Notes on Reinforcement Learning

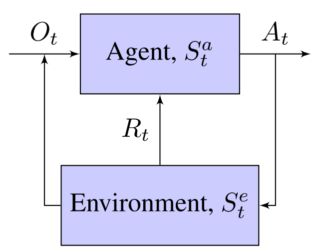
Ara Patapoutian

CONTENTS

I. Introduction

- Reinforcement learning & sequential decision making:
  - *reinforcement learning*, or RL, is the science of sequential decision making by an *agent*
    - the prototypical example of an agent is the human brain
    - *reinforce* = strengthen (borrowed from psychology), or maximize reward
  - the environment is initially unknown
  - RL is like trial and error learning
    - exploration versus exploitation
- Planning versus learning:
  - both planning and RL choose actions that maximize reward as a function of state
  - with *planning*
    - the environment model is known to the agent
    - given the model the agent optimizes a policy
  - with RL
    - no model available, i.e. environment is initially unknown
    - learn from samples or transitions
- RL related fields:
  - ML in computer science
  - reward system in neuroscience
  - classical conditioning in psychology
  - bounded rationality in economics
  - operations research in mathematics
  - optimal control in engineering
- Episodic & continuing environments:
  - in an *episodic* environment, a path terminates
  - whereas a *continuing* environment is non-terminating
- Reward:
  - a *reward hypothesis* states that all goals can be described by the maximization of expected cumulative award
  - the *reward* R_t, at time t indicates how well the agent is doing at time t
  - with *delayed reward* the environment provides a reward only at the end of an episode
    - *temporal credit assignment* is the task of crediting the step(s) that were responsible for the delayed reward
  - RL is based on the reward hypothesis
    - the goal is to select actions that maximize total future reward
- RL characteristics:
  - RL executes actions
    - RL interacts with the environment
  - the reward feedback system can be thought of as partial-supervision
  - sequential processing is similar to an RNN
- The interaction between agent and environment is shown in Fig. 1
  - at each step t, an *agent*
    - executes an *action* A_t
    - receives an *observation* O_t
    - receives a scalar *reward* R_t

Fig. 1
Agent / environment interaction



  - at each step t, the *environment*
    * receives an action A_t
    * executes an observation O_{t+1}
    * executes a scalar reward R_{t+1}
  - unlike game theory, the environment is usually assumed to be static
- History & state:
  - *history* up to time t, is defined as

$$H_t \triangleq A_1, O_1, R_1, \cdots, A_t, O_t, R_t \tag{1}$$

  - given H_t, the agent takes the next action, $H_t \to A_{t+1}$
  - H_t is not compact and is not practical to work with
  - the agent internally models (or summarizes) H_t by an *agent state*

$$S_t^a \triangleq f(H_t) \tag{2}$$

  - similarly, the *environment state* S_t^e is the environment's private representation of H_t
- Markov state:
  - the *Markov state*, or the *information state*, contains all the useful information from history
  - a state S_t, at time t, is Markov iff

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \cdots, S_t) \tag{3}$$

    * a Markov state S_t, is a sufficient statistic of the future
- the Markov process $\langle \mathcal{S}, \mathcal{P} \rangle$:
  - $\mathcal{S}$ is the set of n Markov states
  - $\mathcal{P}$ is $(n \times n)$ *state transition probability*, with

$$\mathcal{P}_{ss'} \triangleq P(S_{t+1} = s' | S_t = s) \tag{4}$$

  - a *Markov process*, or a *Markov chain*, is the tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
- Markov reward process, $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:
  - let $\mathcal{R}_s \in \mathcal{R}$ be the instantaneous reward associated with state s
  - $\mathcal{R}$ is a set of n reward scalars

$$\mathcal{R}_s \triangleq E[\, R_t \,|\, S_t = s \,] \tag{5}$$

  - $\gamma \in [0, 1]$ is the *discount*, a parameter that trades off long-term versus short term reward
  - *Markov reward process*, or MRP, is a tagged (labeled) Markov chain, also see (7)
- Markov decision process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:
  - a *Markov decision process*, or MDP, generalizes an MRP by appending a finite set of actions $\mathcal{A}$ to the system
    * the resulting state transition probability $\mathcal{P}_{ss'}^a$, depends on the action, and is a three dimensional tensor

$$\mathcal{P}_{ss'}^a \triangleq P(S_{t+1} = s' | S_t = s, A_t = a) \tag{6}$$

    * similarly, the resulting reward $\mathcal{R}_s^a$, is two dimensional

$$\mathcal{R}_s^a \triangleq E[\, R_t \,|\, S_t = s, A_t = a \,] \tag{7}$$

- Policy:
  - a *policy* is an agent's behavior function

$$S_t \xrightarrow{\text{policy}} A_t \tag{8}$$

  - a deterministic policy is a function

$$a = \pi(s) \tag{9}$$

  - a stochastic policy is a conditional probability

$$\pi(a|s) \triangleq P(A = a \,|\, S = s) \tag{10}$$

    * a stochastic policy is useful for exploration
    * a stochastic policy generalizes a deterministic policy

- Return & value function:
  - given MRP $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the *return* G_t, associated with some path (episode), is the total discounted reward after t

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{11}$$

    - G_t is a random variable since the path taken is random
  - a *value function* determines how good each state and/or action is
    - the value function is a prediction of future reward
    - the *state-value function* is defined as

$$v(s) \triangleq E[\, G_t \,|\, S_t = s \,] \tag{12}$$

    - the *action-value function* is defined as

$$q(s, a) \triangleq E[\, G_t \,|\, S_t = s, A_t = a \,] \tag{13}$$

      - given $q(s, a)$, a deterministic policy (9), can be optimized by setting

$$\pi(s) = \arg\max_{a \in \mathcal{A}} q(s, a) \tag{14}$$

  - note the progression

$$\begin{aligned} &\text{reward} \to \text{return} \to \text{value functions} \\ &R_t \to G_t \to v(s),\ q(s, a) \end{aligned} \tag{15}$$

- Model:
  - a *model* is an agent's representation of the environment
  - forming a model is optional
  - for an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, modelling implies learning $\mathcal{P}$ and $\mathcal{R}$
    - when the environment is deterministic, then given s and a, s' is unique
- Agent:
  - an agent may include a subset of these components:
    - a policy
    - a value function
    - a model
  - an agent is
    - *policy based* if it uses only a policy
    - *value based* if it uses only a value function
    - *actor critic* if it uses both a policy & a value function
    - *model free* if it does not use a model
    - *model based* if it uses a model
      - both the model free and model based approaches need a policy and/or value function
- Case study: maze,
  - with a deterministic policy, every position on the maze will tell you where to move next,
  - with value function, every position will have a value function or a negative cost function associated with it,
  - a model may try to build the maze map internally.
- Relating different components:

$$\begin{array}{cccccc} & {\scriptstyle s,a}\big\downarrow & & {\scriptstyle s}\big\downarrow & & {\scriptstyle s}\big\downarrow \\ \xrightarrow{\text{model}} & \mathcal{P}, \mathcal{R} & \xrightarrow{\text{Bellman}} & v & \xrightarrow{\arg\max} & \pi \\ & {\scriptstyle s',r'}\big\downarrow & & & & {\scriptstyle a}\big\downarrow \end{array} \tag{16}$$

II. KNOWN-MDP PREDICTION & CONTROL

- Fully observable environment:
  - in a *fully observable environment*,
$$O_t = S_t^e = S_t^a, \tag{17}$$
  - in other words, the agent has full access to the MDP model,
  - since the agent knows the model this falls under planning category,
  - given a known MDP, the goal of this section is to come up with the best policy.
- Policy in MDP:
  - a policy $\pi(a|s)$ (10), fully defines the behaviour of an agent,
  - policies are stationary (time-independent),
  - given an MDP, the goal is find a policy that maximizes some value function,
  - given a policy, an MDP can be reduced to an MRP,
$$\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle \xrightarrow{\pi(a|s)} \langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle, \tag{18}$$
    where
$$\begin{aligned} \mathcal{P}_{ss'}^\pi &\triangleq \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a, \\ \mathcal{R}_s^\pi &\triangleq \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a. \end{aligned} \tag{19}$$
- The value function of an MDP:
  - the value functions of an MDP are w.r.t. some policy $\pi(a|s)$,
  - the *state-value function* (12), of an MDP following policy π, is
$$v_\pi(s) = E_\pi[G_t | S_t = s], \tag{20}$$
  - similarly, the *action-value function* (13), of an MDP following policy π, is
$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a], \tag{21}$$
  - sometimes value function is referred to as *utility*.
- Recursions on value functions:
  - a recursion on the value function can be constructed by decomposing the value function to two components,
  - for state-value function, expanding (12, 11),
$$\begin{aligned} v_\pi(s) &= E_\pi[G_t | S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s], \end{aligned} \tag{22}$$
  - similarly, for action-value function, expanding (13),
$$\begin{aligned} &q_\pi(s, a) = \\ &\quad E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a], \end{aligned} \tag{23}$$
    * in the recursion for $q_\pi(s, a)$, we may end up in different states due to environment.
- $v_\pi(s) \leftrightarrow q_\pi(s, a)$:
  - $q_\pi(s, a) \rightarrow v_\pi(s)$,
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a), \tag{24}$$
  - $v_\pi(s) \rightarrow q_\pi(s, a)$,
$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s'). \tag{25}$$
- Bellman expectation equations:

- *Bellman equation* are obtained by substituting (25) into (24),

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right]. \tag{26}$$

  - we are averaging over both the actions $a \in \mathcal{A}$, and environments with $s' \in \mathcal{S}$,
  - note that, (26) can also be derived from (22),
- similarly, substituting (24) into (25),

$$\begin{aligned} &q_\pi(s,a) = \\ &\quad \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \left[\sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s',a')\right], \end{aligned} \tag{27}$$

  - compare (27) to (23).
- Solution of Bellman equation:
  - in (26), given some policy π, solve for v_π,
  - seeing that (26) is linear, the matrix solution is presented,
  - substituting (19) into (26),

$$\begin{aligned} v_\pi &= \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi \Rightarrow \\ (1-\gamma\mathcal{P}^\pi) v_\pi &= \mathcal{R}^\pi \Rightarrow \\ v_\pi &= (1-\gamma\mathcal{P}^\pi)^{-1}\mathcal{R}^\pi, \end{aligned} \tag{28}$$

  - since, the computational complexity is $O(n^3)$, determining value functions using (28) may not be practical for large n.
- Optimal value functions:
  - for a given s, the *optimal state-value function* $v_*(s)$ is defined as

$$v_*(s) \triangleq \max_\pi v_\pi(s), \tag{29}$$

  - given (s,a), the *optimal action-value function* $q_*(s,a)$, is defined as,

$$q_*(s,a) \triangleq \max_\pi q_\pi(s,a). \tag{30}$$

- Optimal policy theorem:
  - *partial ordering* of policies is defined as follows,

$$\pi \geq \pi' \text{ if } \forall\, s, v_\pi(s) \geq v_{\pi'}(s), \tag{31}$$

  - theorem:
    - there exists an *optimal policy* π_* such that

$$\forall\, \pi,\ \pi_* \geq \pi, \tag{32}$$

    - all optimal policies achieve the optimal value functions,

$$\begin{aligned} v_{\pi_*}(s) =& v_*(s), \\ q_{\pi_*}(s,a) =& q_*(s,a), \end{aligned} \tag{33}$$

  - optimal policy can be found by maximizing over $q_*(s,a)$,

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a q_*(s,a) \\ 0, & \text{otherwise,} \end{cases} \tag{34}$$

    - in other words, there is always a deterministic optimal policy.
- Bellman optimality equation:
  - substituting (34) into (24),

$$v_*(s) = \max_a q_*(s,a), \tag{35}$$

- from (25),

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s'), \tag{36}$$

- substituting (36) into (35), gives Bellman optimality equation,

$$v_*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right], \tag{37}$$

- in matrix form

$$\boxed{v_* = \max_{a \in \mathcal{A}} \left[\mathcal{R}^a + \gamma \mathcal{P}^a v_* \right]} \tag{38}$$

- similarly, substituting (35) into (36),

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a'), \tag{39}$$

- Bellman optimality equation is non-linear,
  * in general no closed form solution,
  * many iterative solutions available.

- Dynamic programming:
  - Richard Bellman coined the term dynamic programming (DP) in 1957,
  - *dynamic programming* is an optimization method where a complex problem is solved by combining the solutions to subproblems in a recursive manner,
  - DP has two properties,
    * optimal substructure,
      · principle of optimality applies,
      · optimal solution can be decomposed into subproblems,
    * overlapping subproblems,
      · subproblems recur many times,
      · solutions can be cached and reused.
- Application of DP on MDP:
  - three DP algorithms that can be applied to MDP are,
    * iterative policy evaluation,
    * policy iteration,
    * value iteration,
  - all three algorithms are of complexity $O(mn^2)$ per iteration, where $|\mathcal{A}| = m$.
- Iterative policy evaluation:
  - the goal of *iterative policy evaluation* is to evaluate a given policy π,
    * i.e. given $\pi \to v_\pi$,
    * the goal is to *predict* v_π,
  - first, generate a random vector v^1,
  - then, iterate over Bellman expectation equation (28), which can be written as

$$v^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v^k, \tag{40}$$

    * v^k is the state-cost function at time k,
  - theorem: as $k \to \infty$, $v^k \to v_\pi$,
  - this is an alternative to solving for an inverse matrix in (28).
- Policy iteration:
  - the goal of *policy iteration* is to find optimal cost value function π_*,
  - given a policy π, iterate,
    * evaluate v_π as discussed in (40),
    * given v_π, improve π, using greedy policy, i.e. choose a deterministic policy that prefers states with higher value function (34),
  - theorem: the above iterations converge to π_*, which is a solution to the Bellman optimality equation (37) or (38),

  - with *modified policy iteration*, rather than wait for convergence, the above iterations are performed few times or just once.
- Value iteration:
  - the goal of *policy iteration* is to find optimal cost value function π_*,
  - the iteration is over (37) or (38),
  - intermediate value functions may not correspond to any policy.
- Asynchronous DP:
  - all three algorithms discussed so far were examples of *synchronous DP*,
    * in synchronous DP, all components of a vector are updated simultaneously,
    * also called flooding algorithm,
  - *asynchronous DP* backs up states individually, in any order,
  - asynchronous DP can significantly reduce computation,
  - asynchronous DP is guaranteed to converge if all states continue to be selected.
- Three ideas for asynchronous DP:
  - *in-place DP*, where memory space is reused by updated values,
  - *prioritised sweeping*, where errors are initially computed and components with largest error values are updated first,
  - *real-time DP*, where a sample of data is generated using monte carlo methods, and states associated with such samples are updated.
- POMDP:
  - in a *partially observable environment*, the agent observes a portion of the environment,

$$S_t^a \neq S_t^e, \tag{41}$$

  - such a formalism is addressed by the *partially observable MDP*, or POMDP,
  - two possible strategies for POMDP are
    * forming *beliefs* of environment state,
      · i.e. assign a probability to every environment state,
    * use RNN to model the agent state.

III. Model-Free Prediction & Control

- Model-free prediction & control system:
  - the environment states and rewards are still observed by the agent,
  - the dynamics $\mathcal{P}$ or the reward function $\mathcal{R}$ are unknown.
- Monte-Carlo method:
  - Monte-Carlo (MC) is a model-free method that learns from episodes of experience,
  - MC learns only from complete episodes,
    * an episode is represented by a sequence of triplets

$$S_i, A_i, R_{i+1}, \tag{42}$$

    * upon termination of an episode, the return G_t (11), becomes available.
- Monte-Carlo policy evaluation:
  - goal: $\pi \to v_\pi(s)$,
  - MC policy evaluation uses empirical value function, instead of expected value (12),
  - associate two variables to each state, a counter $N(s)$ & total return $S(s)$,
  - after termination of an episode, a state is updated as follows,

$$\begin{aligned} N(s) &= N(s) + 1, \\ S(s) &= S(s) + G_t, \end{aligned} \tag{43}$$

  - after multiple episodes, the value function associated with a state is estimated to be

$$v(s) = \frac{S(s)}{N(s)}. \tag{44}$$

- First-visit / every-visit Monte-Carlo policy evaluation:
  - in *every visit* method, a state is updated every time it is visited

    - it could be updated multiple times in an episode,
  - in *first-visit*, a state is updated only the first time it is visited in an episode.
- Recursive updates:
  - can update $v(s)$ (44) recursively, without using cumulative sum $S(s)$,

$$v(s_{t+1}) = v(s_t) + \frac{G_t - v(s_t)}{N(s_t)}, \tag{45}$$

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

$$v(s_{t+1}) = v(s_t) + \alpha(G_t - v(s_t)), \tag{46}$$

    where α is a constant.
- Temporal-difference learning:
  - similar to MC method, *temporal-difference*, or TD method, learns directly from episodes of experience,
  - however, TD can also learn from *incomplete episodes*, by bootstrapping,
    - *bootstrapping* estimates the reward from current time to end of episode,
    - more specifically, the return estimate, called *TD target*, is set to,

$$G_t^{(1)} \triangleq R_{t+1} + \gamma v(s_t + 1), \tag{47}$$

    - TD can learn from non-terminating environments,
  - simplest TD method, called TD(0), uses $G_t^{(1)}$ (47), for G in (46),
  - the *TD error* term is defined to be

$$\delta_t \triangleq G_t^{(1)} - v(s_t), \tag{48}$$

  - TD is more sensitive to initial value than MC.
- Analysis:
  - consider K episodes, each with length T_k,
  - MC converges to a solution with minimum mean-squared error

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} (G_t^k - v(s_t^k))^2, \tag{49}$$

  - TD(0) converges to solution of maximum likelihood Markov model, with

$$\begin{aligned} \hat{\mathcal{P}}_{ss'}^a &= \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s'), \\ \hat{\mathcal{R}}_s^a &= \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}(s_t^k, a_t^k = s, a) r_t^k. \end{aligned} \tag{50}$$

  - TD exploits Markov property
    - usually more efficient in Markov environments,
    - TD implicitly models MDP,
  - MC does not exploit Markov property,
    - usually more effective in non-Markov environments.
- n-step return:
  - TD(0) is 1-step look-ahead algorithm,
  - MC can be thought of an ∞-step look-ahead method,
  - in between lies a spectrum of algorithms, i.e. n-step look-ahead,
  - the 1-step return (47), can therefore be generalized to n-step return, where

$$G_t^{(n)} \triangleq \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n v(s_{t+n}). \tag{51}$$

- λ-return & averaging n-step returns:

– λ-return G_t^λ is a geometric average of n-step returns,

$$G_t^\lambda \triangleq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \tag{52}$$

– the resulting policy evaluation is called TD(λ) method,
– like MC, TD(λ) can only be computed from complete episodes,
– TD(λ) is associated with the same computational cost as TD(0).

- Eligibility traces:
  – *eligibility traces* associate with each state a scalar that reflects the states eligibility to be updated from a reward,
  – the eligibility trace combines recency & frequency heuristics,

$$E_t(s) \triangleq \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s). \tag{53}$$

- Forward versus backward views:
  – the *forward-view* (52), looks into the future to compute G_t^λ,
    * forward view provides theory,
  – the backward view provides mechanism, by associating an eligibility trace for every state s,
    * more specifically, $\delta(s)$ (48), & $v(s)$ are updated as follows

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma v(s_{t+1}) - v(s_t), \\ v(s) &= v(s) + \alpha \delta_t E_t(s), \end{aligned} \tag{54}$$

    * δ_t is a scalar, whereas $v(s)$ is a vector,
    * all states are updated at every t,
  – theorem: the sum of updates is identical for forward-view & backward-view TD(λ), see (54, 46),

$$\sum_t \alpha \delta_t E_t(s) = \sum_t \alpha \left(G_t^\lambda - v(s_t)\right) \mathbf{1}(S_t = s). \tag{55}$$

- On-policy versus off-policy learning:
  – on-policy learning learns on the job,
    * learns about policy π from experience sampled from π,
  – off-policy learning looks over someones shoulder,
    * learns about policy π from experience sampled from another agent.

Algorithm 1 Sarsa(λ)

initialize $Q(s,a)$ arbitrarily, for $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
repeat (for each episode):
    $E(s,a) = 0,\ \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$,
    initialize $\mathcal{S}, \mathcal{A}$,
    repeat (for each step of episode):
        take action A, observe R, S',
        choose A', from S', using policy derived from Q
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$,
        $E(S,A) \leftarrow E(S,A) + 1$,
        $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$,
            $E(s,a) \leftarrow \gamma\lambda E(s,a)$,
        $S \leftarrow S'; A \leftarrow A'$,
    until S is terminal.