Reinforcement Learning: Foundations

Shie Mannor and Yishay Mansour

September 2019

Contents

Chapter 1

Introduction and Overview

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,

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

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). $\square$

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 1.1. *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

Chapter 2

Deterministic Decision Processes

In this chapter we introduce the dynamic system viewpoint of the optimal planning problem. We restrict the discussion here to deterministic (rather than stochastic) systems. We consider two basic settings. The finite-horizon decision problem and its recursive solution via finite-horizon Dynamic Programming. The average cost and it related minimum average weight cycle.

2.1 Discrete Dynamic Systems

We consider a discrete-time dynamic system, of the form:

$$\mathtt{s}_{\mathtt{t}+1} = f_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}), \quad \mathtt{t} = 0, 1, 2, \ldots, \mathtt{T} - 1$$

where

- $\mathtt{t}$ is the time index.
- $\mathtt{s}_{\mathtt{t}} \in \mathcal{S}_{\mathtt{t}}$ is the state variable at time $\mathtt{t}$, and $\mathcal{S}_{\mathtt{t}}$ is the set of possible states at time $\mathtt{t}$.
- $\mathtt{a}_{\mathtt{t}} \in \mathcal{A}_{\mathtt{t}}$ is the control variable at time $\mathtt{t}$, and $\mathcal{A}_{\mathtt{t}}$ is the set of possible control actions at time $\mathtt{t}$.
- $f_{\mathtt{t}} : \mathcal{S}_{\mathtt{t}} \times \mathcal{A}_{\mathtt{t}} \to \mathcal{S}_{\mathtt{t}+1}$ is the state transition function, which defines the *state dynamics* at time $\mathtt{t}$.
- $\mathtt{T} > 0$ is the *time horizon* of the system. It can be finite or infinite.

Remark 2.1. *More generally, the set $\mathcal{A}_{\mathtt{t}}$ of available actions may depend on the state at time* $\mathtt{t}$*, namely:* $\mathtt{a}_{\mathtt{t}} \in \mathcal{A}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}) \subset \mathcal{A}_{\mathtt{t}}$.

Remark 2.2. *The system is, in general, time-varying. It is called* time invariant *if* $f_{\mathtt{t}}, \mathcal{S}_{\mathtt{t}}, \mathcal{A}_{\mathtt{t}}$ *do not depend on the time* $\mathtt{t}$*. In that case we write*

$$\mathtt{s}_{\mathtt{t}+1} = f(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}), \quad \mathtt{t} = 0, 1, 2, \ldots, \mathtt{T}-1; \quad \mathtt{s}_{\mathtt{t}} \in \mathcal{S},\ \mathtt{a}_{\mathtt{t}} \in \mathcal{A}(\mathtt{s}_{\mathtt{t}}).$$

Remark 2.3. *The state dynamics may be augmented by an output equation:*

$$\mathtt{o}_{\mathtt{t}} = \mathcal{O}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}),$$

where $\mathtt{o}_{\mathtt{t}}$ *is the system observation, or the output. In most of this book we implicitly assume that* $\mathtt{o}_{\mathtt{t}} = \mathtt{s}_{\mathtt{t}}$*, namely, the current state* $\mathtt{s}_{\mathtt{t}}$ *is fully observed.*

Example 2.1. ***Linear Dynamic Systems***

A well known example of a dynamic system is that of a linear time-invariant system, where:

$$\mathtt{s}_{\mathtt{t}+1} = A\mathtt{s}_{\mathtt{t}} + B\mathtt{a}_{\mathtt{t}}$$

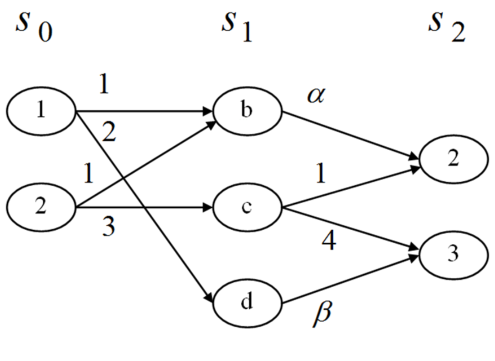
with $\mathtt{s}_{\mathtt{t}} \in \mathbb{R}^n$, $\mathtt{a}_{\mathtt{t}} \in \mathbb{R}^m$, $A \in \mathbb{R}^{n\times n}$ *and* $B \in \mathbb{R}^{n\times m}$*. Here the state and action spaces are evidently continuous (and not discrete).*

Example 2.2. ***Finite models***

Our emphasis here will be on finite state and action *models. A finite state space contains a finite number of points:* $\mathcal{S}_{\mathtt{t}} = \{1, 2, \ldots, n_{\mathtt{t}}\}$*. Similarly, a finite action space implies a finite number of control actions at each stage:*

$$\mathcal{A}_{\mathtt{t}}(\mathtt{s}) = \{1, 2, \ldots, m_{\mathtt{t}}(\mathtt{s})\},\ \ \mathtt{s} \in \mathcal{S}_{\mathtt{t}}$$

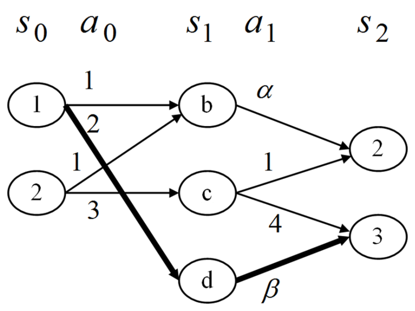
Graphical description: Finite models (over finite time horizons) can be represented by a corresponding decision graph:



Here:

- $\mathtt{T} = 2$, $\mathcal{S}_0 = \{1, 2\}$, $\mathcal{S}_1 = \{b, c.d\}$, $\mathcal{S}_2 = \{2, 3\}$,
- $\mathcal{A}_0(1) = \{1, 2\}$, $\mathcal{A}_0(2) = \{1, 3\}$, $\mathcal{A}_1(b) = \{\alpha\}$, $\mathcal{A}_1(c) = \{1, 4\}$, $\mathcal{A}_1(d) = \{\beta\}$
- $f_0(1, 1) = b$, $f_0(1, 2) = d$, $f_0(2, 1) = b$, $f_0(2, 3) = c$, $f_1(b, \alpha) = 2$, etc.

Definition 2.1. ***Feasible Path***
A feasible path for the specified system is a sequence $(\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s}_{\mathtt{T}-1}, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_\mathtt{T})$ *of states and actions, such that* $\mathtt{s}_\mathtt{t} \in \mathcal{A}_\mathtt{t}(\mathtt{s}_\mathtt{t})$ *and* $\mathtt{s}_{\mathtt{t}+1} = f_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$.



A feasible path

2.2 The Finite Horizon Decision Problem

We proceed to define our first and simplest planning problem. For that we need to specify a *performance objective* for our model, and the notion of *control policies*.

2.2.1 Costs and Rewards

The cumulative cost: Let $\mathtt{h}_\mathtt{T} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_{\mathtt{T}-1}, \mathtt{s}_\mathtt{T})$ denote an $\mathtt{T}$-stage feasible path for the system. Each feasible path $\mathtt{h}_\mathtt{T}$ is assign some cost $\mathcal{C}_\mathtt{T} = \mathcal{C}_\mathtt{T}(\mathtt{h}_\mathtt{T})$.

The standard definition of the cost $\mathcal{C}_\mathtt{T}$ is through the following *cumulative cost functional*:

$$\mathcal{C}_\mathtt{T}(\mathtt{h}_\mathtt{T}) = \sum_{\mathtt{t}=0}^{\mathtt{T}-1} \mathtt{c}_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) + c_\mathtt{T}(\mathtt{s}_\mathtt{T})$$

Here:

- $\mathtt{c}_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$ is the *instantaneous* cost or *single-stage* cost at stage $\mathtt{t}$, and $\mathtt{c}_\mathtt{t}$ is the instantaneous cost function.
- $\mathtt{c}_\mathtt{T}(\mathtt{s}_\mathtt{T})$ is the *terminal* cost, and $\mathtt{c}_\mathtt{T}$ is the terminal cost function.

Note:

- The cost functional defined above is *additive* in time. Other cost functionals are possible, for example the max cost, but additive cost is by far the most common and useful.
- We shall refer to $\mathcal{C}_{\mathtt{T}}$ as the *cumulative* $\mathtt{T}$*-stage cost*, or just the *cumulative cost.*

Our objective is to *minimize* the cumulative cost $\mathcal{C}_{\mathtt{T}}$, by a proper choice of actions. We will define that goal more formally in the next section.

Cost versus reward formulation: It is often more natural to consider *maximizing* reward rather than minimizing cost. In that case, we define the cumulative $\mathtt{T}$-stage return function:

$$\mathcal{V}_{\mathtt{T}}(h_{\mathtt{T}}) = \sum_{\mathtt{t}=0}^{\mathtt{T}-1} \mathtt{r}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}) + \mathtt{r}_{\mathtt{T}}(\mathtt{s}_{\mathtt{T}})$$

Here and $\mathtt{r}_{\mathtt{t}}$ is the instantaneous reward, and $\mathtt{r}_{\mathtt{T}}$ is the terminal reward. Clearly, minimizing $\mathcal{C}_{\mathtt{T}}$ is equivalent to maximizing $\mathcal{V}_{\mathtt{T}}$, if we set:

$$\mathtt{r}_{\mathtt{t}}(\mathtt{s}, \mathtt{a}) = -\mathtt{c}_{\mathtt{t}}(\mathtt{s}, \mathtt{a}) \text{ and } \mathtt{r}_{\mathtt{T}}(\mathtt{s}) = -\mathtt{c}_{\mathtt{T}}(\mathtt{s}).$$

We denote by $\mathbb{T}$ the set of time steps input setting, i.e., $\mathbb{T} = \{1, \dots, \mathtt{T}\}$

2.2.2 Optimal Paths

Our first planning problem is the following $\mathtt{T}$*-stage Finite Horizon Problem*:

- For a given initial state $\mathtt{s}_0$, find a feasible path $\mathtt{h}_{\mathtt{T}} = (\mathtt{s}_0, \mathtt{a}_0, \dots, \mathtt{s}_{\mathtt{T}-1}, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_{\mathtt{T}})$ that minimizes the cost functional $\mathcal{C}_{\mathtt{T}}(\mathtt{h}_{\mathtt{T}})$, over all feasible paths $\mathtt{h}_{\mathtt{T}}$.

Such a feasible path $\mathtt{h}_{\mathtt{T}}$ is called an *optimal path* from $\mathtt{s}_0$.

A more general notion than a path is that of a *control policy*, that specifies the action to be taken at each state. Control policies will play an important role in our Dynamic Programming algorithms, and are defined next.

2.2.3 Control Policies

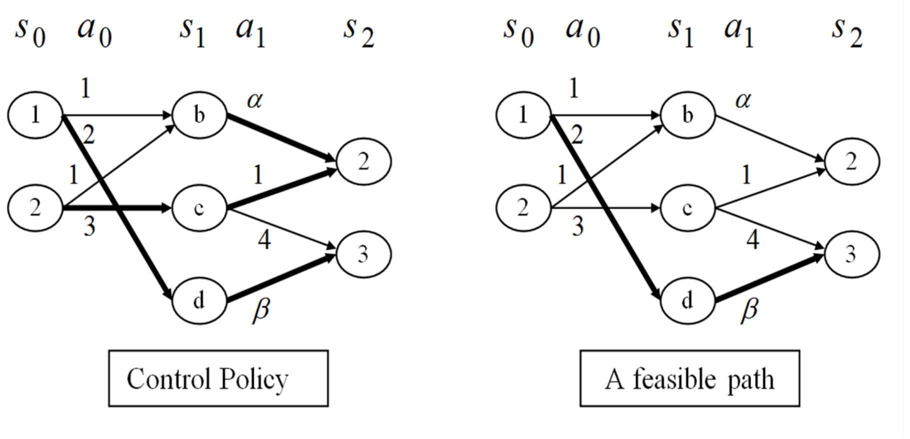
In general we will consider a few classes of control policies. The two basic dimensions in which we will characterize the control policies is their dependence on the history, and their use of randomization.

- A general or **history-dependent** control policy $\pi = (\pi_{\mathtt{t}})_{\mathtt{t}\in\mathbb{T}}$ is a mapping from each possible history $\mathtt{h_t} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s}_{\mathtt{t}-1}, \mathtt{a}_{\mathtt{t}-1}, \mathtt{s_t})$, $\mathtt{t} \in \mathbb{T}$, to an action $\mathtt{a_t} = \pi_{\mathtt{t}}(\mathtt{h_t}) \in \mathcal{A}_{\mathtt{t}}$. We denote the set of general policies by Π_H.
- A **Markov** control policy π is allowed to depend on the current state and time only: $\mathtt{a_t} = \pi_{\mathtt{t}}(\mathtt{s_t})$. We denote the set of Markov policies by Π_M.
- For stationary models, we may define **stationary** control policies that depend on the current state alone. A stationary policy is defined by a single mapping $\pi : \mathcal{S} \to \mathcal{A}$, so that $\mathtt{a_t} = \pi(\mathtt{s_t})$ for all $\mathtt{t} \in \mathbb{T}$. We denote the set of stationary policies by Π_S.
- Evidently, $\Pi_H \supset \Pi_M \supset \Pi_S$.

Randomized (Stochastic) Control policies

- The control policies defined above specify deterministically the action to be taken at each stage. In some cases we want to allow for a random choice of action.
- A general randomized (stochastic) control policy assigns to each possible history $\mathtt{h_t}$ a probability distribution $\pi_{\mathtt{t}}(\cdot|\mathtt{h_t})$ over the action set $\mathcal{A}_{\mathtt{t}}$. That is, $\Pr\{\mathtt{a_t} = \mathtt{a}|\mathtt{h_t}\} = \pi_{\mathtt{t}}(\mathtt{a}|\mathtt{h_t})$. We denote the set of general randomized policies by Π_{HS}.
- Similarly, we can define the set Π_{MS} of Markov randomized (stochastic) control policies, where $\pi_{\mathtt{t}}(\cdot|\mathtt{h_t})$ is replaced by $\pi_{\mathtt{t}}(\cdot|\mathtt{s_t})$, and the set Π_{SS} of stationary randomized (stochastic) control policies, where $\pi_{\mathtt{t}}(\cdot|\mathtt{s_t})$ is replaced by $\pi(\cdot|\mathtt{s_t})$.
- Note that the set Π_{HS} includes all other policy sets as special cases.

Control policies and paths: As mentioned, a control policy specifies an action for each state, whereas a path specifies an action only for states along the path. The definition of a policy, allows us to consider counter-factual events, namely, what would have been the path if we considered a different action. This distinction is illustrated in the following figure.



Induced Path: A control policy π, together with an initial state $\mathtt{s}_0$, specify a feasible path $\mathtt{h}_\mathtt{T} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s}_{N-1}, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_\mathtt{T})$. This path may be computed recursively using $\mathtt{a}_\mathtt{t} = \pi_\mathtt{t}(\mathtt{s}_\mathtt{t})$ and $\mathtt{s}_{\mathtt{t}+1} = f_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$, for $\mathtt{t} = 0, 1, \ldots, \mathtt{T} - 1$.

Remark 2.4. *Suppose that for each state $\mathtt{s}_\mathtt{t}$, each action $\mathtt{a}_\mathtt{t} \in \mathcal{A}_\mathtt{t}(\mathtt{s}_\mathtt{t})$ leads to a different state $\mathtt{s}_{\mathtt{t}+1}$ (i.e., at most one edge connects any two states). We can then identify each action $\mathtt{a}_\mathtt{t} \in \mathcal{A}_\mathtt{t}(\mathtt{s}_\mathtt{t})$ with the next state $\mathtt{s}_{\mathtt{t}+1} = f_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$ it induces. In that case a path may be uniquely specified by the state sequence $(\mathtt{s}_0, \mathtt{s}_1, \ldots, \mathtt{s}_\mathtt{T})$.*

2.2.4 Reduction between control policies classes

We first show a reduction from a general history dependent policies to Randomized Markovian policies. The main observation is that the only influence on the cumulative cost is the expected instantaneous cost $\mathbb{E}[\mathtt{c}_\mathtt{t}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})]$. Namely, let

$$\rho^\pi_\mathtt{t}(\mathtt{s}, \mathtt{a}) = \Pr_{\mathtt{h}_{\mathtt{t}-1}}[\mathtt{a}_\mathtt{t} = \mathtt{a} | \mathtt{s}_\mathtt{t} = s] = \mathbb{E}_{\mathtt{h}_{\mathtt{t}-1}}[\Pr[\mathtt{s}_\mathtt{t} = s, \mathtt{a}_\mathtt{t} = a | \mathtt{h}_{\mathtt{t}-1}]],$$

where $\mathtt{h}_{\mathtt{t}-1}$ is the history of the first $\mathtt{t} - 1$ time steps. Now we can rewrite the expected cost to go as,

$$\mathbb{E}[\mathcal{C}^\pi(\mathtt{s}_0)] = \mathbb{E}[\sum_{\mathtt{t}=1}^{\mathtt{T}-1} \sum_{\mathtt{a} \in \mathcal{A}_\mathtt{t}, \mathtt{s} \in \mathcal{S}_\mathtt{t}} \mathtt{c}_\mathtt{t}(\mathtt{s}, \mathtt{a}) \rho^\pi_\mathtt{t}(\mathtt{s}, \mathtt{a})].$$

This implies that any two policies π and π' for which $\rho^\pi_\mathtt{t}(\mathtt{s}, \mathtt{a}) = \rho^{\pi'}_\mathtt{t}(\mathtt{s}, \mathtt{a})$, for any time $\mathtt{t}$, state $\mathtt{s}$ and action $\mathtt{a}$, would have the same expected cumulative cost for any cost function, i.e., $\mathbb{E}[\mathcal{C}^\pi(\mathtt{s}_0)] = \mathbb{E}[\mathcal{C}^{\pi'}(\mathtt{s}_0)]$

Theorem 2.1. *For any policy $\pi \in \Pi_{HS}$, there is a policy $\pi' \in \Pi_{MS}$, such that for every state* $\mathtt{s}$ *and action* $\mathtt{a}$ *we have,* $\rho^{\pi}(\mathtt{s},\mathtt{a}) = \rho^{\pi'}(\mathtt{s},\mathtt{a})$. *This will imply that,*

$$\mathbb{E}[\mathcal{C}^{\pi}(\mathtt{s}_0)] = \mathbb{E}[\mathcal{C}^{\pi'}(\mathtt{s}_0)]$$

Proof. Given the policy $\pi \in \Pi_{HS}$, we define $\pi' \in \Pi_{MS}$ as follows. For every state $\mathtt{s} \in \mathcal{S}_{\mathtt{t}}$ we define

$$\pi'_{\mathtt{t}}(\mathtt{a}|\mathtt{s}) = \Pr_{\mathtt{h}_{\mathtt{t}-1}}[\mathtt{a}_{\mathtt{t}} = \mathtt{a}|\mathtt{s}_{\mathtt{t}} = \mathtt{s}] = \frac{\rho^{\pi}_{\mathtt{t}}(\mathtt{s},\mathtt{a})}{\sum_{\mathtt{a}' \in \mathcal{A}_{\mathtt{t}}} \rho^{\pi}_{\mathtt{t}}(\mathtt{s},\mathtt{a}')}$$

By definition π' is Markovian (depends only on the time $\mathtt{t}$ and the realized state $\mathtt{s}$). By construction $\rho^{\pi}_{\mathtt{t}}(\mathtt{s},\mathtt{a}) = \rho^{\pi'}_{\mathtt{t}}(\mathtt{s},\mathtt{a})$ is identical, which implies that $\mathbb{E}[\mathcal{C}^{\pi}(\mathtt{s}_0)] = \mathbb{E}[\mathcal{C}^{\pi'}(\mathtt{s}_0)]$. □

Next we show that for any stochastic Markovian policy there is a deterministic Markovian policy with at most the same cumulative cost.

Theorem 2.2. *For any policy $\pi \in \Pi_{MS}$, there is a policy $\pi' \in \Pi_{MD}$, such that*

$$\mathbb{E}[\mathcal{C}^{\pi}(\mathtt{s}_0)] \geq \mathbb{E}[\mathcal{C}^{\pi'}(\mathtt{s}_0)]$$

Proof. The proof is by backward induction on the steps. The inductive claim is:
For any policy $\pi \in \Pi_{MS}$ which is deterministic in $[\mathtt{t}+1,\mathtt{T}]$, *there is a policy $\pi' \in \Pi_{MS}$ which is deterministic in* $[\mathtt{t},\mathtt{T}]$ *and* $\mathbb{E}[\mathcal{C}^{\pi}(\mathtt{s}_0)] \geq \mathbb{E}[\mathcal{C}^{\pi'}(\mathtt{s}_0)]$.
Clearly, the theorem follows from the case of $\mathtt{t} = 0$.

For the base of the induction we can take $\mathtt{t} = \mathtt{T}$, which holds trivially.

For the inductive step, assume that $\pi \in \Pi_{MS}$ is deterministic in $[\mathtt{t}+1,\mathtt{T}]$.

For every $\mathtt{s}_{\mathtt{t}+1} \in \mathcal{S}_{\mathtt{t}+1}$ define

$$\mathcal{C}_{\mathtt{t}+1}(\mathtt{s}_{\mathtt{t}+1}) = \mathcal{C}(path(\mathtt{s}_{\mathtt{t}+1},\ldots,\mathtt{s}_{\mathtt{T}})),$$

where $path(\mathtt{s}_{\mathtt{t}+1},\ldots,\mathtt{s}_{\mathtt{T}})$ is the deterministic path from $\mathtt{s}_{\mathtt{t}+1}$ induced by π.

We define π' to be identical to π for all time steps $\mathtt{t}' \neq \mathtt{t}$. We define $\pi'_{\mathtt{t}}$ for each $\mathtt{s}_{\mathtt{t}} \in \mathcal{S}_{\mathtt{t}}$ as follows:

$$\pi'_{\mathtt{t}}(\mathtt{a}_{\mathtt{t}},\mathtt{s}_{\mathtt{t}}) = \arg\min_{\mathtt{a} \in \mathcal{A}_{\mathtt{t}}} \mathtt{c}(\mathtt{s}_{\mathtt{t}},\mathtt{a}) + \mathcal{C}_{\mathtt{t}}(f_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}},\mathtt{a})). \tag{2.1}$$

Recall that since we have a Deterministic Decision Process $f_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}},\mathtt{a}_{\mathtt{t}}) \in \mathcal{S}_{\mathtt{t}+1}$ is the next state if we take action $\mathtt{a}$ in $\mathtt{s}_{\mathtt{t}}$.

For the analysis, note that π and π' are identical until time $\mathtt{t}$, so they generate exactly the same distribution over paths. At time $\mathtt{t}$, π' is define to minimize the cost to go from $\mathtt{s_t}$, given that we follow π from $\mathtt{t}+1$ to $\mathtt{T}$. Therefore the cost can only decrease. Formally, let $\mathbb{E}^{\pi}[\cdot]$ be the expectation with respect to policy π.

$$\begin{aligned}\mathbb{E}^{\pi}_{\mathtt{s_t}}[\mathcal{C}_{\mathtt{t}}(\mathtt{s_t})] =&\mathbb{E}^{\pi}_{\mathtt{s_t}}\mathbb{E}^{\pi}_{\mathtt{a_t}}[\mathtt{c}(\mathtt{s_t},\mathtt{a_t})+\mathcal{C}_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s_t},\mathtt{a_t}))]\\ \geq&\mathbb{E}^{\pi}_{\mathtt{s_t}}\min_{\mathtt{a_t}\in\mathcal{A}_{\mathtt{t}}}[\mathtt{c}(\mathtt{s_t},\mathtt{a_t})+\mathcal{C}_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s_t},\mathtt{a_t}))]\\ =&\mathbb{E}^{\pi'}_{\mathtt{s_t}}[\mathcal{C}_{\mathtt{t}}(\mathtt{s_t})]\end{aligned}$$

□

Remark 2.5. *The above proof extends very naturally for the case of a stochastic MDP, which implies that $f_{\mathtt{t}}$ is stochastic. The modification of the proof would simply take the expectation over $f_{\mathtt{t}}$ in Eq. 2.1.*

2.2.5 Optimal Control Policies

Definition 2.2. *A control policy $\pi \in \Pi_{MD}$ is called **optimal** if, for each initial state $\mathtt{s}_0$, it induces an optimal path $\mathtt{h_T}$ from $\mathtt{s}_0$.*

An alternative definition can be given in terms of policies only. For that purpose, let $\mathtt{h_T}(\pi;\mathtt{s}_0)$ denote the path induced by the policy π from $\mathtt{s}_0$. For a given return functional $\mathcal{V}_{\mathtt{T}}(\mathtt{h_T})$, denote $\mathcal{V}_{\mathtt{T}}(\pi;\mathtt{s}_0)=\mathcal{V}_{\mathtt{T}}(\mathtt{h_T}(\pi;\mathtt{s}_0))$ That is, $\mathcal{V}_{\mathtt{T}}(\pi;\mathtt{s}_0)$ is the cumulative return for the path induced by π from $\mathtt{s}_0$.

Definition 2.3. *A control policy $\pi \in \Pi_{MD}$ is called optimal if, for each initial state $\mathtt{s}_0$, it holds that $\mathcal{V}_{\mathtt{T}}(\pi;\mathtt{s}_0) \geq \mathcal{V}_{\mathtt{T}}(\tilde{\pi};\mathtt{s}_0)$ for any other policy $\tilde{\pi} \in \Pi_{MD}$.*

Equivalence of the two definitions can be easily established (exercise). An optimal policy is often denoted by π^*.

The standard t_f-stage finite-horizon planning problem: Find a control policy π for the $\mathtt{T}$-stage Finite Horizon problem that minimizes the cumulative cost (or maximizes the cumulative return) function.

The naive approach to finding an optimal policy: For finite models (i.e., finite state and action spaces), the number of feasible paths (or control policies) is finite. It is therefore possible, in principle, to enumerate all $\mathtt{T}$-stage paths, compute the cumulative return for each one, and choose the one which gives the largest return. Let us evaluate the number of different paths and control policies. Suppose for simplicity that number of states at each stage is the same: $|\mathcal{S}_{\mathtt{t}}|=n$, and similarly the number of actions at each state is the same: $|\mathcal{A}_{\mathtt{t}}(x)| = m$ (with $m \leq n$) . The number of feasible $\mathtt{T}$-stage paths for each initial state is seen to be $m^{\mathtt{T}}$. The number of different policies is $m^{n\mathtt{T}}$. For example, for a fairly small problem with $\mathtt{T} = n = m = 10$, we obtain 10^{10} paths for each initial state (and 10^{11} overall), and 10^{100} control policies. Clearly it is not computationally feasible to enumerate them all.

Fortunately, Dynamic Programming offers a drastic simplification of the computational complexity for this problem.

2.3 Finite Horizon Dynamic Programming

The Dynamic Programming (DP) algorithm breaks down the $\mathtt{T}$-stage finite-horizon problem into $\mathtt{T}$ sequential single-stage optimization problems. This results in dramatic improvement in computation efficiency.

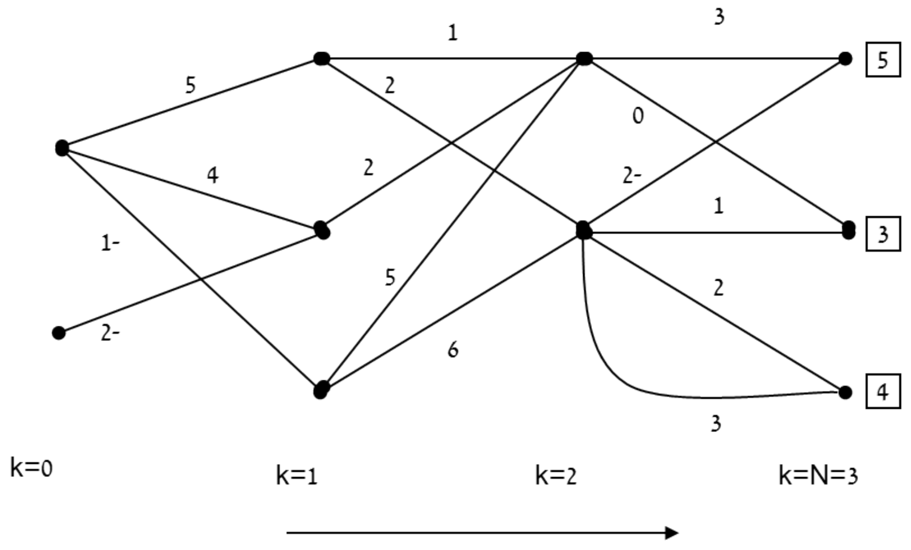
The DP technique for dynamic systems is based on a general observation called Bellman's Principle of Optimality. Essentially it states the following (for deterministic problems):

- **Any sub-path of an optimal path is itself an optimal path between its end point.**

To see why this should hold, consider a sub-path which is not optimal. We can replace it be an optimal sub-path, and improve the return.

Applying this principle recursively from the last stage backward, obtains the (backward) Dynamic Programming algorithm. Let us first illustrate the idea with following example.

Example 2.3. *Shortest path on a decision graph: Suppose we wish to find the shortest path (minimum cost path) from the initial node in* $\mathtt{T}$ *steps.*



The boxed values are the terminal costs at stage T*, the other number are the link costs. Using backward recursion, we may obtain that the minimal path costs from the two initial states are* 7 *and* 3*, as well as the optimal paths and an optimal policy.*

We can now describe the DP algorithm. Recall that we consider the dynamic system

$$\mathtt{s}_{\mathtt{t}+1} = f_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}), \quad \mathtt{t} = 0, 1, 2, \ldots, \mathtt{T}-1$$
$$\mathtt{s}_{\mathtt{t}} \in \mathcal{S}_{\mathtt{t}}, \quad \mathtt{a}_{\mathtt{t}} \in \mathcal{A}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}})$$

and we wish to maximize the cumulative return:

$$\mathcal{V}_{\mathtt{T}} = \sum_{\mathtt{t}=0}^{\mathtt{T}-1} \mathtt{r}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}) + \mathtt{r}_{\mathtt{T}}(\mathtt{s}_{\mathtt{T}})$$

The DP algorithm computes recursively a set of **value functions** $\mathcal{V}_{\mathtt{t}} : \mathcal{S}_{\mathtt{t}} \to \mathbb{R}$, where $\mathcal{V}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}})$ is the value of an optimal sub-path $\mathtt{h}_{\mathtt{t:T}} = (\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}, \ldots, \mathtt{s}_{\mathtt{T}})$ that starts at $\mathtt{s}_{\mathtt{t}}$.

Algorithm 2.1. ***Finite-horizon Dynamic Programming***

1. *Initialize the value function:* $\mathcal{V}_{\mathtt{T}}(\mathtt{s}) = \mathtt{r}_{\mathtt{T}}(\mathtt{s})$, $\mathtt{s} \in \mathcal{S}_{\mathtt{T}}$.

2. *Backward recursion: For* $\mathtt{t} = \mathtt{T}-1, \ldots, 0$, *compute*

$$\mathcal{V}_{\mathtt{t}}(\mathtt{s}) = \max_{\mathtt{a} \in \mathcal{A}_{\mathtt{t}}} \left\{ \mathtt{r}_{\mathtt{t}}(\mathtt{s}, \mathtt{a}) + \mathcal{V}_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s}, \mathtt{a})) \right\}, \quad \mathtt{s} \in \mathcal{S}_{\mathtt{t}}.$$

3. *Optimal policy: Choose any control policy* $\pi^* = (\pi^*_{\mathtt{t}})$ *that satisfies:*

$$\pi^*_{\mathtt{t}}(\mathtt{s}) \in \arg\max_{\mathtt{a} \in \mathcal{A}_{\mathtt{t}}} \left\{ \mathtt{r}_{\mathtt{t}}(\mathtt{s}, \mathtt{a}) + \mathcal{V}_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s}, \mathtt{a})) \right\}, \quad \mathtt{t} = 0, \ldots, \mathtt{T} - 1.$$

Proposition 2.1. *The following holds for finite-horizon dynamic programming:*

1. *The control policy* π^* *computed in Algorithm 2.1 is an optimal control policy for the* $\mathtt{T}$*-stage Finite Horizon problem.*

2. $\mathcal{V}_0(\mathtt{s})$ *is the optimal* $\mathtt{T}$*-stage return from initial state* $\mathtt{s}_0 = \mathtt{s}$*:*

$$\mathcal{V}_0(s) = \max_{\pi} V^{\pi}_0(\mathtt{s}), \qquad \forall \mathtt{s} \in \mathcal{S}_0,$$

where $V^{\pi}_0(\mathtt{s})$ *is the expected return of policy* π *when started at state* $\mathtt{s}$.

Proof. We show that the computed policy π^* is optimal and its return from time t is $\mathcal{V}_{\mathtt{t}}$. We will establish the following inductive claim:

For any time $\mathtt{t}$ *and any state* $\mathtt{s}$*, the path from* $\mathtt{s}$ *define by* π^* *is the maximum return path of length* $\mathtt{T} - \mathtt{t}$*. The value of* $\mathcal{V}_{\mathtt{t}}(\mathtt{s})$ *is the maximum return from* $\mathtt{s}$*.*

The proof is by a backward induction. For the basis of the induction we have: $\mathtt{t} = \mathtt{T}$, and the inductive claim follows from the initialization.

Assume the inductive claim holds for $\mathtt{t}$ prove for $\mathtt{t} + 1$. For contradiction assume there is a higher return path from $\mathtt{s}$. Let the path generated π^* be $P = (\mathtt{s}, \mathtt{s}^*_{\mathtt{T}-\mathtt{t}}, \ldots, \mathtt{s}^*_{\mathtt{T}})$. Let $P_1 = (\mathtt{s}, \mathtt{s}_{\mathtt{T}-\mathtt{t}}, \ldots, \mathtt{s}_{\mathtt{T}})$ be an alternative path. Let $P_2 = (\mathtt{s}, \mathtt{s}_{\mathtt{T}-\mathtt{t}}, \mathtt{s}'_{\mathtt{T}-\mathtt{t}}, \ldots, \mathtt{s}'_{\mathtt{T}})$ be the path generated by following π^* from $\mathtt{s}'_{\mathtt{T}-\mathtt{t}}$. Since P_1 and P_2 are identical except for the last $\mathtt{t}$ stages, we can use the inductive hypothesis, which implies that $\mathcal{V}(P_1) \leq \mathcal{V}(P_2)$. From the definition of π^* we have that $\mathcal{V}(P_2) \leq \mathcal{V}(P)$. Hence, $\mathcal{V}(P_1) \leq \mathcal{V}(P_2) \leq \mathcal{V}(P)$, which completes the proof of the inductive hypothesis. □

Let us make the following observations:

1. The algorithm involves visiting each state exactly once, proceeding backward in time. For each time instant (or stage) $\mathtt{t}$, the value function $\mathcal{V}_{\mathtt{t}}(\mathtt{s})$ is computed for all states $\mathtt{s} \in \mathcal{S}_{\mathtt{t}}$ before proceeding to stage $\mathtt{t} - 1$.

2. The *backward induction* step of Algorithm 2.1, along with similar equations in the theory of DP, is called **Bellman's equation**.

3. Computational complexity: There is a total of $n\mathtt{T}$ states (excluding the final one), and in each we need m computations. Hence, the number of required calculations is $mn\mathtt{T}$. For the example above with $m = n = \mathtt{T} = 10$, we need $O(10^3)$ calculations.

4. A similar algorithm that proceeds forward in time (from $\mathtt{t} = 0$ to $\mathtt{t} = \mathtt{T}$) can be devised. We note that this will not be possible for stochastic systems (i.e., the stochastic MDP model).

5. The celebrated **Viterbi algorithm** is an important instance of finite-horizon DP. The algorithm essentially finds the most likely sequence of states in a Markov chain $(\mathtt{s_t})$ that is partially (or noisily) observed. The algorithm was introduced in 1967 for decoding convolution codes over noisy digital communication links. It has found extensive applications in communications, and is a basic computational tool in Hidden Markov Models (HMMs), a popular statistical model that is used extensively in speech recognition and bioinformatics, among other areas.

2.4 Shortest Paths

We can formulate a DDP problems similar to shortest path problems. Given a directed graph $G(V, E)$, there is a set of goal states $\mathcal{S}_G$, and the goal is to reach one of the goal states. Formally, when we reach a goal state we stay there and have a zero cost. For such a DDP the optimal policy would be to compute a shortest path to one of the goal states.

There is a variety of algorithms for computing shortest paths.

1. Bellman-Ford: handles also negative weights, but assumes no negative cycle. Runs in time $O(|V| \cdot |E|)$.

2. Dijkstra: Assumes non-negative weights. Runs in time $O(|V| \log |E| + |E|)$

2.5 Linear Programming for Finite Horizon

In this section we will use linear programming to derive the optimal policy. We will see that both the primal and dual program will play an important part in defining the optimal policy. We will fix an initial state $\mathtt{s}_0$ and compute the optimal policy for it.

We will start with the primal linear program, which will compute the optimal policy. For each time $\mathtt{t}$, state $\mathtt{s}$ and action $\mathtt{a}$ we will have a variable $x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) \in \{0,1\}$ that will indicate by 1 that at time $\mathtt{t}$ we are at state $\mathtt{s}$ and perform action $\mathtt{a}$. For the terminal states $\mathtt{s}$ we will have a variable $x_{\mathtt{T}}(\mathtt{s}) \in \{0,1\}$ that will indicate whether we terminate at state $\mathtt{s}$.

Our main constraint will be a flow constraint, stating that if we reach state $\mathtt{s}$ at time $\mathtt{t}-1$ then we exit it at time $\mathtt{t}$. Formally,

$$\sum_{\mathtt{a}} x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) = \sum_{\mathtt{s}',\mathtt{a}':f_{\mathtt{t}-1}(\mathtt{s}',\mathtt{a}')=\mathtt{s}} x_{\mathtt{t}-1}(\mathtt{s}',\mathtt{a}').$$

and for terminal states simply

$$x_{\mathtt{T}}(\mathtt{s}) = \sum_{\mathtt{s}',\mathtt{a}':f_{\mathtt{T}-1}(\mathtt{s}',\mathtt{a}')=\mathtt{s}} x_{\mathtt{T}-1}(\mathtt{s}',\mathtt{a}')$$

The return, which we would like to maximize, would be

$$\sum_{\mathtt{t},\mathtt{s},\mathtt{a}} \mathtt{r}_{\mathtt{t}}(\mathtt{s},\mathtt{a}) x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) + \sum_{\mathtt{s}} \mathtt{r}_{\mathtt{T}}(\mathtt{s}) x_{\mathtt{T}}(\mathtt{s})$$

To have a linear program we will also need to relax the constraints of $\{0,1\}$ to $[0,1]$

The resulting linear program is the following.

$$\begin{aligned}
\max_{x_{\mathtt{t}}(\mathtt{s},\mathtt{a}),x_{\mathtt{T}}(\mathtt{s})} \quad & \sum_{\mathtt{t},\mathtt{s},\mathtt{a}} \mathtt{r}_{\mathtt{t}}(\mathtt{s},\mathtt{a}) x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) + \sum_{\mathtt{s}} \mathtt{r}_{\mathtt{T}}(\mathtt{s}) x_{\mathtt{T}}(\mathtt{s}) \\
& \text{such that} \\
& \sum_{\mathtt{a}} x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) = \sum_{\mathtt{s}',\mathtt{a}':f_{\mathtt{t}-1}(\mathtt{s}',\mathtt{a}')=\mathtt{s}} x_{\mathtt{t}-1}(\mathtt{s}',\mathtt{a}'). && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}, \mathtt{t} \in \mathbb{T} \\
& x_{\mathtt{T}}(\mathtt{s}) = \sum_{\mathtt{s}',\mathtt{a}':f_{\mathtt{T}-1}(\mathtt{s}',\mathtt{a}')=\mathtt{s}} x_{\mathtt{T}-1}(\mathtt{s}',\mathtt{a}') && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{T}} \\
& x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) \geq 0 && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}, \mathtt{t} \in \{0,\ldots,\mathtt{T}-1\} \\
& x_{\mathtt{t}}(\mathtt{s},\mathtt{a}) \leq 1 && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}, \mathtt{t} \in \{0,\ldots,\mathtt{T}-1\} \\
& \sum_{\mathtt{a}} x_0(\mathtt{s}_0,\mathtt{a}) = 1 \\
& x_0(\mathtt{s},\mathtt{a}) = 0, && \forall \mathtt{s} \in \mathcal{S}_0, \mathtt{s} \neq \mathtt{s}_0
\end{aligned}$$

Given the primal linear program we can derive the dual linear program.

$$\min_{z_{\mathtt{t}}(\mathtt{s})} \; z_0(\mathtt{s}_0)$$

such that

$$\begin{aligned} & z_{\mathtt{T}}(\mathtt{s}) = \mathtt{r}_{\mathtt{T}}(\mathtt{s}) && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}} \\ & z_{\mathtt{t}}(\mathtt{s}) \geq \mathtt{r}_{\mathtt{t}}(\mathtt{s},\mathtt{a}) + z_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s},\mathtt{a})), && \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}, \mathtt{t} \in \mathbb{T}, \\ & . \end{aligned}$$

One can identify the dual random variables $z_{\mathtt{t}}(\mathtt{s})$ with the optimal vale function $\mathcal{V}_{\mathtt{t}}(\mathtt{s})$. At the optimal solution of the dual linear program one can show that we have

$$z_{\mathtt{t}}(\mathtt{s}) = \max_{\mathtt{a}} \left\{ \mathtt{r}_{\mathtt{t}}(\mathtt{s},\mathtt{a}) + z_{\mathtt{t}+1}(f_{\mathtt{t}}(\mathtt{s},\mathtt{a})) \right\}, \qquad \forall \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{t} \in \mathbb{T},$$

which is the familiar Bellman optimality equations.

2.6 Average cost criteria

The average cost criteria considers the limit of the average costs. Formally:

$$\mathcal{C}^{\pi}_{avg} = \lim_{\mathtt{T}\to\infty} \frac{1}{\mathtt{T}} \sum_{t=1}^{\mathtt{T}} \mathtt{c}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})$$

where the trajectory is generated using π. The aim is to minimize $\mathbb{E}[\mathcal{C}^{\pi}_{avg}]$. This implies that any finite prefix has no influence of the final average cost, since its influence vanishes as $\mathtt{T}$ goes to infinity.

For a deterministic stationary policy, the policy converges to a simple cycle, and the average cost is the average cost of the edges on the cycle. (Recall, we are considering only DDP.)

Given a directed graph $G(V,E)$, let Ω be the collection of all cycles in $G(V,E)$. For each cycle $\omega = (v_1,\ldots,v_k)$, we define $c(\omega) = \sum_{i=1}^{k} c(v_i, v_{i+1})$, where (v_i, v_{i+1}) is the i-th edge in the cycle ω. Let $\mu(\omega) = \frac{c(\omega}{k}$. The *min average cost cycle* is

$$\mu^* = \min_{\omega\in\Omega} \mu(\omega)$$

We show that the min average cost cycle is the optimal policy.

Theorem 2.3. *For any Deterministic Decision Process (DPP) the optimal average costs is μ^*, and an optimal policy is π_ω that cycles around a simple cycle of average cost μ^*.*

Proof. Let ω be a cycle of average cost μ^*. Let π_ω be a deterministic stationary policy that first reaches ω and then cycles in ω. Clearly, $\mathcal{C}_{avg}^{\pi_\omega} = \mu^*$.

We show that for any policy π (possibly in Π_{HS}) we have that $\mathbb{E}[\mathcal{C}_{avg}^{\pi}] \geq \mu^*$. For contradiction assume that there is a policy π' that has average cost $\mu^* - \varepsilon$. Consider a sufficiently long run of length $\mathtt{T}$ of π', and fix any realization θ of it. We will show that the cumulative cost $\mathcal{C}(\theta) \geq (\mathtt{T} - n)\mu^*$, which implies that $\mathbb{E}[\mathcal{C}_{avg}^{\pi}] \geq \mu^* - n\mu^*/\mathtt{T}$.

Given θ, consider the first simple cycle ω in θ. The average cost of ω is $\mu(\omega) \geq \mu^*$. Delete ω from θ, reducing the number of edges by $|\omega|$ and the cumulative cost by $\mu(\omega)|\omega|$. We continue the process until there is no remaining cycles, which implies that we have at most $|V| = n$ nodes remaining. This implies that the costs of ω was at least $(\mathtt{T} - n)\mu^*$. This implies that the average cost of θ is at least $\mathcal{C}(\theta) \geq (1 - \frac{n}{\mathtt{T}})\mu^*$. For $\epsilon > \mu^* n/\mathtt{T}$ we have a contradiction. □

Next we develop an algorithm for computing the minimum average cost cycle, which implies an optimal policy for DDP for average costs. The input is a directed graph $G(V, E)$ with cost $\mathtt{c} : E \to \mathbb{R}$.

We first give a characterization of μ^*. Set a root $r \in V$. Let $F_k(v)$ be paths of length k from r to v. Let $d_k(v) = \min_{p \in F_k(v)} \mathtt{c}(p)$, where if $F_k(v) = \emptyset$ then $d_k(v) = \infty$. The following theorem (due to [R. Karp, A characterization of the minimum cycle mean in digraph, Discrete mathematics, 1978]) gives a characterization of μ^*.

Theorem 2.4.

$$\mu^* = \min_{v \in S} \max_{0 \leq k \leq n-1} \frac{d_n(v) - d_k(v)}{n - k},$$

where we define $\infty - \infty$ as ∞.

Proof. We have two cases, $\mu^* = 0$ and $\mu^* > 0$. We assume that the graph has no negative cycle (we can guarantee this by adding a large number M to all the weights).

We start with $\mu^* = 0$. This implies that we have in $G(V, E)$ a cycle of weight zero, but no negative cycle. For the theorem it is sufficient to show that

$$\min_{v \in S} \max_{0 \leq k \leq n-1} \{d_n(v) - d_k(v)\} = 0.$$

For every node $v \in S$ there is a path of length $k \in [0, n-1]$ of cost $d(v)$, the cost of the shortest path from r to v. This implies that

$$\max_{0 \leq k \leq n-1} \{d_n(v) - d_k(v)\} = d_n(v) - d(v) \geq 0$$

We need to show that for some $v \in V$ we have $d_n(v) = d(v)$, which implies that $\min_{v\in S}\{d_n(v) - d(v)\} = 0$.

Consider a cycle ω of cost $\mathcal{C}(\omega) = 0$ (there is one, since $\mu^* = 0$). Let v be a node on the cycle ω. Consider a path P from r to v which then cycles around ω and has length at least n. The path P is a shortest path to v (although not necessarily simple). This implies that any sub-path of P is also a shortest path. Let P' be a sub-path of P of length n and let it end in $u \in V$. Path P' is a shortest path to u, since it is a prefix of a shortest path P. This implies that the cost of P' is $d(u)$. Since P' is of length n, by construction, we have that $d_n(u) = d(u)$. Therefore, $\min_{v\in S}\{d_n(v) - d(v)\} = 0$, which completes the case that $\mu^* = 0$.

For $\mu^* > 0$ we subtract a constant $\Delta = \mu^*$ from al the costs in the graph. This implies that for the new costs we have a zero cycle and no negative cycle. We can now apply the previous case. It only remains to show that the formula changes by exactly $\Delta = \mu^*$.

Formally, for every edge $e \in E$ let $\mathsf{c}'(e) = \mathsf{c}(e) - \Delta$. For any path p we have $\mathcal{C}'(p) = \mathcal{C}(p) - |p|\Delta$, and for any cycle ω we have $\mu'(\omega) = \mu(\omega) - \Delta$. This implies that for $\Delta = \mu^*$ we have a cycle of cost zero and no negative cycles. We now consider the formula,

$$\begin{aligned}
0 = (\mu')^* &= \min_{v\in V} \max_{0\le k\le n-1}\{\frac{d'_n(v) - d'_k(v)}{n-k}\}\\
&= \min_{v\in V} \max_{0\le k\le n-1}\{\frac{d_n(v) - n\Delta - d_k(v) + k\Delta}{n-k}\}\\
&= \min_{v\in V} \max_{0\le k\le n-1}\{\frac{d_n(v) - d_k(v)}{n-k} - \Delta\}\\
&= \min_{v\in V} \max_{0\le k\le n-1}\{\frac{d_n(v) - d_k(v)}{n-k}\} - \Delta
\end{aligned}$$

Therefore we have

$$\mu^* = \Delta = \min_{v\in V} \max_{0\le k\le n-1}\{\frac{d_n(v) - d_{\mathtt{t}}(v)}{n-k}\}$$

□

We would like now to recover the minimum average cost cycle. The basic idea is to recover the cycle from the minimizing vertices in the formula, but some care need to be taken. It is true that some minimizing pair (v, k) the path of length n from r to v has a cycle of length $n - k$ which is the suffix of the path. The solution is that for the path p, from r to v of length n, any simple cycle is a minimum average cost cycle. (See, ["A note of finding minimum mean cycle", Mmamu Chaturvedi and Ross M. McConnell, IPL 2017]).

The running time of computing the minimum average cost cycle is $O(|V| \cdot |E|)$.

2.7 Historical Notes:

- Dynamic Programming was popularized in the 1950's and 1960's by **Richard Bellman** (1920-1984), an American mathematician. Bellman, who coined the term Dynamic Programming, formulated the general theory and proposed numerous applications in control and operations research.
- **Andrew Viterbi** (born 1935) is an American professor of electric engineer, a pioneer in the field of digital communications, and co-founder of Qualcomm Inc. (together with Irwin Jacobs). He has had close ties with the Technion, and has made many contributions to our department.
- **Richard Karp** (born 1935) is an American professor of computer science, known for his contributions to the theory of computing.

Chapter 3

Markov Chains

3.1 Markov Chains

We start by considering a stochastic dynamics model which does not have any actions, so that we can concentrate on the dynamics.

A Markov chain $\{X_{\mathtt{t}},\ \mathtt{t}=0,1,2,\ldots\}$, with $X_{\mathtt{t}} \in X$, is a discrete-time stochastic process, over a finite or countable state-space X, that satisfies the following Markov property:

$$\mathcal{P}(X_{\mathtt{t}+1}=j|X_{\mathtt{t}}=i, X_{\mathtt{t}-1},\ldots X_0) = \mathcal{P}(X_{\mathtt{t}+1}=j|X_{\mathtt{t}}=i).$$

We focus on time-homogeneous Markov chains, where

$$\mathcal{P}(X_{\mathtt{t}+1}=j|X_{\mathtt{t}}=i) = \mathcal{P}(X_1=j|X_0=i) \overset{\Delta}{=} p_{i,j}.$$

The $p_{i,j}$'s are the transition probabilities, which satisfy $p_{i,j} \geq 0$, and for each $i \in X$ we have $\sum_{j\in X} p_{i,j} = 1$, namely, $\{p_{i,j} : j \in X\}$ is a distribution on the next state following state i. The matrix $P = (p_{i,j})$ is the transition matrix. The matrix is row-stochastic (each row sums to 1 and all entries non-negative).

Given the initial distribution p_0 of X_0, namely $\mathcal{P}(X_0=i) = p_0(i)$, we obtain the finite-dimensional distributions:

$$\mathcal{P}(X_0=i_0,\ldots,X_{\mathtt{t}}=i_{\mathtt{t}}) = p_0(i_0)p_{i_0,i_1}\cdot\ldots\cdot p_{i_{\mathtt{t}-1},i_{\mathtt{t}}}.$$

Define $p_{i,j}^{(m)} = \mathcal{P}(X_m=j|X_0=i)$, the m-step transition probabilities. It is easy to verify that $p_{i,j}^{(m)} = [P^m]_{ij}$ (here P^m is the m-th power of the matrix P).

Example 3.1. [1] *Consider the following two state Markov chain, with transition probability P and initial distribution p_0, as follows:*

$$P = \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{pmatrix} \qquad p_0 = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$$

Initially, we have both states equally likely. After one step, the distribution of states is $p_1 = p_0 P = (0.3\ ,\ 0.7)$. After two steps we have $p_2 = p_1 P = p_0 P^2 = (0.26\ ,\ 0.74)$. The limit of this sequence would be $p_\infty = (0.25\ ,\ 0.75)$, which is called the steady state distribution*, and would be discussed later.*

State classification:

- State j is *accessible* from i (denoted by $i \to j$) if $p_{i,j}^{(m)} > 0$ for some $m \geq 1$.
  For a finite X we can compute the accessibility property as follows. Construct a directed graph $G(X, E)$ where the vertices are the states X and there is a directed edge (i, j) if $p_{i,j} > 0$. State j is accessible from state i iff there exists a directed path in $G(X, E)$ from i to j.

- States i and j are *communicating* states (or communicate) if $i \to j$ and $j \to i$.
  For a finite X, this implies that in $G(X, E)$ there is both a directed path from i to j and from j to i.

- A *communicating class* (or just class) is a maximal collection of states that communicate.
  For a finite X, this implies that in $G(X, E)$ we have i and j in the same strongly connected component of the graph. (A strongly connected component has a directed path between any pair of vertices.)

- The Markov chain is *irreducible* if all states belong to a single class (i.e., all states communicate with each other).
  For a finite X, this implies that in $G(X, E)$ is strongly connected.

- State i has a *period* $d_i = GCD\{m \geq 1 : p_{i,i}^{(m)} > 0\}$, where GCD is the greatest common divisor. A state is a-periodic if $d_i = 1$.
  State i is periodic with period $d_i \geq 2$ if $p_{i,i}^{(m)} = 0$ for $m \pmod{d_i} \neq 0$ and for any m such that $p_{i,i}^{(m)} > 0$ we have $m \pmod{d_i} = 0$.

[1] Simple MC, two states, running example.

- Periodicity is a class property: all states in the same class have the same period. Specifically, if some state is *a-periodic*, then all states in the class are a-periodic.

  Claim 3.1. *For any two states i and j with periods d_i and d_j, in the same communicating class, we have $d_i = d_j$.*

  Proof. For contradiction, assume that $d_j \pmod{d_i} \neq 0$. Since they are in the same communicating class, we have a trajectory from i to j of length $m_{i,j}$ and from j to i of length $m_{j,i}$. This implies that $(m_{i,j} + m_{j,i}) \pmod{d_i} = 0$. Now, there is a trajectory (which is a cycle) of length $m_{j,j}$ from j back to j such that $m_{j,j} \pmod{d_i} \neq 0$ (otherwise d_i divides the period of j). Consider the path from i to itself of length $m_{i,j} + m_{j,j} + m_{j,i}$. We have that $(m_{ij} + m_{jj} + m_{ji}) \pmod{d_i} = m_{jj} \pmod{d_i} \neq 0$. This is a contradiction to the definition of d_i. □

  Therefore, the periodicity is a class property, and all the states in a class have the same period.

Example 3.2. *Add figures explaining the definitions.*

Recurrence:

- State i is *recurrent* if $\mathcal{P}(X_{\mathtt{t}} = i \text{ for some } t \geq 1 | X_0 = i) = 1$. Otherwise, state i is *transient.*
  We can relate the property of recurrent and transient to the expected number of returns to a state.

  Claim 3.2. *State i is transient iff $\sum_{m=1}^{\infty} p_{i,i}^{(m)} < \infty$.*

  Proof. Assume that state i is transient. Let $q_i = \mathcal{P}(X_{\mathtt{t}} = i \text{ for some } t \geq 1 | X_0 = i)$. Since state i is transient we have $q_i < 1$. Let Z_i be the number of times the trajectory returns to state i. Note that Z_i is geometrically distributed with parameter q_i, namely $\Pr[Z_i = k] = q_i^k(1 - q_i)$. Therefore the expected number of returns to state i is $1/(1 - q_i)$ and is finite. The expected number of returns to state i is $\sum_{m=1}^{\infty} p_{i,i}^{(m)}$, and hence if a state is transient we have $\sum_{m=1}^{\infty} p_{i,i}^{(m)} < \infty$.

  For the other direction, assume that $\sum_{m=1}^{\infty} p_{i,i}^{(m)} < \infty$. This implies that there is an m_0 such that $\sum_{m=m_0}^{\infty} p_{i,i}^{(m)} < 1/2$. Consider the probability of returning to i

within m_0 stages. This implies that $\mathcal{P}(X_{\mathtt{t}} = i \text{ for some } t \geq m_0 | X_0 = i) < 1/2$. Now consider the probability $q_i' = \mathcal{P}(X_{\mathtt{t}} = i \text{ for some } m_0 \geq t \geq 1 | X_0 = i)$. If $q_i' < 1$, this implies that $\mathcal{P}(X_{\mathtt{t}} = i \text{ for some } t \geq 1 | X_0 = i) < q_i' + (1 - q_i')/2 = (1 + q_i')/2 < 1$. In this case state i is transient. If $q_i' = 1$, this implies that after at most m_0 stages we are guarantee to return to i, hence the expected number of return to state i is infinite, i.e., $\sum_{m=1}^{\infty} p_{i,i}^{(m)} = \infty$. This is in contradiction to the assumption that $\sum_{m=1}^{\infty} p_{i,i}^{(m)} < \infty$. □

- Recurrence is a class property.
  To see this consider two states i and j such that j is accessible from i and i is recurrent. Since i is recurrant $\sum_{m=1}^{\infty} p_{i,i}^{(m)} = \infty$. Since j is accessible from i there is a k such that $p_{i,j}^{(k)} > 0$. Since i is recurrent, there exists a k' such that $p_{j,i}^{(k')} > 0$. We can lower bound $\sum_{m=1}^{\infty} p_{j,j}^{(m)}$ by $\sum_{m=1}^{\infty} p_{j,i}^{(k')} p_{i,i}^{(m)} p_{i,j}^{(k)} = \infty$. Therefore we showed that state j is recurrent.

- If states i and j are in the same recurrent (communicating) class, then state j is (eventually) reached from state i with probability 1: $\mathcal{P}(X_{\mathtt{t}} = j \text{ for some t} \geq 1 | X_0 = i) = 1$.
  This follows from the fact that both states occur infinitely often, with probability 1.

- Let T_i be the *return time* to state i (number of stages required for $(X_{\mathtt{t}})$ to return to i). If i is a recurrent state, then $T_i < \infty$ w.p. 1.

- State i is *positive recurrent* if $\mathbb{E}(T_i) < \infty$, and null recurrent if $\mathbb{E}(T_i) = \infty$. If the state space X is finite, all recurrent states are positive recurrent.

Example 3.3. Random walk *Consider the following Markov chain over the integers. The states are the integers. The initial states is* 0. *At each state* i, *with probability* $1/2$ *we move to* $i + 1$ *and with probability* $1/2$ *to* $i - 1$. *Namely,* $p_{i,i+1} = 1/2$, $p_{i,i-1} = 1/2$, *and* $p_{i,j} = 0$ *for* $j \notin \{i - 1, i + 1\}$. *We will show that* T_i *is finite with probability* 1 *and* $\mathbb{E}[T_i] = \infty$. *This implies that all the states are null recurrent.*

To compute $\mathbb{E}[T_i]$ *consider what happens in two steps. We are either back to* i, *or at state* $i + 2$ *or* $i - 2$. *Let* $Z_{i,i-1}$ *be the time to move from* i *to* $i - 1$. *Note that we have* $\mathbb{E}[T_i] = 1 + 0.5\mathbb{E}[Z_{i+1,i}] + 0.5\mathbb{E}[Z_{i-1,i}] = 1 + \mathbb{E}[Z_{1,0}]$, *since, due to symmetry,*

$\mathbb{E}[Z_{i,i+1}] = \mathbb{E}[Z_{i+1,i}] = \mathbb{E}[Z_{1,0}]$. For $\mathbb{E}[T_i]$ we also have that

$$\begin{aligned}\mathbb{E}[T_i] =& 2 + \frac{1}{4}\mathbb{E}[Z_{i+2,i}] + \frac{1}{4}\mathbb{E}[Z_{i-2,i}] \\ =& 2 + \frac{1}{4}(\mathbb{E}[Z_{i+2,i+1}] + \mathbb{E}[Z_{i+1,i}]) + \frac{1}{4}(\mathbb{E}[Z_{i-2,i-1}] + \mathbb{E}[Z_{i-1,i}]) = 2 + E[Z_{1,0}]\end{aligned}$$

This implies that we have

$$\mathbb{E}[T_i] = 1 + \mathbb{E}[Z_{1,0}] = 2 + E[Z_{1,0}]$$

Clearly, there is no finite value for $\mathbb{E}[Z_{1,0}]$ which will satisfy the equation, which implies $\mathbb{E}[Z_{1,0}] = \infty$, and hence $\mathbb{E}[T_i] = \infty$.

To show that 0 is a recurrent state, note that the probability that at time $2k$ we are at state 0 is exactly $p^{(2k)}_{0,0} = \binom{2k}{k}2^{-2k} \approx \frac{c}{\sqrt{k}}$, for some $c > 0$. This implies that

$$\sum_{m=1}^{\infty} p^{(m)}_{0,0} \approx \sum_{m=1}^{\infty} \frac{c}{\sqrt{m}} = \infty$$

and there for state 0 is recurrent. (Actually, this shows that all the states are recurrent).

Note that this Markov chain has a period of 2.

Example 3.4. Random walk with jumps. *Consider the following Markov chain over the integers. The states are the integers. The initial states is 0. At each state i, with probability $1/2$ we move to $i+1$ and with probability $1/2$ we return to 0. Namely, $p_{i,i+1} = 1/2$, $p_{i,0} = 1/2$, and $p_{i,j} = 0$ for $j \notin \{0, i+1\}$. We will show that $\mathbb{E}[T_i] < \infty$ (which implies that T_i is finite with probability 1).*

From any state we return to 0 with probability $1/2$, therefore $\mathbb{E}[T_0] = 2$. We will show that for state i we have $\mathbb{E}[T_i] \le 2 + 2 \cdot 2^i$. We will decompose T_i to two parts. The first is the return to 0, this part has expectation 2. The second is to reach state i from state 0. Consider an epoch as the time between two visits to 0. The probability that an epoch would reach i is exactly 2^{-i}. The expected time of an epoch is 2 (the expected time to return to state 0). The expected time to return to state 0, given that we did not reach state i is less than 2. Therefore, $\mathbb{E}[T_i] \le 2 + 2 \cdot 2^i$.

Note that this Markov chain is aperiodic.

Invariant Distribution: The probability vector $\mu = (\mu_i)$ is an invariant distribution or stationary distribution for the Markov chain if $\mu P = \mu$, namely

$$\mu_j = \sum_i \mu_i p_{i,j} \quad \forall j.$$

Clearly, if $X_{\mathtt{t}} \sim \mu$ then $X_{\mathtt{t}+1} \sim \mu$. If $X_0 \sim \mu$, then the Markov chain $(X_{\mathtt{t}})$ is a stationary stochastic process.

Theorem 3.1. *Let $(X_{\mathtt{t}})$ be an irreducible and a-periodic Markov chain over a finite state space X, then there is a unique distribution μ such that $\mu^\top P = \mu^\top > 0$.*

Proof. Assume that x is an eigenvector of P with eigenvalue λ, i.e., we have $Px = \lambda x$. Since P is a stochastic matrix, we have $\|Px\|_\infty \leq \|x\|_\infty$, which implies that $\lambda \leq 1$. Since the matrix P is stochastic, $P\vec{1} = \vec{1}$, which implies that P has an eigenvalue of 1 and this is the maximal eigenvalue. Our first task is to show that there is such an x with $x \geq 0$.

Since the Markov chain is irreducible and a-periodic, there is an integer m, such that P^m has all the entries strictly positive. Namely, for any $i, j \in X$ we have $p_{i,j}^{(m)} > 0$.

We now show that if we have a general property of positive matrices (matrices where are the entries are strictly positive). Let $A = P^m$ be a positive matrix and x an eigenvector of A with eigenvalue 1. First, if x has complex number then $x' = Re(x)$ is an eigenvector of A of eigenvalue 1. Therefore we can assume that $x \in \mathbb{R}^d$. We would like to show that there is an $x \geq 0$ such that $x^\top A = x^\top$. If $x \geq 0$ we are done. If $x \leq 0$ we can take $x' = -x$ and we are done. We would like to show that x cannot have both positive and negative entries.

For contradiction, assume that we have $x_k > 0$ and $x_{k'} < 0$. This implies that for any weight vector $w > 0$ we have $|x^\top w| < |x|^\top w$, where $|x|$ is point-wise absolute value. Therefore,

$$\sum_j |x_j| = \sum_j |\sum_i x_i P_{i,j}| < \sum_j \sum_i |x_i| P_{i,j} = \sum_i |x_i| \sum_j P_{i,j} = \sum_j |x_j|$$

where the first identity follows since x is an eigenvector. The second since P is strictly positive.The third is a change of order of summation. The last follows since P is a stochastic matrix, so each row sums to 1. Clearly, we reached an contradiction, and therefore x cannot have both positive and negative entries.

We have shown so far that there exists a μ such that $\mu^\top P = \mu^\top$ and $\mu \geq 0$. Since $A = P^m$ is strictly positive, then $\mu^\top = \mu^\top A > 0$.

To show the uniqueness of μ, assume we have x and y such that $x^\top P = x^\top$ and $y^\top P = y^\top$ and $x \neq y$. Recall that we showed that both $x > 0$ and $y > 0$. Then there is a linear combination $z = ax + by$ such that for some i we have $z_i = 0$. Note that $z^\top P = z^\top$, but we showed that any such vector is strictly positive. Therefore, $x = y$, and hence μ is unique. $\square$

We define the average fraction that a state $j \in X$ occurs, given that we start with an initial state distribution x_0, as follows:

$$\pi_j^{(m)} = \frac{1}{m}\sum_{t=1}^{m} \mathbb{I}(X_t = j)$$

Theorem 3.2. *Let $(X_{\mathtt{t}})$ be an irreducible and a-periodic Markov chain over a finite state space X. Let μ be the stationary distribution of P. Then, for any $j \in X$ we have*

$$\mu_j = \lim_{m\to\infty} \pi_j^{(m)} = \frac{1}{\mathbb{E}[T_j]}$$

Proof. We have that

$$\mathbb{E}[\pi_j^{(m)}] = \mathbb{E}[\frac{1}{m}\sum_{t=1}^{m} \mathbb{I}(X_t = j)] = \frac{1}{m}\sum_{t=1}^{m} \Pr[X_t = j | X_0 = x_0] = \frac{1}{m}\sum_{t=1}^{m} x_0^\top P^t$$

Let $v_1, \ldots, v_n$ be the eigenvectors of P with eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$. We saw that $v_1 = \mu$, the stationary distribution and $\lambda_1 = 1 > \lambda_i$. Rewrite $x_0 = \sum_i alpha_i v_i$. Since P^m is a stochastic matrix, $x_0 P^m$ is a distribution, and therefore $\lim_{m\to\infty} x_0 P^m = \mu$.

We will be interested in the limit $\pi_j = \lim_{m\to\infty} \pi_j^m$, and mainly in the expected value $\mathbb{E}[\pi_j]$. From the above we have that $\mathbb{E}[\pi_j] = \mu_j$.

A different way to express $\mathbb{E}[\pi_j]$ is using a variable time horizon, with a fixe number of occurrences of j. Let $T_{k,j}$ be the time between the k and $k+1$ occurrence of state j. This implies that

$$\lim_{m\to\infty} \frac{1}{m}\sum_{t=1}^{m} \mathbb{I}(X_t = j) = \lim_{n\to\infty} \frac{n}{\sum_{k=1}^{n} T_{k,j}}$$

Note that the $T_{k,j}$ are i.i.d. as T_j. By the law of large numbers we have that $\frac{1}{n}\sum_{k=1}^{n} T_{k,j}$ converges to $\mathbb{E}[T_i]$. Therefore,

$$\mathbb{E}[\pi_j] = \frac{1}{\mathbb{E}[T_j]}$$

□

Theorem 3.3 (Recurrence of finite Markov chains). *Let $(X_{\mathtt{t}})$ be an irreducible, a-periodic Markov chain over a finite state space X. Then the following properties hold:*

1. *All states are* ***positive recurrent***

2. *There exists a* ***unique stationary distribution*** μ^*

3. ***Convergence*** *to the stationary distribution:* $\lim_{t\to\infty} p_{i,j}^{(t)} = \mu_j \quad (\forall j)$

4. ***Ergodicity****: For any finite* f*:* $\lim_{t\to\infty} \frac{1}{t}\sum_{s=0}^{\mathtt{t}-1} f(X_s) = \sum_i \mu_i f(i) \overset{\Delta}{=} \pi \cdot f$.

Proof. Item (1): It cannot be that all the states are transient, since then all the states will appear only a finite number of times, while the run is unbounded. This implies that all the states are recurrent. [Need to show *positive recurrent*: $\mathbb{E}[T_i] < \infty$]

Item (2): We will show that for each state i we have $\mu^*(i) = 1/\mathbb{E}[T_i]$. Let $Z_{i,k}$ be the time from the $k-1$ time

Item (3):

Item (4): □

For countable Markov chains, there are other possibilities.

Theorem 3.4 (**Countable Markov chains**). *Let* $(X_\mathtt{t})$ *be an irreducible and a-periodic Markov chain over a countable state space* X*. Then:*

1. *Either (i) all states are positive recurrent, or (ii) all states are null recurrent, or (iii) all states are transient.*

2. *If (i) holds, then properties (2)-(4) of the previous Theorem hold as well.*

3. *Conversely, if there exists a stationary distribution* μ *then properties (1)-(4) are satisfied.*

Proof. Item (1):

Item (2):

Item (3):

□

Reversible Markov chains: Suppose there exists a probability vector $\mu = (\mu_i)$ so that

$$\mu_i p_{i,j} = \mu_j p_{j,i}, \qquad i, j \in X. \tag{3.1}$$

It is then easy to verify by direct summation that μ is an invariant distribution for the Markov chain defined by $(p_{i,j})$. This follows since $\sum_i \mu_i p_{i,j} = \sum_i p_{i,j}\mu_j = \mu_j$. The equations (3.1) are called the *detailed balance equations.* A Markov chain that satisfies these equations is called reversible.

Example 3.5 (Discrete-time queue). *Consider a discrete-time queue, with queue length $X_{\mathtt{t}} \in \mathbb{N}_0 = \{0, 1, 2, \dots\}$. At time instant t, $A_{\mathtt{t}}$ new jobs arrive, and then up to $S_{\mathtt{t}}$ jobs can be served, so that*

$$X_{\mathtt{t}+1} = (X_{\mathtt{t}} + A_{\mathtt{t}} - S_{\mathtt{t}})^+.$$

Suppose that $(S_{\mathtt{t}})$ is a sequence of i.i.d. RVs, and similarly $(A_{\mathtt{t}})$ is a sequence of i.i.d. RVs, with $(S_{\mathtt{t}})$, $(A_{\mathtt{t}})$ and X_0 mutually independent. It may then be seen that $(X_{\mathtt{t}},\ t \geq 0)$ is a Markov chain. Suppose further that each $S_{\mathtt{t}}$ is a Bernoulli RV with parameter q, namely $P(S_{\mathtt{t}} = 1) = q$, $P(S_{\mathtt{t}} = 0) = 1 - q$. Similarly, let $A_{\mathtt{t}}$ be a Bernoulli RV with parameter p. Then

$$p_{i,j} = \begin{cases} p(1-q) & : \ j = i+1 \\ (1-p)(1-q) + pq & : \ j = i, \ \ i > 0 \\ (1-p)q & : \ j = i-1, \ \ i > 0 \\ (1-p) + pq & : \ j = i = 0 \\ 0 & : \ \text{otherwise} \end{cases}$$

Denote $\lambda = p(1-q)$, $\eta = (1-p)q$, and $\rho = \lambda/\eta$. The detailed balance equations for this case are:

$$\mu_i \lambda = \mu_{i+1} \eta, \qquad i \geq 0$$

These equations have a solution with $\sum_i \mu_i = 1$ if and only if $\rho < 1$. The solution is $\mu_i = \mu_0 \rho^i$, with $\pi_0 = 1 - \rho$. This is therefore the stationary distribution of this queue.

Chapter 4

Markov Decision Processes and Finite Horizon

In Chapter 2 we considered multi-stage decision problems for *deterministic* systems. In many problems of interest, the system dynamics also involves *randomness*, which leads us to stochastic decision problems. In this chapter we introduce the basic model of Markov Decision Processes, which will be considered in the rest of the book.

4.1 Controlled Markov Chains

A Markov Decision Process consists of two main parts:

1. A controlled dynamic system, with stochastic evolution.
2. A performance objective to be optimized.

In this section we describe the first part, which is modeled as a controlled Markov chain.

Consider a controlled dynamic system, defined over:

- A discrete time axis $\mathbb{T} = \{0, 1, \ldots, \mathtt{T} - 1\}$ (finite horizon), or $\mathbb{T} = \{0, 1, 2, \ldots\}$ (infinite horizon). To simplify the discussion we refer below to the infinite horizon case, which can always be "truncated" at $\mathtt{T}$ if needed.
- A finite state space $\mathcal{S}$, where $\mathcal{S}_{\mathtt{t}} \subset \mathcal{S}$ is the set of possible states at time $\mathtt{t}$.
- A finite action set $\mathcal{A}$, where $\mathcal{A}_{\mathtt{t}}(\mathtt{s}) \subset \mathcal{A}$ is the set of possible actions at time $\mathtt{t}$ and state $\mathtt{s} \in \mathcal{S}_{\mathtt{t}}$.

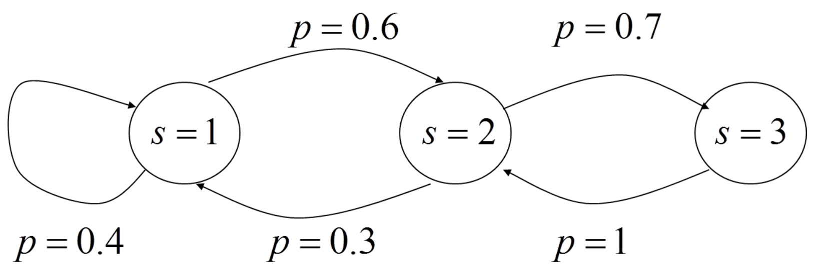


Figure 4.1: Markov chain

State transition probabilities:

- Suppose that at time $\mathtt{t}$ we are in state $\mathtt{s_t} = \mathtt{s}$, and choose an action $\mathtt{a_t} = \mathtt{a}$. The next state $\mathtt{s_{t+1}} = \mathtt{s'}$ is then determined randomly according to a probability distribution $p_{\mathtt{t}}(\cdot|\mathtt{s}, \mathtt{a})$ on $\mathcal{S}_{\mathtt{t+1}}$. That is,

$$\Pr(\mathtt{s_{t+1}} = \mathtt{s'}|\mathtt{s_t} = \mathtt{s}, \mathtt{a_t} = \mathtt{a}) = p_{\mathtt{t}}(\mathtt{s'}|\mathtt{s}, \mathtt{a}), \qquad \mathtt{s'} \in \mathcal{S}_{\mathtt{t+1}}$$

- The probability $p_{\mathtt{t}}(\mathtt{s'}|\mathtt{s}, \mathtt{a})$ is the *transition probability* from state $\mathtt{s}$ to state $\mathtt{s'}$ for a given action $\mathtt{a}$. We naturally require that $p_{\mathtt{t}}(\mathtt{s'}|\mathtt{s}, \mathtt{a}) \geq 0$, and $\sum_{\mathtt{s'} \in \mathcal{S}_{\mathtt{t+1}}} p_{\mathtt{t}}(\mathtt{s'}|\mathtt{s}, \mathtt{a}) = 1$ for all $\mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}_{\mathtt{t}}(\mathtt{s})$.

- Implicit in this definition is the controlled-Markov property:

$$\Pr(\mathtt{s_{t+1}} = \mathtt{s'}|\mathtt{s_t}, \mathtt{a_t}) = \Pr(\mathtt{s_{t+1}} = \mathtt{s'}|\mathtt{s_t}, \mathtt{a_t}, \ldots, \mathtt{s_0}, \mathtt{a_0})$$

- The set of probability distributions

$$P = \{p_{\mathtt{t}}(\cdot|\mathtt{s}, \mathtt{a}) \quad : \quad \mathtt{s} \in \mathcal{S}_{\mathtt{t}}, \mathtt{a} \in \mathcal{A}_{\mathtt{t}}(\mathtt{s}), \mathtt{t} \in \mathbb{T}\}$$

  is called the *transition law* or *transition kernel* of the controlled Markov process.

Stationary Models: The controlled Markov chain is called stationary or time-invariant if the transition probabilities do not depend on the time $\mathtt{t}$. That is:

$$\forall t, \quad p_{\mathtt{t}}(\mathtt{s'}|\mathtt{s}, \mathtt{a}) \equiv p(\mathtt{s'}|\mathtt{s}, \mathtt{a}), \;\; \mathcal{S}_{\mathtt{t}} \equiv \mathcal{S}, \;\; \mathcal{A}_{\mathtt{t}}(\mathtt{s}) \equiv \mathcal{A}(\mathtt{s}).$$

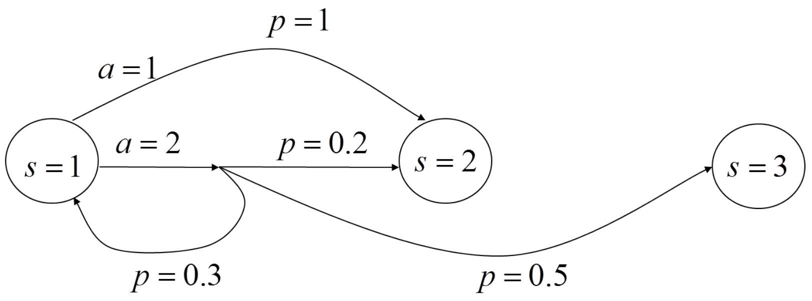


Figure 4.2: Controlled Markov chain

Graphical Notation: The state transition probabilities of a Markov chain are often illustrated via a state transition diagram, such as in Figure 4.1.

A graphical description of a controlled Markov chain is a bit more complicated because of the additional action variable. We obtain the diagram (drawn for state $s = 1$ only, and for a given time $\mathtt{t}$) in Figure 4.2, reflecting the following transition probabilities:

$$p(\mathtt{s}' = 2|\mathtt{s} = 1, \mathtt{a} = 1) = 1$$
$$p(\mathtt{s}'|\mathtt{s} = 1, \mathtt{a} = 2) = \begin{cases} 0.3 & : \quad \mathtt{s}' = 1 \\ 0.2 & : \quad \mathtt{s}' = 2 \\ 0.5 & : \quad \mathtt{s}' = 3 \end{cases}$$

State-equation notation: The stochastic state dynamics can be equivalently defined in terms of a state equation of the form

$$\mathtt{s}_{\mathtt{t}+1} = f_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}, w_{\mathtt{t}}),$$

where $w_{\mathtt{t}}$ is a random variable (RV). If $(w_{\mathtt{t}})_{t\geq 0}$ is a sequence of independent RVs, and further each $w_{\mathtt{t}}$ is independent of the "past" $(\mathtt{s}_{\mathtt{t}-1}, \mathtt{a}_{\mathtt{t}-1}, \ldots \mathtt{s}_0)$, then $(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})_{t\geq 0}$ is a controlled Markov process. For example, the state transition law of the last example can be written in this way, using $w_{\mathtt{t}} \in \{4, 5, 6\}$, with $p_w(4) = 0.3$, $p_w(5) = 0.2$, $p_w(6) = 0.5$ and, for $\mathtt{s}_{\mathtt{t}} = 1$:

$$f_{\mathtt{t}}(1, 1, w_{\mathtt{t}}) = 2$$
$$f_{\mathtt{t}}(1, 2, w_{\mathtt{t}}) = w_{\mathtt{t}} - 3$$

This state algebraic equation notation is especially useful for problems with continuous state space, but also for some models with discrete states. Equivalently, we can write

$$f_{\mathtt{t}}(1, 2, w_{\mathtt{t}}) = 1 \cdot \mathbb{I}[w_{\mathtt{t}} = 4] + 2 \cdot \mathbb{I}[w_{\mathtt{t}} = 5] + 3 \cdot \mathbb{I}[w_{\mathtt{t}} = 6],$$

where $\mathbb{I}[\cdot]$ is the indicator function.

Next we recall the definitions of control policies from Chapter 2.

Control Policies

- A general or **history-dependent** control policy $\pi = (\pi_{\mathtt{t}})_{\mathtt{t}\in\mathbb{T}}$ is a mapping from each possible history $\mathtt{h_t} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s_{t-1}}, \mathtt{a_{t-1}}, \mathtt{s_t})$, and time $\mathtt{t} \in \mathbb{T}$, to an action $\mathtt{a_t} = \pi_{\mathtt{t}}(\mathtt{h_t}) \in \mathcal{A}_{\mathtt{t}}$. We denote the set of general policies by Π_H.

- A **Markov** control policy π is allowed to depend on the current state and time only: $\mathtt{a_t} = \pi_{\mathtt{t}}(\mathtt{s_t})$. We denote the set of Markov policies by Π_M.

- For stationary models, we may define **stationary** control policies that depend on the current state alone. A stationary policy is defined by a single mapping $\pi : \mathcal{S} \to \mathcal{A}$, so that $\mathtt{a_t} = \pi(\mathtt{s_t})$ for all $\mathtt{t} \in \mathbb{T}$. We denote the set of stationary policies by Π_S.

- Evidently, $\Pi_H \supset \Pi_M \supset \Pi_S$.

Randomized Control policies

- The control policies defined above specify deterministically the action to be taken at each stage. In some cases we want to allow for a random choice of action.

- A general randomized control policy assigns to each possible history $\mathtt{h_t}$ a probability distribution $\pi_{\mathtt{t}}(\cdot|\mathtt{h_t})$ over the action set $\mathcal{A}_{\mathtt{t}}$. That is, $\Pr\{\mathtt{a_t} = \mathtt{a}|\mathtt{h_t}\} = \pi_{\mathtt{t}}(\mathtt{a}|\mathtt{h_t})$. We denote the set of history-dependent stochastic policies by Π_{HS}.

- Similarly, we can define the set Π_{MS} of Markov stochastic control policies, where $\pi_{\mathtt{t}}(\cdot|\mathtt{h_t})$ is replaced by $\pi_{\mathtt{t}}(\cdot|\mathtt{s_t})$, and the set Π_{SS} of stationary stochastic control policies, where $\pi_{\mathtt{t}}(\cdot|\mathtt{s_t})$ is replaced by $\pi(\cdot|\mathtt{s_t})$.

- Note that the set Π_{HS} includes all other policy sets as special cases.

The Induced Stochastic Process Let $p_0 = \{p_0(\mathtt{s}), \mathtt{s} \in \mathcal{S}_0\}$ be a probability distribution for the initial state $\mathtt{s}_0$. A control policy $\pi \in \Pi_{HS}$, together with the transition law $P = \{p_{\mathtt{t}}(\mathtt{s}'|\mathtt{s}, \mathtt{a})\}$ and the initial state distribution $p_0 = (p_0(\mathtt{s}),\ \mathtt{s} \in$

$\mathcal{S}_0$), induces a probability distribution over any finite state-action sequence $\mathtt{h}_\mathtt{T} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s}_{\mathtt{T}-1}, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_\mathtt{T})$, given by

$$\Pr(\mathtt{h}_\mathtt{T}) = p_0(\mathtt{s}_0) \prod_{t=0}^{\mathtt{T}-1} p_\mathtt{t}(\mathtt{s}_{\mathtt{t}+1}|\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) \pi_\mathtt{t}(\mathtt{a}_\mathtt{t}|\mathtt{h}_\mathtt{t}),$$

where $\mathtt{h}_\mathtt{t} = (\mathtt{s}_0, \mathtt{a}_0, \ldots, \mathtt{s}_{\mathtt{T}-1}, \mathtt{a}_{\mathtt{T}-1}, \mathtt{s}_\mathtt{t})$. To see this, observe the recursive relation:

$$\begin{aligned}\Pr(\mathtt{h}_{\mathtt{t}+1}) &= \Pr(\mathtt{h}_\mathtt{t}, \mathtt{a}_\mathtt{t}, \mathtt{s}_{\mathtt{t}+1}) = \Pr(\mathtt{s}_{\mathtt{t}+1}|\mathtt{h}_\mathtt{t}, \mathtt{a}_\mathtt{t}) \Pr(\mathtt{a}_\mathtt{t}|\mathtt{h}_\mathtt{t}) \Pr(\mathtt{h}_\mathtt{t}) \\ &= p_\mathtt{t}(\mathtt{s}_{\mathtt{t}+1}|\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) \pi_\mathtt{t}(\mathtt{a}_\mathtt{t}|\mathtt{h}_\mathtt{t}) \Pr(\mathtt{h}_\mathtt{t}).\end{aligned}$$

In the last step we used the conditional Markov property of the controlled chain: $\Pr(\mathtt{s}_{\mathtt{t}+1}|\mathtt{h}_\mathtt{t}, \mathtt{a}_\mathtt{t}) = p_\mathtt{t}(\mathtt{s}_{\mathtt{t}+1}|\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$, and the definition of the control policy π. The required formula follows by recursion.

Therefore, the state-action sequence $\mathtt{h}_\infty = (\mathtt{s}_k, \mathtt{a}_k)_{k \geq 0}$ can now be considered a stochastic process. We denote the probability law of this stochastic process by $\Pr^{\pi,p_0}(\cdot)$. The corresponding expectation operator is denoted by $\mathbb{E}^{\pi,p_0}(\cdot)$. When the initial state $\mathtt{s}_0$ is deterministic (i.e., $p_0(\mathtt{s})$ is concentrated on a single state $\mathtt{s}$), we may simply write $\Pr^{\pi,s}(\cdot)$ or $\Pr^{\pi}(\cdot|\mathtt{s}_0 = \mathtt{s})$.

Under a Markov control policy, the state sequence $(\mathtt{s}_\mathtt{t})_{\mathtt{t}\geq 0}$ becomes a *Markov chain*, with transition probabilities

$$\Pr(\mathtt{s}_{\mathtt{t}+1} = \mathtt{s}'|\mathtt{s}_\mathtt{t} = \mathtt{s}) = \sum\nolimits_{\mathtt{a} \in \mathcal{A}_\mathtt{t}} p_\mathtt{t}(\mathtt{s}'|\mathtt{s}, \mathtt{a}) \pi_\mathtt{t}(\mathtt{a}|\mathtt{s}).$$

This follows since:

$$\Pr(\mathtt{s}_{\mathtt{t}+1} = \mathtt{s}'|\mathtt{s}_\mathtt{t} = \mathtt{s}) = \sum\nolimits_{\mathtt{a} \in \mathcal{A}_\mathtt{t}} \Pr(\mathtt{s}_{\mathtt{t}+1} = \mathtt{s}', \mathtt{a}|\mathtt{s}_\mathtt{t} = \mathtt{s})$$

$$= \sum\nolimits_{\mathtt{a} \in \mathcal{A}_\mathtt{t}} \Pr(\mathtt{s}_{\mathtt{t}+1} = \mathtt{s}'|\mathtt{s}_\mathtt{t} = \mathtt{s}) \Pr(\mathtt{a}|\mathtt{s}_\mathtt{t} = \mathtt{s}) = \sum\nolimits_{\mathtt{a} \in \mathcal{A}_\mathtt{t}} p_\mathtt{t}(\mathtt{s}'|\mathtt{s}, \mathtt{a}) \pi_\mathtt{t}(\mathtt{a}|\mathtt{s})$$

If the controlled Markov chain is stationary (time-invariant) and the control policy is stationary, then the induced Markov chain is stationary as well.

Remark 4.1. *For most non-learning optimization problems, Markov policies suffice to achieve the optimum.*

Remark 4.2. *Implicit in these definitions of control policies is the assumption that the current state* $\mathtt{s}_\mathtt{t}$ *can be fully observed before the action* $\mathtt{a}_\mathtt{t}$ *is chosen . If this is not the case we need to consider the problem of a Partially Observed MDP (POMDP), which is more involved and will be discussed in Chapter 16.*

4.2 Performance Criteria

4.2.1 Finite Horizon Problems

Consider the finite-horizon problem, with a fixed time horizon $\mathtt{T}$. As in the deterministic case, we are given a running reward function $\mathtt{r_t} = \{\mathtt{r_t(s,a)} : \mathtt{s} \in \mathcal{S}_\mathtt{t}, \mathtt{a} \in \mathcal{A}_\mathtt{t}\}$ for $0 \leq \mathtt{t} \leq \mathtt{T}-1$, and a terminal reward function $\mathtt{r_T} = \{\mathtt{r_T(s)} : \mathtt{s} \in \mathcal{S}_\mathtt{T}\}$. The obtained reward are $\mathtt{t}$ is $R_\mathtt{t} = \mathtt{r_t(s_t,a_t)}$ at times $\mathtt{t} \leq \mathtt{T}-1$, and $R_\mathtt{T} = \mathtt{r_T(s_T)}$ at the last stage. (Note that $\mathtt{s_t}, \mathtt{a_t}$ and $\mathtt{s_T}$ are random variables that depend both on the policy π and the stochastic transitions.) Our general goal is to maximize the cumulative return:

$$\sum_{t=0}^{\mathtt{T}} R_\mathtt{t} = \sum_{t=0}^{\mathtt{T}-1} \mathtt{r_t(s_t,a_t)} + \mathtt{r_T(s_T)}.$$

However, since the system is stochastic, the cumulative return will generally be a random variable, and we need to specify in which sense to maximize it. A natural first option is to consider the expected value of the return. That is, define:

$$\mathcal{V}_\mathtt{T}^\pi(\mathtt{s}) = \mathbb{E}^\pi(\sum_{t=0}^{\mathtt{T}} R_\mathtt{t} | \mathtt{s}_0 = \mathtt{s}) \equiv \mathbb{E}^{\pi,s}(\sum_{t=0}^{\mathtt{T}} R_\mathtt{t}).$$

Here π is the control policy as defined above, and $\mathtt{s}$ denotes the initial state. Hence, $\mathcal{V}_\mathtt{T}^\pi(\mathtt{s})$ is the expected cumulative return under the control policy π. Our goal is to find an optimal control policy that maximized $\mathcal{V}_\mathtt{T}^\pi(\mathtt{s})$.

Remarks:

1. Dependence on the next state: In some problems, the obtained reward may depend on the next state as well: $R_\mathtt{t} = \tilde{\mathtt{r}}_\mathtt{t}(\mathtt{s_t}, \mathtt{a_t}, \mathtt{s_{t+1}})$. For control purposes, when we only consider the expected value of the reward, we can reduce this reward function to the usual one by defining

$$\mathtt{r_t(s,a)} \overset{\Delta}{=} E(R_\mathtt{t} | \mathtt{s_t} = \mathtt{s}, \mathtt{a_t} = \mathtt{a}) \equiv \sum\nolimits_{\mathtt{s}' \in S} p(\mathtt{s}'|\mathtt{s},\mathtt{a}) \tilde{r}_\mathtt{t}(s,a,\mathtt{s}')$$

2. Random rewards: The reward $R_\mathtt{t}$ may also be random, namely a random variable whose distribution depends on $(\mathtt{s_t}, \mathtt{a_t})$. This can also be reduced to our standard model for planning purposes by looking at the expected value of $R_\mathtt{t}$, namely

$$\mathtt{r_t(s,a)} = E(R_\mathtt{t} | \mathtt{s_t} = \mathtt{s}, \mathtt{a_t} = \mathtt{a}).$$

3. Risk-sensitive criteria: The expected cumulative return is by far the most common goal for planning. However, it is not the only one possible. For example, one may consider the following risk-sensitive return function:

   $$\mathcal{V}^{\pi}_{\mathtt{T},\lambda}(\mathtt{s}) = \frac{1}{\lambda}\log \mathbb{E}^{\pi,s}(\exp(\lambda \sum_{t=0}^{\mathtt{T}} R_{\mathtt{t}})).$$

   For $\lambda > 0$, the exponent gives higher weight to high rewards, and the opposite for $\lambda < 0$.

In the case that the rewards are stochastic, but have a discrete support, we can construct an equivalent MDP in which all the rewards are deterministic and has the same return. This implies that the important challenge is the stochastic state transition function, and the rewards can be assumed to be deterministic.

[Add HW to do the same for discounted]

Theorem 4.1. *Given an MDP $M(\mathcal{S},\mathcal{A},p,\mathtt{r},\mathtt{s}_0)$, where the rewards are stochastic, with support $\{1,\ldots,k\}$, there is an MDP $M_1(\mathcal{S}_1,\mathcal{A},p_1,\mathtt{r},s_1)$, and a mapping of policies π of M to π_1 policies of M_1, such that $\mathcal{V}^{\pi,M}_{\mathtt{T}}(\mathtt{s}_0) = \mathcal{V}^{\pi_1,M_1}_{\mathtt{T}+1}$*

Proof. The basic idea is to encode the rewards in the states of M'. Let $\mathcal{K} = \{1,\ldots,k\}$. We define a new state space $\mathcal{S}' = \mathcal{S}\times\mathcal{K}$. For each $(\mathtt{s},i)\in\mathcal{S}'$ and action $\mathtt{a}\in\mathcal{A}$ we have $p'((\mathtt{s}',j)|(\mathtt{s},i),\mathtt{a}) = p(\mathtt{s}'|\mathtt{s},\mathtt{a})\Pr[R(\mathtt{s},\mathtt{a})=j]$. The reward is $\mathtt{r}_1((\mathtt{s},i),\mathtt{a}) = i$. The initial state is $s_1 = (\mathtt{s}_0,0)$.

For any policy $\pi(\mathtt{a}|\mathtt{s})$ in M we have a policy π_1 is M_1 define as follows: $\pi_1(\mathtt{a}|(\mathtt{s},i)) = \pi(\mathtt{a}|\mathtt{s})$.

We can map trajectories in M to trajectories in M_1 by simply having $\mathtt{s}_0,\mathtt{a}_0,R_0,\mathtt{s}_1,\mathtt{a}_1,R_1,\mathtt{s}_2\ldots$ to $(\mathtt{s}_0,0),\mathtt{a}_0,0,(\mathtt{s}_1,R_0),\mathtt{a}_1,R_0,(\mathtt{s}_2,R_1)\ldots$ etc. This implies that the rewards are identical (up to a shift of one in the index). Therefore the policies have identical return.l □

4.2.2 Infinite Horizon Problems

We next consider planning problems that extend to an unlimited time horizon, $\mathtt{t} = 0,1,2,\ldots$. Such planning problems arise when the system in question is expected to operate for a long time, or a large number of steps, possibly with no specific "closing" time. Infinite horizon problems are most often defined for stationary problems. In that case, they enjoy the important advantage that optimal policies can be found among the class of stationary policies. We will restrict attention here to stationary models. As before, we have the running reward function $\mathtt{r}(\mathtt{s},\mathtt{a})$, which extends to all $\mathtt{t}\geq 0$. The expected reward obtained at stage $\mathtt{t}$ is $\mathbb{E}[R_{\mathtt{t}}] = \mathtt{r}(\mathtt{s}_{\mathtt{t}},\mathtt{a}_{\mathtt{t}})$.

Discounted return: The most common performance criterion for infinite horizon problems is the expected discounted return:

$$\mathcal{V}_\gamma^\pi(\mathtt{s}) = \mathbb{E}^\pi(\sum_{t=0}^{\infty} \gamma^t \mathtt{r}(\mathtt{s_t}, \mathtt{a_t}) | \mathtt{s}_0 = \mathtt{s}) \equiv \mathbb{E}^{\pi,s}(\sum_{t=0}^{\infty} \gamma^t \mathtt{r}(\mathtt{s_t}, \mathtt{a_t}))$$

Here $0 < \gamma < 1$ is the discount factor. Mathematically, the discount factor ensures convergence of the sum (whenever the reward sequence is bounded). This make the problem "well behaved", and relatively easy to analyze. The discounted return is discussed in Chapter 5.

Average return: Here we are interested to maximize the long-term average return. The most common definition of the long-term average return is

$$\mathcal{V}_{av}^\pi(\mathtt{s}) = \liminf_{\mathtt{T}\to\infty} \mathbb{E}^{\pi,s}(\frac{1}{\mathtt{T}} \sum_{t=0}^{\mathtt{T}-1} \mathtt{r}(\mathtt{s_t}, \mathtt{a_t}))$$

The theory of average-return planning problems is more involved, and relies to a larger extent on the theory of Markov chains. The average return is discussed in Chapter 7.

4.2.3 Stochastic Shortest-Path Problems

In an important class of planning problems, the time horizon is not set beforehand, but rather the problem continues until a certain event occurs. This event can be defined as reaching some goal state. Let $\mathcal{S}_G \subset \mathcal{S}$ define the set of *goal states*. Define

$$\tau = \inf\{t \geq 0 : \mathtt{s_t} \in \mathcal{S}_G\}$$

as the first time in which a goal state is reached. The total expected return for this problem is defined as:

$$\mathcal{V}_{ssp}^\pi(\mathtt{s}) = \mathbb{E}^{\pi,\mathtt{s}}(\sum_{\mathtt{t}=0}^{\tau-1} \mathtt{r}(\mathtt{s_t}, \mathtt{a_t}) + \mathtt{r}_G(\mathtt{s}_\tau))$$

Here $\mathtt{r}_G(\mathtt{s})$, $\mathtt{s} \in \mathcal{S}_G$ specified the reward at goal states. Note that the length of the run τ is a random variable.

Stochastic shortest path includes, naturally, the finite horizon case. This can be shown by creating a leveled MDP where at each time step we move to the next level and terminate at level $\mathtt{T}$. Specifically, we define a new state space $\mathcal{S}' = \mathcal{S} \times \mathbb{T}$, transition function $p((\mathtt{s}', i+1)|\mathtt{s}, \mathtt{a}) = p(\mathtt{s}'|\mathtt{s}, \mathtt{a})$, and goal states $\mathcal{S}_G = \{(\mathtt{s}, \mathtt{T}) : \mathtt{s} \in \mathcal{S}\}$.

Stochastic shortest path includes also the discounted infinite horizon. To see that, add a new goal state, and from each state with probability $1-\gamma$ jump to the goal state and terminate. The expected return of a policy would be the same in both models. Specifically, we add a state $\mathtt{s}_G$, such that $p(\mathtt{s}_G|\mathtt{s}, \mathtt{a}) = 1-\gamma$, for any state $\mathtt{s} \in \mathcal{S}$ and action $\mathtt{a} \in \mathcal{A}$ and $p(\mathtt{s}'|\mathtt{s}, \mathtt{a}) = \gamma p(\mathtt{s}'|\mathtt{s}, \mathtt{a})$. The probability that we do not terminate by time $\mathtt{t}$ is exactly $\gamma^{\mathtt{t}}$. Therefore the expected return is $\sum_{t=1}^{\infty} \gamma^t \mathtt{r}(\mathtt{s}_t, \mathtt{a}_t)$ which is identical to the discounted return.

This class of problems provides a natural extension of the standard shortest-path problem to stochastic settings. Some conditions on the system dynamics and reward function must be imposed for the problem to be well posed (e.g., that a goal state may be reached with probability one). Such problems are known as stochastic shortest path problems, or also episodic planning problems. See more in chapter 6.

4.3 *Sufficiency of Markov Policies

In all the performance criteria defined above, the criterion is composed of sums of terms of the form $\mathbb{E}(\mathtt{r}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}))$. It follows that if two control policies induce the same marginal probability distributions $q_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})$ over the state-action pairs $(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})$ for all $t \geq 0$, they will have the same performance.

Using this observation, the next claim implies that it is enough to consider the set of (stochastic) Markov policies in the above planning problems.

Proposition 4.1. *Let $\pi \in \Pi_{HS}$ be a general (history-dependent, stochastic) control policy. Let*

$$p_{\mathtt{t}}^{\pi,\mathtt{s}_0}(\mathtt{s}, \mathtt{a}) = P^{\pi,\mathtt{s}_0}(\mathtt{s}_{\mathtt{t}} = \mathtt{s}, \mathtt{a}_{\mathtt{t}} = \mathtt{a}), \qquad (\mathtt{s}, \mathtt{a}) \in \mathcal{S}_{\mathtt{t}} \times \mathcal{A}_{\mathtt{t}}$$

Denote the marginal distributions induced by $q_t(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})$ on the state-action pairs $(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}})$, for all $\mathtt{t} \geq 0$. Then there exists a stochastic Markov policy $\tilde{\pi} \in \Pi_{MS}$ that induces the same marginal probabilities (for all initial states $\mathtt{s}_0$).

In Chapter 2 we showed for Deterministic Decision Process for the finite horizon that there is an optimal deterministic policy. The proof that every stochastic history dependent strategy has an equivalent stochastic Markovian policy (Theorem 2.1) showed how to generate the same state-action distribution, and applies to other setting as well. The proof that every stochastic Markovian policy has a deterministic

Markovian policy (Theorem 2.2) depended on the finite horizon, but it is easy to extend it to other settings as well.

4.4 Finite-Horizon Dynamic Programming

Recall that we consider the expected total reward criterion, which we denote as

$$\mathcal{V}^{\pi}(\mathtt{s}_0) = \mathbb{E}^{\pi,\mathtt{s}_0}\left(\sum\nolimits_{\mathtt{t}=0}^{\mathtt{T}-1} \mathtt{r}_{\mathtt{t}}(\mathtt{s}_{\mathtt{t}}, \mathtt{a}_{\mathtt{t}}) + \mathtt{r}_{\mathtt{T}}(\mathtt{s}_{\mathtt{T}})\right)$$

Here π is the control policy used, and $\mathtt{s}_0$ is a given initial state. We wish to maximize $\mathcal{V}^{\pi}(\mathtt{s}_0)$ over all control policies, and find an optimal policy π^* that achieves the maximal reward $\mathcal{V}^*(\mathtt{s}_0)$ for all initial states $\mathtt{s}_0$. Thus,

$$\mathcal{V}^*_{\mathtt{T}}(\mathtt{s}_0) \triangleq \mathcal{V}^{\pi^*}_{\mathtt{T}}(\mathtt{s}_0) = \max_{\pi \in \Pi_{HS}} \mathcal{V}^{\pi}_{\mathtt{T}}(\mathtt{s}_0)$$

4.4.1 The Principle of Optimality

The celebrated principle of optimality (stated by Bellman) applies to a large class of multi-stage optimization problems, and is at the heart of Dynamic Programming. As a general principle, it states that:

The tail of an optimal policy is optimal for the "tail" problem.

This principle is not an actual claim, but rather a guiding principle that can be applied in different ways to each problem. For example, considering our finite-horizon problem, let $\pi^* = (\pi_0, \ldots, \pi_{\mathtt{T}-1})$ denote an optimal Markov policy. Take any state $\mathtt{s}_{\mathtt{t}} = \mathtt{s}'$ which has a positive probability to be reached under π^*, namely $P^{\pi^*,\mathtt{s}_0}(\mathtt{s}_{\mathtt{t}} = \mathtt{s}') > 0$. Then the tail policy $\pi^*_{t:T} = (\pi_{\mathtt{t}}, \ldots, \pi_{\mathtt{T}-1})$ is optimal for the "tail" criterion $\mathcal{V}^{\pi}_{t:T}(\mathtt{s}') = \mathbb{E}^{\pi}\left(\sum_{k=t}^{\mathtt{T}} R_k | \mathtt{s}_{\mathtt{t}} = \mathtt{s}'\right)$.

Note that the reverse is not true. The prefix of the optimal policy is not optimal for the "prefix" problem. When we plan for a long horizon, we might start with non-greedy actions, so we can improve our return in later time steps. Specifically, the first action taken does not has to be the optimal action for horizon $\mathtt{T} = 1$, which is the greedy action.

4.4.2 Dynamic Programming for Policy Evaluation

As a "warmup", let us evaluate the reward of a given policy. Let $\pi = (\pi_0, \ldots, \pi_{\mathtt{T}-1})$ be a given Markov policy. Define the following reward-to-go function, or value function:

$$V^{\pi}_k(\mathtt{s}) = \mathbb{E}^{\pi}\left(\sum\nolimits_{\mathtt{t}=k}^{\mathtt{T}} R_{\mathtt{t}} | \mathtt{s}_k = \mathtt{s}\right)$$

Observe that $V_0^\pi(\mathtt{s}_0) = \mathcal{V}^\pi(\mathtt{s}_0)$.

Lemma 4.1 (Value Iteration). $V_k^\pi(\mathtt{s})$ *may be computed by the backward recursion:*

$$V_k^\pi(\mathtt{s}) = \left\{ \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\; V_{k+1}^\pi(\mathtt{s}') \right\}_{a=\pi_k(\mathtt{s})}, \quad \mathtt{s} \in \mathcal{S}_k$$

for $k = \mathtt{T}-1, \ldots, 0$*, starting with* $V_\mathtt{T}^\pi(\mathtt{s}) = \mathtt{r}_\mathtt{T}(\mathtt{s})$.

Proof. Observe that:

$$\begin{aligned}
V_k^\pi(\mathtt{s}) &= \mathbb{E}^\pi\left(R_k + \sum\nolimits_{\mathtt{t}=k+1}^{\mathtt{T}} R_\mathtt{t} |\; \mathtt{s}_k = \mathtt{s}, \mathtt{a}_k = \pi_k(\mathtt{s}) \right) \\
&= \mathbb{E}^\pi\left(\mathbb{E}^\pi\left(R_k + \sum\nolimits_{\mathtt{t}=k+1}^{\mathtt{T}} R_\mathtt{t} |\; \mathtt{s}_k = \mathtt{s}, \mathtt{a}_k = \pi_k(\mathtt{s}), \mathtt{s}_{k+1} \right) |\mathtt{s}_k = \mathtt{s}, \mathtt{a}_k = \pi_k(\mathtt{s}) \right) \\
&= \mathbb{E}^\pi\left(\mathtt{r}_k(\mathtt{s}_k, \mathtt{a}_k) + V_{k+1}^\pi(\mathtt{s}_{k+1}) | \mathtt{s}_k = \mathtt{s}, \mathtt{a}_k = \pi_k(\mathtt{s}) \right) \\
&= \mathtt{r}_k(s, \pi_k(\mathtt{s})) + \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|s, \pi_k(\mathtt{s}))\; V_{k+1}^\pi(\mathtt{s}')
\end{aligned}$$

The first identity is simply writing the value function explicitly, starting at state $\mathtt{s}$ at time k and using action $\pi(\mathtt{s})$. We split the sum to R_k, immediate reward, and the sum of other rewards, which will be convinient later.The second identity uses the law of total probability, we are conditioning on state $\mathtt{s}_{k+1}$, and taking the expectation over it. The third identity observes that the expected value of the sum is actually the value function at $\mathtt{s}_{k+1}$. The last identity writes the expectation over $\mathtt{s}_{k+1}$ explicitly. □

Remarks:

- Note that $\sum_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\; V_{k+1}^\pi(\mathtt{s}') = \mathbb{E}^\pi(V_{k+1}^\pi(\mathtt{s}_{k+1}) | \mathtt{s}_k = \mathtt{s}, \mathtt{a}_k = \mathtt{a})$.

- For the more general reward function $\tilde{\mathtt{r}}_\mathtt{t}(\mathtt{s},\mathtt{a},\mathtt{s}')$, the recursion takes the form

$$V_k^\pi(\mathtt{s}) = \left\{ \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a}) [\tilde{\mathtt{r}}_k(\mathtt{s},\mathtt{a},\mathtt{s}') + V_{k+1}^\pi(\mathtt{s}')] \right\}_{\mathtt{a}=\pi_k(\mathtt{s})}$$

A similar observation applies to all Dynamic Programming equations below.

4.4.3 Dynamic Programming for Policy Optimization

We next define the optimal value function at each time $k \geq 0$:

$$V_k(\mathtt{s}) = \max_{\pi^k} \mathbb{E}^{\pi^k} \left(\sum\nolimits_{\mathtt{t}=k}^{\mathtt{T}} R_{\mathtt{t}} | \mathtt{s}_k = \mathtt{s} \right), \quad \mathtt{s} \in \mathcal{S}_k$$

The maximum is taken over "tail" policies $\pi^k = (\pi_k, \ldots, \pi_{\mathtt{T}-1})$ that start from time k. Note that π^k is allowed to be a general policy, i.e., history-dependent and stochastic. Obviously, $V_0(\mathtt{s}_0) = \mathcal{V}^*(\mathtt{s}_0)$.

Theorem 4.2 (**Finite-horizon Dynamic Programming**). *The following holds:*

1. *Backward recursion: Set* $V_{\mathtt{T}}(\mathtt{s}) = \mathtt{r}_{\mathtt{T}}(\mathtt{s})$ *for* $\mathtt{s} \in \mathcal{S}_{\mathtt{T}}$.
   For $k = \mathtt{T} - 1, \ldots, 0$, $V_k(\mathtt{s})$ *may be computed using the following recursion:*

$$V_k(\mathtt{s}) = \max_{\mathtt{a} \in \mathcal{A}_k} \left\{ \mathtt{r}_k(\mathtt{s}, \mathtt{a}) + \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s}, \mathtt{a}) \, V_{k+1}(\mathtt{s}') \right\}, \quad \mathtt{s} \in \mathcal{S}_k.$$

2. *Optimal policy: Any Markov policy* π^* *that satisfies, for* $\mathtt{t} = 0, \ldots, \mathtt{T} - 1$,

$$\pi^*_{\mathtt{t}}(\mathtt{s}) \in \arg\max_{\mathtt{a} \in \mathcal{A}_{\mathtt{t}}} \left\{ \mathtt{r}_{\mathtt{t}}(\mathtt{s}, \mathtt{a}) + \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{\mathtt{t}+1}} p_{\mathtt{t}}(\mathtt{s}'|\mathtt{s}, \mathtt{a}) \, V_{\mathtt{t}+1}(\mathtt{s}') \right\}, \quad \mathtt{s} \in \mathcal{S}_{\mathtt{t}},$$

   is an optimal control policy. Furthermore, π^* *maximizes* $\mathcal{V}^\pi(\mathtt{s}_0)$ *simultaneously for every initial state* $\mathtt{s}_0 \in \mathcal{S}_0$.

Note that Theorem 4.2 specifies an optimal control policy which is a deterministic Markov policy.

Proof. **Part (i):**

We use induction to show that the stated backward recursion indeed yields the optimal value function. The idea is simple, but some care is needed with the notation since we consider general policies, and not just Markov policies. The equality $V_{\mathtt{T}}(\mathtt{s}) = \mathtt{r}_{\mathtt{T}}(\mathtt{s})$ follows directly from the definition of $V_{\mathtt{T}}$.

We proceed by backward induction. Suppose that $V_{k+1}(\mathtt{s})$ is the optimal value function for time $k + 1$. We need to show that $V_k(\mathtt{s}) = W_k(\mathtt{s})$, where

$$W_k(\mathtt{s}) \triangleq \max_{\mathtt{a} \in \mathcal{A}_k} \left\{ \mathtt{r}_k(\mathtt{s}, \mathtt{a}) + \sum\nolimits_{\mathtt{s}' \in \mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s}, \mathtt{a}) \, V_{k+1}(\mathtt{s}') \right\}.$$

We will first establish that $V_k(\mathtt{s}) \geq W_k(\mathtt{s})$, and then that $V_k(\mathtt{s}) \leq W_k(\mathtt{s})$.

(a) We first show that $V_k(\mathtt{s}) \geq W_k(\mathtt{s})$. For that purpose, it is enough to find a policy π^k so that $V_k^{\pi^k}(\mathtt{s}) = W_k(\mathtt{s})$.

Fix $\mathtt{s} \in \mathcal{S}_k$, and define π^k as follows: Choose $\mathtt{a}_k = \bar{\mathtt{a}}$, where

$$\bar{a} \in \arg\max_{\mathtt{a}\in\mathcal{A}_k} \left\{ \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}'\in\mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\, V_{k+1}(\mathtt{s}') \right\},$$

and then, after observing $\mathtt{s}_{k+1} = \mathtt{s}'$, proceed with the optimal tail policy $\pi^{k+1}(\mathtt{s}')$ that obtains $V_{k+1}^{\pi^{k+1}(\mathtt{s}')}(\mathtt{s}') = V_{k+1}(\mathtt{s}')$. Proceeding similarly to Subsection 4.4.2 above (value iteration for a fixed policy), we obtain:

$$V_k^{\pi^k}(\mathtt{s}) = \mathtt{r}_k(\mathtt{s},\bar{\mathtt{a}}) + \sum\nolimits_{\mathtt{s}'\in\mathcal{S}_{k+1}} p(\mathtt{s}'|\mathtt{s},\bar{\mathtt{a}})\, V_{k+1}^{\pi^{k+1}(\mathtt{s}')}(\mathtt{s}') \tag{4.1}$$

$$= \mathtt{r}_k(\mathtt{s},\bar{\mathtt{a}}) + \sum\nolimits_{\mathtt{s}'\in\mathcal{S}_{k+1}} p(\mathtt{s}'|\mathtt{s},\bar{\mathtt{a}})\, V_{k+1}(\mathtt{s}') = W_k(\mathtt{s}), \tag{4.2}$$

as was required.

(b) To establish $V_k(\mathtt{s}) \leq W_k(\mathtt{s})$, it is enough to show that $V_k^{\pi^k}(\mathtt{s}) \leq W_k(\mathtt{s})$ for any (general, randomized) "tail" policy π^k.

Fix $\mathtt{s} \in \mathcal{S}_k$. Consider then some tail policy $\pi^k = (\pi_k, \ldots \pi_{\mathtt{T}-1})$. Note that this means that $\mathtt{a}_\mathtt{t} \sim \pi_\mathtt{t}(\mathtt{a}|\mathtt{h}_{k:t})$, where $\mathtt{h}_{k:t} = (\mathtt{s}_k, \mathtt{a}_k, \mathtt{s}_{k+1}, \mathtt{a}_{k+1}, \ldots, \mathtt{s}_\mathtt{t})$. For each state-action pair $\mathtt{s} \in \mathcal{S}_k$ and $\mathtt{a} \in \mathcal{A}_k$, let $(\pi^k|\mathtt{s},\mathtt{a})$ denote the tail policy π^{k+1} from time $k+1$ onwards which is obtained from π^k given that $\mathtt{s}_k = \mathtt{s}$, $\mathtt{a}_k = \mathtt{a}$. As before, by value iteration for a fixed policy,

$$V_k^{\pi^k}(\mathtt{s}) = \sum\nolimits_{\mathtt{a}\in\mathcal{A}_k} \pi_k(\mathtt{a}|\mathtt{s}) \left\{ \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}'\in\mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\, V_{k+1}^{(\pi^k|\mathtt{s},\mathtt{a})}(\mathtt{s}') \right\}.$$

But since V_{k+1} is optimal,

$$V_k^{\pi^k}(\mathtt{s}) \leq \sum\nolimits_{\mathtt{a}\in\mathcal{A}_k} \pi_k(\mathtt{a}|\mathtt{s}) \left\{ \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}'\in\mathcal{S}_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\, V_{k+1}(\mathtt{s}') \right\}$$

$$\leq \max_{\mathtt{a}\in\mathcal{A}_k} \left\{ \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}'\in S_{k+1}} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\, V_{k+1}(\mathtt{s}') \right\} = W_k(\mathtt{s}),$$

which is the required inequality in (b).

Part (ii) (Outline - **exercise**):

Let π^* be the (Markov) policy defined in part 2 of Theorem 4.2. Consider the value iteration (Lemma 4.1). The updates for $V_k^{\pi^*}$ in the value iteration, given the action selection of π^*, are identical to those of V_k. This implies that $V_k^{\pi^*} = V_k$ (formally, by induction of k). Since V_k is the optimal value function, it implies that π^* is the optimal policy.

[[Should we be more formal?]] □

4.4.4 The Q function

Let

$$Q_k(\mathtt{s},\mathtt{a}) \triangleq \mathtt{r}_k(\mathtt{s},\mathtt{a}) + \sum\nolimits_{\mathtt{s}'\in S_k} p_k(\mathtt{s}'|\mathtt{s},\mathtt{a})\, V_k(\mathtt{s}').$$

This is known as the optimal state-action value function, or simply as the *Q-function.* $Q_k(\mathtt{s},\mathtt{a})$ is the expected return from stage k onward, if we choose $\mathtt{a}_k = \mathtt{a}$ and then proceed optimally.

Theorem 4.2 can now be succinctly expressed as

$$V_k(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}_k} Q_k(\mathtt{s},\mathtt{a}),$$

and

$$\pi_k^*(\mathtt{s}) \in \arg\max_{\mathtt{a}\in\mathcal{A}_k} Q_k(\mathtt{s},\mathtt{a}).$$

The Q function provides the basis for the Q-learning algorithm, which is one of the basic Reinforcement Learning algorithms.

4.5 Linear Program

[Good idea to add to get them familiar with the notion and notation for the very simple case.]

To summarize:

- The optimal value function can be computed by backward recursion. This recursive equation is known as the *dynamic programming equation*, *optimality equation*, or *Bellman's Equation.*
- Computation of the value function in this way is known as the *finite-horizon value iteration* algorithm.
- The value function is computed for all states at each stage.
- An optimal policy is easily derived from the optimal value.
- The optimization in each stage is performed in the action space. The total number of minimization operations needed is $\mathtt{T} \times |\mathcal{S}|$ - each over $|\mathcal{A}|$ choices. This replaces "brute force" optimization in policy space, with tremendous computational savings as the number of Markov policies is $|\mathcal{A}|^{\mathtt{T}\times|\mathcal{S}|}$.

Chapter 5

MDPs with Discounted Infinite Return

This chapter covers the basic theory and main solution methods for stationary MDPs over an infinite horizon, with the discounted return criterion. In this case, stationary policies are optimal.

The discounted return problem is the most "well behaved" among all infinite horizon problems (such as average return and stochastic shortest path), and the theory of it is relatively simple, both in the planning and the learning contexts. For that reason, as well as its usefulness, we will consider here the discounted problem and its solution in some detail.

5.1 Problem Statement

We consider a stationary (time-invariant) MDP, with a finite state space $\mathcal{S}$, finite action set $\mathcal{A}$, and transition kernel $P = (P(\mathtt{s}'|\mathtt{s}, \mathtt{a}))$ over the infinite time horizon $\mathbb{T} = \{0, 1, 2, \ldots\}$.

Our goal is to maximize the expected discounted return, which is defined for each control policy π and initial state $\mathtt{s}_0 = s$ as follows:

$$\begin{aligned}\mathcal{V}_\gamma^\pi(\mathtt{s}) &= \mathbb{E}^\pi(\sum_{\mathtt{t}=0}^{\infty} \gamma^{\mathtt{t}} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})|\mathtt{s}_0 = \mathtt{s}) \\ &\equiv \mathbb{E}^{\pi,\mathtt{s}}(\sum_{\mathtt{t}=0}^{\infty} \gamma^{\mathtt{t}} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}))\end{aligned}$$

where $\mathbb{E}^{\pi,\mathtt{s}}$ uses the distribution induced by policy π starting at state $\mathtt{s}$. Here,

- $\mathtt{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the (running, or instantaneous) expected reward function, i.e., $\mathtt{r}(\mathtt{s}, \mathtt{a}) = \mathbb{E}[R|\mathtt{s}, \mathtt{a}]$.
- $\gamma \in (0, 1)$ is the discount factor.

We observe that $\gamma < 1$ ensures convergence of the infinite sum (since the rewards $\mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t})$ are uniformly bounded). With $\gamma = 1$ we obtain the total return criterion, which is harder to handle due to possible divergence of the sum.

Let $\mathcal{V}^*_\gamma(\mathtt{s})$ denote the maximal value of the discounted return, over all (possibly history dependent and randomized) control policies, i.e.,

$$\mathcal{V}^*_\gamma(\mathtt{s}) = \sup_{\pi \in \Pi_{HS}} \mathcal{V}^\pi_\gamma(\mathtt{s}).$$

Our goal is to find an optimal control policy π^* that attains that maximum (for all initial states), and compute the numeric value of the optimal return $\mathcal{V}^*_\gamma(\mathtt{s})$. As we shall see, for this problem there always exists an optimal policy which is a (deterministic) stationary policy.

Note: As usual, the discounted performance criterion can be defined in terms of cost:

$$\mathcal{C}^\pi_\gamma(\mathtt{s}) = \mathbb{E}^{\pi,\mathtt{s}}(\sum_{\mathtt{t}=0}^{\infty} \gamma^\mathtt{t} \mathtt{c}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}))$$

where $\mathtt{c}(\mathtt{s}, \mathtt{a})$ is the running cost function. Our goal is then to minimize the discounted cost $\mathcal{C}^\pi_\gamma(\mathtt{s})$.

5.2 The Fixed-Policy Value Function

We start the analysis by defining and computing the value function for a fixed stationary policy. This intermediate step is required for later analysis of our optimization problem, and also serves as a gentle introduction to the value iteration approach.

For a stationary policy $\pi : \mathcal{S} \to \mathcal{A}$, we define the value function $\mathcal{V}^\pi(\mathtt{s})$, $\mathtt{s} \in \mathcal{S}$ simply as the corresponding discounted return:

$$\mathcal{V}^\pi(\mathtt{s}) \triangleq \mathbb{E}^{\pi,\mathtt{s}} \left(\sum_{\mathtt{t}=0}^{\infty} \gamma^\mathtt{t} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) \right) = \mathcal{V}^\pi_\gamma(\mathtt{s}), \quad \mathtt{s} \in S$$

Lemma 5.1. *For $\pi \in \Pi_{SD}$, the value function $\mathcal{V}^\pi$ satisfies the following set of $|\mathcal{S}|$ linear equations:*

$$\mathcal{V}^\pi(\mathtt{s}) = \mathtt{r}(\mathtt{s}, \pi(\mathtt{s})) + \gamma \sum\nolimits_{\mathtt{s}' \in \mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \pi(\mathtt{s})) \mathcal{V}^\pi(\mathtt{s}') , \quad \mathtt{s} \in \mathcal{S}. \tag{5.1}$$

Proof. We first note that

$$\begin{aligned}\mathcal{V}^{\pi}(\mathtt{s}) &\triangleq \mathbb{E}^{\pi}(\sum_{\mathtt{t}=0}^{\infty}\gamma^{\mathtt{t}}\mathtt{r}(\mathtt{s_t},\mathtt{a_t})|\mathtt{s}_0=\mathtt{s})\\ &= \mathbb{E}^{\pi}(\sum_{\mathtt{t}=1}^{\infty}\gamma^{t-1}\mathtt{r}(\mathtt{s_t},\mathtt{a_t})|\mathtt{s}_1=\mathtt{s}),\end{aligned}$$

since both the model and the policy are stationary. Now,

$$\begin{aligned}\mathcal{V}^{\pi}(\mathtt{s}) &= \mathtt{r}(\mathtt{s},\pi(\mathtt{s})) + \mathbb{E}^{\pi}(\sum_{\mathtt{t}=1}^{\infty}\gamma^{\mathtt{t}}\mathtt{r}(\mathtt{s_t},\pi(\mathtt{s_t}))|\mathtt{s}_0=\mathtt{s})\\ &= \mathtt{r}(\mathtt{s},\pi(\mathtt{s})) + \mathbb{E}^{\pi}\left[\mathbb{E}^{\pi}\left(\sum_{\mathtt{t}=1}^{\infty}\gamma^{\mathtt{t}}\mathtt{r}(\mathtt{s_t},\pi(\mathtt{s_t}))|\mathtt{s}_0=\mathtt{s},\mathtt{s}_1=\mathtt{s}'\right)\middle|\,\mathtt{s}_0=s\right]\\ &= \mathtt{r}(\mathtt{s},\pi(\mathtt{s})) + \sum_{\mathtt{s}'\in\mathcal{S}}p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))\mathbb{E}^{\pi}(\sum_{\mathtt{t}=1}^{\infty}\gamma^{\mathtt{t}}\mathtt{r}(\mathtt{s_t},\pi(\mathtt{s_t}))|\mathtt{s}_1=\mathtt{s}')\\ &= \mathtt{r}(\mathtt{s},\pi(\mathtt{s})) + \gamma\sum_{\mathtt{s}'\in\mathcal{S}}p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))\mathbb{E}^{\pi}(\sum_{\mathtt{t}=1}^{\infty}\gamma^{t-1}\mathtt{r}(\mathtt{s_t},\mathtt{a_t})|\mathtt{s}_1=\mathtt{s}')\\ &= \mathtt{r}(\mathtt{s},\pi(\mathtt{s})) + \gamma\sum_{\mathtt{s}'\in\mathcal{S}}p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))\mathcal{V}^{\pi}(\mathtt{s}').\end{aligned}$$

The first equality is by the definition of the value function. The second equality follows from the law of total expectation, conditioning $\mathtt{s}_1 = \mathtt{s}'$ and taking the expectation over it. By definition $\mathtt{a_t} = \pi(\mathtt{s_t})$. The third equality follows similarly to the finite-horizon case, Lemma 4.1, in Chapter 2.1). The fourth is simple algebra, taking a one multiple of the discount factor γ outside. The last by the observation in the beginning of the proof. $\square$

We can write the linear equations in (5.1) in vector form as follows. Define the column vector $r^{\pi} = (r^{\pi}(\mathtt{s}))_{\mathtt{s}\in\mathcal{S}}$ with components $r^{\pi}(\mathtt{s}) = \mathtt{r}(\mathtt{s},\pi(\mathtt{s}))$, and the transition matrix P^{π} with components $P^{\pi}(\mathtt{s}'|\mathtt{s}) = p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))$. Finally, let $\mathcal{V}^{\pi}$ denote a column vector with components $\mathcal{V}^{\pi}(\mathtt{s})$. Then (5.1) is equivalent to the linear equation set

$$\mathcal{V}^{\pi} = r^{\pi} + \gamma P^{\pi}\mathcal{V}^{\pi} \tag{5.2}$$

Lemma 5.2. *The set of linear equations* (5.1) *or* (5.2)*, with* $\mathcal{V}^{\pi}$ *as variables, has a unique solution* $\mathcal{V}^{\pi}$*, which is given by*

$$\mathcal{V}^{\pi} = (I - \gamma P^{\pi})^{-1}r^{\pi}.$$

Proof. We only need to show that the square matrix $I - \gamma P^\pi$ is non-singular. Let (λ_i) denote the eigenvalues of the matrix P^π. Since P^π is a stochastic matrix (row sums are 1), then $|\lambda_i| \leq 1$. Now, the eignevalues of $I - \gamma P^\pi$ are $(1 - \gamma\lambda_i)$, and satisfy $|1 - \gamma\lambda_i| \geq 1 - \gamma > 0$. □

Combining Lemma 5.1 and Lemma 5.2, we obtain

Proposition 5.1. *Let $\pi \in \Pi_{SD}$. The value function $\mathcal{V}^\pi = [\mathcal{V}^\pi(\mathtt{s})]$ is the unique solution of equation* (5.2)*, given by*

$$\mathcal{V}^\pi = (I - \gamma P^\pi)^{-1} r^\pi.$$

Proposition 5.1 provides a closed-form formula for computing $\mathcal{V}^\pi$. However, for large systems, computing the inverse $(I - \gamma P^\pi)^{-1}$ may be hard. In that case, the following value iteration algorithm provides an alternative, iterative method for computing $\mathcal{V}^\pi$.

Algorithm 5.1. ***Fixed-policy value iteration***

1. *Let $\mathcal{V}_0 = (\mathcal{V}_0(\mathtt{s}))_{\mathtt{s}\in\mathcal{S}}$ be arbitrary.*

2. *For $n = 0, 1, 2, \ldots$, set*

$$\mathcal{V}_{n+1}(\mathtt{s}) = \mathtt{r}(\mathtt{s}, \pi(\mathtt{s})) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \pi(\mathtt{s}))\mathcal{V}_n(\mathtt{s}')\ , \qquad \mathtt{s} \in \mathcal{S}$$

or, equivalently,

$$\mathcal{V}_{n+1} = r^\pi + \gamma P^\pi \mathcal{V}_n.$$

Proposition 5.2 (Convergence of fixed-policy value iteration). *We have $\mathcal{V}_n \to \mathcal{V}^\pi$ component-wise, that is,*

$$\lim_{n\to\infty} \mathcal{V}_n(\mathtt{s}) = \mathcal{V}^\pi(\mathtt{s}), \qquad \mathtt{s} \in \mathcal{S}.$$

Proof. Note first that

$$\begin{aligned}\mathcal{V}_1(\mathtt{s}) &= \mathtt{r}(\mathtt{s}, \pi(\mathtt{s})) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \pi(\mathtt{s}))\mathcal{V}_0(\mathtt{s}') \\ &= \mathbb{E}^\pi(\mathtt{r}(\mathtt{s}_0, a_0) + \gamma\mathcal{V}_0(\mathtt{s}_1)|\mathtt{s}_0 = \mathtt{s}).\end{aligned}$$

Continuing similarly, we obtain that

$$\mathcal{V}_n(\mathtt{s}) = \mathbb{E}^\pi(\sum_{\mathtt{t}=0}^{n-1} \gamma^{\mathtt{t}} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) + \gamma^n \mathcal{V}_0(s_n)|\mathtt{s}_0 = \mathtt{s}).$$

Note that $\mathcal{V}_n(\mathtt{s})$ is the n-stage discounted return, with terminal reward $\mathtt{r}_n(\mathtt{s}_n) = \mathcal{V}_0(\mathtt{s}_n)$. Comparing with the definition of $\mathcal{V}^\pi$, we can see that

$$\mathcal{V}^\pi(\mathtt{s}) - \mathcal{V}_n(\mathtt{s}) = \mathbb{E}^\pi(\sum_{\mathtt{t}=n}^{\infty} \gamma^{\mathtt{t}} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) - \gamma^n \mathcal{V}_0(\mathtt{s}_n) | \mathtt{s}_0 = \mathtt{s}).$$

Denoting $\mathtt{R}_{\max} = \max_{\mathtt{s},\mathtt{a}} |\mathtt{r}(\mathtt{s}, \mathtt{a})|$, $\bar{\mathcal{V}}_0 = \max_s |\mathcal{V}_0(\mathtt{s})|$ we obtain

$$|\mathcal{V}^\pi(\mathtt{s}) - \mathcal{V}_n(\mathtt{s})| \leq \gamma^n (\frac{\mathtt{R}_{\max}}{1-\gamma} + \bar{\mathcal{V}}_0)$$

which converges to 0 since $\gamma < 1$. $\square$

Comments:

- The proof provides an explicit bound on $|\mathcal{V}^\pi(\mathtt{s}) - \mathcal{V}_n(\mathtt{s})|$. It may be seen that the convergence is exponential, with rate $O(\gamma^n)$.
- Using vector notation, it may be seen that

$$\mathcal{V}_n = r^\pi + P^\pi r^\pi + \ldots + (P^\pi)^{n-1} r^\pi + (P^\pi)^n \mathcal{V}_0 = \sum_{\mathtt{t}=0}^{n-1} (P^\pi)^{\mathtt{t}} r^\pi + (P^\pi)^n \mathcal{V}_0.$$

  Similarly, $\mathcal{V}^\pi = \sum_{\mathtt{t}=0}^{\infty} (P^\pi)^{\mathtt{t}} r^\pi$.

In summary:

- Proposition 5.1 allows to compute $\mathcal{V}^\pi$ by solving a set of $|\mathcal{S}|$ linear equations.
- Proposition 5.2 computes $\mathcal{V}^\pi$ by an infinite recursion, that converges exponentially fast.

5.3 Overview: The Main DP Algorithms

We now return to the optimal planning problem defined in Section 5.1. Recall that $\mathcal{V}^*_\gamma(\mathtt{s}) = \sup_{\pi \in \Pi_{HS}} \mathcal{V}^\pi_\gamma(\mathtt{s})$ is the optimal discounted return. We further denote

$$\mathcal{V}^*(\mathtt{s}) \triangleq \mathcal{V}^*_\gamma(\mathtt{s}), \quad \mathtt{s} \in \mathcal{S},$$

and refer to $\mathcal{V}^*$ as the optimal value function. Depending on the context, we consider $\mathcal{V}^*$ either as a function $\mathcal{V}^* : \mathcal{S} \to \mathbb{R}$, or as a column vector $\mathcal{V}^* = [\mathcal{V}(\mathtt{s})]_{\mathtt{s}\in\mathcal{S}}$.

The following optimality equation provides an explicit characterization of the value function, and shows that an optimal stationary policy can easily be computed if the value function is known.

Theorem 5.1 (**Bellman's Optimality Equation**). *The following statements hold:*

1. *$\mathcal{V}^*$ is the unique solution of the following set of (nonlinear) equations:*

$$\mathcal{V}(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}(\mathtt{s}') \right\}, \quad \mathtt{s} \in \mathcal{S}. \tag{5.3}$$

2. *Any stationary policy π^* that satisfies*

$$\pi^*(\mathtt{s}) \in \arg\max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}(\mathtt{s}') \right\},$$

is an optimal policy (for any initial state $\mathtt{s}_0 \in \mathcal{S}$).

The optimality equation (5.3) is non-linear, and generally requires iterative algorithms for its solution. The main iterative algorithms are **value iteration** and **policy iteration**.

Algorithm 5.2. *Value iteration*

1. *Let $\mathcal{V}_0 = (\mathcal{V}_0(\mathtt{s}))_{\mathtt{s}\in\mathcal{S}}$ be arbitrary.*
2. *For $n = 0, 1, 2, \ldots$, set*

$$\mathcal{V}_{n+1}(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}_n(\mathtt{s}') \right\}, \quad \forall \mathtt{s} \in \mathcal{S}$$

Theorem 5.2 (**Convergence of value iteration**). *We have $\lim_{n\to\infty}\mathcal{V}_n = \mathcal{V}^*$ (component-wise). The rate of convergence is exponential, at rate $O(\gamma^n)$.*

Proof. Using our previous results on value iteration for the finite-horizon problem, it follows that

$$\mathcal{V}_n(\mathtt{s}) = \max_{\pi} \mathbb{E}^{\pi,\mathtt{s}}\Big(\sum_{\mathtt{t}=0}^{n-1} \gamma^{\mathtt{t}} R_t + \gamma^n \mathcal{V}_0(\mathtt{s}_n)\Big).$$

Comparing to the optimal value function

$$\mathcal{V}^*(\mathtt{s}) = \max_{\pi} \mathbb{E}^{\pi,\mathtt{s}}\Big(\sum_{\mathtt{t}=0}^{\infty} \gamma^{\mathtt{t}} R_t\Big),$$

it may be seen that that

$$|\mathcal{V}_n(\mathtt{s}) - \mathcal{V}^*(\mathtt{s})| \leq \gamma^n(\frac{\mathtt{R}_{\max}}{1-\gamma} + ||\mathcal{V}_0||_\infty).$$

As $\gamma < 1$, this implies that $\mathcal{V}_n$ converges to $\mathcal{V}^*_\gamma$ exponentially fast. □

The value iteration algorithm iterates over the value functions, with asymptotic convergence. The policy iteration algorithm iterates over stationary policies, with each new policy better than the previous one. This algorithm converges to the optimal policy in a finite number of steps.

Algorithm 5.3. ***Policy iteration***

1. *Initialization: choose some stationary policy π_0.*
2. *For $k = 0, 1, \ldots$:*
   (a) *Policy evaluation: compute $\mathcal{V}^{\pi_k}$.*
   (For example, use the explicit formula $\mathcal{V}^{\pi_k} = (I - \gamma P^{\pi_k})^{-1} r^{\pi_k}$.)
   (b) *Policy Improvement: Compute π_{k+1}, a greedy policy with respect to $\mathcal{V}^{\pi_k}$:*

   $$\pi_{k+1}(\mathtt{s}) \in \arg\max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a}) \mathcal{V}^{\pi_k}(\mathtt{s}') \right\}, \quad \forall \mathtt{s} \in \mathcal{S}.$$

   (c) *Stop if $\mathcal{V}^{\pi_{k+1}} = \mathcal{V}^{\pi_k}$ (or if $\mathcal{V}^{\pi_k}$ satisfied the optimality equation), else continue.*[1]

Theorem 5.3 (Convergence of policy iteration). *The following statements hold:*

1. *Each policy π_{k+1} is improving over the previous one π_k, in the sense that $\mathcal{V}^{\pi_{k+1}} \geq \mathcal{V}^{\pi_k}$ (component-wise).*
2. *$\mathcal{V}^{\pi_{k+1}} = \mathcal{V}^{\pi_k}$ if and only if π_k is an optimal policy.*
3. *Consequently, since the number of stationary policies is finite, π_k converges to the optimal policy after a finite number of steps.*

An additional solution method for DP planning relies on a Linear Programming formulation of the problem. A Linear Program (LP) is simply an optimization problem with linear objective function and linear constraints. We will provide additional details later in this Chapter, in Section 5.10.

[1]YM: note that we compute the value of $k+1$ only in the next iteration

5.4 Contraction Operators

The basic proof methods of the DP results mentioned above rely on the concept of a *contraction operator*. We provide here the relevant mathematical background, and illustrate the contraction properties of some basic Dynamic Programming operators.

5.4.1 The contraction property

Recall that a norm $||\cdot||$ over $\mathbb{R}^n$ is a real-valued function $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}$ such that, for any pair of vectors $x, y \in \mathbb{R}^d$ and scalar a,

1. $||ax|| = |a| \cdot ||x||$,
2. $||x + y|| \leq ||x|| + ||y||$,
3. $||x|| = 0$ only if $x = 0$.

Common examples are the p-norm $||x||_p = (\sum_{i=1}^d |x_i|^p)^{1/p}$ for $p \geq 1$, and in particular the Euclidean norm ($p = 2$). Here we will mostly use the max-norm:

$$||x||_\infty = \max_{1\leq i\leq d} |x_i|.$$

Let $T : \mathbb{R}^d \to \mathbb{R}^d$ be a vector-valued function over $\mathbb{R}^d$ ($d \geq 1$).[2] We equip $\mathbb{R}^d$ with some norm $||\cdot||$, and refer to T as an *operator* over $\mathbb{R}^d$. Thus, $T(v) \in \mathbb{R}^d$ for any $v \in \mathbb{R}^d$. We also denote $T^n(v) = T(T^{n-1}(v))$ for $n \geq 2$. For example, $T^2(v) = T(T(v))$.

Definition 5.1. *The operator T is called a contraction operator if there exists $\beta \in$ $(0, 1)$ (the contraction coefficient) such that*

$$||T(v_1) - T(v_2)|| \leq \beta ||v_1 - v_2||,$$

for all $v_1, v_2 \in \mathbb{R}^d$

5.4.2 The Banach Fixed Point Theorem

The following celebrated result applies to contraction operators. While we quote the result for $\mathbb{R}^d$, we note that it applies in much greater generality to any Banach space (a complete normed space), or even to any complete metric space, with essentially the same proof.

[2]YM: Maybe use another letter. We have T for the horizon. Need to define a Macro for it.

Theorem 5.4 (Banach's fixed point theorem). *Let $T : \mathbb{R}^d \to \mathbb{R}^d$ be a contraction operator. Then*

1. *The equation $T(v) = v$ has a unique solution $\mathcal{V}^* \in \mathbb{R}^d$.*
2. *For any $v_0 \in \mathbb{R}^d$, $\lim_{n\to\infty} T^n(v_0) = \mathcal{V}^*$. In fact, $||T^n(v_0) - \mathcal{V}^*|| \leq O(\beta^n)$, where β is the contraction coefficient.*

Proof. Fix any v_0 and define $v_{n+1} = T(v_n)$. We will show that: (1) there exists a limit to the sequence, and (2) the limit is a fixed point of T.

Existence of a limit v^* of the sequence v_n
We show that the sequence of v_n is a cauchy sequence. We consider two elements v_n and v_{m+n} and bound the distance between them.

$$\begin{aligned}
\|v_{n+m} - v_n\| &= \|\sum_{k=0}^{m-1} v_{n+k+1} - v_{n+k}\| \\
&\leq \sum_{k=0}^{m-1} \|v_{n+k+1} - v_{n+k}\| \quad (\textit{according to the triangle inequality}) \\
&= \sum_{k=0}^{m-1} \|T^{n+k}v_1 - T^{n+k}v_0\| \\
&\leq \sum_{k=0}^{m-1} \beta^{n+k}\|v_1 - v_0\| \quad (\textit{contraction } n+k \textit{ times}) \\
&= \frac{\beta^n(1-\beta^m)}{1-\beta}\|v_1 - v_0\|
\end{aligned}$$

Since the coefficient decreases as n increases, for any $\epsilon > 0$ there exists $N > 0$ such that for all $n \geq N$, $\|\vec{v}_{n+m} - \vec{v}_n\| < \epsilon$. This implies that the sequence is a Cauchy sequence, and hence the sequence v_n has a limit. Let us call this limit v^*. Next we show that v^* is a fixed point of the operator T.

v^* is a fixed point
We need to show that $T(v^*) = v^*$, or equivalently $\|T(v^*) - v^*\| = 0$.

$$\begin{aligned}
0 &\leq \|T(v^*) - v^*\| \\
&\leq \|T(v^*) - v_n\| + \|v_n - v^*\| \; (\textit{according to the triangle inequality}) \\
&= \|T(v^*) - T(v_{n-1})\| + \|v_n - v^*\| \\
&\leq \beta\|\underbrace{v^* - v_{n-1}}_{\to 0}\| + \|\underbrace{v^n - v^*}_{\to 0}\|
\end{aligned}$$

Since v^* is the limit of v_n, i.e., $\lim_{n\to\infty} \|\vec{v}_n - \vec{v^*}\| = 0$ hence

$$\|T\vec{v^*} - \vec{v^*}\| = 0.$$

Thus, v^* is a fixed point of the operator T.

Uniqueness of $\vec{v^*}$
Assume that $T(v_1) = v_1$, and $T(v_2) = v_2$, and $v_1 \neq v_2$. Then

$$\|v_1 - v_2\| = \|T(v_1) - T(v_2)\| \leq \beta\|v_1 - v_2\|$$

Hence, this is in contradiction to $\beta < 1$. Therefore, v^* is unique. □

5.4.3 The Dynamic Programming Operators

We next define the basic Dynamic Programming operators, and show that they are in fact contraction operators.

For a fixed stationary policy $\pi : \mathcal{S} \to \mathcal{A}$, define the fixed policy DP operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ as follows: For any $V = (\mathcal{V}(\mathtt{s})) \in R^{|\mathcal{S}|}$,

$$(T^\pi(V))(\mathtt{s}) = \mathtt{r}(\mathtt{s}, \pi(\mathtt{s})) + \gamma \sum\nolimits_{\mathtt{s}' \in \mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \pi(\mathtt{s}))\mathcal{V}(\mathtt{s}'), \quad \mathtt{s} \in \mathcal{S}$$

In our column-vector notation, this is equivalent to $T^\pi(V) = r^\pi + \gamma P^\pi V$.

Similarly, define the discounted-return **Dynamic Programming Operator** T^* : $\mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ as follows: For any $V = (\mathcal{V}(\mathtt{s})) \in R^{|\mathcal{S}|}$,

$$(T^*(V))(\mathtt{s}) = \max_{\mathtt{a} \in \mathcal{A}} \left\{ \mathtt{r}(\mathtt{s}, \mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}' \in \mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \mathtt{a})\mathcal{V}(\mathtt{s}') \right\}, \quad \mathtt{s} \in \mathcal{S}$$

We note that T^π is a linear operator, while T^* is generally non-linear due to the maximum operation.

Let $||V||_\infty \triangleq \max_{\mathtt{s} \in \mathcal{S}} |\mathcal{V}(\mathtt{s})|$ denote the max-norm of V. Recall that $0 < \gamma < 1$.

Theorem 5.5 (**Contraction property**). *The following statements hold:*

1. *T^π is a γ-contraction operator with respect to the max-norm, namely $||T^\pi(\mathcal{V}_1) - T^\pi(\mathcal{V}_2)||_\infty \leq \gamma||\mathcal{V}_1 - \mathcal{V}_2||_\infty$ for all $\mathcal{V}_1, \mathcal{V}_2 \in R^{|\mathcal{S}|}$.*

2. *Similarly, T^* is an γ-contraction operator with respect to the max-norm.*

Proof. 1. Fix $\mathcal{V}_1, \mathcal{V}_2$. For every state $\mathtt{s}$,

$$\begin{aligned}|(T^\pi(\mathcal{V}_1))(\mathtt{s}) - (T^\pi(\mathcal{V}_2))(\mathtt{s})| &= \left|\gamma \sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))[\mathcal{V}_1(\mathtt{s}') - \mathcal{V}_2(\mathtt{s}')]\right| \\ &\leq \gamma \sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))\,|\mathcal{V}_1(\mathtt{s}') - \mathcal{V}_2(\mathtt{s}')| \\ &\leq \gamma \sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\pi(\mathtt{s}))\,\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty = \gamma\,\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty\ .\end{aligned}$$

Since this holds for every $\mathtt{s} \in \mathcal{S}$ the required inequality follows.

2. The proof here is more intricate due to the maximum operation. As before, we need to show that $|T^*(\mathcal{V}_1)(\mathtt{s}) - T^*(\mathcal{V}_2)(\mathtt{s})| \leq \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty$. Fixing the state $\mathtt{s}$, we consider separately the positive and negative parts of the absolute value:

(a) $T^*(\mathcal{V}_1)(\mathtt{s}) - T^*(\mathcal{V}_2)(\mathtt{s}) \leq \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty$: Let $\bar{\mathtt{a}}$ denote an action that attains the maximum in $T^*(\mathcal{V}_1)(\mathtt{s})$, namely $\bar{\mathtt{a}} \in \arg\max_{\mathtt{a}\in\mathcal{A}} \left\{\mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}_1(\mathtt{s}')\right\}$. Then

$$T^*(\mathcal{V}_1)(\mathtt{s}) = \mathtt{r}(\mathtt{s},\bar{\mathtt{a}}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\bar{\mathtt{a}})\mathcal{V}_1(\mathtt{s}')$$
$$T^*(\mathcal{V}_2)(\mathtt{s}) \geq \mathtt{r}(\mathtt{s},\bar{\mathtt{a}}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\bar{\mathtt{a}})\mathcal{V}(\mathtt{s}')$$

Since the same action $\bar{\mathtt{a}}$ appears in both expressions, we can now continue to show the inequality (a) similarly to 1.

(b) $T^*(\mathcal{V}_2)(\mathtt{s}) - T^*(\mathcal{V}_1)(\mathtt{s}) \leq \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty$. By (a) we have

$$T^*(\mathcal{V}_2)(\mathtt{s}) - T^*(\mathcal{V}_1)(\mathtt{s}) \leq \gamma\|\mathcal{V}_2 - \mathcal{V}_1\|_\infty = \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty.$$

The inequalities (a) and (b) together imply that $|T^*(\mathcal{V}_1)(\mathtt{s}) - T^*(\mathcal{V}_2)(\mathtt{s})| \leq \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty$. Since this holds for any state $\mathtt{s}$, it follows that $||T^*(\mathcal{V}_1) - T^*(\mathcal{V}_2)||_\infty \leq \gamma\|\mathcal{V}_1 - \mathcal{V}_2\|_\infty$.

□

5.5 Proof of Bellman's Optimality Equation

We prove in this section Theorem 5.1, which is restated here:

Theorem (Bellman's Optimality Equation)**.** *The following statements hold:*

1. *$\mathcal{V}^*$ is the unique solution of the following set of (nonlinear) equations:*

$$\mathcal{V}(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}(\mathtt{s}') \right\}, \quad \mathtt{s}\in\mathcal{S}. \tag{5.4}$$

2. *Any stationary policy π^* that satisfies*

$$\pi^*(\mathtt{s}) \in \arg\max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}(\mathtt{s}') \right\},$$

   is an optimal policy (for any initial state $\mathtt{s}_0 \in \mathcal{S}$).

We observe that the Optimality equation in part 1 is equivalent to $V = T^*(V)$ where T^* is the optimal DP operator from the previous section, which was shown to be a contraction operator with coefficient γ. The proof also uses the value iteration property of Theorem 5.2, which is proved in the next section.

Proof of Theorem 5.1: We prove each part.

1. As T^* is a contraction operator, existence and uniqueness of the solution to $V = T^*(V)$ follows from the Banach fixed point theorem (Theorem 5.4). Let $\hat{V}$ denote that solution. It also follows by that theorem (Theorem 5.4) that $(T^*)^n(\mathcal{V}_0) \to \hat{V}$ for any $\mathcal{V}_0$. By Theorem 5.2 we have that $(T^*)^n(\mathcal{V}_0) \to \mathcal{V}^*$, hence $\hat{V} = \mathcal{V}^*$, so that $\mathcal{V}^*$ is indeed the unique solution of $V = T^*(V)$.

2. By definition of π^* we have

$$T^{\pi^*}(\mathcal{V}^*) = T^*(\mathcal{V}^*) = \mathcal{V}^*,$$

   where the last equality follows from part 1. Thus the optimal value function satisfied the equation $T^{\pi^*}\mathcal{V}^* = \mathcal{V}^*$. But we already know (from Prop. 5.2) that $\mathcal{V}^{\pi^*}$ is the unique solution of that equation, hence $\mathcal{V}^{\pi^*} = \mathcal{V}^*$. This implies that π^* achieves the optimal value (for any initial state), and is therefore an optimal policy as stated.

□

5.6 Value Iteration

The value iteration algorithm allows to compute the optimal value function $\mathcal{V}^*$ iteratively to any required accuracy. The Value Iteration algorithm (Algorithm 5.2) can be stated as follows:

1. Start with any initial value function $\mathcal{V}_0 = (\mathcal{V}_0(\mathtt{s}))$.
2. Compute recursively, for $n = 0, 1, 2, \ldots$,

$$\mathcal{V}_{n+1}(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}} \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})[\mathtt{r}(\mathtt{s},\mathtt{a},\mathtt{s}') + \gamma\mathcal{V}_n(\mathtt{s}')], \quad \forall \mathtt{s}\in\mathcal{S}.$$

3. Apply a stopping rule obtain a required accuracy (see below).

In terms of the DP operator T^*, value iteration is simply stated as:

$$\mathcal{V}_{n+1} = T^*(\mathcal{V}_n), \quad n \geq 0.$$

Note that the number of operations for each iteration is $O(|\mathcal{A}|\cdot|\mathcal{S}|^2)$. Theorem 5.2 states that $\mathcal{V}_n \to \mathcal{V}^*$, exponentially fast. The proof follows.

5.6.1 Error bounds and stopping rules:

While we showed an exponential convergence rate, it is important to have a criteria that would depend only on the observed quantities.

Lemma 5.3. *If $\|\mathcal{V}_{n+1} - \mathcal{V}_n\|_\infty < \epsilon\cdot\frac{1-\gamma}{2\gamma}$ then $\|\mathcal{V}_{n+1} - \mathcal{V}^*\|_\infty < \frac{\varepsilon}{2}$ and $|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}^*| \leq \varepsilon$, where π_{n+1} is the greedy policy w.r.t. $\mathcal{V}_{n+1}$.*

Proof. Assume that $\|\mathcal{V}_{n+1} - \mathcal{V}_n\| < \varepsilon\cdot\frac{1-\gamma}{2\gamma}$, and we show that $\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}^*\| < \varepsilon$, which would make the policy π_{n+1} ε-optimal. We bound the difference between $\mathcal{V}^{\pi_{n+1}}$ and $\mathcal{V}^*$. (All the norms are max-norm.) We consider the following:

$$\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}^*\| < \|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}_{n+1}\| + \|\mathcal{V}_{n+1} - \mathcal{V}^*\| \tag{5.5}$$

We now bound each part of the sum separately:

$$\begin{aligned}\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}_{n+1}\| &= \|T^{\pi_{n+1}}(\mathcal{V}^{\pi_{n+1}}) - \mathcal{V}_{n+1}\| \qquad (\textit{because } \mathcal{V}^{\pi_{n+1}} \textit{ is the fixed point of } T^{\pi_{n+1}})\\ &\leq \|T^{\pi_{n+1}}(\mathcal{V}^{\pi_{n+1}}) - T^*(\mathcal{V}_{n+1})\| + \|T^*(\mathcal{V}_{n+1}) - \mathcal{V}_{n+1}\|\end{aligned}$$

Since π_{n+1} is maximal over the actions using $\mathcal{V}_{n+1}$, it implies that $T^{\pi_{n+1}}(\mathcal{V}_{n+1}) = T^*(\mathcal{V}_{n+1})$ and we conclude that:

$$\begin{aligned}\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}_{n+1}\| &\leq \|T^{\pi_{n+1}}(\mathcal{V}^{\pi_{n+1}}) - T^{\pi_{n+1}}(\mathcal{V}_{n+1})\| + \|T^*(\mathcal{V}_{n+1}) - T^*(\mathcal{V}_n)\|\\ &\leq \gamma\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}_{n+1}\| + \gamma\|\mathcal{V}_{n+1} - \mathcal{V}_n\|\end{aligned}$$

Rearranging, this implies that,

$$\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}_{n+1}\| \leq \frac{\gamma}{1-\gamma}\|\mathcal{V}_{n+1} - \mathcal{V}_n\| < \frac{\gamma}{1-\gamma} \cdot \epsilon \cdot \frac{1-\gamma}{2\gamma} = \frac{\epsilon}{2}$$

For the second part of the sum we derive similarly that:

$$\|\mathcal{V}_{n+1} - \mathcal{V}^*\| \leq \frac{\gamma}{1-\gamma}\|\mathcal{V}_{n+1} - \mathcal{V}_n\| < \frac{\gamma}{1-\gamma} \cdot \epsilon \cdot \frac{1-\gamma}{2\gamma} = \frac{\epsilon}{2}$$

Returning to inequality 5.5, it follows:

$$\|\mathcal{V}^{\pi_{n+1}} - \mathcal{V}^*\| \leq \frac{2\gamma}{1-\gamma}\|\mathcal{V}_{n+1} - \mathcal{V}_n\| < \epsilon$$

Therefore the selected policy π_{n+1} is ϵ-optimal. □

5.7 Policy Iteration

The policy iteration algorithm, introduced by Howard (1960), computes an optimal policy π^* in a finite number of steps. This number is typically small (on the same order as $|\mathcal{S}|$).

The basic principle behind Policy Iteration is Policy Improvement. Let π be a stationary policy, and let $\mathcal{V}^\pi$ denote its value function. A stationary policy $\bar{\pi}$ is called π- improving if it is a greedy policy with respect to $\mathcal{V}^\pi$, namely

$$\bar{\pi}(\mathtt{s}) \in \arg\max_{\mathtt{a}\in\mathcal{A}} \left\{ \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}^\pi(\mathtt{s}') \right\}, \quad \mathtt{s} \in \mathcal{S}.$$

Lemma 5.4 (Policy Improvement)**.** *We have $\mathcal{V}^{\bar{\pi}} \geq \mathcal{V}^\pi$ (component-wise), and equality holds if and only if π is an optimal policy.*

Proof. Observe first that

$$\mathcal{V}^\pi = T^\pi(\mathcal{V}^\pi) \leq T^*(\mathcal{V}^\pi) = T^{\bar{\pi}}(\mathcal{V}^\pi)$$

The first equality follows since $\mathcal{V}^\pi$ is the value function for the policy π, the inequality follows because of the maximization in the definition of T^*, and the last equality by definition of the improving policy $\bar{\pi}$.

It is easily seen that T^π is a monotone operator (for any policy π), namely $\mathcal{V}_1 \leq \mathcal{V}_2$ implies $T^\pi(\mathcal{V}_1) \leq T^\pi(\mathcal{V}_2)$. Applying $T^{\bar{\pi}}$ repeatedly to both sides of the above inequality $\mathcal{V}^\pi \leq T^{\bar{\pi}}(\mathcal{V}^\pi)$ therefore gives

$$\mathcal{V}^\pi \leq T^{\bar{\pi}}(\mathcal{V}^\pi) \leq (T^{\bar{\pi}})^2(\mathcal{V}^\pi) \leq \cdots \leq \lim_{n\to\infty} (T^{\bar{\pi}})^n(\mathcal{V}^\pi) = \mathcal{V}^{\bar{\pi}},$$

where the last equality follows by value iteration. This establishes the first claim.

We now show that π is optimal if and only if $\mathcal{V}^{\bar{\pi}} = \mathcal{V}^{\pi}$. We showed that $\mathcal{V}^{\bar{\pi}} \geq \mathcal{V}^{\pi}$. If $\mathcal{V}^{\bar{\pi}} > \mathcal{V}^{\pi}$ then clearly π is not optimal. Assume that $\mathcal{V}^{\bar{\pi}} = \mathcal{V}^{\pi}$. We have the following identities:

$$\mathcal{V}^{\pi} = \mathcal{V}^{\bar{\pi}} = T^{\bar{\pi}}(\mathcal{V}^{\bar{\pi}}) = T^{\bar{\pi}}(\mathcal{V}^{\pi}) = T^{*}(\mathcal{V}^{\pi}) = T^{*}(\mathcal{V}^{\bar{\pi}})$$

where the first equality follows since $\mathcal{V}^{\bar{\pi}}$ is the fixed point of its operator $T^{\bar{\pi}}$. The second follows since we assume that $\mathcal{V}^{\bar{\pi}} = \mathcal{V}^{\pi}$. The third follows since $T^{\bar{\pi}}$ and T^{*} are identical on $\mathcal{V}^{\pi}$.

We have established that: $\mathcal{V}^{\pi} = T^{*}(\mathcal{V}^{\pi})$, and hence $\mathcal{V}^{\pi}$ and π is a fixed point of T^{*} and therefore optimal. $\square$

The policy iteration algorithm performs successive rounds of policy improvement, where each policy π_{k+1} improves the previous one π_k. Since the number of stationary policies is bounded, so is the number of strict improvements, and the algorithm must terminate with an optimal policy after a finite number of steps.

In terms of computational complexity, Policy Iteration requires $O(|\mathcal{A}| \cdot |\mathcal{S}|^2 + |\mathcal{S}|^3)$ operations per step, with the number of steps being typically small. In contrast, Value Iteration requires $O(|\mathcal{A}| \cdot |\mathcal{S}|^2)$ per step, but the number of required iterations may be large, especially when the discount factor γ is close to 1.

5.8 Some Variants on Value Iteration and Policy Iteration

5.8.1 Value Iteration - Gauss Seidel Iteration

In the standard value iteration: $\mathcal{V}_{n+1} = T^{*}(\mathcal{V}_n)$, the vector $\mathcal{V}_n$ is held fixed while all entries of $\mathcal{V}_{n+1}$ are updated. An alternative is to update each element $\mathcal{V}_n(\mathtt{s})$ of that vector as to $\mathcal{V}_{n+1}(\mathtt{s})$ as soon as the latter is computed, and continue the calculation with the new value. This procedure is guaranteed to be "as good" as the standard one, in some sense, and often speeds up convergence.

5.8.2 Asynchronous Value Iteration

Here, in each iteration $\mathcal{V}_n \Rightarrow \mathcal{V}_{n+1}$, only a subset of the entries of $\mathcal{V}_n$ (namely, a subset of all states) is updated. It can be shown that if each state is updated infinitely often, then $\mathcal{V}_n \to \mathcal{V}^{*}$. Asynchronous update can be used to focus the computational effort on "important" parts of a large-state space.

5.8.3 Modified (a.k.a. Generalized or Optimistic) Policy Iteration

This scheme combines policy improvement steps with value iteration for policy evaluation. This way the requirement for exact policy evaluation (computing $\mathcal{V}^{\pi_k} = (I - \gamma P^{\pi_k})^{-1} r^{\pi_k}$) is avoided.

The procedure starts with some initial value vector $\mathcal{V}_0$, and iterates as follows:

- Greedy policy computation:

  Compute $\pi_k \in \arg\max_\pi T^\pi(\mathcal{V}_k)$, a greedy policy with respect to $\mathcal{V}_k$.

- Partial value iteration:

  Perform m_k steps of value iteration, $\mathcal{V}_{k+1} = (T_\gamma^{\pi_k})^{m_k}(\mathcal{V}_k)$.

This algorithm guarantees convergence of $\mathcal{V}_k$ to $\mathcal{V}^*$.

5.9 A Comparison between VI and PI Algorithms

In this section we will compare the convergence rate of the VI and PI algorithms. We show that, assuming that the two algorithms begin with the same approximated value, the PI algorithm converges in less iterations.

Theorem 5.6. *Let $\{VI_n\}$ be the sequence of values created by the VI algorithm (where $VI_{n+1} = T^*(VI_n)$) and let $\{PI_n\}$ be the sequence of values created by PI algorithm, i.e., $PI_n = \mathcal{V}^{\pi_n}$. If $VI_0 = PI_0$, then for all n we have $VI_n \leq PI_n \leq \mathcal{V}^*$.*

Proof. The proof is by induction on n.
Induction Basis: By construction $VI_0 = PI_0$. $PI_0 = \mathcal{V}^{\pi_0}$, and therefore it is clearly bounded by $\mathcal{V}^*$.
Induction Step: Assume that $VI_n \leq PI_n$. For VI_{n+1} we have,

$$VI_{n+1} = T^*(VI_n) = T^{\pi'}(VI_n),$$

where π' is the greedy policy w.r.t. VI_n, i.e.,

$$\pi'(\mathtt{s}) \in \arg\max_{a \in A} \{\mathtt{r}(\mathtt{s}, \mathtt{a}) + \gamma \sum_{\mathtt{s}' \in \mathcal{S}} p(\mathtt{s}'|\mathtt{s}, \mathtt{a}) VI_n(\mathtt{s}')\}.$$

Since $VI_n \leq PI_n$, and $T^{\pi'}$ is monotonic it follows that:

$$T^{\pi'}(VI_n) \quad \leq \quad T^{\pi'}(PI_n)$$

Since T^* is taking the maximum over all policies:

$$T^{\pi'}(PI_n) \;\; \leq \;\; T^*(PI_n)$$

The policy determined by PI algorithm in iteration $n+1$ is π_{n+1} and we have:

$$T^*(PI_n) = T^{\pi_{n+1}}(PI_n)$$

From the definition of π_{n+1}, we have

$$T^{\pi_{n+1}}(PI_n) \;\; \leq \;\; \mathcal{V}^{\pi_{n+1}} = PI_{n+1}$$

Therefore, $VI_{n+1} \leq PI_{n+1}$. Since $PI_{n+1} = \mathcal{V}^{\pi_{n+1}}$, it implies that $PI_{n+1} \leq \mathcal{V}^*$. $\square$

5.10 Linear Programming Solutions

An alternative approach to value and policy iteration is the linear programming method. Here the optimal control problem is formulated as a linear program (LP), which can be solved efficiently using standard LP solvers. There are two formulations: primal and dual. As this method is less related to learning we will only sketch it briefly.

5.10.1 Some Background on Linear Programming

A Linear Program (LP) is an optimization problem that involves minimizing (or maximizing) a linear objective function subject to linear constraints. A standard form of a LP is

$$\text{minimize } b^\top x, \quad \text{subject to } Ax \geq c, x \geq 0. \tag{5.6}$$

where $x = (x_1, x_2, \ldots, x_n)^\top$ is a vector of real variables arranged as a column vector. The set of constraints is linear and defines a *convex polytope* in $\mathbb{R}^n$, namely a closed and convex set U that is the intersection of a finite number of half-spaces. U has a finite number of vertices, which are points that cannot be generated as a convex combination of other points in U. If U is bounded, it equals the convex combination of its vertices. It can be seen that an optimal solution (if finite) will be in one of these vertices.

The LP problem has been extensively studied, and many efficient solvers exist. In 1947, Danzig introduced the Simplex algorithm, which essentially moves greedily along neighboring vertices. In the 1980's effective algorithms (interior point and others) were introduced which had polynomial time guarantees.

Duality: The *dual* of the LP in (5.6) is defined as the following LP:

$$\text{maximize } c^\top y, \quad \text{subject to } A^\top y \leq b, y \geq 0. \tag{5.7}$$

The two dual LPs have the same optimal value, and (in many cases) the solution of one can be obtained from that of the other. The common optimal value can be understood by the following computation:

$$\begin{aligned}\min_{x\geq 0, Ax\geq c} b^\top x &= \min_{x\geq 0}\max_{y\geq 0}\left\{b^\top x + y^\top(c - Ax)\right\}\\ &= \max_{y\geq 0}\min_{x\geq 0}\left\{c^\top y + x^\top(b - Ay)\right\} = \max_{y\geq 0, Ay\leq b} c^\top x.\end{aligned}$$

where the second equality follows by the min-max theorem.

Note: For an LP of the form:

$$\text{minimize } b^\top x, \quad \text{subject to } Ax \geq c,$$

the dual is

$$\text{maximize } c^\top y, \quad \text{subject to } A^\top y = b, y \geq 0.$$

[[Note that one has equality and one has inequality. The first is the symmetric case with $x \geq 0$ and the second is the asymmetric case without $x \geq 0$. Seems too confusing even for me.]]

5.10.2 The Primal LP

Recall that $\mathcal{V}^*$satisfies the optimality equations:

$$\mathcal{V}(\mathtt{s}) = \max_{\mathtt{a}\in\mathcal{A}}\left\{\mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma\sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mathcal{V}(\mathtt{s}')\right\}, \quad \mathtt{s}\in\mathcal{S}.$$

Proposition 5.3. *$\mathcal{V}^*$ is the smallest function (component-wise) that satisfies the following set of inequalities:*

$$v(\mathtt{s}) \geq \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma\sum\nolimits_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})v(\mathtt{s}'), \quad \forall \mathtt{s},\mathtt{a}. \tag{5.8}$$

Proof. Suppose $v = (v(\mathtt{s}))$ satisfies (5.8). That is, $v \geq T^\pi v$ for every stationary policy. Then by the monotonicity of T^π,

$$v \geq T^\pi(v) \;\Rightarrow\; T^\pi(v) \geq (T^\pi)^2(v) \;\Rightarrow\; \ldots \Rightarrow\; (T^\pi)^k(v) \geq (T^\pi)^{k+1}v,$$

so that

$$v \geq T^\pi(v) \geq (T^\pi)^2(v) \geq \ldots \geq \lim_{n\to\infty} (T^\pi)^n(v) = \mathcal{V}^\pi.$$

Now, if we take π as the optimal policy we obtain $v \geq \mathcal{V}^*$ (component-wise). $\square$

It follows from Proposition 5.3 that $\mathcal{V}^*$ is the solution of the following linear program:

Primal LP:

$$\min_{(v(\mathtt{s}))} \sum_{\mathtt{s}\in\mathcal{S}} v(\mathtt{s}), \quad \text{subject to}$$

$$v(\mathtt{s}) \geq \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \textstyle\sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})v(\mathtt{s}'), \quad \forall \mathtt{s},\mathtt{a}.$$

Note that the number of inequality constraints is $|\mathcal{S}| \cdot |\mathcal{A}|$. Also, note that at the optimal solution we will have

$$v(\mathtt{s}) = \max_{\mathtt{a}} \mathtt{r}(\mathtt{s},\mathtt{a}) + \gamma \textstyle\sum_{\mathtt{s}'\in\mathcal{S}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})v(\mathtt{s}'), \quad \forall \mathtt{s}. \tag{5.9}$$

[[Note that the primal LP does not depend on $p_0(\mathtt{s})$. The value of the Primal LP is rather meaning less, so we should be careful what we say about the Dual !]]

5.10.3 The Dual LP

The dual of our Primal LP turns out to be:

Dual LP:

$$\max_{(\mu_{\mathtt{s},\mathtt{a}})} \sum_{\mathtt{s},\mathtt{a}} \mu_{\mathtt{s},\mathtt{a}} \mathtt{r}(\mathtt{s},\mathtt{a})$$

subject to:

$$\mu_{\mathtt{s},\mathtt{a}} \geq 0 \quad \forall \mathtt{s},\mathtt{a}$$

$$\sum_{\mathtt{s},\mathtt{a}} \mu_{\mathtt{s},\mathtt{a}} = \tfrac{1}{1-\gamma}$$

$$p_0(\mathtt{s}') + \gamma \sum_{\mathtt{s},\mathtt{a}} p(\mathtt{s}'|\mathtt{s},\mathtt{a})\mu_{\mathtt{s},\mathtt{a}} = \sum_{\mathtt{a}\in\mathcal{A}} \mu_{\mathtt{s}',\mathtt{a}} \quad \forall \mathtt{s}' \in \mathcal{S}$$

where $p_0 = (p_0(\mathtt{s}'))$ is any probability vector (usually taken as a 0/1 vector).[3] The last equality can be viewed as a conservation of the probabilities. The left hand side is the weight of probabilities (with discounting) entering $\mathtt{s}'$, and the right hand side is the probabilities (with discounting) exiting $\mathtt{s}'$. The equality requires that the two be identical.

Interpretation:

1. The variables $\mu_{\mathtt{s},\mathtt{a}}$ correspond to the "state action frequencies" (for a given policy):
$$\mu_{\mathtt{s},\mathtt{a}} \sim \mathbb{E}(\sum_{\mathtt{t}=0}^{\infty} \gamma^{\mathtt{t}} \mathbb{I}_{\{\mathtt{s}_\mathtt{t}=\mathtt{s},\mathtt{a}_\mathtt{t}=\mathtt{a}\}}) = \sum_{\mathtt{t}=0}^{\infty} \gamma^{\mathtt{t}} \Pr(\mathtt{s}_\mathtt{t} = \mathtt{s}, \mathtt{a}_\mathtt{t} = \mathtt{a}),$$
and $p_0(\mathtt{s}') \sim p(\mathtt{s}_0 = \mathtt{s}')$ is the initial state distribution.

2. It is easy to see that the discounted return can be written in terms of $\mu_{\mathtt{s},\mathtt{a}}$ as $\sum\limits_{s\in\mathcal{S}} \mu_{\mathtt{s},\mathtt{a}}\mathtt{r}(\mathtt{s},\mathtt{a})$, which is to be maximized.

3. The above constraints easily follow from the definition of $\mu_{\mathtt{s},\mathtt{a}}$.

Further comments:

- The optimal policy can by obtained directly from the solution of the dual using:
$$\pi(\mathtt{a}|\mathtt{s}) = \frac{\mu_{\mathtt{s},\mathtt{a}}}{\mu_\mathtt{s}} \equiv \frac{\mu_{\mathtt{s},\mathtt{a}}}{\sum_{\mathtt{a}\in\mathcal{A}} \mu_{\mathtt{s},\mathtt{a}}}$$
This policy can be stochastic if the solution to the LP is not unique. However, it will be deterministic even in that case if we choose f as an *extreme* solution of the LP.

- The number of constraints in the dual is $|\mathcal{S}| \cdot |\mathcal{A}| + (|\mathcal{S}| + 1)$. However, the inequality constraints are simpler than in the primal.

[3] I think p_0 should be the initial distribution vector. In any case, p_0 does not appear in the primal, so it should not appear in the dual. Similarly, $1/(1-\gamma)$. I think simply taking the dual will have $p_(\mathtt{s}) = 1$, and no constraint on the sum of $\mu_{\mathtt{s},\mathtt{a}}$. The dual as written, with p_0 the initial distribution, has as value the expected reward of the optimal policy.

Chapter 6

Stochastic shortest paths

This class of problems provides a natural extension of the standard shortest-path problem to stochastic settings. When we view Stochastic Shortest Paths (SSP) as an extension of the graph theoretic notion of shortest paths, we can motivate it by having the edges no completely deterministic, but rather having a probability of ending a different state. Probably a better view, is to think of the edges as general actions, which induce a distribution over the next state. The goal state can be either a single state or a set of states, both notions would be equivalent.

The SSP problem includes an important sub-category, which is *episodic MDP*. In an episodic MDP we are guarantee to complete the episode in (expected) finite time, regardless of the policy we employ. This will not be true of a general SSP, as some policies

Some conditions on the system dynamics and reward function must be imposed for the problem to be well posed (e.g., that a goal state may be reached with probability one). Such problems are known as stochastic shortest path problems, or also episodic planning problems.

6.1 Definition

Stochastic Shortest Path is an important class of planning problems, where the time horizon is not set beforehand, but rather the problem continues until a certain event occurs. This event can be defined as reaching some goal state. Let $\mathcal{S}_G \subset \mathcal{S}$ define the set of *goal states*.

Define

$$\tau = \inf\{t \geq 0 : \mathtt{s_t} \in \mathcal{S}_G\}$$

as the first time in which a goal state is reached.

The total expected return for Stochastic Shortest Path problem is defined as:

$$\mathcal{V}^{\pi}_{ssp}(\mathtt{s}) = \mathbb{E}^{\pi,\mathtt{s}}(\sum_{\mathtt{t}=0}^{\tau-1} \mathtt{r}(\mathtt{s}_\mathtt{t}, \mathtt{a}_\mathtt{t}) + \mathtt{r}_G(\mathtt{s}_\tau))$$

Here $\mathtt{r}_G(\mathtt{s}),\ \mathtt{s} \in \mathcal{S}_G$ specified the reward at goal states. Note that the length of the run τ is a random variable.

6.1.1 Cost versus reward

While we can always switch the optimization from minimizing costs to maximizing rewards, in this case the two would have a conceptually different optimal policies. If we have non-negative costs, then the optimal policy would try to reach the goal states as fast as possible. (Actually, would minimize the expected cost in reaching the goal states.) When we have non-negative rewards, the optimal policy would try to avoid the goal states.

When we are considering a Deterministic Decision Process we can illustrate the difference. Recall that DMP is simply a graph where the actions are traversing edges. For costs, we are looking for the minimum cost path to the goal. For rewards, we are looking for a cycle with positive costs. (In Chapter 2, we derive an algorithm for the average reward for DDP.) Note that if there is a cycle of negative cost, then the shortest path is not well define (we can get infinite negative cost). Similarly, if we have a positive reward cycle we can get infinite reward.

6.2 Relationship to other models

6.2.1 Finite Horizon Return

Stochastic shortest path includes, naturally, the finite horizon case. This can be shown by creating a leveled MDP where at each time step we move to the next level and terminate at level $\mathtt{T}$. Specifically, we define a new state space $\mathcal{S}' = \mathcal{S} \times \mathbb{T}$, transition function $p((\mathtt{s}', i+1)|\mathtt{s}, \mathtt{a}) = p(\mathtt{s}'|\mathtt{s}, \mathtt{a})$, and goal states $\mathcal{S}_G = \{(\mathtt{s}, \mathtt{T}) : \mathtt{s} \in \mathcal{S}\}$.

6.2.2 Discounted infinite return

Stochastic shortest path includes also the discounted infinite horizon. To see that, add a new goal state, and from each state with probability $1-\gamma$ jump to the goal state and terminate. The expected return of a policy would be the same in both models. Specifically, we add a state $\mathtt{s}_G$, such that $p(\mathtt{s}_G|\mathtt{s},\mathtt{a}) = 1-\gamma$, for any state $\mathtt{s} \in \mathcal{S}$ and action $\mathtt{a} \in \mathcal{A}$ and $p(\mathtt{s}'|\mathtt{s},\mathtt{a}) = \gamma p(\mathtt{s}'|\mathtt{s},\mathtt{a})$. The probability that we do not terminate by time $\mathtt{t}$ is exactly $\gamma^{\mathtt{t}}$. Therefore the expected return is $\sum_{t=1}^{\infty} \gamma^t \mathtt{r}(\mathtt{s}_t,\mathtt{a}_t)$ which is identical to the discounted return.

6.3 Proper versus improper policies

Episodic MDP

6.3.1 Proper policies

Show that if we add $\epsilon > 0$ to all the cost, for sufficiently small ϵ we get the optimal proper policy.

6.4 Belman Operator

Add assumption that any improper policy has infinite cost.

Show that the Belman operator converge.

Chapter 7

Average cost

Chapter 8

Tabular learning: Model based

Until now we looked at planning problems, where we are given a complete model of the MDP, and the goal is to either evaluate a given policy or compute the optimal policy. In this lecture we will start looking at learning problems, where we need to learn from interaction. This lecture will concentrate on *model based* learning, where the main goal is to learn an accurate model. Next lecture we will look at *model free* learning, where we learn a policy without recovering the actual underlying model.

8.1 Effective horizon of discounted return

Before we start looking a learning, we will show an "reduction" from discounted return to finite horizon return. The main issue will be to show that the discounted return has an *effective horizon* such that rewards beyond it have a negligible effect on the discounted return.

Theorem 8.1. *Given a discount factor γ, the discounted return in the first $T = \frac{1}{1-\gamma} \log \frac{R_{max}}{\epsilon(1-\gamma)}$, is within ϵ of the total discounted return.*

Proof. Recall that the rewards are $r_t \in [0, R_{max}]$. Fix an infinite sequence of rewards $(r_0, \ldots, r_t, \ldots)$. We would like to consider the following difference

$$\Delta = \sum_{t=1}^{\infty} r_t \gamma^t - \sum_{t=0}^{T-1} r_t \gamma^t = \sum_{t=T}^{\infty} r_t \gamma^t \leq \frac{\gamma^T}{1-\gamma} R_{max}$$

We like this difference Δ to be bounded by ϵ, hence

$$\frac{\gamma^T}{1-\gamma} R_{max} \leq \epsilon \ .$$

This implies that

$$T\log(1/\gamma) \geq \log\frac{R_{max}}{\epsilon(1-\gamma)}\ .$$

We can bound $\log(1/\gamma) = \log(1+\frac{1-\gamma}{\gamma}) \leq \frac{1-\gamma}{\gamma}$. Since $\gamma < 1$, the theorem follows. $\square$

8.2 Off-Policy Model-Based Learning

We consider the case that we are given as input a sequence of trajectories. Essentially, our input will be composed from quadruples:

$$(s, a, r, s')$$

where r is sampled from $R(s,a)$ and s' is sampled from $p(\cdot|s,a)$.

Our goal is to output an MDP $(S, A, \widehat{r}, \widehat{p})$, where S is the set of states, A is the set of actions, $\widehat{r}(s,a)$ is the approximate expected reward of $R(s,a) \in [0, R_{max}]$, and $\widehat{p}(s'|s,a)$ is the approximate probability of reaching state s' when we are in state s and doing action a.

8.2.1 Mean estimation

We start with a basic mean estimation problem (as you have seen in the Introduction to Machine Learning course). Suppose we are give a random variable $R \in [0,1]$ and would like to approximate its mean $\mu = E[R]$. We observe m samples of R, $r_1, \ldots, r_m$, and compute their mean $\widehat{\mu} = \frac{1}{m}\sum_{i=1}^{m} r_i$.

By the law of large numbers we know that when m goes to infinity we have that $\widehat{\mu}$ converges to μ. We would like to have concrete finite convergence bounds, mainly to derive the value of m as a function of the desired accuracy ϵ.

For this we use concentration bounds (known as Chernoff-Hoffding bounds), which bounds the additive error as follows:

$$\Pr[|\mu - \widehat{\mu}| \geq \epsilon] \leq 2e^{-2\epsilon^2 m}$$

We can also derive relative error bounds, that guarantee for $\epsilon \in (0,1)$,

$$\Pr[\widehat{\mu} \leq (1-\epsilon)\mu] \leq e^{-\epsilon^2 m/2}$$

and

$$\Pr[\widehat{\mu} \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2 m/3}$$

Using the additive concentration bound, we have that for $m \geq \frac{1}{2\epsilon^2}\log(2/\delta)$ we have $|\mu - \widehat{\mu}| \leq \epsilon$, with probability $1-\delta$.

We can now use this bound in order to estimate the expected rewards. For each state-action (s,a) let $\widehat{r}(s,a) = \frac{1}{m}\sum_{i=1}^{m} R_i(s,a)$ be the average of m samples. We can show the following:

Claim 8.1. *If we have $m \geq \frac{R_{max}}{2\epsilon^2}\log\frac{2|S|\,|A|}{\delta}$ samples for each state action (s,a), then with probability $1-\delta$ we have for every (s,a) that $|r(s,a) - \widehat{r}(s,a)| \leq \epsilon$.*

Proof. We will need to scale the random variables to $[0,1]$, which will be achieved by dividing by R_{max}. By the Chernoff-Hoffding bound, for each (s,a) we have that with probability $1 - \frac{\delta}{|S|\,|A|}$ that $|\frac{r(s,a)}{R_{max}} - \frac{\widehat{r}(s,a)}{R_{max}}| \leq \frac{\epsilon}{R_{max}}$.

We bound the probability over all state-action pais using a union bound over all state action pairs.

$$\Pr[\exists(s,a) : |\frac{r(s,a)}{R_{max}} - \frac{\widehat{r}(s,a)}{R_{max}}|| > \frac{\epsilon}{R_{max}}] \leq \sum_{(s,a)} \Pr[|\frac{r(s,a)}{R_{max}} - \frac{\widehat{r}(s,a)}{R_{max}}|| > \frac{\epsilon}{R_{max}}] \leq \sum_{(s,a)} \frac{\delta}{|S|\,|A|} = \delta$$

Therefore, we have that with probability $1-\delta$ for every (s,a) simultaneously we have $|r(s,a) - \widehat{r}(s,a)| \leq \epsilon$. $\square$

8.2.2 Influence of reward estimation errors

We would like to quantify the influence of having inaccurate estimates of the rewards. We will look both at the finite horizon return and the discounted return. We start with the case of finite horizon.

Influence of reward estimation errors: Finite horizon

Fix a policy $\pi \in MD$. We want to compare the return using $r_t(s,a)$ versus $\widehat{r}(s,a)$ and $r_T(s)$ versus $\widehat{r}_T(s)$. We will assume that for every (s,a) and t we have $|r_t(s,a) - \widehat{r}_t(s,a)| \leq \epsilon$ and $|r_T(s) - \widehat{r}_T(s)| \leq \epsilon$. We will show that the difference in return is bounded by $\epsilon(T+1)$, where T is the finite horizon.

Define the expected return of π with the true expectations

$$J_T^\pi(s_0) = E^{\pi,s_0}[\sum_{t=0}^{T} r_t(s_t,a_t) + r_T(s_T)].$$

and with the estimated expectations

$$\widehat{J}_T^\pi(s_0) = E^{\pi,s_0}[\sum_{t=0}^{T} \widehat{r}_t(s_t,a_t) + \widehat{r}_T(s_T)].$$

We are interested in bounding the difference between the two

$$error(\pi) = |J^{\pi}_T(s_0) - \widehat{J}^{\pi}_T(s_0)|.$$

Note that in both cases we use the true transition probability. For a given trajectory $\sigma = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T)$ we define

$$error(\pi, \sigma) = \sum_{t=0}^{T} r_t(s_t, a_t) + r_T(s_T)) - \sum_{t=0}^{T} \widehat{r}_t(s_t, a_t) + \widehat{r}_T(s_T))$$

taking the expercaction over trajectories we have

$$error(\pi) = |E^{\pi,s_0}[error(\pi, \sigma)]|$$

Theorem 8.2. *Assume that for every (s, a) and t we have $|r_t(s, a) - \widehat{r}_t(s, a)| \leq \epsilon$ and for every s we have $|r_T(s) - \widehat{r}_T(s)| \leq \epsilon$. Then, for any policy $\pi \in MD$ we have $error(\pi) \leq \epsilon(T + 1)$.*

Proof. The probability of each trajectory $\sigma = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T)$ is the same under $r_t(s, a)$ and $\widehat{r}_t(s, a)$, since $\pi \in MD$ implies that π depends only on the time t and state s_t. For each trajectory $\sigma = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T)$, we have,

$$\begin{aligned}
error(\pi, \sigma) &= |\sum_{t=0}^{T} r_t(s_t, a_t) + r_T(s_T) - \sum_{t=0}^{T} \widehat{r}_t(s_t, a_t) + \widehat{r}_T(s_T)| \\
&= |\sum_{t=0}^{T} (r_t(s_t, a_t) - \widehat{r}_t(s_t, a_t)) + (r_T(s_T) - \widehat{r}_T(s_T))| \\
&\leq \sum_{t=0}^{T} |r_t(s_t, a_t) - \widehat{r}_t(s_t, a_t)| + |r_T(s_T) - \widehat{r}_T(s_T)| \\
&\leq \epsilon T + \epsilon
\end{aligned}$$

The theorem follows since $error(\pi) = |E^{\pi,s_0}[error(\pi, \sigma)]| \leq \epsilon(T + 1)$. □

Computing approximate optimal policy: finite horizon

We can now describe how to compute a near optimal policy for the finite horizon case. We start with the sample requirement. We need a sample of size $m \geq \frac{1}{2\epsilon^2} \log \frac{2|S|\,|A|\,T}{\delta}$ for each $R_t(s, a)$ and $R_T(s)$. Given the sample, we compute $\widehat{r}_t(s, a)$ and $\widehat{r}_T(s)$. As we saw, with probability $1 - \delta$ we have $|r_t(s, a) - \widehat{r}_(s, a)| \leq \epsilon$ and $|r_T(s) - \widehat{r}_T(s)| \leq \epsilon$. Now we can compute the optimal policy $\widehat{\pi}^*$ for the estimated rewards $\widehat{r}_t(s, a)$ and $\widehat{r}_T(s)$. The main goal is to show that $\widehat{\pi}^*$ is a near optimal policy.

Claim 8.2. *Assume that for every (s,a) and t we have $|r_t(s,a) - \widehat{r}_t(s,a)| \le \epsilon$ and for every s we have $|r_T(s) - \widehat{r}_T(s)| \le \epsilon$. Then,*

$$J_T^{\pi^*}(s_0) - J_T^{\widehat{\pi}^*} \le 2\epsilon(T+1)$$

Proof. From the definition of $error(\pi)$ and since for any $\pi \in MD$ we showed that $error(\pi) \le \epsilon(T+1)$, we have,

$$J_T^{\pi^*}(s_0) - \widehat{J}_T^{\pi^*}(s_0) \le error(\pi^*) \le \epsilon(T+1)$$

and

$$\widehat{J}_T^{\widehat{\pi}^*}(s_0) - J_T^{\widehat{\pi}^*}(s_0) \le error(\widehat{\pi}^*) \le \epsilon(T+1)$$

Since $\widehat{\pi}^*$ is optimal for $\widehat{r}_t$ we have

$$\widehat{J}_T^{\pi^*}(s_0) \le \widehat{J}_T^{\widehat{\pi}^*}(s_0)$$

The claim follows by adding the three inequalities. □

Influence of reward estimation errors: discounted return

Fix a policy $\pi \in SD$. Again, define the expected return of π with the true expectations

$$J_\gamma^\pi(s_0) = E^{\pi,s_0}[\sum_{t=0}^{\infty} r(s_t,a_t)\gamma^t]$$

and with the estimated expectations

$$\widehat{J}_\gamma^\pi(s_0) = E^{\pi,s_0}[\sum_{t=0}^{\infty} \widehat{r}(s_t,a_t)\gamma^t]$$

We are interested in bounding the difference between the two

$$error(\pi) = |J_\gamma^\pi(s_0) - \widehat{J}_\gamma^\pi(s_0)|$$

Note that as for the finite horizon, in both cases we use the true transition probability. For a given trajectory $\sigma = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T)$ we define

$$error(\pi,\sigma) = \sum_{t=0}^{\infty} \gamma^t r_t(s_t,a_t) - \sum_{t=0}^{\infty} \gamma^t \hat{r}_t(s_t,a_t)$$

Theorem 8.3. *Assume that for every (s,a) we have $|r_t(s,a) - \widehat{r}_t(s,a)| \le \epsilon$. Then, for any policy $\pi \in SD$ we have $error(\pi) \le \frac{\epsilon}{1-\gamma}$.*

Proof. The probability of each trajectory $\sigma = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$ is the same under $r(s,a)$ and $\widehat{r}(s,a)$, since $\pi \in SD$. For each trajectory $\sigma = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$, we have,

$$\begin{aligned} error(\pi,\sigma) &= |\sum_{t=0}^{\infty} r(s_t,a_t)\gamma^t - \sum_{t=0}^{\infty} \widehat{r}(s_t,a_t)\gamma^t| \\ &= |\sum_{t=0}^{\infty} (r(s_t,a_t) - \widehat{r}(s_t,a_t))\gamma^t| \\ &\le \sum_{t=0}^{\infty} |r(s_t,a_t) - \widehat{r}(s_t,a_t)|\gamma^t \\ &\le \frac{\epsilon}{1-\gamma} \end{aligned}$$

The theorem follows since $error(\pi) = |E^{\pi,s_0}[error(\pi,\sigma)]| \le \frac{\epsilon}{1-\gamma}$. □

Computing approximate optimal policy: discounted return

We can now describe how to compute a near optimal policy for the discounted return. We need a sample of size $m \ge \frac{R_{max}}{2\epsilon^2} \log \frac{2|S|\,|A|}{\delta}$ for each $R(s,a)$. Given the sample, we compute $\widehat{r}(s,a)$. As we saw, with probability $1-\delta$ we have for every (s,a) that $|r(s,a) - \widehat{r}(s,a)| \le \epsilon$. Now we can compute the policy $\widehat{\pi}^*$ for the estimated rewards $\widehat{r}_t(s,a)$. Again, the main goal is to show that $\widehat{\pi}^*$ is a near optimal policy.

Claim 8.3. *Assume that for every (s,a) we have $|r(s,a) - \widehat{r}(s,a)| \le \epsilon$. Then,*

$$J_\gamma^{\pi^*}(s_0) - J_\gamma^{\widehat{\pi}^*} \le \frac{2\epsilon}{1-\gamma}$$

Proof. From the definition of $error(\pi)$ and since for any $\pi \in SD$ we showed that $error(\pi) \le \frac{\epsilon}{1-\gamma}$, we have,

$$J_\gamma^{\pi^*}(s_0) - \widehat{J}_\gamma^{\pi^*}(s_0) \le error(\pi^*) \le \frac{\epsilon}{1-\gamma}$$

and

$$\widehat{J}_\gamma^{\widehat{\pi}^*}(s_0) - J_\gamma^{\widehat{\pi}^*}(s_0) \le error(\widehat{\pi}^*) \le \frac{\epsilon}{1-\gamma}$$

Since $\widehat{\pi}^*$ is optimal for $\widehat{r}$ we have

$$\widehat{J}_\gamma^{\pi^*}(s_0) \le \widehat{J}_\gamma^{\widehat{\pi}^*}(s_0)$$

The claim follows by adding the three inequalities. □

8.2.3 Estimating the transition probabilities

We now estimate the transition probabilities. Again, we will look at the observed model. Namely, for a given state-action (s,a), we consider the transitions (s,a,s'_i), for $1 \le i \le m$. We define the observed transition distribution,

$$\widehat{p}(s'|s,a) = \frac{|\{i : s'_i = s'\}|}{m}$$

Our main goal would be to evaluate the observed model as a function of the sample size m.

We start by considering two Markov chains which differ by at most α for each transition. Namely, we have Markov chains M_1 and M_2, where $M_1[i,j] = \Pr[i \to j] = p_1(j|i)$ and $M_2[i,j] = \Pr[i \to j] = p_2(j|i)$. We assume that for every i,j we have $|M_1[i,j] - M_2[i,j]| \le \alpha$. We would like to relate α to the statistical distance between the state distributions generated by the two Markov chains. Clearly if $\alpha \approx 0$ then the probabilities will be almost identical, but we like to get a quantitative bound on the difference, which will allow us to derive the sample size m.

We start with a general well-known observation about distributions.

Theorem 8.4. *Let q_1 and q_2 be two distributions over S. Let $f : S \to [0, F_{max}]$. Then,*

$$|E_{s\sim q_1}[f(s)] - E_{s\sim q_2}[f(s)]| \le F_{max}\|q_1 - q_2\|_1$$

where $\|q_1 - q_2\|_1 = \sum_{s\in S} |q_1(s) - q_2(s)|$.

Proof.

$$\begin{aligned}
|E_{s\sim q_1}[f(s)] - E_{s\sim q_2}[f(s)]| &= |\sum_{s\in S} f(s)q_1(s) - \sum_{s\in S} f(s)q_2(s)| \\
&\le \sum_{s\in S} f(s)|q_1(s) - q_2(s)| \\
&\le F_{max}\|q_1 - q_2\|_1
\end{aligned}$$

□

Therefore, we would like to bound the L_1-norm between the state distributions generated by M_1 and M_2.

Theorem 8.5. *Let q_1^t and q_2^t be the distribution over states after trajectories of length t of M_1 and M_2, respectively. Then,*

$$\|q_1^t - q_2^t\|_1 \le \alpha|S|t$$

Proof. Let x_0 be the distribution of the start state. Then $q_1^t = x_0 M_1^t$ and $q_2^t = x_0 M_2^t$. The proof is by induction on t. Clearly, for $t = 0$ we have $q_1^0 = q_2^0 = x_0$.

We start with a basic facts about matrix norms. Let $\|M\|_\infty = max_i \sum_j |M[i,j]|$. Then

$$\|zM\|_1 = \sum_j |\sum_i z[i]M[i,j]| \leq \sum_{i,j} |z[i]| \cdot |M[i,j]| \leq \sum_i |z[i]| \cdot \|M\|_\infty \leq \|z\|_1 \|M\|_\infty \tag{8.1}$$

This implies the following two simple facts. First, let q be a distribution, i.e., $\|q\|_1 = 1$, and M a matrix with all the entries at most α, i.e., $|M[i,j]| \leq \alpha$ which implies $\|M\|_\infty \leq \alpha|S|$. Then,

$$\|qM\|_1 \leq \|q\|_1 \|M\|_\infty \leq \alpha|S| \tag{8.2}$$

Second, let M be a row-stochastic matrix, implies that $\|M\|_\infty = 1$. Then,

$$\|zM\|_1 \leq \|z\|_1 \|M\|_\infty \leq \|z\|_1 \tag{8.3}$$

For the induction step, let $z^t = q_1^t - q_2^t$.

$$\begin{aligned} \|q_1^t - q_2^t\|_1 &= \|x_0 M_1^t - x_0 M_2^t\|_1 \\ &= \|q_1^{t-1} M_1 - (q_1^{t-1} + z^{t-1}) M_2\|_1 \\ &\leq \|q_1(M_1 - M_2)\|_1 + \|z^{t-1} M_2\|_1 \\ &\leq \alpha|S| + \alpha|S|(t-1) = \alpha|S|t \end{aligned}$$

where in the last inequality, for the first term we used the first fact and for the second term we used the second fact with the inductive claim. □

Approximate model and simulation lemma

We define an α-*approximate model* as follows. A model $\widehat{M}$ is an α-approximate model of M if for every state-action (s,a) we have: (1) $|\widehat{r}(s,a) - r(s,a)| \leq \alpha$ and (2) for every s' we have $|\widehat{p}(s'|s,a) - p(s'|s,a)| \leq \alpha$.

The following simulation lemma, for the finite horizon case, guarantees that approximate models have similar return.

Lemma 8.1. *Assume that model $\widehat{M}$ is an α-approximate model of M. For the finite horizon return, for any policy $\pi \in MD$, we have*

$$|J_T^\pi(s_0; M) - J_T^\pi(s_0; \widehat{M})| \leq \epsilon$$

for $\alpha \leq \frac{\epsilon}{R_{max}|S|T^2}$

Proof. By Theorem 8.5 the distance between the state distributions of M and $\widehat{M}$ at time t s bounded by $\alpha|S|t$. Since the maximum reward is R_{max}, by Theorem 8.4 the difference is bounded by $\sum_{t=0}^{T} \alpha|S|tR_{max} \leq \alpha|S|T^2 R_{max}$. For $\alpha \leq \frac{\epsilon}{R_{max}|S|T^2}$ implies that the difference is at most ϵ. $\square$

We now switch to the simulation lemma, for the discounted return case, which also guarantees that approximate models have similar return.

Lemma 8.2. *Assume that model $\widehat{M}$ is an α-approximate model of M. For the discounted return, for any policy $\pi \in MD$, we have*

$$|J^{\pi}_{\gamma}(s_0; M) - J^{\pi}_{\gamma}(s_0; \widehat{M})| \leq \epsilon$$

for $\alpha \leq \frac{c\epsilon(1-\gamma)^2}{R_{max}|S|\log^2(R_{max}/(\epsilon(1-\gamma)))}$, for some constant $c > 0$.

Proof. We briefly sketch the proof. We will use the proof for the finite horizon case. which has $\alpha = \frac{\epsilon/2}{R_{max}|S|T^2}$ to get error at most $\epsilon/2$.

We can now use the effective horizon of the discounted case which is $T = \frac{1}{1-\gamma}\log\frac{R_{max}}{\epsilon(1-\gamma)/2}$ which guarantees that the error increases by at most $\epsilon/2$. We can use this is the finite horizon return bound to derive the lemma. $\square$

putting it all together

We want with high probability $(1-\delta)$ to have accuracy α. To get accuracy α we need $m = O(\frac{1}{\alpha^2}\log(|S|^2|A|/\delta))$ samples for each state-action pair (s,a). Plugging in the value of α we have, for the finite horizon,

$$m = O(\frac{R^2_{max}}{\epsilon^2}|S|^2 T^4 \log(|S|\ |A|/\delta))$$

and for the discounted return

$$m = O(\frac{R^2_{max}}{\epsilon^2}|S|^2 \frac{1}{(1-\gamma)^4}\log(|S|\ |A|/\delta)\log^2\frac{R_{max}}{\epsilon(1-\gamma)})$$

Assume we have a sample of m for each (s,a). Then with probability $1-\delta$ we have an α-approximate model $\widehat{M}$. We find an optimal policy $\widehat{\pi}^*$ for $\widehat{M}$. This implies that $\widehat{\pi}^*$ is a 2ϵ-optimal policy. Namely,

$$|J^*(s_0) - J^{\widehat{\pi}^*}(s_0)| \leq 2\epsilon$$

We can now look on the dependency of the sampling bounds on the parameters.

1. The error scales like $\frac{R_{max}^2}{\epsilon^2}$ which looks like the right bound, even for estimation of random variables expectations.
2. The dependency on the horizon is necessary, although it is probably not optimal.
3. The dependency on the number on the number of states $|S|$, is due to the fact that we like a very high approximation of the next state distribution. Simply we need to approximate $|S|^2$ entries, so for this the bound is reasonable. However, we will show that for approximation the optimal policy we can do better.

8.2.4 Improved sample bound: using approximate value iteration

Recall, that the value iteration works as follows. Initially, we set the values arbitrarily,

$$V_0 = \{V_0(s)\}_{s \in S}$$

In iteration n we compute

$$\begin{aligned} V_{n+1}(s) &= \max_{a \in A}\{r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V_n(s')\} \\ &= \max_{a \in A}\{r(s,a) + \gamma E_{s' \sim p(\cdot|s,a)}[V_n(s')]\} \end{aligned}$$

We showed that $\lim_{n\to\infty} V_n = V^*$, and that the convergence rate is $O(\frac{\gamma^n}{1-\gamma}R_{max})$. This implies that if we run for N, where $N = \frac{1}{1-\gamma}\log\frac{R_{max}}{\epsilon(1-\gamma)}$, we have an error of at most ϵ.

We would like to approximate the value iteration algorithm using a sample. Namely, for each (s,a) we have a sample of size m, i.e., $\{(s,a,s'_i)\}_{i\in[1,m]}$ The value iteration using the sample would be,

$$\widehat{V}_{n+1}(s) = \max_{a \in A}\{r(s,a) + \gamma \frac{1}{m}\sum_{i=1}^{m}\widehat{V}_n(s'_i)\}$$

The intuition is that if we have a large enough sample, it will approximate the value iteration. We set m such that for every (s,a) and any iteration $n \in [1,N]$ we have:

$$|E[\widehat{V}_n(s')] - \frac{1}{m}\sum_{i=1}^{m}\widehat{V}_n(s'_i)| \leq \epsilon$$

and also

$$|\widehat{r}(s,a) - r(s,a)| \leq \epsilon$$

We can have this for $m = O(\frac{V_{max}^2}{\epsilon^2})$ where V_{max} bound the maximum value. I.e., for finite horizon $V_{max} = TR_{max}$ and for discounted return $V_{max} = \frac{R_{max}}{1-\gamma}$.

Assume that we have, for every $s \in S$,

$$|\widehat{V}_n(s) - V_n(s)| \leq \lambda$$

Then

$$|\widehat{V}_{n+1}(s) - V_{n+1}(s)| \leq \epsilon + \gamma\lambda \leq \lambda$$

where the last inequality holds for $\lambda \geq \frac{\epsilon}{1-\gamma}$.

Therefore, if we sample $m = O(\frac{1}{\epsilon^2} \log \frac{N|S|\,|A|}{\delta})$, we have that with probability $1-\delta$ for every (s,a) the approximation is at most ϵ. This implies that the approximate value iteration has error at most $\lambda = \frac{\epsilon}{1-\gamma}$.

The main result is that we can run Value Iteration algorithm for N iterations and approximate well the optimal value function and policy. The implicit drawback is that we are approximating only the optimal policy, and cannot evaluate an arbitrary policy.

8.3 On-Policy Learning

8.3.1 Learning Deterministic Decision Process

Recall that a Deterministic Decision Process is modeled by a directed graph, where the states are the vertices, and each action is associate with an edge.

We first define the observed model. Given an observation set $\{(s_t, a_t, r_t, s_{t+1})\}$, we define an observed model $\widehat{M}$, where $\widehat{f}(s_t, a_t) = s_{t+1}$ and $\widehat{r}(s,a) = r_t$. For (s,a) which do not appear in the observation set, we define $\widehat{f}(s,a) = s$ and $\widehat{r}(s,a) = R_{max}$.

First we claim that for any $\pi \in SD$ the completion can only increase the value.

$$\widehat{V}^\pi(s;\widehat{M}) \geq V^\pi(s;M)$$

The increase holds for any trajectory, and note that once π reaches (s,a) that was not observed, its reward will be R_{max} forever. (This is since $\pi \in SD$.)

We can now present the on-policy learning algorithm.

1. At time T let $\{(s_t, a_t, r_t, s_{t+1}) : 0 \leq t \leq T-1\}$ be the previous observations.

2. Build the observed model $\widehat{M}_T$.

3. Compute $\widehat{\pi}^*_T$, the optimal policy for $\widehat{M}_T$.

4. Use $a_T = \widehat{\pi}^*_T(s_T)$.

5. Observe the reward r_T and the next state s_{T+1}.

We first show that the model $\widehat{M}_T$ cannot change too often.

Lemma 8.3. *The observed model $\widehat{M}_T$ can change at most $|S|\ |A|$ times.*

Proof. Each time we change the observed model, we add one (s, a) to be known. Since there are $|S|\ |A|$ such pairs, this bounds the number of changes. □

Next we show that we either make progress in the next $|S|$ steps or we never make any more changes.

Lemma 8.4. *Either we change $\widehat{M}_T$ in some time $t \in [T, T+|S|]$ or we never change $\widehat{M}_T$.*

Proof. The model is deterministic. If we do not make any update in the next $|S|$ steps, the policy $\widehat{\pi}^*_T \in MD$ will close a cycle and continue on this cycle forever. Hence, the model will never change. □

We can now state the convergence of the algorithm to the optimal policy.

Theorem 8.6. *After $T = |S|^2|A|$ time steps $\widehat{\pi}^*_T$ is optimal.*

Proof. We showed that the number of changes is at most $|S|\ |A|$, and the time between changes is at most $|S|$. This implies that after time $T = |S|^2|A|$ we never change.

The return of $\widehat{\pi}^*_T$ after time T is identical in $\widehat{M}_T$ and M, since all the edges it traverses are known. Let $\widehat{V}^*$ be its return. Assume that the policy π^* has a strictly higher return in M, say $V^* > \widehat{V}^*$. This implies that the return of π^* is at least $V^* > \widehat{V}^*$ in $\widehat{M}_T$, since the rewards in $\widehat{M}_T$ are always at least those in M. This contradicts the fact that $\widehat{\pi}^*_T$ is optimal for $\widehat{M}_T$. □

8.3.2 On-policy learning MDP

Similar to the DDP, we will use the principle of *Optimism in face of uncertainty.* Namely, we substitute the unknown quantities by the maximum possible values.

Similar to DDP, we will partition the state-action pairs (s, a) `known` and `unknown`. The main difference is that in a DDP it is sufficient to have a single sample to move (s, a) from `unknown` to `known`. In a general MDP we need have a larger sample to move (s, a) from `unknown` to `known`. Otherwise, the algorithm would be very similar.

We describe algorithm `R-MAX`, which performs on-policy learning of MDPs.

Initialization: Initially we set for each (s, a) a next state distribution which always return to s, i.e., $p(s|s, a) = 1$ and $p(s'|s, a) = 0$ for $s' \neq s$. We set the reward to be maximal, i.e., $r(s, a) = R_{max}$. We mark (s, a) to be `unknown`.

Execution: At time t. (1) Build a model $\widehat{M_t}$, explained later. (2) Compute $\widehat{\pi}^*_t$ the optimal policy for $\widehat{M_t}$, and (3) Execute $a_t = \widehat{\pi}^*_t(s_t)$ and observe r_t and s_{t+1}.

Building a model At time t, if the number of samples of (s, a) is *exactly* m for the *first time*, then: modify $p(\cdot|s, a)$ to the observed transition distribution, $r(s, a)$ to the average observed reward for (s, a) and mark (s, a) as `known`. Note that we update each (s, a) only once, when it moves from `unknown` to `known`.

Here is the basic intuition for algorithm `R-MAX`. We consider the discounted return. Fix a large enough horizon T, at least the effective horizon. Assume we run $\widehat{\pi}^*_t$ for T time steps. Either, we explore a state-action (s, a) which is `unknown`, in this case we make progress on the exploration. Also this can happen at most $m|S|\,|A|$ times. Alternatively, we did not reach any state-action (s, a) which is `unknown`, in which case we are optimal on the observed model, and near optimal on the true model.

For the analysis consider the event W, which is that event that we visit a state-action (s, a) which is `unknown` during the block of T time steps. For the return of $\widehat{\pi}^*_t$ we have,

$$V^{\widehat{\pi}^*_t} \geq V^* - \Pr[W]\frac{R_{max}}{1-\gamma} - \lambda$$

where λ is the approximation error, and we can bound it by $\epsilon/2$ by setting m large enough.

We consider two cases, depending on the probability of W. First, we consider the case that the probability of W is small. If $\Pr[W] \leq \frac{\epsilon(1-\gamma)/2}{R_{max}}$, then $V^{\widehat{\pi}^*_t} \geq V^* - \epsilon/2 - \epsilon/2$, since we assume that $\lambda \leq \epsilon/2$.

Second, we consider the case that the probability of W is large. If $\Pr[W] > \frac{\epsilon(1-\gamma)/2}{R_{max}}$. Then, there is a good probability to visit an `unknown` (s, a), but this can happen at most $m|S|\,|A|$. Therefore, the expected number of such blocks is at most $m|S|\,|A|\frac{R_{max}}{\epsilon(1-\gamma)/2}$.

This implies that only in

$$m|S|\ |A|\frac{R_{max}}{\epsilon(1-\gamma)/2}$$

blocks, the algorithm $\mathtt{R}-\mathtt{MAX}$ will be more than ϵ sub-optimal, i.e., have an expected return less than $V^*-\epsilon$.

8.4 Bibliography Remarks

The first polynomial time model based learning algorithm is E^3 of [?]. While we did not outline the E^3 algorithm, we did describe the effective horizon and the simulation lemma from there.

The improved sampling bounds for the optimal policy using approximate value iteration is following [?].

The `R-MAX` algorithm is due to [?].

Analysis of related models, especially the `PAC-MDP` model appears in [?, ?].

Chapter 9

Tabular learning: Model free

In this and the next lecture we will look at model free learning algorithms. In this lecture we will look at the Q-learning algorithms, and its on-policy variant SARSA. We will also look at Monte-Carlo methods. In the next lecture we will look at temporal differences algorithms, $TD(\lambda)$, as well as evaluating one policy while following a different policy (using importance sampling). If we will have time next lecture, we will look also at actor-critic methodology.

9.1 Online approximation of mean

Before we start looking at model-free learning algorithms, we look at a very simple task, approximating the average of a random variable using samples. Our main goal would be to do it in an online way while maintaining minimal information, namely the average.

Assume we have a random variable $R \in [0, 1]$ with $\mu = E[R]$. We observe T samples, $r_1, \ldots, r_T$. In the batch setting we simply compute the average of the samples, $\hat{\mu}_T = (1/T)\sum_{t=1}^{T} r_t$. We can bound $|\mu - \hat{\mu}_T|$ using Chernoff-Hoeffding concentration bounds, as we discussed last lecture, when considering model-based learning.

We can actually perform the computation of the average in an online way.

$$\hat{\mu}_T = \frac{1}{T}\sum_{t=1}^{T} r_t = \frac{T-1}{T}\hat{\mu}_{T-1} + \frac{1}{T}r_T = \hat{\mu}_{T-1} + \frac{1}{T}(r_T - \hat{\mu}_{T-1})$$

This lead naturally to the *exponential averaging* where we have a sequence of

$\alpha_t \in (0,1)$ and

$$\bar{\mu}_T = \bar{\mu}_{T-1} + \alpha_T(r_T - \bar{\mu}_{T-1}) = \sum_{t=1}^{T} \beta_t r_t$$

where $\beta_t = \alpha_t \prod_{i=1}^{t-1}(1-\alpha_i)$. Note that $\sum_{t=1}^{T} \beta_t = 1$.

We would like to show that the approximation $\bar{\mu}$ concentrates around the mean μ. For this we will introduce the McDiarmid inequality.

The setting of the McDiarmid inequality is the following. There is a domain X, which can be for example $[0,1]$. We have n random variables $X_i \in X$. There is a function f which maps the realization of the n random variables to a value. Namely, $f : X^n \to \mathbb{R}$.

We define c_i, the sensitivity of the function f to the i-th input, to be

$$c_i = \max_{x} \max_{a_1, a_0 \in X} |f(x_1, \ldots, x_{i-1}, a_0, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, a_1, x_{i+1}, \ldots, x_n)|$$

The basic idea of McDimid's inequality is that with high probability the observed value of f is close to its expected value. Formally,

Lemma 9.1 (McDiarmid's inequality)**.**

$$\Pr[|f(x) - E[f(x)| \geq \epsilon] \leq e^{-2\epsilon^2/(\sum_i c_i^2)}$$

We can use McDirmid's inequality to recover the concentration bounds for a simple average. Define $avg(x_1, \ldots, x_n) = (1/n)\sum_{i=1}^{n} x_i$, where $x_i \in [0,1]$. The sensitivity of avg to any input i is exactly $c_i = 1/n$. This implies that $\sum_i c_i^2 = \sum_i 1/n^2 = 1/n$. Therefore, McDiarmid's inequality gives us,

$$\Pr[|avg(x) - E[avg(x)]| \geq \epsilon] \leq e^{-2\epsilon^2 n}$$

Note that the bound is very similar to the Chernoff-Hoeffding bound.

We can now use the McDiarmid's inequality for the weighted average case. Define $wavg(x_1, \ldots, x_n) = \sum_{i=1}^{n} \beta_i x_i$, where $x_i \in [0,1]$. The sensitivity of $wavg$ to any input i is exactly $c_i = \beta_i$. Therefore, McDiarmid's inequality gives us,

$$\Pr[|wavg(x) - E[wavg(x)]| \geq \epsilon] \leq e^{-\epsilon^2/(\sum_i \beta_i^2)}$$

For the case of exponential averaging, with a fixed parameter α, we have $\beta_t = \alpha(1-\alpha)^{t-1}$. This implies that

$$\sum_{t=1}^{T} \beta_t^2 = \alpha^2 \frac{1-(1-\alpha)^T}{1-(1-\alpha)^2} \approx \frac{\alpha}{2-\alpha}$$

This implies that if we like to have an accuracy ϵ with confidence $1-\delta$ we can set $\alpha \approx \epsilon^2/(\log(1/\delta))$.

9.2 Q-Learning: Deterministic Decision Process

To demonstrate some key ideas of Q-learning, we start with a simplified learning algorithm that is suitable for a *Deterministic Decision Process (DDP)* model, namely:

$$\begin{aligned} s_{t+1} &= f(s_t, a_t) \\ r_t &= r(s_t, a_t) \end{aligned}$$

We consider the discounted return criterion:

$$\begin{aligned} V^\pi(s) &= \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,, \quad \text{given } s_0 = s, a_t = \pi(s_t) \\ V^*(s) &= \max_\pi V^\pi(s) \end{aligned}$$

where V^* is the value function of the optimal policy.

Recall our definition of the Q-function (or *state-action value function*), specialized to the present deterministic setting:

$$Q^*(s, a) = r(s, a) + \gamma V^*(f(s, a))$$

The optimality equation is then

$$V^*(s) = \max_a Q^*(s, a)$$

or, in terms of Q^* only:

$$Q^*(s, a) = r(s, a) + \gamma \max_{a'} Q^*(f(s, a), a')$$

Our learning algorithm runs as follows:

- *Initialize:* Set $\hat{Q}(s, a) = Q_0(s, a)$, for all s, a.
- At each stage $t = 0, 1, \ldots$:
  - Observe s_t, a_t, r_t, s_{t+1}.
  - Update $\hat{Q}(s_t, a_t)$: $\quad \hat{Q}_{t+1}(s_t, a_t) := r_t + \gamma \max_{a'} \hat{Q}_t(s_{t+1}, a')$

We note that this algorithm does not tell us how to choose the actions a_t.

Theorem 9.1 (Convergence of Q-learning for DDP).
Assume a DDP model. If each state-action pair is visited <u>*infinitely-often*</u>*, then* $\lim_{t\to\infty} \hat{Q}_t(s, a) = Q^*(s, a)$*, for all* (s, a).

Proof. Let

$$\Delta_t \triangleq \|\hat{Q}_t - Q^*\|_\infty = \max_{s,a} |\hat{Q}_t(s,a) - Q^*(s,a)| \,.$$

Then at every stage t:

$$\begin{aligned} |\hat{Q}_{t+1}(s_t,a_t) - Q(s_t,a_t)| &= |r_t + \gamma \max_{a'} \hat{Q}_t(s_{t+1},a') - (r_t + \gamma \max_{a''} Q(s_{t+1},a''))| \\ &= \gamma |\max_{a'} \hat{Q}_t(s_{t+1},a') - \max_{a''} Q(s_{t+1},a'')| \\ &\le \gamma \max_{a'} |\hat{Q}_t(s_{t+1},a') - Q(s_{t+1},a')| \ \le \ \gamma \Delta_t \,. \end{aligned}$$

Consider now some interval $[t, t_1]$ over which all state-action pairs (s,a) appear at least once. Using the above relation and simple induction, it follows that $\Delta_{t_1} \le \gamma \Delta_t$. Since $\gamma < 1$ and since there is an infinite number of such intervals by assumption, it follows that $\Delta_t \to 0$, as t goes to infinity. □

9.3 Q-learning: Markov decision process

The Q-learning algorithm:

– initialize $\hat{Q}_0$.
– At time t: Observe (s_t, a_t, r_t, s_{t+1}), and let

$$Q_{t+1}(s_t,a_t) := Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)]$$

Let $\Gamma_t = r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)$. The basic intuition is that if $Q_t = Q^*$ then we have that $E[\Gamma_t] = 0$. The main challenge is that in the updates we use Q_t rather than Q^*. We also need to handle the stochastic nature of the updates, where there is both stochastic rewards and stochastic next state.

The next theorem states the main convergence property of Q-leanring.

Theorem 9.2 (Q-learning convergence)**.**
Assume every state-action pair (s,a) occurs infinitely often, and the step size $\alpha_t(s,a)$ has the properties: (1) $\sum_t \alpha_t(s,a) I(s_t = s, a_t = a) = \infty$, and (2) $\sum_t \alpha_t^2(s,a) I(s_t = s, a_t = a) = O(1)$,
Then, Q_t converges with probability 1 to Q^*

We will not give the details of the proof, but will only outline the main building blocks and the methodology.

9.3.1 Stochastic Approximation

There is a general framework of stochastic approximation algorithms. We will outline the main results of that literature, as we need it to show convergence of the learning algorithms.

The iterative algorithm takes the following general form:

$$X_{t+1} = (1 - \alpha_t(s))X_t(s) + \alpha_t(s)((HX_t)(s) + w_t(s))$$

where H is a (possibly non-linear) operator.

We will mainly look at (B, γ) *well behaved* iterative algorithms, where $B > 0$ and $\gamma \in (0, 1)$, which have the following properties:

1. Step size: (1) $\sum_t \alpha_t(s,a)I(s_t = s, a_t = a) = \infty$, and (2) $\sum_t \alpha_t^2(s,a)I(s_t = s, a_t = a) = O(1)$,

2. Noise: $E[w_t(s)|h_{t-1}] = 0$ and $|w_t(s)| \leq B$, where h_{t-1} is the history up to time t.

3. Contraction: There exists X^* such that for any X we have $\|HX - X^*\|_\infty \leq \gamma\|X - X^*\|_\infty$. (This implies that $HX^* = X^*$. Note that if H is a contracting operator, this property is guarantee to hols.)

The following is the convergence theorem for well behaved iterative algorithms.

Theorem 9.3 (Iterative Stochastic Approximation: convergence).
Let X_t be a sequence that is generated by a (B, γ) well behaved iterative algorithm. Then X_t converges with probability 1 to X^*.

We will not give a proof of this important theorem, but we will try to sketch the main proof methodology.

There are two distinct parts to the iterative algorithms. The part (HX_t) is contracting, in a deterministic manner. If we had only this part (say, $w_t = 0$ always) then the contraction property of H will give the convergence (as we saw before). The main challenge is the addition stochastic noise w_t. The noise is unbiased, so on average the expectation is zero. Also, the noise is bounded by a constant B. This implies that if we average the noise, then the average should be very close to zero.

The proof works in phases. In each phase we have a deterministic contraction using the operator H. This implies that the deterministic contraction implies that the space contracts by $\gamma < 1$. We have to take care of the stochastic noise. We make the phase long enough so that the average of the noise is less than $(1-\gamma)/2$ factor. This implies that the space contracts by $(1+\gamma)/2 < 1$. This implies that each phase contracts by a constant and eventually we have convergence.

9.3.2 Q-learning as an iterative stochastic approximation algorithm

We first define the operator H.

$$(Hq)(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \max_{a'} q(s',a')]$$

The contraction of H is established as follows,

$$\begin{aligned}\|Hq_1 - Hq_2\|_\infty &= \gamma \max_{s,a} |\sum_{s'} p(s'|s,a)[\max_{b_1} q_1(s',b_1) - \max_{b_2} q_2(s',b_2)| \\ &\le \gamma \max_{s,a} \max_{b,s'} |q_1(s',b) - q_2(s',b)| \\ &\le \gamma \|q_1 - q_2\|_\infty\end{aligned}$$

In this section we re-write the Q-learning algorithm to follow the iterative stochastic approximation algorithms, so that we will be able to apply Theorem 9.3.

Recall that

$$Q_{t+1}(s_t,a_t) := (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[r_t + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)]$$

Let $\Phi_t = r_t + \gamma \max_{a'} Q_t(s_{t+1},a')$. This implies that $E[\Phi_t] = (HQ_t)(s_t,a_t)$. We can define the noise term as $w_t(s_t,a_t) = \Phi_t - (HQ_t)(s_t,a_t)$ and have $E[w_t(s_t,a_t)] = 0$. In addition $|w_t(s_t,a_t)| \le \frac{R_{max}}{1-\gamma}$.

We can now rewrite the Q-learning, as follows,

$$Q_{t+1}(s_t,a_t) := (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)[(HQ_t)(s_t,a_t) + w_t(s_t,a_t)]$$

In order to apply Theorem 9.3, we have the properties on the noise w_t, and of the contraction of H. Therefore, we can derive Theorem 9.2, since the step size requirement is part of the theorem.

9.3.3 Step size

We need for the step size to have two properties. The first is that $\sum_t \alpha_t(s,a)I(s_t = s, a_t = a) = \infty$, which intuitively implies that we can potentially reach any value. This is important, since we might have errors on the way, and this guarantees that the step sizes are large enough to possibly reach correct any error. (It does not guarantee that it will correct the errors, only that the step size is large enough to allow it.)

The second requirement from the step size is that $\sum_t \alpha_t^2(s,a)I(s_t=s,a_t=a)=O(1)$. This requirement is that the step size are not too large. It will guarantee that once we are close to the correct value, the step size will be small enough that we actually converge, and not bounce around.

Think of the following experiment. Suppose from some time t we update using only Q^*, then we clearly would like to converge. The large enough step size will guarantee that we will reach Q^* neighborhood. The small enough step size will guarantee that we will converge inside the neighborhood.

Many times step size are simply a function of the number of visits to (s,a), which we denote by $n(s,a)$, and this widely used in practice. Two leading examples are:

1. *Linear step size:* $\alpha_t(s,a)=1/n(s,a)$. We have that $\sum_{n=1}^{N} 1/n=\ln(N)$ and therefore $\sum_{n=1}^{\infty} 1/n=\infty$. Also, $\sum_{n=1}^{\infty} 1/n^2=\pi^2/6$

2. *Polynomial step size*: For $\theta\in(1/2,1)$ we have $\alpha_t(s,a)=1/(n(s,a))^{\theta}$. We have that $\sum_{n=1}^{N} 1/n^{\theta}\approx(1-\theta)^{-1}N^{1-\theta}$ and therefore $\sum_{n=1}^{\infty} 1/n^{\theta}=\infty$. Also, $\sum_{n=1}^{\infty} 1/n^{2\theta}\leq\frac{1}{2\theta-1}$, since $2\theta>1$.

The linear step size, although many times popular in practice, might lead to slow converges. Here is a simple example. We have a single state s and single action a and $r(s,a)=0$. However, we start with $Q_0(s,a)=1$. We will analyze the convergence with the linear step size. Our update is,

$$Q_t=(1-\frac{1}{t})Q_{t-1}+\frac{1}{t}[0+\gamma Q_{t-1}]=(1-\frac{1-\gamma}{t})Q_{t-1}$$

When we solve the recursion we get that $Q_t=\Theta(1/t^{1-\gamma})$. [1] This implies that for $t\leq c(1/\epsilon)^{1/(1-\gamma)}$ we have $Q_t\geq\epsilon$.

In contrast, if we use a polynomial step size, we have,

$$Q_t=(1-\frac{1}{t^{\theta}})Q_{t-1}+\frac{1}{t^{\theta}}[0+\gamma Q_{t-1}]=(1-\frac{1-\gamma}{t^{\theta}})Q_{t-1}$$

When we solve the recursion we get that $Q_t=\Theta(e^{-(1-\gamma)t^{1-\theta}})$. This implies that for $t\geq c\frac{1}{1-\gamma}\log^{1/(1-\theta)}(1/\epsilon)$ we have $Q_t\leq\epsilon$. This is a poly-logarithmic dependency on ϵ, which is much better. Also, not that θ is in our control, and we can set for example $\theta=2/3$. The setting of γ has a huge influence on the objective function and the effective horizon.

[1] $Q_t=\prod_{i=1}^{t}(1-(1-\gamma)/i)\approx\prod_{i=1}^{t}e^{-(1-\gamma)/i}=e^{-\sum_{i=1}^{t}(1-\gamma)/i}\approx e^{-(1-\gamma)\ln t}=t^{-(1-\gamma)}$.

9.4 SARSA: on-policy Q-learning

We would like to have an on-policy variant of Q-learning, which is called SARSA. The name comes from the fact that the feedback we observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, ignoring the subscripts we have SARSA. Note that in this case the actions are actually under the control of the algorithm.

When designing the algorithm we need to think of two contradicting objectives in selecting the actions. The first is the need to explore, perform each action infinitely often. This implies that we need, for each state s, to have that $E[\sum_t \pi_t(a|s)] = \infty$. Then by Borel-Cantelli lemma we have with probability 1 an infinite number of times that we select action a in state s (actually, we need independence of the events, or at least a Martingale property, which holds in our case). On the other hand we would like not only our estimates to converge, as done in Q-learning, but also the return to be near optimal. For this we need the action selection to converge to being greedy with respect to the Q function.

The SARSA algorithm

– initialize $\hat{Q}_0$.
– At time t: Observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, and let

$$Q_{t+1}(s_t, a_t) := Q_t(s_t, a_t) + \alpha_t(s_t, a_t)[r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

where $a_{t+1} = \pi(s_{t+1}; Q_t)$, namely, our policy selects an action a_{t+1} for state s_{t+1}, dependent on the values of Q_t.

Selecting the action: We give two simple ways to select the action by $\pi(s; Q)$. Let

$$\bar{a} = \arg\max Q(s, a)$$

The ϵ_t-greedy, has as a parameter a sequence of ϵ_t and selects either: (1) with probability $1 - \epsilon_t$ sets $\pi(s; Q) = \bar{a}$, or (2) with probability $\epsilon_t/|A|$, selects $\pi(s; Q) = a$, for each $a \in A$. Common values for ϵ_t are linear, $\epsilon_t = 1/t$, or polynomial, $\epsilon_t = 1/t^\theta$.

The *soft-max*, has as a parameter a sequence of $\beta_t \geq 0$ and select $\pi(s; Q) = a$, for each $a \in A$, with probability $\frac{e^{\beta_t Q(s,a)}}{\sum_{a' \in A} e^{\beta_t Q(s,a')}}$. Note that for $\beta_t = 0$ we get the uniform distribution and for $\beta_t \to \infty$ we get the maximum. We would like the schedule of the β_t to go to infinity (become greedy) but need it to be slow enough (so that each action appears infinitely often).

SARSA convergence: Convergence of the Q_t to Q^* would follow immediately from the basic properties of the Q learning, we just need to guarantee that we explore each action infinitely often. Then, the result will follow immediately from the convergence of Q-learning.

The convergence of the return of SARSA to that of Q^* is more delicate. Recall that we do not have such a claim about Q-learning, since it is an off-policy method. For the convergence of the return we need to make our policy 'greedy enough', in the sense that it has enough exploration, but guarantees a high return through the greedy actions.

Expected SARSA algorithm: In SARSA there are two sources of errors, or stochastic behavior. One is from the environment, through the selection of the next state and the rewards, both not under our control. The other is from our policy, which select a stochastic action. We will attempt to reduce that part of the noise in the following variation of the SARSA algorithm.

The Expected SARSA algorithm:
– initialize $\hat{Q}_0$.
– At time t: Observe (s_t, a_t, r_t, s_{t+1}), and let

$$Q_{t+1}(s_t, a_t) := Q_t(s_t, a_t) + \alpha_t(s_t, a_t)[r_t + \gamma E_{a \sim \pi(s_{t+1};Q_t)} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

Select $a_{t+1} = \pi(s_{t+1}; Q_t)$

The min idea is to replace the sample a_{t+1} in the estimate of Q_{t+1}, by an averaging over a_{t+1}. We can do it, since we define the policy $\pi(s; Q)$. By the averaging, Expected SARSA reduces the variance of the estimates and hopefully converges faster.

9.5 Monte-Carlo Algorithm

Monte-Carlo methods learn directly from experience in a model free way. The idea is very simple. In order to estimate the value of a state under a given policy, i.e., $V^\pi(s)$, we consider runs of the policy π from state s and average them. the method does not assume any dependency between the different states, and does not even assume a Markovian environment, which is both a plus (less assumptions) and a minus (longer time to learn). We will concentrate on the case of an episodic MDP, namely, generating finite length episodes in each run. A special case of an episodic MDP is a finite horizon return, where all the episodes can be viewed as having the same length.

Assume we have a fixed policy π, which for each state s selects action a with probability $\pi(a|s)$. Using π we generate an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$. The observed return of the episode is $G = \sum_{i=1}^{k} r_i$. We are interested in the expected return of an episode conditioned on the initial state, i.e., $V^\pi(s) = E[\sum_{i=1}^{k} r_i | s_1 = s]$. Note that k is a random variable, which is the length of the episode.

Fix a state s, and assume you observed returns $G_1, \ldots, G_m$, all starting at state s. The Monte-Carlo estimate for the state s would be $\hat{V}^\pi(s) = \frac{1}{m}\sum_{i=1}^{m} G_i$. The main issue that remains is how do we generate the samples G_i for a state s. Clearly, if we assume we can reset the MDP to any state, we are done, However, we do not want to assume that we can reset the MDP to any state s and start an episode from there.

9.5.1 Generating the samples

Initial state only We use only the initial state of the episode. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ we update only $\hat{V}^\pi(s_1)$. This is clearly an unbiased estimate, but has many drawbacks. First, most likely it is not the case that every state can be an initial state, what do we do with such states. Second, it seems very wasteful, updating only a single state per episode.

First visit We update every state that appears in the episode, but update it only once. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ for each state s that appear in the episode, we consider the first appearance of s, say s_j, and update $\hat{V}^\pi(s)$ using $G = \sum_{i=j}^{k} r_i$. Namely, we compute the actual return from the first visit to state s, and use it to update our approximation.

Every visit We do an update during each step of in the episode. Namely, give an episode $(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)$ for each state s_j that appear in the episode, we update each $\hat{V}^\pi(s_j)$ using $G = \sum_{i=j}^{k} r_i$. Namely, we compute the actual return from every state s_j to the end, and use it to update our approximation. Note that a state can be updated multiple times using this approach.

9.5.2 First versus Every visit

We consider a simple test case. We have a two state MDP, actually a Markov Chain. In the initial state s_1 we have a reward of 1 and with probability $1-p$ we stay in that state and with probability p move to the terminating state s_2. See Figure 9.1.

The expected value is $V(s_1) = 1/p$, which is the expected length of an episode. (Note that the return of an episode is its length, since all the rewards are 1.) Assume

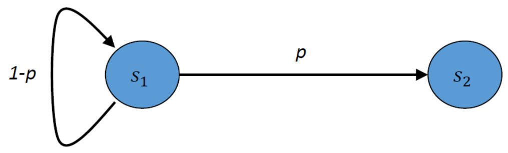


Figure 9.1: The situated agent

we observe a single trajectory, $(s_1, s_1, s_1, s_1, s_2)$, and all the rewards are 1. What would be a reasonable estimate for the expected return from s_1.

First visit take the naive approach, considers the return from the first occurrence of s_1, which is 4, and uses this as an estimate. Every visit considers three runs from state s_1, we have: $(s_1, s_1, s_1, s_1, s_2)$ with return 4, (s_1, s_1, s_1, s_2) with return 3, (s_1, s_1, s_2) with return 2, and (s_1, s_2) with return 1. Every visit averages the four and has $G = (4 + 3 + 2 + 1)/4 = 2.5$. On the face of it, the estimate of 4 seems to make more sense.

9.5.3 Maximum likelihood MDP model

We can consider what is the maximum likelihood model for the MDP. Namely, what value of p would maximize the probability of observing the sequence $(s_1, s_1, s_1, s_1, s_2)$. The likelihood of the sequence is, $(1 - p)^3 p$. We like to solve for

$$p^* = \arg\max(1 - p)^3 p$$

Taking the derivative we have $(1 - p)^3 - 3(1 - p)^2 p = 0$, which give $p^* = 1/4$.

For the maximum likelihood model ML we have $p^* = 1/4$ and therefore $V(s_1; ML) = 4$.

In general the Maximum Likelihood model value does not always coincide with the First Visit Monte-carlo estimate. However we can make the following connection.

Clearly, when updating state s using First Visit, we ignore all the episodes that do not include s, and also for each the remaining episode, that do include s, we ignore the prefix until the fist appearance of s. Lets modify the sample by deleting those parts (episodes in which s does not appear, and for each episode that s appears, start it at the first appearance of s). Call this the *reduced sample.*

Theorem 9.4. *Let M be the maximum likelihood MDP for the reduced sample. The expected value of s in M, i.e., $V(s; M)$, is identical to the* First Visit *estimate of s.*

9.5.4 Every visit minimizes MSE

Recall that the MSE is summing over all the observation the squared error. Assume we have $s_{i,j}$ as the j-th state in the i-th episode, and it has reward $r_{i,j}$. Let $\hat{V}(s_{i,j})$ be the estimate that `Every Visit` for state $s_{i,j}$. (Note that states $s_{i,j}$ are not unique, and we have $s = s_{i_1,j_1} = s_{i_2,j_2}$.) The square error is

$$SE = \sum_{i,j} (\hat{V}(s_{i,j}) - r_{i,j})^2$$

For a fixed state s we have

$$SE(s) = \sum_{i,j:s=s_{i,j}} (\hat{V}(s) - r_{i,j})^2$$

To minimize the square error of s we have

$$V^{se}(s) = \frac{\sum_{i,j:s=s_{i,j}} r_{i,j}}{|(i,j) : s = s_{i,j}|},$$

which is exactly the `Every Visit` Monte-carlo estimate.

9.5.5 First versus Every visit

The `First Visit` updates are unbiased, since the different trajectories are from different episodes. For each episode that update is an independent sample of the return.

For `Every Visit` the situation is more complicated, since there are different updates from the same episode, and therefore they are dependent. Consider the case of Figure 9.1. For a single episode of length k we have that the sum of the rewards is $k(k+1)/2$, since there are updates of lengths $k, \ldots, 1$. The number of updates is k, so we have that the estimate of a single episode is $(k+1)/2$. When we take the expectation we have that $E[(k+1)/2] = (1/p+1)/2$ which is different from the expected value of $1/p$. (Note that the `Every Visit` updates using k and $E[k] = 1/p$ which is also the expected value.)

When we have multiple episodes, the situation gets better. The reason is that we sum separately the rewards, and the number of occurrences. This implies that we have

$$E[V^{ev}(s_1) = \frac{E[k(k+1)/2]}{E[k]} = \frac{1}{p},$$

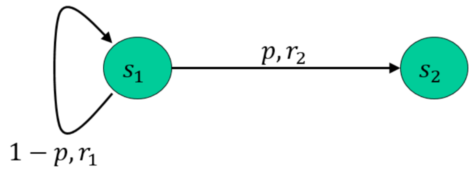


Figure 9.2: The situated agent

since $E[k^2] = 2/p^2 - 1/p$. This implies that if we average many episodes we will get an almost unbiased estimate using `Every Visit`.

We did all this on the example of Figure 9.1, but this indeed generalizes. Given an arbitrary episodic MDP, consider the following mapping. For each episode, mark the places where state s appears (the state we want to approximate its value). We now have a distribution of rewards from going from s back to s. Since we are in an episodic MDP, we also have to terminate, and for this we can add another state, from which we transition from state s and have the reward distribution as the rewards from the last appearance of s until the end of the episode.

This implies that we have two states MDP as described in Figure 9.2.

For this MDP, the value is $V(s_1) = \frac{1-p}{p} r_1 + r_2$. The single episode expected estimate of `Every Visit` is $V(s_1) = \frac{1-p}{2p} r_1 + r_2$. The m episodes expected estimate of `Every Visit` is $V(s_1) = \frac{m}{m+1} \frac{1-p}{p} r_1 + r_2$. This implies that if we have a large number of episodes the bias of the estimate becomes negligible.

9.5.6 Monte-Carlo control

The main idea is to learn the Q function. Simply we do an update for every (s, a). (The updates can be either `Every Visit` or `First Visit`.) The problem is that we need the policy to be "exploring", otherwise we will not have enough information about the actions the policy does not perform.

For the control, we can maintain an estimates of the Q function. Each time we reach a state s, we select a "near-greedy" action, for example, use ϵ-greedy.

We will show that updating from one ϵ-greedy policy to another ϵ-greedy policy, using policy improvement, does increase the value of the policy.

Recall that an ϵ-greedy policy, can be define in the following way. For every state s there is an action $\bar{a}_s$, which is the preferred action. The policy does the following: (1) with probability $1-\epsilon$ selects action $\bar{a}$. (2) for each action $a \in A$, with probability $\epsilon/|A|$, select action a.

Assume we have an ϵ-greedy policy π_1. We compute Q^{π_1} and define π_2 to be ϵ-greedy with respect to Q^{π_1}.

Theorem 9.5. *For any ϵ-greedy policy π_1, the ϵ-greedy improvement policy π_2 has $V^{\pi_2} \geq V^{\pi_1}$.*

Proof. Let $\bar{a}_s = \arg\max_a Q^{\pi_1}(s,a)$.

$$\begin{aligned}
E_{a\sim\pi_1(\cdot|s)}[Q^{\pi_1}(s,a)] &= \sum_{a\in A} \pi_2(a|s) Q^{\pi_1}(s,a) \\
&= \frac{\epsilon}{|A|}\sum_{a\in A} Q^{\pi_1}(s,a) + (1-\epsilon) Q^{\pi_1}(s,\bar{a}_s) \\
&\geq \frac{\epsilon}{|A|}\sum_{a\in A} Q^{\pi_1}(s,a) + (1-\epsilon)\sum_{a\in A} \frac{\pi_1(a|s) - \epsilon/|A|}{1-\epsilon} Q^{\pi_1}(s,a) \\
&= \sum_{a\in A} \pi_1(a|s) Q^{\pi_1}(s,\bar{a}) = V^{\pi_1}(s)
\end{aligned}$$

The inequality follows, since we are essentially concentrating of the action that $\pi^1(\cdot|s)$ selects with probability $1-\epsilon$, and clearly $\bar{a}_s$, by definition, guarantees a higher value.

From the basic policy improvement properties we have that

$$V^{\pi_2} \geq \max_a Q^{\pi_1}(s,a) \geq E_{a\sim\pi_1(\cdot|s)}[Q^{\pi_1}(s,a)] \geq V^{\pi_1}.$$

□

9.5.7 Monte-Carlo: pros and cons

The main benefits of the Monte-Carlo updates are:

1. Very simple and intuitive
2. Does not assume the environment is Markovian
3. Extends naturally to function approximation (more in future lectures)
4. Unbiased updates (using `First Visit`).

The main drawback of the Monte-Carlo updates are:

1. Suited mainly for episodic environment
2. Need to wait for the end of episode to update.
3. Biased updates (using `Every Visit`).

9.6 Bibliography Remarks

The Q-learning outline of the asymptotic convergence and the step size analysis follows [**?**].

Expected SARSA was presented in [**?**]

The comparison of the `First Visit` and `Every Visit` is based on [**?**].

Part of the outline borrows from David Silver class notes and the the book of Sutton and Barto [**?**].

In this lecture we will continue looking at model-free learning. In the previous lecture we presented: (1) Q-learning, which is an off-policy learning algorithm that learns the optimal Q^* function, (2) `SARSA` which is an on-policy variant of Q-learning where we need to select actions, and (3) Monte-Carlo which is a model-free algorithm to learn the value function from episodes.

In this lecture we will look on temporal differences methods, which works in an online fashion. We will start with $TD(0)$ which uses the most recent observations for the updates, we will continue with methods that allow for a longer look-ahead, and then continue with $TD(\lambda)$ which averages multiple look-ahead estimations.

In general, temporal differences (TD) methods, learn directly from experience, and therefore are model-free methods. Unlike Monte-Carlo algorithms, they will use incomplete episodes for the updates, and they are not restricted to episodic MDPs. The TD methods update their estimates given the current observation and in that direction, similar in spirit to Q-learning and SARSA.

9.7 TD(0)

Fix a policy π. The goal is to learn the value function $V^\pi(s)$ for every $s \in S$. (The same goal as Monte-Carlo learning.) The methods will maintain an estimate $V_t(s)$ for $V^\pi(s)$ and use it for the updates. Unlike Monte-Carlo, there will be an interaction between the different estimates.

As a starting point, we can recall the *value iteration* algorithm.

$$V_{t+1}(s) = E^\pi[r(s,a) + \gamma V_t(s')]$$

We have shown that value iteration converges, namely $V_t \to V^\pi$.

Assume we sample (s_t, a_t, r_t, s_{t+1}). Assume that $\pi \in SD$, namely a stationary deterministic policy. Then,

$$E^\pi[\hat{V}_t(s_t)] = E^\pi[r_t + \gamma \hat{V}_t(s_{t+1})] = E^\pi[r(s,a) + \gamma \hat{V}_t(s') | s = s_t, a = \pi(s)]$$

The $TD(0)$ will do an update in the direction of the sample, namely, $[r_t + \gamma \hat{V}_t(s_{t+1})]$.

$TD(0)$ **Algorithm**

– Initialize $\hat{V}(s)$ arbitrarily.
– Update using (s_t, a_t, r_t, s_{t+1}):

$$\hat{V}(s_t) = \hat{V}(s_t) + \alpha_t(s_t, a_t)[r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)]$$

where $\alpha_t(s,a)$ is the step size for (s,a) at time t.

We define the *temporal difference* to be

$$\Delta_t = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$$

The $TD(0)$ update becomes:

$$\hat{V}(s_t) = \hat{V}(s_t) + \alpha_t(s_t, a_t)\Delta_t$$

In class we gave as a motivating example the driving example from [**?**]. (See the slides.)

We would like to compare the $TD(0)$ and the Monte-Carlo (MC) algorithms. Here is a simple example with four states $S = \{A, B, C, D\}$ where $\{C, D\}$ are terminal states and in $\{A, B\}$ there is one action (essentially, the policy selects a unique action). Assume we observe eight episodes. One episode is $(A, 0, B, 0, C)$, one episode $(B, 0, C)$, and six episodes $(B, 1, D)$. We would like to estimate the value function of the non-terminal states. For $V(B)$ both $TD(0)$ and MC will give $6/8 = 0.75$. The interesting question would be: what is the estimate for A? MC will average only the trajectories that include A and will get 0 (only one trajectory which gives 0 reward). The $TD(0)$ will consider the value from B as well, and will give an estimate of 0.75. (Assume that the $TD(0)$ continuously updates using the same episodes until it converges.)

We would like to better understand the above example. For the example the empirical MDP will have a transition from A to B, with probability 1 and reward 0, from B we will have a transition to C with probability 0.25 and reward 0 and a transition to D with probability 0.75 and reward 1. (See, Figure 9.3.) The value of A in the empirical model is 0.75. In this case the empirical model agrees with the $TD(0)$ estimate, we show that this holds in general.

Let the empirical model of a sample be define as follows. Let $n(s,a)$ be the number of times (s,a) appears in the sample. Given a sample (s_t, a_r, r_t, s_{t+1}) for $1 \leq t \leq T$, we define the empirical model as follows. The rewards $\hat{r}(s,a)$ are the average rewards of (s,a), i.e., $\hat{r}(s,a) = \frac{\sum_{t:s_t=s,a_t=a} r_t}{n(s,a)}$ and $\hat{p}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$, where $n(s,a,s')$ is the number of samples that have $s_{t+1} = s'$ when $s_t = s$ and $a_t = a$.

The main theorem would state that the value of π on the empirical model is identical to that of $TD(0)$ (running on the sample until convergence, namely, continuously sampling uniformly $t \in [1, T]$ and using (s_t, a_r, r_t, s_{t+1}) for the $TD(0)$ update).

Theorem 9.6. *Let V^π_{TD} be the estimated value function of π when we run $TD(0)$ until convergence. let V^π_{EM} be the value function of π on the empirical model. Then $V^\pi_{TD} = V^\pi_{EM}$.*

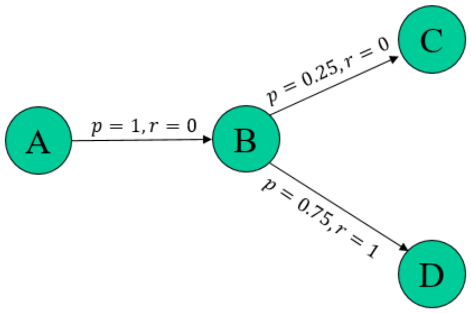


Figure 9.3: The situated agent

Proof sketch. The update of $TD(0)$ is $\hat{V}(s_t) = \hat{V}(s_t) + \alpha_t(s_t, a_t)\Delta_t$, where $\Delta_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$. At convergence we have $E[\Delta_t] = 0$ and hence,

$$\hat{V}(s) = \frac{1}{n(s,a)} \sum_{s_{t+1}: s_t = s, a_t = a} r_t + \gamma\hat{V}(s_{t+1}) = \hat{r}(s,a) + \gamma E_{s' \sim \hat{p}(\cdot|s,a)}[\hat{V}(s')]$$

where $a = \pi(s)$. □

The proof of the convergence of $TD(0)$ is very similar to that of Q-learning. We will show that $TD(0)$ is an instance of the stochastic approximation algorithm, as presented in previous lecture, and the convergence proof will follow from this.

Theorem 9.7 (Convergence $TD(0)$)**.** *If the step size has the properties that for every (s,a) we have $\sum_t \alpha_t(s,a) = \infty$ and $\sum_t \alpha_t^2(s,a) = O(1)$, then $\hat{V}$ converges to V^π, with probability* 1.

We will show the convergence using the general theorem for stochastic approximation iterative algorithm (which we saw in the previous lecture).

We first define a linear operator for the policy π,

$$(Hv)(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s))v(s')$$

This is a contracting operator

$$\begin{aligned}\|Hv_1 - Hv_2\|_\infty &= \gamma \max_s |\sum_{s'} p(s'|s,\pi(s))(v_1(s') - v_2(s'))| \\ &\le \gamma \max_{s'} |v_1(s') - v_2(s')| \\ &\le \gamma \|v_1 - v_2\|_\infty\end{aligned}$$

This implies that H is γ-contracting.

We now would like to re-write the $TD(0)$ update to resemble the stochastic approximation iterative algorithm. The $TD(0)$ update is,

$$V_{t+1}(s_t) = (1-\alpha_t)V_t + \alpha_t \Phi_t$$

where $\Phi_t = r_t + \gamma V_t(s_{t+1})$. We would like to consider the expected value of Φ_t. Clearly, $E[r_t] = r(s_t, \pi(s_t))$ and $s_{t+1} \sim p(\cdot|s_t, a_t)$. This implies that $E[\Phi_t] = (HV_t)(s_t)$. Therefore, we can define the noise term w_t as follows,

$$w_t(s_t) = [r_t + \gamma V_t(s_{t+1})] - (HV_t)(s_t) \ ,$$

and have $E[w_t] = 0$. We can bound $|w_t| \leq V_{max} = \frac{R_{max}}{1-\gamma}$, since the value function is bounded by V_{max}.

Returning to $TD(0)$, we can write

$$V_{t+1}(s_t) = (1-\alpha_t)V_t + \alpha_t[(HV_t)(s_t) + w_t(s_t)]$$

The requirement of the step size appears in the statement of the theorem (and so holds). The noise w_t has both $E[w_t] = 0$ and $|w_t| \leq V_{max}$. And the operator H is contracting with a fix-point V^π. Therefore, we established Theorem 9.7.

When we compare $TD(0)$, to MC and both to Dynamic Programming (DP) we can view it as different ways of computing the value function.

MC We have $V^\pi(s) = E[R_t|s_t = s]$. In MC we observe the return, R_t, of episodes, and their mean is what we like to estimate.

$TD(0)$ We have $V^\pi(s) = E[r_t + \gamma V^\pi(s_{t+1})|s_t = s]$. In $TD(0)$ we observe samples of r_t, we use our estimate for $V^\pi(s_{t+1})$ and the expectation is what we like to estimate (assuming our estimates converge).

DP We have $V^\pi(s) = \sum_a \pi(a|s)[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V^\pi(s')]$. In DP we have the entire model, and use it to compute expectations.

We can see the difference between $TD(0)$ and MC in the Markov Chain in Figure 9.4. To get an approximation of state s_2, i.e., $|\hat{V}(s_2) - \frac{1}{2}| \approx \epsilon$. The Monte-Carlo will require $O(1/(\beta\epsilon^2))$ episodes (out of which only $O(1/\epsilon^2)$ start at s_2) and the $TD(0)$ will require only $O(1/\epsilon^2 + 1/\beta)$ since the estimate of s_3 will converge after $1/\epsilon^2$ episodes which start from s_1.

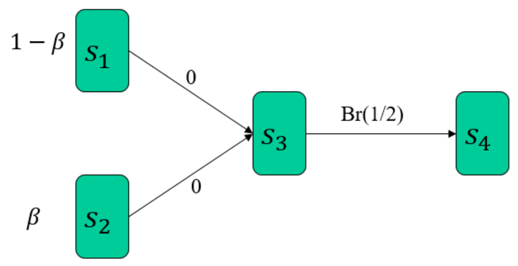


Figure 9.4: Markov Reward Chain

9.8 TD: Multiple look-ahead

The $TD(0)$ uses only the current reward and state. Given (s_t, a_t, r_t, s_{t+1}) it updates $\Delta_t = R_t^{(1)}(s_t) - \hat{V}(s_t)$ where $R_t^{(1)}(s_t) = r_t + \gamma \hat{V}(s_{t+1})$. We can also consider a two step look-ahead as follows. Given $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$ we can update using $\Delta_t^{(2)} = R_t^{(2)}(s_t) - \hat{V}(s_t)$ where $R_t^{(2)}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 \hat{V}(s_{t+2})$. Using the same logic, we have that this is a temporal difference that uses a two time steps.

We can generalize this to any n-step look-ahead and define $R_t^{(n)}(s_t) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \hat{V}(s_{t+n})$ and updates $\Delta_t^{(n)} = R_t^{(n)}(s_t) - \hat{V}(s_t)$.

We can relate the $\Delta_t^{(n)}$ to the Δ_t as follows:

$$\Delta_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i \Delta_{t+i}$$

This follows since

$$\sum_{i=0}^{n-1} \Delta_{t+i} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \sum_{i=0}^{n-1} \gamma^{i+1} \hat{V}(s_{t+i+1}) - \sum_{i=0}^{n-1} \gamma^i \hat{V}(s_{t+i}) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \hat{V}(s_{t+n}) - \hat{V}(s_t)$$

Using the n-step look-ahead we have $\hat{V}(s_t) = \hat{V}(s_t) + \alpha_t \Delta_t^{(n)}$ where $\Delta_t^{(n)} = R_t^{(n)}(s_t) - \hat{V}(s_t)$. Note that the operator $R_t^{(n)}$ is γ^n-contracting, namely

$$\|R_t^{(n)}(V_1) - R_t^{(n)}(V_2)\|_\infty \leq \gamma^n \|V_1 - V_2\|_\infty$$

We can use any parameter n for the n-step look-ahead. If the episode ends before step n we can pad it with rewards zero. This implies that for $n = \infty$ we have that n-step look-ahead is simply the Monte-Carlo estimate. However, we need to select some parameter n. An alternative idea is to simply average over the possible parameters n. One simple way to average is to use exponential averaging with a parameter $\lambda \in (0, 1)$. This implies that the weight of parameter n is $(1 - \lambda)\lambda^{n-1}$.

This leads us to the $TD(\lambda)$ update:

$$\hat{V}(s_t) = \hat{V}(s_t) + \alpha_t(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \Delta_t^{(n)}$$

Remark: While both γ and λ are used to generate exponential decaying values, their goal is very different. The discount parameter γ defines the objective of the MDP, the goal that we like to maximize. The exponential averaging parameter λ is used to average over the different look-ahead parameters, and is selected to optimize the convergence.

In the slides there is an example of a random walk and its convergence taken from [**?**].

The above describes the forward view of $TD(\lambda)$, where we average over future rewards. If we will try to implement it in a strict way this will lead us to wait until the end of the episode, since we will need to first observe all the rewards. Fortunately, there is an equivalent form of the $TD(\lambda)$ which uses a *backward view.* The backward view updates at each time step, using an incomplete information. At the end of the episode, the updates of the forward and backward updates will be the same.

The basic idea of the backward view is the following. Fix a time t and a state $s = s_t$. We have at time t a temporal difference $\Delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t)$. Consider how this Δ_t affects all the previous times $\tau < t$ where $s_\tau = s = s_t$. The influence is exactly $(\gamma\lambda)^{t-\tau}\Delta_t$. This implies that for every such τ we can do the desired update, however, we can aggregate all those updates to a single update. Let,

$$e_t(s) = \sum_{\tau \leq t} (\gamma\lambda)^{t-\tau} = \sum_{\tau=1}^{t} (\gamma\lambda)^{t-\tau} I(s_\tau = s)$$

The above $e_t(s)$ defines the *eligibility trace* and we can compute it online using

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

which result in the update

$$\hat{V}_{t+1}(s) = \hat{V}_t(s) + \alpha_t e_t(s) \Delta_t$$

Note that for $TD(0)$ we have that $\lambda = 0$ and the eligibility trace becomes $e_t(s) = I(s = s_t)$. This implies that we update only s_t and $\hat{V}_{t+1}(s_t) = \hat{V}_t(s_t) + \alpha_t \Delta_t$.

TD(λ) **algorithm**

– Initialization: Set $\hat{V}(s) = 0$ (or any other value), and $e_0(s) = 0$.
– Update: observe (s_t, a_t, r_t, s_{t+1}) and set

$$\begin{aligned}\Delta_t &= r_t + \gamma \hat{V}_t(s_{t+1}) - \hat{V}(s_t)\\ e_t(s) &= \gamma\lambda e_{t-1}(s) + I(s = s_t)\\ \hat{V}_{t+1}(s) &= \hat{V}_t(s) + \alpha_t e_t(s)\Delta_t\end{aligned}$$

To summarize, the benefit of $TD(\lambda)$ is that it interpolates between $TD(0)$ and Monte-carlo updates, and many times achieves superior performance to both. One can show the equivalence of the forward and backward $TD(\lambda)$ updates (see [**?**]). Also, $TD(\lambda)$ can be written as a stochastic approximation iterative algorithm, and one can derive its convergence.

9.8.1 $SARSA(\lambda)$

We can use the idea of eligibility traces also in other algorithms, such as SARSA. Recall that given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ the update of SARSA is

$$r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

Similarly, we can define an n-step look-ahead $q_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n Q_t(s_{t+n}, a_{t+n})$ and set $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(q_t^{(n)} - Q_t(s_t, a_t))$.

We can now define $SARSA(\lambda)$ using exponential averaging with parameter λ. Namely, we define $q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$. This makes the forward view of $SARSA(\lambda)$ to be $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(q_t^\lambda - Q_t(s_t, a_t))$.

Similar to $TD(\lambda)$, we can define a backward view using eligibility traces:

$$\begin{aligned}e_0(s,a) &= 0\\ e_t(s,a) &= \gamma\lambda e_{t-1}(s,a) + I(s = s_t, a = a_t)\end{aligned}$$

For the update we have

$$\begin{aligned}\Delta_t &= r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\\ Q_{t+1}(s,a) &= Q_t(s,a) + \alpha_t e_t(s,a)\Delta_t\end{aligned}$$

9.9 Importance Sampling

Importance sapling is a simple general technique to estimate the mean with respect to a given distribution, while sampling from a different distribution. To be specific, let Q be the sampling distribution and P the evaluation distribution. The basic idea is the following

$$E_{x\sim P}[f(x)] = \sum_x P(x)f(x) = \sum_x Q(x)\frac{P(x)}{Q(x)}f(x) = E_{x\sim Q}[\frac{P(x)}{Q(x)}f(x)]$$

This implies that given a sample $\{x_1, \ldots, x_m\}$ from Q, we can estimate $E_{x\sim P}[f(x)]$ using $\sum_{i=1}^m \frac{P(x_i)}{Q(x_i)} f(x_i)$.

The importance sampling gives an unbiased estimator, but the variance of the estimator might be huge, since it depends on $P(x)/Q(x)$.

We would like to apply the idea of importance sampling to learning in MDPs. Assume that there is a policy π that selects the actions, and there is a policy ρ that we would like to evaluate. For the importance sampling, given a trajectory, we need to take the ratio of probabilities under ρ and π.

$$\frac{\rho(s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{T+1})}{\pi(s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{T+1})} = \prod_{t=1}^{T} \frac{\rho(a_t|s_t)}{\pi(a_t|s_t)}$$

where the eq7uality follows since the reward and transition probabilities are identical, and cancel.

For Monte-Carlo, the estimates would be

$$G^{\rho/\pi} = \prod_{t=1}^{T} \frac{\rho(a_t|s_t)}{\pi(a_t|s_t)} (\sum_{t=1}^{T} r_t)$$

and we have

$$\widehat{V}^{\rho}(s_1) = \widehat{V}^{\rho}(s_1) + \alpha(G^{\rho/\pi} - \widehat{V}^{\rho}(s_1))$$

This updates might be huge, since we are multiplying many small numbers.

For the $TD(0)$ the updates will be

$$\Delta_t^{\rho/\pi} = \frac{\rho(a_t|s_t)}{\pi(a_t|s_t)} r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$$

and we have

$$\widehat{V}^{\rho}(s_1) = \widehat{V}^{\rho}(s_1) + \alpha(\Delta_t^{\rho/\pi} - \widehat{V}^{\rho}(s_1))$$

This update is much more stable, since we have only one factor multiplying the observed reward.

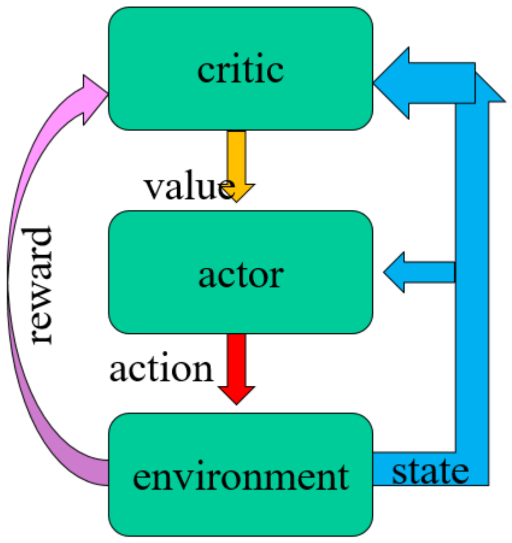


Figure 9.5: The situated agent

9.10 Actor-critic methodology

Actor-critic gives a general methodology of building reinforcement learning algorithms. It is composed from an actor, that selects the actions, ad a critic, that learns the value function. The actor observes the current state, and the value function, and selects an action. The critic, observes the current state (and action) and reward and outputs a value function. See Figure 9.5.

Chapter 10

Tabular learning: off-policy

Chapter 11

Large state space: value function approximation

This lecture starts looking at the case where the MDP models are large. In the current lecture we will look at approximating the value function. In the next lecture we will consider learning directly a policy and optimizing it.

When we talk about a large MDP, it can be in one of a few different reasons. The most common is having a large state space. For example, Backgammon has 10^{20} states, Go has 10^{170} and robot control has continuous state space. Another dimension is the action space, which can be even continuous in many applications (say, robots). Finally, we might have a complex dynamics which are hard to describe succinctly. The main challenge is that we need our algorithm to scale up to this enormous spaces.

Today we will consider function approximation for large MDPs, mainly large state spaces. Previously, we had a look-up table for the value function $V^\pi(s)$ or $Q^\pi(s,a)$, which we updated every time we encountered the state. In a large state space this will be infeasible. Not only that we do not have the memory, but even more importantly, we are unlikely to observe states re-occurring. We will need to make (implicit) assumptions about the MDP, which can be viewed similarly to our assumption in learning a classifier.

Specifically, we will have the value functions parameterized by *weights* w, namely, $\hat{V}(s;w) \approx V^\pi(s)$ and $\hat{Q}(s,a;w) \approx Q^\pi(s,a)$. We would like to guarantee generalization from the observed states and trajectories to unobserved states and trajectories. In the process we will update the weights w, using some learning procedure, most notably using MC or TD learning.

When considering a value function there are a few interpretations what exactly we mean. It can either be: (1) mapping from a state s to its expected return, i.e., $\hat{V}^\pi(s; w)$. (2) mapping from state-action pairs (s, a) to their expected return, i.e., $\hat{Q}^\pi(s, a; w)$. (3) mapping from states s to expected return of each action, i.e., $\langle \hat{Q}^\pi(s, a_i; w) : a_i \in A \rangle$. All the interpretations are valid, and our discussion will not distinguish between them. (Actually, for $\langle \hat{Q}^\pi(s, a_i; w) : a_i \in A \rangle$ we implicitly assume that the number of actions is small.)

We now need to discuss how will we build the approximating function. For this we can turn to the rich literature in Machine Learning and consider the popular hypothesis classes. For example: (1) Linear functions, (2) Neural networks, (3) Decision trees, (4) Nearest neighbors, (5) Fourier or wavelet basis, etc. We will concentrate on linear functions and neural networks, mainly since gradient based methods apply to them very naturally.

11.1 Basic Approximation

Before we start the discussion on the learning methods, we will do a small detour. We will discuss the effect of having an error in the value function we learn, and its effect on the outcome. Assume we have a value function V such that $\|V - V^*\|_\infty \leq \epsilon$. Let π be the greedy policy with respect to V, namely,

$$\pi(s) = \arg\max_a [r(s, a) + \gamma E_{s' \sim p(\cdot|s,a)}[V(s')]$$

We will consider in this lecture (mainly) the discounted return with a discount parameter $\gamma \in (0, 1)$.

Theorem 11.1. *Let V such that $\|V - V^*\|_\infty \leq \epsilon$ and π be the greedy policy with respect to V. Then,*

$$\|V^\pi - V^*\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma}$$

Proof. Consider two operators, the first H^π converges to V^π and

$$(H^\pi v)(s) = r(s, \pi(s)) + \gamma E_{s' \sim p(\cdot|s,\pi(s))}[v(s')]$$

The second H^* converges to V^* and

$$(H^* v)(s) = \max_a [r(s, a) + \gamma E_{s' \sim p(\cdot|s,a)}[v(s')]]$$

Since π is greedy with respect to V we have $H^\pi V = H^* V$ (but this does not hold for other value functions V').

Then,

$$\begin{aligned}\|V^\pi - V^*\|_\infty &= \|H^\pi V^\pi - V^*\|_\infty \\ &\le \|H^\pi V^\pi - H^\pi V\|_\infty + \|H^\pi V - V^*\|_\infty \\ &\le \gamma\|V^\pi - V\|_\infty + \|H^* V - H^* V^*\|_\infty \\ &\le \gamma\|V^\pi - V\|_\infty + \gamma\|V - V^*\|_\infty \\ &\le \gamma(\|V^\pi - V^*\|_\infty + \|V^* - V\|_\infty) + \gamma\|V - V^*\|_\infty\end{aligned}$$

where in the second inequality we used the fact that since since π is greedy with respect to V then $H^\pi V = H^* V$.

Reorganizing the inequality and recalling that $\|V^* - V\|_\infty \le \epsilon$, we have

$$(1-\gamma)\|V^\pi - V^*\|_\infty \le 2\epsilon\gamma$$

□

The above theorem says that if we have a small error in L_∞ norm, the effect is bounded. However, in most cases we will not be able to guarantee such an approximation! This is since we are unable even to verify the L_∞ norm of two value function efficiently, since we need to consider *all* states.

11.2 From RL to ML

We would like to reduce our reinforcement learning problem to a supervised learning problem. This will enable us to use any of the many techniques of machine learning to address the problem. Let us consider the basic ingredients of supervised learning. The most important ingredient is having a labeled sample, which is sampled i.i.d.

Let us start by considering an idealized setting. Fix a policy π that we like to learn its value function. We like to generate a sample

$$\{(s_1, V^\pi(s_1)), \ldots, (s_m, V^\pi(s_m))\}$$

We first need to discuss how to sample the states s_i in an i.i.d. way. We can generate trajectory, but we need to be careful, since adjacent states are definitely dependent! One solution is to space the sampling from the trajectory using the mixing time of π. This will give us samples s_i which are from the stationary distribution of π and are almost independent. For the action, we clearly will use π, no problem there.

The hardest, and most confusing, ingredient is the labels $V^\pi(s_i)$. In machine leaning we actually assume that someone gives us the labels to build a classifier (or alternatively, in unsupervised learning, we have no clear ground truth). If someone is able to give us $V^\pi(s)$, it seems like assuming the problem away, since this is what we would like to learn.

Actually we need to define three things: (1) states and actions, which we will do using π, (2) a loss function, which will give the tradeoffs between different approximation errors, and (3) labels, which we are not give explicitly.

We will define a differential loss function $J(w)$. This will enable us to compute the gradient of the loss function $\nabla_w J(w) = \langle \frac{\partial}{\partial w_1} J(w), \ldots, \frac{\partial}{\partial w_d} J(w)\rangle$, and update the weights in the negative direction of the gradient, namely, $\Delta_w = -0.5\alpha\nabla_w J(w)$, where α is the step size of the learning rate. In supervised learning we are guaranteed that the process will converge to a local minimum (or a saddle point).

Let us start by defining the loss function,

$$J(w) = \sum_s \mu(s)(V^\pi(s) - \hat{V}^\pi(s;w))^2$$

where $\mu(s)$ is the steady state distribution of π. So we can compute the gradient and update the weights. The Stochastic Gradient Descent (SGD) is simply sampling a single state s and using it, i.e.,

$$\Delta w_t = \alpha(V^\pi(s) - \hat{V}^\pi(s;w))\nabla\hat{V}^\pi(s;w)$$

We now need to build the sample, namely, how we replace $V^\pi(s)$. The basic idea is to find an unbiased estimator U_t such that $E[U_t|s_t] = V^\pi(s_t)$. We now will present a few different approaches to derive such estimators.

The most simple approach is to use Monte-Carlo sampling. Recall that in Monte-Carlo we have $R_T(s) = \sum_{t=1}^T r_t$, starting at the first visit of s. Clearly, we have $E[R_T(s)] = V^\pi(s)$, since samples are independent, so we can set $U_T(s) = R_T(s)$. The update becomes,

$$\Delta w_t = \alpha(R_t(s) - \hat{V}^\pi(s;w_t))\nabla_w\hat{V}^\pi(s;w)$$

We can try to use the same idea for $TD(0)$. The estimate of $TD(0)$ for s_t is $R_t(s_t) = r_t + \gamma\hat{V}^\pi(s_{t+1};w)$. The first problem is that this is a biased estimator since

$$V^\pi(s_t) = E[r_t + \gamma V^\pi(s_{t+1})] \neq E[r_t + \gamma\hat{V}(s_{t+1};w)]$$

We have an additional problem with the gradient, since we have w influencing also the target $R_t(s_t)$, something we do not have in MC (or generally in ML). For this reason $\nabla_w \hat{V}(s_t; w)$ is called a semi-gradient. The update becomes,

$$\Delta w(s) = \alpha[r_t + \gamma \hat{V}(s_{t+1}; w) - \hat{V}(s_t)]\nabla_w \hat{V}(s_t; w)$$

Finally we get to $TD(\lambda)$. Similar to $TD(0)$ we can now define the forward update to be,

$$\Delta w = \alpha[R_t^\lambda - \hat{V}(s_t; w)]\nabla_w \hat{V}(s_t; w)$$

and the backward update,

$$\begin{aligned} e_t &= \gamma\lambda e_{t-1} + \nabla_w \hat{V}(s_t; w) \\ \Delta_t &= r_t + \gamma \hat{V}(s_{t+1}; w) - \hat{V}(s_t; w) \\ \Delta w &= \alpha \Delta_t e_t \end{aligned}$$

11.3 Linear Functions

We first start with the state encoding. In general we assume that each state s is encoded by a vector $x(s) \in \mathbb{R}^d$. For notation, let $x(s) = (x_1(s), \ldots, x_d(s))$. This will be useful for any hypothesis, and specifically, linear function and neural networks.

A linear function is characterized by a vector of weights $w \in \mathbb{R}^d$. The value of the function is

$$\hat{V}(s; w) = w^\top x(s)$$

This implies that the SGD updates become

$$w_{t+1} = w_t + \alpha[U_t - \hat{V}(s_t; w_t)]x(s_t)$$

The main benefit of the linear function is when $d \ll |S|$. When $d = |S|$ we can simply encode a look-up table using w. Simply set $x_i(s) = I(s = s_i)$. Then, $\hat{V}(s; w) = \sum_{i=1}^d w_i I(s = s_i)$. This implies that $\hat{V}(s_i; w) = w_i$.

For a linear function the gradient is simply $x(s)$, i.e., $\nabla_w w^\top x(s) = x(s)$. The Monte-Carlo update will become

$$\Delta w = \alpha[R_t - \hat{V}(s_t; w_t)]x(s_t)$$

For $TD(0)$ we have

$$\Delta w = \alpha[r_t + \gamma \hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t)]x(s_t)$$

For $TD(\lambda)$, for the forward view we have

$$\Delta w = \alpha[R_t^\lambda - \hat{V}(s_t; w_t)]x(s_t)$$

For the backward view we have,

$$\begin{aligned} e_t &= \gamma\lambda e_{t-1} + x(s_t) \\ \Delta_t &= r_t + \gamma\hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t) \\ \Delta w &= \alpha\Delta_t e_t \end{aligned}$$

We would like to discuss the convergence of the different algorithms. Recall that the loss function is strongly convex, i.e.,

$$J(w) = \sum_s \mu(s)(V^\pi(s) - \hat{V}^\pi(s; w))^2$$

Therefore there is a unique w_{min} which minimizes $J(w)$. Monte-Carlo is simply SGD, and therefore the convergence and its rate follow from known results in optimization.

The convergence of TD is more problematic, since it uses a semi-gradient and not the true gradient. The good news is that for linear functions, when using an on-policy, there is a proof of convergence. However, the convergence is not to w_{min} but rather to w_{TD}. The difference is due to the fact that we are minimizing the expression

$$\sum_s \mu(s)(r(s, \pi(s)) + \gamma E_{s'}[\hat{V}(s'; w)] - \hat{V}^\pi(s; w))^2$$

The difference in the losses can be bounded as follows

$$J(w_{TD}) \leq \frac{1}{1-\gamma} J(w_{min})$$

11.4 Off-policy

We would like to see what is the effect that the samples come following a different policy, namely, an off-policy setting. There is no issue for Monte-Carlo, and the same logic would still be valid. For TD, we did not have any problem in the look-up model. We would like to see what can go wrong when we have a function approximation setting.

Consider the following part of an MDP (see Figure 11.1) consists of two modes, with a transition from the first to the second, with reward 0. The main issue is that the linear approximation gives the first node a weight w and the second $2w$. Assume we start with some value $w_0 > 0$. Each time we have an update for the two states we have

$$w_{t+1} = w_t + \alpha[0 + \gamma(2w_t) - w_t] = [1 + \alpha(2\gamma - 1)]w_t$$

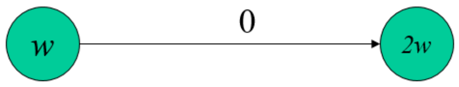


Figure 11.1: Two state snippet of an MDP

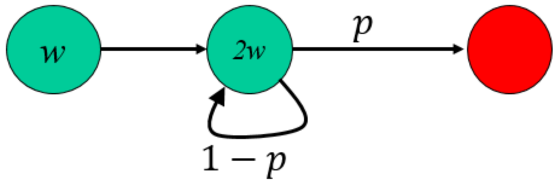


Figure 11.2: The three state MDP. All rewards are zero.

For $\gamma > 0.5$ we have $\alpha(2\gamma - 1) > 0$, and w_t diverges.

We are implicitly assuming that the setting is off-policy, since in an on-policy, we would continue from the second state, and eventually lower the weight.

To have a "complete" example consider the three state MDP in Figure 11.2. All the rewards are zero, and the main difference is that we have a new terminating state, that we reach with probability p.

Again, assume that we start with some $w_0 > 0$ We have three types of updates, one per possible transition. When we transition from the initial state to the second state we have

$$\Delta w = \alpha[0 + \gamma(2w_t) - w_t] \cdot 1 = \alpha(2\gamma - 1)w_t$$

The transition from the second state back to itself has an update,

$$\Delta w = \alpha[0 + \gamma(2w_t) - (2w_t)] \cdot 2 = -4\alpha(1 - \gamma)w_t$$

The transition to the terminal state we have

$$\Delta w = \alpha[0 + \gamma 0 - (2w_t)] \cdot 2 = -4\alpha w_t$$

When we use on-policy, we have all transitions. Assume that the second transition happens $n \geq 0$ times. Then we have

$$\frac{w_{t+1}}{w_t} = (1 + \alpha(2\gamma - 1))(1 - 4\alpha(1 - \gamma))^n(1 - 4\alpha) < 1 - \alpha$$

This implies that w_t converges to zero, as desired.

Now consider an off-policy that truncates the episodes after n transitions of the second state, where $n \ll 1/p$. This implies that in most updates we have

$$\frac{w_{t+1}}{w_t} = (1 + \alpha(2\gamma - 1))(1 - 4\alpha(1-\gamma))^n > 1$$

and therefore, for the right setting of n we have that the weight w_t diverges.

We might hope that the divergence is due to the online nature of the TD updates. We can consider an algorithm that in each iteration minimizes the square error. Namely,

$$w_{t+1} = \arg\min_w \sum_s [\hat{V}(s_t; w) - E^\pi[r_t + \gamma \hat{V}(s_{t+1}; w_t)]]^2$$

For the example of Figure 11.2 we have that

$$w_{t+1} = \frac{6\gamma - 4p}{5} w_t$$

So for $6\gamma - 4p > 5$ we have divergence. (Recall that $\gamma \approx 1 - \epsilon$ and $p \approx \epsilon$, so this would hold in most settings we are most interested in.)

What we have seen that there is a *deadly triad*:

1. Function Approximation

2. Bootstrapping methods, such as TD.

3. Off-policy training.

When the three conditions hold, we have counter examples for the convergence.

Here is a summary of the known convergence and divergence results in the literature:

	algorithm	look-up table	linear function	non-linear
on-policy	MC	+	+	+
on-policy	$TD(0)$	+	+	-
on-policy	$TD(\lambda)$	+	+	-
off-policy	MC	+	+	+
off-policy	$TD(0)$	+	-	-
off-policy	$TD(\lambda)$	+	-	-

11.5 Applications: games

11.5.1 Backgammon

Tesauro developed the TD-gammon in the late 1980's and 1990's. The system uses a neural network approximation, using a $TD(\lambda)$ updates. The initial system was able to win against other computer programs, which where hand-designed, and later matched the performance of the best human. For the training the system used self-play, namely playing against itself.

The neural network has a single hidden layer (with 40/80/160 nodes, in the three different generations of the project). There are 198 inputs. The hidden layer has sigmoidal nodes with $\sigma(z) = \frac{1}{1+e^{-z}}$. The output y can be viewed as the probability of winning from a given position.

The 198 inputs are composed as follows. For each of the 24 slots and for each of the two colors, we have four Boolean inputs. The first indicates if there is a single chip, the seocnd indicates two or more chips, the third indicates exactly three chips and the fourth indicates four or more chips. We have two Boolean variables that indicate who turn it is (one for white and one for black). Finally, we have two aggregate inputs, per color: (1) number of chips divided by 2 and (2) number of chips removed divided by 15.

The TD-gammon uses the standard updates.

$$\Delta w_t = \alpha(y_t - y_{t-1}) \sum_{k=1}^{t} \nabla \lambda^{t-k} \nabla_w y_k$$

When the game ends we replace y_t by z the outcome of the game.

The evaluation of the position is done using a max-min tree, whose depth is 1/2/3. The max-min tree has max layer where the value of a node is the maximum of their children, and min layer, where the value of a node is the minimum of their children. Given the tree (of size at most 20/400/8000 in different generations) we evaluate the leaves using the neural network, and propagate the values in the min-max tree to get the predicted value of a position.

The network is initialized with small random weights. Note that due to the game nature, even with random policies the game terminates. The training is done using self-play, where the number of games was $300K/800K/1500K$ in different generations.

One very intriguing aspect of the TD-gammon is that it *improved the human performance*. For example, when the first role is $4-1$, expects did the 1 move in location 6, while the TD-gammon moved location 24. Following the TD-gammon, experts changed their view and started playing those positions in the same way that TD-gammon did.

11.5.2 Attari games

The system was developed by DeepMind researchers during $2013-2015$. The system learns to play from the raw video frames, a variety of 49 games. The learning uses a neural network and employs Q-learning (and a Deep-Q-Network, DQN).

The neural network works as follows. There is a preprocessing stage that maps frames of size 210×160 with 128-colors, to 84×84 and 4-colors. This mapping is fixed and shared by all games. The neural network has three convolutional layers followed by one completely connected layer. The output layer has multiple outputs. The nodes in the network use RelU gates.

The DQN innovations are: (1) Experience reply, reusing the same example many times, (2) fixed Q-target, when setting the target update, and (3) clipping the error. To better understand the innovations, we need to consider the flow of the training.

In the training, we first select an action a_t, using ϵ-greedy policy. We observe the reward and the next state (s_t, a_t, r_t, s_{t+1}), and store them in the reply memory. We sample a mini-batch from the reply memory. For each sample we use the fixed-Q-target (more latter). We use the SGD to minimize the MSE.

For the fixed-Q-target, we do the update as follows. We have two sets of weights w_t, the current weights, and w_t^- as the fixed weights. We update w_t to w_t^- every (large) number of plays, setting $w_t = w_t^-$. The update at time t becomes:

$$\Delta w_t = \alpha \Gamma_t \nabla_{w_t} Q(s_t, a_t; w_t)$$

where

$$\Gamma = r_t + \gamma \max_a Q(s_{t+1}, a; w_t^-) - Q(s_t, a_t; w_t)$$

Note that w_t^- is used only to estimate the future return from s_{t+1}. The clipping enforces $\Gamma_t \in [-1, +1]$.

The system is able to match human performance in 29 out of 49 Attari games.

11.6 Bibliography Remarks

The work of Gerald Tesauro on TD-gammon is described in [?, ?].

The DeepMind system for playing Attari games is described in [?].

Part of the outline borrows from David Silver class notes and the the book of Sutton and Barto [?].

Chapter 12

Large state space: value function approximation: Policy Gradient Methods

This lecture continue looking at the case where the MDP models are large. In the previous lecture we looked at approximating the value function. In this lecture we will consider learning directly a policy and optimizing it.

The main advantages and disadvantages of policy optimization (rather than value function approximation) are the following:

1. Continuous action space. In such a case the policy optimization method is the most natural and the one which is used in practice (for applications such as robotics).

2. Convergence: While the policy optimization would normally converge, it will most likely converge to a local optimum.

3. High dimensional spaces: It can be fairly effective in selecting the actions (in contrast to learning accurate values).

4. Evaluation time would typically be longer (compared to value function approaches).

5. Stochastic Policy: allows naturally to have a stochastic policy.

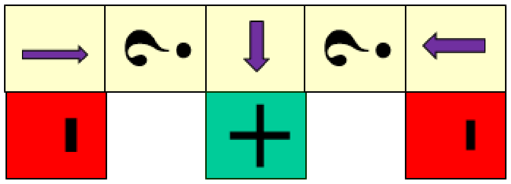


Figure 12.1: Grid-world example

12.1 Stochastic policies

We have seen that the optimal policy in an MDP is deterministic, so what benefit can be in considering a stochastic policy?

The main benefit is in situations where there is a miss-specification of the model. The main issue is that the state encoding might create a system which is not Markovian anymore, by coalescing certain states, which have identical encoding. We will give two example of this phenomena.

Aliased Grid-world

Consider the example in Figure 12.1. The green state is the good goal and the red ones are the bad. The encoding of each state is the location of the walls. In each state we need to choose a direction. The problem is that we have two states which are indistinguishable (marked by question mark).

It is not hard to see that any deterministic policy would fail from some start state (either the left or the right one). Alternatively, we can use a randomized policy in those states,with probability half go right and probability half go left. For such a policy we have a rather short time to reach the green goal state (and avoid the red states).

The issue here was that two different states had the same encoding, and thus violated the Markovian assumption.

Zero-sum games

The MDP model was not design for interactive zero-sum games, however, in many of the application we saw how to learn a policy to play a board game (such as backgammon). The main issue with a more general game, is the the optimal policy might be stochastic.

Consider a penny-matching game, which each player select a bit $\{0, 1\}$. If the two selected bits are identical the first player wins and if they differ the second player wins. The best policy for each player is stochastic (selecting each bit with probability half).

An important observation is that if one of the player play deterministically (or almost deterministically), then the other player can win (or almost always win). For this reason, any ϵ-greedy would have a poor performance.

Here we violated the assumption that the rewards depend only on the state. In this example they depend indirectly on the policy selected.

12.2 Policy optimization

We will assume that each state s has an encoding $\phi(s) \in \mathbb{R}^{d_1}$. The policy will have a parametrization $\theta \in \mathbb{R}^{d_2}$. The policy will be based on the two encodings, and we have $\pi(a|s,\theta)$, which is the probability of selecting action a when observing state s (or, equivalently, $\phi(s)$), and having a policy parametrization θ.

The optimization problem would be:

$$\theta^* = \arg\max_{\theta} J(\theta)$$

where $J(\theta)$ is the expected return of the policy $\pi(\cdot|\cdot,\theta)$.

This maximization problem can be solved in multiple ways. We will mainly look at gradient based methods.

We start be giving a few example on how to parameterize the policy. The first is a *log linear policy*. We will assume an encoding of the state and action pairs, i.e., $\phi(s,a)$. The linear part will compute $\mu(s,a) = \phi(s,a)^\top \theta$. Given the values of $\mu(s,a)$ for each $a \in A$, the policy select action a with probability proportional to $e^{\mu(s,a)}$. Namely,

$$\pi(a|s,\theta) = \frac{e^{\mu(s,a)}}{\sum_{b \in A} e^{\mu(s,b)}}$$

The second example is a *Gaussian policy*, where the action space is a real number, i.e., $A = \mathbb{R}$. The encoding is of states, and the actions are any real number. Given a state s we compute $\mu(s) = \phi(s)^\top \theta$. We select an action a from the normal distribution with mean $\mu(s)$ and variance σ^2, i.e., $N(\mu(s), \sigma^2)$. (The policy has an additional parameter σ.)

We would like to use the policy gradient to optimize the expected return of the policy. Let $J(\theta)$ be the expected return of the policy $\pi(\cdot|\cdot,\theta)$. We will compute the gradient of $J(\theta)$, i.e., $\nabla_\theta J(\theta)$. The update of the policy parameter is

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} J(\theta_t)$$

One challenge we will have to address is to relate $\nabla_\theta J(\theta)$ to the gradients $\nabla \pi(a|s,\theta)$.

12.2.1 Finite differences methods

This methods can be used even when we do not have a representation of the gradient of the policy. This can be done by introducing perturbations.

The simplest case is component-wise gradient estimates. Let e_i be a unit vector, i.e., has in the i-entry a value 1 and a value 0 in all the other entries. The perturbation that we will add is δe_i for some $\delta > 0$. We will use the following approximation:

$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{\hat{J}(\theta + \delta e_i) - \hat{J}(\theta)}{\delta}$$

where $\hat{J}(\theta)$ is unbiased estimator of $J(\theta)$. A more symmetric approximation is sometimes better,

$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{\hat{J}(\theta + \delta e_i) - \hat{J}(\theta - \delta e_i)}{2\delta}$$

The problem is that we need to average many samples of $\hat{J}(\theta \pm \delta e_i)$ to overcome the noise. Another weakness is that we need to do the computation per dimension. In addition, the selection of δ is also critical. A small δ might have a large noise rate that we need to overcome (by using many samples). A large δ run the risk of facing the non-linearity of J.

Rather then doing the computation per dimension, we can do a more global approach and use a least squares estimation of the gradient. Consider a random vector u_i, then we have

$$J(\theta + \delta u_i) \approx J(\theta) + \delta (u_i)^\top \nabla J(\theta)$$

We can define the following least square problem,

$$G = \arg\min_x \sum_i (J(\theta + \delta u_i) - J(\theta) - \delta (u_i)^\top x)^2,$$

where G is our estimate for $\nabla J(\theta)$.

We can move to matrix notation and define $\Delta J^{(i)} = J(\theta + \delta u_i) - J(\theta)$ and $\Delta J = [\cdots, \Delta J^{(i)}, \cdots]^\top$. We define $\Delta\theta^{(i)} = \delta u_i$, and the matrix $[\Delta\Theta] = [\cdots \Delta\theta^{(i)}, \cdots]^\top$, where the i-th row is $\Delta\theta^{(i)}$.

We would like to solve for the gradient, i.e,

$$\Delta J \approx [\Delta\Theta] x$$

This is a standard least square problem and the solution is

$$G = ([\Delta\Theta]^\top [\Delta\Theta])^{-1} [\Delta\Theta]^\top \Delta J$$

One issue that we neglected is that we actually do not a have the value of $J(\theta)$. The solution is to solve also for the value of $J(\theta)$.

We can define a matrix $M = [1, [\Delta\Theta]]$ vector of unknowns $x = [J(\theta), \nabla J(\theta)]$ and have the target be $z = [\cdots, J(\theta + \delta u_i), \cdots]$. We can solve for $z \approx Mx$.

12.3 Policy Gradient Theorem

The policy gradient theorem will relate the gradient of the expected return $\nabla J(\theta)$ and the gradients of the policy $\nabla \pi(a|s, \theta)$.

We will state the policy gradient theorem for the episodic return, but it also holds for the discounted return and average reward return. Recall that the episodic return is $\sum_{t=1}^{T} r_t$ and $V^\pi(s) = E[\sum_{t=1}^{T} r_t | s_1 = s]$.

Theorem 12.1 (Policy Gradient Theorem). *For any policy $\pi(\cdot|\cdot;\theta)$, we have*

$$\nabla J(\theta) \propto \sum_{s \in S} \mu(s) \sum_{a \in A} Q^\pi(s,a) \nabla_\theta \pi(a|s;\theta)$$

where $\mu(s) = \nu(s)/T$ and $\nu(s) = \sum_{t=1}^{T} \Pr[s_t = s | s_1, \theta]$.

Proof. For each state s we have

$$\begin{aligned}
\nabla V^\pi(s) &= \nabla \sum_a \pi(a|s) Q^\pi(s,a) \\
&= \sum_a \nabla \pi(a|s) Q^\pi(s,a) + \pi(a|s) \nabla Q^\pi(s,a) \\
&= \sum_a \nabla \pi(a|s) Q^\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla V^\pi(s') \\
&= \sum_{x \in S} \sum_{t=1}^{T} \Pr[s_t = x | s_1 = s, \pi] \sum_a \nabla \pi(a|x) Q^\pi(x,a)
\end{aligned}$$

where the first identity follows since by averaging $Q^\pi(s,a)$ over the actions a with the probabilities induce by $\pi(a|s)$ we have right expectation of the immediate reward. The next state is distributed correctly, and therefore the identity holds. The second equality follows from the gradient of a multiplication. The third follows since

$\nabla Q^\pi(s,a) = \nabla[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V^\pi(s'|s,a)]$. The last identity follows from unrolling s' to s'', etc. and then reorganizing the terms.

Using this we have

$$\begin{aligned}
\nabla J(\theta) &= \nabla V^\pi(s_1)\\
&= \sum_s \left(\sum_{t=1}^{T} \Pr[s_t = s|s_1,\pi]\right) \sum_a \nabla\pi(a|s)Q^\pi(s,a)\\
&= \sum_s \nu(s) \sum_a \nabla\pi(a|s)Q^\pi(s,a)\\
&= T\sum_s \mu(s) \sum_a \nabla\pi(a|s)Q^\pi(s,a)
\end{aligned}$$

□

The Policy Gradient Theorem gives us a way to compute the gradient. We can sample states from the distribution $\mu(s)$ using the policy π. We still need to resolve the sampling of the state.

We can use the following simple identity,

$$\nabla f(x) = f(x)\frac{\nabla f(x)}{f(x)} = f(x)\nabla \log f(x)$$

This implies that we can restate the Policy Gradient Theorem as,

$$\nabla J(\theta) \propto \sum_{s\in S}\mu(s)\sum_{a\in A}\pi(a|s;\theta)Q^\pi(s,a)\nabla_\theta \log\pi(a|s;\theta) = E^\pi[Q^\pi(s,a)\nabla_\theta \log\pi(a|s;\theta)]$$

We can now see what does the theorem says about some simple policy class. For the log-linear policy we have $\log\pi(a|s) \propto \phi(s,a)^\top\theta$, and then

$$\nabla \log\pi(a|s) = \phi(s,a) - \sum_b \pi(b|s;\theta)\phi(s,b)$$

and the update is $\Delta\theta \propto \alpha U\phi(s,a)$ where $E[U] = Q^\pi(s,a)$.

For Gaussian we have $\nabla\log\pi(a|s;\theta) \propto (a-\mu(s))\phi(s)/\sigma^2$ and update is $\Delta\theta \propto \alpha U(a-\mu(s))\phi(s)/\sigma^2$ where $E[U] = Q^\pi(s,a)$.

12.4 REINFORCE: Monte-Carlo updates

The REINFORCE algorithm uses a Monte-Carlo updates. Given a episode $(s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ for each $t \in [1, T]$ updates,

$$\theta \leftarrow \theta + \alpha R_{t:T} \nabla \log \pi(a_t | s_t; \theta)$$

where $R_{t:T} = \sum_{i=t}^{T} r_i$.

We can extend the REINFORCE to add a baseline function. The baseline function $b(s)$ can depend in an arbitrary way on the state, but does not depend on the action. The main observation would be that we can add or subtract any such function from our estimate U, and it will still be unbiased. This follows since

$$\sum_a b(s) \nabla \pi(a|s;\theta) = b(s) \nabla \sum_a \pi(a|s;\theta) = b(s) \nabla 1 = 0$$

Given this, we can restate the Policy Gradient Theorem as,

$$\nabla J(\theta) \propto \sum_{s \in S} \mu(s) \sum_{a \in A} (Q^\pi(s,a) - b(s)) \nabla_\theta \pi(a|s;\theta)$$

This gives us a degree of freedom to select $b(s)$. Note that by setting $b(s) = 0$ we get the old theorem. In many cases it is reasonable to use for $b(s)$ the value of the state, i.e., $b(s) = V^\pi(s)$. The motivation for this is to reduce the variance of the estimator. If we assume that the magnitude f gradients $\|\nabla \pi(a|s;\theta)\|$ is similar for all action $a \in A$, we are left with $E^\pi[(Q^\pi(s,a) - b(s))^2]$ which is minimized by $b(s) = E^\pi[Q^\pi(s,a)] = V^\pi(s)$.

We are left with the challenge of approximating $V^\pi(s)$. On the one hand this is part of the learning. On the other hand we have developed tools to address this in the previous lecture on value function approximation. We can use $V^\pi(s) \approx V(s;w) = b(s)$. The good news is that any $b(s)$ will keep the estimator unbiased, so we do not depend on $V(s;w)$ to be unbiased.

We can now describe the REINFORCE algorithm with baseline function. We will use a Monte-Carlo sampling to estimate $V^\pi(s)$ and this will define our function $b(s)$. We will update using $U - b(s)$ where $E[U] = Q^\pi(s,a)$. More specifically, our algorithm will have a class of value approximation function $V(\cdot;w)$ and a parameterized policy $\pi(\cdot|\cdot;\theta)$, and in addition to step size parameters, $\alpha, \beta > 0$. Given an episode $(s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$ for each $t \in [1, T]$ we compute the $R_{t:T}$ during the times $[t, T]$, i.e., $R_{t:T} = \sum_{i=t}^{T} r_i$. The error in time t is $\Gamma_t = R_{t:T} - V(s_t; w)$. The updates are

$$\Delta w = \alpha \Gamma_t \nabla V(s_t; w)$$

and

$$\Delta\theta = \beta\Gamma_t \nabla \log \pi(a_t|s_t;\theta)$$

We can extend this to handle also TD updates. We will use an actor-critic algorithm. We will use a Q-value updates for this (but can be done similarly with V-values).

The critic maintains an approximate Q function $Q(s, a; w)$. For each time t it defines the TD error to be $\Gamma_t = r_t + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)$. The update will be $\Delta w = \alpha\Gamma\nabla Q(s_t, a_t; w)$. The critic send the actor the TD error Γ.

The actor maintains a policy π which is parameterized by θ. Given a TD error Γ it updates $\Delta\theta = \beta\Gamma\nabla \log \pi(a_t|s_t;\theta)$. Then it selects $a_{t+1} \sim \pi(\cdot|s_{t+1};\theta)$.

We need to be careful in the way we select the function approximation $Q(\cdot; w)$ since it might introduce a bias. The following theorem gives a guarantee that we will not have such a bias.

A value function is *compatible* if,

$$\nabla_w Q(s, a; w) = \nabla_\theta \pi(a|s;\theta)$$

The expected square error of w is

$$SE(w) = E^\pi[(Q^\pi(s, a) - Q(s, a; w))^2]$$

Theorem 12.2. *Assume that Q is compatible and w minimizes $SE(w)$, then,*

$$\nabla_\theta J(\theta) = E^\pi[\nabla \log \pi(a|s;\theta) Q(s, a; w)]$$

Proof. Since w minimizes $SE(w)$ we have

$$\begin{aligned} 0 &= \nabla_w SE(w) \\ &= \nabla_w E^\pi[(Q^\pi(s, a) - Q(s, a; w))^2] \\ &= 2E^\pi[(Q^\pi(s, a) - Q(s, a; w))\nabla_w Q(s, a; w)] \end{aligned}$$

Since Q is compatible, we have $\nabla_w Q(s, a; w) = \nabla_\theta \pi(a|s;\theta)$ which implies,

$$0 = E^\pi[(Q^\pi(s, a) - Q(s, a; w))\nabla_\theta \log \pi(a|s;\theta)]$$

and have

$$E^\pi[Q^\pi(s, a)\nabla_\theta \log \pi(a|s;\theta)] = E^\pi[Q(s, a; w)\nabla_\theta \log \pi(a|s;\theta)]$$

This implies that by substituting this in the policy gradient theorem we have

$$\nabla_\theta J(\theta) = E^\pi[\nabla \log \pi(a|s;\theta) Q(s, a; w)]$$

□

We can summarize the various updates for the policy gradient as follows:

- REINFORCE (which is a Monte-Carlo estimate) uses $E^{\pi}[\nabla \log \pi(a|s;\theta) R_t]$.
- Q-function with actor-critic uses $E^{\pi}[\nabla \log \pi(a|s;\theta) Q(a_t|s_t;w)]$.
- A-function with actor-critic uses $E^{\pi}[\nabla \log \pi(a|s;\theta) A(a_t|s_t;w)]$m where $A(a|s;w) = Q(s,a;w) - V(s;w)$.
- TD with actor-critic uses $E^{\pi}[\nabla \log \pi(a|s;\theta)\Delta]$, where Δ is the TD error.

12.5 Application

12.5.1 RoboSoccer: training Aibo

Aibo is a small robot that was used in the Robo-Soccer competitions. One of the main tasks was to get the robot to walk fast and stable. For this the UT Austin team used reinforcement learning, and specifically policy gradient.

The robot is controlled through 12 parameters which include: (1) For the front and rear legs tree parameters: height, x-pos., y-pos. (2) For the locus: length/skew multiplier. (3) height of the body both front and rear. (4) Time (per foot) to go through locus, and (5) time (per foot) on ground or in the air.

The training was done in episodes. The Aibo was trained to walk between two landmarks. (See Figure 12.2.) The optimization used a Policy gradient improvement using Finite difference method. Note that since the actual policy is unknown, there are not that many alternatives.

Given a set of parameters θ, there was a perturbation introduced in the following way. The value of θ_i was modified to $\theta_i + z_i$ where z_i is selected at random from $\{+\epsilon_i, 0, -\epsilon_i\}$. The values of the ϵ_i we selected to be small compared to the value of θ_i. After the perturbation we have a new vector θ'. Many such vectors are sampled. For each vector the policy is evaluated (the return is the time to walk a certain distance, and we like to minimize the time).

After running many such polices, the update is done as follows. For each attribute $i \in [1,12]$, we split the policies to three subsets, according to the value of z_i, and compute the average return in each subset. If the best outcome is for $z_i = 0$ then we set $A_i = 0$. Otherwise, we set $A_i = avg_i(+\epsilon_i) - avg_i(-\epsilon_i)$. The parameter vector is set to

$$\theta \leftarrow \theta + \alpha \frac{A}{\|A\|}$$

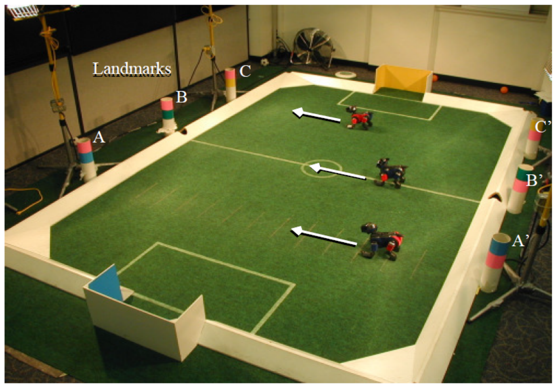


Figure 12.2: The Aibo training environment

(See also Figure 12.3.)

More figures and data are in the slides.

12.5.2 AlphaGo

AlphaGo and its successor AlphaGo Zero are the most advanced computer Go players, and the first to beat the best professional players. We will concentrate on AlphaGo. AlphaGo Zero has an additional benefit that it does not have any prior information about the game (mainly in the form of games between human experts).

The training has roughly three phases. The first phase learns from from human games (played by experts). Those games are used to derive a network SL (Supervised Learning) to predict next move of the human, and a Rollout which is a simple but very fast prediction (3 orders faster than SL).

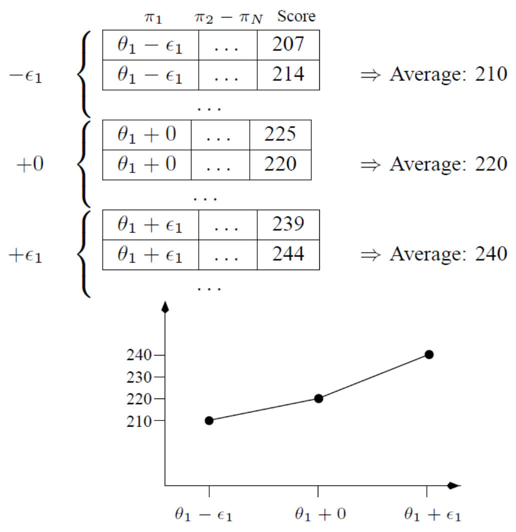
During phase two an RL network is trained. It is initialized with SL weights, and is optimized on self-play against itself.

In phase three, a value network is trained to predict the winner of the game (trained on the RL network self-play games). (See Figure 12.4.)

Phase 1

The SL network is trained to predict the human moves. It is a deep network (13 layers), and the goal is to output $p(a|s;\sigma)$ where σ are the parameters of the network. The parameters are update using a gradient step,

$$\Delta\sigma \propto \frac{\partial \log p(a|s;\sigma)}{\partial\sigma}$$



π_1
$\pi_2 - \pi_N$
Score
$-\epsilon_1$
$\theta_1 - \epsilon_1$ ... 207
$\theta_1 - \epsilon_1$ ... 214
$\Rightarrow$ Average: 210
$+0$
$\theta_1 + 0$ ... 225
$\theta_1 + 0$ ... 220
$\Rightarrow$ Average: 220
$+\epsilon_1$
$\theta_1 + \epsilon_1$ ... 239
$\theta_1 + \epsilon_1$ ... 244
$\Rightarrow$ Average: 240
240
230
220
210
$\theta_1 - \epsilon_1$
$\theta_1 + 0$
$\theta_1 + \epsilon_1$

Figure 12.3: The policy updates

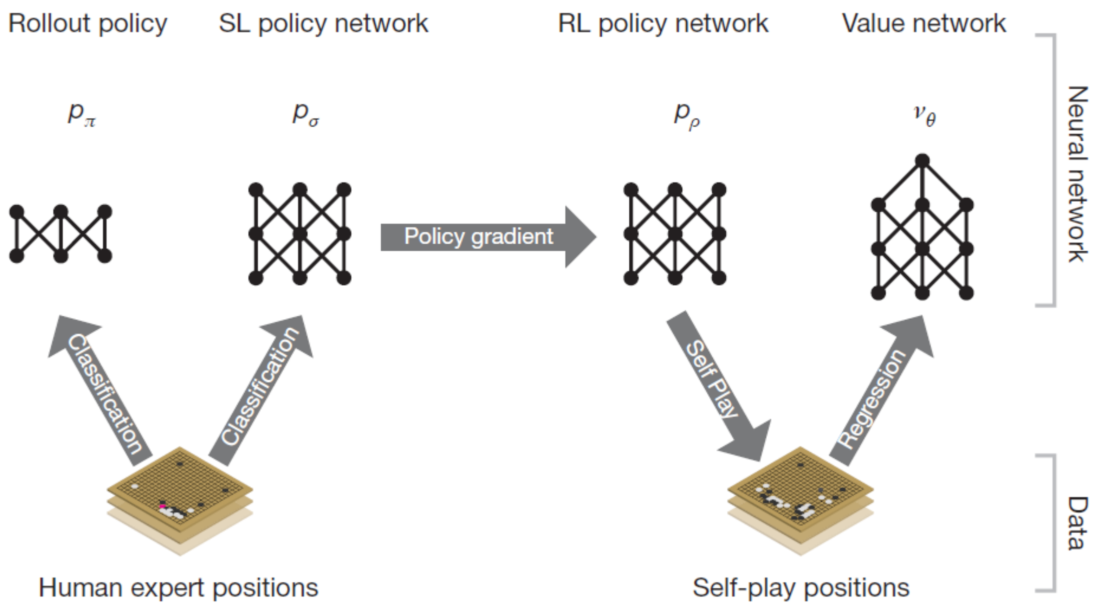


Figure 12.4: The AlphaGo training process

Overall, this network increased the prediction accuracy (compared to previous computer programs) from 44% to 55%. We should note that the increase in accuracy translated to a much more significant improvement in the performance.

In addition to SL, another very fast classifier Rollout was build. It uses a linear function approximation and predicts using a soft-max. The accuracy is only 24% but it is much faster ($2\mu s$ compared to $3ms$, a factor of 1500).

Phase 2

We learn an RL network that outputs $p(a|s;\rho)$ where ρ are the parameters of the network. The network structure is identical to that of SL, and the RL is initialized to the weights of SL, i.e., σ.

The RL is trained using self-play. Rather then playing against the most recent network, the opponent is selected at random between the recent RL configurations. This is done to avoid overfitting. The rewards of the game are given only at the end (win or lose).

The training is done using SGD with policy gradient

$$\Delta\rho \propto \frac{\partial \log p(a|s;\rho)}{\partial\rho}$$

The learned RL network outperformed SL (wins 80%) of the games. However, SL is better in predicting expert human behavior.

Phase 3

We learn a value function $V(s;\theta)$. The goal is to predict the probability that RL will win, and it is trained on the self-play data of RL. It uses a SGD update

$$\Delta\theta \propto \frac{\partial V(s;\theta)}{\partial\theta}$$

When the training was done on complete games, there was a serious problem of over-fitting. (The training error was 0.19 while the test error was 0.37). For this reason, the training is limited to only one position per game. The error rate is about 0.22.

The system uses a Monte-Carlo Search Trees (MCST) to evaluate the positions. The tree is used to perform a lookahead. It uses the Rollout policy to generate trajectory from the given position. The prediction is an average of the value network prediction $V(s;\theta)$ and the outcome of the Rollout policy. To see a run of the Monte-Carlo Search Tree see Figure 12.5.

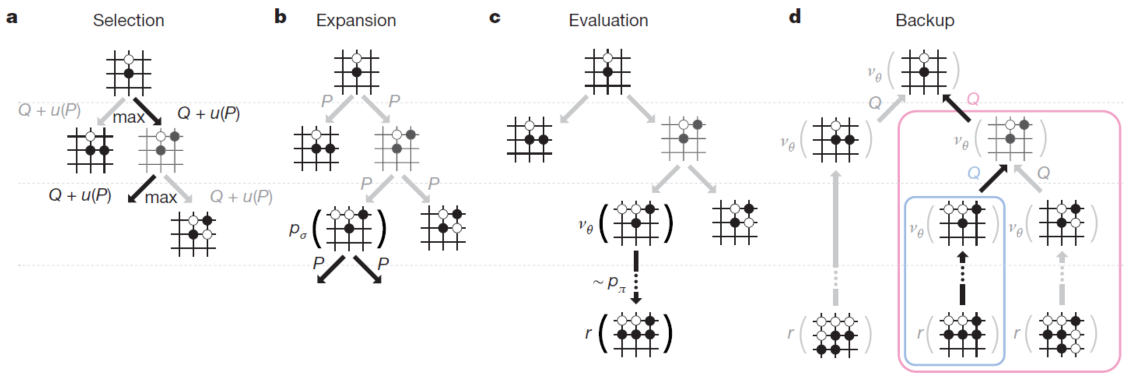


Figure 12.5: AlphaGo use of Monte-Carlo trees

12.6 Bibliography Remarks

The training of Aibo for RoboSoccer is by [**?**].

The work on AlphaGo is by [**?**].

Part of the outline borrows from David Silver class notes and the the book of Sutton and Barto [**?**].

Chapter 13

Large state space: tree based optimization

Chapter 14

Deep RL

Chapter 15

Multi-Arm bandits

15.1 Stochastic Bandits and Regret Minimization

We consider a simplified model of an MDP where there is only a single state and a fixed set A of k actions (a.k.a., arms). We consider a finite horizon problem, where the horizon is T. Clearly, the planning problem is trivial, simply select the action with the highest expected reward. We will concentrate on the learning perspective, where the expected reward of each action are unknown.

At each round $1 \leq t \leq T$ the player selects and executes an action. After executing the action, the player observes the reward of the action. However, the rewards of the other actions in A are not revealed to the player.

The reward for action i at round t is denoted by $r_t(i) \sim D_i$, where the support of the reward distribution D_i is $[0, 1]$. We assume that the rewards are i.i.d. (independent and identically distributed).

Motivation

1. **News**: a user visits a news site and is presented with a news header. The user either clicks on this header or not. The goal of the website is to maximize the number of clicks. So each possible header is an action in a bandit problem, and the clicks are the rewards

2. **Medical Trials**: Each patient in the trial is prescribed one treatment out of several possible treatments. Each treatment is an action, and the reward for each patient is the effectiveness of the prescribed treatment.

3. **Ad selection**: In website advertising, a user visits a webpage, and a learning algorithm selects one of many possible ads to display. If an advertisement is

displayed, the website observes whether the user clicks on the ad, in which case the advertiser pays some amount $v_a \in [0,1]$. So each advertisement is an action, and the paid amount is the reward.

Model

- A set of actions $A = \{a_1 \ldots, a_k\}$
- Each action a_i has a reward distribution D_i over $[0,1]$.
- The expectation of distribution D_i is:

$$\mu_i = E_{X \sim D_i}[X]$$

- $\mu^* = \max_i \mu_i$ and $a^* = \arg\max_i \mu_i$.
- a_t is the action the learner chose at round t

$$Regret = \max_{i \in A} \sum_{t=1}^{T} \underbrace{r_t(i)}_{\text{Random variable}} - \sum_{t=1}^{T} \underbrace{r_t(a_t)}_{\text{Random variable}}$$

$$\begin{aligned} \text{Pseudo Regret} &= \max_i E\left[\sum_{t=1}^{T} r_t(i)\right] - E\left[\sum_{t=1}^{T} r_t(a_t)\right] \\ &= \mu^* \cdot T - \sum_{t=1}^{T} \mu_{a_t} \end{aligned}$$

We will use extensively the following concentration bound.

Theorem 15.1 (Hoeffding's inequality)**.** *Given $X_1, \ldots, X_m$ i.i.d random variables s.t $X_i \in [0,1]$ and $E[X_i] = \mu$.*

$$Pr[\underbrace{\frac{1}{m}\sum_{i=1}^{m} X_i - \mu}_{\frac{1}{m}S} \geq \epsilon] \leq \exp(-\tfrac{\epsilon^2 m}{2})$$

15.1.1 Warmup: Full information $k = 2$

We start with a simple case where there are two actions and we observe the reward of both actions at each time t. We will analyze the greedy policy, which selects the action with the higher average reward (so far).

The greedy policy at time t does the following:

- We observe $\langle r_t(1), r_t(2) \rangle$
- Define
$$avg_t(i) = \frac{1}{t} \sum_{\tau=1}^{t} r_\tau(i)$$
- In time $t+1$ we choose:
$$a_{t+1} = \arg\max_{i \in \{1,2\}} avg_t(i)$$

We now would like to compute the expected regret of the greedy policy. W.l.o.g., we assume that $\mu_1 \geq \mu_2$, and define $\Delta = \mu_1 - \mu_2 \geq 0$.

$$\text{Pseudo Regret} = \sum_{t=1}^{\infty} (\mu_1 - \mu_2) \Pr\left[avg_t(2) \geq avg_t(1)\right]$$

Clearly, at any time t,

$$E[avg_t(2) - avg_t(1)] = \mu_2 - \mu_1 = -\Delta$$

We can define a random variable $X_t = r_t(2) - r_t(1) + \Delta$ and $E[X_t] = 0$. Since $(1/t)\sum_t X_t = avg_t(2) - avg_t(1) + \Delta$, by Theorem 15.1

$$Pr[S_2(t) \geq S_1(t)] = Pr\left[avg_t(2) - avg_t(1) + \Delta \geq \Delta\right] \leq e^{-\Delta^2 \frac{t}{2}}$$

We can now bound the regret as follows,

$$
\begin{aligned}
E\left[\text{Pseudo Regret}\right] &= \sum_{t=1}^{\infty} \Delta \Pr\left[S_2(t) \geq S_1(t)\right] \\
&\leq \sum_{t=1}^{\infty} \Delta e^{-\Delta^2 \frac{t}{2}} \\
&\leq \int_0^{\infty} \Delta e^{-\Delta^2 \frac{t}{2}} dt \\
&= \left[\frac{2}{\Delta} e^{-\Delta^2 \frac{t}{2}}\right]_0^{\infty} \\
&= \frac{2}{\Delta}
\end{aligned}
$$

Notice that this bound does not depend on T!

15.1.2 Stochastic Multi-Arm Bandits

We will now see that we cannot get a regret that does not depend on T for the bandits case. Considering the following example:

$$a_1 \sim Br\left(\frac{1}{2}\right)$$

For action a_2 there are two alternatives, each with probability $1/2$,

$$a_2 \sim Br\left(\frac{1}{4}\right)\left(w.p.\frac{1}{2}\right) \qquad or \qquad a_2 \sim Br\left(\frac{3}{4}\right)\left(w.p.\frac{1}{2}\right)$$

Assume by way of contradiction

$$E\left[\sum_{i\in\{1,2\}} \Delta_i |T_i|\right] = E\left[PseudoRegret\right] = R$$

where R does not depend on T.
By Markov inequality:

$$Pr\left[PseudoRegret \geq 2R\right] \leq \frac{1}{2}$$

Since μ_1 is known, an optimal algorithm will first check a_2 in order to decide which action is better and stick with it.

Assuming $\mu_2 = \frac{1}{4}$, and the algorithm decided to stop playing a_2 after M rounds, Then:

$$PseudoRegret = \frac{1}{4}M$$

Thus,

$$Pr\left[PseudoRegret \geq 2R\right] = Pr\left[M \geq 8R\right] \leq \frac{1}{2}$$

And,

$$Pr\left[M < 8R\right] > \frac{1}{2}$$

Hence, the probability that after $8R$ rounds, the algorithm will stop playing a_2 (if $\mu_2 = \frac{1}{4}$) is at least $\frac{1}{2}$. This implies that there is some sequence of $8R$ outcomes which will result is stopping to try action a_2. For simplicity, assume that the sequence is the all zero sequence.

Assume $\mu_2 = \frac{3}{4}$, but all $8R$ first rounds, playing a_2 yield the value zero (with probability $\left(\frac{1}{4}\right)^{8R}$). We assumed that after $8R$ zeros for action a_2 the algorithm will stop playing a_2, even though it is the preferred action. In this case, we will get:

$$PseudoRegret = \frac{1}{4}(T - M) \approx \frac{1}{4}T$$

The expected Pseudo Regret is,

$$E\left[PseudoRegret\right] = R \geq \underbrace{\frac{1}{2}}_{a_2 \sim Pr\left(Br\left(\frac{3}{4}\right)\right)} \cdot \underbrace{\left(\frac{1}{4}\right)^{8R}}_{Pr\left(all\ 0 | a_2 \sim Pr\left(Br\left(\frac{3}{4}\right)\right)\right)} \cdot (T - 8R) \approx e^{-O(R)}T$$

Which implies that:

$$R = O\left(\log T\right)$$

Contrary to the assumption that R does not depend on T.

15.2 Explore-Then-Exploit

1. We choose a parameter M. For M phases we choose each action once (for a total of kM rounds of exploration).

2. After kMrounds we always choose the action that had highest average reward during the explore phase.

Define:

$$T_j = \{t : a_t = j, t \leq k \cdot M\}$$
$$\hat{\mu}_j = \frac{1}{M} \sum_{t \in T_j} r_j(t)$$
$$\mu_j = E[r_j(t)]$$
$$\Delta_j = \mu^* - \mu_j$$

where Δ_j is the difference in expected reward of action j and the optimal action.

We can now write the regret as a function of those parameters:

$$E\left[\text{Pseudo regret}\right] = \underbrace{\sum_{j=1}^{k} \Delta_j \cdot M}_{Explore} + \underbrace{(T - k \cdot M) \sum_{j=1}^{k} \Delta_j Pr\left[j = \arg\max_i \hat{\mu}_i\right]}_{Exploit}$$

For the analysis define:

$$\lambda = \sqrt{\frac{8 \log T}{M}}$$

By Theorem 15.1 we have

$$\Pr\left[|\hat{\mu}_j - \mu_j| \geq \lambda\right] \leq 2e^{-\frac{\lambda M}{2}} = \frac{2}{T^4}$$

which implies (using the union bound) that

$$\Pr \underbrace{\left[\exists_j : |\hat{\mu}_j - \mu_j| \geq \lambda\right]}_{B} \leq \frac{2k}{T^4} \underset{for\, k \leq T}{\leq} \frac{2}{T^3}$$

Define the "bad event" $B = \{\exists_j : |\hat{\mu}_j - \mu_j| \geq \lambda\}$. If B did not happen then for each action j, such that $\hat{\mu}_j \geq \hat{\mu}^*$, we have

$$\mu_j + \lambda \geq \hat{\mu}_j \geq \hat{\mu}^* \geq \mu^* - \lambda$$

therefore:

$$2\lambda \geq \mu^* - \mu_j = \Delta_j$$

and therefore:

$$\Delta_j \leq 2\lambda$$

Then, we can bound the expected regret as follows:

$$E[Regret] \leq \underbrace{\left(\sum_{j=1}^{k} \Delta_j\right) M}_{Explore} + \underbrace{(T - k \cdot M) \cdot 2\lambda}_{\text{B didn't happen}} + \underbrace{\frac{2}{T^3} \cdot T}_{\text{B happened}}$$

$$\leq k \cdot M + 2 \cdot \sqrt{\frac{8 \log T}{M}} \cdot T + \frac{2}{T^2}$$

If we optimize the number of exploration phases M and choose $M = T^{\frac{2}{3}}$, we get:

$$k \cdot T^{\frac{2}{3}} + 2 \cdot \sqrt{8 \log T} \cdot T^{\frac{2}{3}} + \frac{2}{T^2}$$

which is sub-linear but more than the $O(\sqrt{T})$ rate we would expect.

15.3 Improved Regret Minimization Algorithms

We will look at some more advanced algorithms that mix the exploration and exploitation.

Define:

$n_t(i)$ - the number of times we chose action i by round t

$\hat{\mu}_t(i)$ - the average reward of action i so far, that is:

$$\hat{\mu}_t(i) = \sum_{t=1}^{T} r_i(t) \mathbb{I}\left(a_t = i\right) \frac{1}{n_i(t)}$$

Notice that $n_i(t)$ is a random variable and not a number!

We would like to get the following result:

$$\Pr\left[|\hat{\mu}_t(i) - \mu_i| \leq \underbrace{\sqrt{\frac{8 \log T}{n_i(t)}}}_{\lambda_t(i)}\right] \geq 1 - \frac{2}{T^4}$$

We would like to look at the m^{th} time we sampled action i:

$$\hat{\mathbb{V}}_m(i) = \frac{1}{m} \sum_{\tau=1}^{m} r_i(t_\tau)$$

Where the t_τ's are the rounds when we chose action i

Now we fix m and get:

$$\forall i \forall m \quad \Pr\left[\left|\hat{\mathbb{V}}_m(i) - \mu_i\right| \leq \sqrt{\frac{8\log T}{m}}\right] \geq 1 - \frac{2}{T^4}$$

and notice that $\hat{\mu}_t(i) \equiv \hat{\mathbb{V}}_i(m)$ when $m = n_i(t)$.

Define the "good event" G:

$$G = \{\forall_i \forall_t \, |\hat{\mu}_i(t) - \mu_i| \leq \lambda_i(t)\}$$

The probability of G is,

$$Pr\,(G) \geq 1 - \frac{2}{T^2}$$

15.4 Refine Confidence Bound

Define the upper confidence bound:

$$UCB_t(i) = \hat{\mu}_t(i) + \lambda_t(i)$$

and similarly, the lower confidence bound:

$$LCB_t(i) = \hat{\mu}_t(i) - \lambda_t(i)$$

if G happened then:

$$\forall i \forall t \quad \mu_i \in [LCB_t(i), UCB_t(i)]$$

Therefore:

$$Pr\left[\forall i \forall t \quad \mu_i \in [LCB_t(i), UCB_t(i)]\right] \geq 1 - \frac{2}{T^2}$$

15.4.1 Successive Elimination

We maintain a set of actions S.

Initially $S = A$

In each phase:

- We try every $i \in S$ once

- For each $j \in S$ if there exists $i \in S$ such that:

$$UCB_t(j) < LCB_t(i)$$

  We remove j from S, that is we update:

$$S \leftarrow S - \{j\}$$

We will get the following results:

- As long as action i is still in S, we have tried action i exactly the same number of times as all of any other action $j \in S$.
- The best action, under the assumption that the event G holds, is never eliminated from S.

Under the assumption of G we get:

$$\mu^* - 2\lambda \leq \hat{\mu}^* - \lambda = LCB_* < UCB_i = \hat{\mu_i} + \lambda \leq \mu_i + 2\lambda$$

Where $\lambda = \lambda_i = \lambda^*$ because we have chosen action i and the best action the same number of times so far.

Therefore, assuming event G holds,

$$\Delta_i = \mu^* - \mu_i \leq 4\lambda = 4\sqrt{\frac{8\log T}{n_t(i)}}$$

$$\Rightarrow \quad n_T(i) \leq \frac{c}{\Delta_i^2}\log T$$

This implies that

$$\begin{aligned} E\left[\text{Pseudo Regret}\right] &= \sum_{i=1}^{k} \Delta_i n_i(t) \\ &\leq \sum_{i=1}^{k} \frac{c}{\Delta_i}\log T + \underbrace{\frac{2}{T^2}\cdot T}_{\text{The bad event}} \end{aligned}$$

meaning that the expected pseudo regret is bounded by $O\left(\frac{1}{T}\right)$.

15.4.2 Upper confidence bound (UCB)

The UCB algorithm simply uses the UCB bound. The algorithm works as follows:

- We try each action once (for a total of k rounds)
- Afterwards we choose:

$$a_t = \arg\max_i UCB_t(i)$$

If we chose action i then, assuming G holds, we have

$$UCB_t(i) \geq UCB_t(a^*) \geq \mu^*$$

where a^* is the optimal action.

Using the definition of UCB and the assumption that G holds, we have

$$UCB_t(i) = \hat{\mu}_t(i) + \lambda_t(i) \leq \mu_i + 2\lambda_t(i)$$

Since we selected action i at time t we have

$$\mu_i + 2\lambda_t(i) \geq \mu^*$$

Rearranging, we have,

$$2\lambda_t(i) \geq \mu^* - \mu_i = \Delta_i$$

Each time we chosen action i, we could not have made a very big mistake because:

$$\Delta_i \leq \ 2 \cdot \sqrt{\frac{8 \log T}{n_t(i)}}$$

And therefore if i is very far off from the optimal action we would not choose it too many times. We can bound the number of time action i is used by,

$$n_t(i) \leq \frac{c}{\Delta_i^2} \log T$$

And over all we get:

$$\begin{aligned} E\left[\text{Pseudo Regret}\right] &= \sum_{i=1}^{k} \Delta_i E\left[n_t(i)\right] + \underbrace{\frac{2}{T^2} \cdot T}_{\text{The bad event}} \\ &\leq \sum_{i=1}^{k} \frac{c}{\Delta_i} \cdot \log T + \frac{2}{T} \end{aligned}$$

15.5 Best Arm Identification

We would like to identify the best action, or an almost best action. We can define the goal in one of two ways.

PAC criteria An action i is ϵ-optimal if V. The PAC ctriteria is that, given $\epsilon, \delta > 0$,, with probability at least $1 - \delta$, find an ϵ optimal action.

Exact identification Given $\Delta \leq \mu^* - \mu_i$ (for every suboptimal action i), find the optimal action a_*, with probability at least $1 - \delta$.

15.5.1 Naive Algorithm (PAC criteria):

We sample each action i for $m = \frac{8}{\epsilon^2} \log \frac{2k}{\delta}$ times, and return $a = \arg\max_i \hat{\mu}_i$.

For rewards in $[0, 1]$, then, by Theorem 15.1, for every action i we have

$$Pr\left[\underbrace{|\hat{\mu}_i - \mu_i| > \frac{\epsilon}{2}}_{\text{bad event}}\right] \leq 2e^{-\left(\frac{\epsilon}{2}\right)^2 m/2} = \frac{\delta}{k}$$

By union bound we get:

$$Pr\left[\exists_i \, |\hat{\mu}_i - \mu_i| > \frac{\epsilon}{2}\right] \leq \delta$$

If the bad event $B = \{\exists_i \, |\hat{\mu}_i - \mu_i| > \frac{\epsilon}{2}\}$ did not happen, then both: (1) $\mu^* - \frac{\epsilon}{2} \leq \hat{\mu}^*$ and (2) $\mu_i + \frac{\epsilon}{2} \leq \hat{\mu}_i$.

This implies,

$$\Rightarrow \; \mu_i + \frac{\epsilon}{2} \geq \hat{\mu}_i \geq \hat{\mu}^* \geq \mu^* - \frac{\epsilon}{2}$$

$$\Rightarrow \; \epsilon \geq \mu^* - \mu_i$$

And therefore $a = \arg\max_i \hat{\mu}_i$ is the optimal action in probability $1 - \delta$

We would like to slightly improve the sample size of this algorithm

15.5.2 Median Algorithm

The idea: the algorithm runs for l phases, after each phase we eliminate half of the actions. This elimination allows us to sample each action more times in the next phase which makes eliminating the optimal action less likely.

```
**Input**: $\epsilon, \delta > 0$
**Output**: $\bar{a} \in A$
**Init**: $S_1 = A$, $\epsilon_1 = \frac{\epsilon}{4}$, $\delta_1 = \frac{\delta}{2}$, $l = 1$
**Repeat**:
    $\forall_i \in S_l$, sample action i, $\frac{1}{\left(\frac{\epsilon_l}{2}\right)^2} \log\left(\frac{3}{\delta_l}\right)$ times
    $\hat{\mu}_i \leftarrow$ mean (only of samples during the $l^{th}$phase)
    median$_l \leftarrow median\{\hat{\mu}_i : i \in S_l\}$
    $S_{l+1} \leftarrow \{i \in S_l : \hat{\mu}_i \geq \text{median}_l\}$
    $\epsilon_{l+1} \leftarrow \frac{3}{4}\epsilon_l$
    $\delta_{l+1} \leftarrow \frac{\delta_l}{2}$
    $l \leftarrow l + 1$
**Until** $|S_l| = 1$
```

Algorithm 1: Best Arm Identification

Complexity: During phase l we have $|S_l| = \frac{k}{2^{l-1}}$ actions.

$$\epsilon_l = \frac{3}{4}\epsilon_{l-1} = \frac{\epsilon}{4}\left(\frac{3}{4}\right)^{l-1}, \ \delta_l = \frac{\delta}{2^l}$$

$$\Rightarrow \sum \epsilon_l \leq \epsilon, \ \sum \delta_l \leq \delta$$

The total number of samples is therefore:

$$ccc \sum_l |S_l| \cdot \frac{4}{\epsilon_l^2} \log \frac{3}{\delta_l} = \sum_l \frac{k}{2^{l-1}} \frac{64}{\epsilon^2} \left(\frac{16}{9}\right)^{l-1} \log \frac{3 \cdot 2^l}{\delta}$$

$$= \sum_l k \left(\frac{8}{9}\right)^{l-1} \left[c \cdot \frac{\log \frac{1}{\delta}}{\epsilon^2} + \frac{\log 3}{\epsilon^2} + \frac{l}{\epsilon^2}\right]$$

$$= O\left(\frac{k}{\epsilon^2} \log \frac{1}{\delta}\right)$$

Correctness:

Theorem 15.2. $Pr\left[\underbrace{\max_{j \in S_l} \mu_j}_{action\ l} \leq \underbrace{\max_{j \in S_{l+2}} \mu_j}_{action\ l+1} + \epsilon_l\right] \geq 1 - \delta_l$

Proof. We do the proof for $l = 1$, general l is similar. Define $E_1 = \left\{\hat{\mu}^* < \mu^* - \frac{\epsilon_1}{2}\right\}$. We have $Pr\left[E_1\right] \leq \frac{\delta_1}{3}$. If E_1 did not happen, we define a bad set:

$$\text{Bad} = \{j \ : \ \mu^* - \mu_j \geq \epsilon_1, \ \hat{\mu}_j \geq \hat{\mu}^*\}$$

Consider an action j such that $\mu^* - \mu_j \geq \epsilon_1$, then:

$$Pr[\hat{\mu}_j \geq \hat{\mu}^* | \underbrace{\hat{\mu}^* \geq \mu^* - \frac{\epsilon_1}{2}}_{\neg E_1}] \leq Pr[\hat{\mu}_j \geq \mu_j + \frac{\epsilon_1}{2} | \neg E_1] \leq \frac{\delta_1}{3}$$

Note that the probability is not negligible. We will show that it cannot happen to too many such actions. We will bound the expectation of the size of *Bad*,

$$E[|\text{Bad}| | \neg E_1] \leq k\frac{\delta_1}{3}$$

with Markov's inequality we get:

$$Pr\left[|\text{Bad}| \geq \frac{k}{2} \middle| \neg E_1\right] \leq \frac{E\left|\text{Bad}\right|}{k/2} = \frac{2}{3}\delta_1$$

with probability $1 - \delta_1$: $\hat{\mu}^* \geq \mu^* - \frac{\epsilon_1}{2}$ and $|\text{Bad}| \leq \frac{k}{2}$. Therefore: $\exists_j \notin \text{Bad}$ and $j \in S_{l+1}$. $\square$

Chapter 16

POMDP

The lecture today is about Partially Observable MDP (POMDP). The main difference is that we will not observe the current state but only a signal that depends on it.

Example:
Assume we have four states, in linear order: $1, 2, 3, 4$. We have an action UP that with probability 0.90 moves from i to $i - 1$ and with probability 0.1 moves to $i + 1$. Similarly, we have an action DOWN that with probability 0.90 moves from i to $i + 1$ and with probability 0.1 moves to $i - 1$. (In state $i = 1$ if we need to move to $i - 1$ we stay in $i = 1$ and in state $i = 4$ if we need to move to $i + 1$ we stays in state $i = 4$.)

One of the states, state 2 has a reward, while the other states do not have any reward. (See Figure 16.1.) The only observation we see is the reward.

Initially, we know we are in a random state without a reward. This implies that the distribution of where we are is $(1/3, 0, 1/3, 1/3)$. Assume we did an action UP and did not observe a reward. We now can compute the posterior of our probability distribution.

First, we compute the state probability distribution, assuming any current state.

1. Assuming we are in state $i = 1$, our state distribution will be $(0.9, 0.1, 0, 0)$.

2. Assuming we are in state $i = 2$, our state distribution will be $(0.9, 0, 0.1, 0)$.

3. Assuming we are in state $i = 3$, our state distribution will be $(0, 0.9, 0, 0.1)$.

4. Assuming we are in state $i = 4$, our state distribution will be $(0, 0, 0.9, 0.1)$.

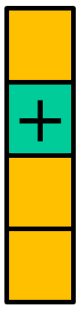


Figure 16.1: The four state POMDP

Since our prior state distribution was $(1/3, 0/1/3, 1/3)$ our posterior state distribution (before observing the reward) is $(0.3, 1/3, 0.3, 1/15)$.

Given that we observed "no reward" we are not instate 2 so the posterior state distribution is $(0.45, 0, 0.45, 0.1)$.

16.1 POMDP: model

The model of a POMDP will be similar to that of an MDP with an additional observation function. Namely,

- S, a finite set of states.
- A, a finite set of actions.
- $p(s'|s, a)$, a probability distribution of next state given that we are in state s and do action a.
- $R(s, a)$ a random variable for the reward in state s doing action a, and $r(s, a) = E[R(s, a)]$.
- q_0 the initial state (can also be a distribution over states)
- Ob is an observability distribution over O a set of observable signals. O can be either finite of infinite. $Ob(o|s', a)$ is the probability that we observe signal $o \in O$ if we reach state s' after doing action a. (Note that in an MDP we have the observable signals $O = S$ and $Ob(o|s', a) = s'$).

16.1.1 Belief state

The belief state tracks the state distribution, which captures our current state. Each time we perform an action and observe an observation we compute a new belief state, which is our posterior distribution, given our action and observation.

The belief state is a sufficient statistics, implying that it has all the information needed from the history. (Also, two histories which reach the same belief state, we should behave identical for them.) Formally, a belief state b is a distribution over states, i.e., $b \in [0,1]^{|S|}$ and $\sum_s b(s) = 1$.

Given a belief state b and an action a and observation o, we need to compute the next belief state $b'(s) = \Pr[s'|o,a,b]$. This will be a deterministic function, and we will have $b' = T(b,a,o)$.

Belief state computation:
We compute the next belief state as follows:

$$\begin{aligned} b'(s') &= \Pr[s'|o,a,b] \\ &= \frac{\Pr[o|s',a,b]\Pr[s'|a,b]}{\Pr[o|a,b]} \\ &= \frac{\Pr[o|s',a]\sum_s \Pr[s'|a,b,s]\Pr[s|a,b]}{\Pr[o|a,b]} \\ &= \frac{Ob(o|s',a)\sum_s p(s'|a,s)b(s)}{\Pr[o|a,b]} \end{aligned}$$

where the first identity is the definition of the new belief state. The second identity is Bayes rule. The third identity is adding a conditioning over s. The last identity is translating the probabilities to the POMDP parameters.

From POMDP to infinite MDP
Consider the following MDP. The set of states are B all the possible belief states. (Note that B is infinite!) The set of actions is A (finite). For each belief state b and action a we define the expected reward $r(b,a) = \sum_s b(s)r(s,a)$. The transition probability is to move to $b' = T(b,a,o)$ and is deterministic. (Namely, $\Pr[b' = T(b,a,o)|b,a,o] = 1$ and $\Pr[b' \neq T(b,a,o)|b,a,o] = 0$.)

16.2 Value function

The value function will be mapping belief states to expected return. The return can be any of the return we discussed: finite horizon, discounted, of average reward.

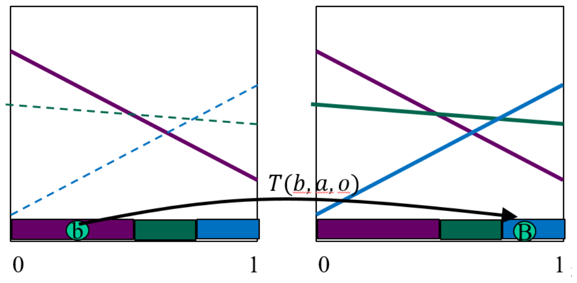


Figure 16.2: Computing $V_2^*(b|a_1, o_1)$

We would like to characterize the optimal value function. For a finite horizon it will be piece-wise linear and convex. (For discounted it will be convex.) Our algorithmic challenge is to handle the infinite state spaces. We will concentrate on the case of finite horizon.

For simplicity, we start with a horizon of length $H = 1$. Namely, we would like to maximize our immediate reward. For each action, our immediate reward is a linear function in the belief state. Namely, $r(b, a) = \sum_s b(s)r(s, a)$. The optimal action for $H = 1$ is the action a that maximizes the expected reward. Namely, $V_1^*(b) = \max_a r(b, a) = \max_a[\sum_s b(s)r(s, a)]$. This implies that the optimal value function is a maximum of linear functions. This gives us a finite representations. It also partitions the space of belief states to regions, which in each region there is an optimal action. The partition to the regions is done by intersecting hyper-planes. The region in which a_1 is the best action is define by $\{b : \forall a \neq a_1 \sum_s b(s)r(s, a_1) \geq \sum_s b(s)r(s, a)\}$.

We will now consider a horizon of $H = 2$. We will partition the task to three steps. In the first step, we will assume that we are in belief state b perform action a_1 and observe o_1. In the second step, we assume we are in belief state b and perform action a_1. In the third step, we only assume we are in belief state b.

Assume we we are in belief state b perform action a_1 and observe o_1. We would like to compute $V_H(b|a_1, o_1)$ for $H = 2$. For the first step, given that we do action a_1, we can compute the expected immediate reward $\sum_s b(s)r(s, a_1)$. Now, we move from belief state b to belief state $b' = T(b, a_1, o_1)$. When we reach b' we have only one step remaining, so we perform the best action and get $V_1^*(b')$. This implies that $V_2^*(b|a_1, o_1) = r(b, a_1) + V_1^*(b')$. (See Figure 16.2.) This implies that $V_2^*(b|a_1, o_1)$ is piece-wise linear.

We can now do the same for each possible observations (assume the set of observations is finite, for simplicity). This implies that we have the values of $V_2^*(b|a_1, o)$ for any $o \in O$. This allows us to compute $V_2^*(b|a_1)$, by simply averaging over the observations. $V_2^*(b|a_1) = \sum_o V_2^*(b|a_1, o)\Pr[o|b, a_1]$. We can compute $\Pr[o|b, a] = \sum_{s,s'} b(s)p(s'|s, a)Ob(o|s', a)$.

Given that we can compute $V_2^*(b|a)$ for any action $a \in A$, we can deduce $V_2^*(b) = \max_a V_2^*(b|a)$.

We can now compute for a longer horizon, say $H = 3$, using dynamic programming. We want to compute $V_3^*(b)$. We first compute the function $V_2^*(\cdot)$. For each action a_i and observation o_j we compute $V_3^*(b|a_i, o_j)$. This is done by first computing $r(b, a_i)$ and $b' = T(b, a_i, o_j)$. Then we compute $V_3^*(b|a_i, o_j) = r(b, a_i) + V_2^*(b')$.

Optimal value function

For the finite horizon the optimal value function $V_H^*(b)$ is define as follows,

$$V_H^*(b) = \max_a [r(b, a) + \sum_o \Pr[o|b, a] V_{H-1}^*(T(b, a, o))]$$

and the optimal policy is

$$\pi_H^*(b) = \arg\max_a [r(b, a) + \sum_o \Pr[o|b, a] V_{H-1}^*(T(b, a, o))]$$

Theorem 16.1. *For the finite horizon, the function V_H^* is piece-wise linear and convex. In addition, there exists a collection of $\Theta = \{\theta_i \in \mathbb{R}^{|S|}\}$ such that $V_H^*(b) = \max_{\theta_i \in \Theta} \sum_s b(s)\theta_i(s)$.*

For the discounted setting we have

$$V^*(b) = [r(b, a) + \gamma \sum_o \Pr[o|b, a] V_{H-1}^*(T(b, a, o))]$$

Theorem 16.2. *For the discounted return, the function V^* is convex.*

Policy Tree

How does the optimal finite horizon policy can be implemented. We can create a tree. In each node of the tree, we can label it by the action we perform. We label the outgoing edges by the observation we receive. We start at the root. The action at the root is the action we perform at the initial state. Given the observation we receive, we continue to the next node in the tree. The label of that node is the action we perform and so on. The depth of the tree is H for a finite horizon of length H.

16.2.1 Value Iteration:

We can run a value iteration in the POMDP (actually, on the belief states. We can write the update as follows:

$$\begin{aligned} V_t^{a,o}(b) =& \frac{r(b,a)}{|O|} + \gamma \Pr[o|b,a] V_t(T(b,a,o)) \\ V_t^a(b) =& \sum_o V_t^{a,o}(b) \\ V_{t+1}^*(b) =& \max_a V_t^a(b) \\ V_{t+1}^*(b) =& \max_a [r(b,a) + \gamma \sum_o V_t^*(T(b,a,o) \end{aligned}$$

Assume that V_t is piece-wise linear. Namely, there exists a set Θ_t such that

$$V_t(b) = \max_{\theta \in \Theta_t} \theta^\top b$$

For $V_t^{a,o}$ we have that $b' = T(b,a,o)$ is linear in b, i.e., there is a matrix M such that $b' = Mb$. (Note that M depends on a and o.) The value of $V_t^{a,o}$ is

$$V_t^{a,o}(b') = \max_{\theta \in \Theta_t} \theta^\top b' = \max_{\theta \in \Theta_t} \theta^\top M b$$

This implies that we can define $\Theta_t^{a,o} = \{\theta^\top M : \theta \in \Theta_t\}$.

For V_t^a we have that

$$V_t^a(b) = \sum_o V_t^{a,o}(b) = \sum_o \max_{\theta \in \Theta_t^{a,o}} \theta^\top b = \max_{\theta \in \Theta_t^a} \sum_o \theta^\top b$$

where

$$\Theta_t^a = \{\sum_o \theta^{a,o} | \theta^{a,o} \in \Theta_t^{a,o}\}$$

For V_{t+1}^* we have

$$V_{t+1}^*(b) = \max_a V_t^a(b) = \max_{\theta \in \Theta_{t+1}} \theta^\top b$$

where $\theta_{t+1} = \cup_a \Theta_t^a$.

The complexity of value iteration depends on Θ_t. For each action a and observation o we have $|\Theta_t^{a,o}| \leq |\Theta_t|$. For each action a we have $|\Theta_t^a| \leq |\Theta_t^{a,o}|^{|O|} \leq |\Theta_t|^{|O|}$. For Θ_{t+1} we have $|\Theta_{t+1}| \leq \sum_a |\Theta_t^a| \leq |A| \cdot |\Theta_t|^{|O|}$.

The complexity is exponential in $|O|$ in each iteration! Pruning can help, but the exponential time seems unavoidable.

16.2.2 Hardness

Many computational problems related to POMDPs are hard. For the finite horizon, even with a unique start state, it is P-SPACE complete to compute the return of the optimal policy.

For the finite horizon, in the case that there are no observations, it is NP complete to compute the return of the optimal policy. (We will show this simple hardness result.)

Assume we have no observations. This implies that we need to compute a sequence of H actions. The optimal policy will be the sequence of H actions that will maximize the return. We will show a reduction to SAT.

Given a SAT formula over n variables we create the following POMDP. For each clause we create the following POMDP gadget. We have always two actions 0 and 1. We create two lines of length n, one line ends in a state with reward 1 and the other in a state with reward 0. All other rewards will be zero. We "think" of the state in the line as a number in $[1, n]$ and we associate the state i it with the variable x_i.

We start at the line ending with 0. All the transition of states in the line ending in 1 simply continue to the next state, i.e., from i to $i+1$ (for both actions). For the line ending in zero, for each state i where x_i nor $\bar{x}_i$ do not appear in the clause, we continue to the next state on this line, with both actions.

For state i in the line that ends in 0, if x_i appears in the clause, then in state s_i with action 1 we move to the $i+1$ state in the line ending in 1, and with action 0 we move to state $i+1$ in the line ending in zero. Similarly, if $\bar{x}_i$ appears in the clause, then in state s_i with action 0 we move to the $i+1$ state in the line ending in 1, and with action 1 we move to state $i+1$ in the line ending in zero.

It is not hard to see that the sequence of n actions will lead to the reward 1 if and only if it represents an satisfying assignment to the clause.

We now take the gadgets for all the clauses, and select our initial state to uniformly at random to be the start state of one of the gadget (which represent a clause).

If the SAT formula is satisfiable, there is an assignment that satisfies all the clauses. Therefore guarantees a return of 1.

Otherwise for each sequence of action, which represent an assignment, there is some clause which is not satisfied. This implies that the return is at most $1 - 1/m$, where m is the number of clause.

Remark: Actually, we can get from the same reduction also a hardness to approximate result. We simply consider the MAX-SAT problem. For 3-SAT it is NP-hard to decide distinguish between satisfying $7/8 + \epsilon$ of the terms or all the terms. This implies that it is NP-hard to distinguish between a return of $7.8+\epsilon$ or 1.

16.2.3 Policy Plan: Automata

A simple class to describe the policy is a finite automata with output, which is called a Moore automata. The outputs will be the selected actions and the inputs are the observations. While it is true that this class of policies is not universal, namely, not any policy can be represented in this way, many policies can be.

One nice observation regarding this class of polices is that we can efficiently compute the value function of a given automata. Assume that π is described by an automata. The value of π from state s when its automata is in state σ is

$$V^{\pi}(s;\sigma) = r(s,a) + \gamma \sum_{o,s'} p(s'|s,a) Ob(o|s'a) V^{\pi}(s';\sigma')$$

where $a = \pi(\sigma, o)$ the action selected by π when in state σ of the automata and observes o, and σ' is the next state in the automata, given we are in state σ and observe o.

This implies that we have a set of linear equations and we can solve them efficiently!

16.3 Reusable trajectories

Our goal is given a set of deterministic policies Π, to estimate for each $\pi \in \Pi$ the value $V^{\pi}(s_0)$, its expected return. We want a set of trajectories that would be relatively small compared to Π, hopefully logarithmic in $|\Pi|$.

We will generate a set of trajectories Γ in the following simple way. We use a completely random policy to select the actions. Namely, $\pi_R(a|b) = 1/|A|$. We generate m such trajectories.

We estimate the return of a policy $\pi \in \Pi$ in the following way. Let Γ_π all the trajectories that agree with π. (Recall, π is deterministic.) We set

$$\hat{V}^{\pi}(s_0) = \frac{1}{|\Gamma_\pi|} \sum_{h \in \gamma_\pi} return(h)$$

The main issue will be: how large m has to be, in order to guarantee with probability $1-\delta$ that the error for any $\pi \in \Pi$ in the estimation will be at most ϵ. We will look at m as a function of the error ϵ, the confidence δ, the number of policies $|\Pi|$, and the maximum return V_{max}.

The proof follows in the following steps. Let $acc_\pi(h) = 1$ if π agrees on all the actions in h. The following lemma states that for any policy, the probability that it agrees with the trajectory of length H is exactly $1/|A|^H$.

Lemma 16.1. *For a finite horizon H, for any policy π we have* $\Pr_h[acc_\pi(h) = 1] = \frac{1}{|A|^H}$.

Let D_π be the distribution over histories generated by policy π. The following lemma states that the distribution of the random policy, condition of policy π agreeing with the actions, is identical to that of generated by policy π.

Lemma 16.2.
$$D_\pi(h) = D_{\pi_R}(h|acc_\pi(h) = 1)$$

The following lemma states that with high probability, for every $\pi \in \Pi$ we have that the size of $|\Gamma_\pi|$ is exponential in the horizon.

Lemma 16.3. *For*
$$m > 8|A|^H \frac{V_{max}^2}{\epsilon^2} \log(2|\Pi|/\delta)$$
with probability $1 - \delta/2$ for every $\pi \in \Pi$ we have
$$|\Gamma_\pi| \geq \frac{m}{2^{H+1}} > 4\frac{V_{max}^2}{\epsilon^2} \log(2|\Pi|/\delta)$$

The following lemma states that if every Γ_π is of sufficient size, then for each policy π we have a good approximation.

Lemma 16.4. *If for every $\pi \in \Pi$ we have*
$$|\Gamma_\pi| \geq \frac{m}{2|A|^H} > 4\frac{V_{max}^2}{\epsilon^2} \log(2|\Pi|/\delta)$$
then with probability $1 - \delta/2$, for every $\pi \in \Pi$ we have
$$|\hat{V}^\pi(s_0) - V^\pi(s_0)| \leq \epsilon$$

The theorem concludes and establishes the sample size for a PAC guarantee.

Theorem 16.3. *For*
$$m > 8|A|^H \frac{V_{max}^2}{\epsilon^2} \log(2|\Pi|/\delta)$$
with probability $1 - \delta$, for every $\pi \in \Pi$ we have
$$|\hat{V}^\pi(s_0) - V^\pi(s_0)| \leq \epsilon$$