CS 181 Reinforcement Learning (2 of 2)

A Model-Based (brief!)

Optimism (1) Sample (2)

Act (M steps) $\xrightarrow{\text{Data}}$ Learn $\longrightarrow$ Plan

Key challenge: explore vs. exploit

(1) "Optimism under uncertainty"

$N(s,a)$ # times visited (s,a)

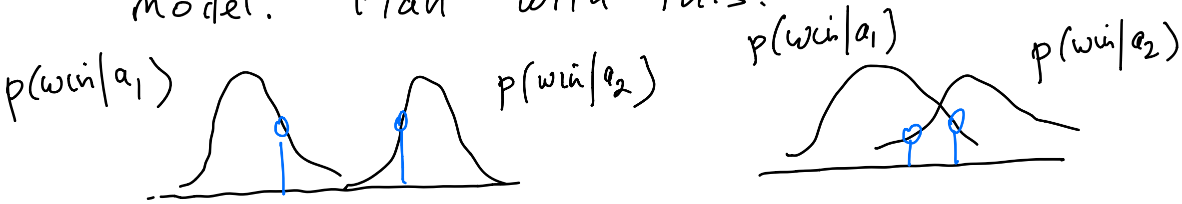
$\boxed{\text{Learning}}$ If $N(s,a)$ is small, assume next state has high reward.

$\longrightarrow R_{max}$

Plan with the "optimistic" model. Either do well, or learn new things.

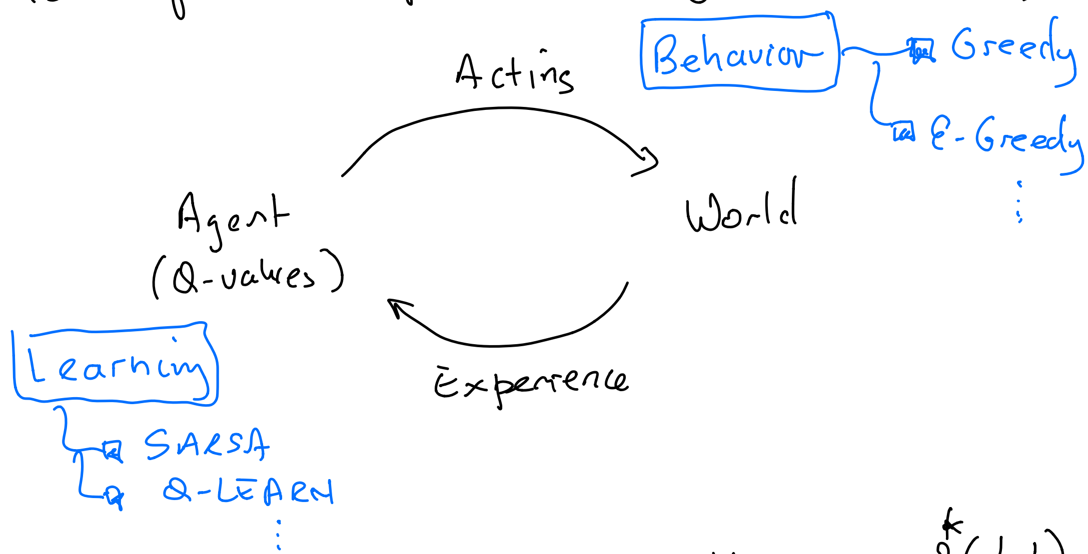
(2) "Posterior sampling" (Thompson sampling)

Maintain a posterior (Dirichlet) on transition $p(\cdot \mid s,a)$. Sample to get a model. Plan with this.



play a_2 sometimes play a_1

$\boxed{B}$ Model free Reinforcement Learning (Action-Value)



<u>Recall</u>

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a')$$

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

<u>SARSA</u>

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

<u>On-policy</u>. Learns Q-values that correspond with behavior.

<u>Q-LEARN</u>

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t(s,a)\left[\overbrace{r + \gamma \max_{a'} Q(s',a') - Q(s,a)}^{\text{TD error}}\right]$$

<u>Off-Policy</u> Learns Q-values that correspond to the Bellman equations (Q^*).

Require $\alpha_t(s,a) = 0$ unless (s,a) visited at t.

Theorem Q-LEARN converges to Q^* as $t \to \infty$ as long as

① $\sum_t \alpha_t(s,a) = \infty$, all s,a

② $\sum_t \alpha^2(s,a) < \infty$, all s,a

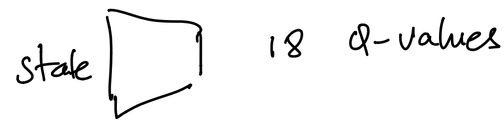
For SARSA to converge to Q^*, also need

③ Behavior is "greedy in the limit" (eventually follows Q-values)

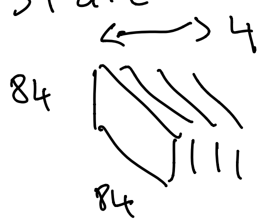
Notes

1. For $\sum_t \alpha_t = \infty$, need to visit each (s,a) infinitely often [role of ε-greedy]
2. For $\sum_t \alpha_t^2 < \infty$, need to reduce learning rate; e.g., $\alpha_t(s,a) = 1/N_t(s,a)$
3. For "Greedy in limit", typical $\varepsilon_t(s) = \frac{1}{N_t(s)}$ ∎

[C] Deep-Q Networks



State



Raw input, gray scaled, pixel map. 84 x 84 pixels. 4 most recent frames $\Big\}\ 2^{84 \times 84 \times 4}$

$\boxed{\text{Idea one}}$ Tabular representation fails! Parameterize $Q(s, a; \underline{w})$, use differentiable deep network (CNN)

$\boxed{\text{Idea two}}$ Gradient descent on TD-error

$$\underline{w} \leftarrow \underline{w} \underbrace{- \frac{1}{2}\alpha_t \nabla_{\underline{w}} \Big[r + \gamma \max_{a'} Q(s', a'; \underline{w}_{old}) - Q(s, a; \underline{w}) \Big]^2}_{+\, \alpha_t \left(r + \gamma \max_{a'} Q(s', a'; \underline{w}_{old}) - Q(s, a; \underline{w}) \right) \nabla_{\underline{w}} Q(s, a; \underline{w})}$$

"frozen network"

$\boxed{\text{Idea three}}$ Experience replay

Put (s, a, r, s') into replay buffer, and do minibatch gradient descent steps.
(Re-use experience)

D Model-free : Policy learning

Adopt a differentiable policy $\pi_\theta(a|s)$, parameters θ. Learn directly!

- ✓ Policy might be simpler than Q-function
- ✓ Handle continuous actions ✓ Introduce expert knowledge

$\boxed{\text{How to train?}}$ $\theta \leftarrow \theta + \alpha_t \underbrace{\nabla_\theta J(\theta)}_{\text{performance } (\approx V^\pi)}$

Data histories $h = (s, a, r, s', a', \dots)$

Reward $r(h) = \sum_t r_t$. $J(\theta) = \mathbb{E}_h[r(h)]$

Likelihood $\mu_\theta(h) = \prod_t \pi_\theta(a_t|s_t)\, p(s_{t+1}|s_t, a)$

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_h[r(h)] = \nabla_\theta \int_h \mu_\theta(h)\, r(h)\, dh = \int_h r(h) \nabla_\theta \mu_\theta(h)\, dh$$

Challenge $\nabla_\theta \mu_\theta(h)$ seems to depend on $p(\cdot|s,a)$

Useful identity $\nabla_\theta \mu_\theta = \mu_\theta \frac{1}{\mu_\theta} \nabla_\theta \mu_\theta = \mu_\theta \nabla_\theta \ln \mu_\theta$

$$= \int_h r(h) \mu_\theta(h) \nabla_\theta \Big[\sum_t \ln \pi_\theta(a_t|s_t) + \underbrace{\sum_t \ln p(s_{t+1}|s_t, a_t)}_{\text{doesn't depend on } \theta} \Big] dh$$

$$= \mathbb{E}_h\Big[r(h) \sum_t \nabla_\theta \ln \pi_\theta(a_t|s_t)\Big]$$

SGD update. Given history $h = (s, a, r, s', a' \ldots)$

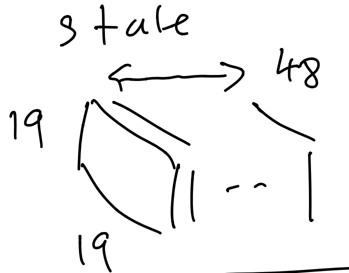
$$\theta \leftarrow \theta + \alpha\, r(h) \sum_t \nabla_\theta \ln \pi_\theta (a_t | s_t)$$

Can also use mini-batches

Notes ① this is an "on policy" method because it's updating the policy that it's using to behave in the world.

② can be combined with "Actor-Critic" methods to speed-up + stabilize learning.

$\boxed{E}$ AlphaGo



48 hand crafted, per-position features.

$\boxed{\text{Supervised learn}}$

π_{SL} π_{fast}

$\boxed{\pi_{SL}}$ Train deep net to predict expert moves

state → $19\times19+1$ (pass) softmax

13 conv. layers

log likelihood loss

3 weeks, 50 GPUs, 30M expert pos.

$\boxed{\pi_{fast}}$ Train linear soft max model w/ log likelihood loss + hand coded features

$\boxed{\text{Reinforcement L}}$

π_θ V_θ

$\boxed{\pi_\theta}$ state → $19\times19+1$

Initialize net. to π_{SL}

Reward +1 win
-1 loss 0 draw

1M games in self play vs. earlier versions

Policy gradients.

1 day, 50 GPUs, 1M games

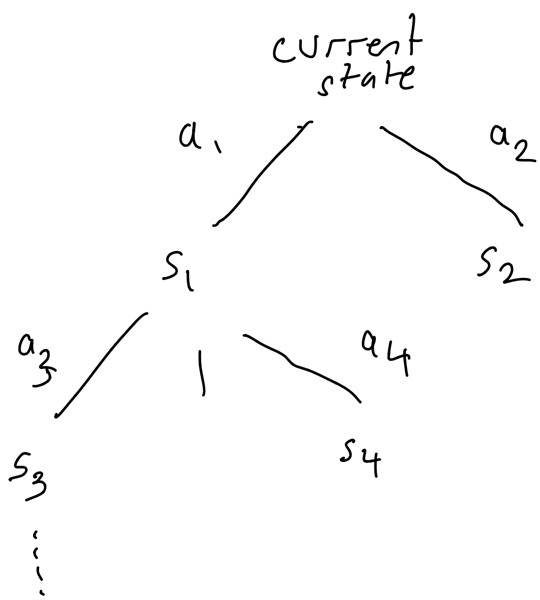
$\boxed{V_\theta}$ state → $[0,1]$ $p(\text{win})$

Supervised learning.

Target: win or loss coming from Monte Carlo eval. of π_θ. 1 week 50 GPUs 30M positions

How to play?

Guided Monte Carlo Tree Search
(48 CPUs, 8 GPUs, 40 threads !!)



Guided expansion using π_{SL}

and increasingly Q values that back-up from leaves

s_L leaf $V(s_L) = \frac{1}{2} v_\theta(s_L) + \frac{1}{2} G$

rollout π_{fast}

G win/loss

Didn't use π_θ. Found π_{SL} for guiding expansion was more useful.