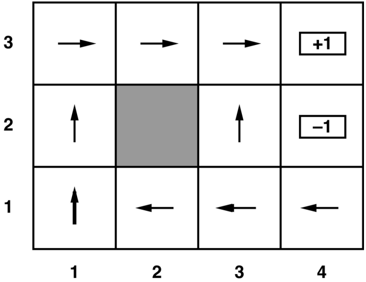


(Pendleton Ward/Cartoon Network)

- In passive reinforcement learning the agent's policy is fixed.
- Agent doesn't make a choice about how to act.

UNIVERSITY OF LINCOLN

Passive learning



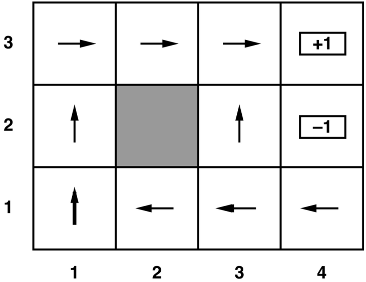
- We think of agents performing *runs* through the state space, like this:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow$$
$$(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \ldots$$

- Actions are dictated by the policy.

UNIVERSITY OF LINCOLN

Passive learning

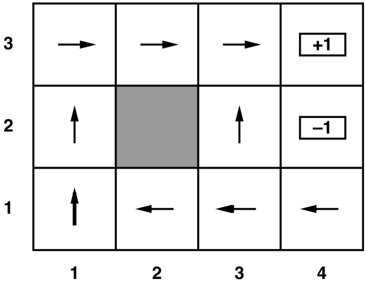


- We think of agents performing *runs* through the state space, like this:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow \\ (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \ldots$$

- Note the rewards attached to each state.

Passive learning



- We assume that the rewards are directly experienced by the agent.

Passive learning

- The utility $U^{\pi}(s)$ of a state s under policy π is the expected sum of the (discounted) rewards obtained when following π.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t)\right]$$

  where S is the state reached at t from s when executing π.
- So if we run the policy for long enough, we will compute the utility of the states from the onward rewards.

UNIVERSITY OF LINCOLN

Direct utility estimation

- We can estimate the utility of a state by the rewards generated along the run from that state.
- Direct utility estimation.
- Each run gives us one or more samples for the utility of a state.

Direct utility estimation

- Given the run:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow$$
$$(1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

  a sample utility of $(1,1)$ from the run above is the sum of the rewards all the way to a goal state.
  - 0.72 in this case.
- The same run will produce two samples for $(1,2)$ and $(1,3)$.
  - 0.76 and 0.84
  - 0.8 and 0.88
- (Here we set the discount to 1).
- We then average the samples so far to get the current estimated value.

UNIVERSITY OF LINCOLN

Estimate utilities

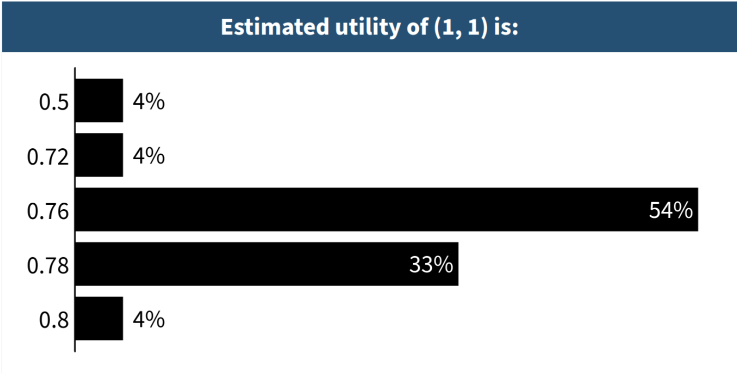
- Given this run:

$$(1,1)_{-0.04} \rightarrow (1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1}$$

What is the estimated utility of state $(1,1)$?

UNIVERSITY OF LINCOLN

Estimate utilities



- 0.78 is the right answer.
- (If you got 0.76, it is because you didn't average over the two estimates.)

Direct utility estimation

- So we know how to calculate:
  - Rewards
  - Utilities

UNIVERSITY OF LINCOLN

Probability estimation

- As the agent moves it can calculate a sample estimate of $P(s'|s, \pi(s))$
- Each time it moves it creates a new sample for one state.
- Given:

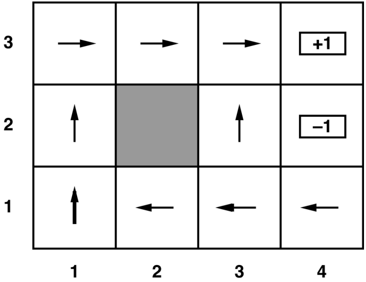
$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

we get:

$$\begin{aligned} P((1,2)|(1,1), Up) &= 1 \\ P((1,2)|(1,3), Right) &= 0.5 \\ P((2,3)|(1,3), Right) &= 0.5 \\ &\vdots \end{aligned}$$

UNIVERSITY OF LINCOLN

Estimate probabilities

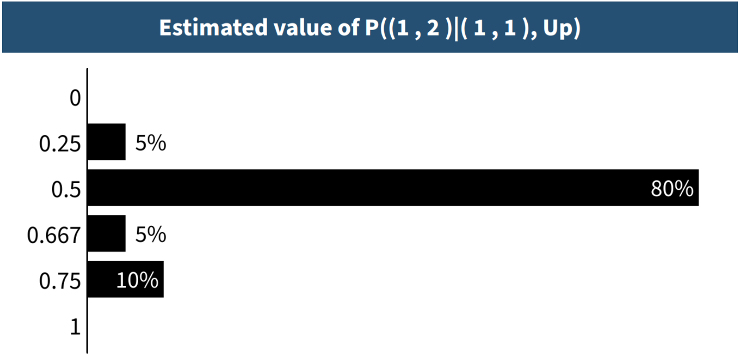


- Given this run under the policy above:

$$(1,1)_{-0.04} \rightarrow (1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04}$$
$$\rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1}$$

What is the estimated value of $P((1,2)|(1,1), Up)$?

Estimate probabilities



- 0.5 is the right answer.

Probability estimation

- So we can calculate:
  - Rewards
  - Utilities
  - Probabilities

Probability estimation

- So we know how to calculate:
  - Rewards
  - Utilities
  - Probabilities
- None of it is much more complicated than counting.

Direct utility estimation

- So, over time, the agent builds up estimates of:

3	**0.812**	**0.868**	**0.918**	**+ 1**
2	**0.762**		**0.660**	**–1**
1	**0.705**	**0.655**	**0.611**	**0.388**
	1	**2**	**3**	**4**

and $P(s'|s, \pi(s))$, for every s, s' for the given $\pi(s)$.

Passive learning

- What does a solution look like?
- A list of states s_i.
- Each state has a utility estimate associated with it $U(s)$.
- Each state has an action associated with it, $\pi(s)$.
- Each state action pair has a probability distribution:

$$\mathbf{P}(S'|s, \pi(s))$$

over the states S' that it gets to from s by doing $\pi(s)$.

- (May not encounter every state.)

UNIVERSITY OF LINCOLN

Aside

- We haven't said where the states come from.
- Sometimes we will know what they are.
- (Implied by the existence of the policy).
- Other times we will learn the states as we go along.
- Basic requirement is that we can distinguish states from one another.

Passive learning

- How does an agent decide what to do?
- Then the agent just computes each step using one-step lookahead on the expected value of actions.
- Picks the action a with the greatest expected utility.
- The resulting policy may well differ from π.
- Its data on actions will be limited because it has only been trying π.

UNIVERSITY OF LINCOLN

Passive learning

- Has to vary π if it wants to learn the full space.
- But is this worth it?
- After all, once we have an idea of how to act to get to the goal, is more learning justified?
- Tradeoff *exploration* and *exploitation*

UNIVERSITY OF LINCOLN

Tradeoff

Tradeoff

- But explore less over time.

Problem with direct utility estimation

- Treats utilities of states as independent.
- But we know that they are connected.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

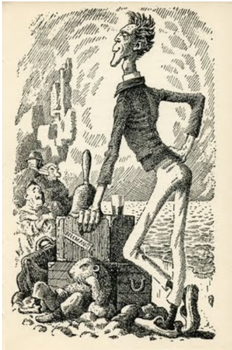
- Ignoring the connection means that learning may converge slowly.
- So another approach to utility estimation: adaptive dynamic programming.
- Still doing passive reinforcement learning.
- But doing it *smarter*.

UNIVERSITY OF LINCOLN

Adaptive dynamic programming

- We can improve on direct utility estimation by applying a version of the Bellman equation.
- This says:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s, a) U(s')$$



(Mervyn Peake)

- The utility of a state is the reward for being in that state plus the expected discounted reward of being in the next state.

UNIVERSITY OF LINCOLN

Adaptive dynamic programming

- What we actually have here:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- We know π, so we know what action we will carry out.
- We have π, because it is passive learning.

UNIVERSITY OF LINCOLN

Adaptive dynamic programming

- So, how to we benefit from applying the Bellman equation?
- Bellman states a constraint on utilities, but what does that mean in practice?
- Two approaches:
  1. Directly solve the Bellman equations
  2. Apply value iteration
- (Rather like policy iteration.)

UNIVERSITY OF LINCOLN

Solving the Bellman equations

- The fixed policy version of the Bellman equation is:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- This is just a set of simultaneous equations.
  (Unlike the standard version of the Bellman equation, there is no max to complicate things.)
- Can just plug results into an LP solver
- Updates all the utilities of all the states where we have experienced the transitions.

Solving the Bellman equations

- Note that updated values are *estimates*.
- They are no better than the estimated values of utility and probability we had before.
- We just get quicker convergence because the utilities are consistent.

Using value iteration

- Can also use value iteration to update the utilities we have for each state.
- Update using:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U_i(s')$$

until convergence.

- Again, the results are still estimates, and no better than the estimates we got from direct estimation or solving the Bellman equations.

UNIVERSITY OF LINCOLN

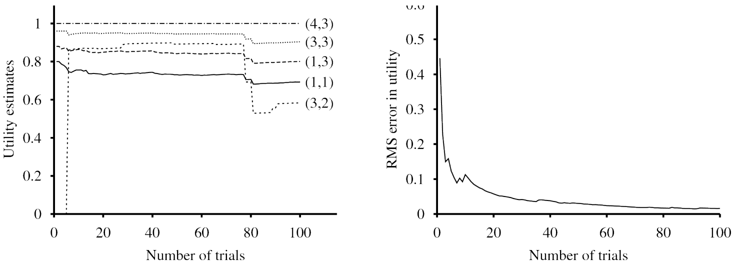
Adaptive dynamic programming

- In all cases:
  1. Direct utility estimation
  2. ADP: solving Bellman equations
  3. ADP: applying value iteration

  what we get out depends on what we put in.
- The quality of the utility estimates will depend on how well we have *explored* the space.
- Roughly this is how many times we have encountered each state.

Adaptive dynamic programming

- Results:



- Typically quicker than direct utility estimation.
- Error is for $U(1,1)$.

UNIVERSITY OF LINCOLN

Adaptive dynamic programming

- Still passive learning, so a solution is as before:
- A list of states s_i.
- Each state has a utility estimate associated with it $U(s)$.
- Each state has an action associated with it, $\pi(s)$.
- Each state action pair has a probability distribution:

$$\mathbf{P}(S'|s, \pi(s))$$

over the states S' that it gets to from s by doing $\pi(s)$.

After learning

- Now, to get the utilities, the agent started with a fixed policy, so it always knew what action to take.
- It used this to get utilities.
- Having gotten the utilities, it could use them to choose actions.
    - Just picks the action with the best expected utility in a given state.
- However, there is a problem with doing this.

UNIVERSITY OF LINCOLN

Problems

- The transition model is a maximum likelihood estimate (Just the sample average).
- Maximum likelihood models tend to overfit.
- Maximum likelihood action selection can be dangerous.

UNIVERSITY OF LINCOLN

Problems

- Might not yet have experienced the bad effects of an action:

(Calista Condo/South Jersey Times)

- Maybe your autonomous car learnt that running a red light saves time.

UNIVERSITY OF LINCOLN

Problems

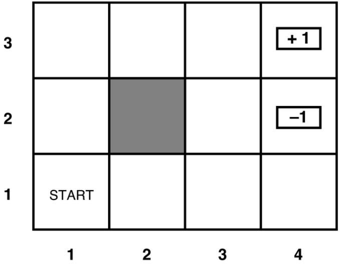
- Of course, this kind of over-reliance on not-fully-explored state/action spaces is what people do all the time.
- There is no way to be sure that the action your maximum likelihood-based reinforcement learner is picking doesn't have possible bad outcomes.
- Usually tackle this by ensuring wide exploration.

UNIVERSITY OF LINCOLN

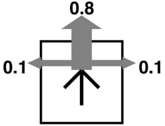
Temporal difference learning

- In the previous approach we used the fact that we are learning in the context of an MDP.
- Another way to use Bellman (= constraints between states).
- Use the observed transitions to adjust the utilities of the states.
- Let's look at an example.

Temporal difference learning



(a)



(b)

- Consider this trajectory:

$$(1,1)_{-0.04} \overset{Up}{\rightarrow} (1,2)_{-0.04} \overset{Up}{\rightarrow} (1,3)_{-0.04} \overset{Right}{\rightarrow} (2,3)_{-0.04} \overset{Right}{\rightarrow}$$

$$(3,3)_{-0.04} \overset{Right}{\rightarrow} (3,2)_{-0.04} \overset{Up}{\rightarrow} (3,3)_{-0.04} \overset{Right}{\rightarrow} (4,3)_{+1}$$

UNIVERSITY OF LINCOLN

Temporal difference learning

- Consider the transition from $(1,3)$ to $(2,3)$.

$$(1,1)_{-0.04} \overset{Up}{\rightarrow} (1,2)_{-0.04} \overset{Up}{\rightarrow} (1,3)_{-0.04} \overset{Right}{\rightarrow} (1,2)_{-0.04} \overset{Up}{\rightarrow}$$
$$(1,3)_{-0.04} \overset{Right}{\rightarrow} (2,3)_{-0.04} \overset{Right}{\rightarrow} (3,3)_{-0.04} \overset{Right}{\rightarrow} (4,3)_{+1}$$

- Assume that we have utility estimates:

$$U^{\pi}(1,3) = 0.84$$
$$U^{\pi}(2,3) = 0.92$$

(These are the values from the run we considered earlier.)

- These results should be linked by a Bellman-type update.

Temporal difference learning

- In other words, we should expect:

$$U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$$

  and so $U^{\pi}(1,3) = 0.88$
- Currently have $U^{\pi}(1,3) = 0.84$
- So maybe the current estimate is too low.

UNIVERSITY OF LINCOLN

Temporal difference learning

- In other words, we should expect:

$$U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$$

  and so $U^{\pi}(1,3) = 0.88$
- Currently have $U^{\pi}(1,3) = 0.84$
- So maybe the current estimate is too low.
- Now generalise the idea.

UNIVERSITY OF LINCOLN

Temporal difference learning

- The *temporal difference* update for a transition from s to s' is:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

- α is a *learning rate*.
  - Controls how quickly we update the utility when we have new information.
- The rule is called "temporal difference" because the update occurs between successive states.

Temporal difference learning

- Compare the ADP update:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

with the TD update:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

UNIVERSITY OF LINCOLN

Temporal difference learning

- The ADP update:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

  can be read as a statement about the stopping condition.
- No change in values when both sides of the equation are equal.
- Connects the utility of s with that of ***all*** its successor states.

Temporal difference learning

- The TD update:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

  only adjusts the utility of s with that of a single successor s'.
- Yet to manages to reach the same equilibrium.
- How?

UNIVERSITY OF LINCOLN

Temporal difference learning

- TD update:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

- In the long run, the transition from s to s' will happen *exactly* in proportion to:

$$P(s'|s, \pi(s))$$

- So $U^{\pi}(s')$ will be averaged into $U^{\pi}(s)$ exactly the right amount.

UNIVERSITY OF LINCOLN

Temporal difference learning

- Well, ok, that is a bit of a simplification.
- We need to adjust α over time.
- Need to ensure that:

$$\sum_{t=1}^{\infty} \alpha(t) = \infty$$

$$\sum_{t=1}^{\infty} \alpha^2(t) < \infty$$

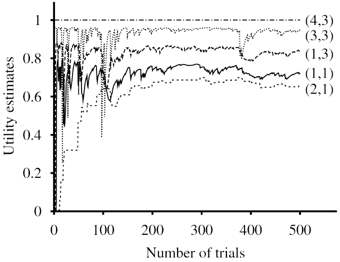
  to guarantee convergence.
- This is satisfied if:

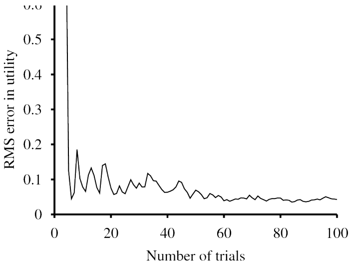
$$\alpha(t) = O(1/t)$$

UNIVERSITY OF LINCOLN

Temporal difference learning

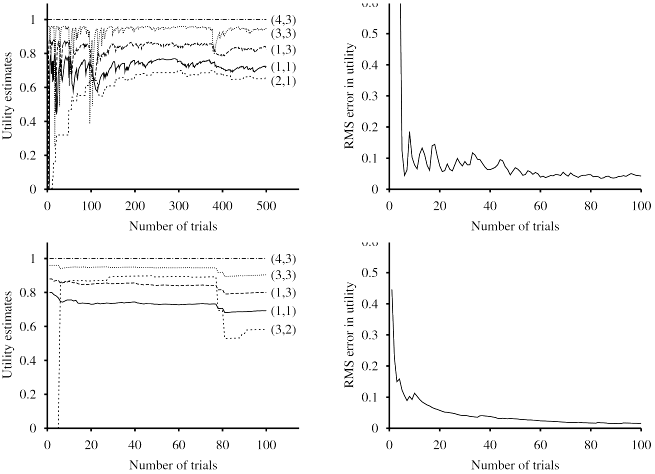
- Results:





- Error is for $U(1, 1)$.

Temporal difference learning



- A bit slower and noisier than ADP

UNIVERSITY OF LINCOLN

Temporal difference learning

- The final thing to note is that TD learning is *model free*.
- There is no transition model.
- That makes it easier to apply (no need to count transition probabilities).
- Learning reduces to applying the TD rule on transition from one state to another.

UNIVERSITY OF LINCOLN

Active reinforcement learning

- The passive reinforcement learning agent is told what to do.
- Fixed
- An active reinforcement learning agent must decide
  what to do.
  (While learning)
- We'll think about how to do this by adapting the passive ADP learner.

UNIVERSITY OF LINCOLN

Active reinforcement learning

- We can use exactly the same approach to estimating the transition function.
- Sample average of the transitions we observe.
- But computing utilities is more complex.

Active reinforcement learning

- When we had a policy, we could use the simple version of the Bellman equation:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- But we don't have a policy.
- When we have to choose actions, we need the utility values to base our choice of action on.
- How do we get utilities?

UNIVERSITY OF LINCOLN

Active reinforcement learning

- We use value iteration.
- At any stage, we can run:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

  to stability to compute a new set of utilities.
- So establishing utilities is not so hard.

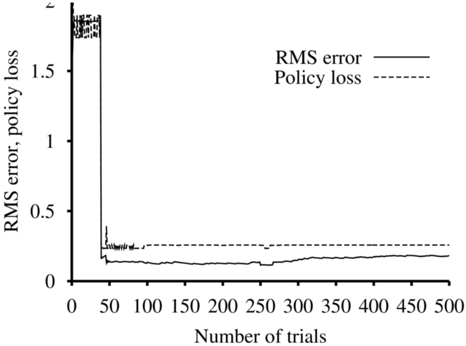
UNIVERSITY OF LINCOLN

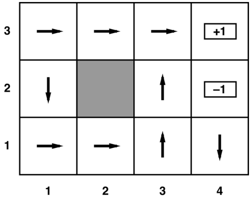
Active reinforcement learning

- Deciding what to do, what action to take, is the next issue.
- Normally after running value iteration we would choose the action with the highest expected utility.
- Greedy agent
- Could do that while we are learning.
- This turns out not to be so great an idea.
- Typically a greedy agent will not learn the optimal policy

Active reinforcement learning

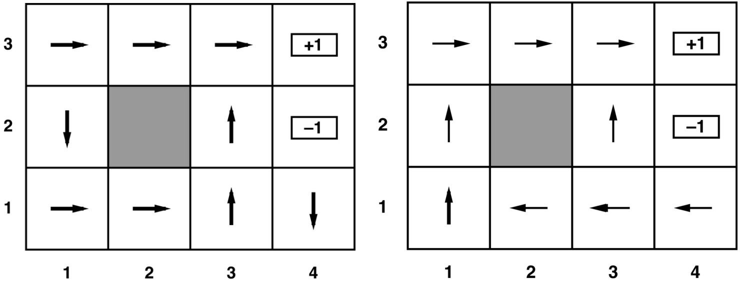
- On the usual example:





- Graph is error compared with optimal utility values.

Greedy vs optimal



- Greedy (left) and optimal (right)
- Greedy prefers the lower route, despite the danger of -1.

Exploration

- The issue is that once the agent finds a run that leads to a good reward, it tends to stick to it.
- It stops exploring.

Exploration

UNIVERSITY OF LINCOLN

Exploration

- A typical approach is to change the estimated utility assigned to states in value iteration.
- Manipulate the values to force the learner to explore.
- Then, once exploration is sufficient, we just let it do its thing.

Exploration

- To do this, we can use:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} f\left(\sum_{s'} P(s'|s,a) U_i(s'), N(s,a) \right)$$

where:

- $N(s,a)$ counts how many times we have done a in s,
- $f(u,n)$ provides an exploration-happy estimate of the utility of a state.

UNIVERSITY OF LINCOLN

Exploration

- For example:

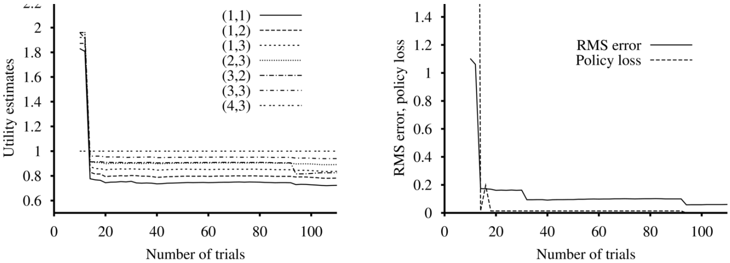
$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

  R^+ is an optimistic reward, and N_e is the number of times we want the agent to be forced to pick an action in every state.

- We force the learner to pick each state/action pair N_e times.
- N_e becomes another parameter that has to be adjusted until we find good solutions.

UNIVERSITY OF LINCOLN

Exploration



- Slow to converge on U, but quickly finds a policy that is close to optimal.

UNIVERSITY OF LINCOLN

Active reinforcement learning: solution

- A list of states $s_1, \ldots s_n$.
- Each state has a utility estimate associated with it $U(s)$.
- Each state has a set of actions associated with it, $a_1, \ldots a_m$.
- Each state/action pair has a probability distribution:

$$\mathbf{P}(S'|s, a_i)$$

over the states s' that it gets to from s by doing a_i.

UNIVERSITY OF LINCOLN

Model-free active learning

- The form of active reinforcement learning we have just looked at learns a transition model.
- What about a model free version?
- Can quite easily define an active version of temporal difference learning.

Q-learning

- Q-learning is a model-free approach to active reinforcement learning.
- It doesn't need to learn $P(s'|s, a)$.
- Revolves around the Q-value of a state/action pair, $Q(s, a)$
- $Q(s, a)$ denotes the value of doing a in s, so that:

$$U(s) = \max_a Q(s, a)$$

- Easier to learn than $U(s)$

Q-learning

- We can write:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q(s',a')$$

- Note that the sum is over s'
- Can compute estimates of $Q(s,a)$ by running value-iteration style updates on this.
- But it wouldn't be model-free.

UNIVERSITY OF LINCOLN

Q-learning

- However, we can write the update rule as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

  and recalculate everytime that a is executed in s and takes the agent to s'.
- Again, α is the learning rate.
- Note the similarity between this update, and the one for TD-learning (slide 68).

Action selection

- Since Q-learning is an active approach to reinforcement learning, we have to choose which a' to select in s'.
- Again greedy selection is usually a poor choice.
- Typical approach is to force exploration as we did before.

Q-learning

#AIMA3e **function** Q-Learning_Agent(*percept*) **returns** an action
inputs: *percept*, a percept indicating the current state s' and reward signal r'
persistent: Q, a table of action values indexed by state and action, initially zero
N_{sa}, a table of frequencies for state-action pairs, initially zero
s, a, r, the previous state, action, and reward, initially null

if Terminal?(s) then $Q[s, \text{None}] \leftarrow r'$
if s is not null **then**
increment $N_{sa}[s, a]$
$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
$s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
return a

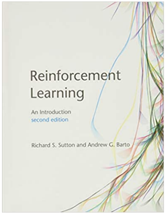
- Note that α is a function of the number of visits to s, a.
- Ensures convergence.

Q-learning: solution

- A list of state action pairs $\langle s_i, a_j \rangle$.
- Each state/action pair has $Q(s_i, a_j)$.
- For a given s_i, just pick the a_j to maximise $Q(s_i, a_j)$.

And more . . .

- Lots more we could look at.



Summary

- We started by looking at the difficulties of extending this work to partially observable worlds.
- Then we looked at how reinforcement learning can help solve MDPs for which we don't have a model.
- Looked at both *passive* and *active* methods.
- Looked at both *model-based* and *model-free* methods.

UNIVERSITY OF LINCOLN

Version History

- Version 1.0, 30th December 2021
- Version 1.1, 10th January 2021
  - Added in the answers to the polls.
  - Added the policy on slide 38.
- Note that I did not remove the slides (69–73) that I skipped over, because I thought that change of slide numebring would be confusing when looking back at the video.

UNIVERSITY OF LINCOLN