Introduction to Reinforcement Learning

"Reinforcement learning..., is simultaneously a problem, a class of solution methods that work well on the problem, and the field that studies this problem and its solution method" - Rich

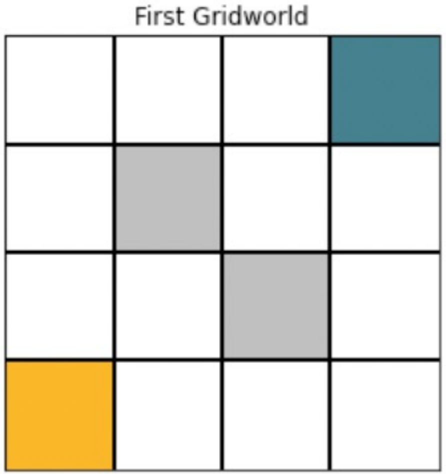
Resources:

- Reinforcement Learning An Introduction Richard S. Sutton and Andrew G. Barto
    - http://www.incompleteideas.net/book/the-book.html (free pdf)
- https://rltheory.github.io/

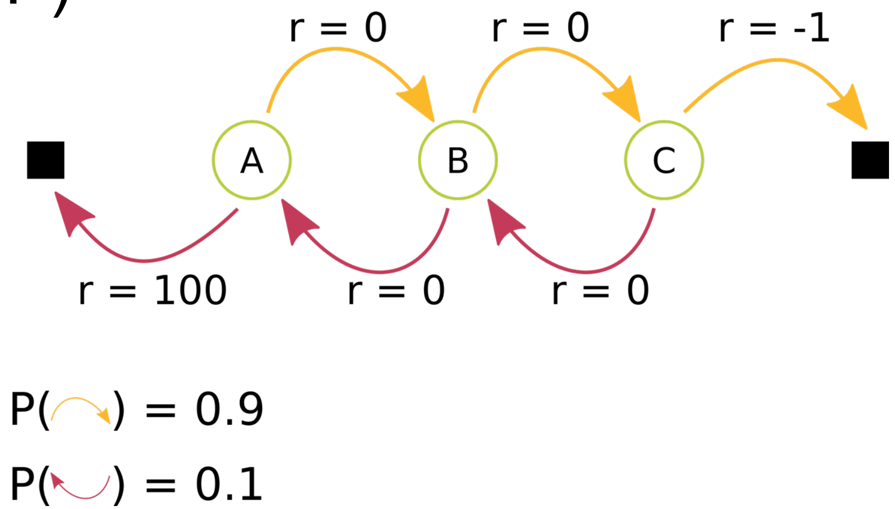
Some of these slides originally prepared by Tian Tian

Let’s start with your homework

- Simple grid world
- Need to get to the blue square while avoiding the grey squares
- In RL parlance:
  - Each square is a state
  - Actions are up, down, left, right
  - Yellow square is current state
  - Blue square is terminal state that gives positive reward
  - Grey square is terminal state that gives negative reward
  - White squares are non-terminal, give 0 reward



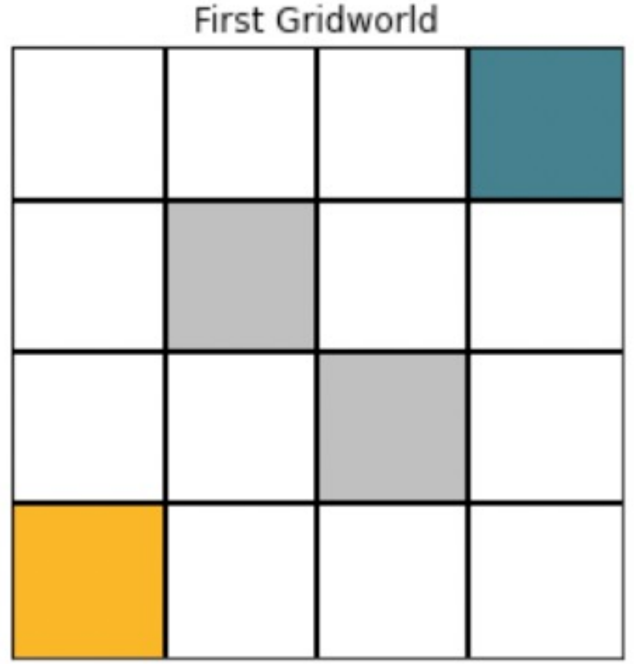
Markov Decision Process (MDP)



- 5-tuple: $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$
- Set of states: $\mathcal{S}$
- Set of actions: $\mathcal{A}$
- Discount factor: $\gamma \in [0, 1)$
- Reward: $r = (r_a(s))_{s,a}$
  - reward obtained for taking action a in state s
- Transition dynamics: $P = (P_a(s))_{s,a}$
  - set of next state distributions for each state-action pair
- Finite MDP means the state set and action set are finite
- There are several pages for notation at the beginning of the Sutton & Barto book

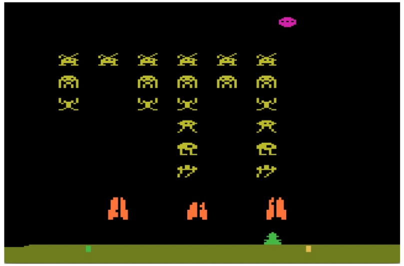
Markov Decision Process (MDP)

- 5-tuple: $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$
- Set of states: $\mathcal{S}$
- Set of actions: $\mathcal{A}$
- Discount factor: $\gamma \in [0, 1)$
- Reward: $r = (r_a(s))_{s,a}$
  - reward obtained for taking action a in state s
- Transition dynamics: $P = (P_a(s))_{s,a}$
  - set of next state distributions for each state-action pair
- Finite MDP means the state set and action set are finite



Atari games

- 5-tuple: $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$
- Set of states: $\mathcal{S}$
- Set of actions: $\mathcal{A}$
- Reward: $r = (r_a(s))_{s,a}$
  - reward obtained for taking action a in state s
- Transition dynamics: $P = (P_a(s))_{s,a}$
  - set of next state distributions for each state-action pair



Select the actions that lead to best reward

- We will learn a policy π_t
- Policy defines a mapping from state to prob. dist. over actions

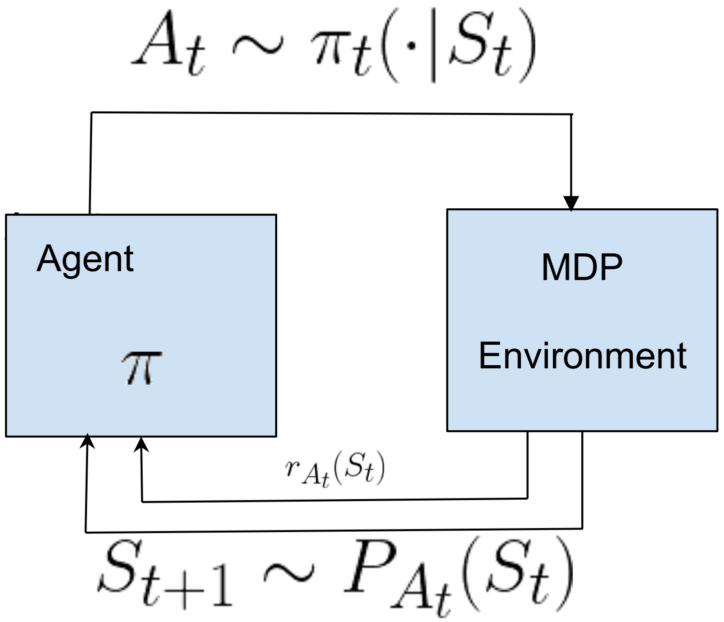
    - Policy π_t where for each t = 0,1,2,...:

$$\pi_t : \mathcal{S} \to \mathcal{M}(\mathcal{A})$$

$\mathcal{M}(\mathcal{A})$ is a set of probability distribution over A

- Policies can be complex or simple
    - Random policy
    - Look-up table
    - Neural network

Agent-environment interaction in an MDP



- Policy π_t where for each t = 0,1,2,...:

$$\pi_t : \mathcal{S} \to \mathcal{M}(\mathcal{A})$$

$\mathcal{M}(\mathcal{A})$ is a set of probability distribution over A

- The MDP and the agent interaction give rise to an infinite sequence of state-action pairs:

$$S_0, A_0, S_1, A_1, \ldots$$

Transition Dynamics: Markov Property

- A history at time step t: $H_t = (S_0, A_0, ..., S_{t-1}, A_{t-1}, S_t)$
- Transition probability:

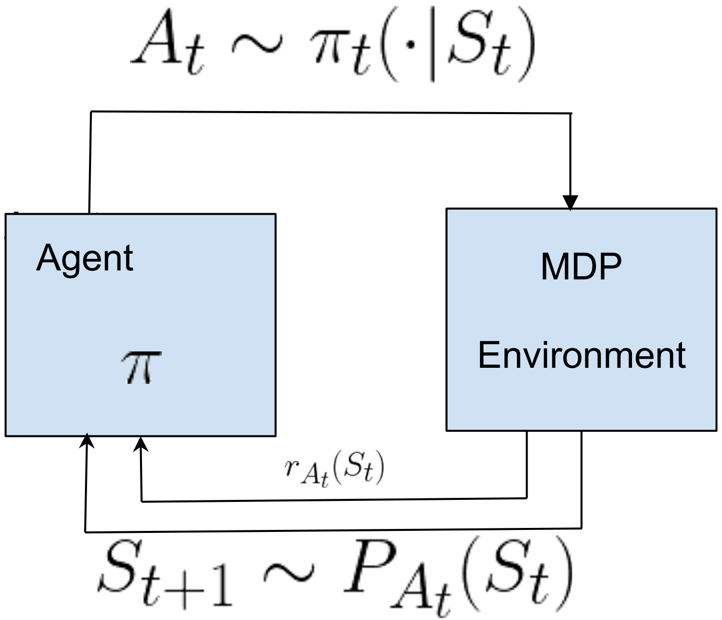
$$Pr\{S_{t+1} = s' | H_t, A_t\} = P_{A_t}(S_t, s') \text{ for all } s' \in \mathcal{S}$$

Markov property: the current state includes information about all aspects of the past agent-environment interaction

Side point: S_{t+1} may not be deterministic for a given A_t, s'

Our task

- Our task is to learn a policy π that maximizes the reward $r_{A_t}(S_t)$



But not just on this time step….

Total reward for an episode

- We would like to account for all future reward

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} \cdots$$

- But, should future rewards be weighted differently?

Return and value function and optimality

- Define *Return* over $S_0, A_0, S_1, A_1, ...$

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- $\gamma \in [0, 1)$
- What is the effect of γ^t ?

Note also

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

$$= r_{t+1} + \gamma[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}]$$

$$= r_{t+1} + \gamma[G_{t+1}]$$

Return and value function and optimality

- Value function of a policy maps states to values:

$$s \in \mathcal{S}, v^{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$$

- Optimal policy satisfies:

$$v^{\pi} = v^{*} \text{ where } v^{*} : \mathcal{S} \to \mathbb{R}$$

$$v^{*}(s) = \sup_{\pi} v^{\pi}(s), \qquad s \in \mathcal{S}$$

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

where $p(s', r|s, a) = Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$

Bellman Equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Random policy

- Choose randomly from the available options
  - Each equally likely
- Easy to implement
  - `action = np.random.randint(N)`
- Obviously, doesn't work well
- Doesn't learn

Great! Let's learn!

- What should we learn?
  - A policy π
- We need to know the expected return when
  - Taking action a
  - In state s
  - Under policy π

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$$

Great! Let's learn!

- If we had the best policy, we would know the optimal state-value function

$$v_*(s) = \max_{\pi} \; v_\pi(s)$$

Great! Let's learn!

- Optimal action-value function

$$
\begin{aligned}
q_*(s,a) &= \max_\pi q_\pi(s,a) \\
&= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]
\end{aligned}
$$

Dynamic Programming

Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

S&B pp. 75

Policies

- Simple policies
  - Greedy

$$Q_t(A_t) = \max_a Q_t(a)$$

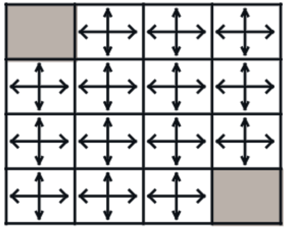
  - Epsilon greedy, ε-greedy
    - With prob ε select uniformly from all possible actions, ignoring Q

v_k for the random policy

greedy policy w.r.t. v_k

$k = 0$

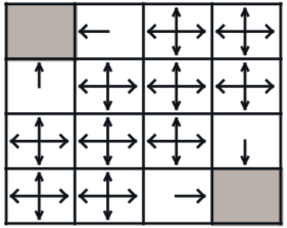
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

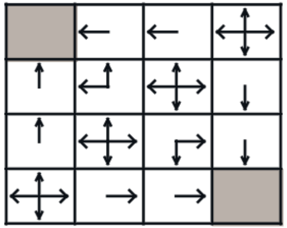
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



$k = 3$

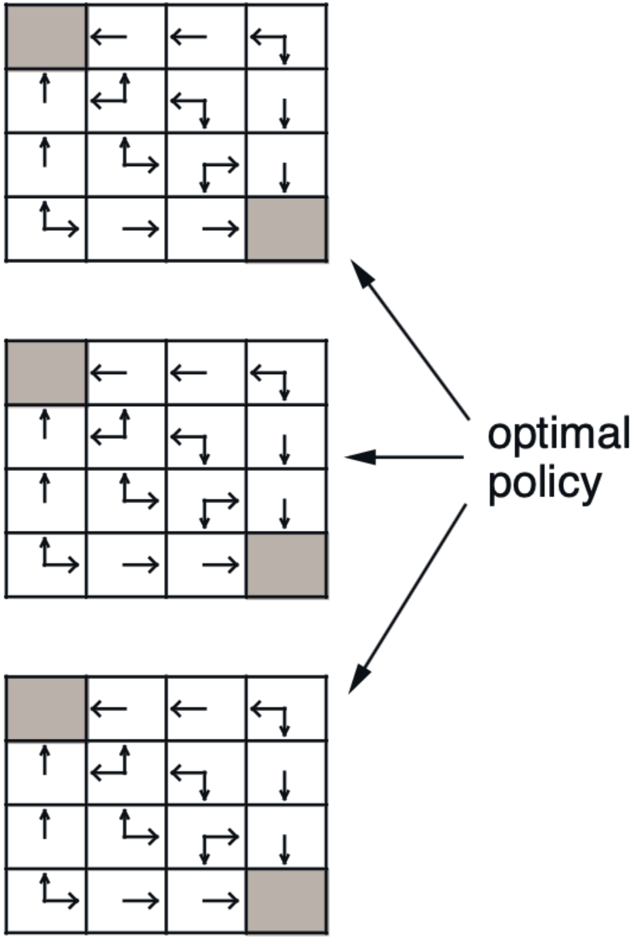
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

S&B pp. 80

Policy Iteration

- Problem: policy eval can be expensive
- May not need to go to k=\infty

$k = 3$

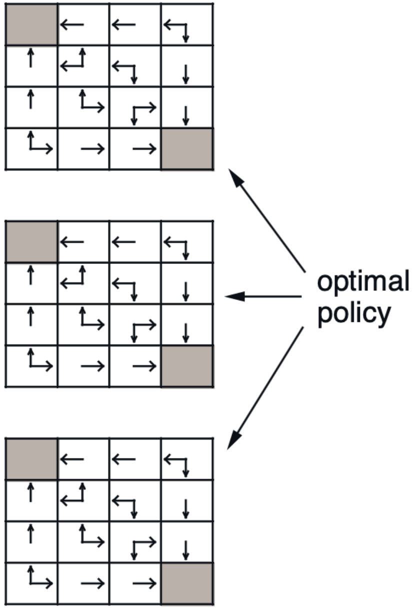
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\quad \Delta \leftarrow 0$
| $\quad$ Loop for each $s \in \mathcal{S}$:
| $\qquad v \leftarrow V(s)$
| $\qquad V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\quad \pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

States vs state-action pairs

- V tells us the value of a state
  - Greedy policy selects the next best state based on its value
- We can also learn value of state-action pairs

$$Q(S_t, A_t)$$

TD update rule for Q

- Given S_t A_t, observe R_{t+1}, S_{t+1}, A_{t+1}

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- (SARSA)

Questions?

Some cool examples

- https://openai.com/blog/emergent-tool-use/

A Q from past lectures

- EM convergence

- EM will converge, but not necessarily to the global optimum
  - Converge -> find a solution point where gradient = 0