Reinforcement learning

Computational Approaches to Neuroscience (NSCI 850)

Gunnar Blohm

Outline

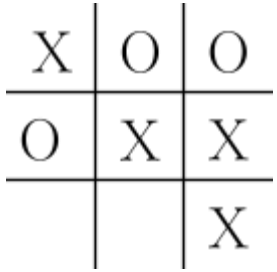
- Introduction
- The reinforcement learning problem
  - Agent-environment interactions
  - Markov properties
  - Value functions
- Solutions
  - TD(0)
  - On-policy TD control (Sarsa)
  - Off-policy Q-learning
  - Actor-Critic methods

Introduction

- Reinforcement learning (RL) = learning how to map situations into actions that maximize reward!
  - E.g. an industrial controller adjusts parameters of a process in real time to optimize production wrt. cost, quality etc.
- Elements of RL
  - Policy: learning agent's way of behaving. Mapping between estimated states of the environment and actions to be taken
  - Reward function: defines RL goal. Maps current state to reward
  - Value function: defines what's good in the long run. Value is approx. the total amount of reward an agent can expect to accumulate

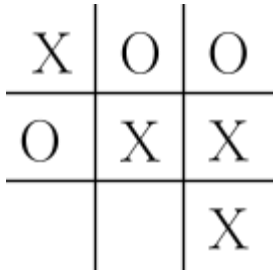
Example

- Tic-tac-toe
  - The policy is a rule that tells the player what move to make for every state of the game, i.e. every possible configuration of Xs and Os
  - For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against the opponent
  - This evaluation would then direct which policy or policies were considered next



Example

- Tic-tac-toe: RL
  - Define state value = table of the probability of winning from that state for each possible state
  - Initialize at chance (50%)
  - States with 3Xs = 100%, states with 3Os = 0% (assume we're playing Xs)
  - Select the move that you predict would lead to the greatest value (not reward), i.e. probability of winning = **exploitation** (greedy choice)
  - Occasionally, we select random moves = **exploration**!

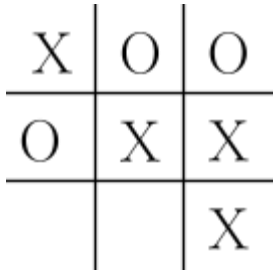


Example

- RL
  - Given current state s and future state s' (after move), and the value function V(s), the update can be written as

$$V(s) \leftarrow V(s) + \alpha \cdot [V(s') - V(s)]$$

  - = **temporal difference (TD) learning**!
  - α determines learning rate



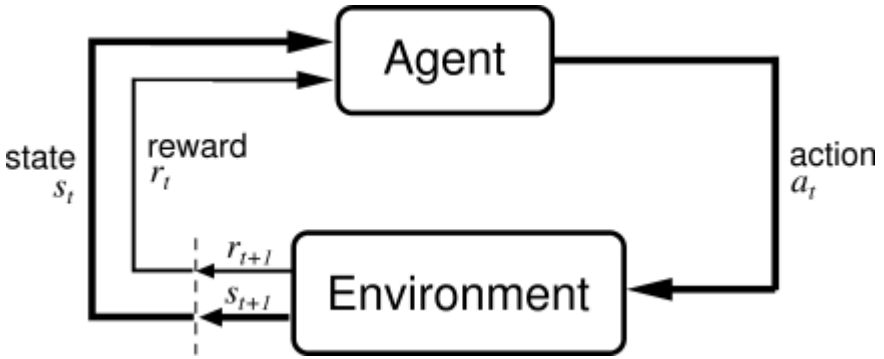
- Reinforcement learning differs from other types of learning – it *evaluates* the actions taken rather than *instructs* by giving correct actions

The reinforcement learning problem

Agent-environment interface

- Learning from interacting with the environment
  - Environment = external interactions
    - States of environment $s_t \in S$, with S possible states
  - Agent = learner and decision maker
    - Actions $a_t \in A(s_t)$, with $A(s_t)$ set of possible actions for state s_t
    - Reward $r_t \in R$
    - State and reward at time t lead to new action resulting in s_{t+1}, r_{t+1}
    - Agent's policy $\pi_t = \pi_t(s, a)$: probability that $s_t = s$ and $a_t = a$
  - Goal: learn π_t to max.

    total reward



Returns (sum of rewards)

- Goal of RL: maximize expected return

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

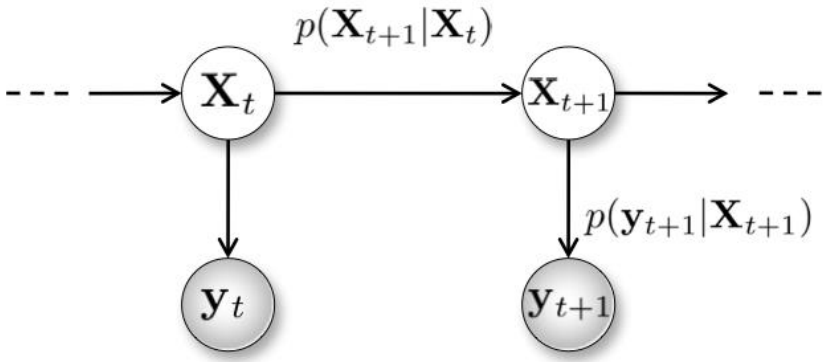
  - γ: discount rate $(0 \leq \gamma \leq 1)$
  - T can be finite or ∞

  - Works for ***episodic*** tasks or ***continuous*** tasks

Markov property

- Agent makes decision based only on state!

$$P\{s_{t+1} = s', r_{t+1} = r \mid s_t, r_t\}$$



  - s' = next state

  - Markov property: decisions and values are only a function of current state!

- An RL task that satisfies the Markov property is called a Markov Decision Process (MDP)

Markov Decision Processes

- A finite MDP (finite state and action spaces) is defined by its state and action sets and by the one-step dynamics of the environment
  - Given state s and action a, the probability for each possible next state s' is (transition probabilities)

$$P_{ss'}^{a} = P\{s_{t+1} = s' | \, s_t = s, a_t = a\}$$

  - Thus, expected value is

$$R_{ss'}^{a} = E\{r_{t+1} | \, s_t = s, a_t = a, s_{t+1} = s'\}$$

  - Note: information about the *distribution* of the expected rewards is lost…

Value Functions

- *Value functions* = functions of states (or state-action pairs) that estimate *how good* it is for the agent to be in (or perform an action in) a given state
- $V^{\pi}(s)$: state value function for policy π

$$V^{\pi}(s) = E_{\pi}\{R_t | \ s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

  - E_{π}: expected value given policy π
- $Q^{\pi}(s, a)$: value of an action under policy π

$$Q^{\pi}(s, a) = E_{\pi}\{R_t | \ s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Optimal value functions

- RL solution = find π that maximizes reward
  - π is better than π' if $V^{\pi} \geq V^{\pi'} \ \forall \ s \in S$
  - Best policy π^* leads to ***optimal state value function***

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
$$= \max_{a} E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

  - Optimal policies share the same ***optimal action-value function***

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$
$$= E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

Solutions

Overview

- Dynamic programming (model-based)
  - Well developed mathematically
  - But: require complete & accurate model of environment
- Monte-Carlo Methods (model-free)
  - Don't require a model and are conceptually simple
  - But: not suited for step-by-step incremental computations
- Temporal-difference learning
  - No model needed, fully incremental
  - But: more complex to analyze
  - **Most widely used!**
    - TD(0)
    - On-policy TD control (Sarsa)
    - Off-policy Q-learning
    - Actor-Critic methods
- Bayesian learning…

TD Prediction

- Policy evaluation = prediction problem
  - Estimate V^{π} for a given policy π (simplest TD = TD(0))

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

Prediction error

$$\leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

  - Actual return R_t

- TD(0):

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$
  until s is terminal

Example

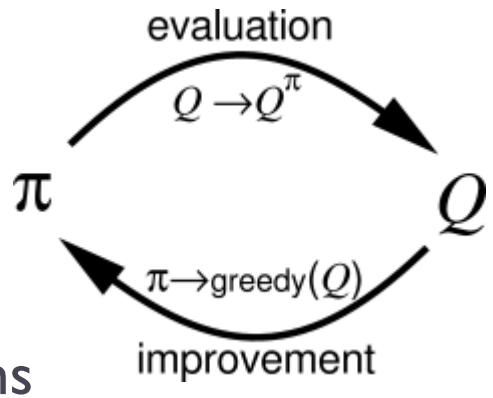
- Gambling dataset
  - 2 slot machines
  - 70%-30% reward probabilities
  - High / low volatility (2 different datasets)
    - Reward probabilities are switching between slot machine
    - Fast vs slow switch rates
- Plot choices, running average choices, real probabilities
  - d{.}.choice
  - d{.}.prep.feedbackprob
- Compute TD(0)

- Later: implement Actor-Critic control & parameter search

Sarsa: on-policy TD control

- How can we approximate optimal policies?

$$\pi(s) = arg\max_a Q(s,a)$$



- On-policy control (online)
  - Evaluate / improve policy used to make decisions

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- Sarsa:

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Chose action according to current policy with probability (1-ε) and random action with probability ε

Q-learning: off-policy control

- Off-policy control (offline): functions are separated
  - Policy used to generate behavior (*behavior* policy)
  - ≠ policy evaluated and improved (*estimation* policy)
  - Advantage: estimation policy may be deterministic (e.g., greedy), while behavior policy can sample all possible actions

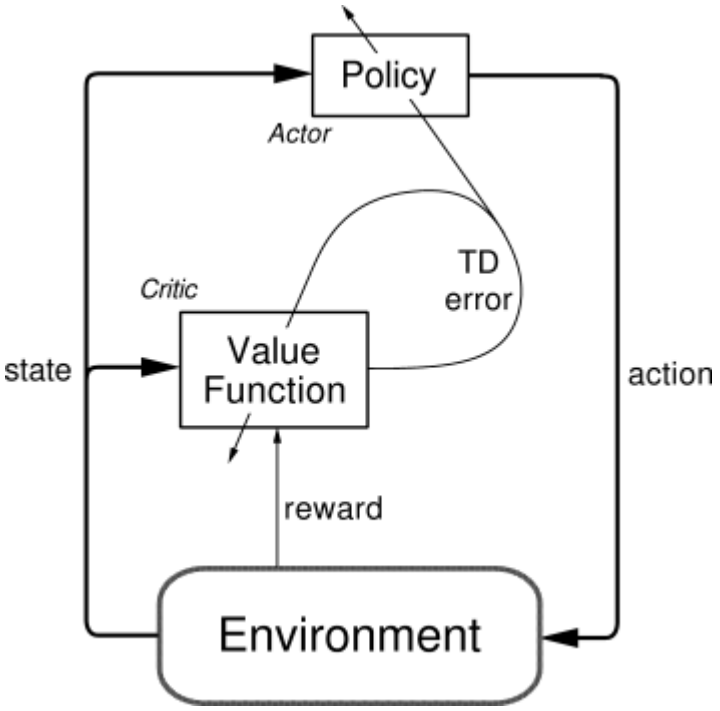
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

- Q-learn:

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
    $s \leftarrow s'$;
  until s is terminal

Actor-Critic Approach

- Explicit separation of policy and value functions
  - Actor: policy structure
  - Critic: estimated value function
    - Criticizes actions made by the actor
    - Critique = TD error
  - Learning is on-policy: critic learns about and critiques current policy



Actor-Critic Approach

- Critique evaluates new state
  - Better or worse than previous state?

$$\delta(\mathrm{t}) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

  - If $\delta(\mathrm{t}) > 0$, increase tendency to select action a_t,
  - If $\delta(\mathrm{t}) < 0$, decrease tendency to select action a_t
- If actions are generated by Gibbs softmax method

$$\pi_t(s,a) = P\{a_t = a \mid s_t = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}$$

  - Then: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta(\mathrm{t})$
  - Or: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t[1 - \pi_t(a_t, s_t)]$ (with credit assignment)

Probability of not selecting a

Further readings

- Sutton & Barto, "Reinforcement learning: an introduction". MIT Press 2005.
  - http://incompleteideas.net/book/bookdraft2017nov5.pdf
- https://hannekedenouden.ruhosting.nl/RLtutorial/Instructions.html
- McClelland, "Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises". MIT Press 2015, 2nd ed. Chapter 9