01DRO1
Decision Making under Uncertainty

2017/2018

Reinforcement Learning II.

Lecture 9

17.4.2018

Readings:

- 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:

Project report (or presentation) due today: send by email. Include source files (for implementations).

Presentations: choose time slot (20min) on DRO1 web and send me email

DRO1: 1.5, 8.5 no class (holidays).

DROS: 2 cancelled lectures (28.2 and 10.4) will be 22.5 and 23.5 (usual schedule)

Mid-term test marks and correct solutions, see DRO1 web

Where are we?

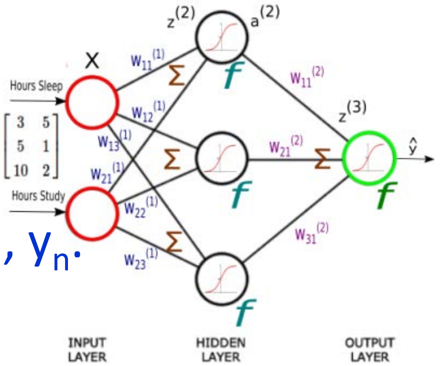
Recap: do we need learning in DM?

Learning is inevitable for any intelligent agent.

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

Recap: Learning in 'machine' terms



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

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

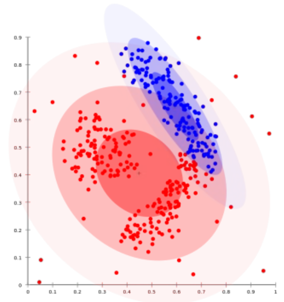
  Goal: to *learn how to design* the correct output y_j given input x_j.

  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 $a_1, a_2, \ldots$ which influence the environment state, and receives rewards (or punishments) $r_1, r_2, \ldots$

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

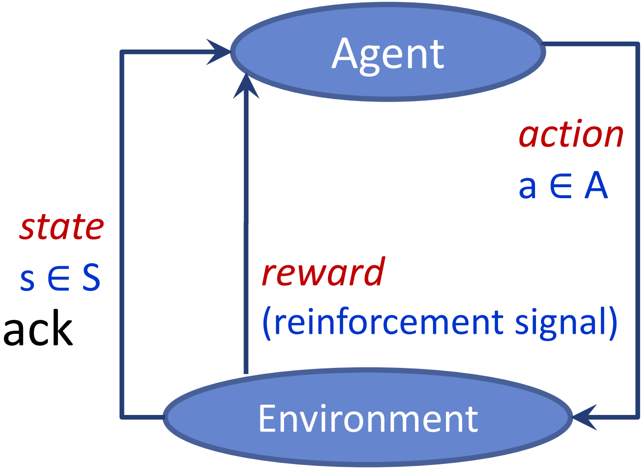
Recap: RL

Reinforcements used to train animals:

- Negative reinforcements (pain and hunger)
- Positive reinforcements (pleasure and food)

Reinforcement Learning $\neq$ Supervised Learning

Recap: basics of RL



- *Interact* with a system through states and actions
- *Receive rewards* (reinforcement) as performance feedback

Formal RL definition: MDP with **unknown** transition and/or reward models:

$\Rightarrow$ agent cannot simulate interaction with environment in advance, to predict future outcomes.

$\Rightarrow$ the optimal policy is learned through **sequential interaction** and evaluative feedback.

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

Recap: 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

Recap:

- 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*

Temporal difference TD(0) algorithm

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

- Collect experience s’, s, a, r
- Update $$V^{\pi}(s)=V^{\pi}(s)+\alpha\underbrace{\left(r(s)+\gamma V^{\pi}(s')-V^{\pi}(s)\right)}_{\textit{Temporal difference}}$$
- α -is learning rate, must satisfy $\Sigma_t \alpha_t \to \infty$, $\Sigma_t (\alpha_t)^2 < \infty$

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.

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

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

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

Q-learning (alternative TD method)

- $Q^{\pi}(s,a) = E_{\pi}\left[r_0 + \gamma\, r_1 + \gamma^2 r_2 + \ldots \middle| s,a\right]$ 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:

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

Today..

Reinforcement learning has many faces:

value iteration, policy iteration, linear programming, Q-learning, TD(), value function approximation, SARSA (State–action–reward–state–action) algorithm, Least Squares TD, Least Squares PI, policy gradient, inverse reinforcement learning, reward shaping, hierarchical reinforcement learning, inference-based methods, exploration vs. exploitation

On-policy and off-policy

Major MC assumptions (infinite sampling and exploring all possible states) are *not* realistic. Need to continually explore

– On-policy method: update Q-values based on s’ and *current policy* action, i.e. assume the current (estimation) policy will be followed

– Off-policy method: update Q-values based on s’ and *greedy policy* action (irrespectively of the real current policy)

- Give no difference if greedy policy is used
- The policy used to generate behaviour (*behaviour* policy), may be unrelated to the policy that is evaluated and improved (*estimation* policy).
- An advantage is the estimation policy may be deterministic, while the behaviour policy can continue to sample all possible actions. Agent can use a behaviour policy that is good at exploring, then infer optimal policy from that.

Types of RL algorithms

model availability	**Model-based (indirect):** $T(s'\|a,s)$ and $R(s',a,s)$ are known (ADP)
	Model-free (direct): $T(s'\|a,s)$ and $R(s',a,s)$ are unknown; only data (s',a,s) are at disposal (direct evaluation, TD(), Q-learning)
	Model-learning RL: estimate $T(s'\|a,s)$ and $R(s',a,s)$ from transition data
interaction level	**Offline:** data collected in advance (Q-iteration, PI)
	Online: Policy π^{opt} learnt by interacting with the environment (Q-learning, SARSA)
optimal policy search	**Off-policy:** find Q^{opt}, use it to compute π^{opt} (Q-learning,Q-iteration)
	On-policy: find Q^{π}, improve π and repeat (SARSA)

RL and MDP

	MDP with known $T(s'|a,s)$ **and** $R(s',a,s)$	**Unknown MDP Model- Based**	**Unknown MDP Model-Free**
Compute V^{opt}, Q^{opt}, π^{opt}	use Value Iteration (VI) or Policy Iteration (PI)	VI/PI on approximated MDP (Adaptive DP)	Q-learning
Evaluate a fixed policy π^{opt}	Policy Evaluation (PE)	PE on approximated MDP	Value Learning

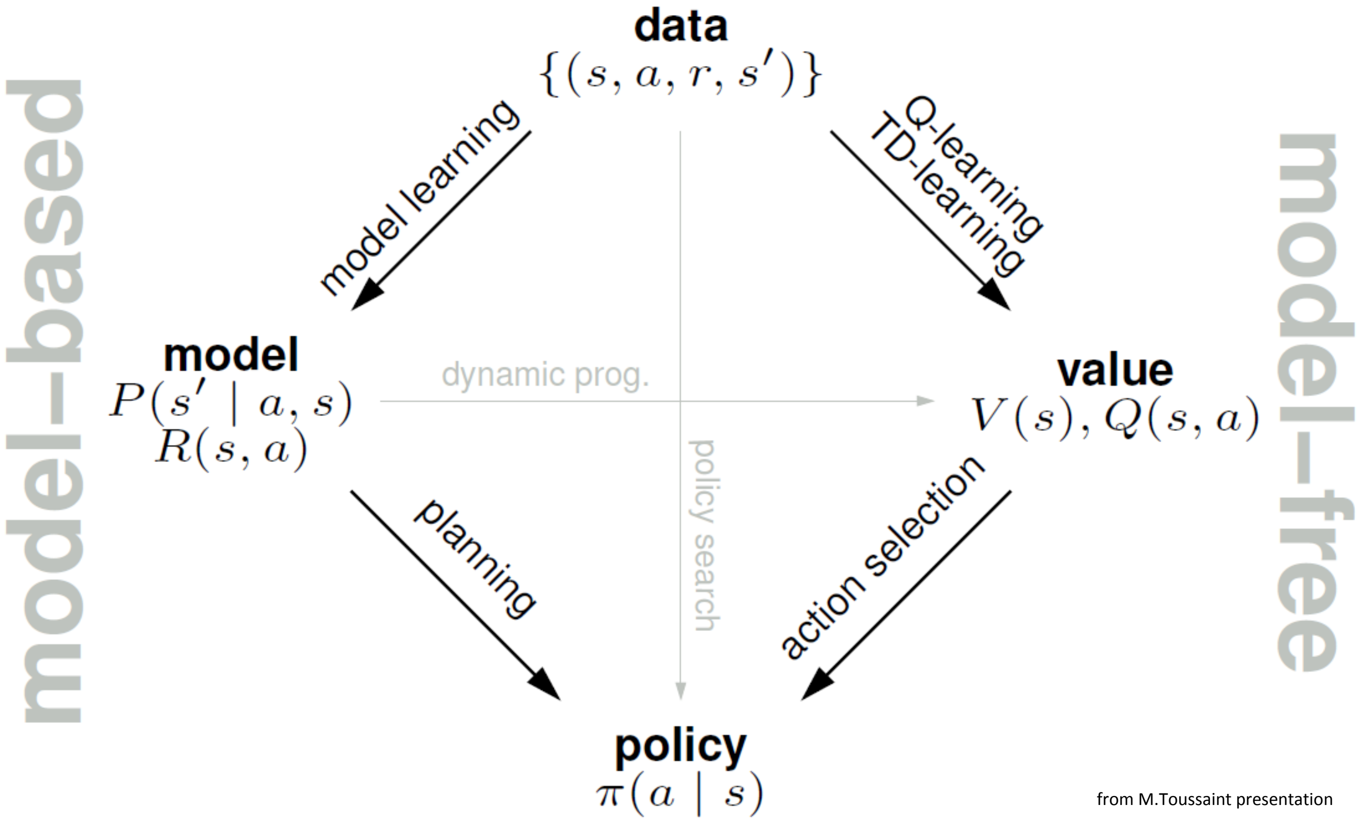
What can we learn?

We have a sequence of data:

(s,a,r,s')=(*state, action, immediate reward, next state*)

- **Model-based RL**
  - learn to predict next state, i.e. estimate $T(s'|s, a)$
  - learn to predict immediate reward, i.e. estimate $p(r|s,a)$
- **Model-free RL**
  - learn to predict value of state $V(s)$ or value of action state pair $V(s, a)$
- **Direct policy search**
  - performs policy evaluation

Learning in MDP



from M.Toussaint presentation

Model-Based RL

- Learn the model empirically via growing experience (s,a,r,s')
  - Collect outcomes (s, a) for sufficient time (so-called learning phase)
  - Fit transition model $T'(s'|a,s)$ and estimate reward $R'(s,a)$ for instance by using empirical observed distributions (or update some prior beliefs)

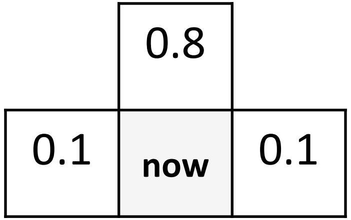
    $T'(s'|a,s) = \#(s',a,s) / \#(s,a)$ (discrete case)
- Solve the MDP with the learned approximate model (S, A, T', R') as if the learned model were correct (using standard VI with the estimated T')

Example: Adaptive/Approximate DP (learn transitions and rewards from observations then update the values of the states)

For continuous case: Least Squares Value Iteration, Stochastic Optimal Control

Grid example: passive ADP

3				+1
2				-1
1	START			
	1	2	3	4



Three training sequences of (state,action,reward):

$(1,1)_{-0.4}\rightarrow(1,2)_{-0.4}\rightarrow(1,3)_{-0.4}\rightarrow(1,2)_{-0.4}\rightarrow(1,3)_{-0.4}\rightarrow(2,3)_{-0.4}\rightarrow(3,3)_{-0.4}\rightarrow(4,3)_{+1}$

$(1,1)_{-0.4}\rightarrow(1,2)_{-0.4}\rightarrow(1,3)_{-0.4}\rightarrow(2,3)_{-0.4}\rightarrow(3,3)_{-0.4}\rightarrow(3,2)_{-0.4}\rightarrow(3,3)_{-0.4}\rightarrow(4,3)_{+1}$

$(1,1)_{-0.4}\rightarrow(2,1)_{-0.4}\rightarrow(3,1)_{-0.4}\rightarrow(3,2)_{-0.4}\rightarrow(4,2)_{-1}$

$p((1,2)|(1,3),r)=1/3$

$p((2,3)|(1,3),r)=2/3$

Substitute in $V^{\pi}(s) = R(s, a) + \gamma\Sigma_{s'}T'(s'|\pi(s), s)\, V^{\pi}(s')$

Adaptive Dynamic Programming

- Utilities of neighboring states are mutually constrained, Bellman equation:

  $V(s) = R(s) + \gamma \, \Sigma_{s'} T'(s'|a,s) \, V(s')$

- *Estimate* $T'(s'|a,s)$ *from the frequency with which* s' *is reached when executing* a *in* s.
- Can use VI: initialize utilities based on the rewards and update all values based on the above equation.
- Can be *intractable* given a big state space.

Model-free RL

No need to learn the transition model and reward function

Common approaches:

- **Direct evaluation**

- Repeatedly execute the policy
- Value of the state s = the *average sum* of discounted rewards accumulated from s onwards (over all times the state s was visited)

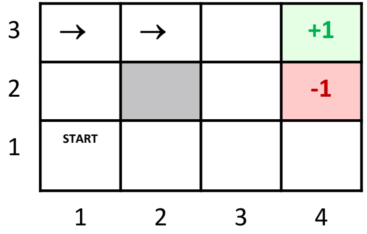
Note: easy; corrupt info about state connections; long time to learn

- Temporal Difference Learning

  Learning from every experience, update V and/or Q any transition

- Q-Learning

Direct evaluation (model-free)



- Developed in the late 1950's in the adaptive control theory.
- Rule: keep a running average of rewards for each state.

For each training sequence, compute the reward-to-go for each state in the sequence and update the utilities.

$(1,1)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (2,3)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (4,3)_{+1}$

0.72 0.76 0.8 0.84 0.88 0.92 0.96 1.0

V(1.2)=(0.76+0.84)/2=0.8

V(1.3)=(0.8+0.88)/2=0.84

Model-free RL

No need to learn the transition model and reward function

Common approaches:

- Direct evaluation

- Repeatedly execute the policy
- Value of the state s = the *average sum* of discounted rewards accumulated from s onwards (over all times the state s was visited)

Note: easy; corrupt info about state connections; long time to learn

- **Q-Learning** a Temporal Difference Learning

  Learning from every experience, update V and/or Q any transition

Model-free: Q-learning

Given new data (s,a,r,s') , compute an average *without* knowing T() and R()

$$Q^{new}(s, a) = (1-\alpha)\, Q^{old}(s, a) + \alpha\, [r \underbrace{+ \gamma \max_{a'} Q^{old}(s', a')]}_{\approx Q(s,a)}$$

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

Reinforcement

- more reward than expected $r > Q^{old}(s,a) - \gamma \max_{a'} Q^{old}(s', a') \Rightarrow$ increase $Q(s,a)$
- less reward than expected $r < Q^{old}(s,a) - \gamma \max_{a'} Q^{old}(s', a') \Rightarrow$ decrease $Q(s,a)$

Notes:

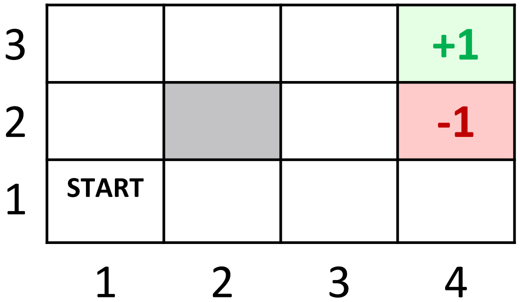
Q-learning is off-policy as agent estimates $Q(s, a)$ while executing π

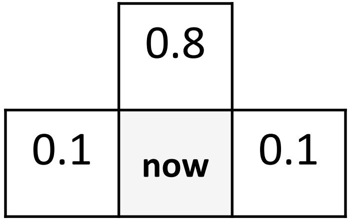
Q-Learning is the first provably convergent direct adaptive optimal control algorithm

Automatically focuses on the proper part of the state space, have to explore enough

More efficient using *eligibility traces* (V may depend on sequence of states; *all* values where you've been recently are updated)

Grid example





Non-deterministic actions (transition model and reward function are unknown to the agent)

Every state except of terminal states has reward -0.04; action = {left, right, up}

Given policy π. Follow the policy for many epochs

Three training sequences of (state,action,reward):

$(1,1)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (2,3)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (4,3)_{+1}$

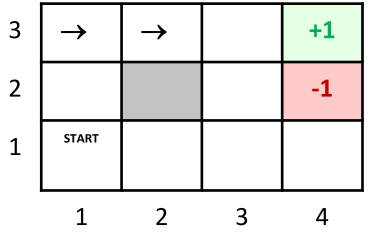
$(1,1)_{-0.4} \rightarrow (1,2)_{-0.4} \rightarrow (1,3)_{-0.4} \rightarrow (2,3)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (3,2)_{-0.4} \rightarrow (3,3)_{-0.4} \rightarrow (4,3)_{+1}$

$(1,1)_{-0.4} \rightarrow (2,1)_{-0.4} \rightarrow (3,1)_{-0.4} \rightarrow (3,2)_{-0.4} \rightarrow (4,2)_{-1}$

Q-learning

For each s and a initialise $Q(s,a)$ (can be 0 or random). Observe current state s.
Make a loop:

- select a and receive immediate reward r
- observe new state s'
- Update $Q(s,a) = Q(s,a) + \alpha(r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- set $s = s'$



In grid example (r=0 for non-terminal states)

$Q((1,3),right) = Q((1,3),right) + \alpha(r((1,3)) + \gamma \max_{a'} Q((2,3),a') - Q((1,3),right))$

Approximate RL

Q-learning needs all to keep all Q-values **=>** not realistic as too many states

- to experience all of them
- to store the values in memory

To avoid this problem => generalisation is used, i.e. knowledge gained on a small training data is transferred to similar situations.

- Use feature-based representation with *feature function* reflecting important property of the state
- Approximate $Q(s,a) = \Sigma_i w_i f_i(s,a)$, then update weights of important/active features rather than $Q(s,a)$

  $w_i^{new} = w_i^{old} + \alpha\, [r - w^{old}(s, a) + \gamma \max_{a'} w_i^{old}(s', a')]\, f_i(s,a)$

Note: Often very effective, helps RL scale up to very large MDPs, but convergence is *not* guaranteed

Model-free RL

No need to learn the transition model and reward function

Common approaches:

- Direct evaluation
  - Repeatedly execute the policy
  - Value of the state s = the *average sum* of discounted rewards accumulated from s onwards (over all times the state s was visited)

Note: easy; corrupt info about state connections; long time to learn

- Q-Learning a **Temporal Difference Learning**

  Learning from every experience, update V and/or Q any transition

Model-free: SARSA (TD-learning of $Q(s,a)$)

Given new data (s,a,r,s',a'), where $a' = \pi(s')$

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

$$= Q^{old}(s, a) + \alpha\, [r(s',a,s) + \gamma Q^{old}(s', a') - Q^{old}(s, a)],$$

Reinforcement

- more reward than expected $r > Q^{old}(s,a) - \gamma Q^{old}(s',a') \Rightarrow$ increase $Q(s,a)$
- less reward than expected $r < Q^{old}(s,a) - \gamma Q^{old}(s',a') \Rightarrow$ decrease $Q(s,a)$

TD update adjusts Q estimate to agree with Bellman equation

Typical parameter values:

- discount factor $\gamma > 0{,}9$
- learning rate $\alpha < 0.5$ ($\alpha \in (0, 1]$) or decrease with time
- exploration probability $\varepsilon \approx 0.1$ or decrease with time
- decay rate for eligibility traces $\lambda \in [0.5, 0{,}9]$ ($\lambda \in [0, 1]$)

Exploration-Exploitation: examples

- call a taxi service you know or try a new one

- select your favourite pub/club or try a new

- play a move in a game you know as the best or play experimental (risky) move
- ‘classical’ example of E2 problem: *multi-armed bandit*

E2: problem or dilemma?

To ensure convergence, RL methods need to sample all actions at every state sufficiently often

- Execute best estimated action, i.e action with the highest value => exploit
  - Note: the learned model can never be the real one => suboptimal results
- Try an action with lower estimated value (or random action) for which we may gather more useful knowledge => explore
  - Agent can learn the very precise model
  - It can be of use if some parts of model are never used

Problem:

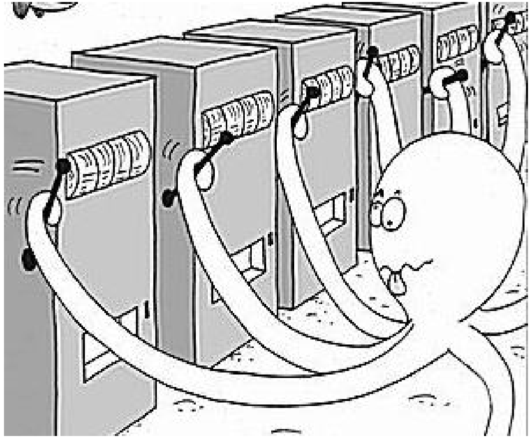
- The best long-term strategy may involve short-term *sacrices*
- Gather *enough* information to make the best overall decisions

Balance exploitation-exploration is important.

Multi armed bandit: example of E2

- **A** – known set of m actions (# of arms)
- $R(r, a) = Pr(r|a)$ unknown probability distribution over rewards
- Agent selects action (arm) $a_t \in \mathbf{A}$, bandit generates a reward r_t
- Goal: maximize $\Sigma_t r_t$
- Rewards of not chosen actions are *unknown*

To solve the n-armed bandit problem: you must explore a variety of actions and exploit the best actions

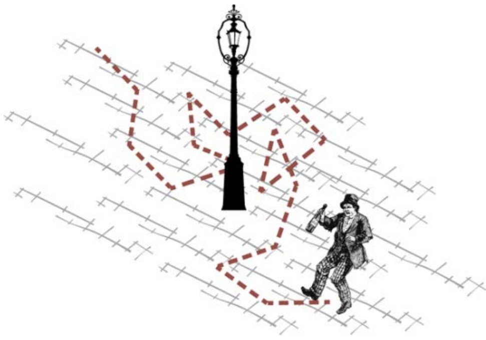


Tradeoff: immediate vs. long-term profit

Problem: choosing actions with the highest expected utility ignores their contribution to learning.

- A *random-walk agent* learns faster but never uses that knowledge.
- A *greedy agent* learns very slowly and acts based on current, inaccurate knowledge.

Exploitation- Exploration tradeoff crucial for performance of online RL.



Some principles solving the E2 dilemma

- Naive exploration

  Adding noise to greedy policy (e.g. ε-greedy: choose best $a^* = \arg\max_a Q(s,a)$ with probability $1/[\varepsilon(n-1)]$, n – is a number of trials, otherwise execute random action) or even simpler: execute random action with probability ε, and best with $(1-\varepsilon)$

- Optimistic initialisation

  Assume the best until proven otherwise, e.g. initialise $Q(a)$ to high value

- Optimism in the Face of Uncertainty

  Prefer actions with uncertain values, e.g. more uncertainty about an action value=> more important to explore that action

- Information State Search

  Search (ahead) incorporating value of information

.. and many other heuristic and theoretically sound variants.

E2 dilemma: value of info

If we know *value of information*, we can trade-off exploration and exploitation *optimally*

- Exploration gains information, but how to quantify the value of this info?
- How much reward does an agent want to give up to get that info?
- Uncertain situations imply higher information gains
- Exploring uncertain situations more is reasonable

Optimal exploration strategy: utopia or not?

What does optimal mean? Try to formalize exploration as POMDP (policy learning is trivial for bandits)

- Use goodness measure of exploration step: measure regret of the total mistake:

  regret=*expected reward of best action* – *actual reward of taken action*

  Keep track of average reward for each action; exploit: take $a=\text{argmin}_a\ regret(a)$

  But: optimal exploration has higher regret than optimal exploitation
- Bayesian model identification (see Lecture 7) + explicitly reason about value of information (small grid problems)
- In large or infinite MDP – intractable, optimal methods do not work ☹ Way out: solve using small-scale methods

Other common exploration methods

- Boltzmann exploration: execute each *a* with $p(a) = e^{Q(s,a)/T} / \Sigma_a e^{Q(s,a)/T}$

  parameter $T \rightarrow \infty$ means all actions equally likely, $T \rightarrow 0$ means a^* will be chosen with near certainty

  (states with lower energy will always have a higher probability of being occupied than the states with higher energy)

- Confidence-based methods: use statistical tests to estimate a confidence bounds on estimated Q-values, then choose actions based on mean values but with an "exploration bonus" if high uncertainty (e.g.,large upper confidence bound)

Exploration and Q-learning

Q-learning converges to optimal Q-values even if the agent acting suboptimally and if

- Every state is visited infinitely often (due to exploration)
- The action selection becomes greedy as time approaches infinity
- The learning rate a is decreased fast enough but not too fast
- At early stages of learning, estimations can be unrealistic low
- In the early phase of search agent is more willing to explore.

Bayesian RL

- dealing with uncertainty, where ‘classic RL’ does not.
- includes modelling the transition-function (value-function, policy, reward) *probabilistically*.

Bayesian RL:

- solves E2 by planning in belief space.
- is generally computationally intractable, but approximations exist
- efficiently chooses samples to learn from
- suitable when sample cost is high.

Bayesian Reinforcement Learning

MDP is unknown, but can be learned based on experience

- Let MDP be parameterised as $T(s'|s,a)=\theta_{s',a,s.}$ Then having experience $(s_1,a_1,\ldots, s_t,a_t)$ the posterior $b(\theta)=p(\theta|s_1, a_1,\ldots, s_t,a_t)$ can be estimated.
- given a posterior belief b about MDP we plan to maximise policy value in this distribution of MDPs:

$$V^{opt}(s,a) = \max_a \left[R(s,a) + \gamma \sum_{s'} \int_{\theta} T(s'|,a,s,\theta) b(\theta) d\theta \, V^{opt}(s',b') \right]$$

details see for instance freely available: M. Ghavamzadeh et al. *Bayesian Reinforcement Learning: A Survey* Foundations and Trends in Machine Learning Vol.8, No.5-6 359–483 2015

Bayesian RL

- A prior distribution over model parameters is available
- Updates distribution based on observed transitions
- Chooses actions with greatest expected long-term value
- Has *no* exploration-exploitation dilemma as solves it *optimally*
- Difficult to work with large POMDP, with high-dimensional continuous state space => tractability & efficiency

What can RL do well now?

- To learn from human-expert training behaviour .

  drive a car

- To learn simple abilities (skills) with noisy observations (having rich experience at disposal)

  robot-manipulator

- To provide high quality results in domains with simple and known rules

  play a game

What remains to be a challenge?

- Not clear what the reward function should be
- Not clear what the role of prediction should be
- Speed of RL is very far from humans
- Transfer learning in RL remains to be an open problem
- E2
- Generalisation over state and actions
- Partial observability
- Multiple-agents
- Non-stationary reward and transition models
- Proof of convergence, suboptimality conditions (convergence and optimality are difficult to achieve when state spaces are large)
- …

Beyond RL

- Inverse RL: learning reward function from experience
  => preference elicitation

- Transfer learning: transfer knowledge between different examples/domains
- Meta-learning: learning to learn

Transfer learning

- use experience from one set (type) of problems (source domain) for faster learning and better performance in a new task (target domain)
- The more diversity we observe in source domain, the richer knowledge we transfer => randomisation
- Ensures that the differences are functionally irrelevant, it is not granted
- Tools: physical rules => models (model-based RL); policies; learning methods

Meta-learning

- Closely related to multi-task learning
- Use past experience to find more efficient deep RL algorithm
- Learning to learn how to:
  - explore more efficiently
  - avoid actions that bring no reward
  - acquire the proper features more quickly

hard optimisation problem; works well in smaller tasks.

Problem with large MDPs:

RL can solve large problems (e.g. backgammon 1020 states; computer Go 10170 states), but

- too many states and/or actions to store (limited memory)
- to learn the value of each state individually needs lot of time

Way out:

- generalise observed states to yet unobserved
- estimate V or Q with function approximation:
  - linear combinations of features
  - neural networks
  - nearest neighbour
  - wavelet-based, …

Deep RL: what is it?

Deep RL= RL + NN

RL uses neural networks to approximate:

- Policies (select next action)
- Value functions (measure goodness of states or state-action pairs)
- Models (predict next states and rewards)

What has proven to be a challenge in Deep RL ?

- Human able to learn quickly. Compare to them Deep RL is *slow*
- Transfer learning (use past knowledge/experience) in deep RL is *not* covered
- Form and importance of reward

Deep RL is good now (beginning 2018)

- in domains with simple and fixed rules (Go, ATARI)
- to learn from human-expert behaviour, i.e. learn from imitating (robots, driving, etc)
- to learn simple abilities from rich experience (robots)

- Picture sources https://dir.indiamart.com/, http://gigabotics.com, https://github.com/