Department of Engineering Cybernetics

TTK4215 Adaptive Control

Assignment 11: Reinforcement Learning

Problem 1: Reinforcement Learning and MDPs

Use the lecture slides, the internet, or the book by Sutton and Barto as source (link posted on Blackboard) and answer the following questions:

- a) Sketch a diagram that shows the *agent/environment interface* with relevant signals, and explain briefly the idea behind reinforcement learning (RL).

  Solution: Reinforcement learning is a computational approach to learning from interaction, a formal approach to learning by "trial and error". It addresses the question of how an autonomous agent that senses and acts in an unknown environment can learn to choose optimal actions to achieve its goals. The agents goals are defined using an indirect, delayed reward signal that rewards "good" behaviour (and punishes "bad" behaviour). The goal is to somehow maximize the total reward.

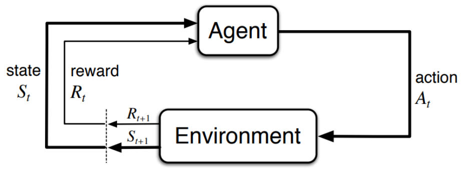


Figure 1: Diagram illustrating the agent/environment interface.

- b) Try to come up with three examples (other than those in the slides) of sequential decision processes where you state what the agent and environment is, and identify the signals included in your answer in a).

  Solution: Lots of possible answers here. Should be in the following format:

  Pick-and-Place Robot

  - Agent: The pick-and-place control system.
  - Environment: The robot (including sensors and actuators) and its workspace.
  - State: Joint angles and velocities.
  - Action: Voltages applied to motors at each joint.

  - Reward: e.g. +1 for each object successfully picked up and placed. In addition, a small negative reward might be given to discourage "jerky" motion.

- c) How does RL differ from *supervised learning*?

  Solution: Supervised learning is learning from a training set of labeled examples provided by a knowledgable external supervisor. In RL, there is no supervisor, and the agent rather learns from its own experience from acting in an environment based on a (indirect, delayed) reward signal that defines what is good and bad. An inherent challenge in RL is the problem of *temporal credit assignment*: After a sequence of actions, which actions in the sequence are to be credited with producing the reward seen at the end?

- d) The RL problem is formally stated using *Markov Decision Processes (MDPs).* What are the main components of an MDP?

  Solution: An MDP consists of

  - A set of possible states, $\mathcal{S}$.
  - A set of actions, $\mathcal{A}(s)$, that might vary with the state $s \in \mathcal{S}$.
  - A state-transition model $p(s'|s,a)$, which defines the probability of reaching state $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$.
  - A reward function $r(s,a,s')$, where $r \in \mathcal{R} \subset \mathbb{R}$.
  - A discount factor (or discount rate) $0 \leq \gamma \leq 1$, which determines the present value of future rewards.

  If the sets $\mathcal{S}$ and $\mathcal{A}$ are finite, we say we have a *finite MDP*. In general, the reward function might be stochastic. Then, $r(s,a,s')$ denotes the expected reward. We may combine the state-transition probabilities and reward function into one function $p(s',r|s,a)$. This function defines the *dynamics* of the MDP (see Sutton & Barto p.48). Note that some authors explicitly include γ in the definition of the MDP, while others do not.

- e) What is the *Markov property*?

  Solution: The Markov property is a property of the state, where the current state includes all information that is relevant for the future. That is, the probability of each possible value for S_t and R_t depends only on the immediate preceding state and action, S_{t-1} and A_{t-1}, and, given them, not at all on earlier states and actions.

- f) Even though some of the terminology is different, a lot of what we have studied in control theory fits in to the framework of RL and MDPs. Try to pair the objects mentioned in your answers in a) and d) with the terminology and notation from (optimal) control theory.

  Solution:

  - Agent - Controller
  - Environment - Plant
  - Action - Control input (u)
  - State - State (x)
  - Policy - Control law u(x)
  - Value function - (Negative of) cost function

- g) *Deep reinforcement learning* has become an extremely popular field of research during the last decade. What is the role of artificial neural networks in RL?

  Solution: Traditional RL methods for finite MDPs are *tabular methods*. When the state space becomes very large, these methods suffers from the *curse of dimensionality*. Deep neural networks enables nonlinear function approximation of value functions and/or policy representations in problems where the state and action spaces are very large, or even continuous (infinite number of allowable states). Function approximation also enables *generalization*, where experience from one situation might carry over to similar situations that the agent has not seen before.

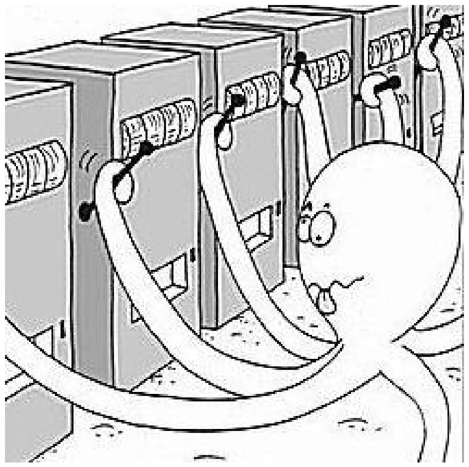
- h) Briefly search the internet and list the names of five RL algorithms (deep or not).

  Solution: Some algorithms:

  - Q-Learning
  - SARSA
  - REINFORCE
  - Deep Q-Networks (DQN)
  - Vanilla Policy Gradient (VPG)
  - Trust Region Policy Optimization (TRPO)
  - Proximal Policy Optimization (PPO)
  - Deep Deterministic Policy Gradients (DDPG)
  - Twin Delayed DDPG (TD3)
  - Advantage Actor Critic (A2C)
  - Soft Actor-Critic (SAC)

Problem 2: k-Armed Bandit Problem

Consider the following simplified learning problem, which is named by analogy to a slot machine, or ”one-armed bandit”, except that is has k levers instead of one.



You are faced repeatedly with a choice among k different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your objective is to maximize the expected total reward over some time period, for example over 1000 action selections, or time steps.

Each of the k actions has en expected or mean reward given that that action is selected. Let A_t denote the action selected at time step t, and let R_t (a real number) be the corresponding reward. The *value* of an arbitrary action a is the expected reward given that a is selected:

$$q_*(a) = \mathbb{E}[R_t | A_t = a]. \tag{1}$$

If we knew $q_*(a)$, then we would always know which action is the best. Therefore, let us try to estimate $q_*(a)$ from experience. Let the estimated value at time step t, be denoted $Q_t(a)$.

Answer the following questions:

- a) Write an equation for $Q_t(a)$ based on averaging the actual rewards received when acting in the environment.

  Solution: This method is called the *sample-average* method.

  $$Q_t(a) = \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{2}$$

  $\mathbb{1}_{predicate}$ denotes the random variable that is 1 if *predicate* is true and 0 if is not. If the denominator is zero, then we instead define $Q_t(a)$ as some default value, e.g. 0.

- b) Most practical reinforcement learning algorithms are incremental in nature. Focus now on a single action, and let Q_n denote the estimate of its action value after it has been selected $n-1$ times. Write an incremental update formula for Q_{n+1}.

Solution: Given Q_n and the nth reward, R_n, the new average of all n rewards can be computed by

$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]. \quad (3)$$

This update rule is of a form that occurs frequently in RL. The general form is

$$\text{NewEstimate} \leftarrow \mathit{OldEstimate} + \mathit{StepSize}\ [\mathit{Target} - \mathit{OldEstimate}], \quad (4)$$

where the expression $[\mathit{Target} - \mathit{OldEstimate}]$ is an *error* in the estimate. By taking steps toward the "Target", the error is reduced.

- c) Based on our best estimates $Q_t(a)$ for all k possible actions a, how would you act *greedily* at time step t to maximize reward?

  Solution: At any time step there is at least one action whose estimated value $Q_t(a)$ is greatest. These are called greedy actions. The selection method that selects one of these can be written

  $$A_t = \arg\max_a Q_t(a). \quad (5)$$

  If several actions have the same value, select one of them arbitraily.

- d) An important aspect of RL is to balance *exploration* and *exploitation*. Why is this important? Propose a simple modification of the greedy *policy* in c) to allow for exploration.

  Solution: Greedy action selection always exploits current knowledge to maximize immediate reward. A simple modification is to allow for taking a non-greedy action every once in a while, say with small probability ϵ. This action could select randomly from all other actions with equal probability. Methods utilizing such selection rules are called ϵ-greedy methods. This enables an RL agent to *explore* the environment, and improve the estimate of non-greedy action values, thus potentially improving the long-term reward.

Hint Read section 2.1-2.4 in Sutton and Barto ($\approx$ 6 pages).

Remark The k-armed bandit problem considered here has only one state. To implement the incremental update would involve storing an array of length k with one entry for each of the Q-values. When there are multiple states s, one would work with the *action-value function* $q_*(s,a)$ instead, and a *table* with an entry for each state-action combination (hence the term *tabular methods*). *Q-Learning* is an RL algorithm that tries to learn optimal behaviour by acting greedily with respect to $Q_t(s,a)$, an estimate of $q_*(s,a)$.

Problem 3: Value Functions and Dynamic Programming

A *policy* is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. When the actions taken are deterministic, then the policy is simply a mapping from state to action, i.e. $A_t = \pi(S_t)$.

- a) A crucial concept in reinforcement learning is the notion of *value function.* Define the (state-) value function $v_\pi(s)$ for a policy π.

  Solution: The value function of a state s under a policy π is the expected return when starting in s and following π thereafter.

  $$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \ , \quad \text{for all } s, \tag{6}$$

  where G_t is the return

  $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{7}$$

  and $\gamma \in [0, 1]$ is a discount factor to enable bounded return also in the case of infinite time horizons. The value chose for γ can be used to prioritize between the importance of short-term reward and the long-term performance of the agent. In the simple case of $\gamma = 1$, the return is simply the sum of all future rewards $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots$

- b) Define the optimal policy and optimal value function in terms of v_π.

  Solution: The optimal policy π_* is the policy that maximizes $v_\pi(s)$ for all states s:

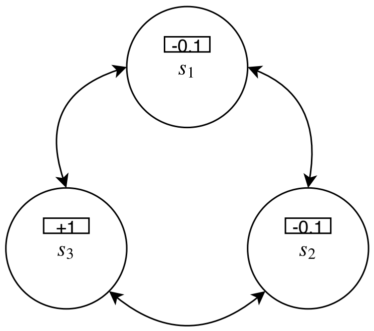
  $$\pi_*(s) \triangleq \arg\max_\pi v_\pi(s) \ , \quad \text{for all } s. \tag{8}$$

  The optimal (state-) value function v_* is the value function of an optimal policy:

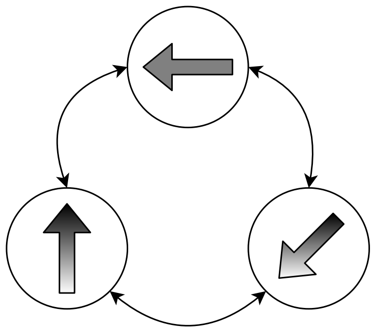
  $$v_*(s) = \max_\pi v_\pi(s). \tag{9}$$

- c) Consider an agent situated in the simple environment shown in Figure 2a. The environment has three states, s_1, s_2 and s_3. At each time step t, the agent has the choice between two actions, which is to either move in the clockwise (CW), or counterclockwise (CCW) direction, i.e. $A_t \in \{CW, CCW\}$. The environment contains a stochastic element in the sense that there is a 70% chance of success, and a 30% chance of moving in the opposite direction of what the agent intended. When entering a new state, the agent receives a positive or negative reward $r(s')$ indicated by the numbers in Figure 2a.

  The *state-transition model* $p(s'|s, a)$ contains the probabilities of reaching state s', if action a is taken in state s. When an MDP has a finite number



(a) A simple environment.



(b) An optimal policy.

of possible state and actions, the state-transition model $p(s'|s,a)$ can be represented by a three-dimensional table. Draw a table representation of $p(s'|s,a)$ for the simple environment shown in Figure 2a. Hint: There should be one probability (one number) for all combinations of s', s, a, and for each pair of s, a these should sum to one.

Solution:

s	a	s'	$p(s'\|s,a)$
s_1	CW	s_1	0.0
		s_2	0.7
		s_3	0.3
	CCW	s_1	0.0
		s_2	0.3
		s_3	0.7
s_2	CW	s_1	0.3
		s_2	0.0
		s_3	0.7
	CCW	s_1	0.7
		s_2	0.0
		s_3	0.3
s_3	CW	s_1	0.7
		s_2	0.3
		s_3	0.0
	CCW	s_1	0.3
		s_2	0.7
		s_3	0.0

- d) Give a general expression for the number of elements needed in $p(s'|s,a)$ as a function of the number of states and actions.

  Solution: If n_s is the number of states, and n_a is the number of actions, the table for $p(s'|s,a)$ will have $n_a n_s^2$ entries.

- e) The expression for v_π can be organized in a form which expresses a relationship between the value of a state and the values of its successor states. When dealing with deterministic rewards, the *Bellman equation* for the value function v_π for a deterministic policy $\pi(s)$ is given by

$$v_\pi(s) = \sum_{s'} p(s'|s, \pi(s)) \left[r(s') + \gamma v_\pi(s')\right], \tag{10}$$

where γ is the *discount factor*, and the sum is over every possible successor state s'. We have one Bellman equation for every state s, so in our example environment, there are three.

Equation (10) can be used to iteratively evaluate, i.e. calculate v_π for a given policy π. This is done in the *policy evaluation* step of the *policy iteration* algorithm. Note that when there are n states, equation (10) form n linear equations in n unknowns, so when there are not too many states, policy evaluation can be done directly using linear algebra techniques.

An optimal policy for the simple example environment with $\gamma = 0.9$ is shown in Figure 2b, and can be written $\pi(s_1) = CCW$, $\pi(s_2) = CW$ and $\pi(s_3) = CW$. For the given optimal policy, calculate $v_\pi(s)$ for each state by using equation (10) and forming a linear set of equations in the form $Ax = b$ and solving for x (x is a vector of values for $v_\pi(s)$ for all possible states s).

Solution:

$$v_\pi(s_1) = p(s_2|s_1, CCW)\Big(r(s_2) + \gamma v_\pi(s_2)\Big) + p(s_3|s_1, CCW)\Big(r(s_3) + \gamma v_\pi(s_3)\Big) \tag{11}$$

$$v_\pi(s_2) = p(s_1|s_2, CW)\Big(r(s_1) + \gamma v_\pi(s_1)\Big) + p(s_3|s_2, CW)\Big(r(s_3) + \gamma v_\pi(s_3)\Big) \tag{12}$$

$$v_\pi(s_3) = p(s_1|s_3, CW)\Big(r(s_1) + \gamma v_\pi(s_1)\Big) + p(s_2|s_3, CW)\Big(r(s_2) + \gamma v_\pi(s_2)\Big) \tag{13}$$

Inserting values gives

$$v_\pi(s_1) = 0.3\Big(-0.1 + 0.9 \cdot v_\pi(s_2)\Big) + 0.7\Big(1 + 0.9 \cdot v_\pi(s_3)\Big) \tag{14}$$

$$v_\pi(s_2) = 0.3\Big(-0.1 + 0.9 \cdot v_\pi(s_1)\Big) + 0.7\Big(1 + 0.9 \cdot v_\pi(s_3)\Big) \tag{15}$$

$$v_\pi(s_3) = 0.7\Big(-0.1 + 0.9 \cdot v_\pi(s_1)\Big) + 0.3\Big(-0.1 + 0.9 \cdot v_\pi(s_2)\Big). \tag{16}$$

Rearranging gives the linear set of equations

$$\begin{bmatrix} 1 & -0.27 & -0.63 \\ -0.27 & 1 & -0.63 \\ -0.63 & -0.27 & 1 \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \end{bmatrix} = \begin{bmatrix} 0.67 \\ 0.67 \\ -0.1 \end{bmatrix}. \tag{17}$$

Solving (17) for $v_\pi(s_i)$ gives

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ v_\pi(s_3) \end{bmatrix} = \begin{bmatrix} 3.7239 \\ 3.7239 \\ 3.2515 \end{bmatrix}. \tag{18}$$

- f) Optimal value functions v_* satisfy the *Bellman optimality condition* given by the Bellman equation

  $$v_*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s') + \gamma v_*(s')\right], \tag{19}$$

  where we again have assumed deterministic rewards.

  Since the value function v_π calculated in e) is for an optimal policy, v_π equals v_*. Show that the value function calculated in e) satisfies (19) for $s = s_1$. Hint: Direct insertion and check that LHS equals RHS. There are two actions, so calculate the sum for each action, and take the maximum.

  Solution:

  $$\begin{aligned} v_*(s_1) = \max\Big\{ & p(s_2|s_1, CW)\,(r(s_2) + \gamma v_*(s_2)) + p(s_3|s_1, CW)(r(s_3) + \gamma v_*(s_3)), \\ & p(s_2|s_1, CCW)(r(s_2) + \gamma v_*(s_2)) + p(s_3|s_1, CCW)(r(s_3) + \gamma v_*(s_3))\Big\} \\ = \max\Big\{ & 0.7(-0.1 + 0.9 \cdot 3.7239) + 0.3(1 + 0.9 \cdot 3.2515), \\ & 0.3(-0.1 + 0.9 \cdot 3.7239) + 0.7(1 + 0.9 \cdot 3.2515)\Big\} \\ = \max & \{3.4540, 3.7239\} = 3.7239, \end{aligned} \tag{20}$$

  which matches the value from e).

- g) Based on v_*, the optimal policy can be calculated using

  $$\pi_*(s) = \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s') + \gamma v_*(s')\right], \tag{21}$$

  which chooses the action that maximizes the expected value of the successor state. Note that knowledge of $p(s'|s,a)$ is needed to do a one step search forward using the model to see which action gives the best value in the next state.

Show that the optimal policy given in e) together with the optimal value function calculated in e) satisfy (21) for $s = s_1$. Hint: There are two actions, so calculate the sum for each action. The LHS should equal the actions that maximizes the sum.

Solution: $\pi_*(s_1) = CCW$, follows immediately from the solution of f).