Reinforcement Learning

Author: Stephan Schiffel

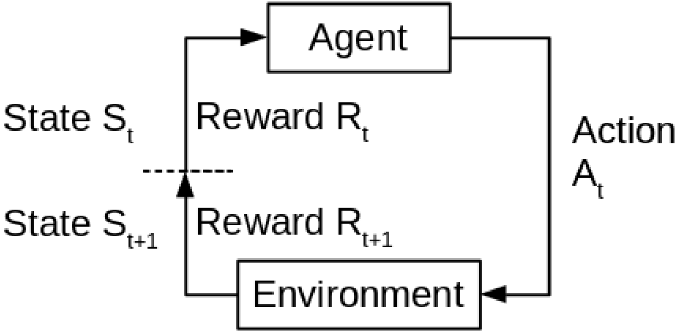
Adapted by Stefán Ólafsson

Outline

Reinforcement Learning

- Idea: an agent is learning from the interactions with an environment
- What to learn: utility of states, which actions to do
- Goal: maximizing (long-term) pay-off
- Observations: states and reward/cost of actions

Agent-Environment Interface



Agent ...

- interacts with environment at discrete time steps $t = 0, 1, 2, \ldots$
- observes state S_t and responds with action A_t
- observed resulting reward R_{t+1} and state S_{t+1}

Challenges of RL

- Evaluative feedback (reward)
- Delayed consequences / feedback
- Need for trial and error / exploration and exploitation
- Non-stationary processes

Some Successes of RL

- World's best Backgammon player (Tesauro 1995)
- Acrobatic helicopter autopilots (Ng, Abbeel, Coates et.al. 2006)
- Widely used in placement and selection of ads on the web
- Strategic decision making in Jeopardy (IBM's Watson 2011)
- Human-level performance on Atari games from pixel-level input (Google Deepmind 2015)
- Super human-level play in Go and Chess (AlphaGo, AlphaZero 2017)
- Self-play to learn *Diplomacy* (Cicero 2022)
- RL with human feedback (ChatGPT 2022)

Outline

Multi-armed Bandits

- One-armed Bandit = slot machine

- Suppose you are in a casino and have a multiple slot machines
  $\Rightarrow$ Multi-armed Bandit
- Goal: maximize your profits

Examples

- Action 1 – Reward is always 8
  value of action 1 is $q_*(1) =$
- Action 2 – 88% chance of 0, 12% chance of 100
  $q_*(2) =$
- Action 3 – reward is uniformly random between -10 and 35
  $q_*(3) =$
- Action 4 – a third 0, a third 21, and a third from
  $\{8, 9, \ldots, 18\}$
  $q_*(4) =$

Examples

- Action 1 – Reward is always 8
  value of action 1 is $q_*(1)$ =8
- Action 2 – 88% chance of 0, 12% chance of 100
  $q_*(2) =$
- Action 3 – reward is uniformly random between -10 and 35
  $q_*(3) =$
- Action 4 – a third 0, a third 21, and a third from
  $\{8, 9, \ldots, 18\}$
  $q_*(4) =$

Examples

- Action 1 – Reward is always 8
  value of action 1 is $q_*(1)$ =8
- Action 2 – 88% chance of 0, 12% chance of 100
  $q_*(2) = 0.88 \times 0 + 0.12 \times 100 = 12$
- Action 3 – reward is uniformly random between -10 and 35
  $q_*(3) =$
- Action 4 – a third 0, a third 21, and a third from
  $\{8, 9, \ldots, 18\}$
  $q_*(4) =$

Examples

- Action 1 – Reward is always 8
  value of action 1 is $q_*(1) = 8$
- Action 2 – 88% chance of 0, 12% chance of 100
  $q_*(2) = 0.88 \times 0 + 0.12 \times 100 = 12$
- Action 3 – reward is uniformly random between -10 and 35
  $q_*(3) = \sum_{i=-10}^{35} \frac{1}{46} \times i = \frac{-10+35}{2} = 12.5$
- Action 4 – a third 0, a third 21, and a third from $\{8, 9, \ldots, 18\}$
  $q_*(4) =$

Examples

- Action 1 – Reward is always 8
  value of action 1 is $q_*(1) = 8$
- Action 2 – 88% chance of 0, 12% chance of 100
  $q_*(2) = 0.88 \times 0 + 0.12 \times 100 = 12$
- Action 3 – reward is uniformly random between -10 and 35
  $q_*(3) = \sum_{i=-10}^{35} \frac{1}{46} \times i = \frac{-10+35}{2} = 12.5$
- Action 4 – a third 0, a third 21, and a third from $\{8, 9, \ldots, 18\}$
  $q_*(4) = \frac{1}{3} \times 21 + \frac{1}{3} \times \frac{8+18}{2} = 20$

The k-armed Bandit Problem

- Infinite Sequence of time steps $t = 1, 2, 3, \ldots$
- Each step you choose an action A_t from k possibilities and receive a reward R_t
- Reward is probabilistic and comes from some unknown (but fixed) distribution

$$q_*(a) = \mathbb{E}[R_t | A_t = a], \forall a \in 1, \ldots, k$$

- True values $q_*(a)$ are unknown
- Still, you must maximize the total reward
- You must both try actions to learn their values (explore) and prefer those that appear best (exploit)

Exploration vs. Exploitation

- Suppose you learn estimates

$$Q_t(a) \approx q_*(a), \forall a$$

- For example, estimate action values as sample-averages:

$$Q_t(a) = \frac{\text{sum of rewards when } a \text{ was taken}}{N_t(a)}$$

  where $N_t(a)$ is number of times a was taken before time t.
- Sample-average estimates converge to true values:

$$\lim_{N_t(a) \to \infty} Q_t(a) = q_*(a)$$

- **Greedy action** at time t: $A_t^* = \operatorname{argmax}_a Q_t(a)$
- **Exploitation** = picking the greedy action: $A_t = A_t^*$
- **Exploration** = picking a different action: $A_t \neq A_t^*$

Selection Strategy: ϵ-Greedy

- Mostly select the greedy action (with probability $1 - \epsilon$), but
- with small probability ϵ select a random action
- Perhaps the simplest way to balance exploration and exploitation

Selection Strategy: Upper Confidence Bound (UCB)

- A clever way of reducing exploration over time
- Estimate an upper bound on the true action value
- Select the action with the highest upper bound (optimistic estimated value)

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

Update Rule for Q

- Goal: update the action value estimate one step at a time (and forget the details about the previous time steps)
- Keeping a running average:

$$Q_{t+1}(a) = \begin{cases} Q_t(a) + \frac{1}{N_t(a)}\left[R_t - Q_t\right] & \text{if } A_t = a \\ Q_t(a) & \text{otherwise} \end{cases}$$

$$N_{t+1}(a) = \begin{cases} N_t(a) + 1 & \text{if } A_t = a \\ N_t(a) & \text{otherwise} \end{cases}$$

- General update rule:

$$NewEstimate = OldEstimate + StepSize\left[Target - OldEstimate\right]$$

Tracking a Non-Stationary Problem

- Suppose the true action values (and the reward distribution) change slowly over time
- Then we say the problem is non-stationary
- In this case sample-averages are not a good idea (Why?)
- Better is an exponential, recency-weighted average:

$$Q_{n+1} = Q_n + \alpha * [R_n - Q_n]$$

  where **step size** α is a constant $0 \leq \alpha \leq 1$
- There is bias due to Q_1 that becomes smaller over time.

Outline

Markov Decision Process

- ... is a (typically non-deterministic) state transition system
- Defined by transition probabilities:

$$p(s', r|s, a) = P(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$$

- Transition probabilities between states depend on agent's action
- State transitions yield rewards
- Markov property: The future only depends on current state and the agent's future actions, not on past states or actions

Goal: Learning a Policy

- Policy π is a mapping from states to action probabilities

$$\pi(a|s) = \text{ probability that } A_t = a \text{ if } S_t = s$$

- Special case: deterministic policy

$$\pi(s) = \text{ the action taken with prob. 1 in state } s$$

- Reinforcement Learning methods specify how π changes over time as a result of the agent's experience
- Typically, $\pi(a|s)$ is defined in relation to $Q(s, a)$
- Goal: Learning a policy that increases reward in the long run

Return = cumulative reward over time

- $G_t \ldots$ return from time t onwards

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $\gamma, 0 \leq \gamma \leq 1$ is the **discount rate**.

- shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted
- If $\gamma = 1$, return is the cumulative future reward
- If $\gamma = 0$, return is only the immediate reward
- Typically, $\gamma \in \{0.9, 0.99, \ldots\}$

4 Value Functions

- $v_\pi(s) \dots$ expected return in state s if all future actions are picked with policy π
- $q_\pi(s, a) \dots$ expected return in state s if the first action is a and afterwards actions are picked with policy π
- $v_*(s) = \max_\pi v_\pi(s)$ (optimal value of a state)
- $q_*(s, a) = \max_\pi q_\pi(s, a)$ (optimal value of a state-action pair)
- All of these are theoretic values, in practice we only have their estimates $V_t(s)$ and $Q_t(s, a)$

4 Value Functions

- $v_\pi(s) \dots$ expected return in state s if all future actions are picked with policy π
- $q_\pi(s, a) \dots$ expected return in state s if the first action is a and afterwards actions are picked with policy π
- $v_*(s) = \max_\pi v_\pi(s)$ (optimal value of a state)
- $q_*(s, a) = \max_\pi q_\pi(s, a)$ (optimal value of a state-action pair)
- All of these are theoretic values, in practice we only have their estimates $V_t(s)$ and $Q_t(s, a)$
- Optimal policy: π_* is optimal if it only picks optimal actions, i.e.

$$\pi_*(a|s) > 0 \text{ only when } q_*(s, a) = \max_b q_*(s, b)$$

- In other words: A policy is optimal iff it is greedy wrt. q_*.

Bellman Equation for State Values

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

- In case of finite MDPs (finitely many states and actions), this is a set of (linear) equations.
- The unique solution is v_π.

Bellman Equation for State Values

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

- In case of finite MDPs (finitely many states and actions), this is a set of (linear) equations.
- The unique solution is v_π.
- Similarly,

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

- However, these equations are non-linear.

Bellman Equation for Action Values

- Note that,

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

  and

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

- Therefore,

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

  and

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

Outline

Monte-Carlo Methods

- Idea: observe a whole episode (sequence of state transitions and rewards) and learn $Q_t(s, a)$ from the observed return G
- Define policy (somewhat) greedy wrt. $Q_t(s, a)$ and generate more episodes to learn from
  $\Rightarrow$ simultaneously learns $Q(s, a)$ and $\pi(a|s)$
- Converges to (something close to) π_* under reasonable constraints
- No model necessary (transition probabilities can be unknown)

Policy Evaluation (Estimating v_π)

- Repeatedly, generate episodes starting in state s and observe the return (cumulative discounted reward) G
- For every observation (s, G) update $V(s)$ with

$$V(s) \leftarrow V(s) + \alpha \left[G - V(s) \right]$$

- $V(s)$ will eventually converge to $v_\pi(s)$, as long as all states s are visited infinitely often (and $\alpha \rightarrow 0$ with $t \rightarrow \infty$).

Policy Evaluation (Estimating v_π)

- Repeatedly, generate episodes starting in state s and observe the return (cumulative discounted reward) G
- For every observation (s, G) update $V(s)$ with

$$V(s) \leftarrow V(s) + \alpha\,[G - V(s)]$$

- $V(s)$ will eventually converge to $v_\pi(s)$, as long as all states s are visited infinitely often (and $\alpha \to 0$ with $t \to \infty$).
- Compare with the Bellman-Equation! Why does it make sense?

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

Learning a Policy with Monte-Carlo

(On-Policy Monte-Carlo Control)

- Initialize: $Q(s, a)$ arbitrarily
- Let $\pi(a|s)$ be ϵ-greedy wrt. $Q(s, a)$
- Repeat:
  - Generate an episode using π
  - For each pair s, a in the episode
    - G is the return following s, a
    - $Q(s, a) \leftarrow Q(s, a) + \alpha \left[G - Q(s, a)\right]$
  - Let π be epsilon-greedy wrt. the updated $Q(s, a)$
- $\Rightarrow \pi$ converges to best ϵ-greedy policy (if $\alpha = \frac{1}{N(s,a)}$).

Monte-Carlo Tree Search

- Similar to on-policy Monte-Carlo control
- Only keeps track of $Q(s, a)$ for selected states, but eventually learns $Q(s, a)$ for all states
- Learns separate $Q(s, a)$ and policies for all agents simultaneously
- Uses UCB instead of ϵ-greedy, that is, it reduces exploration over time $\Rightarrow$ ensures convergence to optimal policy (in case of perfect information games)

Variation: Temporal Difference Learning

- Only observe single state transitions instead of whole episodes
- For every observation (s, s', r) update $V(s)$ with

$$V(s) \leftarrow V(s) + \alpha\,[r + \gamma V(s') - V(s)]$$

- Works for continuous environments (infinitely long episodes)

Variation: Function Approximation

- So far, $Q(s, a)$ (or $V(s)$) are represented as a lookup-table with an entry for each s, a
- For interesting problems:
  - Too many states
  - Too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, \vec{\theta}) \approx V(s) \approx v_\pi(s) \quad (1)$$
$$\hat{q}(s, a, \vec{\theta}) \approx Q(s, a) \approx q_\pi(s, a) \quad (2)$$

  - Generalize from seen states to unseen states
  - Update $\vec{\theta}$ (instead of V or Q) using MC or TD learning

Summary

- Agent-environment interface
- Multi-arm bandits
- Markov decision processes
- Bellman equation
- Monte-Carlo is a sampling-based solution of the Bellman equation interleaved with updates of the policy