Reinforcement Learning Course by David Silver

Yarylgassimov Sultan's Notes

— 2023 —

21.01.2023

L.1: Intro to Reinforcement Learning.

Books: Sutton & Barto. An introduction to reinforcement learning. 1998.

Szepesvari. Algorithms for RL. 2010.

more mathematical.

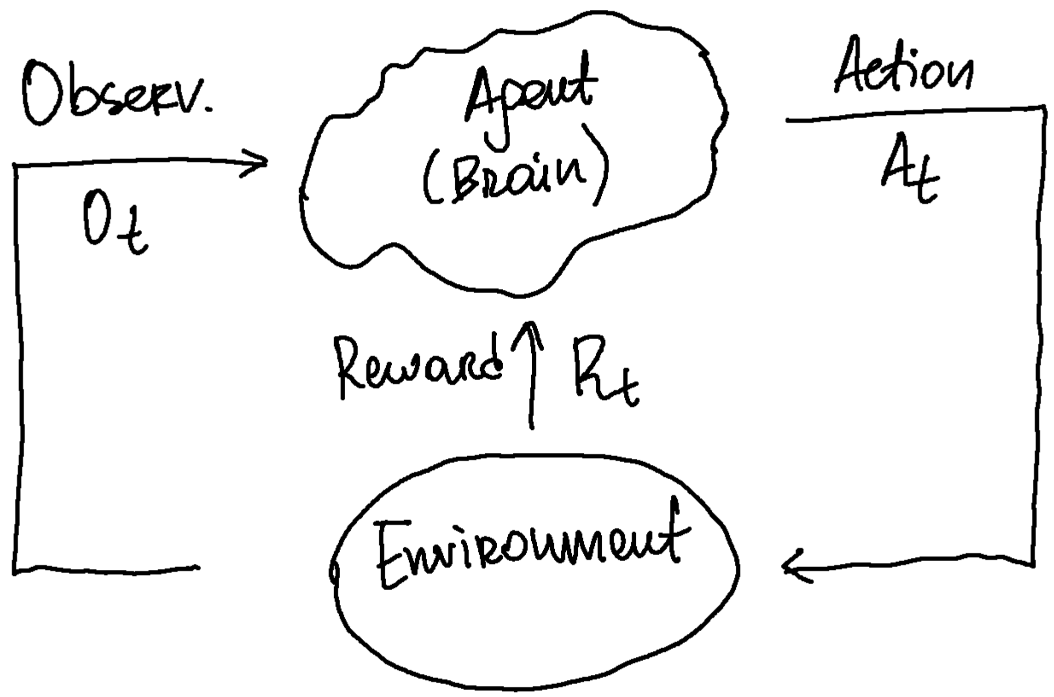
Example: • manage an investment portfolio

Reward R_t — scalar feedback signal.

Def. (Reward hypothesis) All goals can be described by the maximisation of expected cumulative reward.

Goal: select actions to max. total future reward

• You cannot use greedy algos.



The <u>history</u> is the sequence:

$$H_t = A_1, O_1, R_1, \ldots, A_t, O_t, R_t$$

<u>State</u> is the information used to determine what happens next.

$$S_t = f(H_t)$$

<u>Environment state</u> S_t^e is the environment's private representation.

not usually visible to the agent.

<u>Agent state</u> S_t^a is the agent's internal representation. $S_t^a = f(H_t)$

<u>Def.</u> A state S_t is <u>Markov</u> if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$

- "The future is independent of the past, given the present" : $H_{1:t} \longrightarrow S_t \longrightarrow H_{t+1:\infty}$
- i.e. the state is sufficient statistic of the future.
- The environment state is Markov.
- The history is Markov.

nice case.

Full observability: agent directly observes environment state. $O_t = S_t^a = S_t^e$.

Formally, this is Markov decision process (MDP).

Partial observability: agent indirectly observes environment. $S_t^a \neq S_t^e$

Formally, this is partially observable MDP (POMDP).

Agent must creates its own S_t^a, e.g.:

- Whole history
- Beliefs. $S_t^a = (\mathbb{P}[S_t^e = s^1], \dots, \mathbb{P}[S_t^e = s^n])$.
- RNN. $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$.

Inside an RL agent

- Policy: agent's behaviour. $a = \underset{\text{policy}}{\pi}(s)$
- Value function: prediction of future reward
- Model: predicts what env. will do next.

Maybe less, maybe more.

Policy may be stochastic: $\pi(a|s) = \mathbb{P}[A = a | S = s]$

$V_\pi(s) = E_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots | S_t = s]$

γ - discount reward (e.g. 0.99).

About model:

Transitions: $\mathcal{P}$ predict the next state (dynamics)

Rewards: $\mathcal{R}$ predicts the next (immediate) reward, e.g.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S' = s' \mid S = s, A = a]$$

$$\mathcal{R}^a_s = \mathbb{E}[R \mid S = s, A = a]$$

Categorizing RL agents: implicit

- Value based: No policy, value function.
- Policy based: policy, no value function.
- Actor critic: policy & value function.
- Model free: policy and/or value function, no model.
- Model based: policy and/or val. funct, model.

Exploitation – exploration.

25.01.2023

L2. Markov Decision Processes

Almost all RL problems can be formalized as MDP

Def. A state S_t is Markov $\iff$

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

State transition prob.: $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

State transition matrix:

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

each row sums to 1.

Def. A Markov Process (or Markov Chain) is a tuple $\langle S, \mathcal{P} \rangle$:

- S is a (finite) set of sets
- $\mathcal{P}$ is a prob. matrix

Markov Reward Process

Def. MRP is a tuple $\langle S, P, R, \gamma \rangle$:

- R – reward function, $R_s = \mathbb{E}[R_{t+1} | S_t = s]$
- γ – discount factor, $\gamma \in [0,1]$

* give value to vertices

Def. The return G_t (goal) is the total discounted reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Def. The value function $v(s)$ of an MRP is the expected return starting from state s:

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

Bellman Equation for MRPs:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} \, v(s')$$

$$v = \mathcal{R} + \gamma \mathcal{P} v \quad \text{(in matrices)}$$

$$\begin{pmatrix} v(1) \\ \dots \\ v(n) \end{pmatrix} = \begin{pmatrix} \mathcal{R}_1 \\ \dots \\ \mathcal{R}_n \end{pmatrix} + \gamma \begin{pmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \ddots & \dots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{pmatrix} \begin{pmatrix} v(1) \\ \dots \\ v(n) \end{pmatrix}$$

* n - number of states.

It can be solved directly:

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

$O(n^3)$ — comp. complexity.

Markov Decision Processes

Def. A MDP is a tuple $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{A}$ – a finite set of actions
- $\mathcal{P}$ is a state transition prob. matrix:

  $$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- $\mathcal{R}$ is a reward function

  $$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Def. A <u>policy</u> π is a distribution over action, given states:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

A policy defines the behaviour of an agent.
There, policy depends only on s, not time
i.e. policy are stationary: $\forall t > 0$, $A_t \sim \pi(\cdot | S_t)$

Def. A <u>state-value function</u> $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following π

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Def. The action-value function $q_\pi(s,a)$

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

expected return taking action a and following π from state s.

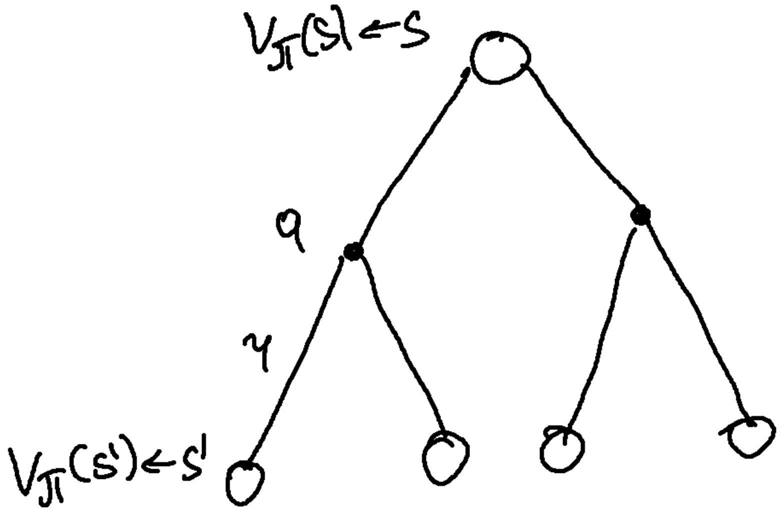
Bellman Expectation Equation:

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

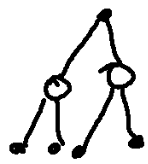
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\, q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s')$$



$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s')\right)$$



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s')\, q_\pi(s',a')$$

mean by π

In MRP :

$$V_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi V_\pi$$

$$V_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Def. The optimal state-value function $V_*(S)$ is the max. value function over all policies:

$$V_*(s) = \max_{\pi} V_\pi(s)$$

The optimal action-value function $q_*(s,a)$ is the max. action-value function over all π:

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

* it's max reward, not the best path.
↳ if it's known then we can find the best path.
↳ we should find q_* then go by increasing q_* path.

Def. $\pi \geq \pi'$ if $V_\pi(s) \geq V_{\pi'}(s), \forall s$

Theor. For any MDP

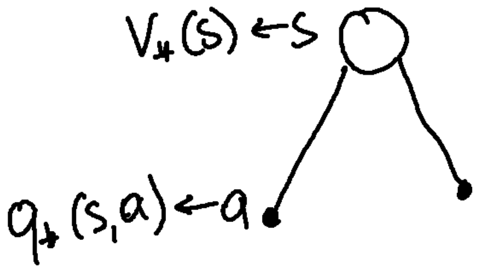
- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $V_{\pi_*}(s) = V_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_*(a,s) \\ 0, & \text{otherwise} \end{cases}$$

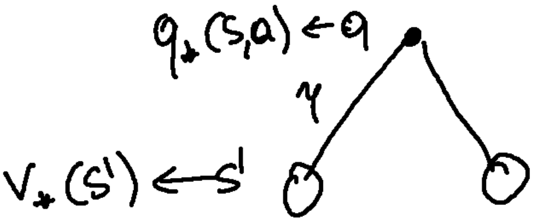
There's always optimal deterministic solution of MDP.

How do we find q_*? By looking backwards.

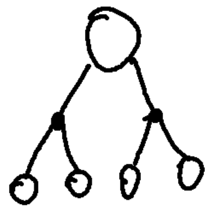
Bellman Optimality Equation



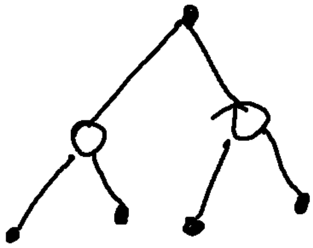
$$V_*(s) = \max_a q_*(s,a)$$



$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')$$



$$V_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')$$



$$q_*(s,a) = \mathcal{R}_s^a + \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

Bellman Optimality Equation is non-linear.
Many iterative solutions.

Extensions to MDPs

26.01.2023

L3. Planning by Dinamic Programming

Introduction

DP assumes full knowledge of the MDP.
It's used for <u>planning</u> in an MDP.

* Viterbi algorithm - what's

Policy Evaluation

Problem: evaluate policy π
Solution: iterative application of Bellman expectation backup

$$V_1 \longrightarrow V_2 \longrightarrow \dots \longrightarrow V_\pi$$

Using sychronous backups

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s') \right)$$

$$V^{k+1} = \mathcal{R}^\pi + \gamma P^\pi V^k$$

Policy Iteration

- Given a policy π
  - Evaluate π: $V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$
  - Improve π: $\pi' = \text{greedy}(V_\pi)$.

In general, need more iterations of improvement/evaluation

Always converges to π_*.

Modified policy iteration:

- Does policy evaluation need to converge to V_π?
- Introduce stopping condition:
    - ϵ-convergence
    - k iterations.

Value iteration

Theor. (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $V_\pi(s) = V_*(s) \Leftrightarrow$ for $\forall$ state s' reachable from s, π achieves the optimal value from state s', $V_\pi(s') = V_*(s')$

$$V_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V_*(s')$$

Intuition: start w/ final rewards and work backwards.

Problem: find optimal policy π

Solution: iterative application of Bellman optimality backup

$$v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$$

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

in matrices: $$v_{k+1} = \max_{a \in \mathcal{A}} \left(\mathcal{R}^a + \gamma \mathcal{P}^a v_k \right)$$

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Eval.
Control	Bellman Exp. Eq. + Greedy Policy Improvement	Policy Iteration
Control	Bellman Opt. Eq.	Value Iter.

Algos based on state-value function: $O(mn^2)$ per iter. $\rightarrow v_\pi(s), v_*(s)$

Algos based on action-value function: $O(m^2n^2)$ per iter. $\hookrightarrow q_\pi(s,a), q_*(s,a)$

Extensions to Dynamic Programming

Three simple ideas for asynchronous DP:

1) In-place dp
2) Prioritised sweeping
3) Real-time dp

1) In synchr. you save 2 versions of v.
In-place stores only one.

2) Use magnitude of Bellman error to guide state selection:

$$\left| \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s') \right) - V(s) \right|$$

Backup the state with largest remaining Bellman error

3) Only states that are relevant to agent
Use agent's XP to guide the selection of states
After each time step S_t, A_t, R_{t+1}
Backup S_t

$$v(S_t) \leftarrow \max_{a \in A} \left(R_{S_t}^a + \gamma \sum_{s' \in S} P_{S_t s'}^a v(s') \right)$$

Large DP suffers curse of dimensionality, so we'll use sampling.

Contraction mapping

↳ Math is in lecture notes.

27.01.2023

L4. Model-free prediction

Introduction

Estimate the value function of unknown MDP.
→ policy evaluation

Monte-Carlo learning

Can only be applied to episodic MDPs:
- All episodes must terminate

Goal: learn V_π from episodes of XP under π

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

Return: $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$

Value function: $V_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$

Monte-Carlo uses emperical mean return.

$V(s) \longrightarrow v_\pi(s)$ as $N(s) \to \infty$, $V(s) = S(s)/N(s)$

sum of total return where s was visited

number of times s is visited first time

First-visit-MC evaluation

Every-visit MC policy Estimation:

$N(s)$ increments every-time s encountered

$S(s) \leftarrow S(s) + G_t$ every-time s encountered

$V(s) = S(s)/N(s)$

The mean $\mu_1, \mu_2, \dots$ of sequence $x_1, x_2, \dots$ can be computed incrementally:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Incremental MC Updates

For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

In non-stationary problems, it can be useful to track running mean, i.e. forget old episodes:

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

Temporal-Difference Learning

TD learns from incomplete episodes, by bootstrapping.

TD updates a guess towards a guess.

Simple TD learning algo TD(0)

- Update $V(S_t)$ toward estimated return

$$R_{t+1} + \gamma V(S_{t+1}) \quad \text{(TD target)}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)}_{\delta_t \text{ - TD error}}$$

TD target is biased estimate of $V_\pi(S_t)$
It has lower variance than the return

MC has high variance, zero bias
TD has low variance, some bias

MC converges to solution w/ minimum MSE
TD converges to the solution of max likelihood Markov Model.

→ exploits Markov property

→ doesn't exploit Markov property

n-Step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

$G_t^{(\infty)}$ - MC

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t)\right)$$

Averaging n-Step Returns

e.g. $\frac{1}{2} G^{(2)} + \frac{1}{2} G^{(4)}$

TD(λ)

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\lambda} - V(S_t) \right)$$

if terminal state, λ^{T-t-1}

Can only be computed for complete episodes.

$\lambda = 1$ – MC

$\lambda = 0$ – TD(0)

Eligibility traces

- Frequency heuristic: assign credit to most freq. state
- Recency heuristic: — " — " — " — recent state

$$E_0(s) = 0$$

$$E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbb{1}(S_t = s)$$

Backward view TD(λ)

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(S) \leftarrow V(S) + \alpha \delta_t E_t(S)$$

When $\lambda = 0 \Rightarrow E_t(S) = \mathbb{1}(S_t = S) = 1 \Rightarrow TD(0)$

$\lambda = 1 \Rightarrow MC$

<u>Theor.</u> The sum of offline updates is identical for forward-view and backward-view TD(λ):

$$\sum_{t=1}^{T} \alpha \delta_t E_t(S) = \sum_{t=1}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbb{1}(S_t = S)$$

27.02.2023

L5. Model-free control

Introduction

Model-free prediction: evaluates policy π

find best possible policy

Note: Try model easy football and try RL

On-policy: "learn on the job"

learn about π from XP using π

Off-policy: "Look over someone's sholder"

learn about policy π from XP sampled from $\mu \neq \pi$

```latex
\begin{tikzcd}
\pi \arrow[r, bend left, "\substack{\text{evaluation}\\ V \to V^{\pi}}"] & V \arrow[l, bend left, "\substack{\pi \to \text{greedy}(V)\\ \text{improvement}}"]
\end{tikzcd}
```

Greedy Action Selection $\to$ Bandit problem $\to$ Exploration problem

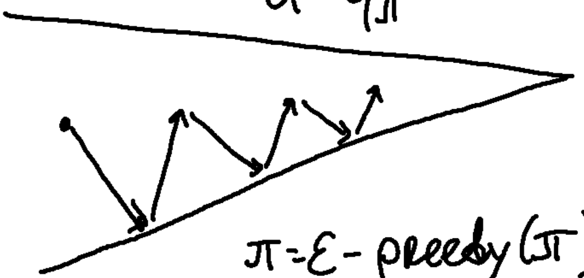
ε-greedy exploration:

$$\pi(a|s) = \begin{cases} \varepsilon/m + 1 - \varepsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \varepsilon/m & \text{otherwise} \end{cases}$$

Theorem $\forall$ ε-greedy policy π, the ε-greedy π' with respect to q_π is an improvement, $V_{\pi'}(s) \geq V_\pi(s)$.

policy improvement theorem

Idea: update V every episode:



Def. Greedy in the Limit with Infinite Exploration (GLIE):

- all state-action pairs are explored infinitely many times, $\lim_{k\to\infty} N_k(s,a) = \infty$
- the policy converges on a greedy policy,

$$\lim_{k\to\infty} \pi_k(a|s) = \mathbb{1}\left(a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q_k(s,a')\right)$$

e.g. ε reduces to zero at $\varepsilon_k = 1/k$.

GLIE MC Control

- Sample k^{th} episode using π: $\{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$
- $\forall\, S_t$ and A_t in the episode

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

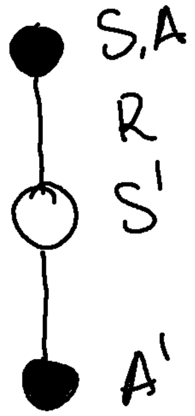
- Improve policy based on new action-value function

$$\varepsilon \leftarrow 1/k$$

$$\pi \leftarrow \varepsilon\text{-greedy}(Q)$$

Theor. GLIE MC control converges to the optimal action-value function, $Q(s,a) \longrightarrow q_*(s,a)$

Updating Action-Value Functions w/ Sarsa



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

Theor. Sarsa converges to the optimal action-value function, $Q(s,a) \longrightarrow q_*(s,a)$ under the following conditions:

- GLIE sequences of policies $\pi_t(a|s)$
- Robbins-Monro sequences of step-sizes α_t:

$$\sum_{n=1}^{\infty} \alpha_t = \infty ; \quad \sum_{n=1}^{\infty} \alpha_t^2 < \infty .$$

n-Step Sarsa

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q^n_{(t)} - Q(S_t, A_t) \right)$$

Sarsa (λ)

$$q_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{\lambda} - Q(S_t, A_t) \right)$$

forward-view (not online because wait till the end of an episode)

Backward View Sarsa (λ)

Use eligibility traces

$$E_0(s,a) = 0$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbb{1}(S_t = s, A_t = a)$$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

Off-policy learning

Evaluate $\pi(a|s)$ to compute $V_\pi(s)$ or $q_\pi(s,a)$ while following $\mu(a|s)$.

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

Important because e.g. see what people did and do better by robot.

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X) f(X) =$$
$$= \sum Q(X) \frac{P(X)}{Q(X)} f(X)$$
$$= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)} f(X)\right]$$

Importance Sampling for Off-policy MC

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\pi/\mu} - V(S_t)\right)$$

Very high variance, so don't work

Importance Sampling for Off-policy TD

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t)\right)$$

Much lower variance

Q-learning

$$A_{t+1} \sim \mu(\cdot|S_t)$$

$$A' \sim \pi(\cdot|S_t)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

$\exists$ π is greedy with respect to $Q(s,a)$

$\exists$ μ is ε-greedy w.r.t. $Q(s,a)$

Then, $R_{t+1} + \gamma Q(S_{t+1}, A') = R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$

Q-learning Control Algorithm (SARSAMAX)

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

Theor. Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

28.02.2023

L6. Value Function Approximation.

Introduction

$$\hat{v}(S, w) \approx v_\pi(S)$$

$$\hat{q}(S, a, w) \approx q_\pi(S, a)$$

w - parameters (of NN, for example)

we'll consider approximators as
- linear comb. of features
- NN

we require that methods are suitable for non-iid and non-stationary data.

Incremental Methods

Value Function Approx. using SGD

VFA Represent state by feature vector:

$$x(S) = \begin{pmatrix} x_1(S) \\ \dots \\ x_n(S) \end{pmatrix}$$

Linear VFA: $\hat{v}(S, w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$

Update $= \alpha \left(v_\pi(S) - \hat{v}(S, w) \right) x(S)$

$=$ step-size $\times$ error $\times$ feature value

Table lookup is special case of linear VFA:

$$x^{table}(S) = \begin{pmatrix} \mathbb{1}(S=S_1) \\ \cdots \\ \mathbb{1}(S=S_n) \end{pmatrix}$$

Incremental Prediction Algorithms:

- For MC:

$$\Delta w = \alpha \left(G_t - \hat{v}(S_t, w) \right) \nabla_w \hat{v}(S_t, w)$$

- For TD(0):

$$\Delta w = \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \right) \nabla_w \hat{v}(S_t, w)$$

- For TD(λ):

Forward: $\Delta w = \alpha \left(G_t^{\lambda} - \hat{v}(S_t, w) \right) \nabla_w \hat{v}(S_t, w)$

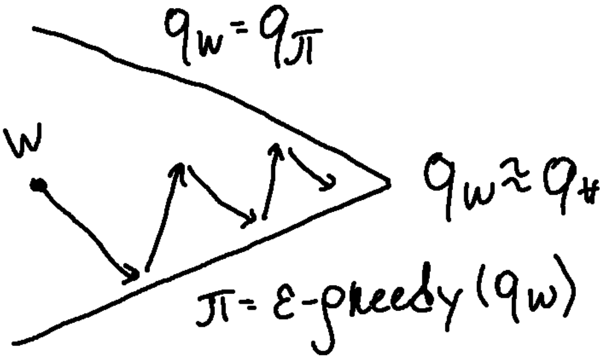
Backward:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)$$

$$E_t = \gamma \lambda E_{t-1} + x(S_t)$$

$$\Delta w = \alpha \delta_t E_t$$

Control with VFA



Approximate policy evaluation

$$\hat{q}(\cdot, \cdot, w) \approx q_\pi$$

Action-value Function Approximation

AVFA $\quad \hat{q}(S,A,w) \approx q_\pi(S,A)$

$$\Delta w = \alpha \left(q_\pi(S,A) - \hat{q}(S,A,w) \right) \nabla_w \hat{q}(S,A,w)$$

feature: $X(S,A) = \begin{pmatrix} x_1(S,A) \\ \cdots \\ x_n(S,A) \end{pmatrix}$

linear AVFA: $\hat{q}(S,A,w) = X(S,A)^T w$

Incremental Control Algorithms:

Same as prediction, but use $\hat{q}(S_t, A_t, w)$ instead of $\hat{v}(S_t, w)$

Mountain Car problem

Convergence of Prediction Algos

On/Off-Policy	Algo	Table	Linear	Non-linear
On	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Gradient TD all ✓

Convergence of Control Algos

Algo	Table	Linear	Non-linear
MC	V	(V)	X
Sarsa	V	(V)	X
Q-learning	V	X	X
Gradient Q-learning	V	V	X

(V) = chatters around

Batch Methods

Gradient descent is not sample efficient

Batch try to find the best fitting value function

experience

$$\mathcal{D} = \{\langle S_1, V_1^{\pi} \rangle, \dots, \langle S_T, V_T^{\pi} \rangle\}$$

Least squares algo:

$$LS(w) = \sum_{t=1}^{T} \left(V_t^{\pi} - \hat{V}(S_t, w) \right)^2 =$$

$$= \mathbb{E}_{\mathcal{D}}\left[\left(V^{\pi} - \hat{V}(S, w) \right)^2 \right] \longrightarrow \min$$

SGD with Experience Replay

Repeat: 1. Sample state, value from XP

$$\langle S, V^{\pi} \rangle \sim \mathcal{D}$$

2. Apply SGD update:

$$\Delta w = \alpha \left(V^{\pi} - \hat{v}(S, w) \right) \nabla_w \hat{v}(S, w)$$

Converges to least squares solution:

$$w^{\pi} = \underset{w}{\operatorname{argmin}}\, LS(w)$$

Linear LS-prediction: condition(?)

$$\alpha \sum_{t=1}^{T} x(S_t) \left(V_t^{\pi} - x(S_t)^T w \right) = 0;$$

$$w = \left(\sum_{t=1}^{T} x(S_t) x(S_t)^T \right)^{-1} \sum_{t=1}^{T} x(S_t) V_t^{\pi}$$

LSMC: $V_t^{\pi} \approx G_t$

LSTD: $V_t^{\pi} \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$

LSTD(λ): $V_t^{\pi} \approx G_t^{\lambda}$

Converges on- and off-policy both MC & TD.

LS Policy Iteration.

01.03.2023

L7. Policy Gradient Methods.

Introduction

$$\pi_\theta(s,a) = \mathbb{P}[a|s,\theta]$$

+: policy methods can be stochastic

in partially observed env. (using features) we lose Markov Property $\Longrightarrow$ no deterministic solution.

Goal: given $\pi_\theta(s,a)$ w/ θ, find best θ

To measure quality:

- start value: $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}(v_1)$
- average value: $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
- average reward per time-step:

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a$$

$d^{\pi_\theta}(s)$ — stationary distribution.

Find θ that maximizes J.

Finite Difference Policy Gradient

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \cdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix} ; \quad \frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon}$$

MC Policy Gradient

likelihood ratios : $\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$

score function: $\nabla_\theta \log \pi_\theta(s,a)$.

softmax policy

$\pi_\theta(s,a) \propto e^{\phi(s,a)^T \theta}$ ← linear comb.

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

gaussian policy

$\mu(s) = \phi(s)^T \theta$ ← can be parametrized

$a \sim N(\mu(s), \sigma^2)$

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a - \mu(s))\,\phi(s)}{\sigma^2}$$

$\phi(s)$

features

$\phi(s,a)$

Theorem: For $\forall$ diff. policy $\pi_\theta(s,a)$, for any of the policy obj. functions $J \in \{J_1, J_{avR}, \frac{1}{1-\gamma} J_{avV}\}$, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^{\pi_\theta}(s,a)\right]$$

Algorithm idea: use return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t,a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t,a_t)\, v_t$$

$\boxed{\text{50:27: } \leftarrow \text{full algo.}}$

Actor-Critic Policy Gradient

We use critic to estimate action-value function

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

critic params : w

actor params : θ

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q_w(s,a)\right]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)$$

The critic is solving policy evaluation problem.
We can do this using MC, TD, TD(λ), LS.

57:24 Algo

Theorem (Compatible Function Approximation Theor)

If the following conditions are satisfied:

① $\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$

② w minimizes MSE

$$\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]$$

Then the policy gradient is exact,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q_w(s,a)\right]$$

Reducing Variance Using a Baseline

→ Reduce variance \wo changing expectation

We subtract baseline function $B(S)$

A good baseline $B(S) = V^{\pi_\theta}(S)$

advantage function

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(S)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, A^{\pi_\theta}(s,a)\right]$$

Estimating the Advantage Function

We can use TD error as unbiased estimate of the $A^{\pi_\theta}(s,a)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, \delta^{\pi_\theta}\right]$$

$$\delta_v^{v} = r + \gamma V_v(s') - V_v(s)$$

in practice, can use approx. TD error.

Policy Gradient w/ Eligibility Traces

$$\Delta\theta = \alpha(v_t^{\lambda} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t,a_t)$$

$$\delta^{v} = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

$$e_{t+1} = \lambda e_t + \nabla_\theta \log \pi_\theta(s,a)$$

$$\Delta\theta = \alpha \delta^{v} e_t$$

Can be applied online, to incomplete sequences.

Natural Policy Gradient

$$\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

Fisher information matrix

$$G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T\right]$$

05.03.2023.

L8. Integrating Learning and Planning.

Introduction

learn model from XP.

Model-Based Reinforcement Learning

- a model $\mathcal{M}$ is a representation of MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η.
- assume the $\mathcal{S}$ and $\mathcal{A}$ are known
- $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represent state transition & reward.
- assume independence:

$$\mathbb{P}[S_{t+1}, R_{t+1} \mid S_t, A_t] = \mathbb{P}[S_{t+1} \mid S_t, A_t] \, \mathbb{P}[R_{t+1} \mid S_t, A_t]$$

Goal: Estimate $\mathcal{M}_\eta$ from XP $\{S_1, A_1, R_1, \dots, S_T\}$

$$S_1, A_1 \longrightarrow R_2, S_2$$

$$\dots$$

$$S_{T-1}, A_{T-1} \longrightarrow R_T, S_T$$

- $s, a \rightarrow r$ – regression problem
- $s, a \rightarrow s'$ – density estimation problem
- pick loss function
- find params η that minimizes loss

Table lookup model

$$\mathcal{P}_{s,s'}^{a} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}(S_t, A_t, S_{t+1} = s, a, s')$$

$$\mathcal{R}_{s}^{a} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}(S_t, A_t = s, a) R_t$$

Alternatively, record all $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$ and then sample $\langle s, a, \cdot, \cdot \rangle$

Planning with a model:
- value iteration
- policy iteration
- tree search

Sample-Based planning

Use model to generate samples

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$

$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

Apply model-free RL: MC control, Sarsa, Q-learning

→ Usually more efficient.

Integrated Architectures

We consider two sources of XP:

- Real : sampled from env.
- Simulated : sampled from model $\mathcal{M}_\eta$

Dyna-architecture :

- learn and plan value function (and/or policy) from real and simulated XP.

$\boxed{\text{Dyna-Q algo} \mid 54:11}$

The changed environment is harder/easier

→ need exploration: Dyna-Q+
revisit states that haven't been visited for a while.

Simulation-Based Search

Forward Search : build search tree with the curr. s_t
use model of MDP to look ahead
just sub-MDP from now.

Simulate from now → apply model-free RL

Monte-Carlo Tree Search (Evaluation)

Given $\mathcal{M}_\nu$

Simulate: $\{\underline{S_t}, A_t^k, R_{t+1}^k, S_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_{\nu,\pi}$

Build search-tree

Evaluate states $Q(s,a)$:

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbb{1}(S_u, A_u = s,a)\, G_u \xrightarrow{P} q_\pi(s,a)$$

Select action: $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(S_t, a)$

MC control applied to simulated XP

$Q(S,A) \longrightarrow q_*(S,A)$

maximin – the best way to act is to think that opponent is acting the best way too

TD search applies Sarsa to sub-MDP.

TD Search

For each step of simulation, update $Q(S,A)$ using Sarsa:

$$\Delta Q(S,A) = \alpha(R + \gamma Q(S',A') - Q(S,A))$$

select actions based action values $Q(S,a)$.

e.g. ε-greedy

may also use function approx. for Q.

Dyna-2

stores: long-term memory

short-term (working) memory

$\rightarrow$ updated from real XP

updated from sim. XP

19.03.2023.

L9. Exploration & Exploitation

Introduction

Approaches to exploration:

1. Random expl. (ε-greedy)
2. Optimism in the face of uncertainity (prefer to explore uncertainity)
3. Information state space (agent's information as part of its state)

State-action vs. Parameter exploration

pick diff. A each time S is visited

$\pi(A|S,\underline{u})$

We'll focus on state-action exploration.

Multi-Armed Bandits

tuple $\langle \mathcal{A}, \mathcal{R} \rangle$

$\mathcal{A}$ - actions ("arms")

$\mathcal{R}^a(r) = \mathbb{P}[R=r \mid A=a]$ - unknown

$A_t \in \mathcal{A}$ generates $R_t \sim \mathcal{R}^{A_t}$

$\sum_{\tau=1}^{t} R_\tau \longrightarrow \max$

Def. Action-value: $q(a) = \mathbb{E}[R \mid A=a]$

Def. Opt. value v_*: $v_* = q(a^*) = \max_{a \in \mathcal{A}} q(a)$

Def. Regret: $I_t = \mathbb{E}[v_* - q(A_t)]$

Def. Total regret: $L_t = \mathbb{E}\left[\sum_{j=1}^{t} v_* - q(A_j)\right]$

Count $N_t(a)$; Gap $\Delta_a = v_* - q(a)$

$$L_t = \sum_{a \in \mathcal{A}} \mathbb{E}[N_t(a)] \Delta_a$$

Good algos ensures small counts for large gaps.

Greedy can lock onto a subopt. solution.

Optimistic initialization: $Q(a) = r_{max}$

Then act greedily: $A_t = \operatorname{argmax}_{a \in \mathcal{A}} Q_t(a)$

Optimistic greedy has linear total regret.

ε-greedy explores forever
has linear total regret.

pick a decay schedule for $\varepsilon_1, \varepsilon_2, \dots$

e.g. $c > 0$

$$d = \min_{a \mid \Delta_a > 0} \Delta_a$$

$$\varepsilon_t = \min\left\{1, \frac{c|\mathcal{A}|}{d^2 t}\right\}$$

Decaying ε_t-greedy has logarithmic total regret.

Theor. (Lai & Robbins)

Asymptotic total regret is at least logarithmic in numbers of steps:

$$\lim_{t \to \infty} L_t \geq \log t \sum_{a \mid \Delta_a > 0} \frac{\Delta_a}{KL(R^a \| R^{a^*})}$$

Upper Confidence Bounds

Estimate upper conf. bound $U_t(a)$

$$A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a) + U_t(a)$$

Theor. (Hoeffding's Inequality)

Let $X_1, \dots, X_n$ be i.i.d random variables in $[0,1]$, and let $\bar{X} = \frac{1}{t} \sum_{j=1}^{t} X_j$. Then:

$$\mathbb{P}[\mathbb{E}[X] > \bar{X}_t + u] \leq e^{-2tu^2}$$

Condition on selecting a:

$$\mathbb{P}[q(a) > Q_t(a) + U_t(a)] \leq e^{-2N_t(a)U_t(a)^2}$$

Pick p : $e^{-2N_t(a)U_t(a)^2} = p$

Solve U: $U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$

Reduce p, e.g. $p = t^{-4}$

Ensures we select opt. act. $t \to \infty$: $U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}$

UCB1 algo: $A_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a) + \sqrt{\frac{2\log t}{N_t(a)}}$

Theor. $\lim_{t\to\infty} L_t \leq 8 \log t \sum_{a \mid \Delta_a > 0} \Delta_a$

Bayesian Bandits

Exploits prior knowledge.

e.g. Gaussians

Probability matching

$\pi(a) = \mathbb{P}[Q(a) = \max_{a'} Q(a') \mid R_1, \dots, R_{t-1}]$

select action a according to $\pi(a)$

Thompson sampling:

$\pi(a) = \mathbb{E}[\mathbb{1}(Q(a) = \max_{a'} Q(a')) \mid R_1, \dots, R_{t-1}]$

Sample from posterior and select max.

Value of Information

Information State Space

$\tilde{s}$ - summary of all information

Action A causes a transition to a new $\tilde{S}'$ with prob. $\tilde{P}^{A}_{\tilde{S}\tilde{S}'}$

MDP $\tilde{M} = \langle \tilde{S}, \mathcal{A}, \tilde{P}, R, \gamma \rangle$

In Bernoulli case, MDP can be solved by DP.
The solution is Gittins Index.
e.g. drug problem (work / don't work).

Contextual Bandits.

tuple $\langle \mathcal{A}, S, R \rangle$

MDPs

- for unknown or poorly est. state, replace reward function with r_{max}.
- $\tilde{S} = \langle S, I \rangle$ accumulated information

29.03.2023, "Bonus Lecture"

L10. Classic Games.

best response $\pi^i_*(\pi^{-i})$ is optimal policy against all other opponents' fixed policies

Nash equilibrium : $\pi^i = \pi^i_*(\pi^{-i})$
→ every player's policy is a best response
→ if found, game is solved.

Nash equilibrium is fixed-point of self-play RL

For general games, Nash equil. isn't unique, but we'll look at classic games where its unique.

Two-Player Zero-Sum Games

White & Black $R^1 + R^2 = 0$

rewards

Methods of finding Nash eq.:

- Game tree search
- Self-play RL

Perfect information: all visible

Imperfect information: not all visible

Minimax Search

$$V_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

$$V_*(s) = \max_{\pi_1} \min_{\pi_2} V_\pi(s)$$

minimax policy $\pi = \langle \pi_1, \pi_2 \rangle$ is a Nash eq.

instead, we use value function $v(S,w) \approx V_*(S)$

Chinook solved checkers in 2007

We can apply RL algorithms (MC, TD(0), TD(λ)) to games by making them self-play.

Policy improvement w/ afterstates

$$q_*(s,a) = V_*(\text{succ}(s,a))$$

$$A_t = \arg\max_a V_*(\text{succ}(S_t,a)) \quad \text{for white}$$

$$A_t = \arg\min_a \text{ — // — // — // —} \quad \text{for black}$$

TD performed poorly in chess, checkers because of tactical nature of games.

TD Root

update value towards sucessor search value

$$V_+(S_t, w) = \min_{s \in leaves(S_t)}\!\max\, v(s, w)$$

TD Leaf

update search value towards sucessor search value

TreeStrap

— / —— / —— / deeper — / —

Simulation-Based Search

UCT algo: MC + UCB algorithm

Smooth UCT Search

↳ Imperfect information games