# Final Exam

ELEN E6885: Introduction to Reinforcement Learning

December 6, 2019

**Problem 1** (**20 Points, 2 Points each**)

True or False. No explanation is needed.

1. Reinforcement learning uses the formal framework of Markov decision processes (MDP) to define the interaction between a learning agent and its environment in terms of states, actions and rewards. True

2. MDP instances with small discount factors tend to emphasize near-term rewards. True

3. If the only difference between two MDPs is the value of discount factor $\gamma$, then they must have the same optimal policy. False

4. Compared to $\epsilon$-greedy and softmax, UCB does not explore by sampling but rather inflates the estimated expected payout according to its uncertainty. True

5. For an infinite horizon MDP with a finite number of states and actions and with a discount factor $\gamma \in (0, 1)$, value iteration is not guaranteed to converge. False

6. Typically, each iteration of a basic version of Monte Carlo Tree Search (MCTS) consists of four steps: selection, expansion, simulation and backup. True

7. The on-line $\lambda$ return algorithm is equivalent to the on-line TD($\lambda$) in terms of the total update to a value function at the end of an episode. False

8. In deep Q-network (DQN), experience replay randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. True

9. The objective of using advantage function is to reduce variance of policy gradient. True

10. Actor-critic algorithms maintain two sets of parameters: "critic" updates value function parameters and "actor" updates policy function parameters. True

**Problem 2** (**18 Points, 3 Points each**)

Short-answer questions.

1. Consider a 100x100 grid world domain where the agent starts each episode in the bottom-left corner, and the goal is to reach the top-right corner in the least number of steps. To learn an optimal policy to solve this problem you decide on a reward formulation in which the agent receives a reward of +1 on reaching the goal state and 0 for all other transitions. Suppose you try two variants of this reward formulation, (P1), where you use discounted returns with $\gamma \in (0, 1)$, and (P2), where no discounting is used. As a consequence, a good policy can be learnt in (P1) but no learning in (P2), why?

   Solution: In (P2), since there is no discounting, the return for each episode regardless of the number of steps is +1. This prevents the agent from learning a policy which tries to minimize the number of steps to reach the goal state. In (P1), the discount factor ensures that the longer the agent takes to reach the goal state, the lesser reward it gets. This motivates the agent to find the shortest path to the goal state.

   Rubrics: 1 point for just mentioning the reason is the discount factor; 2 more points for correctly explaining how the discount factor causes the phenomenon.

2. Assume the goal of a reinforcement learning problem is to learn a policy that can be used by a robot in the real world. However, we only have access to simulation software, not the robot directly. We know that the simulation software is built using the transition model $P_{\text{sim}}(s, a, s')$ which is unfortunately different than the transition model that governs our real robot, $P_{\text{real}}(s, a, s')$. Given that samples are drawn from the simulator, a few options are proposed below as the new update rule for the Q-value functions for the real world robot:

   (a) $Q(s,a) \leftarrow Q(s,a) + \alpha \left( P_{\text{sim}}(s,a,s')[r + \gamma \max_{a'} Q(s',a')] - Q(s,a) \right)$

   (b) $Q(s,a) \leftarrow Q(s,a) + \alpha \left( \frac{P_{\text{real}}(s,a,s')}{P_{\text{sim}}(s,a,s')}[r + \gamma \max_{a'} Q(s',a')] - Q(s,a) \right)$

   (c) $Q(s,a) \leftarrow Q(s,a) + \alpha \left( \frac{P_{\text{sim}}(s,a,s')}{P_{\text{real}}(s,a,s')}[r + \gamma \max_{a'} Q(s',a')] - Q(s,a) \right)$

   Which one above is the correct update rule? Provide an explanation for your choice.

   Solution: (*b*) is correct. This is the importance sampling idea in off-policy learning. The Q-value functions should be weighted by $\frac{P_{\text{real}}(s,a,s')}{P_{\text{sim}}(s,a,s')}$, so that they are correct in expectation instead of sampling from the correct distribution directly.

   Rubrics: 1 point for answering (*b*); 2 more points for answering importance sampling or the idea of how to recover the expectation by sampling from a different distribution.

3. What is the disadvantage of Watkins's $Q(\lambda)$ algorithm?

   Solution: The disadvantage of Watkins's $Q(\lambda)$ is that early in learning, the eligibility trace will be "cut" (zeroed out) frequently resulting in little advantage to traces.

   Rubrics: 2 point for answering in Watkins's $Q(\lambda)$ algorithm, the eligibility trace will be "cut" (zeroed out) when a non-greedy action is taken; 1 more point for pointing out that it will happen frequently in the early phase of learning.

4. In solving a multi-arm bandit problem using the policy gradient method, are we assured of converging to the optimal solution?

Solution: No. Depending upon the properties of the function whose gradient is being ascended, the policy gradient approach may converge to a local optimum.

Rubrics: 1 point for answering "No"; 2 more point for mentioning converging to a local minimum or something alike.

5. Compared with the stochastic policy gradient, the deterministic policy gradient (DPG) can be estimated much more efficiently and perform better in a high-dimensional task, i.e., much lower computation cost. Why?

   Solution: DPG is the expected gradient of action-value function which integrates over the state space, while the stochastic policy gradient integrates over the state and action space.

   Rubrics: 2 point for answering that DPG integrates only over the state space; 1 more point for mentioning that the stochastic policy gradient integrates over both the state and action space.

6. List two benefits of using multiple agents in an asynchronous manner in A3C.

   Solution: First, it helps to stabilize the training. Since each agent has its own copy of the environment, agents are allowed to explore different parts of the environment as well as to use different policies at the same time. In other words, different agents will likely experience different states and transitions. Therefore, when agents update the global parameters with their local parameters in an asynchronous manner, the global parameters update will be less correlated than using a single agent.

   Second, the nature of multi-threads in A3C indicates that A3C needs much less memory to store experience, i.e., no need to store the samples for experience replay as that used in DQN.

   Furthermore, the practical advantages of A3C is that it allows training on a multi-core CPU rather than GPU. When applied to a variety of Atari games, for instance, agents achieve a better result with asynchronous methods, while using far less resource than these needed on GPU.

   Rubrics: Listing any two of the three benefits above earns full credits. 1.5 points each

**Problem 3 (15 Points)**

Consider a 2-state MDP. The transition probabilities for the two actions $a_1$ and $a_2$ are summarized in the following two tables, where the row and column represents from-state and to-state, respectively. For example, the probability of transition from $s_1$ to $s_2$ by taking action $a_1$ is 0.8. The reward function is $\mathcal{R}_{s_1}^{a_1} = \mathcal{R}_{s_1}^{a_2} = 1$ and $\mathcal{R}_{s_2}^{a_1} = \mathcal{R}_{s_2}^{a_2} = 0$. Assume the discount factor $\gamma = 0.5$.

Table 1: Transition probabilities

(a) action $a_1$

| | $s_1$ | $s_2$ |
|---|---|---|
| $s_1$ | 0.2 | 0.8 |
| $s_2$ | 0.6 | 0.4 |

(b) action $a_2$

| | $s_1$ | $s_2$ |
|---|---|---|
| $s_1$ | 0.6 | 0.4 |
| $s_2$ | 0.2 | 0.8 |

1. [4 Pts] Let $v_1^*, v_2^*$ denote the optimal state-value function of state $s_1, s_2$, respectively. Write down the Bellman optimality equations with resepect to $v_1^*$ and $v_2^*$.

   Solution: The Bellman optimality equation is as follows:

$$v_1^* = \max(1 + 0.1v_1^* + 0.4v_2^*, 1 + 0.3v_1^* + 0.2v_2^*)\,,$$
$$v_2^* = \max(0.3v_1^* + 0.2v_2^*, 0.1v_1^* + 0.4v_2^*)\,.$$

   Rubrics: 2 points for each correct equation.

2. [3 Pts] Based on your answer to the previous question, prove that $v_1^* > v_2^*$.

   Solution: Based on the Bellman optimality equations for $v_1^*$ and $v_2^*$, we have $v_1^* = 1+v_2^* > v_2^*$.

   Rubrics: 3 points for any correct proof; 1 point if trying to prove by contradiction but the proof itself is not fully correct.

3. [8 Pts] Based on the fact that $v_1^* > v_2^*$, find out the optimal value function $v_1^*$ and $v_2^*$. And write down the optimal policy of this MDP.

   Solution: Given that $v_1^* > v_2^*$, from the Bellman optimality equations, we have

$$v_1^* = 1 + 0.3v_1^* + 0.2v_2^*\,,$$
$$v_2^* = 0.3v_1^* + 0.2v_2^*\,,$$

   i.e., $v_1^* = 1.6$ and $v_2^* = 0.6$. The optimal policy from states $s_1$ and $s_2$ is to take action $a_2$ and $a_1$, respectively.

   Rubrics: 1 point for each correct equation above; 2 more points for each correct value of the value function; 1 more point for each correct optimal action.

**Problem 4 (10 Points)**

Consider a stochastic policy following a normal distribution where the parameters are the mean and variance of the normal distribution to which the actions are selected. That is,

$$\pi(a; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \qquad \sigma > 0.$$

To implement the policy gradient algorithm, we first need to compute score functions. In this problem, please compute the score function with respect to $\mu$ and $\sigma$, respectively. In other words, compute

$$\frac{\partial \ln \pi(a; \mu, \sigma)}{\partial \mu} \qquad \text{and} \qquad \frac{\partial \ln \pi(a; \mu, \sigma)}{\partial \sigma}.$$

Solution:

$$\frac{\partial \ln \pi(a; \mu, \sigma)}{\partial \mu} = \frac{\partial \ln \frac{1}{\sqrt{2\pi\sigma^2}}}{\partial \mu} + \frac{\partial -\frac{(a-\mu)^2}{2\sigma^2}}{\partial \mu} = \frac{a-\mu}{\sigma^2}.$$

$$\frac{\partial \ln \pi(a;\mu,\sigma)}{\partial \sigma} = \frac{\partial \ln \frac{1}{\sqrt{2\pi\sigma^2}}}{\partial \sigma} + \frac{\partial -\frac{(a-\mu)^2}{2\sigma^2}}{\partial \sigma} = -\frac{1}{\sigma} + \frac{(a-\mu)^2}{\sigma^3}.$$

Rubrics: 5 points each. For each solution, with correct derivations but incorrect results, students can earn 3 points.

**Problem 5 (12 Points)**

*Deep Q-Network* (DQN) by Google DeepMind is a ground-breaking result that successfully combines Q-learning with deep neural network using novel strategies to stabilize the Q-learning. In the training process of DQN, the agent learns directly from the high dimensional raw data, using the same network architecture and hyperparameters for all games.

1. [4 Pts] In DQN, optimizing $\theta$ can be done sequentially by minimizing a sequence of loss functions $\mathcal{L}_i(\theta_i)$ that is optimized at each iteration,

$$\mathcal{L}_i(\theta_i) = E_{s,a,r,s'\sim\rho}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i) - Q(s,a;\theta_i)\right)^2\right], \tag{1}$$

where $\rho$ is a joint probability distribution over states $s$, $s'$, reward $r$ and action $a$. DQN adopts the *fixed target Q-network*, which uses an older set of the parameters $\theta^-$ for the Q learning target. Please rewrite the loss function $\mathcal{L}_i$ with the fixed Q learning target.

Solution:

$$\mathcal{L}_i(\theta_i) = E_{s,a,r,s'\sim\rho}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta^-) - Q(s,a;\theta_i)\right)^2\right].$$

Rubrics: No partial credits.

2. [4 Pts] Then, to optimize the weights $\theta_i$ of the deep neural network at iteration $i$, we differentiate the loss function (with the fixed Q learning target) with respect to $\theta$, i.e., $\nabla_{\theta_i}\mathcal{L}_i$. Please write $\nabla_{\theta_i}\mathcal{L}_i$ in terms of $\nabla_{\theta_i}Q(s,a;\theta_i)$ .

Solution:

$$\nabla_{\theta_i}\mathcal{L}_i(\theta_i) = 2E_{s,a,r,s'\sim\rho}\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a';\theta^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i}Q(s,a;\theta_i)\right]. \tag{2}$$

Rubrics: 2 points if missing the constant factor 2.

3. [4 Pts] One can see that it is nontrivial to compute the full expectations in the above gradient. What method is widely used to iteratively find the optimal $\theta$ without computing the expectation?

Solution: Stochastic gradient decent (SGD).

Rubrics: Any expression that is equivalent to SGD can earn full credits.

**Problem 6 (25 Points)**

Consider the small grid world MDP as shown in Fig. 1. The states are grid squares, identified by their row and column numbers (row first). The agent always starts in state $(1, 1)$. There are multiple terminal states with corresponding rewards shown as numbers inside squares. Rewards are 0 in non-terminal states. The reward for a state is received during the transition as the agent moves into that state. There are four possible actions (North, East, South and West) from each non-terminal state. Each action will deterministically cause the agent to move into the corresponding neighboring state (or stay in place if the action tries to move out of the grid). The following are three episodes from runs of the agent through this grid-world:

$$(1,1), N, 0, (2,1), E, 0, (2,2), S, -60, (1,2),$$
$$(1,1), N, 0, (2,1), E, 0, (2,2), E, 0, (2,3), N, +60, (3,3),$$
$$(1,1), N, 0, (2,1), E, 0, (2,2), E, 0, (2,3), S, +80, (1,3),$$

where $N, E$ and $S$ represents North, East and South action, respectively, and the number after each action is an immediate reward. For example, $(2,2), S, -60, (1,2)$ means that the agent took action South from state $(2,2)$, received an immediate reward $-60$ and ended up in state $(1,2)$. Assume the discount factor $\gamma = 0.5$.

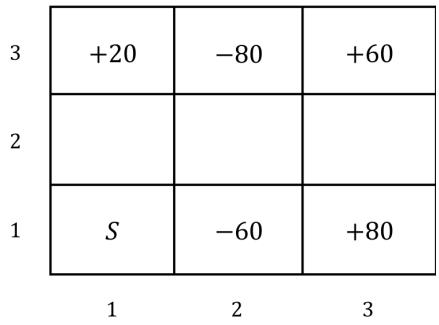


Figure 1: Small grid world MDP

1. [4 Pts] Using a learning rate of $\alpha = 0.1$, and assuming initial Q values of 0, what updates to $Q((2,2), E)$ does *on-line* TD(0) method make after the above three episodes?

   Solution: There is no update to $Q((2,2), E)$ after three episodes, i.e., $Q((2,2), E) = 0$.

   Rubrics: 1 point for deriving $Q((2,2), E) = 0$ after the first episode; 1.5 more point each for deriving $Q((2,2), E) = 0$ after the second and the third episode.

2. [11 Pts] Using a learning rate of $\alpha = 0.1$, and assuming initial Q values of 0, what updates to $Q((2,2), E)$ does *on-line backward-view* TD($\lambda$) method make after the above three episodes? Use $\lambda = 0.2$.

   Solution: There is no update to $Q((2,2), E)$ after the first episode. The accumulating eligibility trace of the state-action pair $((2,2), E)$ is $0, 0, 1, \gamma\lambda = 0.1$ during the second and the third episode. Therefore, the update to $Q((2,2), E)$ after the second episode is:

   $$Q((2,2), E) = Q((2,2), E) + 0.1 \times (60 - Q((2,3), N)) \times 0.1 = 0.6.$$

And the update to $Q((2,2),E)$ after the third episode is:

$$\begin{aligned} Q((2,2),E) &= Q((2,2),E) + 0.1 \times (0 + Q((2,3),S) - Q((2,2),E)) \times 1 \\ &= 0.6 - 0.1 \times 0.6 \times 1 = 0.54, \\ Q((2,2),E) &= Q((2,2),E) + 0.1 \times (80 - Q((2,3),S)) \times 0.1 \\ &= 0.54 + 0.1 \times 80 \times 0.1 = 1.34. \end{aligned}$$

Rubrics: 1 point for deriving $Q((2,2),E) = 0$ after the first episode; for the second episode, 2 points for finding out the correct eligibility trace sequence, 2 more points for deriving $Q((2,2),E) = 0.6$; for the third episode, 2 points for finding out the correct eligibility trace sequence, 2 more points each for deriving $Q((2,2),E) = 0.54$ after step $(2,2), E, 0, (2,3)$ and $Q((2,2),E) = 1.34$ after step $(2,3), S, +80, (1,3)$.

3. [10 Pts] Consider a feature based representation of the Q-value function:

$$\hat{Q}(s,a,\boldsymbol{w}) = x_1(s,a)w_1 + x_2(s,a)w_2 + x_3(s,a)w_3,$$

where $x_1(s,a)$ is the row number of the state $s$, $x_2(s,a)$ is the column number of the state $s$, and $x_3(s,a) = 1,2,3,4$ if $a$ is $N, E, S, W$, respectively. For example, if $s = (1,1)$ and $a = N$, then $\boldsymbol{x}(s,a) = [x_1(s,a), x_2(s,a), x_3(s,a)]^\top = [1,1,1]^\top$. Given that all $w_i$ are initially 0, what are their values using *on-line backward-view* TD($\lambda$) after the first episode? Use $\lambda = 0.2$ and learning rate of $\alpha = 0.1$.

Solution: By the definition of features, we have $\boldsymbol{x}((1,1),N) = [1,1,1]^\top$, $\boldsymbol{x}((2,1),E) = [2,1,2]^\top$, $\boldsymbol{x}((2,2),S) = [2,2,3]^\top$. The accumulating eligibility trace is $\boldsymbol{e}_t = \gamma\lambda\boldsymbol{e}_{t-1} + \boldsymbol{x}(s,a)$. So the sequence of eligibility traces in the first episode is $[1,1,1]^\top$, $[2.1, 1.1, 2.1]^\top$ and $[2.21, 2.11, 3.21]^\top$. Therefore, the update to weights after the first episode is:

$$\begin{aligned} w_1 &= 0 + \alpha \times (-60) \times 2.21 = -13.26, \\ w_2 &= 0 + \alpha \times (-60) \times 2.11 = -12.66, \\ w_3 &= 0 + \alpha \times (-60) \times 3.21 = -19.26. \end{aligned}$$

Rubrics: 2 points each for finding out the correct eligibility trace at each step; 1 more point for figuring out that only the last step would cause weight update; 1 more point each for a correct weight update.