Reinforcement Learning **Spring Semester, 2018/9**

Lecture 1: February 27, 2019

Lecturer: Yishay Mansour *Scribe: ym*

1.1 Course Administration

1.1.1 classes

- Lectures will be held Wednesdays 10-13 in Dan David 001.
- Recitations will be held Wednesdays 14-15 and 15-16 in Melamed (006).

1.1.2 resources

The course web site is `http://rl-tau-2019.wikidot.com/`. The web site will include the lecture slides and information about recitations.

1.1.3 Prerequisites

Introduction to Machine Learning (or an equivalent course).

1.1.4 requirements

The final grade would be composed from the following:

20% Homework (both theory and programming).

20% Final project: deep Reinforcement Learning on Atari game.

60% Final exam.

As usual, a student need to pass the final exam to pass the course.

1.2 Motivation for RL

In the recent years there is a renew interest in Reinforcement learning. The new interest is grounded in the emerging applications of Reinforcement learning.

Over the years reinforcement learning has proven to be highly successful for playing board games over a long history. Gerald Tesauro in 1992 developed the TD-gammon, which uses a two layer neural-network to achieve a high performance backgammon computer game. The network was trained by bootstrapping it from scratch and learned a temporal differences function. One of the amazing features of the TD-gammon was that even in the *first* move, it used a different game move than the grandmasters, who later adopted the new game move once they observe TD-gammon playing it.

Recently in 2015-7, Deep Mind have developed a deep neural-network to plat Go, which is able to beat the best Go players in the world. Developing a human-level computer Go program has been a challenge that has persisted for decades.

Early in 1962, Artur Samuel developed a checkers game, which was at the level of the best human. His original framework included many of the ingredients which latter contributed to Reinforcement learning, as well as search heuristic for large domains.

To complete the picture of computer board games, we should mention Deep Blue, from 1996, which was able to beat the world champion Kasparov. This program was mainly built on a heuristic search, and developed a new simple hardware to support it. Recently, Deep Mind came out with a deep neural-network that matched the best chess programs (which are already much better than any human players).

Another domain, popularized by Deep Mind, is playing Atari video games, which where popular in the 1980's. Deep Mind were able to show how deep neural networks can achieve human level performance, using only the video picture as input (and having no additional information about the goal of the game).

Looking in to the future, the real promise of Reinforcement learning is learning in interactive settings. Probably one the most important applications is robotics, where reaching a human level performance would have far outreaching consequences.

Origins of reinforcement learning: Reinforcement learning spans a large number of disciplines. Naturally, we are going to look through the lens of Computer Science and Machine Learning. In Engineering, there are many disciplines related to optimal control. Other notable origins are in Operation Research, where the initial mathematical works have originated. Addition disciplines include: Neuroscience, Psychology and Economics.

Mathematical Model The main mathematical model we will use is Markov Decision Process (MDP). The model tries to capture uncertainty in the dynamics of the environment, the actions and our knowledge. The main focus would be on decision making, namely, selecting actions. The evaluation would consider the long term effect of the actions, trading-off immediate rewards with long-term gains.

In contrast to Machine Learning, the reinforcement learning model would have a *state*, and the algorithm will influence the state through its actions. The algorithm would be faced with an inherent tradeoff between exploitation (getting the most reward given the current information) and exploration (gathering more information about the environment).

1.3 Markov Decision Process (MDP)

Our main formal model would be a Markov Decision Process (MDP) will be composed from:

- S: a finite set of states.
- $s_0 \in S$: is the start state.
- A: a finite set of actions.
- $\delta : S \times A \to \Delta(S)$: a stochastic transition probability function, where $\Delta(S)$ is the set of probability distributions over S. We denote by $\delta(\cdot|s, a)$ the distribution of next state, when doing action a in state s.
- R: an immediate reward. In general R would depend on the current state s and action a, and in general it can be a random variable. In most cases we will focus on the expectation of $R(s, a)$. We will assume that the immediate reward R is bounded, specifically, that it is always in the range $[0, 1]$.

A run of the MDP is characterized by a *trajectory*, which has quadruples (s_t, a_t, r_r, s_{t+1}), where s_t is the state at time t, a_t is the action performed at time t in s_t, r_t is the immediate reward, and s_{t+1} is the next state. Clearly, r_t is distributed according to $R(s_t, a_t)$ and s_{t+1} is distributed according to $\delta(\cdot|s_t, a_t)$.

Return function We like to combine the immediate rewards to a single value, which we call *return*, that we will optimize. The decision to reduce all the immediate rewards to a single value is already a major modeling decision. When defining the return we will consider whether earlier immediate rewards are more important than later rewards. Also, there is an issue whether the system is *terminating*, namely terminates after a finite number of steps, or is *continuous*, which implies that it runs forever. Usually, the return would be linear in the immediate rewards, and we will be interested in maximizing expected return. For that reason, we will be mostly interested in the expectation of the immediate rewards.

Popular return functions include:

1. *Finite horizon:* There is a parameter H and the return is the sum of the first H rewards, i.e., $\sum_{t=1}^{H} r_t$.

2. *Infinite discounted return:* There is a parameter $\gamma \in (0,1)$ and the discounted return is $\sum_{t=0}^{\infty} \gamma^t r_t$. Note that since $r_t \in [0,1]$, the return is bounded by $1/(1-\gamma)$.

3. *Average Reward:* where we take the limit of the average immediate reward. Specifically, $\lim_{T\to\infty} \frac{1}{T} E[\sum_{t=1}^{T} r_t]$.

4. *Sum of the rewards:* this applies only to the case of terminating MDP, since otherwise it might be infinite.

Example of an MDP: Consider an inventory control problem. At day t we have $x_t \geq 0$ remaining items from previous days. We order $a_t \geq 0$ new items, which is our action. We have a demand of $d_t \geq 0$, the amount consumer want to buy at day t. We have $s_t = \min(d_t, x_t + a_t)$ items bought. For day $t+1$ we have x_{t+1} remaining items, i.e., $x_{t+1} = \max(0, x_t + a_t - d_t)$.

We can now formalize the immediate reward, based on P, the profit per item, $J(\cdot)$ the cost to order, and $C(\cdot)$ the cost of inventory. Our immediate reward is,

$$R(x_t, a_t) = P s_t + J(a_t) - C(x_{t+1})$$

Our goal can be to maximize a discounted return function over the immediate rewards, where the discounting represents the interest rate.

Action selection: We assume that the state of the system is "observable", namely, we know in which state we are. Our goal would be to select action as to maximize the expected return. For the remainder of the lecture we focus on the discounted return.

Let a policy be a mapping from states to actions. Due to the Markov property of the system, we can show that the optimal history-dependent strategy is a deterministic policy, namely, it does not depend on the history and selects at each state a single action.

Theorem 1.1 *There exists a deterministic policy which maximizes the discounted return.*

Note that the optimal policy does not depend on the start state!

Multi-Arm Bandits (MAB): A simple well-studied variant of the MDP model are MABs, which are essentially an MDP with a single state. Given the model it is clear that the optimal policy would select the action with the highest immediate reward. However, when we need to learn the stochastic rewards, we have a tradeoff between using the action with the highest observed immediate reward versus trying new actions, and getting a better approximation of their expectation.

1.4 Planning

Planning problems capture the case that we are given a complete model of the MDP and would like to perform some task. Two popular tasks are the following:

- **Policy evaluation**: Given a policy π evaluate its expected return.
- **Optimal control**: Compute an optimal policy π^*. For the infinite discounted return we have that π^* maximizes the return from any start state.

We start with policy evaluation. An important ingredients in the planning process would be the following two value functions, which depend on the policy π.

- $V^\pi(s)$, which is the expected return of π starting from state s.
- $Q^\pi(s,a)$, which is the expected return of π starting from state s doing action a and then following π.

We denote by $V^*(s)$ and $Q^*(s,a)$ the value function and the Q-value function, respectively, of the optimal policy π^*. For the optimal policy we have,

$$\forall s \in S \quad V^*(s) = \max_\pi V^\pi(s)$$

Namely, the optimal policy is optimal from any start state.

Policy Evaluation We can now solve the policy evaluation problem for the discounted return. We can write the following identities.

$$\forall s \in S: \quad V^\pi(s) = E_{s' \sim \delta(s,\pi(s))}[R(s,\pi(s)) + \gamma V^\pi(s')]$$

This gives a system of linear equations where the unknowns are $V^\pi(s)$. We have $|S|$ equations and $|S|$ unknowns, so there exists some solution.

Optimal Control: We can write the identity for the optimal Q-function.

$$Q^\pi(s,a) = E_{s' \sim \delta(s,\pi(s))}[R(s,a) + \gamma V^\pi(s')]$$

Note that for a deterministic policy π we have $V^\pi(s) = Q^\pi(s,\pi(s))$.

The following theorem gives a characterization of the optimal policy (also known as Bellman Eq).

Theorem 1.2 *A policy π is optimal if and only if at each state s we have*

$$V^\pi(s) = \max_a \{Q^\pi(s,a)\}$$

Proof: We will show here only the *only if* part (the other direction will be give later in the course). Assume that there is a state s and action a such that

$$V^{\pi}(s) < Q^{\pi}(s, a)$$

The strategy of performing in s action a and then using π outperforms π, and so π is not optimal.

The improvement would be valid for each visit of s, so the policy that performs action a in s improves over π (and is also a policy, a mapping from states to actions). □

Computing the optimal policy: There are three popular algorithms to compute the optimal policy given an MDP model.

- *Linear Program*
- *Value Iteration*: at iteration t compute

  $$V_{t+1}(s) = E_{s' \sim \delta a, \pi(s)}[R(s, \pi(s)) + \gamma V_t(s')]$$

  We will show that at each iteration the distance from the optimal value function V^* is decreased by $1 - \gamma$, namely $\|V_{t+1} - V^*\|_{\infty} \leq (1 - \gamma)\|V_t - V^*\|_{\infty}$.
- *Policy Iteration*:

  $$\pi_{t+1}(s) = \arg\max_{a}\{Q^{\pi_t}(s, a)\}.$$

  We will show that the number of iterations of policy iteration is bounded by that of value iteration, but each iteration is computationally more intensive.

1.5 Learning Algorithms

We now would like to address the case that the MDP model is unknown. We still have the two primary tasks: (1) policy evaluation, and (2) optimal control.

We will use two different approaches. The *model based* approach, first learns a model and then uses it. The *model free* approach learns directly a policy.

1.5.1 Model Based Learning

The basic idea is very intuitive. We like to estimate the model from observations. Given a trajectory, we can decompose it to quadruplets $(s_t, a_t, r_t, s_{t_{+1}})$. We can learn both the immediate rewards $R(s, a)$ and the transition function $\delta(s, a)$ from those quadruplets.

Building the observed model (off-policy): Given (s_t, a_t, r_t, s_{t+1}) we define the observed model as follows. Let $\#(s,a) = \sum_{t=1}^{T} I(s_t = s, a_t = a)$, where $I(\cdot)$ is the indicator function. The observed reward would be

$$\hat{R}(s,a) = \sum_{t=1}^{T} r_t \frac{I(s_t = s, a_t = a)}{\#(s,a)}.$$

The observed next state distribution would be:

$$\hat{\delta}(s'|s,a) = \frac{\sum_{t=1}^{T} I(s_t = s, a_t = a, s_{t+1} = s')}{\#(s,a)}.$$

Given an observed model, we can compute an optimal policy for the observed model. The following intuitive claim can be (and would be) made formal (later in the course).

Claim 1.3 *If the observed model is "accurate" then the optimal policy for the observed model is a near optimal policy for the true model.*

There is a hidden assumption that we have enough samples for each state s and action a. An important question is how many samples we need for each (s,a) to get an accurate observed model.

Building the observed model (on-policy): In an on-policy the learner can control the actions. How can it use this ability to accelerate the learning. The simple idea is to try to visit states which we have not visit sufficiently, which will help us to build an accurate observed model.

A basic idea is to split the states to two parts. Well observed states, from which we have sufficient samples for each action. Relatively unknown states, from which we have not sampled sufficiently. The idea is that from the well sampled states, we have a good model. Now we can "imagine" that the immediate reward in the relatively unknown states is maximal, while the immediate rewards in the well observed states is zero. We will also assume that once we get to a relatively unknown state we stay in such a state. Given such a model, we can solve for the planning problem. The optimal policy for this (imaginary) model will find a shortest path to the relatively unknown states, giving us an additional observation for those states. Eventually, states would move from the relatively unknown states to the well known states. Once the set of relatively unknown states is empty, we are done with the learning phase.

Monte-Carlo Methods For those methods it is easiest to think of a terminating MDP which works in episodes. The Monte Carlo algorithm runs an episode using the policy. Given the trajectories we like to build an observed model. However, there might be statistical issues. Unlike the continuous trajectory, now only the first arrival to a state is an independent sample! Additional visits to the state might be correlated with previous outcomes!

1.5.2 Model free learning

Model free learning algorithm try to learn directly the value function or the policy, and circumvent learning the model.

There is a variety of model free learning algorithms. They differ by the value function they estimate, Q or V, and whether they are off-policy or on-policy. The main challenge is to analyze their dynamics and guarantee their convergence.

1.5.3 Q-learning: off-policy

The Q-learning algorithm estimates directly the optimal learning function. It receives as input a long trajectory and outputs an estimate for the Q function.

The idea is to focus on the difference between the current estimate and the observed values. Given the time t quadruple (s_t, a_t, r_t, s_{t+1}), we define

$$\Delta_t = Q_t(s_t, a_t) - r_t - \gamma \max_a Q_t(s_{t+1}, a)$$

We have a learning rate $\alpha_t(s, a)$ which may depend on the number of times, until time t, we executed (s, a). We now update our estimate to Q^{t+1} as follows,

$$Q_{t+1}(s_t, a_t) = Q_{t+1}(s_t, a_t) - \alpha(s_t, a_t)\Delta_t$$

Note that once $Q_t = Q^*$ then $E[\Delta_t] = 0$. The challenge in the analysis is to study the dynamics of the stochastic process and show the convergence.

1.5.4 Temporal Differences

The Temporal Differences (TD) computes an estimate to the V value function of the current policy. The error at time t is

$$\Delta_t = V_t(s_t) - r - V_t(s_{t+1})$$

and, using the learning rate $\alpha_t(s_t, a_t)$ we update

$$V_{t+1}(s_t) = V_t(s_t) - \alpha_t(s_t, a_t)\Delta_t$$

The above is called $TD(0)$, which focuses on the last transition. In general we can add a parameter λ which defines $TD(\lambda)$. The parameter allows us to update the current value also using recent observed rewards. The λ can be viewed as a discounting, which is used for the update (and not the return!). The eligibility of a state counts how many times a state is visited (discounted by λ). The eligibility trace of a state s is

$$e_t(s) = \sum_{i=1}^{t} (\lambda\gamma)^{t-i} I(s_{t-i} = s) = (\lambda\gamma)e_{t-1}(s) + I(s_t = s)$$

Given the eligibility trace the new estimated value, in *every* state s, is

$$V_{t+1}(s) = V_t(s) - \alpha_t(s_t, a_t)\Delta_t e_t(s)$$

The idea is that we propagate the rewards faster to recently visited states.

1.6 Large state MDP

In the previous learning algorithms we assumed that the number of states is sufficiently small, since we build lookup table index by states. However, in many applications the number of states is exponential in the number of natural parameters. Overcoming this challenge can be done in many ways, here are a few popular ones.

1. *Restricted value function:* the main idea is to use a value function from a limited class. Two extreme solutions are linear functions and deep neural networks. This is similar to supervised learning, where we learn using a given function class. Especially popular are Deep Q-Networks (DQN) which learn the Q values.

2. *Restricted policy class:* We fix a policy class $\Pi = \{\pi : S \to A\}$. The main challenge is given $\pi \in \Pi$ to estimate V^π or Q^π. Given the estimate of Q^π we can improve the policy by computing a greedy policy $\pi' = \arg\max Q^\pi$. The quality clearly depends on the approximation of both the Q^π and the ability to fit $\pi' = \arg\max Q^\pi$ to Π.

   One challenge is how to compute the gradient of a policy, to update its parameters. The challenge is that the update influences not only the action probabilities, but also the distribution of the states.

3. *Restricted MDP model:* We can make assumptions about the MDP structure, for example, MAB assumes a single state.

4. *Generative model:* We are given an implicit representation of the MDP using a generative model. The model, given (s, a) return (r, s'), appropriately distributed.

1.7 Course Schedule

- Part 1: MDP basics and planning
- Part 2: MDP learning Model Based and Model free
- Part 3: Large state MDP Policy Gradient, Deep Q-Network
- Part 4: Special MDPs Bandits and Partially Observable MDP
- Part 5: Advanced topics