Department of Engineering Cybernetics

TTK4215 Adaptive Control

Assignment 11: Reinforcement Learning

Problem 1: Reinforcement Learning and MDPs

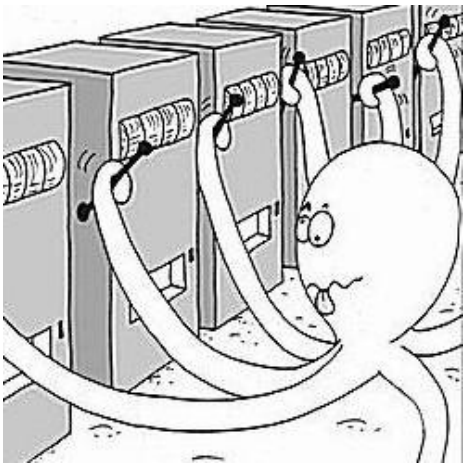
Use the lecture slides, the internet, or the book by Sutton and Barto as source (link posted on Blackboard) and answer the following questions:

- a) Sketch a diagram that shows the *agent/environment interface* with relevant signals, and explain briefly the idea behind reinforcement learning (RL).
- 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).
- c) How does RL differ from *supervised learning*?
- d) The RL problem is formally stated using *Markov Decision Processes (MDPs)*. What are the main components of an MDP?
- e) What is the *Markov property*?
- 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.
- 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?
- h) Briefly search the internet and list the names of five RL algorithms (deep or not).

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.
- 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}.
- 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?
- 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.

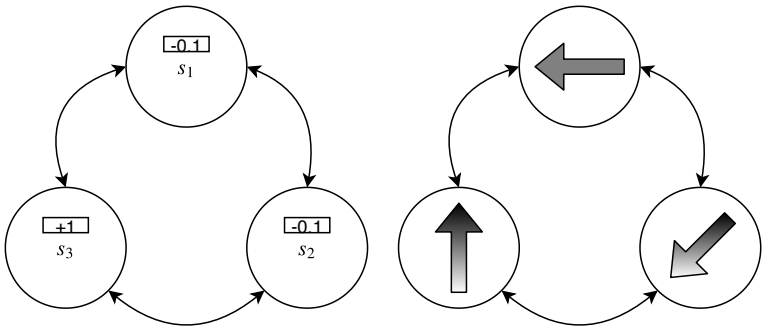
Hint Read section 2.1-2.4 in Sutton and Barto ($\approx$ 6 pages).

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.
- b) Define the optimal policy and optimal value function in terms of v_π.
- c) Consider an agent situated in the simple environment shown in Figure 1a. 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 1a.



(a) A simple environment. (b) An optimal policy.

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

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 1a. 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.

- 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.

- 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

$$v_\pi(s) = \sum_{s'} p(s'|s,\pi(s)) \left[r(s') + \gamma v_\pi(s') \right], \tag{2}$$

  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 (2) 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 (2) 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 1b, 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 using equation (2) 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).

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

$$v_*(s) = \max_a \sum_{s'} p(s'|s,a) \left[r(s') + \gamma v_*(s') \right], \tag{3}$$

  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 (3) 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.

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

$$\pi_*(s) = \arg\max_a \sum_{s'} p(s'|s,a) \left[r(s') + \gamma v_*(s') \right], \tag{4}$$

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 (4) 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.