Introduction to Reinforcement Learning

Model-free

Raphael Cousin - raphael.cousin90@gmail.com
`https://github.com/racousin/rl_introduction`

Avril 19, 2019

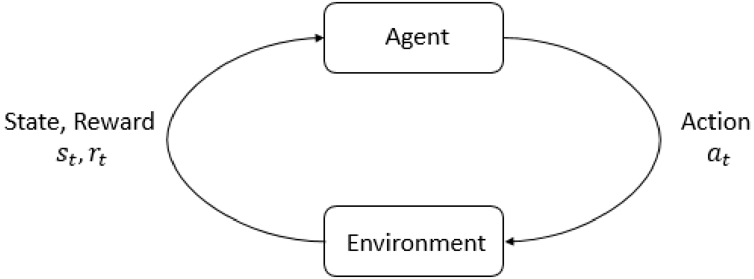
I.1)Introduction

- Supervised learning $\rightarrow$ maps an input to an output based on example input-output pairs.
- Unsupervised learning $\rightarrow$ models the probability density of inputs (using latent variables).
- Reinforcement learning (RL) $\rightarrow$ maps actions to take in an environment to maximize long term reward.

I.2)Introduction

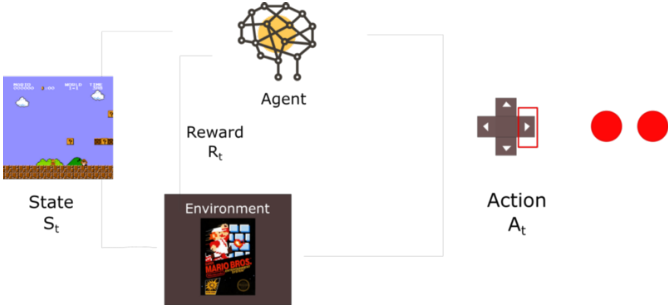
In RL problems:

- Unknown "correct" actions $\rightarrow$ no explicit supervision for a learning algorithm to try to mimic.
- Provide our algorithms only a reward function $\rightarrow$ indicates to the learning agent when it is doing well, and when it is doing poorly.



I.3)Introduction

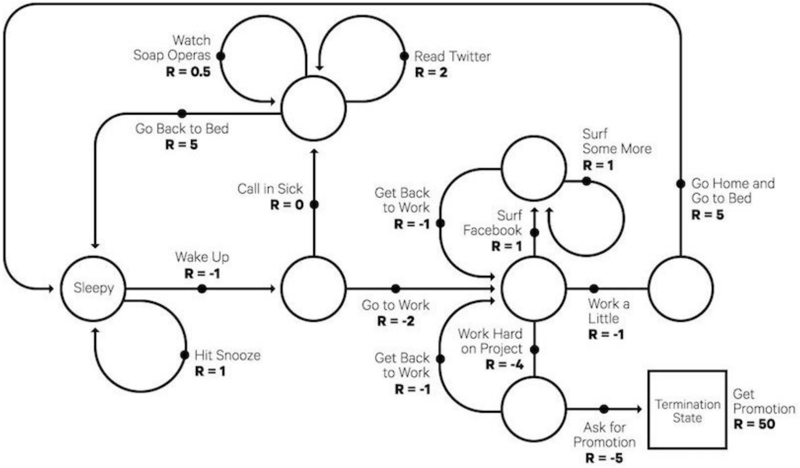
Applications: Neural Architecture Search, Autonomous vehicule, robot legged locomotion, cell-phone network routing, marketing strategy selection, factory control, efficient web-page indexing, video games.



I.4) Markov Decision Process

- S set of all valid states
- A set of all valid actions
- The transition model P
  $P_{ss'}^{a} = P(s'|s,a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} P(s', r | s, a)$
- The reward function R
  $R(s,a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r | s, a)$
- ρ_0 starting state distribution

I.5)Example: Markov Decision Process



I.6)Policy and trajectory

- Policy (agent brain): rule used to decide what actions to take
  $a_t \sim \pi(\cdot|s_t) \to \mathbb{P}(a_t|s_t)$
- Trajectories: sequence of states and actions
  $\tau = (s_0, a_0, s_1, a_1, ...)$
- States follow:
  $s_0 \sim \rho_0(\cdot)$
  $s_{t+1} \sim P(\cdot|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(\cdot|s_t, a_t)$

I.7)Reward and Return

The future reward, also known as return, is a total sum of rewards going forward.

- Finite-horizon undiscounted return:
  $G_t = \sum_{k=0}^{T} R_{t+k+1}$.
- Infinite-horizon discounted return:
  $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

With $\gamma \in]0, 1[$

I.8)The MDP Objective

- The optimization problem:
  $\pi^* = \arg\max_\pi J(\pi)$
- With the expected return:
  $J(\pi) = E_{\tau \sim \pi}[G(\tau)] = \int_\tau P(\tau|\pi)G(\tau)$
- We know the probability of a T -step trajectory is:
  $P(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$

I.9)Value Functions

- Value Function, $\rightarrow$ expected return in state s, according to π :
  $V^{\pi}(s) = E_{\tau \sim \pi}[G_t \,|s_t = s]$
- The Optimal Value Function $\rightarrow$ expected return in state s, according to optimal policy:
  $V^{*}(s) = \max_{\pi} E_{\tau \sim \pi}[G_t \,|s_t = s] = E_{\pi^*}[G_t \,|s_t = s]$

I.10)Action Value Functions

- The Action-Value Function $\rightarrow$ expected return taking action a in state s, and then according π:
  $Q^{\pi}(s,a) = E_{\tau\sim\pi}[G_t \,|s_t = s, a_t = a]$
- The Optimal Action-Value Function $\rightarrow$ expected return taking action a in state s, and then according to optimal policy:
  $Q^{*}(s,a) = \max_{\pi} E_{\tau\sim\pi}[G_t \,|s_t = s, a_t = a] = E_{\pi^*}[G_t \,|s_t = s, a_t = a]$

I.11)Bellman Equations

- **Idea :** The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]
\end{aligned}
$$

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

I.12)Bellman Equations development

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

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

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

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

I.13)Bellman Equations Optimality

Bellman equations for the optimal value functions

$$V_*(s) = \max_{a\in\mathcal{A}} Q_*(s,a)$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_*(s')$$

$$V_*(s) = \max_{a\in\mathcal{A}} \big(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V_*(s')\big)$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} \max_{a'\in\mathcal{A}} Q_*(s',a')$$

I.14)The MDP Solution

Dynamic Programming allows to resolve the MDP optimization problem ($\pi^* = \arg\max_\pi E_{\tau\sim\pi}[G(\tau)]$). It is an iterative process:

- Policy evaluation
- Policy improvment

I.15) Policy evaluation

Policy Evaluation: compute the state-value V_π for a given policy π ($\forall s$):

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[r + \gamma V_\pi(s') | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} P(s', r | s, a)(r + \gamma V_\pi(s')) \end{aligned}$$

I.16) Policy Improvement

Policy Improvement: generates a better policy $\pi' \geq \pi$ by acting greedily. Compute Q from V $(\forall a, s)$:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} P(s', r | s, a)(r + \gamma V_\pi(s')) \end{aligned}$$

Update greedily: $\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a)$ $(\forall s)$

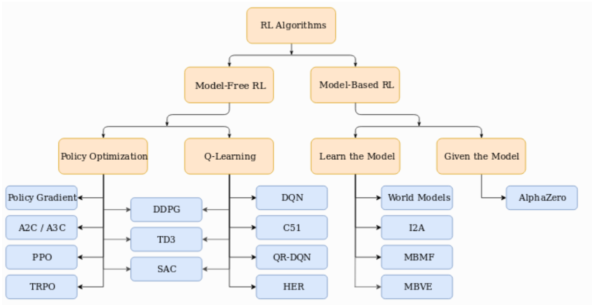
I.17) Dynamic Programming

Policy Iteration: iterative procedure to improve the policy when combining policy evaluation and improvement.

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} \ldots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluation}} V_* \quad (1)$$

This policy iteration process works and always converges to the optimality.

I.18)Bestiarry of RL Algorithms



To simplify:

- **Model free:** large data + low complexity
- **Model based:** less data + high complexity

II.1)Model-Free

The objective is the same as MDP:

- $J(\pi) = \int_\tau P(\tau|\pi)G(\tau) = E_{\tau\sim\pi}[G(\tau)]$
- The optimization problem:
  $\pi^* = \arg\max_\pi J(\pi)$

But we don't know the transition model P. The constraint is therefore to interact intelligently with the environment to obtain the information needed to solve the problem.

- **Q-learning:** learn the action value function Q
  $(\pi'(s) = \arg\max_{a\in\mathcal{A}} Q_\pi(s,a))$
- **Policy Optimization:** learn directly the policy π

II.2)Definition

- **On-policy:** Use the deterministic outcomes or samples from the target policy to train the algorithm.
- **Off-policy:** Training on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy.

II.3) Monte-Carlo

- To evaluate $V(s) = E_{\tau\sim\pi}[G_t \,|s_t = s\,]$
- From complete episodes $S_1, A_1, R_2, \ldots, S_T$ to compute $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$.
- The empirical value function is : $V(s) = \frac{\sum_{t=1}^{T} \nVdash[S_t=s] G_t}{\sum_{t=1}^{T} \nVdash[S_t=s]}$
- As, well, the empirical action-value function is :
- $Q(s, a) = \frac{\sum_{t=1}^{T} \nVdash[S_t=s, A_t=a] G_t}{\sum_{t=1}^{T} \nVdash[S_t=s, A_t=a]}$

II.4) Monte-Carlo Algorithm

Initialize Q $Q(s,a) \forall s, a$.

1. Generate an episode with the policy π (extract from Q ϵ-greedy)
2. Evaluate Q using the episode:
   $q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(\nVdash[S_t=s, A_t=a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} \nVdash[S_t=s, A_t=a]}$
3. Improve the policy greedily: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
4. Iterate

II.5) Temporal difference/bootstrapping

- Remember $V(S_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$
- So $R_{t+1} + \gamma V(S_{t+1})$ is an unbiased estimate for $V(S_t)$
- As well $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$is an unbiased estimate for $Q(S_t, A_t)$
- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target.
- $\alpha improvement$:

$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$

This observation motivates the following algorithm.

II.6) SARSA Algorithm

Initialize Q function $Q(s,a) \forall s, a$
$S_t =$ initial state, act with π to get A_t, R_{t+1}, S_{t+1}

1. Act with π to get $A_{t+1}, R_{t+2}, S_{t+2}$
2. Evaluate Q using the observation step: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
3. Improve the policy greedily: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
4. $A_t = A_{t+1}, R_{t+1} = R_{t+2}, S_{t+1} = S_{t+2}$. Iterate

II.7) Q-learning

- Remember
  $Q_{\pi^*}(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q * (S_{t+1}, a')|S_t = s, A_t = a]$
- So $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is an unbiased estimate for $Q(S_t, A_t)$
- $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is called the Q target.
- α improvement:
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$

This observation motivates the following algorithm.

II.8) Q-learning Algorithm

Initialize Q function $Q(s, a) \forall s, a$
$S_t =$ initial state

1. Act with π to get A_t, R_{t+1}, S_{t+1}
2. Evaluate Q using the observation step:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$
3. Improve the policy greedily: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$
4. Iterate

II.9)Policy Optimization

- Parametrization of policy, π_θ.
- We aim to maximize the expected return $J(\pi_\theta) = E_{\tau \sim \pi_\theta}[G(\tau)]$.
- Gradient ascent:
  $\theta_{k+1} = \theta_k + \alpha \left. \nabla_\theta J(\pi_\theta) \right|_{\theta_k}$.
- We can proof that:
  $\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)]$
  `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`

II.10)Monte Carlo Policy gradient estimation

- Set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ from the policy π_θ
- $\hat{\nabla} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t,$

II.11) Reinforce/VPG algorithm

Initialize policy π_θ

1. Generate episodes $\mathcal{D} = \{\tau_i\}_{i=1,\ldots,N}$ with the policy π_theta
2. Update policy (apply gradient ascent) $\theta \leftarrow \theta + \alpha \hat{\nabla}$
3. Iterate

References

Course:

http://incompleteideas.net/book/the-book-2nd.html

https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/

https://github.com/kengz/awesome-deep-rl

Framework:

https://spinningup.openai.com/en/latest/

https://gym.openai.com/envs/#atari

https://github.com/deepmind/bsuite

What we haven't seen

- **AlphaGo** Monte-Carlo-Tree-search `https://medium.com/applied-data-science/how-to-build-your-own-alphazero-ai-using-python-and-keras`
- **MBRL** `https://arxiv.org/abs/1907.02057`
- **Bandit** `https://arxiv.org/abs/1802.09127`
- **Intrinsic reward** `https://pathak22.github.io/noreward-rl/`