ECE 586 – Markov Decision Processes and Reinforcement Learning

Introduction

Instructor: Dimitrios Katselis *Acknowledgment: R. Srikant's Notes and Discussions*

1 Reinforcement Learning

Reinforcement learning is a branch of artificial intelligence that formalizes a trial-and-error philosophy of learning. More precisely, an agent interacts over time with a stochastic environment and learns through this interaction. At every time instant, the agent observes the current state of the world and selects an action. Based on the selected action, the agent receives a reward from the environment (or alternatively pays a cost). The RL problem is then to design an agent that selects actions according to some optimization criteria. Since the trial-and-error approach is driven by the received rewards, possible optimization criteria correspond to maximizing some form of expected long-term reward (or minimizing some form of expected long-term cost). In other words, RL corresponds to a disciplined approach for **sequential decision making**. From the previous trial-and-error description, one may conclude that RL algorithms are motivated by natural processes behind human decision making, in the sense that rewards or costs provide positive or negative reinforcement for particular actions. Any algorithm for finding an optimal agent is a possible solution, or RL algorithm, to the RL problem.

Various classes of methods for sequential decision making to maximize the sum of received rewards exist. In increasing sophistication level, a first class corresponds to the **multi-armed bandits**, which apply to instances where there is no contextual information available to guide the decision making. A second class corresponds to the **contextual bandits**, where contextual information is taken into account. Moreover, there are RL algorithms that can handle more general setups than the aforementioned classes.

We finally note that when designing an RL algorithm, computation time considerations in attaining the optimal agent are crucial. More explicitly, an RL algorithm that can find a solution in less time than a different algorithm is more efficient. RL problems are typically organized in repeated **episodes** either until a solution is found or timeout occurs. Therefore, more efficient algorithms require less episodes to yield a solution. We will consider algorithmic efficiency in the described sense in subsequent lectures.

2 Differences of RL from other Machine Learning methods

RL has certain differences from other optimization paradigms. First, RL is **unsupervised**. In other words, there are no labeled data pointing to best actions. The decision making is driven by a reward signal, which reinforces certain actions more than others. Also, **action sequencing** is very important in RL, since it determines the realized trajectory of the agent. Another major characteristic of the RL paradigm is that an action taken at a particular time instant may heavily impact the reward signal after many subsequent steps. This corresponds to what is called **delayed**

feedback. Finally, the actions of the agent affect the realized trajectory and by extension the observations made.

3 Connection to Control Theory

Control theory studies dynamical systems and their optimization over time. Often, certain system variables can be unknown or imperfectly observed and the system evolution may be deterministic or stochastic. Focusing on discrete time, the system state $x_k \in \mathcal{X}$ of a deterministic dynamical system obeys the following **plant equation** or **law of motion**:

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1. \tag{1}$$

Here, k corresponds to the time index and

- $u_k \in \mathcal{U}$ is the control or decision variable or action selected at time k,
- N is the time or planning horizon corresponding to how many decisions are made or actions are taken,
- f_k is a function describing the state evolution (can be time-invariant).

At time k, the control action is constrained to $\mathcal{U}_k(x_k) \subset \mathcal{U}$, where $\mathcal{U}_k(x_k) \neq \emptyset$. We consider the class of **admissible** control policies (or **control laws**)

$$\mu = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}, \tag{2}$$

where $u_k = \mu_k(x_k) \in \mathcal{U}_k(x_k)$. Given an initial state x_0 and an admissible policy $\mu = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$, (1) becomes:

$$x_{k+1} = f_k(x_k, \mu_k(x_k)), \qquad k = 0, 1, \ldots, N-1. \tag{3}$$

In addition to the plant equation, a principal feature of a control problem is a **cost function** that is additive over time (**separable cost**). Assuming that the admissible policy μ is applied and the cost incurred at time k is $c_k(x_k, u_k)$, the total cost for the considered horizon and for a fixed initial state x_0 is

$$J_N^\mu(x_0) = c_N(x_N) + \sum_{k=0}^{N-1} c_k(x_k, u_k),$$

where $c_N(x_N)$ is the terminal cost. Our goal is then to solve the following problem:

$$\min_{\mu \in \mathcal{M}} J_N^\mu(x_0). \tag{4}$$

Here, $\mathcal{M}$ is the set of all admissible control laws. The optimal policy is denoted by μ^*. Alternatively, one may consider the problem of optimally selecting the sequence of control variables $u_0, u_1, \ldots, u_{N-1}$ in a such a way that the total cost $c_N(x_N) + \sum_{k=0}^{N-1} c_k(x_k, u_k)$ is minimized. We note here a subtle distinction: Although (4) corresponds to an optimization over the Cartesian product of some function spaces, formalizing this problem as a minimization over the control variables $u_0, u_1, \ldots, u_{N-1}$ corresponds to a conventional optimization formulation.

Optimality Principle: *By choosing a sequence of controls, we generate a trajectory. Consider any point on an optimal trajectory. Then, the remaining trajectory is optimal for the problem initiated at that point. More formally, consider an optimal control policy* $\mu^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ *and assume that we are in state* x_k *at time* k*. Suppose we wish to minimize the* ***cost-to-go*** *from time* k *to time* N*, namely* $c_N(x_N) + \sum_{r=k}^{N-1} c_r(x_r, u_r)$*. Then, the subpolicy* $\{\mu_k^*, \ldots, \mu_{N-1}^*\}$ *is optimal for this subproblem.*

Formalizing the optimality principle, we are led to the **Dynamic Programming (DP) algorithm**:

Consider any initial state x_0*. Define the backward recursion*

$$J_N(x_N) = c_N(x_N) \tag{5}$$

$$J_k(x_k) = \inf_{u_k \in \mathcal{U}_k(x_k)} \{c_k(x_k, u_k) + J_{k+1}(f_k(x_k, u_k))\}, \quad k = N-1, N-2, \ldots, 0. \tag{6}$$

Then, the optimal cost $J_N^{\mu^*}(x_0)$ *of problem (4) corresponds to* $J_0(x_0)$ *of the above recursion. Moreover, if* $u_k^* = \mu_k^*(x_k)$ *minimizes the right-hand side of (6) for every* x_k *and* k*, then an optimal policy is* $\mu^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$*.*

Eq. (6) is often called **optimality equation** or **DP equation** or **Bellman equation**.

We now note that a discrete-time stochastic control system is characterized by a plant equation of the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1, \tag{7}$$

where w_k is a noisy variable or random disturbance. Suppose that w_k is drawn from $P_k(\cdot|x_k, u_k)$, which depends explicitly on x_k, u_k but not on $w_0, \ldots, w_{k-1}$. To accommodate any source of randomness we may allow in the problem, the optimal control problem is defined on the basis of optimizing the expected cost

$$J_N^{\mu}(x_0) = E\left[c_N(x_N) + \sum_{k=0}^{N-1} c_k(x_k, u_k, w_k) \middle| x_0\right], \tag{8}$$

where the expectation is with respect to the joint distribution of all the involved random variables given x_0. In this case, the optimality principle and the DP recursion are still valid, by taking the expectation into account when it is needed. Bellman's equation in this case becomes:

$$J_k(x_k) = \inf_{u_k \in \mathcal{U}_k(x_k)} \{E\left[c_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) | x_k, u_k\right]\}, \quad k \leq N-1. \tag{9}$$

Difference from RL: $\{f_k\}$ is a known function sequence. In other words, the state transitions of the system can be accurately modeled.

Note: Although a μ^* for both deterministic and stochastic systems is tied to x_0, it is often the case that such a μ^* is optimal for all initial states x_0.

Stochastic optimal control problems of interest to us are those that can be modeled as Markov Decision Processes with finite state and action spaces.

4 Markov Decision Processes

To rigorously formulate RL problems, the framework of **Markov Decision Processes (MDP)** is used. MDPs correspond to the stochastic decision making models underlying RL problems. In RL, the underlying model is either unknown or rather complicated (e.g., too large) to solve in order to find an optimal policy. In MDPs, the system state summarizes all the past information. Selecting the system state is therefore a key component of the modeling process. The aforementioned summary of past information is encapsulated in the **Markov property**, which corresponds to the conditional independence of the future from the past given the current state.

To make the discussion a bit more precise, an MDP is a controlled process characterized by Markovian dynamics and a separable cost function. Focusing on the basic case of finite state finite horizon MDPs, the key ingredients are:

- $\mathcal{X}$ is the state space with finite cardinality X, i.e., $\mathcal{X} = \{1, 2, \ldots, X\}$,
- x_k is the state of the controlled Markov chain at time $k = 0, 1, 2, \ldots, N$,
- $\mathcal{U}$ is the control or action or decision space with finite cardinality U, i.e., $\mathcal{U} = \{1, 2, \ldots, U\}$,
- $u_k \in \mathcal{U}$ or $u_k \in \mathcal{U}(x_k) \subset \mathcal{U}$ or $u_k \in \mathcal{U}_k(x_k) \subset \mathcal{U}$ is the action taken at time k,
- $P_k(u)$ is a $X \times X$ transition probability matrix with elements

$$P_{k,ij}(u) = P(x_{k+1} = j | x_k = i, u_k = u), \quad \text{for} \quad i, j \in \mathcal{X} \text{ and } u \in \mathcal{U}.$$

  In this course, we will focus on time-homogeneous MDPs, and therefore the corresponding transition probabilities will be time-invariant (denoted by $P_{ij}(u)$). The corresponding transition matrix for each action u will be denoted by $P(u)$. We also note here that $P = [P(u)]$ is a third-order tensor, while if the transition probabilities are time-varying, then $P = [P_k(u)]$ is a fourth-order tensor.

- $c_k(i, u)$ or $c_k(i, u, j)$ is the one-stage cost when the system is at state $i \in \mathcal{X}$ at time k, the action $u \in \mathcal{U}$ is taken by the agent and the system transitions to state $x_{k+1} = j$,
- $c_N(i)$ is the terminal cost for each state $i \in \mathcal{X}$,
- ρ is the initial state distribution such that $x_0 \sim \rho$.

Note: In the context of MDPs, a separable cost or reward function is optimized. When a cost function is optimized, we end up with a minimization problem defined on the basis of one-stage costs $c_k(i, u)$ or $c_k(i, u, j)$. When a reward function is optimized, we end up with a maximization problem with one-stage rewards $r_k(i, u)$ or $r_k(i, u, j)$. In both cases, $x_k = i, u_k = u$ and $x_{k+1} = j$. In this course, we will mostly assume that the one-stage cost or reward signals and the transition probabilities $P_{k,ij}(u)$ are time-invariant (except for possible discounting factors for the cost or reward signals) and we will drop the temporal index k.

Thus, an MDP model (in full generality) is the 6-tuple

$$(\mathcal{X}, \mathcal{U}, P_{k,ij}(u), c_k(i, u), c_N(i), \rho) \quad i, j \in \mathcal{X}, \quad u \in \mathcal{U}, \quad k = 0, 1, \ldots, N-1.$$

Moreover, in the context of MDPs we require perfect state information. In other words, the state x_k is fully observable at time k when the agent chooses the action u_k. Therefore, at time k the following information is available to the decision-maker (agent or controller):

$$\mathcal{F}_k = \{x_0, u_0, x_1, u_1, \ldots, x_{k-1}, u_{k-1}, x_k\}.$$

The behavior of the agent is modeled by a **policy**. At time k, the agent uses all the available information in $\mathcal{F}_k$ to decide the action u_k. Mathematically, this is described by the mapping $u_k = \mu_k(\mathcal{F}_k)$, where $\mu_k : \mathcal{F}_k \to \mathcal{U}$ (or $\mu_k : \mathcal{F}_k \to \mathcal{U}_k(x_k)$) is the policy or strategy or decision rule at time k. As in the context of control theory, a policy μ corresponds to the sequence of decision rules used by the agent up to time $N-1$:

$$\mu = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}.$$

Episodic scenario: In the finite time horizon scenario that we have considered so far, the planning window is up to time N, where N is deterministic. In stochastic optimal control and in RL, N can be a **stopping time** (random variable). Although $N = \infty$ is a valid choice and corresponds to infinite horizon problems, the most usual assumption in practice is that

$$N < \infty \quad \text{almost surely (a.s.)},$$

if the underlying problem is expected to have an end in finite, but otherwise random time N. With this assumption, the aggregate trajectory $x_0, u_0, c_0, x_1, u_1, c_1, \ldots, x_{N-1}, u_{N-1}, c_{N-1}, x_N$ is called **episode**. In plain words, episode is the simulation length from an initial state to some terminal state. For example, when playing chess an episode can be considered the entire game, which ends if the player wins, loses or achieves a draw. Since it is unknown beforehand when the terminal state will be reached, the planning horizon N is a random variable.

Goal: Obtain an optimal policy μ^* by solving

$$\mu^* = \arg\min_{\mu} J_N^{\mu}(x_0), \tag{10}$$

where $J_N^{\mu}(x_0)$ is given by $J_N^{\mu}(x_0) = E\left[c_N(x_N) + \sum_{k=0}^{N-1} c_k(x_k, u_k) \middle| x_0\right]$.

Note: If $\mathcal{X}$ and $\mathcal{U}$ are finite, then the minimum always exists. Also, μ^* may not be unique. For instance, if $c_k(x, u)$ is a constant for all pairs (x, u), then all policies are optimal.

As mentioned earlier, an MDP is an instance of a stochastic control problem. Since at time k, $u_k = \mu_k(\mathcal{F}_k)$, this formalizes the notion of closed loop control in such problems. In contrast to open loop optimization, here u_k affects x_l, u_l and c_l for any $l > k$. In open loop problems, the control variables $u_0, u_1, \ldots, u_{N-1}$ are all determined at time 0.

Finally, (stochastic) DP programming provides the optimal solution to an MDP. We repeat Bellman's algorithm here using the introduced notation for clarity (assuming deterministic costs):

For any state $i \in \mathcal{X}$, let $J_N(i) = c_N(i)$. Define the backward recursion

$$J_k(i) = \min_{u \in \mathcal{U}} \{c_k(i,u) + \sum_{j \in \mathcal{X}} P_{k,ij}(u) J_{k+1}(j)\} \tag{11}$$

$$\mu_k^*(i) = \arg\min_{u \in \mathcal{U}} \{c_k(i,u) + \sum_{j \in \mathcal{X}} P_{k,ij}(u) J_{k+1}(j)\}, \quad k = N-1, N-2, \ldots, 0. \tag{12}$$

Then, for any initial state x_0, the optimal cost $J_N^{\mu^*}(x_0)$ of problem (10) corresponds to $J_0(x_0)$ of the above recursion and an optimal policy corresponds to $\mu^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$.

The above algorithm has been presented in full generality (time-varying costs and and transition probabilities) to demonstrate the power of DP.

Difference from RL: Consider the time-invariant case. In the RL framework, the transition probabilities $P_{ij}(u)$ are unknown.

Note: In the context of stochastic adaptive control, there are methods which estimate the unknown transition probabilities and update the control policy at the same time. Alternatively, algorithms that do not require explicit knowledge of these probabilities in determining an optimal policy may be employed. This second class of methods, usually called **simulation-based methods**, correspond to the main body of the RL literature. Through the trial-and-error methodology and the reinforcement of actions with higher rewards or lower costs, RL algorithms determine the policy by simulating a trajectory through parts of the state space, which are more probable. In other words, no effort is wasted in low-probability parts of the state space.

Possible Policies: The most general form of policy μ corresponds to randomized mappings of the form $u_k \sim \mu_k(\mathcal{F}_k)$, where μ_k is a probability distribution. Therefore, u_k is a randomized action dependent on all past history. Clearly, this set of policies contains all possible policies of the form $u_k = \mu_k(\mathcal{F}_k)$, where μ_k is a deterministic mapping. We denote this set as $\mathcal{GP}$ and we refer to such strategies as **general policies**. A subset of $\mathcal{GP}$ corresponds to randomized policies of the form $u_k \sim \mu_k(x_k)$ (i.e., again μ_k is a probability distribution). We denote this set as $\mathcal{RMP}$ and we refer to such strategies as **randomized Markovian policies**. Clearly, this set of policies contains all possible policies of the form $u_k = \mu_k(x_k)$, where μ_k is a deterministic mapping. We denote this set as $\mathcal{DMP}$ and we refer to such strategies as **deterministic Markovian policies**. Therefore, to summarize,

$$\mathcal{DMP} \subset \mathcal{RMP} \subset \mathcal{GP}.$$

It turns out that a solution μ^* of problem (10) lies always in $\mathcal{DMP}$. To see this fact, we note that DP programming gives an explicit construction of an optimal policy μ^* and this policy lies in $\mathcal{DMP}$ by construction. This implies that the expectation in the definition of $J_N^{\mu}(x_0)$ for problem (10) is only with respect to $\{x_1, x_2, \ldots, x_N\}$ for a given x_0.

5 Bandit Processes and the Multi-Armed Bandit Problem

Multi-armed bandit (MAB) problems are a class of sequential resource allocation problems concerned with allocating one or more resources among several alternative (competing) projects.

Bandit Process: A bandit process (single-armed) is described by a machine or an arm or a project and is characterized by the trajectory $\{x_0, r_0, x_1, r_1, \ldots, x_n, r_n, \ldots\}$, where x_n is the state of the machine after it has been operated n times and r_n is the reward provided by the nth usage of the machine. In its basic setup, a bandit process is a special type of an MDP with a binary action space $\mathcal{U} = \{0, 1\}$, where the control action $u = 0$ can be thought as "freeze" and the control action $u = 1$ can be thought as "continue". For $u_k = 1$, the reward $r_k = r(x_k)$ is obtained and the system transitions to state x_{k+1} according to $x_{k+1} \sim P(\cdot|x_k, u_k = 1)$. In full generality, the state transitions can instead of a transition matrix be modeled as $x_{k+1} = f_k(x_0, x_1, \ldots, x_k, w_k)$, where $f_k(\cdot)$ is known and $\{w_k\}_{k\geq 0}$ is a sequence of independent random variables that are also independent of x_0 and have a known probabilistic description. Therefore, in the general setup, a bandit process is not necessarily a Markov process.

The Multi-Armed Bandit problem (classical setup): A multi-armed bandit (MAB) process (m-armed) is a family of m independent single-armed bandit processes. The system has a single controller. At every time instant the controller chooses to operate a **single** machine, while all other machines are frozen. Assuming that $l = 1, 2, \ldots, m$ is the machine index, each machine is characterized by the trajectory

$$\{x_l(n_l(0)), r_l(x_l(n_l(0))), x_l(n_l(1)), r_l(x_l(n_l(1))), \ldots, x_l(n_l(t)), r_l(x_l(n_l(t))), \ldots\}_{t=0,1,2,\ldots}, \;\; l = 1, \ldots, m,$$

where $n_l(t)$ denotes the number of times machine l has been chosen. The action space $\mathcal{U}$ corresponds to vectors $u = (u_1(t), u_2(t), \ldots, u_m(t))$, which are the row vectors of an $m \times m$ identity matrix (natural basis):

$$\mathcal{U} = \{(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, 0, \ldots, 1)\}.$$

This is due to the fact that a single machine is operated at every time instant, while the rest remain frozen. The system dynamics are described by:

$$x_l(n_l(t+1)) = \begin{cases} f_{n_l(t)}(x_l(0), \ldots, x_l(n_l(t)), w_l(n_l(t))), & \text{if } u_l(t) = 1 \\ x_l(n_l(t)), & \text{if } u_l(t) = 0 \end{cases}, \quad l = 1, \ldots, m \tag{13}$$

and

$$n_l(t+1) = \begin{cases} n_l(t) + 1, & \text{if } u_l(t) = 1 \\ n_l(t), & \text{if } u_l(t) = 0 \end{cases}, \quad l = 1, \ldots, m. \tag{14}$$

As before, $\{w_l(t)\}_{l=1,t\geq 0}^{m}$ is a collection of independent random variables, which are also independent from $\{x_1(0), \ldots, x_m(0)\}$ and their probabilistic description is known. We note here that (13) is general enough to capture the basic scenario with Markovian dynamics by appropriately specifying $\{f_{n_l(t)}\}_{l=1}^{m}$. Additionally, each machine produces a reward only when it is operated:

$$r_l(t) = \begin{cases} r_l(x_l(n_l(t))), & \text{if } u_l(t) = 1 \\ 0, & \text{if } u_l(t) = 0 \end{cases}.$$

Goal: Consider the basic scenario with Markovian dynamics. The objective is to solve the following **infinite horizon discounted reward** problem:

$$\max_{\mu} E\left[\sum_{t=0}^{\infty} a^t \sum_{l=1}^{m} r_l(t) \middle| x_1(0), \ldots, x_m(0)\right], \tag{15}$$

over the set of all realizable policies, i.e., policies which at every time instant use only causal information. For consistency with the previous notation, we set

$$J^{\mu}(x_1(0),\ldots,x_m(0)) = E\left[\sum_{t=0}^{\infty} a^t \sum_{l=1}^{m} r_l(t)\middle| x_1(0),\ldots,x_m(0)\right].$$

An optimality equation as in the previous sections can be also formalized here.

MABs provide a very rich modeling framework, e.g., clinical trials, portofolio management, etc.

Remarks:

1. MABs are a paradigm of **unsupervised learning**.
2. The rewards of actions not taken are unknown.
3. **Exploration** is required to solve (15).
4. MABs are simpler than RL problems, since we know which action is responsible for each reward. As mentioned earlier, in the RL setup, the impact of a current action may appear after several time instants in the future (delayed feedback). Therefore, the reward (or cost) at every time instant may be the aggregate effect of several prior actions.

5.1 Special Case of Interest: Stochastic MABs

Consider a bandit problem with m arms corresponding to m distributions $\nu_1, \nu_2, \ldots, \nu_m$, all of them having support the interval $[0, 1]$. The support selection is motivated for simplicity reasons. These distributions are unknown to the agent. Suppose that at time k the agent selects an arm (distribution) with index I_k (i.e., $u_k = I_k$). She then receives reward $r_{I_k,k} \sim \nu_{I_k}$. The agent's goal is to determine which distribution produces the highest expected reward, while maximizing her revenue at the same time. Therefore, she needs to determine an optimal policy for this purpose. The performance measure for a time horizon N corresponds to the **cumulative regret**, defined as:

$$R_N = \max_{i\in\{1,2,\ldots,m\}} \sum_{k=1}^{N} r_{i,k} - \sum_{k=1}^{N} r_{I_k,k}.$$

We note here that there is a version of the problem in which the time horizon N is unknown beforehand.

Regret Intuition: The regret is a measure of how much the bandit algorithm (or the agent's policy) "regrets" not playing the arm with maximum total reward over the course of N time instants. Also the definition of the regret assumes that at every time instant all reward distributions (arms) $\nu_1, \nu_2, \ldots, \nu_m$ are sampled, but the agent can only observe the reward produced by her chosen arm.

There are various ways that one may try to minimize the above regret, e.g., in the worst case or in expectation or with high probability. The stochastic MABs correspond to minimizing the **expected regret**:

$$E[R_N] = E\left[\max_{i\in\{1,2,\ldots,m\}} \sum_{k=1}^{N} r_{i,k} - \sum_{k=1}^{N} r_{I_k,k}\right].$$

Due to the existence of a maximization inside the expectation, the above performance metric is difficult to work with. Therefore, the alternative **pseudo-regret** is the usual objective:

$$\bar{R}_N = \max_{i \in \{1,2,\ldots,m\}} E\left[\sum_{k=1}^{N} r_{i,k} - \sum_{k=1}^{N} r_{I_k,k}\right].$$

Pseudo-regret Intuition: The pseudo-regret is a measure of how much the bandit algorithm "regrets" not choosing always the arm with the highest **expected** reward. We note here that for each distribution $\nu_1, \nu_2, \ldots, \nu_m$ the expected reward is nothing else but the corresponding mean value. Additionally, $\bar{R}_N \leq E[R_N]$ always. Therefore, it should be kept in mind that minimizing or upper bounding $\bar{R}_N$ has no explicit implications about the behavior of $E[R_N]$ under the same bandit algorithm.

Let $\mu_1, \mu_2, \ldots, \mu_m$ be the mean values of $\nu_1, \nu_2, \ldots, \nu_m$, respectively. Then,

$$\bar{R}_N = N \max_i \mu_i - \sum_{k=1}^{N} E[\mu_{I_k}] = N \max_i \mu_i - \sum_{i=1}^{m} E[N_{i,N}]\mu_i,$$

where $E[N_{i,N}]$ corresponds to the expected number of times arm i will be chosen in the course of N trials. Minimizing $\bar{R}_N$ corresponds to an **exploration-exploitation tradeoff**: the agent needs to try all arms sufficiently many times to determine the best one. Nevertheless, any time a suboptimal hand is chosen, e.g., arm r with $\mu_r < \max_i \mu_i$, a one-step regret is incurred:

$$\Delta_r = \max_i \mu_i - \mu_r.$$

Insufficient exploration can lead the bandit algorithm to erroneously decide a suboptimal arm as the optimal arm. Too much exploration implies that suboptimal hands are played for large fractions of trials and since N is fixed, an increased regret is incurred.

Note: From the above discussion, it should be intuitively clear that hard problems correspond to similar looking hands with different mean values.

Lai and Robbins Key Results: They provided policies for which

$$\bar{R}_N = O(\log N),$$

i.e., these algorithms achieve a sublinear pseudo-regret. Lai and Robbins also showed that no algorithm can lead to a better regret.

Benchmark Algorithm: An algorithm is said to solve the stochastic MAB problem if it achieves logarithmic pseudo-regret. A scheme that achieves such a regret is the **Upper Confidence Bound** (UCB) algorithm. Moreover, an early algorithm used for stochastic MABs, which has attracted a lot of interest lately, is called **Thompson sampling**. This algorithm makes use of a Bayesian approach and has been observed to perform well empirically. Thompson sampling has been also shown to achieve logarithmic pseudo-regret.

6 Contextual Bandits

The simplest way to think about contextual bandits is to consider them as **bandits with partial information**. The usual definition is the following repetitive scheme:

For $t = 1, 2, \ldots, N$:

1. *The environment produces some context $x_t \in \mathcal{X}$.*
2. *The controller chooses an action $u_t = \mu_t(x_t)$.*
3. *The environment produces the reward $r_t = r_t(u_t)$.*

The term "contextual bandits" is equivalent to terms like associative reinforcement learning, associative bandits, learning with partial feedback.

Note: The remarks for MABs made at the end of the first part of Section 5 are still valid. The difference of contextual bandits from MABs is the context $x_t \in \mathcal{X}$ (side information). No x_t exists in MABs.

Although the above small reference to contextual bandits is provided for encyclopedic reasons, contextual bandits will not be considered in this course.

7 Some Background Material: Descent Optimization Methods

Suppose we want to numerically solve the problem:

$$\min_{x \in \mathbb{R}^n} f(x).$$

A well-known algorithm for this task is the **steepest descent**:

$$x_{k+1} = x_k - \epsilon_k \nabla f(x_k).$$

The sequence $\{\epsilon_k\}_{k \geq 0}$ corresponds to the stepsizes used by the above algorithm and has to be judiciously chosen to provide appropriate convergence. Various approaches exist on how to select the stepsizes, e.g., fixed stepsizes, optimal line search, Armijo rule, etc. A generalization of the steepest descent is the **gradient descent**:

$$x_{k+1} = x_k + \epsilon_k d_k.$$

Here, we re-use the notation ϵ_k for the stepsizes for notational conservatism. For this algorithm to perform the desired task we require that

$$\nabla f(x_k)^T d_k < 0, \quad \forall k \geq 0.$$

Many machine learning problems can be formulated as optimization problems of the form:

$$\min_x F(x), \tag{16}$$

where $F(x) = E_y[f(x; y)]$ with f being a loss function. Here, x corresponds to a model that has to be inferred and y is some random variable or random vector. In practice, we observe only samples from distributions that are typically unknown to us. Assuming that we have n i.i.d. observations $\{y_i\}_{i=1}^n$, which we call **training data**, $F(x) \approx n^{-1} \sum_{i=1}^n f(x; y_i)$ and (16) is replaced by:

$$\min_x \frac{1}{n} \sum_{i=1}^n f(x; y_i). \tag{17}$$

This surrogate problem is often said to rely on the **empirical risk minimization** principle, which corresponds to a **probably approximately correct (PAC)** learning algorithm under some sufficient condition. Examples of loss functions correspond to logistic regression, squared error loss (linear regression), principal component analysis, neural network loss, Huber loss, etc. Depending on the choice of f and the domain of x, the optimization problem (17) can be convex (linear and logistic regression, SVMs, etc.) or nonconvex (e.g., neural networks). Convex problems are easy to solve, while nonconvex problems arising in machine learning are typically NP-hard. General gradient methods have to compute or to estimate the gradient:

$$\nabla F(x) = \nabla E_y[f(x;y)] \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_x f(x;y_i).$$

For large-scale optimization problems (very large n), the above computation of the gradient per iteration is cumbersome.

Idea: Stochastic Gradient Descent (SGD): Replace in steepest or gradient descent the gradient $\nabla F(x)$ by

$$\nabla_x f(x;y_j),$$

where y_j is a randomly chosen training datum.

Remarks:

1. Typically, we choose j uniformly at random from the set $\{1,2,\ldots,n\}$.

2. A different option is the **mini-batch SGD**. Here, instead of a single j, a random subset $I_k \subset \{1,2,\ldots,n\}$ of size $m \ll n$ is drawn at every iteration and the gradient $\nabla F(x)$ is approximated as
$$\nabla F(x) \approx \frac{1}{m}\sum_{i\in\mathcal{I}_k} \nabla_x f(x;y_i).$$

3. Both SGD and mini-batch SGD rely on unbiased estimators of the true gradient:
$$E_y\left[\nabla_x f(x;y_j)\right] = \nabla F(x), \quad \forall j \in \{1,2,\ldots,n\},$$
$$E_y\left[\frac{1}{m}\sum_{i\in\mathcal{I}_k} \nabla_x f(x;y_i)\right] = \nabla F(x), \quad \forall \mathcal{I}_k \subset \{1,2,\ldots,n\}.$$
The mini-batch SGD has lower variance than the SGD, but it is computationally more expensive.

Convergence: Under the assumption that $\sum_{k=1}^{\infty} \epsilon_k = \infty$ and $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$ the SGD converges to a local minimum almost surely.

Relevance to MDPs and RL: Gradient descent and SGD characterize the structure of main algorithms in the MDP/RL framework such as the **Q-learning** and **policy gradient** algorithms.