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.

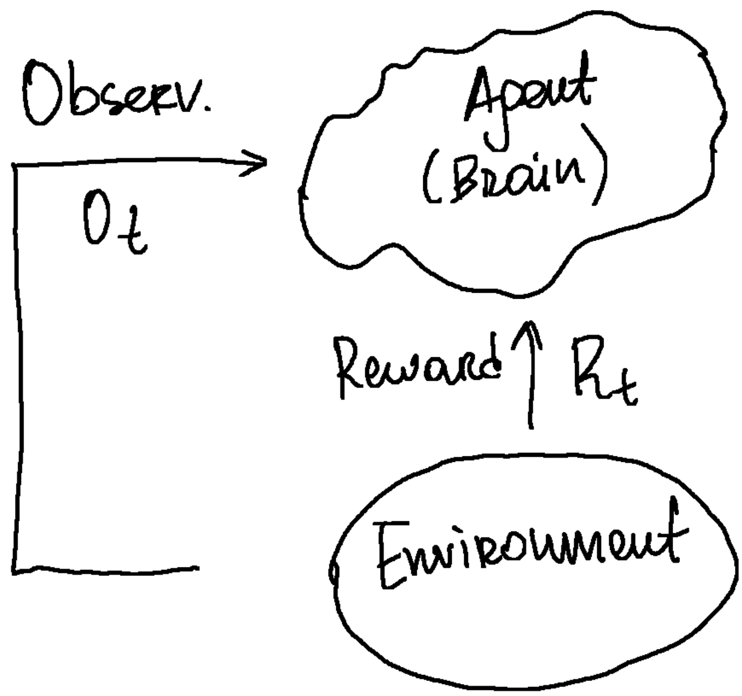
Example: • manage an investment portfolio

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 history is the sequence:

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

State is the information used to determine what happens next.

$$S_t = f(H_t)$$

Environment state S_t^e is the environment's private representation.

not usually visible to the agent.

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

Def. A state S_t is Markov if and only if

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

- "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.

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

Formally, this is Markov decision process (MDP).

<u>Partial observability</u>: 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. $S_t^a = (\mathbb{P}[S_t^e = s^1], \ldots, \mathbb{P}[S_t^e = s^n])$.
- RNN. $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$.

Inside an RL agent

policy

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

Maybe less, maybe more.

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

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

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.

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

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

Categorizing RL agents:

- Value based: No policy (implicit), 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 $\Leftrightarrow$

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

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

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. <u>MRP</u> is a tuple $\langle S, P, R, \gamma \rangle$:

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

* give value to vertices

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

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

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

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

Bellman Equation for MRPs:

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

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

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

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

$$\begin{pmatrix} v(1) \\ \cdots \\ v(n) \end{pmatrix} = \begin{pmatrix} R_1 \\ \cdots \\ R_n \end{pmatrix} + \gamma \begin{pmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \cdots \\ P_{n1} & \cdots & P_{nn} \end{pmatrix} \begin{pmatrix} v(1) \\ \cdots \\ v(n) \end{pmatrix}$$

* 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 policy π is a distribution over action, given states:

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

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 state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following π

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

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

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

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

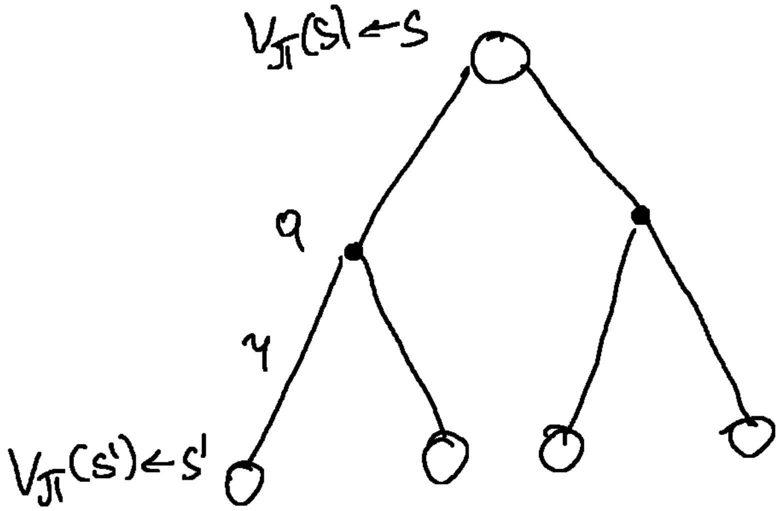
Bellman Expectation Equation:

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

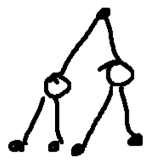
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

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

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



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



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

mean by π

In MRP :

$$V_\pi = R^\pi + \gamma P^\pi V_\pi$$

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

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

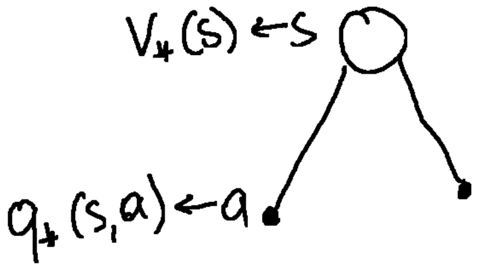
- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $V_{\pi_*}(s) = V_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$

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

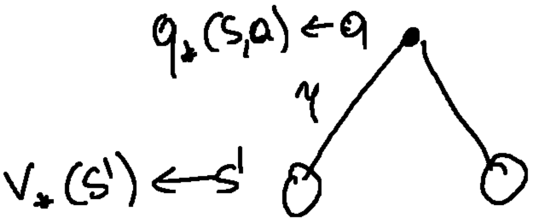
There's always optimal deterministic solution of MDP.

How do we find q_*? By looking backwards.

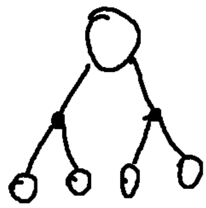
Bellman Optimality Equation



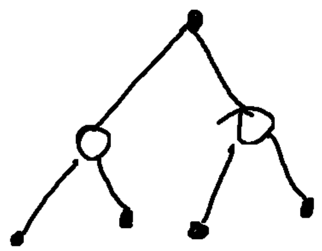
$$V_*(s) = \max_a q_*(s,a)$$



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



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



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

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

Extensions to MDPs

26.01.2023

L3. Planning by Dynamic Programming

Introduction

DP assumes full knowledge of the MDP.
It's used for planning 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, $V_\pi(s) = V_*(s) \iff$ 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 → $v_\pi(s), v_*(s)$

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

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

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

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

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 $\mu_1, \mu_2, \ldots$ of sequence $x_1, x_2, \ldots$ 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
  $R_{t+1} + \gamma V(S_{t+1})$ (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 $\lambda = 0 \Rightarrow E_t(S) = \mathbb{1}(S_t = S) = 1 \Rightarrow TD(0)$

$\lambda = 1 \Rightarrow MC$

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

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