Lecture 5: A Survey of Model-Free Reinforcement Learning

Optimal policies for Lagrangian turbulence – Dr. Robin Heinonen

Aqtivate workshop on data-driven and model-based tools for complex flows and complex fluids

June 3-7

Going model-free

- So far, we have always assumed the dynamics of the system are **known**:
    1. In Optimal Control, we have the underyling differential equation
    2. In MDP, we know the state transition and reward probabilities $\Pr(s', r|a, s)$
    3. In POMDP, we know the obs likelihood $\Pr(o|s, a)$
- This allowed us to plan an optimal strategy a priori
- In the real world, **this knowledge is a luxury**. Dynamics are frequently complicated and rarely known precisely if at all
- In this final lecture: suppose we have MDP with $\Pr(s', r|a, s)$ unknown. How to develop strategies in systems where we don't have a model?
- Idea: learn from **experience** and **system exploration** how to estimate V_π or Q_π and/or improve policy
- This is a rich subject and I cannot cover everything

A first attempt: Monte Carlo methods

- Simplest model-free method
- Follow a given π in repeated trials ("episodes") and record $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{N-1}, a_{N-1}, r_N$
- Suppose we visit s at time t in some episode. Record $R_s = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{N-t-1} r_N$
- After M visits, estimate $V_\pi(s) \approx \frac{1}{M}(R_{s,1} + \cdots + R_{s,M})$
- This can be used to rank policies, but not good systematic way to obtain π^*

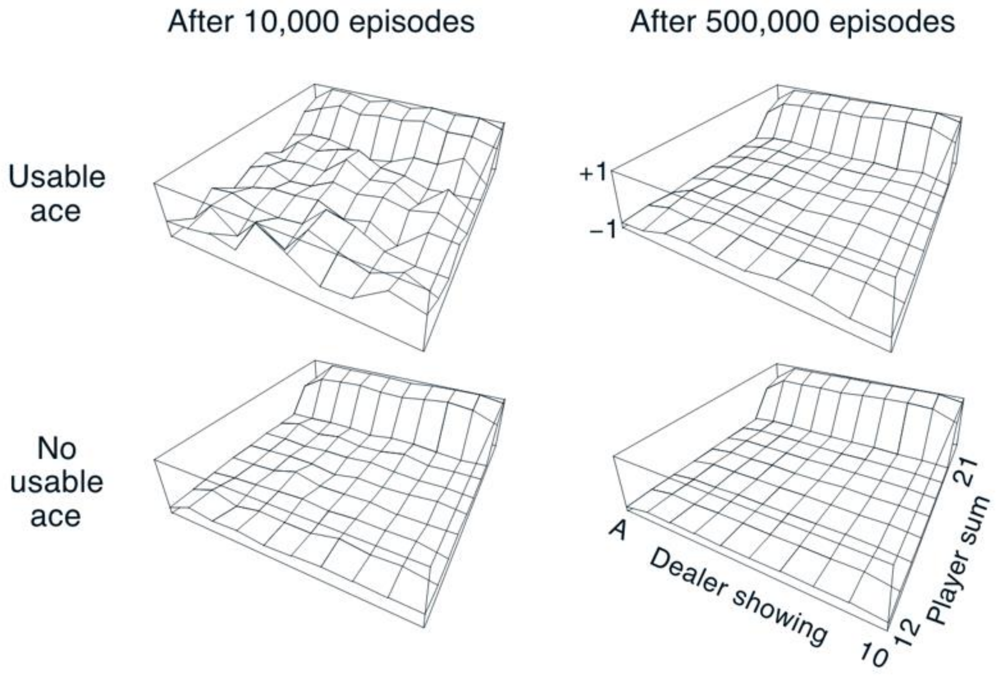


Figure 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation. ■

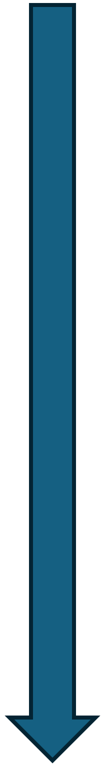
On-policy Monte Carlo

- A better approach: try to estimate $Q^*(s, a)$. Then we know optimal policy $\pi^*(s) = \arg\max_a Q^*(s, a)$.
- Why doesn't this work for V^*?
- This leads to simple algorithm:
  1. Generate an episode (or batch of episodes) using policy π
  2. Make MC estimate of $Q_\pi(s, a)$
  3. Update policy $\pi \leftarrow \arg\max_a Q_\pi(s, a)$
  4. Repeat
- Example of **on-policy** learning: exploration using same policy that we want to converge to π^*

How to explore?

- MC has good theoretical guarantees **provided you are able to explore the set of state-action pairs effectively**
- Potential issue: how to explore with a policy that is supposed to be near-optimal?
- **ε-greedy exploration**: with probability $0 < \varepsilon < 1$, take **random action**. ε typically decays over episodes
- Alternative: **off-policy** control. Policy used to update $Q^*(s, a)$ is **not** the same as policy used to explore. Independent **behavior** policy and **target** policy

ε-greedy method



SARSA

- MC can be extremely slow, since learning is **episodic**
- *Temporal difference learning*: class of methods to learn **action-by-action.** Often converges much faster
- SARSA: on-policy TD. Named because it uses tuples $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ to update Q:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)}_{\text{By Bellman, averages to zero when } Q = Q^*}]$$

- Actions are chosen by

being ε-greedy wrt Q

- $\alpha \in (0,1)$ is hyperparameter

called “learning rate.” Must be tuned

- Idea: slowly nudge Q towards Bellman optimality

Q-learning

- **Off-policy** TD learning
- Uses slightly different update rule
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$
- Both Q-learning and SARSA guaranteed to converge to Q^* as long as all state-action pairs continue to be visited
- Main difference: SARSA uses a_{t+1} chosen ε-greedily to update Q. Q-learning updates Q using *greedy* action (not necessarily the one taken)

SARSA vs. Q-learning

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

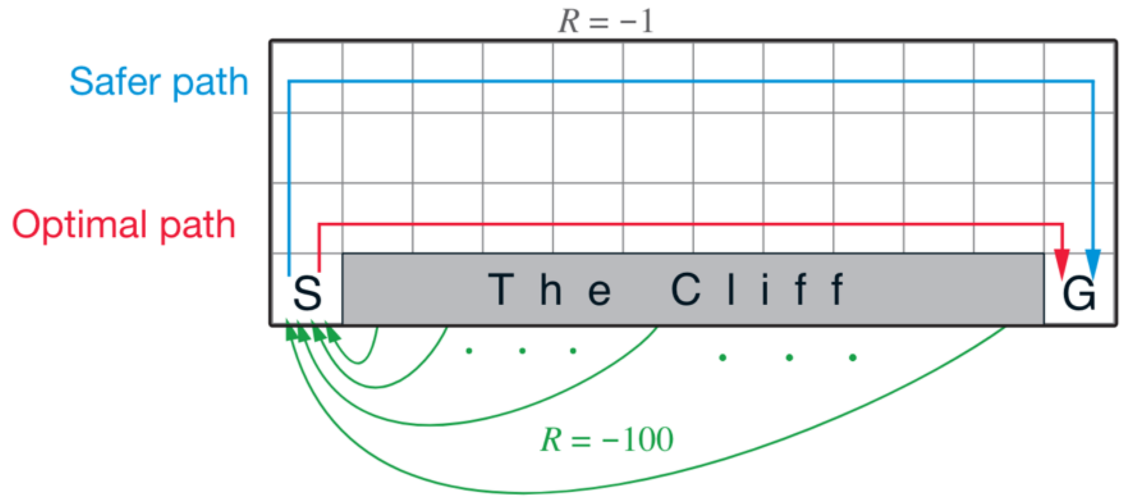
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

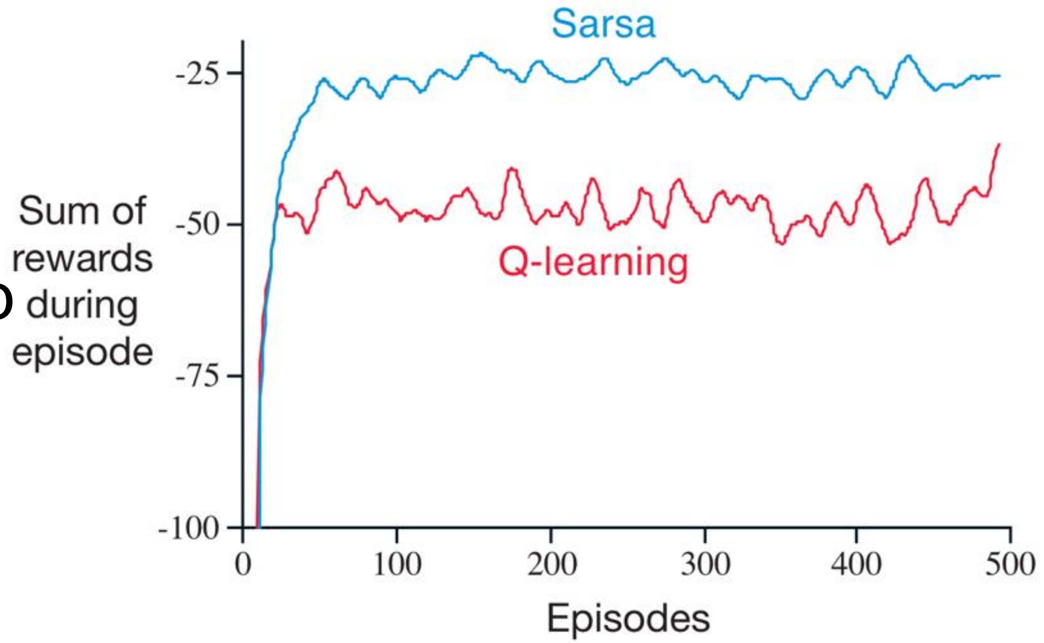
True or false? Q-learning is the same as SARSA when $\varepsilon = 0$ (greedy action selection)

False! Q-learning will not necessarily take the same action that it used in update

Example: cliff-walking problem

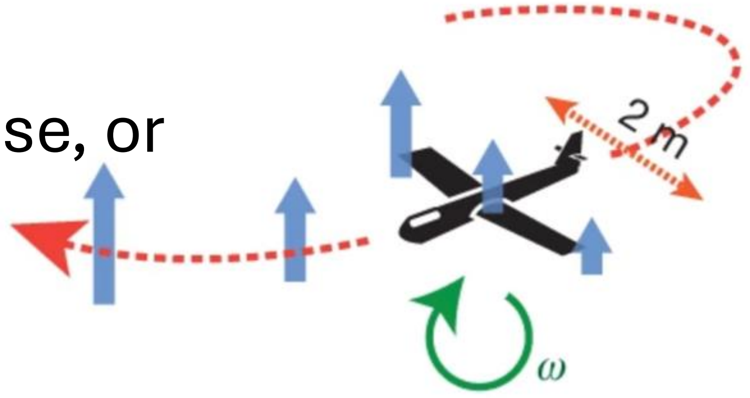
- Want to get from S to G
- If enter "cliff" region: $r = -100$, sent back to S
- Otherwise $r = -1$
- When $\varepsilon > 0$ SARSA learns to take safer but suboptimal path
- Q-learning takes optimal path but occasionally falls off the cliff due to randomness

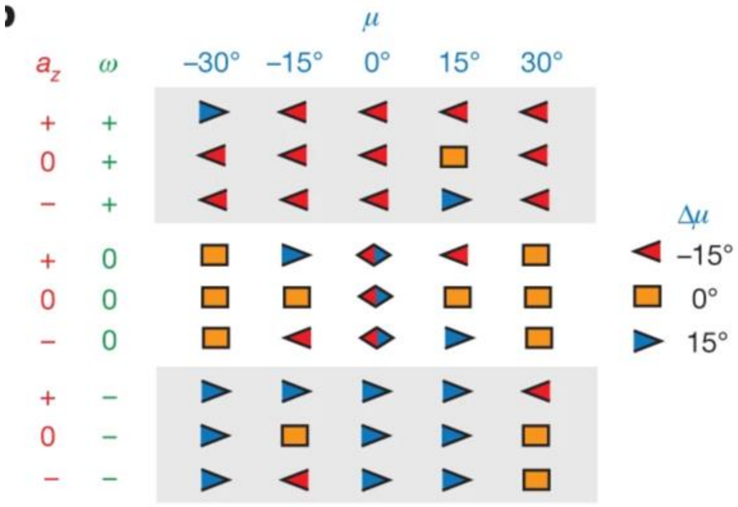




Example: learning to soar in turbulence

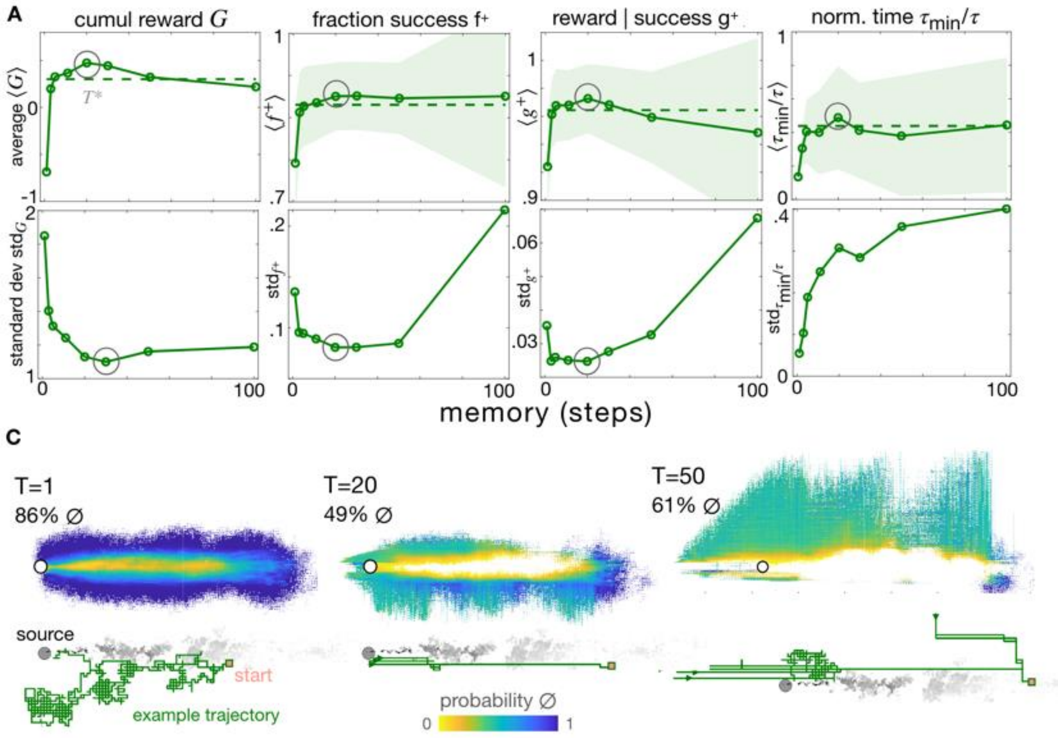
- Birds of prey use turbulent thermal plumes to glide upwards without flapping (“soaring”)
- Reddy *et al.* (*PNAS* 2016, *Nature* 2018) used Q-learning to learn strategy to move upwards in thermal plume, both in simulation *and* in the field
- States: discretized upward wind acc. a_z and spatial gradient of wind across wings ω (torque), bank angle μ
- Reward: a_z
- Actions: increase, decrease, or maintain μ





Example: olfactory search without a map

- Rando *et al*. (arXiv 2024) took alternative approach to olfactory search in turbulent flow
- No spatial map $b(\mathbf{x})$. Instead crafted features based on the last N measurements of c (after filtering out $c < c_{thr}$)
- Features: mean filtered c and fraction of time $c > c_{thr}$
- Used Q-learning to learn search strategy
- When agent is lost (filtered $c = 0$ for all t in memory), used Q-learning again. New state number of timesteps since last detection
- Agent found the source ~95% of the time



Deep Q-learning

- So far $Q(s,a)$ has always been a big matrix, $\pi^*(s)$ just a lookup table
- Alternate approach (Mnih *et al*. 2013) Q is instead a deep neural network
- May include memory using recurrent network, e.g. LSTM
- Instead of TD algorithm, use stochastic gradient descent on the loss

$$L = E\left[\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)^2\right]$$

(minimize Bellman error)

- Approximating Q, V and/or π by NN is common theme in modern RL

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Mnih et al used DQL to achieve then state-of-the-art performance on several Atari games

Reward shaping

- Frequently, rewards are **sparse**. E.g. in search problems, typically only receive reward when arrive at target
- Sparse rewards can make learning very slow
- One technique to accelerate learning: **give small extra "fake" reward** $F(s, s')$. "Reward shaping"
- In general this breaks guarantees of convergence to optimality
- But if $F(s, s') = \gamma\phi(s') - \phi(s)$ for some $\phi: S \to \mathbb{R}$, **can prove that the shaping preserves** $\boldsymbol{\pi}^*$ (exercise)
- "**Potential-based** reward shaping" (Ng *et al.* 1999)
- Example: for olfactory search, a non-potential-based shaping might be reward for detection. $\phi(\mathbf{x}) = -c\|\mathbf{x}\|_1$ ($\boldsymbol{x} = \mathbf{x}_a - \mathbf{x}_s$) generates good potential-based shaping, i.e. reward moving closer to source

Policy gradient methods

- A different way to do of model-free RL
- **Parametrize the policy**. Common choice is "softmax"

$$\pi(a|s, \boldsymbol{\theta}) = \frac{\exp\big(\beta h(s, a, \boldsymbol{\theta})\big)}{\sum_{a'} \exp(\beta h(s, a', \boldsymbol{\theta}))}$$

with $h(s, a, \boldsymbol{\theta})$ a "preference function" (frequently a NN)

- Advantages: don't need ε-greedy, also can model situations where optimal policy is stochastic (e.g. rock-paper-scissors, poker)
- **Idea of policy gradient**: iterate $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\, \widehat{\nabla_{\boldsymbol{\theta}} J}$

where J is a performance index (e.g. $V_\pi(s_0)$) and $\widehat{\nabla_{\boldsymbol{\theta}} J}$ is **stochastic estimate of the gradient**

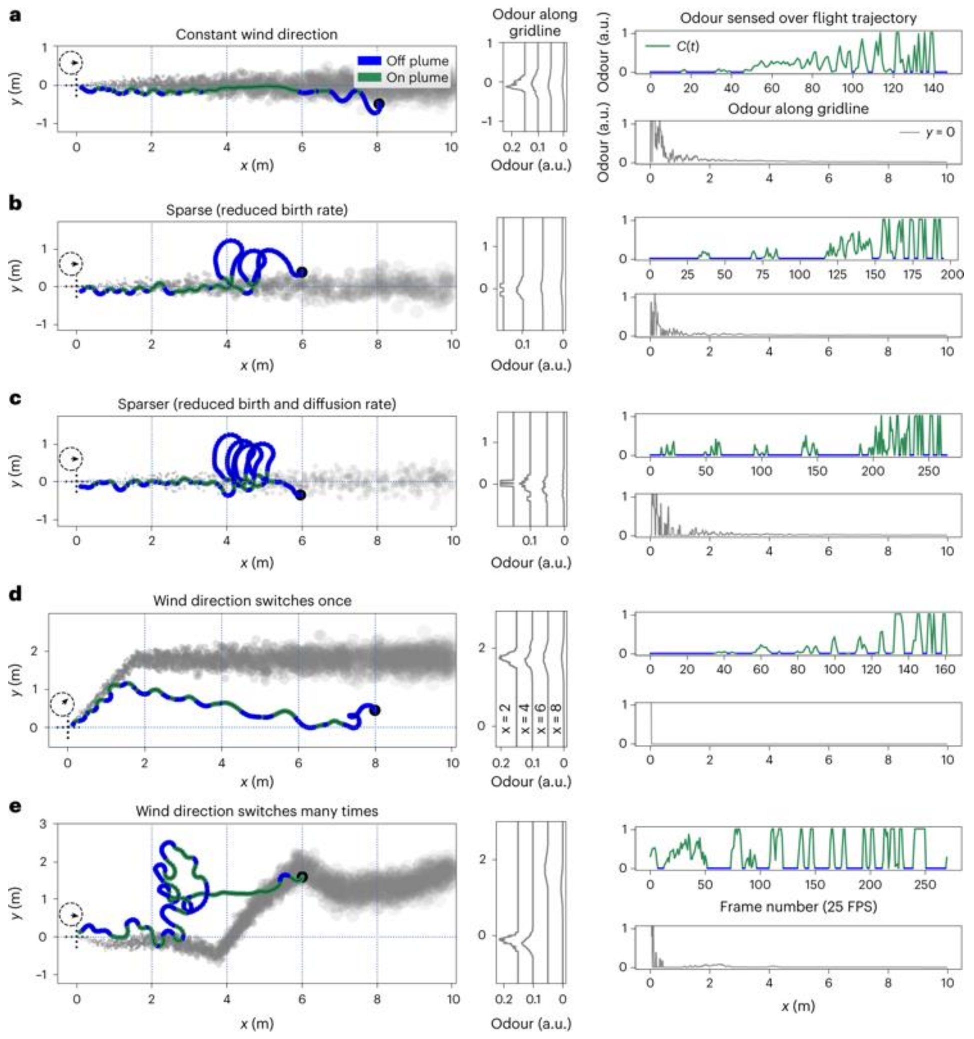
- Theorem: $\nabla_{\boldsymbol{\theta}} J \propto \sum_s p_\pi(s) \sum_a Q_\pi(s, a) \nabla_{\boldsymbol{\theta}}\, \pi(a|s, \boldsymbol{\theta})$
- In particular, **don't need to compute derivatives wrt states**!

REINFORCE

- Simplest policy gradient method. Williams (1992)
- Borrows ideas from Monte Carlo: compute episodic returns $R_t = r_{t+1} + \gamma r_{t+2} + \cdots$ starting from each visited state s_t
- Using policy gradient theorem, can show that $\nabla_{\boldsymbol{\theta}} J \propto E_\pi[R_t \nabla_{\boldsymbol{\theta}} \log \pi(a_t | s_t, \boldsymbol{\theta})]$ where $J = V_\pi(s_0)$
- Leads to following algorithm. For each t of each episode:
  1. Compute return $R_t = r_{t+1} + \gamma r_{t+2} + \cdots$ starting from state s_t
  2. Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t R_t \nabla_{\boldsymbol{\theta}} \log \pi(a_t | s_t, \boldsymbol{\theta})$

Example: model-free olfactory search

- Singh *et al*. (*Nature Mach. Intell*. 2023) used "proximal policy optimization" (a state-of-the-art policy gradient method) for model-free olfactory search
- Policy represented by an RNN
- Not a real flow, rather a simple diffusive plume model
- Plume was allowed to bend in space
- States: local wind vel. and concentration
- Reward shaped using potential $\phi(\mathbf{x}) = -\,||\mathbf{x}||$, also a non-potential-based penalty for leaving plume

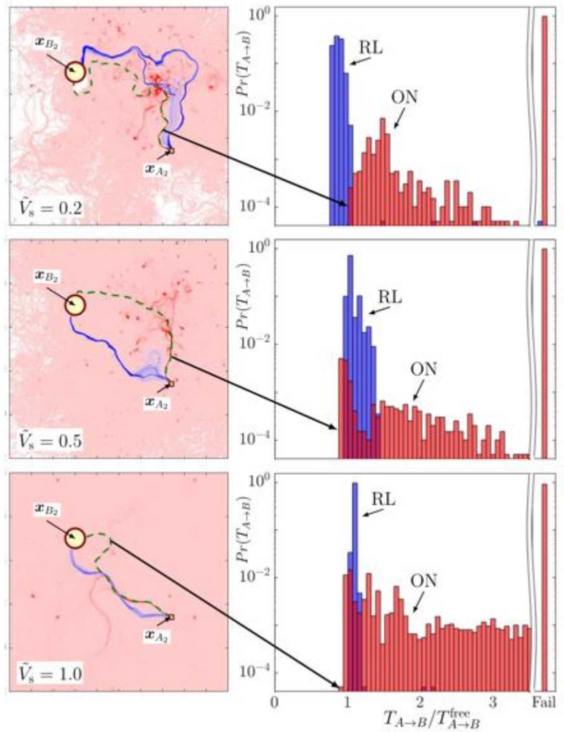


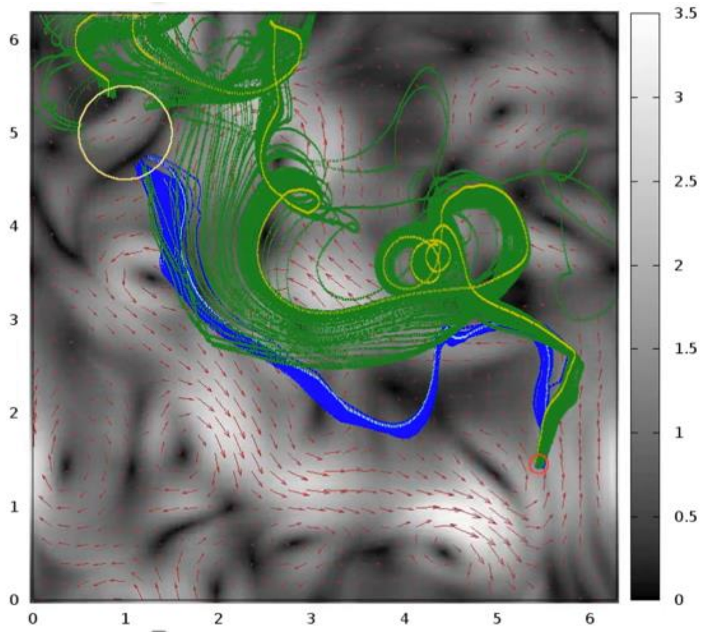
Actor-critic methods

- One of the most successful classes of RL methods. Combines policy gradient with value-function learning
- In addition to $\pi(a|s, \boldsymbol{\theta})$, also parametrize $V(s, \boldsymbol{w})$ (e.g. NN)
- π is the "actor," V is the "critic" which evaluates actions chosen by π
- Simplest version: one-step AC. For each episode:
  1. Take action $a \sim \pi(a|s, \boldsymbol{\theta})$, observe s', r
  2. Estimate Bellman error $\delta = r + \gamma V(s', \boldsymbol{w}) - V(s, \boldsymbol{w})$
  3. Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_w \delta \nabla_{\boldsymbol{w}} V(s, \boldsymbol{w})$ (gradient descent of δ^2)
  4. Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta})$ (t current timestep)
  5. Repeat until episode done

Example: Zermelo problem

- Biferale $et\ al.$ (2019) used applied AC to Zermelo problem in chaotic flow
- States: gridworld discretization of arena
- Actions: $\theta_j = \frac{j\pi}{4}, j = 0,1,\ldots,7$
- Policy parametrization: softmax with action preference $h(s_i, a_j, \boldsymbol{q}) = q_{ij}$
- Value function parametrization: $V(s_i, \boldsymbol{w}) = w_i$
- Potential-based reward shaping
- Able to solve both time-independent *and* time-dependent cases, without stability issues!





Conclusion

- Powerful tools exist for decision making when the dynamics are unknown
- Modern methods marry model-free RL with NNs for function approximation
- Challenge: need to choose algorithm, hyperparameters, set of states, reward + shaping. Best choices not always obvious, often more art than science
- Also: training may take a very long time and is not always very stable