COMP767: Reinforcement Learning

Michael Noukhovitch

Winter 2018,

Notes written from Doina Precup's lectures, David Silver's online lectures, and Sutton & Barto 2018 preprint

Contents

1 Introduction

1.1 Definitions

Reinforcement learning is:

agent-oriented learning learning by interacting with an environment

trial and error only given delayed evaluative feedback

science of the mind one which is neither natural science nor applied technology

Framework:

1. agent percieves the **state** of the environment
2. based on the state, it chooses an **action**
3. the action gives the agent a **reward**
4. a **policy** aims to maximize the agent's **long term expected reward**

1.2 Key Factors of RL

- trial and error search
- environment is stochastic
- reward may be delayed
- balancing exploration and exploitation

1.3 Classical Challenges

- reward
- information is sequential
- delayed consequences
- balancing exploration/exploitation
- non-stationarity
- fleeting nature of time and online data

2 Bandit

2.1 Definition

One-armed bandit Simplest RL problem

- pull the lever
- get some reward

- choose the best lever!

k-armed bandit extends to k arms

- at every time step t, choose an action A_t from k possibilties
- recieve a reward R_t dependent only on the action taken (i.i.d)
- $q_*(a) = \mathbb{E}[R_t | A_t = a], \forall a \in 1, \dots k$

2.2 Action Selection

greedy the action with the current highest expected value (best one so far)

exploitation choosing the greedy action

exploration choosing not the greedy action

ε-greedy balance explore/exploit by choosing exploration (random) with probability ε

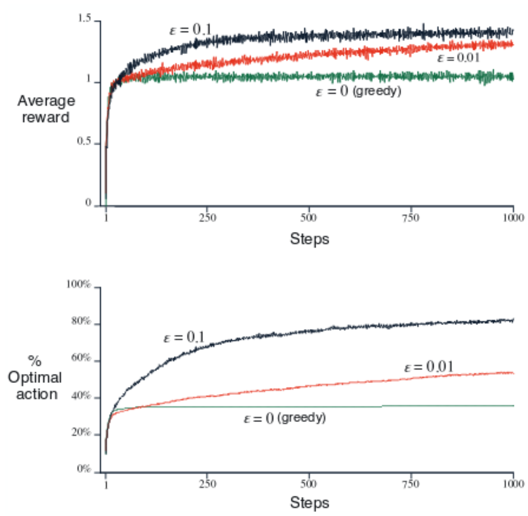

Figure 1: ϵ-greedy methods on 10-arm bandit

2.3 Learning Rules

Learn the best policy by learning the reward for an action

2.3.1 Averaging

For a single action, update the new estimate based on old estimate and step size (α), with all actions being equal

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

2.3.2 Recency-Weighted Average

stationary if the true action values DO NOT change over time

if our bandit is non-stationary, then we need to put more weight on recent samples

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

2.3.3 Optimistic

Previously we assumed $Q_1(a) = 0$, but we can start optimistically (e.g. $Q_1(a) = 5$) to encourage early exploration

2.3.4 Upper Confidence Bound

Reduce exploration over time after starting confident

- estimate upper bound on true action values
- select the action with the largest upper bound

$$A_t = \underset{a}{\operatorname{argmax}}[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}}]$$

2.3.5 Gradient-Bandit Algorithms

Don't need to learn specific rewards, just learn the **preference** $H_t(a)$, and try and make the probability of choosing an action $\pi_t(a)$ be proportional to it.

$$\begin{aligned} \pi_t(a) &\propto e^{H_t(a)} \\ &= \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}} \end{aligned}$$

if the reward for an action is better than average, increase its preference

$$H_{t+1} \quad = H_t(a) + \alpha(R_t - \bar{R}_t)(1_{a=A_t} - \pi_t(a))$$

where $\bar{R}_t$ = average R_i

2.3.6 Associative Search

associative a task where the situation/state of the agent changes the reward for an action

contextual bandit not just trial-and-error search, but also association between state and action values

full reinforcement learning trial-and-error search, association between state and action, and actions affecting the next state of the agent

2.4 Evaluations

regret the difference between best option and the one we chose $\max_a q_*(a) - q_t(a)$

expected total regret $\mathbb{E}[\sum_t \text{regret}_t]$ (optimal for UCB, Thomson sampling)

best response regret for T experimental trials after policy is fixed

2.5 Conclusions

- simple methods that can be built on
- learn from feedback
- appear to have a goal

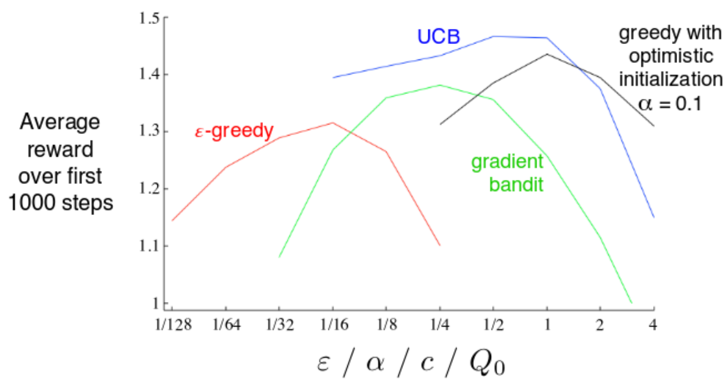

Figure 2: bandit algorithm comparison

3 Markov Decision Processes

3.1 Markov Reward Processes

3.1.1 Markov

markov property future independent of past given present

markov chain memoryless random process with states S and transition probs P, $< S, P >$

markov reward process markov chain with values: rewards R, discount factor γ

return sum of discounted rewards $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} \ldots$

value function long-term value of state s, $v(s) = E[G_t | S_t = s]$

3.1.2 Bellman Equations

Breaking value function into present and future

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

3.2 Markov Decision Processes

3.2.1 Policy

markov decision process MRP with actions A

finite MDP finite number of states, actions, and rewards

policy distribution over actions, given states $\pi(a|s)$

trajectory sequence of actions, states, and rewards

3.2.2 Value Function

state-value function expected return starting from s following π, $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$

action-value function expected return starting from s, taking action a, then following π $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

3.2.3 Bellman Expectation Equations

$$\begin{aligned} v_\pi(s) &= \sum_{a\in A} \pi(a|s) q_\pi(s,a) \\ q_\pi(s,a) &= R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_\pi(s') \end{aligned}$$

substitute one into the other to get their Bellman equations, succinctly

$$v_\pi = R^\pi + \gamma P^\pi v_\pi \tag{1}$$

3.2.4 Optimal Value

optimal state-value function maximum value function $v_*(s) = \max_\pi v_\pi(s)$

optimal action-value function maximum action-value function $q_*(s) = \max_\pi q_\pi(s,a)$

3.2.5 Optimal Policy

policy ordering $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}$ $\forall s$

optimal policy theorem $\exists$ optimal policy $\pi_* \geq \pi'$ $\forall \pi'$ and $v_{\pi_*} = v_*$

3.2.6 Bellman Optimality Equations

$$v_*(s) = \max_a q_*(s,a)$$
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

no closed form, so solve using iterative solutions

3.3 Extensions to MDPs

3.3.1 Infinite MDPs

- countably infinite states and/or action spaces
- continuous states and/or action spaces
- continuous time

3.3.2 POMDPs

POMDP partially observable MDP, observations O, observation function Z

history sequence of actions, observations, rewards

belief state probability dist over states given history $b(h)$

3.3.3 Average Reward MDP

recurrent each state visited infinite amount of times

aperiodic each state visited without any systematic period

ergodic MC stationary distribution $d^\pi(s) = \sum_{s' \in S} d^\pi(s') P_{s's}$

ergodic MDP if some MC induced by a policy is ergodic, uses *average reward* ρ

$$\rho^\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}[\sum_{t=1}^{T} R_t]$$
$$\begin{aligned} \tilde{v}_\pi(s) &= \mathbb{E}_\pi[\sum_{k=1}^{\infty} (R_{t+k} - \rho^\pi) | S_t s = s] \\ &= \mathbb{E}_\pi[R_{t+1} - \rho^\pi) + \tilde{v}_\pi(S_{t+1} | S_t s = s] \end{aligned}$$

4 Dynamic Programming

4.1 Introduction

dynamic programming solving problems by breaking down into subproblems

optimal substructure subproblems solve a larger problem

overlapping subproblems subproblems recur many times

used either for

- planning: MDP and policy $\rightarrow$ value function
- control: MDP $\rightarrow$ optimal value function, optimal policy

4.2 Policy Evalutation

4.2.1 Iterative Policy Evaluation

synchronous backups iterative evaluation of π using Bellman

$$v^{k+1} = R^{\pi} + \gamma P^{\pi} v^{k}$$

- update $v_{k+1}(s)$ from $v_k(s')$
- for iteration $k+1$
- for all states $s \in S$
- where s' is successor state of s

4.3 Policy Iteration

4.3.1 Policy Iteration Basics

1. evaluate the policy with **Bellman Expectation**, estimate v_π
2. improve the policy **greedily**, generate $\pi' \geq \pi$

always converges to optimal policy π_*

4.3.2 Convergence

convergence when policy no longer improves

$$\begin{aligned} v_\pi(s) &= v_{\pi'}(s) \\ &= \max_{a \in A} q_\pi(s, a) \end{aligned}$$

is the bellman optimality equation, $\pi = \pi_*$ $\square$

4.3.3 Modified Policy Iteration

achieve optimal policy without fully converging

ϵ-convergence converge after no more than ϵ change

k-iterations just stop after k

4.4 Value Iteration

4.4.1 Principle of Optimality

A policy $\pi(a|s)$ achieves optimal value at s iff it achieves optimal value at any successor state s

4.4.2 Value Iteration

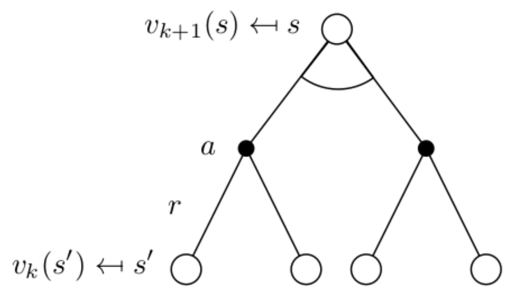

Figure 3: value-iteration

Bellman Optimality Backup, iteratively

$$v_{k+1} \leftarrow \max_{a \in A} R^a + \gamma P^a v_k$$

always converges to optimal value v_*

4.5 Extensions to DP

asynchronous DP back up states individually in any order

in-place DP don't store v_{old} only keep updated value function

prioritized sweeping update states based on their magnitude of Bellman error

real time DP only update states that agent actually visits

sample backups break curse of dimensionality by sampling instead of full backup

approximate DP approximate the value function $\hat{v}(s, w_k)$
train new $\hat{v}(s, w_{k+1})$ on results of optimality backup $s \rightarrow Bellman(\hat{v}(s, w_k))$

4.6 Contraction Mapping

contraction mapping theorem for any metric space V, complete under operator $T(v)$, where T is a γ-contraction then T converges to a fixed point at rate γ

Bellman Backup

$$\begin{aligned} T^\pi(v) &= R^\pi + \gamma P^\pi v \\ ||T^\pi(u) - T^\pi(v)||_\infty &= ||R^\pi + \gamma P^\pi u - R^\pi + \gamma P^\pi v||_\infty \qquad &= ||\gamma P^\pi (u-v)||_\infty \\ &\leq ||\gamma P^\pi||u-v||_\infty||_\infty \\ &\leq \gamma ||u-v||_\infty \end{aligned}$$

so $T(v)$ is a γ-contraction

5 Model-Free Prediction

model-free prediction estimate the value function of an unknown MDP

5.1 Monte-Carlo Learning

MC learning sample complete episodes using value = mean return

sampling update samples an expectation

5.1.1 MC Policy Evaluation

learn v_π from episodes under π, using the average of the return after visiting state s

every visit MC average returns for every visit to s

first visit MC average returns for only the first visit to s (in an episode)

5.1.2 Incremental Monte-Carlo

the mean of a sequence can be computed incrementally

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k + \mu_{k-1})$$

so we can make our MC updates incremental, and use constant step size α

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

5.2 Temporal Difference Learning

5.2.1 Basic TD

TD learning update value function towards estimated return, bootstrapping

bootstrapping update involves an estimate

For basic TD(0)

TD target estimated return $R_{t+1} + \gamma V(S_{t+1})$

TD error actual - estimated $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

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

5.2.2 Advantages of TD vs MC

- learn before knowing the final outcome
  - learn online after every step
- learn without the final outcome
  - learn from incomplete/non-terminating sequences
- low variance, some bias (vs. high variance, no bias)
  - more efficient
  - more sensitive to initial value
  - also converges (except w/ function approximation)
- exploits the Markov property
  - optimizes for max-likelihood Markov model
  - more effective in Markov environments

5.2.3 Unified View

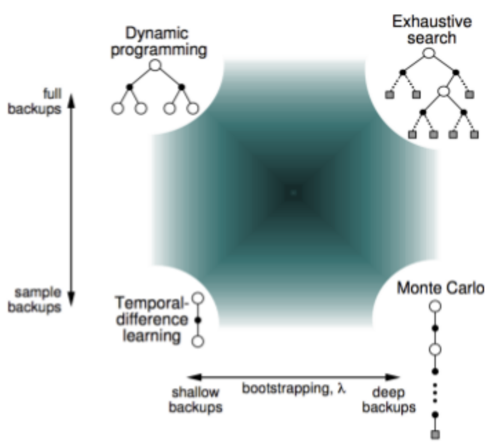

Figure 4: unified view of RL

5.3 TD-λ

5.3.1 n-step TD

TD(n) extension of TD to deeper, n-step backups

online update immediately update value function

offline update update value function at end of episode

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^n V(S_{t+n})$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

5.3.2 TD-λ

TD-λ use factor λ to combine all n-step returns

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$$

5.3.3 Eligibility Traces

frequency heuristic assign credit to most frequent states

recency heuristic assign credit to most recent states

eligibility trace combine both, $E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{1}(S_t = s)$

5.3.4 Backward-View TD-λ

forward-view look into future to compute G_t^{λ}

- offline, has to wait until end of episode

backward-view look into past and compute for any sequence, online

- keep eligibility trace for every state
- update value in proportion to eligibility trace $E_t(S)$ and TD-error δ_t

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(S_t) \leftarrow V(S) + \alpha \delta_t E_t(s)$$

5.4 Summary

Offline updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\parallel$	$\parallel$	$\parallel$
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\parallel$	$\nparallel$	$\nparallel$
Forward view	TD(0)	Forward TD(λ)	MC
	$\parallel$	$\parallel$	$\parallel$
Exact Online	TD(0)	Exact Online TD(λ)	Exact Online TD(1)

Figure 5: TD-λ summary

6 Value Function Approximation

6.1 Introduction

6.2 Incremental Methods

6.3 Batch Methods

6.3.1 Linear Least Squares Prediction

$\hat{v}(s, w) = x(s)^T w$

- sidenote: we want feature vectors $x \in X$ where $X^T X$ is full rank

since the expected update is 0

$$E_D[\Delta w] = 0$$
$$...$$

but since we don't know v_t^π

LS Monte Carlo $v_t^\pi \approx G_t$

LS TD $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$

LS TD(λ) $v_t^\pi \approx G_t^\lambda$

6.3.2 Least Squares Control

LS policy iteration

- policy evaluation with LS Q-learning
- greedy policy improvement

LS Q-learning approximate $q_\pi(s, a) \approx \hat{q}(s, a, w) = x(s, a)^T w$

- must learn off policy

7 Temporal Abstraction

Overview of *Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning* (Sutton, Precup, Singh 1999)

7.1 Options Framework

7.1.1 Definition

Options an MDP over an augmented state space

1. initiation set I_o
2. policy for that option π_o
3. termination condition β_o

7.1.2 Options

a set of options and a policy induces a Semi-MDP

7.1.3 Bellman Equations

$$\begin{aligned}
q(s, o) &= \sum_a \pi_o(a, s)(r(s, a) +) \\
u(s', o) &= (1 - \beta_o(s'))q(s', o) +_o (s') \sum_{o'} \mu(o'|s')q(s', o') \\
&= q(s', o) - \beta_o(s')(q(s', o) - v(s')) \\
&= q(s', o) - \beta_o(s')A(s', o)
\end{aligned}$$

7.2 Intra-Option Value Learning

derive TD-style algorithm in a similar way

$$q(S_t, O_t) = q(S \tag{2}$$

7.3 Option Models

7.3.1 Bellman Equations

7.3.2 Bellman Equations at SMDP level

everything behaves like an MDP over transformed reward and transition functions/models

7.3.3 TD at SMDP level