– SUS 2020–
Lecture 12: Reinforcement Learning: Common Approaches

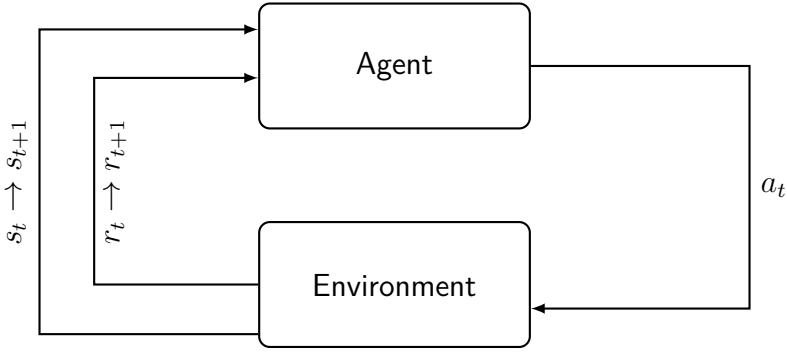
Hung Son Nguyen

MIM
University of Warsaw

Based on slides by Zoubin Ghahramani and Carl Edward Rasmussen

Outline

Intelligent agent systems



The Agent at each step t receives a representation of the environment's *state*, $s_t \in S$ and it selects an action $a_t \in A(s_t)$. Then, as a consequence of its action the agent receives a *reward*, $r_{t+1} \in R \in \mathbb{R}$.

Markov Decision Process (MDPs)

MDP is a 5-tuple $(S, A, \mathcal{P}, \mathcal{R}, \gamma)$ where:

S	finite set of states: $s \in S$
A	finite set of actions: $a \in A$
$\mathcal{P}$	state transition probabilities: $\mathcal{P}^a_{ss'} = Pr(s_{t+1} = s' \vert s_t = s, a_t = a)$
$\mathcal{R}$	expected reward for state-action-nexstate: $\mathcal{R}^a_{ss'} = \mathbb{E}[r_{t+1} \vert s_t = s, a_t = a, s_{t+1} = s']$
γ	the discount factor : $\gamma \in [0, 1]$

(1)

Policy and total reward

A *policy* (or policy map) describes the probability of taking action a when in state s

$$\pi : A \times S \to [0, 1]$$
$$\pi(a, s) = Pr(a_t = a | s_t = s)$$

The total *reward* is expressed as:

$$R_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite.

Value (V) function

Value function describes *how good* is to be in a specific state s under a certain policy π. For MDP:

$$V^{\pi}(s) \;=\; \mathbb{E}_{\pi}[R_t|s_t = s] \;=\; \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\Big| s_t = s\Big]. \tag{3}$$

Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

Optimal value

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{4}$$

Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$Q^{\pi}(s,a) \;=\; \mathbb{E}_{\pi}[R_t|s_t=s,a_t=a] \;=\; \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\Big|s_t=s,a_t=a\Big]. \quad (5)$$

The optimal value-action function:

$$q_*(s,a)=\max_{\pi} q^{\pi}(s,a) \quad (6)$$

Clearly, using this new notation we can redefine V^*, equation 4, using $Q^*(s,a)$, equation 6:

$$V^*(s)=\max_{a\in A(s)} Q^{\pi *}(s,a) \quad (7)$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

Optimal Policies and Values

- **Optimal Policy:** π^* such that $V^{\pi^*}(s) \geq V^{\pi}(s)\ \forall s$.
  There may be more than one optimal policy.
  Question: Is there always at least one optimal policy? YES
- **Optimal state value function**:

$$V^*(s) = \max_{\pi} V^{\pi}(s)\ \forall s$$

- **Optimal state-action value function**:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)\ \forall s.$$

  This is the expected return of action a in state s, thereafter following optimal policy.

$$Q^*(s,a) \ = \ \mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a].$$

Bellman Optimality Equation

N *nonlinear* equations in N unknowns for V^*.

$$\begin{aligned} V^*(s) &= \max_a Q^{\pi^*}(s,a) = \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right] \\ &= \max_a \mathbb{E}_{\pi^*}\left[r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right] \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'}\Big(\mathcal{R}^a_{ss'} + \gamma V^*(s')\Big) \end{aligned}$$

NA *nonlinear* equations in NA unknowns for Q^*

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| s_t = s, a_t = a\right] \\ &= \sum_{s'} \mathcal{P}^a_{ss'}\Big(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\Big) \end{aligned}$$

Outline

Dynamic programming (DP) algorithm

This applied the Generalized Policy Iteration (GPI):

$$\pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{imp}} \pi_1 \xrightarrow{\text{eval}} V_{\pi_1} \xrightarrow{\text{imp}} \pi_2 \xrightarrow{\text{eval}} \dots \xrightarrow{\text{imp}} \pi_* \xrightarrow{\text{eval}} V_*$$

Policy improvement step:

$$\begin{aligned} \pi'(s) &\leftarrow \arg\max_a Q^\pi(s,a) \quad \forall s \\ &= \arg\max_a \mathbb{E}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a] \end{aligned}$$

Value evaluation step:

$$\begin{aligned} V_{k+1}(s) &\leftarrow \max_a \mathbb{E}[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \Big(\mathcal{R}^a_{ss'} + \gamma V_k(s') \Big) \end{aligned}$$

Algorithm 1: Policy Iteration

begin

1. Initialisation

$V(s) \in \mathbb{R}$, (e.g $V(s) = 0$) and $\pi(s) \in A$ for all $s \in S$,

$\Delta \leftarrow 0$

2. Policy Evaluation

while $\Delta \geq \theta$ *(a small positive number)* **do**

foreach $s \in S$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

3. Policy Improvement

policy-stable $\leftarrow$ *true*

foreach $s \in S$ **do**

old-action $\leftarrow \pi(s)$ $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$

policy-stable $\leftarrow$ *old-action* $= \pi(s)$

if *policy-stable* return V $\approx V_*$ and $\pi \approx \pi_*$, else go to 2

Value Iteration

- Do we really need to wait until convergence of the evaluation step?
- In fact, we can improve after **one** sweep of evaluation!

$$\begin{aligned} V_{k+1}(s) &\leftarrow \max_a \mathbb{E}[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_k(s') \right) \end{aligned}$$

converges: $V_k \longrightarrow V^*$.
At each step we also have a policy.

- Problem: it is still not feasible to update the value of every single state. E.g. backgammon has 10^{20} states!
- Bellman called this the **curse of dimensionality**

Algorithm 2: Value Iteration

begin

Initialise $V(s) \in \mathbb{R}$, e.g $V(s) = 0$

$\Delta \leftarrow 0$

while $\Delta \geq \theta$ *(a small positive number)* **do**

foreach $s \in S$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

ouput: Deterministic policy $\pi \approx \pi_*$ such that $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right]$

Asynchronous dynamic programming

These are in-place iterated dynamic programming (DP) algorithms that are not organized in terms of systematic sweeps over all the states.

- *States are backed-up in order visited or randomly.*
  To converge the algorithms must continue to visit every state.
- Key idea in RL: We can run the DP algorithm at the same time as the agent is *actually experiencing* the MDP.
- This leads to an **exploration vs exploitation tradeoff**: act so as to visit new parts of state space or exploit already visited part of state-space?
- An example of a simple exploration strategy are ε-greedy policies:

$$\pi_\varepsilon(s,a) = (1-\varepsilon)\pi(s,a) + \varepsilon \cdot u(a)$$

  where $u(a)$ is a uniform distribution over actions.
- *Can you think of anything wrong with this?*

Monte-Carlo Methods

Recall: $V^{\pi}(s) = \mathbb{E}_{\pi}[R_t|s_t = s]$.

Monte-Carlo (MC) methods uses a simple idea: It learns from episodes of raw experience without modeling the environmental dynamics and computes the observed mean return as an approximation of the expected return.

To compute the empirical return R_t, MC methods need to learn from complete episodes: $s_1, a_1, r_1, ..., s_T$ to compute $R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$.

The empirical mean return for state s is:

$$V(s) = \frac{\sum_{t=1}^{T} \mathbf{1}_{[s_t=s]} R_t}{\sum_{t=1}^{T} \mathbf{1}_{[s_t=s]}}$$

We may count the visit of state s every time so that there could exist multiple visits of one state in one episode ("every-visit"), or only count it the first time we encounter a state in one episode ("first-visit"). Thus:

$$Q(s,a) = \frac{\sum_{t=1}^{T} \mathbf{1}_{[s_t=s,a_t=a]} R_t}{\sum_{t=1}^{T} \mathbf{1}_{[s_t=s,a_t=a]}}$$

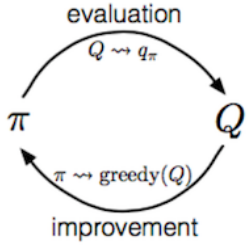
Monte Carlo Method

1. Improve the policy greedily with respect to the current value function:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

2. Generate a new episode with the new policy π (i.e. using algorithms like ε-greedy helps us balance between exploitation and exploration.)
3. Estimate Q using the new episode:

$$q_\pi(s, a) = \frac{\sum_{t=1}^{T} \left(\mathbf{1}_{[s_t = s, a_t = a]} \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} \mathbf{1}_{[s_t = s, a_t = a]}}$$



Algorithm 3: Monte Carlo first-visit

begin

Initialise for all $s \in S, a \in A(s)$:

$Q(s,a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s,a) \leftarrow$ empty list

while *forever* **do**

Choose $S_0 \in S$ and $A_0 \in A(S_0)$, all pairs have probability > 0

Generate an episode starting at S_0, A_0 following π **foreach** *pair* s, a *appearing in the episode* **do**

$R \leftarrow$ return following the first occurrence of s, a

Append G to $Returns(s,a))$

$Q(s,a) \leftarrow average(Returns(s,a))$

foreach s *in the episode* **do**

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Temporal-Difference Learning

- Similar to Monte-Carlo methods, Temporal-Difference (TD) Learning is model-free and learns from episodes of experience.
- However, TD learning can learn from incomplete episodes and hence we don't need to track the episode up to termination.
- TD learning is so important that Sutton & Barto (2017) in their RL book describes it as "one idea ... central and novel to reinforcement learning".
- TD learning methods update targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns as in MC methods. This approach is known as bootstrapping.
- The key idea in TD learning is to update the value function $V(s_t)$ towards an estimated return

$$R_{t+1} + \gamma V(s_{t+1})$$

(known as "TD target")

Monte Carlo and TD

- Question: how do you trade off length of sampled trajectory, vs previously estimated values?

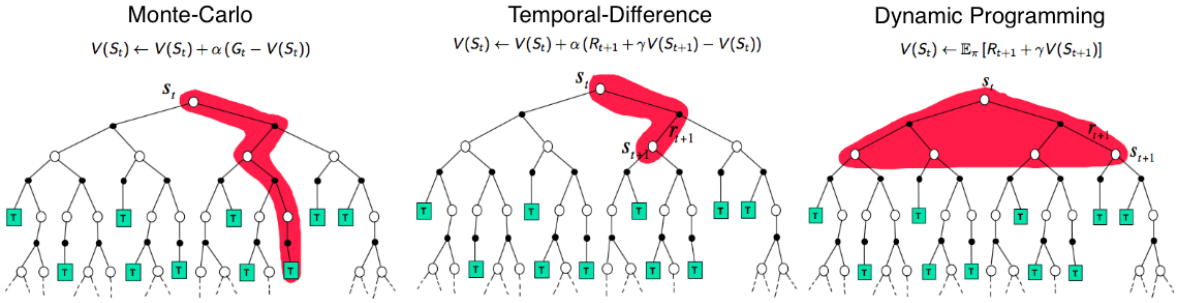
$$
\begin{aligned}
V(s_t) &\leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad (8)\\
V(s_t) &\leftarrow V(s_t) + \alpha[r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - V(s_t)]\\
V(s_t) &\leftarrow V(s_t) + \alpha[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3}) - V(s_t)]\\
&\vdots\\
V(s_t) &\leftarrow V(s_t) + \alpha[R_t - V(s_t)] \qquad (9)
\end{aligned}
$$

- Equation (8) is **Temporal Difference learning, TD(0)**. TD(λ) approximates the range eqn (8)–(9), where higher λ is closer to the full MC method.
- TD(λ) has been proven to converge
- These are general methods for controlling the **bias-variance tradeoff**.
- Similarly, for action-value estimation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Comparison

Comparison of the backup diagrams of Monte-Carlo, Temporal-Difference learning, and Dynamic Programming for state value functions.



Some Temporal-Difference Learning Methods:

- SARSA: On-Policy TD control
- Q-Learning: Off-policy TD control
- Deep Q-Network: substitutes the Q function with a deep neural network called Q-network.

SARSA - State-action-reward-state-action)

Definitions:

- *On-policy* methods evaluate or improve the current policy used for control.
- *Off-policy* methods evaluate or improve one policy, while acting using another (behavior) policy.

In off-policy methods, the behavior policy must have non-zero probability for each state-action the evaluated policy does.

SARSA:

is on-policy greedy control

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

n-step Sarsa:

Define the n-step Q-Return

$$q^{(n)} = R_{t+1} + \gamma Rt + 2 + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa update $Q(S, a)$ towards the n-step Q-return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^{(n)} - Q(s_t, a_t) \right]$$

Forward View Sarsa(λ):

$$q_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

Forward-view Sarsa(λ):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^{\lambda} - Q(s_t, a_t) \right]$$

Algorithm 4: Sarsa(λ)

begin

Initialise $Q(s, a)$ arbitrarily and $Q(terminal - state, \cdot) = 0$

foreach *episode* $\in$ *episodes* **do**

Choose a from s using policy derived from Q (e.g., ε-greedy)

while s *is not terminal* **do**

Take action a, observer r, s'

Choose a' from s' using policy derived from Q (e.g., ε-greedy)

$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$

$s \leftarrow s'$

$a \leftarrow a'$

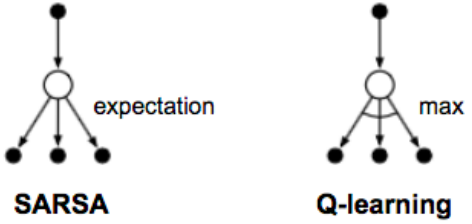
Q Learning

Q Learning: off-policy greedy control

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Converges if $\forall a, s$ are visited and updated infinitely often.

We can also combine the bias-variance ideas with Q and SARSA, to get $Q(\lambda)$ and SARSA(λ).



Algorithm 5: Q Learning

begin

Initialise $Q(s, a)$ arbitrarily and $Q(terminal - state, \mathring{u}) = 0$

foreach *episode* $\in$ *episodes* **do**

while s *is not terminal* **do**

Choose a from s using policy derived from Q (e.g., ϵ-greedy)

Take action a, observer r, s'

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

$s \leftarrow s'$

Deep Q Learning

- Created by $DeepMind$, Deep Q Learning, DQL,
- substitutes the Q function with a deep neural network called Q-network.
- It also keep track of some observation in a $memory$ in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\underbrace{(r + \gamma \max_a Q(s', a'; \theta_{i-1})}_{\text{target}} - \underbrace{Q(s, a; \theta_i))}_{\text{prediction}}^2\right] \quad (10)$$

Where θ are the weights of the network and $U(D)$ is the experience replay history.

Algorithm 6: Deep Q Learning

begin

Initialise replay memory D with capacity N

Initialise $Q(s, a)$ arbitrarily

foreach *episode* $\in$ *episodes* **do**

while s *is not terminal* **do**

With probability ϵ select a random action $a \in A(s)$

otherwise select $a = \max_a Q(s, a; \theta)$

Take action a, observer r, s'

Store transition (s, a, r, s') in D

Sample random minibatch of transitions (s_j, a_j, r_j, s'_j) from D

Set $y_j \leftarrow \begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$

Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$

$s \leftarrow s'$

Function Approximation

For very large or continuous state spaces it is hopeless to store a table with all the state values or state-action values.

It makes sense to use **function approximation**

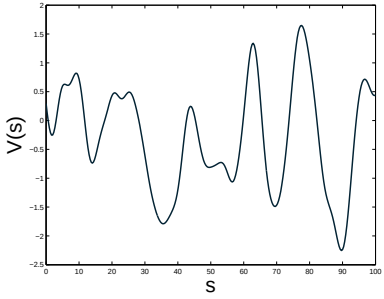
$$V(s) = f_\theta(s)$$

e.g. basis function representation:

$$V(s) = \sum_i \theta_i \phi_i(s)$$

Similarly for $Q(s, a)$.

This should hopefully lead to better **generalization**



Function Approximation

Gradient descent methods:

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \alpha[v_t - V_t(s_t)]\frac{\partial V_t(s_t)}{\partial \boldsymbol{\theta}}$$

where α is a learning rate and v_t is a measured/estimated value. See chapter 8 of Sutton and Barto.

Optimal Control

Optimal Control: The engineering field of optimal control covers exactly the same topics as RL, except the state and action space is usually assumed to be continuous, and the model is often known.

The **Hamilton-Jacobi-Bellman** optimality conditions are the continuous state generalization of the Bellman equations.

A typical elementary problem in optimal control is the linear quadratic Gaussian control **LQG** problem. Here the cost function is quadratic in states $\mathbf{x}_t$ and actions $\mathbf{u}_t$, and the system is a linear-Gaussian state-space model.

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \epsilon_t$$

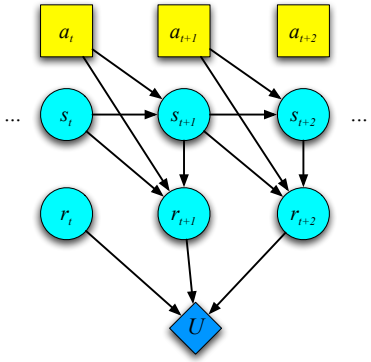
For this model the optimal policy can be computed from the estimated state. It's a linear feedback controller:

$$\mathbf{u}_t = L\hat{\mathbf{x}}_t$$

The optimal policy here happens not depend on the uncertainty in $\mathbf{x}_t$. This is not generally the case.

Influence Diagrams

You can extend the framework of directed acyclic probabilistic graphical models (a.k.a. Bayesian networks) to include **decision nodes** and **value nodes**. These are called **influence diagrams**. Solving an influence diagram corresponds to finding the settings of the decision nodes that maximize the expectation of the value node.

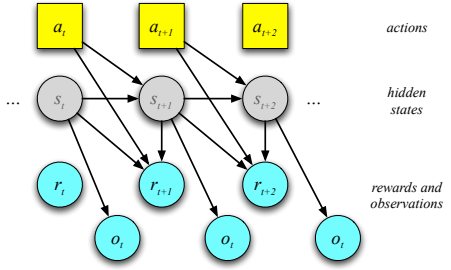


It is possible to convert the problem of solving an influence diagram into the problem of doing inference in a (usually multiply connected) graphical model (Shachter and Peot, 1992). Exact solutions can be computationally intractable.

Like other graphical models, influence diagrams can contain both observed and **hidden** variables...

POMDPs

POMDP = Partially-observable Markov decision problem.
The agent does not observe the full state of the environment.
What is the optimal policy?



- If the agent has the correct model of the world, it turns out that the optimal policy is a (piece-wise linear) function of the **belief state**, $P(s_t|a_1, \ldots, a_{t-1}, r_1, \ldots, r_t, o_1, \ldots, o_t)$.
  Unfortunately, the belief state can grow exponentially complex.
- Equivalently, we can view the optimal policy as being a function of the entire sequence of past actions and observations (this is the usual way the policy in influence diagrams is represented).
  Unfortunately, the set of possible such sequences grows exponentially.

Efficient methods for approximately solving POMDPs is an active research area.

RL summary

Central ideas in reinforcement learning

- the difference between *reward* and *value*
- Bellman consistency of values
  $V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \Big(\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \Big)$
- Policy Iteration = Policy Evaluation + Policy Improvement
- Requires knowledge of and represenatation of transitions and rewards
- Key idea: run the algorithms as we're experiencing the MDP
- on-policy and off-policy methods (SARSA and Q-learning) don't require knowledge of transition probabilities and rewards
- exploration vs. exploitation

Some References on Reinforcement Learning

- Kaelbling, L.P., Littman, M.L. and Moore, A.W. (1996) Reinforcement Learning: A Survey. Journal of Aritificial Intelligence Research 4:237-285.
- Sutton, R.S. and Barto, A.G. (2000) Reinforcement Learning: An Introduction. MIT Press.
  `http://www.cs.ualberta.ca/~sutton/book/ebook`
- Bertsekas, D.P. and Tsitsiklis, J.N. (1996) Neuro-Dynamic Programming. Athena Scientific.
- Bryson, A.E. and Ho, Y.-C. (1975) Applied Optimal Control. Hemisphere Publishing Corp. Washington DC.