

Multi-Agent Systems - Lecture 6 (13.12.06)

Summarized by Itai Ashlagi

1 Optimal Teaching and MDP's

In the previous lecture we observed that the teacher-student model is an instance of partially controlled multi agent systems. The teacher is the controlled agent, which "knows" the model, whereas the student is the uncontrolled learning agent. We begin by defining an optimal teaching policy. A *teaching policy* is a function from the set of histories to the set of actions that returns an actions. That is, after each iteration the teaching policy determines the next action. The teacher's goal is to teach the agent "fast".

Let $u(a)$ be the value the teacher assigns to a student's action, a . Let π be the teacher's policy, and let $Pr_{\pi,k}$ be the probability distribution on the student's actions at iteration k . That is, $Pr_{\pi,k}$ is the probability the student will play action a at iteration k . The value of the strategy (policy) π is defined by

$$val(\pi) = \sum_{k=0}^{\infty} \gamma^k E_k(u),$$

where $E_k(u)$ is the expected value of u . That is,

$$E_k(u) = \sum_{a \in A_s} Pr_{\pi,k}(a)u(a),$$

where A_s is the student's set of actions. γ is a discounted factor, which may be interpreted in this model as follows: the teacher does not know how long she will teach, and therefore learning "earlier" is more successful. An optimal teaching policy is a policy $\hat{\pi}$, such that $\hat{\pi} \in \arg \max_{\pi} val(\pi)$, i.e. a policy that maximizes the discounted expected value of the student's actions.

We next define a Markov Decision process (MDP), and we will show that teaching can be viewed as an MDP. Therefore, we will be able to apply well-known solutions in MDP's, such as value-iteration, to find the optimal teaching policy.

In an MDP a decision maker, at each time, observes her current state, receives her payoff, which depends on the state, and chooses an action. The

current state and her action determine the distribution over the next state. The goal of the decision maker is to maximize some function of the payoffs obtained in the process.

Definition 1 An MDP is a tuple (S, A, P, r) , where S is the state space, A is the decision maker's set of possible actions, $P : S \times S \times A \rightarrow [0, 1]$ is the probability of a transition between states given the decision maker's action, and $r : S \rightarrow \mathbb{R}$ is the reward function.

A *policy*¹ is a function $\pi : S \rightarrow A$. For every policy π of the decision maker, and an initial state $s \in S$, P induces a probability distribution $P_{s,\pi,k}$ over S . That is $P_{s,\pi,k}(s')$ is the probability that the decision maker will be in state s' in the k^{th} iteration under policy π , given that the current state is s .

The discounted sum of the expected values of the payoffs received by the decision maker using policy π , starting at state $s \in S$ is

$$\sum_{k=0}^{\infty} \gamma_0^k \left(\sum_{s' \in S} P_{s,\pi,k}(s') r(s') \right). \quad (1)$$

As usual, an optimal policy for the decision maker will be a policy π that maximizes (1).

We next show that the teaching-student model can be viewed as an MDP. Suppose the student's set of possible states is Σ , his action set is A_s , and the teacher's action set is A_t . We assume: (i) There exists a transition function, $\tau : \Sigma \times A_s \times A_t \rightarrow \Sigma$, which determines the student's next state as a function of the current state and the joint action; (ii) The student action at state s is given by a function ρ , which is defined as follows: $\rho(s, a)$ is the probability of choosing action a at state s ; and (iii) The teacher observes the student's state and his chosen actions.

Under assumptions (i)-(iii), the teacher only needs to choose an action in A_t for any state in Σ . That is, a policy for the teacher is a function $\pi : \Sigma \rightarrow A_t$.

We define the teacher's MDP as follows: $TMDP = (\Sigma, A_t, P, U)$, where

$$P(s, s', a_t) \triangleq \sum_{a_s \in A_s} \rho(s, a_s) \delta_{s', \tau(s, a_s, a_t)},^2$$

and

$$U(s) \triangleq \sum_{a_s \in A_s} \rho(s, a_s) u(a_s).$$

¹Also called a Markov policy.

² $\delta_{a,b} = 1$ if $a = b$, and 0 otherwise.

Claim: The optimal teaching policy is given by the optimal policy in the TMDP.

Proof. The optimal policy π in the TMDP is a policy that for each s maximizes

$$\sum_{k=0}^{\infty} \gamma_0^k \left(\sum_{s' \in S} P_{s, \pi, k}(s') U(s') \right).$$

Hence, π maximizes

$$\sum_{k=0}^{\infty} \gamma_0^k \left(\sum_{s' \in S} P_{s, \pi, k}(s') \sum_{a_s \in A_s} \rho(s', a_s) u(a_s) \right). \quad (2)$$

However, (2) =

$$\sum_{k=0}^{\infty} \gamma_0^k \sum_{a_s \in A_s} \sum_{s' \in S} \rho(s', a_s) P_{s, \pi, k}(s') u(a_s). \quad (3)$$

Observe that $\sum_{s' \in S} \rho(s', a_s) P_{s, \pi, k}(s')$ is the probability that a_s is the action chosen by the student at time k , given the initial state s . Hence, (3) is exactly $val(\pi)$. \square

1.1 Q-Learning

In this section we explore a classic learning algorithm for the student called called *Q-learning*. In this algorithm, the learner (student) keeps for each state-action pair (s, a) , a Q-value $q(s, a)$, which is updated during the algorithm. The learner, at every state s chooses an action according to some distribution, receives a payoff r , and moves to a new state s' . The update rule for $q(s, a)$ is defined as follows:

$$q(s, a) = (1 - \alpha)q_{old}(s, a) + \alpha[r + \gamma V(s')],$$

where

$$V(s') = \max q(s', a_s).$$

Here α is called the *learning rate*; γ is called the *discount factor*.³ We assume that $0 \leq \gamma < 1$, and $0 < \alpha < 1$. $V(s')$ is the current estimate of the value of the best policy on s' . Hence, the student's update rule is a convex combination of what is experienced so far, and the "best" action that the student can choose in the new state that has been reached.

³When $\gamma = 0$, the algorithm is the classic reinforcement learning.

The Q-values should help determine the behavior of the student. We assume that the student chooses his actions based on the Boltzmann distribution. That is, the probability that action a is chosen at state s , $P_s(a)$, is:

$$P_s(a) = \frac{e^{\frac{q(s,a)}{T}}}{\sum_{a' \in A} e^{\frac{q(s,a')}{T}}}.$$

T is called the *temperature*. Usually, T is initialized to a large number, in order to make the action choice more random, and is decreased "slowly".

Q-learning (under weak assumptions) has the following property (Watkins and Dayan (1992)):

Proposition 1 *Consider an MDP, in which the rewards at each state are unknown, and the transition functions are also unknown, then the Q-learning algorithm converges to the optimal policy of the MDP.*

2 Reinforcement Learning

Our interest is to learn how to behave in an unknown environment based on observed feedback. We saw in the previous section a classic learning algorithm - Q-learning. The following is an example for a reinforcement learning problem.

Example 1 *Consider an agent that learns to navigate. The environment is an MDP (there is a single decision maker), in which states are positions, actions are moves (together with directions) and the agent receive rewards per "good" or "bad" states.*

The following are central issues in Reinforcement learning:

- Modelling the environment
- Setting optimality criteria.
- Learn the model.
- Exploration vs. Exploitation
- Efficiency.

In this section we will assume that our environment is a stochastic game (SG) with two players - the agent and the adversary. In an SG, the players play a sequence (possibly infinite) sequence of (one-shot) games from some given set of games. At each stage (which is a game) the players choose an action, a reward is given to each player, and move to play a new game. The

new game is determined by a probability distribution over the joint actions (the distribution depends on the current game).

SG's are more general than MDP's and repeated games. A repeated game is an SG in with a single state. In MDP's the adversary has a single action at each state.

We make the following assumptions on the model. The agent knows the identity of the game played at every stage, but not the payoffs of the game and the the transition probabilities. After each stage game the agent observes his payoff and the actions played (his and the adversary's). In addition we assume the agent knows the maximal possible payoff R_{max} at the beginning of the SG.

A policy for an agent in an SG determines a probability distribution over the set of actions for each possible history, i.e. at each stage the policy tells the agent how to choose an action as function of the history observed.

Our goal: Attain near optimal guaranteed expected average reward "quickly" (Attain quickly almost the expected average reward an agent can guarantee itself against an arbitrary adversary behavior, when the agent knows the payoff matrices and the transition probabilities).

In the following subsection we present polynomial time algorithm which obtains the above goal.

2.1 R-max

In this section we present the R-max algorithm. ⁴ As mentioned above, the goal is to attain near optimal guaranteed expected average reward "quickly".

Let M be a stochastic game. We define the *value* of an agent's policy π , using the expected average reward, as follows. Let ρ be an adversary's policy. We denote by $U_M(s, \pi, \rho, T)$ the expected T-step average reward. Let $U_M(s, \pi, T) = \min_{\rho} U_M(s, \pi, \rho, T)$ denote the value that a policy can guarantee in T steps, given the initial state s . We denote by $U_M(s, \pi) = \liminf_{T \rightarrow \infty} U_M(s, \pi, T)$. The value of π is defined to be $U_M(\pi) = \min_{s \in S} U_M(s)$. ⁵

The R-max algorithm parameters are the SG M , the maximal possible reward R_{max} , the error bound ϵ , δ - the failure probability, and T, which is

⁴Based on the paper: R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. The lecture slides can be found in <http://mas.81n.org/r-max.ppt>.

⁵One could ask why not to try to achieve the maximal value over all possible states, or at least the value for the initial state. We claim that this is the only option, since an agent which has no prior knowledge, does not know which actions are "better" at the first stage, and therefore the agent may choose any action, and end at any state after the first stage.

the ϵ -return mixing time of an optimal policy. ⁶

The R-max algorithm.

Initialization: States - original states +1 fictitious state. All games-matrix entries are marked *unknown*. All joint actions lead to the fictitious state with probability 1. The agents payoff is R_{max} everywhere.

Repeat:

1. Compute an optimal T-step policy for the current state.
2. Execute current policy.
3. Update the model. After a joint action has been played, which corresponds to an *unknown* entry: (i) record payoff in the game-matrix; (ii) record the observed transition; and (iii) once enough transitions from this entry have been recorded (the state has been visited enough times), update the transition model based on the observed frequencies, and mark the entry *known*.

One can observe that the $R - max$ algorithm initializes an optimistic fictitious SG.

The following is the main result:

Theorem 2 *Let M be an SG with N states and k actions. Let $0 < \delta < 1$, and $\epsilon > 0$ be constants denoting error bounds. Denote the policies for M whose ϵ -return mixing time is T by $Opt(\prod_M(\epsilon, T))$. Then with probability of no less than $1 - \delta$, the R-max algorithm will attain an expected return of $Opt(\prod_M(\epsilon, T)) - 2\epsilon$ within a number of steps polynomial in $N, k, T, \frac{1}{\epsilon}$, and $\frac{1}{\delta}$.*

The proof makes use of the next couple of lemmas. First we give the following definition.

Definition 2 *Let M and \bar{M} be SG's over the same state and action spaces. We say that \bar{M} is an α -approximation of M if for every state s we have:*

1. *If $P_M(s, t, a, a')$ and $P_{\bar{M}}(s, t, a, a')$ are the probabilities of transition from state s to state t , given that the joint action carried out by the agent and the adversary is (a, a') , in M and \bar{M} respectively, then $P_M(s, t, a, a') - \alpha \leq P_{\bar{M}}(s, t, a, a') \leq P_M(s, t, a, a') + \alpha$.*
2. *For every state s , the same stage game is associated with s in \bar{M} and in M (and thus the rewards will be identical).*

The following is called the simulation Lemma:

⁶Roughly speaking, a policy π has a ϵ -return mixing time if for every $t \geq T$ $U_M(s, \pi, t) > U_M(\pi) - \epsilon$.

Lemma 1 *Let M and \bar{M} be SG's over N states, where \bar{M} is an $\frac{\epsilon}{NT_{R_{max}}}$ -approximation of M , then for every state s , agent policy π , and adversary policy ρ , we have that*

$$|U_M(s, \pi, \rho, T) - U_{\bar{M}}(s, \pi, \rho, T)| \leq \epsilon.$$

The following lemma, which is called the "implicit explore or exploit lemma", is the key behind the proof of the main theorem. We first need the following notation. Let M be a SG, and M_L the SG used in $R-max$, where L is the set of unknown states. $R^{M_L}-max$ denotes the optimal policy for the induced SG M_L .

Lemma 2 *Let M be an SG. Let L and M_L be as above. Let ρ be an arbitrary policy for the adversary, let s be some state, and $0 < \alpha < 1$. Then either (1) $V_{R-max} > OPT_M(\prod \epsilon, T) - \alpha$, where V_{R-max} is the expected T -step average reward for the $R^{M_L}-max$ policy on M ; or (2) An unknown entry will be played in the course of running $R-max$ on M for T steps with a probability of at least $\frac{\alpha}{R_{max}}$.*

The intuition behind Lemma 2 is the following: The $R-max$ algorithm either explores efficiently or exploits efficiently. The adversary can prevent the agent to explore or exploit; However, it cannot prevent the agent from doing both. Hence, the agent either exploits or explores. The lemma provides that either one of the two is done by the agent, efficiently. The proof will be given in the next lecture.