Introduction to Reinforcment Learning

Model-free

Raphael Cousin

Avril 19, 2019

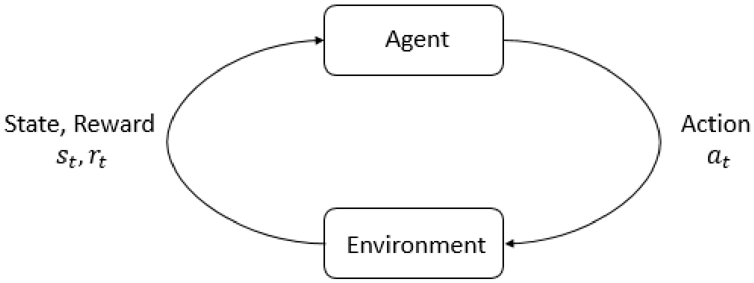
I.1)Introduction

- 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 cumulative reward.

I.2)Introduction

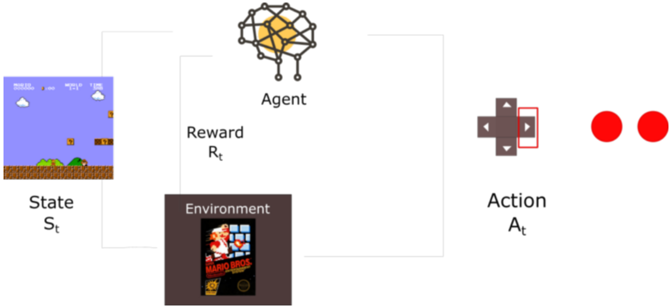
In RL problems:

- 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

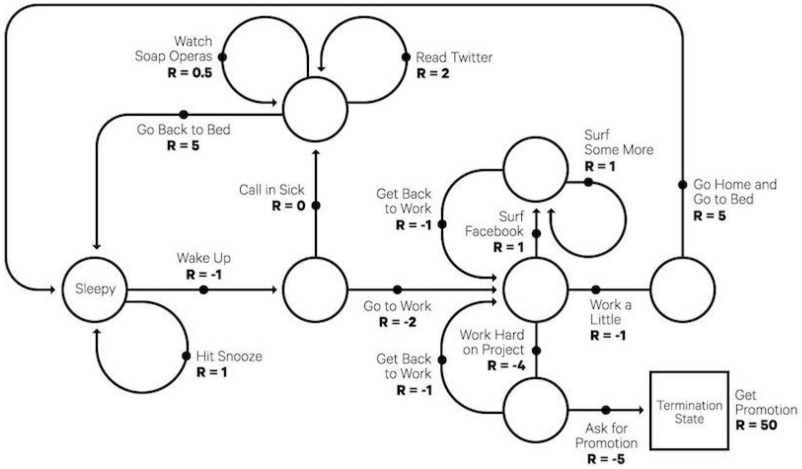
Applications: autonomous helicopter flight, robot legged locomotion, cell-phone network routing, marketing strategy selection, factory control, efficient web-page indexing, chess, video games.



I.4)Formalism: Markov Decision Process

- S set of all valid states
- A set of all valid actions
- The transition function P records the probability of transitioning from state s to s' after taking action a while obtaining reward r. $P^a_{ss'} = P(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} P(s', r|s,a)$
- The reward function R predicts the next reward triggered by one action:$R(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s,a)$
- 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
  $a_t \sim \pi(\cdot|s_t)$
- 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:
  $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

I.8)The RL Problem

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

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:
  $V^{*}(s) = \max_{\pi} E_{\tau\sim\pi}[G_t \,|s_t = s]$

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:
  $Q^{*}(s,a) = \max_{\pi} E_{\tau\sim\pi}[G_t \,|s_t = s, a_t = a]$

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

I.12)Bellman Equations

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

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

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

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

I.12)Bellman Optimality Equations

Bellman equations for the optimal value functions

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

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

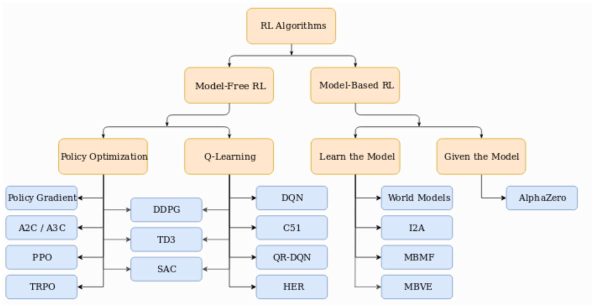
- $$V_*(s) = \max_{a \in \mathcal{A}} \big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s')\big)$$

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

I.13)Advantage Functions

- **Idea :**It is not how good an action is in an absolute sense, but only how much better it is than others on average.
- The advantage function $\rightarrow$ how much better to take action a in state s, over randomly $(\pi(\cdot|s))$, following π forever after.
- $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

I.14)Bestiarry of RL Algorithms



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

I.14)Model-base model

- **Given the model:** Dynamic Programming, Monte-Carlo-Tree-search
- **Learn the model:** Pilco

I.14)Model-free model

- **Q-learning:**
- **Policy optimization:**

I.14)Other paradigms

- **Bandit**
- **intrinsct reward**

II.1) Dynamic Programming

Policy Evaluation: compute the state-value V_π for a given policy π:
$V_{t+1}(s) =$

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

II.2) Dynamic Programming

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

$Q_\pi(s,a) =$

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

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

II.2) Dynamic Programming

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

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

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

III.1) Model-free / Monte-Carlo

Monte-Carlo Methods is an on-policy model: MCM computes the empirical return G_t, 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 for state s 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 : $Q(s,a) = \frac{\sum_{t=1}^{T} \nVdash[S_t=s,A_t=a]G_t}{\sum_{t=1}^{T} \nVdash[S_t=s,A_t=a]}$
- Improve the policy greedily with respect to the current value function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$
- Generate a new episode with the new policy π (i.e. using algorithms like ϵ-greedy helps us balance between exploitation and exploration.)
- Estimate Q using the new episode: $q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(\nVdash[S_t=s,A_t=a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} \nVdash[S_t=s,A_t=a]}$

III.1) Model-free / Temporal difference

TD learning can learn from incomplete episodes and hence we don't need to track the episode up to termination.

- Value Estimation: The key idea in TD learning is to update the value function $V(S_t)$ towards an estimated return $R_{t+1} + \gamma V(S_{t+1})$. To what extent we want to update the value function is controlled by the learning rate hyperparameter α:

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t \tag{2}$$

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{3}$$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{4}$$

- Similarly, for action-value estimation: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$.

III.1) Model-free / on policy SARSA

II.3) SARSA: On-Policy TD control

At time step t, we start from state S_t and pick action according to Q values, $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$; ϵ-greedy is commonly applied.
With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}
Then pick the next action in the same way as in step 1
$A_{t+1} = \arg\max_{a \in \mathcal{A}} Q(S_{t+1}, a)$
Update the action-value function
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
$t = t + 1$ and repeat from step 1.
In each update of SARSA, we need to choose actions for two steps by following the current policy twice (in Step 1. 3.).

III.1) Model-free / off policy Q-learning

TD learning can learn from incomplete episodes and hence we don't need to track the episode up to termination.

- At time step t, we start from state S_t and pick action according to Q values, $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$; ϵ-greedy is commonly applied.
  With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}
  Update the action-value function
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$
  $t = t + 1$ and repeat from step 1.

IV.1)Policy Optimization

- Parametrization of policy, π_θ.
- We aim to maximize the expected return $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$.
- Gradient ascent:
  $\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)|_{\theta_k}$.
- We can proof that:
  $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]$

III.4)Policy gradient estimation

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

III.5) REINFORCE On-Policy Policy Gradient

1. Initialize Θ at random
2. Generate one episode $\tau S_1, A_1, R_2, S_2, A_2, \ldots, S_T$
3. For $t = 1, 2, \ldots, T$
    1. Estimate the the return R_t since the time step t.
    2. Apply gradient ascent $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla \ln \pi(A_t | S_t, \theta)$

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

III.6)Policy gradient improvement

Don't Let the Past Distract You

- $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]$.
- Taking a step with this gradient pushes up the log-probabilities of each action in proportion to $R(\tau)$, the sum of all rewards ever obtained. But this doesn't make much sense.
- Agents should really only reinforce actions on the basis of their consequences. Rewards obtained before taking an action have no bearing on how good that action was: only rewards that come after.

III.8)Don't Let the Past Distract You

- $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})]$.
- In this form, actions are only reinforced based on rewards obtained after they are taken.
- $\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$,

III.10)Baselines in Policy

- For any function b which only depends on state,
- $E_{a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = 0.$
- Add or subtract any number of terms like this from our expression, don't change expectation.
- $\nabla_\theta J(\pi_\theta) =$
  $E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\right)].$

III.11)Baselines in Policy

- Any function b used in this way is called a baseline.
- Interesting choice is $V^\pi(s_t)$.
- Empirically, $b(s_t) = V^\pi(s_t)$ reduce variance. Conceptually, it encodes the intuition that if an agent gets what it expected, it should “feel” neutral about it.

III.12)Approximate $V^{\pi}(s_t)$

- Basic method for learning $V_\phi \rightarrow$ minimize a mean-squared-error objective:
- $\phi_k = \arg\min_\phi E_{s_t, \hat{R}_t \sim \pi_k}[\left(V_\phi(s_t) - \hat{R}_t\right)^2]$,
- π_k is the policy at epoch k. Using gradient descent, from previous ϕ_{k-1}.

III.13) Gradient estimation summary

What we have seen is that the policy gradient has the general form $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t]$,
where Φ_t could be any of

- $\Phi_t = R(\tau)$
- $\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$
- $\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$ All of these choices lead to the same expected value for the policy gradient, despite having different variances.

III.14) Other gradient estimation

- Action-Value Function.
  $\Phi_t = Q^{\pi_\theta}(s_t, a_t)$
- The Advantage Function. Recall $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, describes how much better or worse it is than other actions on average (relative to the current policy).
  $\Phi_t = A^{\pi_\theta}(s_t, a_t)$

III.16)Actor critic On-Policy Policy Gradient

If the value function is learned in addition to the policy, we would get Actor-Critic algorithm.
Critic: updates value function parameters w and depending on the algorithm it could be action-value Q(a—s;w) or state-value V(s;w).
Actor: updates policy parameters , in the direction suggested by the critic, (a—s;).

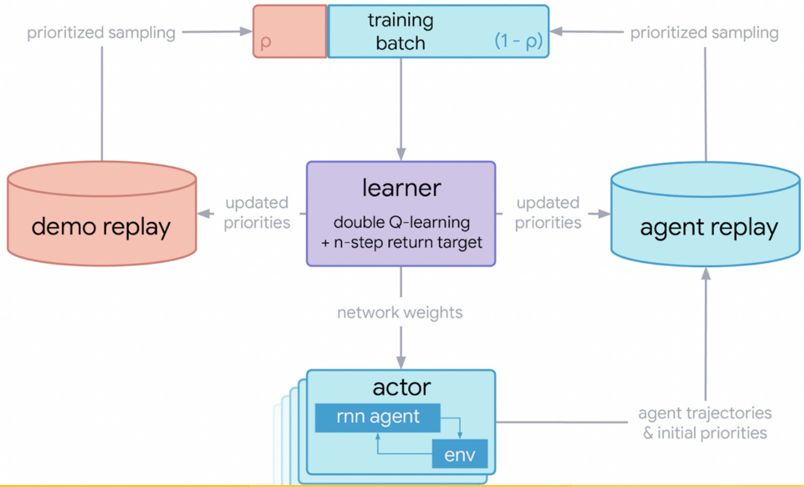
IV.2) Other approach

The evaluation of these techniques is mainly empirical. The most common evaluation was on Atari rl$_kaiser2019modelgames.Theenvironmentalmostalreadyexist, itwasjust$
For a lot of games, a simple DQN algorithm was able to be much more efficient than human in the games. See **??**
But for other it was not good at all. If we take a look as the games with the worst results. They are 2 types of games:

- the games with very sparse reward, for example platform games that implies a long suit of specific actions before obtain a reward. In this kind of games, the random exploration has no chance to obtain a reward (even after million of games), so their are no matter to learn, the agent is stuck to play randomly for ever
- the gamers with lot of negative reward, for example a game that many actions bring a loss, it the kind of game the agent learn to do nothing to avoid bad action

IV.3) rl with demonstration

An idea to help the learning process is based on human demonstrations. We create trajectories made by human and we add this trajectories to the memory of the agent. Then we let the agent overtake the master.



This approach is not always practicable. Furthermore the fact of learning has to be not only measured by the score achieved but also the complexity and energy consumption. Other methods try to improve the fact of learning:

- PILCO PILCO$_levine2018reinforcementisbasedonBayesianinference(andG$ $drivenalgorithmsrl_pathak2017curiosity, rl_burda2018largecomputear$
- Behaviour Suite for Reinforcement Learning bsuit$_osband2019behaviourisacollectionofcarefully -$ $designedexperimentsthatinvestigatecorecapabilitiesofareinforceme$
  - To collect clear, informative and scalable problems that capture key issues in the design of efficient and general learning algorithms.
  - To study agent behavior through their performance on these shared benchmarks.

References

TODO