Reinforcement Learning

MATH 7339

Greg Rohlicek

March 1, 2023

What is reinforcement learning?

Very generally, a statistical learning method that seeks to optimize behavior of some agent in an environment, such that positive actions are rewarded and negative actions are penalized. Elements (in most cases) include:

- **Agent:** The decision-maker in a RL problem
- **Environment Model:** Something to mimic an environment in which an agent acts. Typically a Markov Decision Process (more on that later).
- **Policy (π):** A definition/rule of the agent's behavior at a given time; dictates agent's choices.
- **Reward Function and Value Function:** dictates the reward received by the agent as it acts. Goal: maximize our total reward received.
    - **Reward vs. Value:** Reward is typically received at a single time step, whereas value is typically long-term aggregate reward.

RL = Supervised or Unsupervised learning?

Well, not exactly: RL isn't typically used to label data, divide categories, etc. $\rightarrow$ doesn't fit nicely into supervised or unsupervised learning.
Rather, RL serves to maximize the reward an agent receives as it takes actions in its environment.

Expected reward

Let:

- $\mathbf{A}$:= set of actions our agent can take; $A_t = a$ denoting action a taken at time t
- R_t := reward received at time t

For a given action $A_t = a$, we'll receive some reward. Expected reward can be formalized as:
$q_*(a)$:= Expected reward at time t, given we chose action a.
Alternatively:

$$q_*(a) := \mathbf{E}[R_t | A_t = a]$$

Expected reward vs. actual reward

While $q_*(a)$ is solidly the expected return on an action a, since R_t is typically a random variable, the actual reward we receive may be different from this expectation.
Let's define true, observed reward $Q_t(a)$ as:

$$Q_t(a) = \frac{\text{Sum of reward received from taking } a \text{ before time } t}{\text{number of times we chose action } a \text{ before time } t}$$

Alternatively, expressing Q in the context of a single action a:

$$Q_n = \frac{R_1 + R_2 + ... + R_{n-1}}{n-1} : R_i := \text{reward after } i^{th} \text{ selection of } a$$

Notice: as $t \to \infty : Q_t(a) \to q_*(a)$. (Note: default value can be set for Q before a given action has been taken at all. Typically initialized to 0.)

Example: k-armed Bandit

Bandit = slot machine. Clarifying this now.
Structure of this problem:

- k arms to pull $\rightarrow$ k total actions we can take.
- Each of the k arms provides some reward according to a **stationary** random distribution.
- Environment: “model-free", that is, no notion of states. We’re working with the exact same k arms at every timestep.
- **Goal:** Maximize our total long-term reward. That is, find the most profitable action to take/which of the k arms has the highest expected reward $q*$.

k-armed Bandit (cont.)

One approach: *ϵ-greedy method*— Calculate our optimal action as:

$$A_t = \underset{a}{\operatorname{argmax}}\ Q_t(a)$$

Then, take this action with probability $(1 - \epsilon)$ while taking some random action with probability ϵ.
Result: Our agent will take the optimal action most of the time, but to compensate for incorrect guesses, we explore other options an ϵ fraction of the time.
Given that rewards from all k arms follow stationary distributions, with enough exploration we will eventually converge upon the true expected reward for all k arms/actions.

Convergence of Q to q_*

Let's now use Q_n to generalize some subsequent Q_{n+1}:

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i$$

$$= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right)$$

$$\vdots$$

$$= Q_n + \frac{1}{n}(R_n - Q_n)$$

We can think of this as: “Predicted reward for a given action equals previous prediction plus some weighted error term."

Convergence of Q to q_*

Robbins-Monro Algorithm: for some sequence α_n, the sequence converges if:

$$\sum_{n=0}^{\infty} \alpha_n = \infty$$

$$\sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

Going back to the general Q_{n+1} on the previous slide, let $\alpha_n(a) = \frac{1}{n} \ \forall a \in \mathbf{A}$. Now we can write:

$$Q_{n+1} = Q_n + \alpha_n(a)(R_n - Q_n)$$

Since $\alpha_n(a) = \frac{1}{n}$ satisfies the above conditions, our prediction will converge (so long as, of course, $R_n - Q_n$ is bounded)

Stationary vs. Non-Stationary problems

Suppose we choose $\alpha_n(a)$ such that our prediction doesn't converge (for example: $\alpha =$ some constant). How will that impact our policy?

Follow-up questions:

- *why* will it impact our optimal policy?
- is it ever desirable to have a prediction that doesn't converge?

Stationary vs. Non-Stationary problems

“Why will it impact our optimal policy?" – *Recall*: k-armed bandit problem assigns rewards from each arm according to a stationary random distribution.

- If there’s any variance associated with a given reward, we’re stuck updating our prediction around some mean reward rather than converging upon it

“Is it ever desirable to have a prediction that doesn’t converge?" – For non-stationary distributions of rewards. More generally, when there exist multiple **states** in our environment model.

Non-Stationary problems: Markov Decision Processes

Markov Decision Process (MDP) overview:

- $\mathbf{S}$:= state space; set of states $s \in \mathbf{S}$
- $\mathbf{A}$:= action space; set of actions $a \in \mathbf{A}$

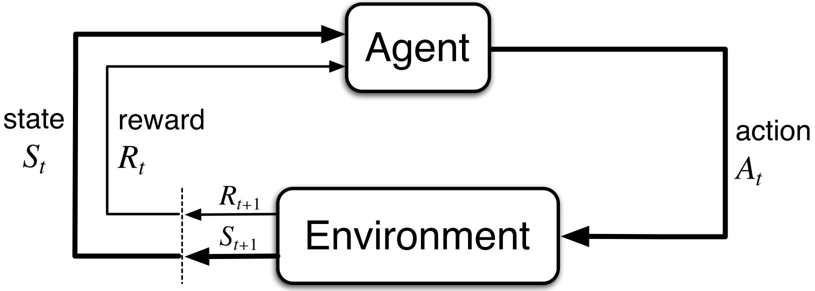
$$\begin{tikzcd}
0 \arrow[loop left, "1"] & 1 \arrow[l, "1-p"'] \arrow[r, bend left, "p"] & 2 \arrow[l, bend left, "1-p"] \arrow[r, "p"] & 3 \arrow[loop right, "1"]
\end{tikzcd}$$

State transition probability:

$$p(s'|s, a) = \mathbf{Pr}\{S_t = s'|S_{t-1} = s, A_{t-1} = a\}$$

Markov Decision Processes (cont.)

Contextualizing MDP for reinforcement learning, we must incorporate the notion of state and reward together:



Agent interacts with environment to make decisions and learn. Our non-stationary RL problem comprises the following finite sets:

- **S**:=set of states
- **A**:=set of actions
- **R**:=set of rewards

Markov Decision Processes (cont.)

Writing an updated MDP probability incorporating reward r:

$$p(s', r|s, a) = \mathbf{Pr}\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

And, using this to express expected reward:
Expected reward, state-action pair:

$$r(s, a) = \mathbf{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a)$$

Expected reward, state-action-next state triple:

$$r(s, a, s') = \mathbf{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s']$$

$$= \sum_{r \in R} r \left(\frac{p(s', r|s, a)}{p(s'|s, a)} \right)$$

Discounted Return

Now that we have introduced the notion of state, let us now try and quantify the long-term value, or **return** of a given state. First, let us define **return** as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1} R_T$$

Where:

- G_t:= return, starting at time t until horizon time T. Follows that $G_T = 0$
- γ:= discount rate; usually $0 \leq \gamma \leq 1$. Method of avoiding infinite reward/returns. Extreme values:
  - $\gamma = 0$: only care about immediate reward
  - $\gamma = 1$: care about all rewards received equally

Discounted Return (cont.)

Rewriting our definition of return:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

Here, we see that G_t is finite so long as:

- Our reward sequence R_k is bounded
- At most one of: $T = \infty$, $\gamma = 1$ is true

Niche case: $R_k = 1 \; \forall k \rightarrow G_t = \frac{1}{1-\gamma}$

State-Value Function

Now that we're dealing with states and have defined some long-term return G_t, we can express our expected future return if we were to begin in some state s (called the **state-value function** for state s):

$$v(s) = \mathbf{E}[G_t | S_t = s] = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1} | S_t = s\right]$$

Note: since this is an expected value of total long-term return, it is fundamentally different from expected reward of a given state. In particular, sufficiently large future reward may be deemed more important than present reward.

State-Action expected return

Similarly, we can formalize the expected return for a state-action pair, called the **action-value function**—that is, expected return for taking an action *a* in state *s*:

$$q(s,a) = \mathbf{E}[G_t | S_t = s, A_t = a] = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1} | S_t = s, A_t = a\right]$$

Optimal value under specific policies

Recall: a given **policy** π dictates the probabilities of taking a given action a at some time t:

- $\pi_t(a) = \mathbf{Pr}\{A_t = a\}$

We can think of our state-value and state-action equations in terms of a specific policy (rather than a general or greedy case):

- $v_\pi(s) :=$ expected return under policy π, starting in state s
- $q_\pi(s, a) :=$ expected return under policy π, taking action a in state s

In both cases, this denotes an implicit conditioning of our expected value on the probabilities of our actions.

Optimal value under specific policies

For a specific policy π and our finite set of actions, it follows that:

$$\sum_a \pi(a|s) = 1$$

Knowing this, we can link this to our state-action function q_π to express our state-value function in terms of action-value:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

Essentially: this is a calculation of value of all possible action, averaged by the probability of that action occurring.

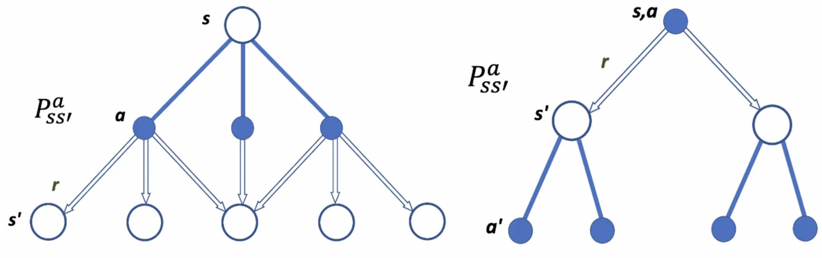
Bellman Equations for a specific policy

Let us now write out our state-value and action-value function more explicitly—that is, in terms of current state s, action a, reward r, and next state s'. These are referred to as our **Bellman Equations**:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')]$$

Bellman Equations for a specific policy



Left: graphical illustration of $v_\pi(s)$. **Right:** graphical illustration of $q_\pi(s,a)$
In both cases, we're beginning in some state/state-action pair, advancing to some next state s' with probability $P_{ss'}^{a}$, and receiving a reward r for taking action a in state s.

Bellman Optimality Equations - optimality across many policies

In considering different policies, it follows that some policies may be better than others in terms of total return. Furthermore, a state-value function or action-value function following one policy may be sub-optimal compared to one following a different policy. Therefore, it would serve us well to generalize our Bellman Equations to arrive at optimal values—thus referred to as our **Bellman Optimality Equations**.

First, let:

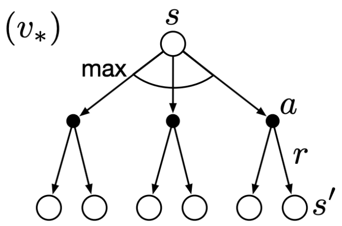
- $v_*(s) = \max_{\pi} v_\pi(s)$
- $q_*(s, a) = \max_{\pi} q_\pi(s, a)$

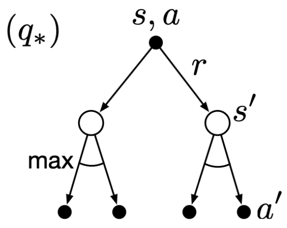
Bellman Optimality Equations (cont.)

From here, and given our policy-specific Bellman Equations, it is straightforward to generalize them for a policy-agnostic Bellman Optimality Equation:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma(\max_{a'} q_*(s', a'))]$$





Further topics in reinforcement learning

Generally speaking, material up until this point can serve as a fundamental knowledge base for more advanced methods in RL.
In practice, it is often difficult to definitively calculate some optimal policy, optimal action, etc.—too much potential for unknown.
Within the context of a Markov view of the world: we may not, for instance, always have a full view of our state space, or of state transition probabilities, etc.
Therefore, most advanced techniques in RL serve to estimate these things when we're unable to create a clean Markov Decision Process view of our problem—we may not know our optimal policy π_*, but we can approach it through repeated trial and error, for instance.

Brief example: Q-Learning

Q-Learning: a method for approximating the optimal policy of a Markov Decision Process, based upon a *learned* expected reward function $Q(s, a)$ for each state-action pair (s, a).

When is this useful?

- Full view of state space $\mathbf{S}$
- No model; unknown state transition probabilities (thus, Q-Learning is referred to as "model-free")
- No known policy to follow (thus Q-Learning is also referred to as "off policy")
    - SARSA: "on policy" counterpart algorithm to Q-Learning

Q-Learning

Q-Learning Algorithm

(First, initialize $Q(s, a)$ to some initial value(s) $\forall s, a$)

$$Q^{\text{new}}(s,a) = Q(s,a) + \alpha \left[R_t + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]$$

Where:

- $Q(s, a) :=$ current value for state action pair (s, a)
- $\alpha :=$ learning rate
- $R_t + \gamma \max_{a'} Q^{\text{old}}(s', a') :=$ Old discounted estimate of return in our new state (see: expected value in our Bellman Optimality Equations)

Q-Learning

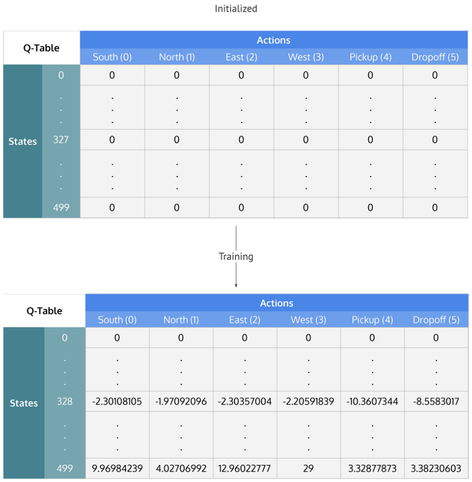
Putting this all together, what is the Q-Learning algorithm doing?

Answer: to show optimality in our Bellman Optimality Equation, we multiplied our expected return by a probability of our next state. Since this is unknown, Q-Learning instead does the following:

- Receive a reward for taking action a in state s
- Calculate an error of [new value]-[old value], scale it by some learning rate α
- Update our previous estimation of $Q(s, a)$ by this scaled error calculation

Q-Learning

Eventually, after enough time/iterations, we will approximate an optimal value function with our learned $Q(s, a)$:


Initialized
Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Sources

Textbooks:

- *Reinforcement Learning: An Introduction* — R. Sutton and A. Barto
- *Applied Reinforcement Learning With Python* — T. Beysolow

Lectures:

- *Introduction to Reinforcement Learning* — D. Silver, Google DeepMind (https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver)
- *CS 234: Reinforcement Learning* — E. Brunskill, Stanford University (https://web.stanford.edu/class/cs234/)

Sources (cont.)

Lectures (cont.)

- *Deep Reinforcement Learning through Policy Optimizaton* — P. Abbeel, J. Schulman, OpenAI/UC Berkeley
- *6.S897/HST.956: Machine Learning for Healthcare* — F. D. Johansson, MIT

Blogs

- *Towards Data Science:* https://towardsdatascience.com

Misc. required viewing

- *AlphaGo*: documentary of DeepMind's RL-based training of Go-playing agent. Available on YouTube for free: https://www.youtube.com/watch?v=WXuK6gekU1Y