Lecture 14

Markov Decision Processes and Reinforcement Learning

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Uncertain knowledge and Reasoning
  - ✔ Probability and Bayesian approach
  - ✔ Bayesian Networks
  - ✔ Hidden Markov Chains
  - ✔ Kalman Filters
- Learning
  - ✔ Supervised
    Decision Trees, Neural Networks
    Learning Bayesian Networks
  - Unsupervised
    EM Algorithm
- Reinforcement Learning
- Games and Adversarial Search
  - Minimax search and Alpha-beta pruning
  - Multiagent search
- Knowledge representation and Reasoning
  - Propositional logic
  - First order logic
  - Inference
  - Plannning

Recap

Supervised	$(x_1, y_1)(x_2, y_2)\ldots$	$y = f(x)$
Unsupervised	$x_1, x_2, \ldots$	$\Pr(X = x)$
Reinforcement	(s, a, s, a, s) + rewards at some states	$\pi(s)$

Reinforcement Learning

Consider chess:

- we wish to learn correct move for each state but no feedback available on this
- only feedback available is a reward or reinforcement at the end of a sequence of moves or at some intermediary states.
- agents then learn a transition model

Other examples, backgammon, helicopter, etc.

Recall:
Environments are categorized along several dimensions:

fully observable	partially observable
deterministic	stochastic
episodic	sequential
static	dynamic
discrete	continuous
single-agent	multi-agents

Markov Decision Processes

Sequential decision problems: the outcome depends on a sequence of decisions. Include search and plannig as special cases.

- search (problem solving in a state space (detrministic and fully observable)
- planning (interleaves planning and execution gathering feedback from environment because of stochasticity, partial observability, multi-agents. Belief state space)
- learning
- uncertainty

Environment:

	Deterministic	Stochastic
Fully observable	A*, DFS, BFS	MDP

Reinforcement Learning

- MDP: fully observable environment and agent knows reward functions
- Now: fully observable environment but no knoweldge of how it works (reward functions) and probabilistic actions

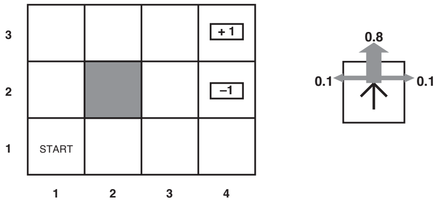
Outline

Terminology and Notation

Sequential decision probelm in a fully observable, stochastic environment with Markov transition model and additive rewards

$s \in S$	states
$a \in A(s)$	actions
s_0	start state
$p(s'\|s, a)$	transition probability; world is stochastic; Markovian assumption
$R(s)$ or $R(s, a, s')$	reward
$U([s_0, s_1, \ldots, s_n])$ or $V()$	utility function depends on sequence of states (sum of rewards)

Example:



(a) (b)

- A fixed action sequence is not good becasue of probabilistic actions
- Policy π: specification of what to do in any state
- Optimal policy π^*: policy with highest expected utility

Highest Expected Utility

$$U([s_0, s_1, \ldots, s_n]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots + \gamma^n R(s_n)$$

$$U^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] = R(s) + \gamma \sum_{s'} \Pr(s'|s, a \in \pi(s)) U^\pi(s')$$

looks onwards, dependency on future neighbors

Optimal policy:

$$U^{\pi^*}(s) = \max_\pi U^\pi(s)$$
$$\pi^*(s) = \text{argmax}_\pi\, U^\pi(s)$$

Choose actions by max expected utilities (Bellman equation):

$$U^{\pi^*}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s, a) U(s')$$
$$\pi^*(s) = \text{argmax}_{a \in A(s)} \left[R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s, a) U(s')\right]$$

Value Iteration

1. calculate the utility function of each state using the iterative procedure below
2. use state utilities to select an optimal action

For 1. use the following iterative algorithm:

```
function VALUE-ITERATION(mdp, ε) returns a utility function
  inputs: mdp, an MDP with states S, actions A(s), transition model P(s' | s, a),
            rewards R(s), discount γ
          ε, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
                   δ, the maximum change in the utility of any state in an iteration

  repeat
     U ← U'; δ ← 0
     for each state s in S do
        U'[s] ← R(s) + γ max_{a ∈ A(s)} Σ_{s'} P(s' | s, a) U[s']
        if |U'[s] − U[s]| > δ then δ ← |U'[s] − U[s]|
  until δ < ε(1 − γ)/γ
  return U
```

Q-Values

- For 2. once the **optimal** U^* values have been calculated:

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \left[R(s) + \gamma \sum_{s'} \Pr(s'|s,a) U^*(s') \right]$$

  Hence we would need to compute the sum for each a.

- Idea: save Q-values

$$Q^*(s,a) = R(s) + \gamma \sum_{s'} \Pr(s'|s,a) U^*(s')$$

  so actions are easier to select:

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} Q^*(s,a)$$

Example

```
python gridworld.py -a value -i 1 --discount 0.9 --noise 0.2 -r 0 -k 1 -t
```

```
VALUES AFTER 1 ITERATIONS
---------------------------------------------------
| |    0     |    1     |    2     |    3     |
---------------------------------------------------
| |    ^     |    ^     |          |  ------  |
| |          |          |          | |      | |
|2|   0.00   |   0.00   |  0.00 >  | | 1.00 | |
| |          |          |          | |      | |
| |          |          |          |  ------  |
---------------------------------------------------
| |    ^     |          |          | -------  |
| |          |  #####   |          | |       | |
|1|   0.00   |  #####   |  < 0.00  | | -1.00 | |
| |          |  #####   |          | |       | |
| |          |          |          | -------  |
---------------------------------------------------
| |    ^     |    ^     |    ^     |          |
| |          |          |          |          |
|0| S: 0.00  |   0.00   |   0.00   |   0.00   |
| |          |          |          |          |
| |          |          |          |    v     |
---------------------------------------------------
```

```
Q-VALUES AFTER 1 ITERATIONS
-----------------------------------------------------------------------
| |        0        |        1        |        2        |        3        |
-----------------------------------------------------------------------
| |     /0.00\      |     /0.00\      |      0.09       |                 |
| |                 |                 |                 |                 |
| |                 |                 |                 |    [ 1.00 ]     |
|2|<0.00       0.00>|<0.00       0.00>| 0.00       0.72>|                 |
| |                 |                 |                 |                 |
| |                 |                 |                 |                 |
| |     \0.00/      |     \0.00/      |      0.09       |                 |
-----------------------------------------------------------------------
| |     /0.00\      |                 |     -0.09       |                 |
| |                 |                 |                 |                 |
| |                 |      #####      |                 |    [ -1.00 ]    |
|1|<0.00       0.00>|      #####      |<0.00      -0.72 |                 |
| |                 |      #####      |                 |                 |
| |                 |                 |                 |                 |
| |     \0.00/      |                 |     -0.09       |                 |
-----------------------------------------------------------------------
| |     /0.00\      |     /0.00\      |     /0.00\      |     -0.72       |
| |                 |                 |                 |                 |
| |                 |                 |                 |                 |
|0|<0.00   S   0.00>|<0.00       0.00>| <0.00      0.00>| -0.09     -0.09 |
| |                 |                 |                 |                 |
| |                 |                 |                 |                 |
| |     \0.00/      |     \0.00/      |     \0.00/      |     \0.00/      |
-----------------------------------------------------------------------
```

Example

```
python gridworld.py -a value -i 2 --discount 0.9 --noise 0.2 -r 0 -k 1 -t
```

```
VALUES AFTER 2 ITERATIONS
---------------------------------------------------
| |    0     |    1     |    2     |    3     |
---------------------------------------------------
| |    ^     |          |          |  ------  |
| |          |          |          | |      | |
|2|   0.00   |  0.00 >  |  0.72 >  | | 1.00 | |
| |          |          |          | |      | |
| |          |          |          |  ------  |
---------------------------------------------------
| |    ^     |          |    ^     | -------  |
| |          |  #####   |          | |      | |
|1|   0.00   |  #####   |   0.00   | | -1.00 | |
| |          |  #####   |          | |      | |
| |          |          |          | -------  |
---------------------------------------------------
| |    ^     |    ^     |    ^     |          |
| |          |          |          |          |
|0| S: 0.00  |   0.00   |   0.00   |   0.00   |
| |          |          |          |          |
| |          |          |          |    v     |
---------------------------------------------------
```

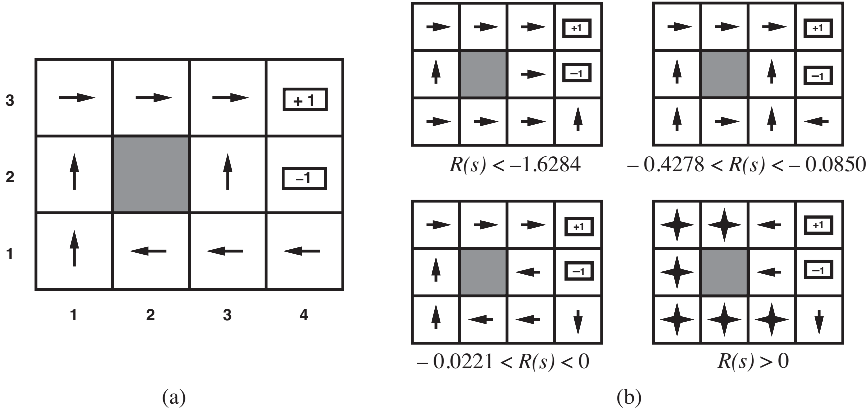
```
Q-VALUES AFTER 2 ITERATIONS
------------------------------------------------------------------------
| |       0        |       1        |        2        |        3         |
------------------------------------------------------------------------
| |    /0.00\      |      0.06      |      0.61       |                  |
| |                |                |                 |                  |
| |                |                |                 |    [ 1.00 ]      |
|2|<0.00     0.00>| 0.00     0.52>|  0.06     0.78> |                  |
| |                |                |                 |                  |
| |                |                |                 |                  |
| |    \0.00/      |      0.06      |      0.09       |                  |
------------------------------------------------------------------------
| |    /0.00\      |                |     /0.43\      |                  |
| |                |                |                 |                  |
| |                |     #####      |                 |   [ -1.00 ]      |
|1|<0.00     0.00>|     #####      | 0.06      -0.66 |                  |
| |                |     #####      |                 |                  |
| |                |                |                 |                  |
| |    \0.00/      |                |     -0.09       |                  |
------------------------------------------------------------------------
| |    /0.00\      |    /0.00\      |     /0.00\      |      -0.72       |
| |                |                |                 |                  |
| |                |                |                 |                  |
|0|<0.00  S  0.00>|<0.00     0.00>| <0.00     0.00> | -0.09      -0.09 |
| |                |                |                 |                  |
| |                |                |                 |                  |
| |    \0.00/      |    \0.00/      |     \0.00/      |     \0.00/       |
------------------------------------------------------------------------
```

Example

```
python gridworld.py -a value -i 3 --discount 0.9 --noise 0.2 -r 0 -k 1 -t
```

```
VALUES AFTER 3 ITERATIONS
--------------------------------------------------
| |    0     |    1     |    2     |    3      |
--------------------------------------------------
| |          |          |          |  ------   |
| |          |          |          | |      |  |
|2|  0.00 >  |  0.52 >  |  0.78 >  | | 1.00 |  |
| |          |          |          | |      |  |
| |          |          |          |  ------   |
--------------------------------------------------
| |    ^     |          |    ^     | -------   |
| |          |  #####   |          ||       |  |
|1|   0.00   |  #####   |   0.43   || -1.00 |  |
| |          |  #####   |          ||       |  |
| |          |          |          | -------   |
--------------------------------------------------
| |    ^     |    ^     |    ^     |           |
| |          |          |          |           |
|0| S: 0.00  |   0.00   |   0.00   |   0.00    |
| |          |          |          |           |
| |          |          |          |     v     |
--------------------------------------------------
```

```
Q-VALUES AFTER 3 ITERATIONS
------------------------------------------------------------------------
| |       0        |       1        |       2        |        3        |
------------------------------------------------------------------------
| |      0.05      |      0.44      |      0.70      |                 |
| |                |                |                |                 |
| |                |                |                |    [ 1.00 ]     |
|2| 0.00     0.37>| 0.09     0.66>| 0.48     0.83> |                 |
| |                |                |                |                 |
| |                |                |                |                 |
| |      0.05      |      0.44      |      0.45      |                 |
------------------------------------------------------------------------
| |     /0.00\     |                |     /0.51\     |                 |
| |                |                |                |                 |
| |                |     #####      |                |    [ -1.00 ]    |
|1|<0.00     0.00>|     #####      | 0.38     -0.65 |                 |
| |                |     #####      |                |                 |
| |                |                |                |                 |
| |     \0.00/     |                |     -0.05      |                 |
------------------------------------------------------------------------
| |     /0.00\     |     /0.00\     |     /0.31\     |      -0.72      |
| |                |                |                |                 |
| |                |                |                |                 |
|0|<0.00  S  0.00>|<0.00     0.00>| 0.04     0.04  | -0.09     -0.09 |
| |                |                |                |                 |
| |                |                |                |                 |
| |     \0.00/     |     \0.00/     |      0.00      |     \0.00/      |
------------------------------------------------------------------------
```



Balancing of risk and reward.

Outline

1. Markov Decision Processes

2. Reinforcement Learning

Terminology and Notation

$s \in S$	states
$a \in A(s)$	actions
s_0	start state
$p(s'\|s, a)$	transition probability; world is stochastic;
$R(s)$ or $R(s, a, s')$	reward

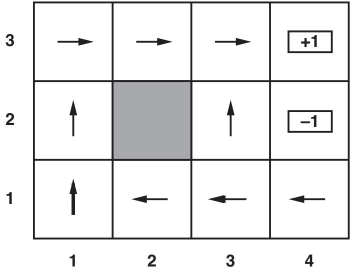
In reinforcement learning, we do not know p and R

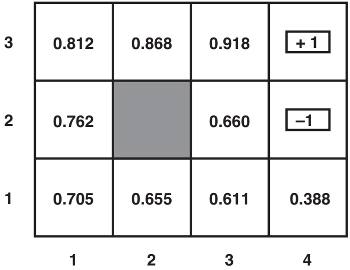
Agent	knows	learns	uses
utility-based agent	p	$R \leftarrow U$	U
Q-learning		$Q(s, a)$	Q
reflex agent		$\pi(s)$	π

Passive RL: policy fixed
Active RL: policy can be changed

Passive RL

Perform a set of trials and build up the utility function table





Passive RL

- Direct utility estimator (Monte Carlo method that waits until the end of the episode to determine the increment to $U_t(s)$
- Temporal difference learning (wait only until the next time step)

Passive RL
Temporal difference learning

If a nonterminal state s_t is visited at time t, then update the estimate for U_t based on what happens after that visit and the old estimate.

- Exponential Moving average:
  Running interpolation update:

  $$\bar{x}_n = (1-\alpha)\bar{x}_{n-1} + \alpha x_n$$
  $$\bar{x}_n = \frac{x_n + (1-\alpha)x_{n-1} + (1-\alpha)^2 x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

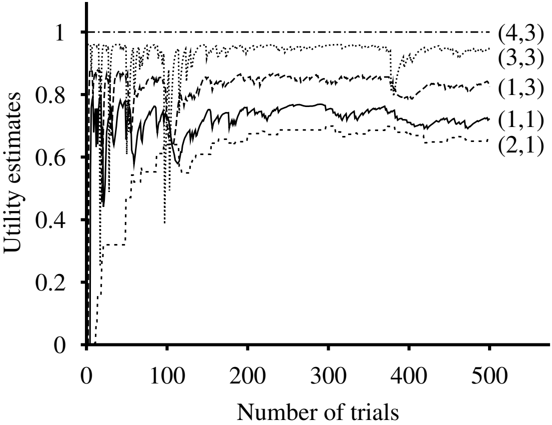
  Makes recent samples more important, forgets about past (old samples were wrong anyway)

- α learning rate: if a function that decreases then the average converges.
  E.g. $\alpha = 1/N_s$, $\alpha = N_s/N$, $\alpha = 1000/(1000+N)$,

$$\begin{aligned} NewEstimate &\leftarrow (1-\alpha) OldEstimate + \alpha AfterVist \\ U(s) &\leftarrow (1-\alpha)U(s) + \alpha[(r + \gamma U(s')] \\ U(s) &\leftarrow U(s) + \alpha(N_s)[r + \gamma U(s') - U(s)] \end{aligned}$$

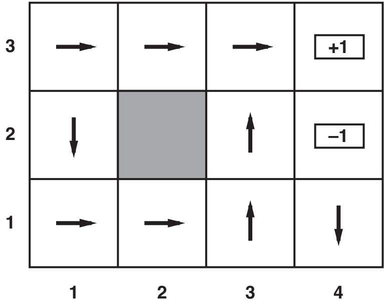
```
Initialize U(s) arbitrarily, π to the policy to be evaluated;
repeat /* for each episode                                    */
|   Initialize s;
|   repeat /* for step of an episode                          */
|   |   a ← action given by π for s
|   |   Take action a, observe reward r and next state s'
|   |   U(s) ← U(s) + α(N_s)[r + γU(s') − U(s)]
|   |   s ← s'
|   until s is terminal;
until convergence;
```

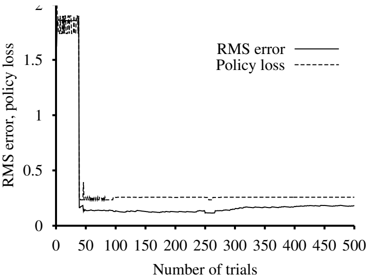
Learning curves



Active Learning

- Greedy agent, recompute a new π in TD algorithm
- Problem:
  actions not only provide rewards according to the learned model but also influence the learning by affecting the percepts that are received.





Adjust the TD with a Greedy in the Limit of Infinite Exploration

Exploration/Exploitation

- Simplest: random actions (ϵ-greedy)
  - every time step, draw a number in $[0, 1]$
  - if smaller than ϵ act randomly
  - if largerthan ϵ act according to greedy policy
  - ϵ can be lowered with time
- Another solution: exploration function

Active RL
Q-learning

- We can choose the action we like with the goal of learning optimal policy

$$\pi^*(s) = \mathrm{argmax}_a \left[R(s) + \gamma \sum_{s'} \Pr(s'|s,a) U^*(s') \right]$$

- same as in value iterations algorithm but not off-line
- Q-values are more useful to be learned:

$$Q^* = R(s) + \gamma \sum_{s'} \Pr(s'|s,a) U^*(s')$$
$$\pi(s) = \mathrm{argmax}_a\, Q^*(s,a)$$

- Sarsa algorithm learns Q values same way as TD algorithm

In value iteration algorithm

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

Same with Q

$$Q_{i+1}(s,a) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s,a) Q_i(s',a')$$

Sample based Q^* learning:

- observe sample s, a, s', r
- consider the old estimate $Q(s,a)$
- derive the new sample estimate

$$Q^*(s,a) \leftarrow R(s) + \gamma \max_{a'} \sum_{s'} \Pr(s'|s,a) Q^*(s',a')$$

$$sample = R(s) + \gamma \max_{a'} Q^*(s',a')$$

- Incorporate the new estimate in running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha sample$$

Initialize $Q(s, a)$ arbitrarily;
repeat /* for each episode */
  Initialize s;
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  repeat /* for step of an episode */
    Take action a, observe reward r and next state s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal;
until convergence;

Note: update is not

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

since by ϵ-greedy we allow not to choose the best

Example


```
python gridworld.py -a q -k 10 --discount 0.9 --noise 0.2 -r 0 -e 0.1 -t | less
```

(note: now episodes are used in training, and there are no iterations, rather steps that end at terminal state)

Properties

Converges

- if explore enough and
- α is small enough
- but α does not decrease too quickly

$\rightsquigarrow$ Learns optimal policy without following it