# Reinforcement Learning

- MDP Slides (./files/mdps.pdf) from Prof. Zico Kolter (http://www.cs.cmu.edu/~15780/) at CMU
- MDP Slides (./files/mdp09.pdf), DMP tutorial (http://www.autonlab.org/tutorials/mdp.html) from Prof. Andrew W. Moore (http://www.cs.cmu.edu/~awm/tutorials.html) at CMU
- RL Slides (./files/rl.pdf) from Prof. Zico Kolter (http://www.cs.cmu.edu/~15780/) at CMU

Collected by Prof. Seungchul Lee
iSystems Design Lab
UNIST
http://isystems.unist.ac.kr/

Table of Contents



## 1. Markov Decision Processes

**Decision making under uncertainty**

- Markov decision processes (MDP) and their extensions provide an extremely generaly way to think about how we can act optimally under uncertainty.
- For many medium-sized problems, we can use the techniques from this lecture to compute an optimal decision policy.

**Markov decision processes**

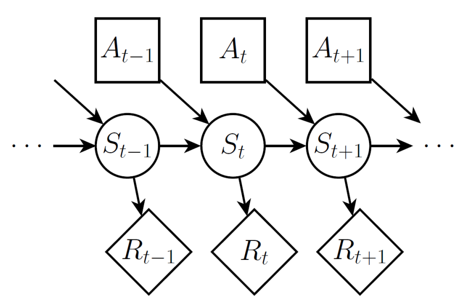
- MDP is defined by: states, actions, transition probabilities, and rewards
- States encode all information of a system needed to determine how it will evolve when taking acitions, with system governed by the state transition probabilities

$$P(s_{t+1} \mid s_t, a_t)$$

Note that transitions only depend on current state and action, not past states/actions (Markov assumption)

- Goal for an agent is to take actions that maximize expected reward.

**Graphical model representation of MDP**




### 1.1. Formal MDP definition

A Markov decision process is defined by:

- A set of *states* $S$ (assumed for now to be discrete)
- A set of *actions* $A$ (also assumed discrete)
- *Transition probabilites* $P$, which defined the probability distribution over next states given the current state and current action

$$P(S_{t+1} \mid S_t, A_t)$$

- Transitions only depend on the *current* state and action (Markov assumption)
- A *reward* function $R : S \rightarrow \mathbb{R}$, mapping states to real numbers (can also define rewards over state/action pairs)

## 1.2. Policies and value functions

A *policy* is a mapping from states to actions $\pi : S \rightarrow A$ (can also define stochastic policeis)

A *value* function for a policy, written $V^\pi : S \rightarrow \mathbb{R}$ gives the exptected sum of discounted rewards when acting under that policy

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s, a_t = \pi(s_t), s_{t+1} \mid s_t, a_t \sim P\right]$$

where $\gamma < 1$ is a *discount factor* (also formulations for finite horizion, infinite horizon average reward)

Can also define value function recursively via the **Bellman equation**

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P\left(P(s' \mid s, \pi(s))\right) V^\pi(s')$$

### Computing the policy value

Let $v^\pi \in \mathbb{R}^{|S|}$ be a vector of values for each state, $r \in \mathbb{R}^{|S|}$ be a vector of rewards for each state

Let $P^\pi \in \mathbb{R}^{|S| \times |S|}$ be a matrix containing probabilities for each transition under policy $\pi$

$$P^\pi_{ij} = P\left(s_{t+1} = i \mid s_t = j, a_t = \pi(s_t)\right)$$

Then Bellman equation can be written in vector form as

$$\begin{aligned} v^\pi &= r + \gamma P^\pi v^\pi \\ (I - \gamma P^\pi) v^\pi &= r \\ v^\pi &= (I - \gamma P^\pi)^{-1} r \end{aligned}$$

*i.e.*, computing value for a policy requires solving a linear sytsem, but expensive

### Optimal policy and value function

The *optimal policy* is the policy that achieves the highest value for every state

$$\pi^* = \arg\max_\pi V^\pi(s)$$

and its value function is written $V^* = V^{\pi^*}$ (but there are an exponential number of policies, so this formulation is not very useful)

Instead, we can directly define the optimal value function using the **Bellman optimality equation**

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) V^*(s')$$

and optimal policy is simply the action that attains this max

$$\pi^*(s) = \arg\max_a \sum_{s' \in S} P(s' \mid s, a) V^*(s')$$

## 1.3. Computing the optimal policy

How do we compute the optimal policy? (or equivalently, the optimal value function?)

### 1. Value iteration

Approach #1: **value iteration** repeatedly update an estimate of the optimal value function according to Bellman optimality equation

1. initialize an estimate for the value function arbitrarily

$$\hat{V}(s) \leftarrow 0, \quad \forall s \in S$$

2. repeat, update

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a)\hat{V}(s'), \quad \forall s \in S$$

### 2. Policy iteration

Another approach to computing optimal policy

Policy iteration algorithm

1. initialize policy $\hat{\pi}$ (e.g., randomly)
2. Compute value of policy, $V^{\pi}$ (e.g., via solving linear system, as discussed previously)
3. Update $\pi$ to be greedy policy with respect to $V^{\pi}$
$$\pi(s) \leftarrow \arg\max_{a} \sum_{s' \in S} P(s' \mid s, a) V^{\pi}(s')$$
4. If policy $\pi$ changed in last iteration, return to step 2

### Policy iteration or value iteration?

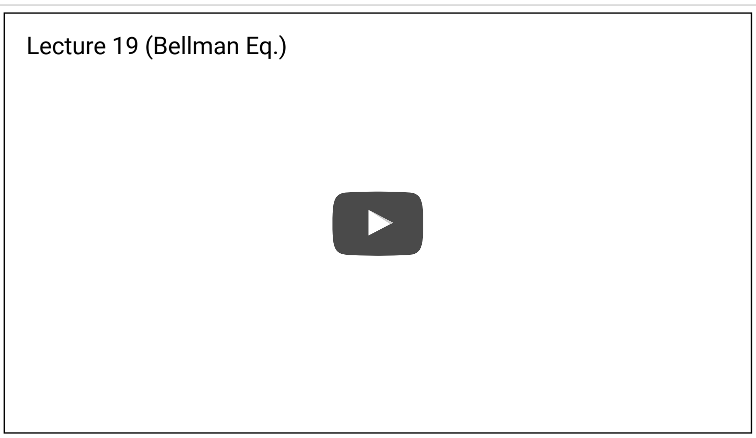
- Policy iteration requires fewer iterations that value iteration, but each iteration requires solving a linear system instead of just applying Bellman opertor
- In practice, policy iteration is often faster, especially if the transition probabilities are structured (e.g., sparse) to make solution of linear system efficient

In [2]:

```
%%html
<iframe src="https://www.youtube.com/embed/XLEpZzdLxI4"
 width="560" height="315" frameborder="0" allowfullscreen></iframe>
```

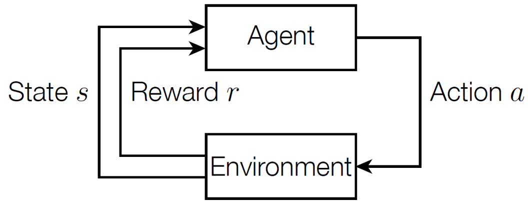

In [1]:

```
%%html
<iframe src="https://www.youtube.com/embed/4R6kDYDcq8U"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```




# 2. Reinforcement Learning

**Agent interaction with environment**




**Markov decision processes**

Recall a (discounted) Markov decision process is defined by:

$$M = (S, A, P, R)$$

- $S$: set of states
- $A$: set of actions
- $p$: $S \times S \to [0, 1]$: transition probability distribution $P(s' \mid s)$
- $R : S \to \mathbb{R}$: reward function, where $R(S)$ is reward for state $s$

The RL twist: we do not know $P$ or $R$, or they are too big to enumerate (only have the ability to act in MDP, observe states and rewards)

Policy $\pi : S \to A$ is a mapping from states to actions

We can determine the value of a policy by solving a linear systems, or via the iteration (similar to a value iteration, but for a fixed policy)

$$\hat{V}^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s' \in S} P(s' \mid s, \pi(s)) \hat{V}^{\pi}(s'), \quad \forall s \in S$$

We can determine the value of *optimal* policy $V^*$ using a value iteration:

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \hat{V}(s'), \quad \forall s \in S$$

**How can we compute these quantities when $P$ and $R$ are unknown?**

- model-based RL
- model-free RL

## 2.1. Model-based RL

A simple approach: just estimate the MDP from data

Agent acts in the work (according to some policy), observes experience

$$s_1, r_1, a_1, s_2, r_2, a_2, \cdots, s_m, r_m, a_m$$

We form the empirical estimate of the MDP via the counts

$$\hat{P}(s' \mid s, a) = \frac{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s, a_i = a, s_{i+1} = s'\}}{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s, a_i = a\}}$$

$$\hat{R}(s) = \frac{\sum_{i=1}^{m} \mathbf{1}\{s_i = s\} r_i}{\sum_{i=1}^{m} \mathbf{1}\{s_i = s\}}$$

Now solve the MDP $(S, A, \hat{P}, \hat{R})$

- Will converge to correct MDP (and hence correct value function/policy) given enough samples of each state
- How can we ensure we get the "right" samples? (a challenging problem for all methods we present here, stay tuned)
- Advantages (informally): makes "efficient" use of data
- Disadvantages: requires we build the actual MDP models, not much help if state space is too large

## 2.2. Model-free RL

Temporal difference methods (TD, SARSA, Q-learning): directly learn value function $V^\pi$ or $V^*$ (or a slight generalization of value function, that we will see shortly)

Direct policy search: directly learn optimal policy $\pi^*$ (covered in a later lecture)

### 2.2.1. Temporal difference (TD) methods

Let's consider computing the value function for a fixed policy via the iteration

$$\hat{V}^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s' \in S} P(s' \mid s, \pi(s))\, \hat{V}^{\pi}(s'), \quad \forall s \in S$$

Suppose we are in some state $s_t$, receive reward $r_i$, take action $a_t = \pi(s_t)$ and end up in state $s_{t+1}$

We cannot update $\hat{V}^{\pi}$ for all $s$, but can we update for $s_t$?

$$\hat{V}^{\pi}(s_t) \leftarrow r_t + \gamma \sum_{s' \in S} P(s' \mid s, a_t) \hat{V}^{\pi}(s')$$

...No, because we still cannot compute this sum

But, $s_{t+1}$ is a sample from the distribution $P(s' \mid s_t, a_t)$, so we could perform the update

$$\hat{V}^{\pi}(s_t) \leftarrow r_t + \gamma \hat{V}^{\pi}(s_{t+1})$$

- It is too "harsh" assignment if we assume that $s_{t+1}$ is the only possible next state;
- instead "smooth" the update using some $\alpha < 1$

$$\hat{V}^{\pi}(s_t) \leftarrow (1-\alpha) \left(\hat{V}^{\pi}(s_t)\right) + \alpha \left(r_t + \gamma \hat{V}^{\pi}(s_{t+1})\right)$$

This is the ***temporal difference (TD)*** algorithm.

TD lets us learn the value function of a policy $\pi$ directly, without ever constructing the MDP.

But is this really that helpful?

- Consider trying to execute greedy policy w.r.t. estimated $\hat{V}^{\pi}$
- We need a model anyway.

### 2.2.2. SARSA and Q-learning

$Q$ function (for MDPs in general) are like value functions but defined over **state-action pairs**

$$\begin{aligned} Q^{\pi}(s,a) &= R(s) + \sum_{s' \in S} P(s' \mid s, a) Q^{\pi}(s', \pi(s')) \\ Q^{*}(s,a) &= R(s) + \sum_{s' \in S} P(s' \mid s, a) \max_{a'} Q^{*}(s', a') \\ &= R(s) + \sum_{s' \in S} P(s' \mid s, a) V^{*}(s') \end{aligned}$$

*i.e.*, $Q$ function is a value of starting state $s$, taking action $a$, and then acting according to $\pi$ (or optimally for $Q^*$)

We can easily construct analogues of value iteration or policy evaluation to construct $Q$ functions directly given an MDP.

$Q$ function leads to new TD-like methods.

As with TD, observe state $s$, reward $r$, take action $a$ (but not necessarily $a = \pi(s)$), observe next sate $s'$

- SARSA: estimate $Q^{\pi}(s,a)$

$$\hat{Q}^{\pi}(s,a) \leftarrow (1-\alpha) \left(\hat{Q}^{\pi}(s,a)\right) + \alpha \left(r_t + \gamma \hat{Q}^{\pi}(s', \pi(s'))\right)$$

- $Q$-learning: estimate $Q^{*}(s,a)$

$$\hat{Q}^{*}(s,a) \leftarrow (1-\alpha) \left(\hat{Q}^{*}(s,a)\right) + \alpha \left(r_t + \gamma \max_{a'} \hat{Q}^{*}(s', a')\right)$$

Again, these algorithms converge to true $Q^{\pi}, Q^{*}$ if all state-action pairs are seen frequently enough

The advantage of this approach is that we can now select actions without a model of MDP

- SARSA, greedy policy w.r.t. $Q^{\pi}(s,a)$

$$\pi'(s) = \max_{a} \hat{Q}^{\pi}(s,a)$$

- $Q$-learning, optimal policy

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

So with $Q$-learning, for instance, we can learn optimal policy without model MDP.

In [1]:

```
%%html
<iframe src="https://www.youtube.com/embed/un-FhSC0HfY"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```


## David Silver's Lecture

- UCL homepage for slides (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html))
- DeepMind for RL videos (https://www.youtube.com/watch?v=2pWv7GOvuf0 (https://www.youtube.com/watch?v=2pWv7GOvuf0))
- An Introduction to Reinforcement Learning, Sutton and Barto pdf (./files/SuttonBook.pdf)

In [4]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js')
```