

01DRO1

Decision Making under Uncertainty

2017/2018

Reinforcement Learning I.

Lecture 6

27.3.2018

Readings:

Books:

- L.P. Kaelbling, M. L. Littman, and A.W. Moore. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4 (1996) 237–285
- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge MA USA, 1998
- Csaba Szepesvári. Algorithms for Reinforcement Learning. Morgan & Claypool, 2010.

Reminder:

Mid-term assignment – Apr 3 (Lectures 1-6) .

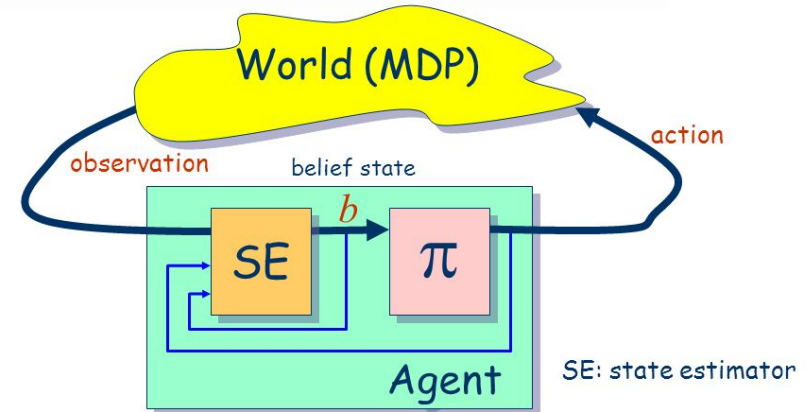
Where are we?

Last time.. POMDP

POMDP is a very general model for sequential DM allowing

- uncertainty in action effects
- uncertainty in knowledge of system state, noisy observations
- multiple possibly conflicting objectives

POMDP Framework



Last time.. Decomposition of POMDP

- Belief state updated by Bayesian conditioning is a sufficient statistic providing all relevant information about the history.
- We can define **belief MDP** with a state set consisting of all possible belief states (i.e. map unsolvable POMDP into MDP) as follows:
 - B – a set of *belief* states - ***comprises original state space S***
 - A – a set of actions – ***no change***
 - O – a set of observations – ***newly introduced***
 - $\mathcal{T}: B \times B \times A \rightarrow [0,1]$ – state transition model $Pr(b_{t+1} | b_t, a_t)$ - ***newly defined*** as $Pr(b_{t+1} | b_t, a_t) = \sum_{o \in O} Pr(b_{t+1} | o_{t+1}, b_t, a_t) Pr(o_{t+1} | b_t, a_t)$
 - $\rho: B \times A \rightarrow \mathbb{R}$ - bounded reward function on belief states $\rho(b_t, a_t)$ - ***newly defined*** as $\rho(b_{t+1}, a_{t+1}) = \sum_{s \in S} Pr(b_{t+1} | o_{t+1}, b_t, a_t) r(s_{t+1}, a_t)$, where $r(s_{t+1}, a_t)$ is original reward

Last time.. Bayes belief state update

- Let $b_{t+1} = b_{t+1}(s_{t+1}) = \Pr(s_{t+1} | o_{t+1}, a_t, b_t)$, be **belief state** (degree of belief in s_{t+1})
s.t. $0 \leq b(s) \leq 1, \forall s \in S, \sum_{s \in S} b(s) = 1$.

- Upon taking new action a_t and observing o_{t+1} , $b(s_t)$ should be updated, i.e.
 $(b_t, a_t) | o_{t+1} \rightarrow b_{t+1}$.

$$b_{t+1}(s_{t+1}) = \Pr(s_{t+1} | o_{t+1}, a_t, b_t) = \frac{\Pr(o_{t+1} | s_{t+1}, a_t, b_t) \Pr(s_{t+1} | a_t, b_t)}{\Pr(o_{t+1} | a_t, b_t)} \quad \text{Bayes' theorem}$$

$$= \frac{\Pr(o_{t+1} | s_{t+1}, a_t) \sum_{s \in S} \Pr(s_{t+1} | a_t, b_t, s_t) \Pr(s_t | a_t, b_t)}{\Pr(o_{t+1} | a_t, b_t)} \quad \text{chain rule}$$

$$= \frac{\Pr(o_{t+1} | s_{t+1}, a_t) \sum_{s \in S} \Pr(s_{t+1} | a_t, b_t, s_t) b_t(s_t)}{\Pr(o_{t+1} | a_t, b_t)} \quad \text{belief definition}$$

$$\propto \Pr(o_{t+1} | s_{t+1}, a_t) \sum_{s \in S} \Pr(s_{t+1} | a_t, s_t) b_t(s_t)$$

Today..

Outline

- What is reinforcement learning?
- Temporal-Difference learning
- Q-learning
- Exploration-Exploitation dilemma
- Bayesian RL

Problem of Learning

while interacting with environment to achieve some goal.

- Baby playing. Play has some goal.

No teacher. Sensory-motor connection to environment.

Cause implies effect

Action implies consequences

Baby learns *how* to achieve her goals

- learning to drive, learning to play piano,..

Environment's response *affects* our subsequent actions

We observe the *effects* of our actions then.



" Emergency stop - I'll tick that. "

What is learning?

Learning is any relatively permanent change in behaviour that occurs as a result of practice and experience.

(psychology textbook)

Learning is the process by which a relatively stable modification in stimulus–response relations is developed as a consequence of functional environmental interaction via the senses.

(S.Lachman, psychologist)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

(T.Mitchel, computer scientist)

- The same set of ideas and mathematical tools have emerged in many of different fields, sometimes with different emphasis (different goals, tasks, etc.).

Presence of Learning

Psychology: perception, movement control, reinforcement learning, mathematical psychology...

Cognitive Science: philosophy of mind, computational linguistics, ...

Economics: decision theory, game theory, operational research, ...

Statistics: learning theory, data mining, learning and inference from data, ...

Control Engineering: signal processing, system identification, adaptive and optimal control, information theory, robotics, ...

Computer Science: AI, theory of computation, computer vision, information retrieval, ...

Computational Neuroscience: neuronal networks, neural information processing...

Learning: do we need it in DM?

Learning can help to find:

- solution that cannot be found in advance. Reasons:
 - environment is too complex
 - environment is not fully known
- decision that is gradually improving
- solution that adapts to time-varying environment

Types of (Machine) Learning

Machine learning is an interdisciplinary field focusing on both the mathematical foundations and practical applications of systems that learn, reason and act.

- 'Supervised' Learning

'Teacher' tells the agent what to remember (learn from labeled examples)

- Unsupervised Learning

The agent discovers unknown phenomena on its own (cluster unlabeled examples)

- Reinforcement Learning

Environment provides hints to the agent (learn from interaction)

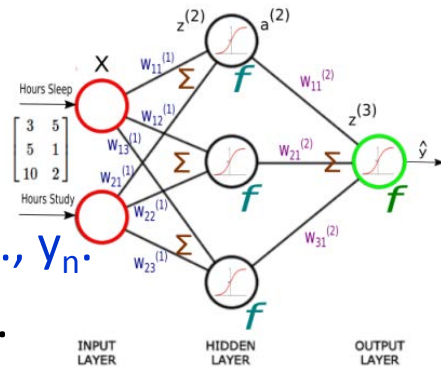
Learning in 'machine' terms

Let an agent observe a sequence of inputs: x_1, x_2, \dots, x_n

- **Supervised learning:** The agent is given desired outputs y_1, y_2, \dots, y_n .

Goal: to *learn how to design* the correct output y_j given input x_j .

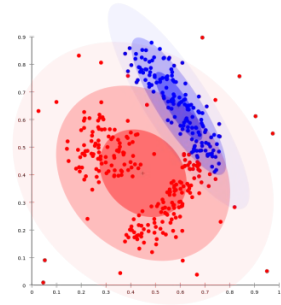
Typical example: neural network



- **Unsupervised learning:**

Agent's goal is to *build a model* of x that can be used for other tasks (reasoning, decision making, predicting, etc.)

Typical example: find patterns in data; clustering



- **Reinforcement learning:** The agent can take actions a_1, a_2, \dots which influence the environment state, and receives rewards (or punishments) r_1, r_2, \dots

Agent's goal is to learn how to act to maximise long-term reward.

Supervised learning

- The agent is given desired outputs y_1, y_2, \dots, y_n .

Goal: to *learn how to produce* the correct output y_j given a new input x_j . discover a function that approximates true function $y_n = f(x_n)$

the agent observes some example input–output pairs and learns a function that maps from input to output.

Supervised learning = choosing the hypothesis h that is most probable given the data

$$h = \operatorname{argmax}_{h \in H} P(h | \text{data}) .$$

By Bayes' rule this is equivalent to

$$h = \operatorname{argmax}_{h \in H} P(\text{data} | h) P(h) .$$

- Occam's razor

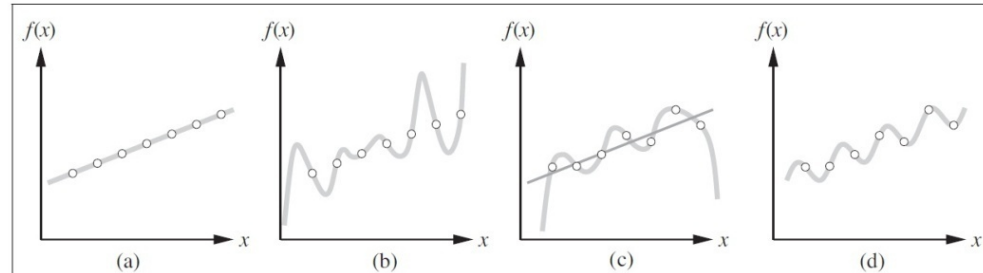


Figure 18.1 (a) Example $(x, f(x))$ pairs and a consistent, linear hypothesis. (b) A consistent, degree-7 polynomial hypothesis for the same data set. (c) A different data set, which admits an exact degree-6 polynomial fit or an approximate linear fit. (d) A simple, exact sinusoidal fit to the same data set.

Problem with Supervised Learning

- Supervised learning is sometimes unrealistic: where will correct answers come from?
 - use of redundant information as a way of overcoming lack of extensive training data
- In many cases, the agent will receive a *single* reward, after a long sequence of actions/decisions.
- Environment changes => the agent must adjust its action choices.

On-line issue



ML: summary.

- **Supervised learning:**

mostly classification tasks; predicting y based on x ; regression, etc

- **Unsupervised learning:**

mostly finding useful (suitable) representation of data.

dimensionality reduction; outlier detecting; data compression, clustering; etc

Note: In practice, these distinction are not always so crisp. In semi-supervised learning we are given a few labeled examples and must make what we can of a large collection of unlabeled examples

.. not the topics of this lecture

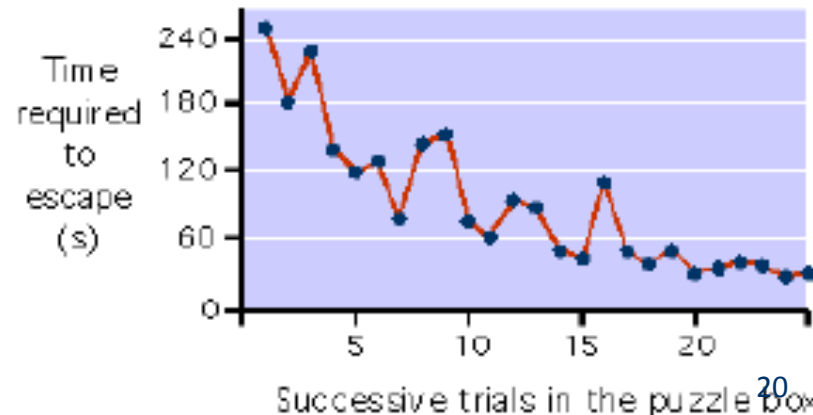
Reinforcement learning: origins

- Psychology: **operant (instrumental) conditioning** is a learning process through which the strength of a behavior is modified by reward or punishment
- **Principle of reinforcement** intr. by B.F. Skinner who considered human actions dependent on consequences of previous actions.
 - Bad consequences => higher chance the action will not be repeated;
 - Good consequences => higher chance of the action being repeated.
- Important: experience must occur in the past.

Selectional: try alternatives and use good ones.

Associative: associate alternatives with particular situations.

Trial and Error Learning Theory, see E.L.Thorndike,
Animal Intelligence: Experimental studies, Macmillan, 1911.



RL: Targeted DM tasks

- Stochastic DM problems where models (transition probability, reward, even *state* space) are *unknown* but knowledge *can be collected* while interacting with the environment.

Examples: adaptive control; robotics;

- DM problems where simulation models are available (e.g., values s_{t+1}, r_{t+1} given $s_1, s_2, \dots, s_t, r_1, r_2, \dots, r_t$), but explicit transition model or reward is not given
- DM problems with large state spaces where explicit dynamic programming is *not* tractable or *not* feasible

Examples of applications

Robotics (motion control):

- **Observations:** camera images; sound; joint angles, ...
- **Actions:** joint torques, motor current
- **Rewards:** stay balanced, navigate to target location, serve (and possibly protect) human being

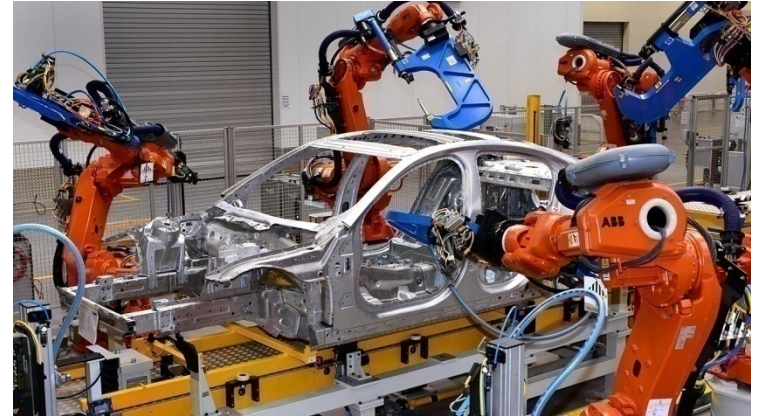


photo from Robotics Business Review

Examples of applications

Stock (inventory) management:

- Observations: current stock levels, seasonal demands,...
- Actions: number of units of a product to purchase/move/sell
- Rewards: profit, profit, profit



Other recent examples

- Superhuman Go using supervised learning + policy gradients + Monte Carlo search + value functions
Silver et al. Mastering the game of Go with deep neural networks and tree search, Nature 529.7587, pp. 484-489, 2016.
- Robotic manipulation using guided policy search (supervised learning, with supervision provided by a simple trajectory-centric RL)
S. Levine et al. End-to-end training of deep visuomotor policies, JMRL 17, pp.1-40, 2016.
- General AI: playing ATARI using deep Q-learning, policy gradients (Google patented “Deep RL”)
Mnih et al, Human-level control through deep reinforcement learning, Nature 518 pp.529-533, 2015.

Reinforcement learning: a number of different views

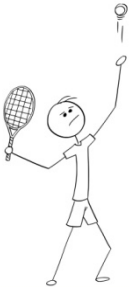
- Neuroscience
- Statistics: bandit problems
- Operational Research (Dynamic programming)
- Control theory
- AI and Computer science

From DM perspective: RL is learning how to behave/act based on interactions with environment.

By other words, solving a (PO)MDP without a model specified in advance

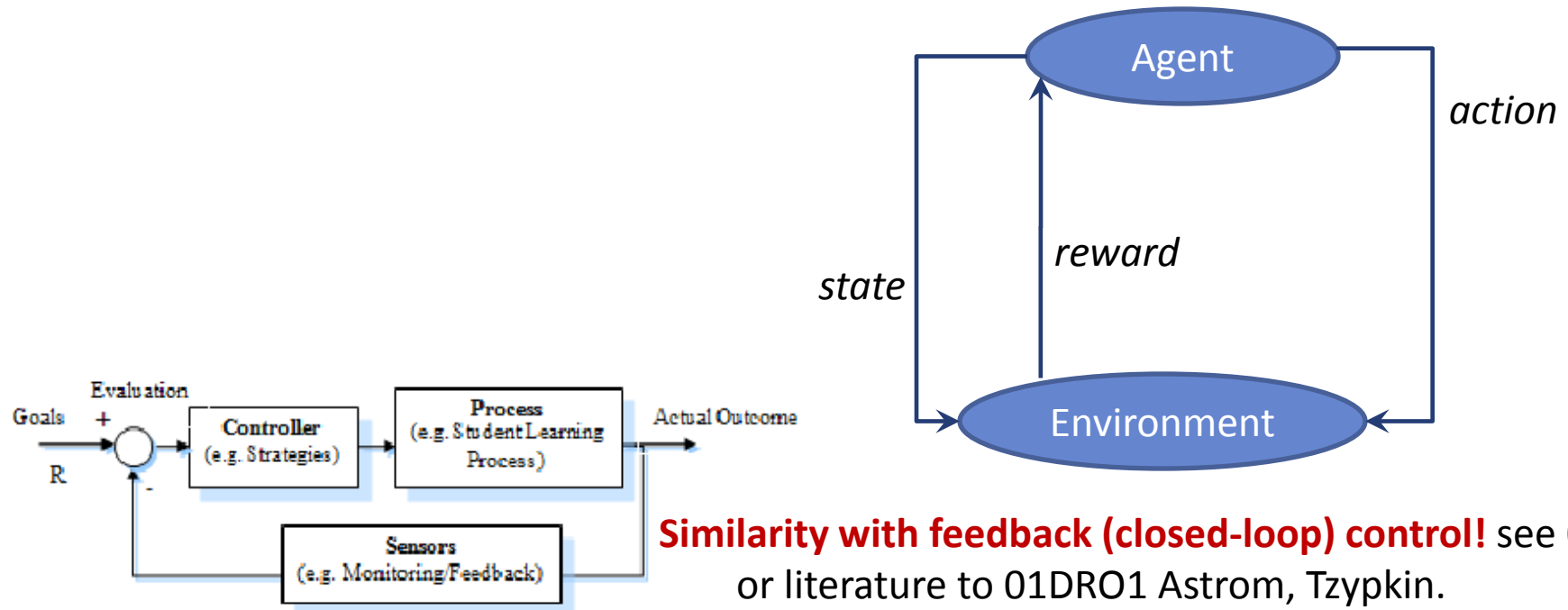
RL: what and how

- Using feedback/rewards to learn a successful agent function
- Rewards may be provided following *each* action, or when the agent reaches a terminal state.
- Rewards can be components of the original utility function or they can be ‘hints’ providing feedback to the agent , for instance “bad tone” (singing), “good serve” (tennis), etc.



RL: basic idea

- Agent's utility is given by reward function
- Agent receives feedback in the form of rewards
- Agent must learn to act to maximise expected rewards



From MDP to RL

- **Given:** MDP as we know it
 - a set of states $s \in S$
 - a set of actions (per state) A
actions may be stochastic
 - a model $T(s',a,s)$
 - a reward function $R(s',a,s)$
- **Aim:** to find an optimal policy $\pi(s)$, i.e. way of selecting actions that gets you the most reward

From MDP to RL

- **Given:** MDP as we know it
 - a set of states $s \in S$
 - a set of actions (per state) A
actions may be stochastic
 - ~~a model $T(s',a,s)$~~
 - ~~a reward function $R(s',a,s)$~~
- **Aim:** to find an optimal policy $\pi(s)$, i.e. way of selecting actions that gets you the most reward

New aspect: $T(s',a,s)$ or $R(s',a,s)$ are **unknown**, i.e we do not know which states are good or what the actions do.

Solution: try actions and learn resulting states

Reinforcement Learning

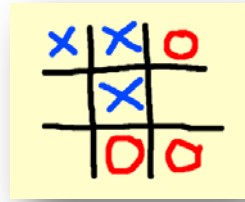
MDP with unknown transition and reward models

- A set of **states** $s \in S$
- A set of **actions** (per state) A
- A set of **reinforcement signals** (rewards)

Aim of RL: Learn an optimal policy $\pi(s)$ while interacting with the environment




cf. **Aim of (pure) MDP:** to find an optimal policy $\pi(s)$ given transition and reward model, that gets you the most reward

Example: Tic-Tac-Toe



Let's play against an imperfect opponent



- How might we construct an agent that will find the imperfections in  play and learn to maximise chances of winning?
- This (simple) problem cannot be solved by 'classical' techniques:
 - classical "minimax" solution (game theory) is not correct here as it assumes a particular way of playing by .
 - DP can compute an optimal solution for any co-player, but requires a complete specification of co-player, including the probabilities with which  makes each move in each board state.

Tic-Tac-Toe: RL

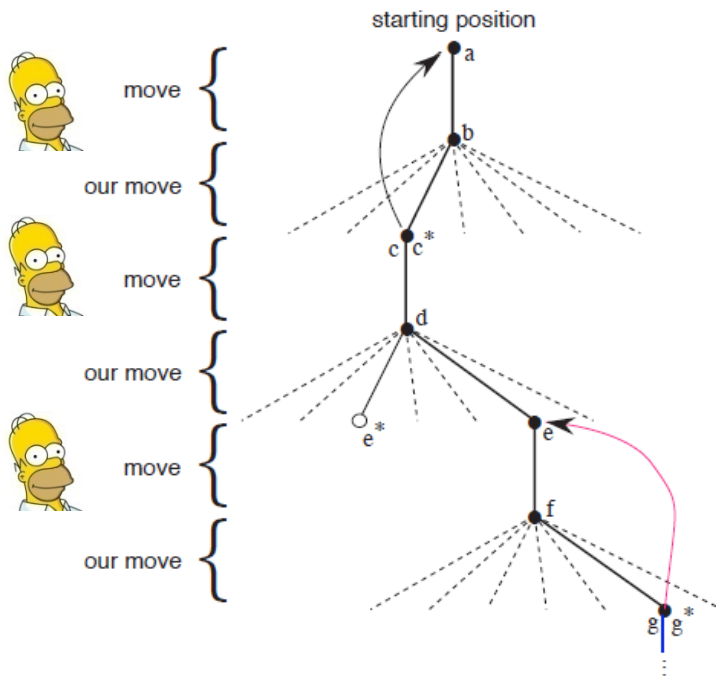


Figure 1.1: A sequence of tic-tac-toe moves. The solid lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make. Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e^* , was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing *backups* as suggested by the curved arrows and detailed in the text.

RL Main characteristics

- **Reinforcement signals** (reinforcements): rewards
- **Temporal 'credit' assignment**: when a reward is received, which action should be credited?
- **Exploration/exploitation tradeoff**: as agent learns, should it exploit its current knowledge to maximise its rewards or explore to refine its knowledge?
- **Long-time learning**: reinforcement learning only

RL: basic picture

RL is learning *what to do* so as to maximise a numerical reward signal. Agent is *not* told what actions to take, but must discover them itself by trying them out and seeing what the reward is.

Agent acting in (finite) MDP (S, A, T, R) , A and S - known, and at each t :

- observes environment state s_t that depends on the agent's actions a_1, \dots, a_{t-1}
- takes action $a_t^* = \pi(s_t)$
- receives reinforcement signal $r_t = R(r_t | s_t, a_t^*)$ where $R(\cdot)$ is a reward model *unknown* to the agent.

Note difference between RL and supervised:

- agent has no access to reward – must learn it from interaction
- state depends on agent's previous actions in unknown way

RL: basic picture

RL is learning *what to do* so as to maximise a numerical reward signal. Agent is *not* told what actions to take, but must discover them itself by trying them out and seeing what the reward is.

Agent acting in (finite) MDP (S, A, T, R) , A and S - known, and at each t :

- observes environment state s_t that depends on the agent's actions a_1, \dots, a_{t-1}
- takes action $a_t^* = \pi(s_t)$
- receives reinforcement signal $r_t = R(r_t | s_t, a_t^*)$ where $R(\cdot)$ is a reward model *unknown* to the agent.

Key problem: what is the best action to take at time $t+1$ given data collected:
 $(s_1, \dots, s_t, a_1, \dots, a_t, r_1, \dots, r_t)$?

Types of RL

- **Passive vs Active Learning**

Passive learning: the agent executes a fixed policy and tries to evaluate it

Analogous to policy *evaluation* in PI

Active learning: the agent updates its policy as it learns and attempts to find an optimal (or at least good) policy

Analogous to *solving* the underlying MDP

- **Model-based vs Model-free Learning**

Model-based: learn transition T and reward R model (or approximated models) and use them to determine optimal policy

Model free: derive optimal policy *without* explicit learning the model

Note: model-free RL = indirect adaptive control; model-based RL = direct adaptive control, see Astrom, 01DR012 webpage

What is RL?

- Model-based Learning
 - Adaptive DP - basically learns T and R , then perform policy evaluation based on the underlying MDP (T and R)
- Model-free Learning
 - Direct Evaluation - performs policy evaluation.
 - Temporal Difference (T-D) Learning - performs policy evaluation
 - Q-Learning - learns optimal state-action value function Q^*

Model-Based RL

- Learn the model empirically via growing experience
 - Collect outcomes (s, a) for sufficient time (so-called learning phase) generate samples (i.e. run policy) if simulation used
 - Create an approximate model of MDP (S, A, T', R')
 - Fit transition model $T'(s,a,t)$ and estimate reward $R'(s,a)$ for instance by using empirical observed distributions (or update some prior beliefs)
- Solve the MDP with the learned approximate model (S, A, T', R') as if the learned model were correct (using standard value or policy iteration, for instance)

Example: Adaptive/Approximate DP (learn transitions and rewards from observations then update the values of the states), see literature on 01DR01 web, Sutton and Barto.

Model-based RL: remarks

1. *Performance* during learning phase could be bad
2. What *length of learning phase* should be to get high quality (S, A, T', R') ?
3. How to ensure collecting *sufficient* amount of data to approximate T' & R' ?
4. We have no T', R' we need to collect them and cover (S,A) -space => *exploration* problem.
5. Model minimises error of fit => convergence, however no guarantee that better model \equiv better policy
6. State value predictions can be updated using DP with approximate model. It can bring more value from experience cf. waiting for state visits either in real-world or simulation.
7. Uncertainty in model can be exploited to plan DP steps effectively

Model-based RL: summary

The negative items are critical in on-line case. If we can use simulation model for learning => they are less problematic (except of item 3).

Generally model-based methods can be valuable approach though model-free methods are more common.

Details on these methods see for instance Sutton and Barto.

Model-free RL

No need to learn the transition model and reward function

Common approaches:

- **Direct evaluation**

- Repeatedly execute the policy
- Estimate the value of the state s as the average over all times the state s was visited of the sum of discounted rewards accumulated from state s onwards
- slowly converges as Doesn't exploit the fact that utilities of states are not independent

- **Temporal Difference Learning**

Learning from every experience, update V any transition

- **Q-Learning**

Temporal difference TD(0) algorithm

Model-free method that performs policy evaluation. At each time:

- Collect experience s', s, a, r
- Update
$$V^\pi(s) = V^\pi(s) + \alpha \underbrace{(r(s) + \gamma V^\pi(s') - V^\pi(s))}_{\text{Temporal difference}}$$
- α - is learning rate, must satisfy $\sum_t \alpha_t \rightarrow \infty, \sum_t (\alpha_t)^2 < \infty$
- $r(s)$ – current reward
- $V(s')$ – the estimated value of the next state

The update is stochastic variant of DP.

If α is appropriately decreased with number of times $n(s)$ a state is visited (e.g. $\alpha(s) = 1/n(s)$), then $V^\pi(s)$ converges to correct value.

$$V^\pi(s) = r(s) + \gamma \sum_{s'} T(s', a, s) V^\pi(s')$$

TD(): how it works

Reinforcement :

more reward than expected $r(s_t) > \gamma V_{old}(s_{t+1}) - V_{old}(s_t) \Rightarrow \text{increase } V(s_t)$

less reward than expected $r(s_t) < \gamma V_{old}(s_{t+1}) - V_{old}(s_t) \Rightarrow \text{decrease } V(s_t)$

Copyright 2001 by Randy Glasbergen. www.glasbergen.com



"It's a special hearing aid. It filters out criticism and amplifies compliments."

TD(λ)

- TD(0): looks one step ahead while adjusting value estimates. Slow convergence.
- general TD(λ): update is applied to every state *depending* on the degree to which it has been visited in the past. if $\lambda=1 \Rightarrow$ update all the states depending on the number of times they were visited. Expensive computationally, fast convergence (*Adaptive Heuristic Critic* \equiv PI where VF doesn't computed as a set of lin.eq but via TD(0), details see Kaelbling et al.: RL: A Survey)
- TD(0) is a core of majority RL approaches as it
 - allows full online implementation: remember where you've been recently ("eligibility trace")
 - converges to true value of stationary policy
 - easy for implementation.

From TD approach to DM

Q-learning

- $Q^\pi(s, a) = E_\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots | s, a]$ is called **Q-function** or (state-action)-value function. Q-function is a expected total reward from taking action **a** at state **s**.

expected reinforcement
of **a** in **s** and subsequent
optimal choosing actions

$$Q^{opt}(s, a) = r(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a'),$$
$$V^{opt}(s) = \max_{a'} Q(s', a'), \quad \pi^{opt} = \arg \max_{a'} Q^{opt}(s', a')$$

- **Q-learning rule:**

$$Q_{new}(s, a) = (1 - \alpha) Q_{old}(s, a) + \alpha [r(s) + \gamma \max_{a'} Q_{old}(s', a')]$$

Q-learning algorithm

For each state s and action a initialize $Q(s,a)$ (can be zero). Then

1. Observe current state s and take action a
2. Receive immediate reward r
3. Observe new state s'
4. Update $Q(s,a)$
5. set $s=s'$

Q-learning

- Q-Learning (Watkins, 1988) is the first provably convergent direct adaptive optimal control algorithm, cf. non-direct adaptive control (Tzypkin)
- Impact on RL:
 - very practical for use in AI
 - simple update rule: sample-based Bellman backup
 - Q-learning guaranteed to converge to optimal Q-values if all actions tried infinitely often
 - Off-policy method: learns value (V, Q) of policy different from that being executed
 - Can be extended to multi-step returns ($Q(\lambda)$).

Remark: direct adap.control (AC)- the estimated parameters are *directly* used in the adaptive controller. Indirect AC- the estimated parameters are used to calculate controller parameters.

Comments

- Bellman's principle of optimality is the core of the methods
- it refers to the recursive thinking of what makes a path optimal (the *recursive* property of the optimal value function)
- TD(), Q-learning are methods to converge to a function obeying the Bellman optimality equation:

$$V^{opt}(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t) [r(s_{t+1}, a_t, s_t) + \gamma V^{opt}(s_{t+1})], \forall s_t \in S$$