Reinforcement Learning

Based on Reinforcement Learning by Richard S. Sutton

Notes/Solutions by Eashan Garg

Fall 2019

Contents

1 Introduction

1.1 Definitions

Definition. Reinforcement Learning

(Policy) Defines the behavior of a learning agent based on environmental stimuli

(Reward Signal) The end goal of any reinforcement learning problem

(Reward) Feedback to the learning agent, this quantity needs to be maximized

(Value Function) Long-run quantification of good action by the policy

(Value) Total amount of expected reward that an agent can accumulate throughout the future

(Model) Mimics the environment that a learning agent will engage with. RL can be either *model-based* or *model-free*

1.2 History

A brief history of reinforcement learning across three separate threads of research.

Optimal Control:

Late 1950s: The term *Optimal Control* was coined to describe the problem of minimizing/maximizing a dynamical system's performance over time.

Bellman 1957a: Dynamic Programming, suffers from the curse of dimensionality.

Bellman 1957b: Markov Decision Processes (MDPs).

Ronald Howard 1960: Policy Iteration Method for MDPs.

Werbos 1987: More interrelation between learning and dynamic-programming.

Chris Watkins 1989: Reinforcement Learning with MDP formalism.

Bertsekas and Tsitsklis 1996: The term *Neurodynamic Programming* is coined to combine dynamic programming and ANNs.

Trial-and-Error Learning:

Bain 1850s and Morgan 1894: Basis for learning by 'groping and experiment'.

Edward Thorndike 1911: Law of Effect describes the effect of reinforcing events on action selection.

Pavlov 1927: Reinforcement is the strengthening of a pattern as per stimulus (reinforcer) with a response.

Thomas Ross 1933: Machine that found its way through a simple maze and remembered the path through switch settings.

Turing 1948: 'Pleasure-pain system' to impact system configurations.

Shannon 1952: Maze-running mouse Theseus uses trial-and-error to find its way through a maze, although the maze itself also remembers the directions of the mouse.

Minsky 1954: Analog machine built of SNARCs (Stochastic Neural-Analog Reinforcement Calculators) that represent synaptic connections of the brain.

Farley and Clark 1954: Digital simulation of trial-and-error neural network.

Interjection — Confusion over the difference between reinforcement learning and supervised learning caused a drop in research in the 1960s and 1970s.

Minsky 1961: Several issues relevant to trial-and-error learning, such as prediction, expectation, and the *basic credit-assignment problem*.

Michie 1961: Trial-and-error system to learn tic-tac-toe called MENACE.

Michie and Chambers 1968: Improved with GLEE, pole-balancing teacher.

Andreae 1969: STeLLA learned by trial-and-error in interaction with an environment.

Widrow, Gupta, and Maitra 1973: Least-Mean-Square (LMS) modifications allow it to learn from success/failure signals as opposed to training examples. Called *selective bootstrap adaptation*, learning from a *critic*.

Tsetlin 1973: Origins of Learning Automata published posthumously.

Cross 1973: Application of Bush and Mostellar learning theory to classical economic models.

Tzanakou 1974: Alopex algorithm stochastically detects correlations between actions and reinforcement.

John Holland 1975: General theory of adaptive systems based on selection principles.

John Holland 1976: Introduced *classifier systems* including association and value functions. Key takeaway: *Genetic Algorithm*

Harry Klopf 1981a: Distinction of reinforcement and supervised learning.

Temporal-Difference Learning:

Minsky 1954: *secondary reinforcers* are stimuli paired with a primary reinforcer like pain - found significance for integration with artificial intelligence.

Arthur Samuel 1959: Checkers-playing program implemented using temporal-difference ideas.

Klopf 1972: *Generalized Reinforcement* where each component views all inputs as rewards/punishments. This idea was linked with trial-and-error learning.

Witten 1977: Combined all three threads of learning.

Barto, Sutton, Anderson 1984: *actor-critic architecture* applied to the pole-balancing problem.

Holland 1986: Incomporated temporal-difference into bucket-brigade algorithm.

Sutton 1988: Separated temporal-difference from control and introduced the $TD(\lambda)$ algorithm.

Chris Watkins 1989: Development of Q-Learning.

Gerry Tesuaro 1992: TD-Gammon plays Backgammon well.

2 Multi-armed Bandits

2.1 k-armed Bandits

Problem Statement: Select from k options repeatedly over n time steps. After each iteration, receive a reward from a fixed probability distribution. How to maximize the reward over n time steps?

Action A_t at time step t, with reward R_t. Value $q_*(a)$ can be quantified as:

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

Unfortunately, $q_*(a)$ is not known, so estimate $Q_t(a)$ at time step t and action a such that $Q_t(a) + \epsilon \approx q_*(a)$.

Moves with maximum expected value are known as *greedy actions*. Selecting a greedy action is known as exploitation, while selecting a different move is known as exploration. A motivating problem, how to balance exploitation and exploration in order to maximize empirical value?

2.2 Action-value Methods

Simple action-value method (Sample-average): Take the average of received rewards.

$$Q_t(a) := \frac{\sum_{i=1}^{t-1} R_i(A_i = a)}{\sum_{i=1}^{t-1}(A_i = a)}$$

Simple action-selection method (Greedy): Select the option with highest estimated value.

$$A_t := argmax_a Q_t(a)$$

This only uses exploitation, how about exploration? ϵ-greedy method: Select a uniformly random option with probability , else select the greedy option.
These methods are asymptotically guaranteed to select the optimal action with $1 - \epsilon$ certainly in the far future.

2.3 Incremental Implementation

Is it possible to compute sample averages in constant time-per-step and constant memory?

Incrementally calculate the new averages with constant computations. Take the formulation:

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_n + \frac{1}{n}[R_n - Q_n]$$

Where $\alpha_t(a) = \frac{1}{n}$ is the step size and $R_n - Q_n$ is the error.

2.4 Tracking a Nonstationary Problem

What if reward probabilities changed over time? Averages aren't as effective anymore, so give more weight to more recent rewards.

Use a constant step-size parameter α in place of $\frac{1}{n}$. Why?

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-1} R_i$$

Sum of the weights is one (weighted average), and the quantity $\alpha(1-\alpha)^{n-1}$ reduces in size as the reward becomes more stale. Known as an *exponential recency-weighted average.*

Relax the constraints and assume the step-size is variable. Important conditions for convergence to true action values with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \tag{1}$$

Sample-average with $\frac{1}{n}$ meets both these conditions, while exponential recency-weighted average with α does not. This is expected, since the latter formulation is important for nonstationary problems.

2.5 Optimistic Initial Values

All these estimates are biased by their initial estimates. This isn't much of a problem for the stationary method, but nonstationary methods can still be impacted by this initial value.

For stationary methods, take advantage of these priors, and use $Q_1(a)$ as a prior for the model! If $Q_1(a)$ is initialized to relatively larger values for all a, this encourages exploration as early rewards are 'disappointing.' This is unfortunately not very useful for nonstationary methods, since the exploratory phase is temporal in nature, and not specific to the initialization of the problem.

2.6 Upper-Confidence-Bound Action Selection

More room for improvement: ϵ-greedy methods weight all actions equally during exploration. Performance could improve if there was preference for actions that **could** be optimal. Two important heuristics for tendency towards what is optimal are estimated distance to what is optimal, and uncertainty of the estimate.

$$A_t := argmax_a \left[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}} \right]$$

$N_t(a)$ refers to the number of times action a is selected, where $N_t(a) = 0$ makes a a maximizing action. c controls the exploration vs exploitation tradeoff.

This is known as *upper confidence bound* (UCB) action selection, since the $argmax$'d quantity acts like an upper bound. Term one is the estimated value and term two is the uncertainity, while c is a sort of confidence interval.

2.7 Gradient Bandit Algorithms

Another approach, learn a numerical preference for each action, or $H_t(a)$. Compare relative preferences to decide what actions to take.

$$H_{t+1}(A_t) := H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \pi_t(A_t)),$$
$$H_{t+1}(a) := H_t(a) - \alpha(R_t - \overline{R_t})\pi_t(a), \forall a \neq A_t$$

α is the step-size parameter, and $\overline{R_t}$ is the average of all rewards until time t (exclusive). If the reward R_t is larger than $\overline{R_t}$, then A_t's probability is increased, else it is decreased. Other actions engage with the opposite action.

π_t is defined as a soft-max distribution, or the Gibbs/Boltzmann distribution.

$$PrA_t = a := \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}$$

Adding optimistic initialization has no effect here, due to the baseline term $\overline{R_t}$.

Test