

# Section overview

## Model-Free Control

- 1 Motivation
- 2 Reinforcement Learning 101
- 3 Lets go Markov
- 4 Markov Decision Process
- 5 Dynamic Programming
- 6 Model-Free Learning
- 7 Model-Free Control
  - MC Policy Improvement
  - On-Policy Methods
  - TD control
  - Off-Policy methods
  - Q-Learning

# Model-Free RL

- We completed the basics of Model-Free Learning: Estimate the value function of an unknown MDP
- We need to understand how to do Model-Free Control: Optimise the value function of an unknown MDP

We can use Model-Free Control in two important scenarios:

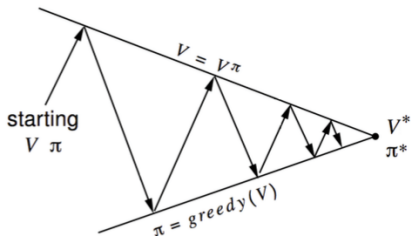
- ① MDP model is known, but is too big to use (Curse of Dimensionality), except by sampling
- ② MDP model is unknown, but experience can be sampled.

# Model-Free Control

How do we find a policy in a model-free world, where value function or action-value functions are learned by MC or TD evaluation? We follow the idea of generalised policy iteration (GPI) for MDPs. Then and here GPI maintains both an approximate policy and an approximate value function, yet converges to an optimal policy.

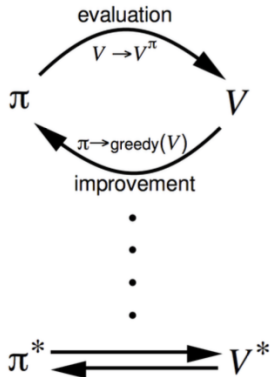
- The value function is repeatedly altered to more closely approximate the value function for the current policy.
- and the policy is repeatedly improved with respect to the current value function.

# Generalised Policy Iteration on State Value function



**Policy evaluation** Estimate  $v_\pi$   
e.g. Iterative policy evaluation

**Policy improvement** Generate  $\pi' \geq \pi$   
e.g. Greedy policy improvement



# MC Policy Improvement

- Policy improvement is done by making the policy greedy with respect to the current value function. Here we limit ourselves to an action-value function, and therefore no model (we are model-free after all) is needed to construct the greedy policy.
- For any action-value function  $Q(s, a)$ , the corresponding greedy policy is the one that, for each  $s \in \mathcal{S}$  deterministically chooses an action with maximal action-value:

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

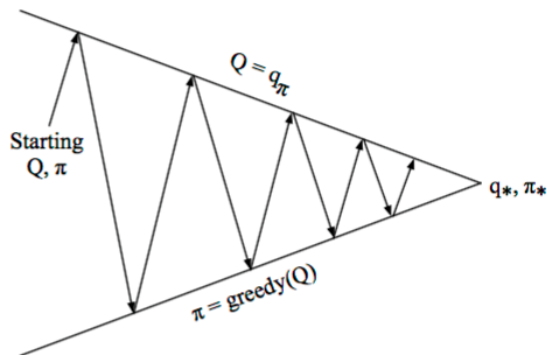
- Policy improvement then can be done by constructing each  $\pi^{k+1}$  as the greedy policy with respect to  $Q^{\pi_k}$ .

## Working with $V$ or $Q$ implies model-knowledge

- Greedy policy improvement over  $V(s)$  requires model-knowledge of MDP:  
$$\pi'(s) = \arg \max_{a \in \mathcal{A}} \mathcal{R}_{sa}^{s'} + \mathcal{P}_{ss'}^a V(s')$$
- Greedy policy improvement over  $Q(s, a)$  is model-free:  
$$\pi'(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

In the previous lectures we considered transitions from state to state and learned the values of states. Now we consider transitions from state-action pair to state-action pair, and learn the value of state-action pairs. Formally these two cases are identical: they are both Markov chains with a reward process.

# Generalised Policy Improvement on State-Action Value function



Policy evaluation Monte-Carlo policy evaluation,  $Q = q_\pi$

Policy improvement Greedy policy improvement?

## Exploring starts

How do we know that we have collected enough data to estimate the value of a state? For the moment, let us assume that we do observe an infinite number of episodes. Also, we assume we start in many different initial states. i.e. by performing **exploring starts** (start in random states).

Thus, many episodes are experienced, with the approximate action-value (Q) function approaching the true function asymptotically. Under these assumptions, Monte Carlo methods will compute each  $V^\pi$  or  $Q^\pi$  exactly, for arbitrary  $\pi$ .



# MC Policy Improvement - Theorem I

The policy improvement theorem we already encountered in DP applies also here, but we need to take care that we deal with mean return being now replaced by an average empirical return.

We have to make two assumptions to make the proof work:

- ① policy evaluation can be done with an infinite number of episodes.
- ② episodes have exploring starts to ensure we experience the full world.

Thus, Monte Carlo methods can be used to find optimal policies given only sample episodes and no other knowledge of the environment's dynamics. For practical algorithms we will need to remove both assumptions.

# MC Policy Improvement - Proof sketch [OPTIONAL] I

The proof sketch goes as follows, for  $\pi_k$  and  $\pi_{k+1}$  because, for all  $s \in \mathcal{S}$ :

$$Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg \max_a Q^{\pi_k}(s, a)) \quad (38)$$

$$= \max_a Q^{\pi_k}(s, a) \quad (39)$$

$$\geq Q^{\pi_k}(s, \pi_k(s)) \quad (40)$$

$$= V^{\pi_k}(s) \quad (41)$$

Thus, each  $\pi^{k+1}$  is better or just as good as  $\pi^k$ , in the latter case they are optimal policies. This in turn assures us that the overall process converges to the optimal policy and optimal value function. In this way Monte Carlo methods can be used to find optimal policies given only sample episodes and no other knowledge of the environment's dynamics.

## Exploring Starts vs Starting to Explore

How can we avoid the unlikely assumption of exploring starts? The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them.

- 1 **on-policy** methods, which attempt to evaluate or improve the policy that is used to make decisions
- 2 **off-policy** methods, that evaluate or improve a policy different from that used to generate the data.

# On-Policy vs Off-Policy Learning

- **On-policy** learning is "Learn on the job"
- Learn about policy  $\pi$  from experience sampled from  $\pi$
- **Off-policy** learning is "Look over someone's shoulder"
- Learn about policy  $\pi$  from experience sampled from  $\pi'$

# On-Policy Methods: Soft control

## Definition

**Soft policies** have in general  $\pi(a, s) > 0 \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ . I.e. we have a finite probability to **explore** all actions.

## $\epsilon$ -greedy policies

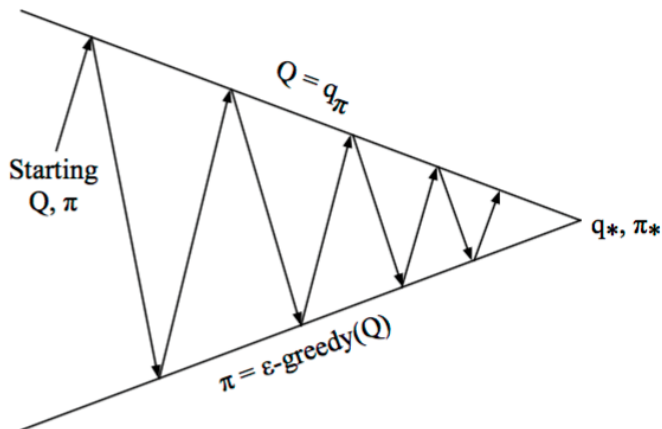
**$\epsilon$ -greedy** policies are a form of soft policy, where the greedy action  $a^*$  (as selected from being greedy on the value or action-value function and choosing the  $\arg \max$  action) has a high probability of being selected, while all other actions available in the state, have an equal share of an  $\epsilon$  probability (that allows us to explore the non-greedy/ optimal action).

### Definition

$\epsilon$ -greedy policy with  $\epsilon \in [0, 1]$ .

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|}, & \text{if } a^* = \underset{a}{\operatorname{argmax}} Q(s, a) \\ \frac{\epsilon}{|A(s)|}, & \text{if } a \neq a^* \end{cases} \quad (42)$$

# Soft policy policy improvement



Policy evaluation Monte-Carlo policy evaluation,  $Q = q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# On-policy $\epsilon$ -greedy first-visit MC control algorithm

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \leftarrow$  arbitrary

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

$\pi(a|s) \leftarrow$  an arbitrary  $\epsilon$ -soft policy

Repeat forever:

(a) Generate an episode using  $\pi$

(b) For each pair  $s, a$  appearing in the episode:

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

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

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each  $s$  in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all  $a \in \mathcal{A}(s)$ :

$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$



# Convergence of exploratory MC algorithms

## Definition

Greedy in the Limit with Infinite Exploration (**GLIE**)

- 1 All state-action pairs are explored infinitely many times,  
 $\lim_{k \rightarrow \infty} N_k(s, a) = \infty$
- 2 The policy converges on a greedy policy,  
 $\lim_{k \rightarrow \infty} \pi_k(a, s) = (a == \arg \max_{a' \in \mathcal{A}} Q_k(s, a'))$

where the infix comparison operator `==` evaluates to 1 if true and 0 else.

GLIE basically tells us when a schedule for adapting the exploration parameter is sufficient to ensure convergence.

## Example

The  $\epsilon$ -greedy operation is GLIE if  $\epsilon$  reduces to zero with  $\epsilon_k = \frac{1}{k}$ .

# MC Batch Learning to Control

MC methods can be batched, when the transitions are collected and then the estimates are computed once based on a large number of transitions.

```
1: procedure MONTECARLOBATCHOPTIMISATION( $n$ )
2:   Init
3:      $\hat{Q}(s, a) \leftarrow$  arbitrary value, for all  $s \in S, a \in A$ .
4:      $\pi \leftarrow \epsilon$ -greedy( $\hat{Q}$ )
5:   EndInit
6:   repeat (For each batch)
7:      $Returns(s, a) \leftarrow$  an empty list, for all  $s \in S$ .
8:     for  $i = 1$  to  $n$  do
9:       Get trace,  $\tau$ , using  $\pi$ .
10:      for all  $(s, a)$  appearing in  $\tau$  do
11:         $R \leftarrow$  return from first appearance of  $(s, a)$  in  $\tau$ .
12:        Append  $R$  to  $Returns(s, a)$ 
13:      for all  $s \in S, a \in A$  do
14:         $\hat{Q}(s, a) \leftarrow \text{average}(Returns(s, a))$ 
15:       $\pi \leftarrow \epsilon$ -greedy( $\hat{Q}$ )
16:   until Done
17:   return greedy( $\hat{Q}$ )
```



# MC Iterative Learning to Control

MC methods can be iterative, when after each transition the memory is changed and the transition is thrown away

```
1: procedure MONTECARLOITERATIVEOPTIMISATION( $n$ )  
2:   Init  
3:      $\hat{Q}(s, a) \leftarrow$  arbitrary value, for all  $s \in S, a \in A$ .  
4:      $\pi \leftarrow \varepsilon$ -greedy( $\hat{Q}$ )  
5:   EndInit  
6:   for  $i = 1$  to  $n$  do  
7:     Get trace,  $\tau$ , using  $\pi$ .  
8:     for all  $(s, a)$  appearing in  $\tau$  do  
9:        $R \leftarrow$  return from first appearance of  $(s, a)$  in  $\tau$ .  
10:       $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha [R - \hat{Q}(s, a)]$   
11:     $\pi \leftarrow \varepsilon$ -greedy( $\hat{Q}$ )  
12:  return greedy( $\hat{Q}$ )
```



# Summary MC control I

- 1 In designing Monte Carlo control methods we have followed the overall schema of generalised policy iteration (GPI). Rather than use a model to compute the value of each state, they simply average many returns that start in the state. Because a state's value is the expected return, this average can become a good approximation to the value.
- 2 Maintaining sufficient exploration is an issue in Monte Carlo control methods. It is not enough just to select the actions currently estimated to be best, because then no returns will be obtained for alternative actions, and it may never be learned that they are actually better.

## Summary MC control II

- ③ One approach is to ignore this problem by assuming that episodes begin with state-action pairs randomly selected to cover all possibilities. Such exploring starts can sometimes be arranged in applications with simulated episodes, but are unlikely in learning from real experience.
- ④ In on-policy methods, the agent commits to always exploring and tries to find the best policy that still explores. In off-policy methods, the agent also explores, but learns a deterministic optimal policy that may be unrelated to the policy followed.

## Summary MC control III

- 5 Off-policy Monte Carlo prediction refers to learning the value function of a target policy from data generated by a different behaviour policy. Such learning methods are all based on some form of importance sampling, that is, on weighting returns by the ratio of the probabilities of taking the observed actions under the two policies.

# MC vs TD control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to  $Q(S, A)$
  - Use  $\epsilon$ -greedy policy improvement
  - Update every time-step

## Sarsa: On-Policy TD Control I

- To derive on-policy TD control, we follow the established pattern of generalised policy iteration (GPI) we encountered for DP and MC, only this time using TD methods for the evaluation or prediction part.
- The first step is to learn an action-value function rather than a state-value function. In particular, for an on-policy method we must estimate  $Q^\pi(s, a)$  for the current behaviour policy  $\pi$  and for all states  $s$  and actions  $a$ . This can be done using essentially the same TD method described above for learning  $V^\pi$ .

$$Q(S, A) \leftarrow Q(S, A) + \alpha (r + \gamma Q(S', A') - Q(S, A)) \quad (43)$$



## Sarsa: On-Policy TD Control II

- The theorems (we touched upon) assuring us the convergence of state values under TD evaluation also apply to state-action pairs.
- SARSA reflects that the main function for updating the Q-value depends on the current state of the agent  $s_1$ , the action the agent chooses  $a_1$ , the reward  $r_2$  the agent gets for choosing this action, the state  $s_2$  that the agent will now be in after taking that action, and finally the next action  $a_2$  the agent will choose in its new state.

# SARSA - On-Policy learning TD control

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

    Repeat (for each step of episode):

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

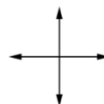
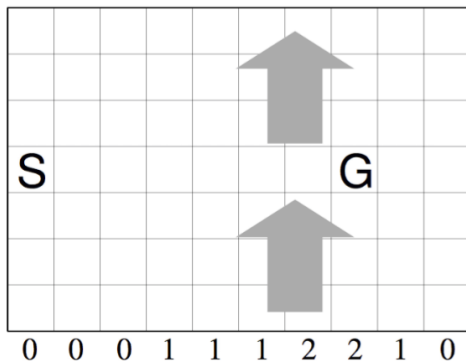
$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

    until  $S$  is terminal

You may have noticed, that taking every letter in the quintuple  $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$  yields the word SARSA.

# SARSA on Windy Grid World I

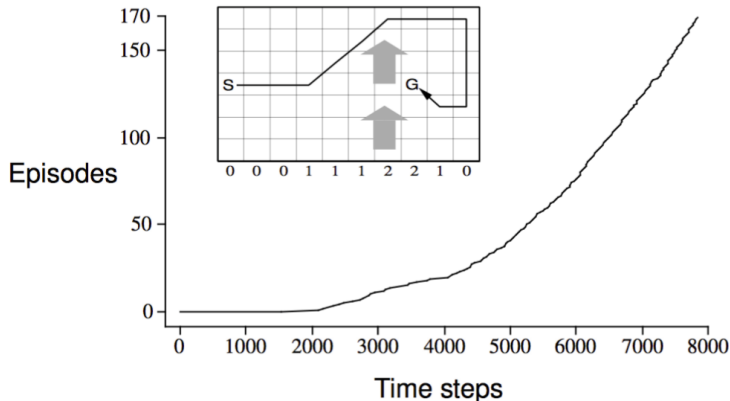


standard  
moves

Start from  $S$ ,  
reward is  $-1$  per time-step, until reaching terminal state  $G$ . Wind is blowing  
from below with strength 0,1,2 (units of displacement).

# SARSA on Windy Grid World I

This graph is a new way of looking at the learning curve, it shows how many steps the algorithm takes per episode.



At the start there is a lot of exploration (long episodes) but very quickly the episode become shorter.

# Convergence of SARSA

## Theorem

*SARSA converges to the optimal action-value function  $Q(s, a) \rightarrow Q^\infty(s, a)$ , under the following conditions*

- ① *GLIE sequence of policies  $\pi^k(a, s)$*
- ② *Robbins-Munro sequence of step-sizes  $\alpha_t$* 
  - ①  $\sum_{t=1}^{\infty} \alpha_t = \infty$
  - ②  $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

So, we have to control both the exploration  $\epsilon$  and the learning rate  $\alpha$  over time.

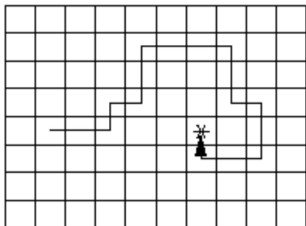
## Example

What series for  $\alpha$  satisfy these conditions?

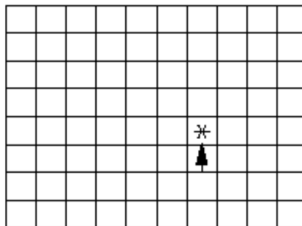
- ① reciprocals of constant positive constants? No, diverges.
- ② reciprocals of powers of 2 of the steps ? Yes, converges to 2.

## Traces, SARSA and sparse rewards I

Path taken



Action values increased  
by one-step Sarsa



We observe: almost all updates did not benefit the  $Q$  function, as the reward was sparse. Each time we run the trace we would slowly propagate backward the value through many episodes.

## How can we deal with sparse rewards?

- One approach, called SARSA-Lambda uses so called Eligibility traces to propagate sparse rewards through a trace.

## Traces, SARSA and sparse rewards II

- Other approaches, such as Hindsight Experience Replay (HER) convert unsuccessful episodes into artificially rewarded ones (we basically failed to do X, but we have achieved Y)

# Off-Policy methods

- We estimated value  $Q^\pi$  given a supply of episodes generated using a policy  $\pi$ .
- Suppose now that all we have are episodes generated from a different policy  $\pi'$ . That is, suppose we wish to estimate  $Q^\pi$  or  $V^\pi$  but all we have are episodes following another policy  $\pi'$ .
- We call  $\pi$  the **target policy** because learning its value function is the target of the learning process, and we call  $\pi'$  the **behaviour policy** because it is the policy controlling the agent and generating behaviour.
- The overall problem is called **off-policy** learning because it is learning about a policy given only experience off (not following) that policy.
- Another mnemonic way to think of is to imagine the cockpit of the agent, and to think if the target policy is switched "on" to run the agent or if it is switched "off" and the agent is left to his own behaviour (policy).



## What behaviour policies work for off-policy learning?

In order to use episodes from  $\pi'$  to estimate values for  $\pi$ , we must require that every action taken under  $\pi$  is also taken, at least occasionally, under  $\pi'$ . That is, we require that  $\pi(a, s) > 0 \Rightarrow \pi'(a, s) > 0$ . This is called the assumption of **coverage**. I.e. coverage requires that  $\pi'$  must be stochastic where it is not identical to  $\pi$ .

One of the most important breakthroughs in reinforcement learning was the development of off-policy TD control algorithm known as Q-learning (Watkins, 1989).

# Towards Q Learning

- We now consider off-policy learning of action-values  $Q(s, a)$
- The next action is chosen using behaviour policy  
 $a_{t+1} \sim \pi'(\cdot|s_t)$
- But we consider alternative successor action  $a' \sim \pi(\cdot|s_t)$
- Then, we update  $Q(s_t, a_t)$  in direction of the value of our alternative (better) action

$$\max_a Q(s_{t+1}, a) \tag{44}$$

## Q-Learning: Off-Policy TD Control

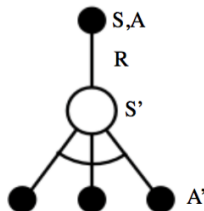
- We now we allow **both** behaviour  $\pi'$  and target policies  $\pi$  to improve.
- The target policy  $\pi$  is greedy w.r.t.  $Q(s, a)$

$$\pi(S_{t+1}) = \arg \max Q(s_{t+1}, a') \quad (45)$$

- The behaviour policy  $\pi'$  is e.g.  $\epsilon$ -greedy w.r.t.  $Q(s, a)$
- The Q-learning target then simplifies too

$$\begin{aligned} & r_{t+1} + \gamma Q(s_{t+1}, a') \\ = & r_{t+1} + \gamma Q(s_{t+1}, \arg \max Q(s_{t+1}, a')) \\ = & r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a') \end{aligned}$$

# Q-Learning: Off-Policy TD Control



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Note: We have a switch in notation in the figures  $R$  is immediate return ( $r$ )

# Q-Learning algorithm

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

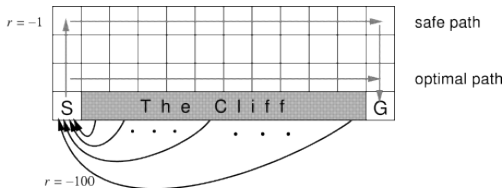
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$ ;

    until  $S$  is terminal

- We have no **explicit** policies written here. We only have a representation of the  $Q$  function.
- The target policy is implicit in the greedy term  $\max_a Q(S', a)$
- The behaviour policy is the  $\epsilon$ -greedy version of the target policy.
- Both policies are updated on each step, because we update the  $Q$  function after each step.

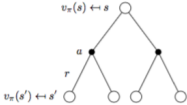

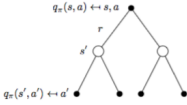
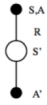
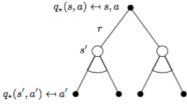
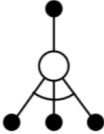
# Cliff walking example - SARSA vs Q-Learning



## Example: Asynchronous Agents I

- Instead of experience replay, train multiple agents in parallel
- Update weights asynchronously resulting in improved exploration
- Experience replay = off-policy training

# Summarising

	<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>



# Summarising

<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	$V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
Q-Value Iteration	Q-Learning
$Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$	$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

where  $x \stackrel{\alpha}{\leftarrow} y$  is defined as  $x \rightarrow x + \alpha(y - x)$ .

# Summary MC learning and control I

The Monte Carlo methods presented in this chapter learn value functions and optimal policies from experience in the form of sample episodes. This gives them at least three kinds of advantages over DP methods:

- 1 In designing Monte Carlo control methods we have followed the overall schema of generalized policy iteration (GPI). Rather than use a model to compute the value of each state, they simply average many returns that start in the state. Because a state's value is the expected return, this average can become a good approximation to the value.

## Summary MC learning and control II

- ② Maintaining sufficient exploration is an issue in Monte Carlo control methods. It is not enough just to select the actions currently estimated to be best, because then no returns will be obtained for alternative actions, and it may never be learned that they are actually better.
- ③ One approach is to ignore this problem by assuming that episodes begin with state-action pairs randomly selected to cover all possibilities. Such exploring starts can sometimes be arranged in applications with simulated episodes, but are unlikely in learning from real experience.
- ④ In on-policy methods, the agent commits to always exploring and tries to find the best policy that still explores. In off-policy methods, the agent also explores, but learns a deterministic optimal policy that may be unrelated to the policy followed.

## Summary MC learning and control III

- 5 Off-policy Monte Carlo prediction refers to learning the value function of a target policy from data generated by a different behaviour policy. Such learning methods are all based on some form of importance sampling, that is, on weighting returns by the ratio of the probabilities of taking the observed actions under the two policies.

# Summary I

## 1 Motivation

- Artificial Intelligence

## 2 Reinforcement Learning 101

- Introduction to RL
- Bayesian Decision Making
- Why probabilistic reasoning?
- Talking about Bayes

## 3 Lets go Markov

- Markov Process
- Markov Reward Process
- From state to action: Policy

## 4 Markov Decision Process

- Value function
- Policy Evaluation
- Optimal value function

# Summary II

- Bellmann Optimality
- 5 Dynamic Programming
  - Policy Improvement
  - Policy Iteration
  - Value Iteration
  - Backup strategies
- 6 Model-Free Learning
  - Monte Carlo Learning
  - TD Learning
- 7 Model-Free Control
  - MC Policy Improvement
  - On-Policy Methods
  - TD control
  - Off-Policy methods
  - Q-Learning

# The future is in Representation

- Better representations of state – the original Deep RL and Partially Observable Markov Decision Processes (POMDPs)
- Better representations of the world and world-models – model-based RL and actor-critics
- Better representations of actions...

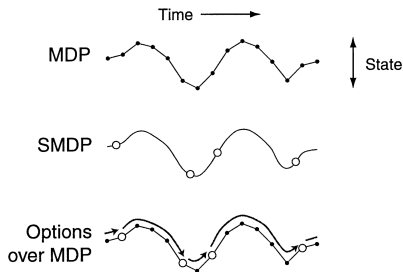
# Representation for Temporal abstracts: Actions, Options, and Action Grammars I

Consider a traveler deciding to undertake a journey to a distant city.

- To decide whether or not to go, the benefits of the trip must be weighed against costs.
- choices must be made at each leg of the trip, e.g., whether to fly or to drive, whether to take a taxi or to arrange a ride
- choices what bus to take, what taxi company to hire



# Representation for Temporal abstracts: Actions, Options, and Action Grammars II



## Where to learn from here

This was so far the first half of the course and we have covered the basics of Reinforcement Learning, this theory will allow you to understand why Deep RL in the second half really works. If you want to learn more look out for the latest work on Arxiv.org (this is not peer-reviewed, so bogus or poor work may be shown here, but it is always the most up to date). Much better for quality are top conferences:

- NeurIPS (biggest RL meeting 1000 attendees in the RL workshop)
- ICML
- ICLR (deep learning)
- RLDM (also more neuroscience/psychology)
- IROS (top robotics conference)  
<https://www.iros2020.org/ondemand/> (no fees and on demand)



# Change log

- Version 1.0 Initial version from MLNC course (1/10/2015)
- Version 2.00 updated information to match new academic year 2016/2017
- Version 3.00 updated information to match new academic year 2017/2018
- Version 4.00 updated information to match new academic year 2018/2019
- Version 5.00 updated information to match new academic year 2019/2020
- Version 5.02 updated typos on slide 102 (page count wrt 5.00)
- Version 5.03 made definitions of optimal more clear on slide 113 (page count wrt 5.00). Also updated slide 161 (first visit vs every visit example to make clear that  $\gamma = 0$ )
- Version 5.04 fixed typo ( $Q(S, \Pi(s))$  not  $Q(S), \Pi(s)$ ) on slide p. 135. updated p. 223 to make q learning algorithm more clear.
- Version 5.05 fixed missing bracket open on slide (found by Michael Brockdorff)
- Version 6.00 updated information to match new academic year 2020/2021
- Version 6.01 changed order of slides in Learning Control section, made proof of MC Policy Improvement optional, fixed
- Version 6.1 changed order of slides in Learning Control section, made proof of MC Policy Improvement optional, fixed
- Version 6.11 updated eq 30 (slide 135) halting condition (thank you Anonymous Piazza student), slide 204 changed the formatting of bullet points so as to make clear that these are two separate lists (thank you Dong Chen), slide 162 (improved clarity of Online vs Batch MC figure)
- Version 7.01 updated the course for the new term (Autumn 2021)