CS168: The Modern Algorithmic Toolbox
Lecture #14: Markov Chain Monte Carlo

Tim Roughgarden & Gregory Valiant*

May 11, 2016

The previous lecture covered several tools for inferring properties of the distribution that underlies a random sample. In this lecture we will see how to design distributions and sampling schemes that will allow us to solve problems we care about. In some instances, the goal will be to understand an existing random process, and in other instances, the problem we hope to solve has no intrinsic randomness, yet we will design a random process that will allow us to understand the problem.

1 The Monte Carlo Method/Simulation

The Monte Carlo Method is simply the approach of learning about a distribution, process, or system, via random sampling (i.e. via simulating the process). Especially today, this seems like a completely obvious idea. In part, this is because the Monte Carlo method is fundamentally intertwined with computers: we take fast simulation/computation for granted—without computers, the Monte Carlo method might not make much sense.

1.1 Extremely Brief History

Historically, the rise of the Monte Carlo method closely paralleled the availability of computing resources.

- The first real example of the Monte Carlo method is usually attributed to the Compte de Buffon, who described what is now known as "Buffon's needle". He proposed that $\pi \approx 3.1415\ldots$ could be approximated by tossing a large number of needles onto a floor that is covered in parallel floor panels. The probability that a randomly tossed needle (assuming a random location and random angle of rotation) lies entirely in a single floor panel can be computed as a simple expression involving π, and the ratio of the needle length to the width of each floor panel. Hence by counting the ratio of needles

that lie entirely within one floor panel (versus those that lie across two panels), Buffon suggested that π can be accurately estimates. Indeed, using Chebyshev's inequality from last lecture, and the bounds on the sum of independent 0/1 random variables, given n needles, the error in the ratio will be accurate to $O(1/\sqrt{n})$.

- The first real appreciation for the potential of Monte Carlo simulations occurred in the context of the Manhattan Project during the 1940's (and then later during the development of the hydrogen bomb). Many of the models of physics—particularly at the atomic level—are intrinsically probabilistic. Rather than trying to explicitly calculate the aggregate large-scale properties that result from the probabilistic behaviors of many particles, Ullam (soon joined by Von Neumann) realized that the newly available computers offered an alternative to trudging through the extremely difficult math: the Monte Carlo method. (The name "Monte Carlo Method" is attributed to Ullam, in reference to Ullam's gambling-addict uncle. . . .)

2 Markov Chain Monte Carlo

The Markov Chain Monte Carlo approach is simply the Monte Carlo approach applied to *Markov Processes*—namely, it is sampling from a distribution defined via a stochastic process known as a Markov Process.

Definition 2.1 A *Markov Process* has two parts: a set of *states* $S = \{s_1, s_2, \ldots\}$, and a transition function $P : S \times S \to [0, 1]$, where $P(s_i, s_j)$ is interpreted as the probability that the process transitions to state s_j given that it is in state s_j. Hence for any s_i, $\sum_j P(s_i, s_j) = 1$.

Given a Markov process defined by S and P, and an *initial state* $X_0 \in S$, the corresponding stochastic process $X_0, X_1, \ldots$ corresponds to, for all $i \geq 1$, selecting X_i from the distribution defined by $P(X_i, \cdot)$. The assumption that the distribution of X_i depends only on X_{i-1} is known as the *Markov Property*—namely that to predict the next step in the chain, X_i, all the relevant history in the entire sequence $X_0, \ldots, X_{i-1}$ is encapsulated in X_{i-1}. This is also sometimes referred to as the "memory-less property".

Note that we can also think of the transition probabilities, P, as a matrix, where there i, jth entry is $P(s_i, s_j)$.

Example 2.2 To give a toy example, consider modeling the weather as a Markov process. (Most weather models are very complicated Markov processes, including the models that estimate the probability of El Nino winters.) Suppose the states $S = \{sunny, cloudy, rainy\}$, and the transitions are defined by

$$P = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.6 & 0.2 & 0.2 \\ 0.5 & 0.2 & 0.3 \end{pmatrix},$$

where the top row represents $P(sunny, sunny) = 0.7, P(sunny, cloudy) = 0.1, P(sunny, rainy) = 0.1$, and the second row represents $P(cloudy, sunny) = 0.6, P(cloudy, cloudy) = 0.2$, etc.

Given a Markov model, we can calculate things like $\Pr[X_1 = rain | X_0 = sunny]$, or, more generally, calculate the distribution of the state after t timesteps, X_t, given that $X_0 = s$.

2.1 Calculating Probabilities in Markov Models

Given a Markov Process, let $D(t, s)$ represent the distribution over values of X_t, given that $X_0 = s$. Note that we can represent $D(t, s)$ as a vector, whose ith entry represents $\Pr[X_t = s_i | X_0 = s]$.

There are two ways to estimate $D(t, s)$: direct computation, and via simulation (i.e. Markov Chain Monte Carlo).

Direct Computation: One can always directly calculate $D(t, s)$ from the matrix of transitions, P. To do this, note that $D(0, s) = [0 \dots 0, 1, 0, \dots]$, where this is the indicator vector of state s. For any $t \geq 1$,

$$D(t, s) = D(t - 1, s) \cdot P = D(0, s) \cdot P^t.$$

For example, revisiting the above example, we can compute

$$\Pr[X_{10} = rain | X_0 = sunny] = [1 \quad 0 \quad 0] \cdot P^{10} = [.65, .2, .15].$$

This calculation of P^t involves $O(\log t)$ matrix multiplications, by the "repeated squaring" trick of first calculating $P^2, P^4, P^8, P^{16}, \dots, P^{2^{\lfloor \log t \rfloor}}$ and then multiplying together the subset of these that make P^t. If the number of states is small, and P can be easily stored in memory, this calculation is rather efficient.

Markov Chain Monte Carlo [MCMC]: The second approach to estimating $D(t, s)$ is via Monte Carlo simulation. Simply start with $X_0 = s$, and for all $i = 1, \dots, t$, simulate the selection of X_i according to the distribution defined by P and X_{i-1}. If we do this k times, we have obtained k samples drawn from the distribution $D(t, s)$. This will take time $k \cdot t \cdot updateT$, where $updateT$ is the time to sample from the distribution defined by a row of P. For Markov chains that have a huge number of states, (and hence storing P in memory is prohibitive), often $updateT$ is still relatively small, and hence Markov Chain Monte Carlo is a reasonable way to estimate $D(t, s)$.

Example 2.3 One example of a powerful and surprising application of MCMC is in the development of computer programs for playing "Go". This game is played on a 19×19 board, and two players alternate placing black and white pebbles. One can place a pebble in any empty square, and the goal is to "capture" as many squares as possible, given a simple set of rules for defining what this means. Up until early 2016 when Google's Go player beat the current human Go champion, Go was viewed as a notoriously difficult game for computers, partially because there are so many possible moves, and hence it is hard to do anything resembling a deep and reasonably complete search over the game tree. Given this, it becomes essential that a go player (or computerized player) be able to accurately estimate the quality of different potential moves.

A relatively recent breakthrough in computer go was the following surprising observation: to evaluate the quality of a move, one can simply consider the probability that a given player will win *if both players play the rest of the game RANDOMLY—by simply alternating random moves*. This sequence of random plays corresponds to a Markov process. The set of states is the set of board configurations, and the transition function simply captures the random plays: at each time, a player randomly selects an empty square and puts his/her pebble there. When all the squares are filled, the game ends, and the Markov process enters either the 'win' state or the 'lose' state.

It is obviously impossible to explicitly calculate this probability of winning/losing, since the number of states of the chain is at least 2^{192}. Nevertheless, one can estimate these probabilities via MCMC (by simply randomly playing out the game many times, and looking at the fraction of the times a given player wins in the simulation). Note that if we are trying to estimate this probability to $\pm\epsilon$, it suffices to take $\approx 1/\epsilon^2$ random playouts (this follows from Chebyshev's inequality, together with a bound on the variance of a sum of independent 0/1 random variables, since each of the random playouts is independent).

The apparent reality that this is a good way to evaluate go moves means something special about go. My interpretation is the following: this means that go positions are either good or bad, and that the quality of the position is independent of the skill of the player who is playing the game. A good position is good irrespective of whether it is two skilled players playing eachother, or two idiots (i.e. random players) playing eachother. Of course, a good player will beat a bad player, but the configurations that are desirable for a player does not depend on the player's skill.

Example 2.4 PAGERANK: The single idea that put google on the map, originally, was the idea of using a MCMC to evaluate the relevance of a given webpage. The Markov Chain crudely models a person surfing the web. The states of the Markov Chain is the set of all web pages. The transitions are defined as follows: with probability $p = 0.8$, transition from the current web page to a random outgoing link (i.e. click on a random link), and with probability 0.2, transition to a random web page (i.e. close the window and open a new web browser, and go to a random page). The claim is that for large t, $D(t, s)$ is the distribution of web pages where probabilities correspond to the relevance of webpages. As we will see below, for large enough t, $D(t, s)$ is essentially independent of the initial state, s.

As with the MCMC for evaluating 'go' configurations, the above idea seems extremely simple/basic. This is the power of MCMC—one can often set up an extremely simple model of something, run MCMC, and end up with surprisingly accurate/informative predictions.

3 The Stationary Distribution

To what extent does $D(t, s)$ depend on the initial state, s? The intuition that a random walk will eventually "forget" where it started, holds in many cases (for example, the Pagerank algorithm—after all how does one begin on the cs168 webpage, and end up on some bieber

fan-site 45 minutes later?). The following theorem, known as the fundamental theorem of Markov chains, gives extremely simple conditions for this to hold:

Theorem 3.1 (The Fundamental Theorem of Markov Chains) *Consider a Markov Chain that satisfies the following two conditions:*

1. *For all pairs of states, s_i, s_j, it is possible to eventually get to state s_j if one starts in state s_i.*

2. *The chain is* aperiodic*: for a pair of states, s_i, s_j, consider the set of times $\{t_1, t_2, \ldots\}$ consisting of all t for which $\Pr[X_t = s_j | X_0 = s_i] > 0$. A chain is aperiodic if $gcd(\{t_1, \ldots\}) = 1$. (Basically, as long as the chain is not a directed cycle, or a bipartite graph, etc.)*

Then for all states s,

$$\lim_{t \to \infty} D(t, s) \to \pi,$$

where the distribution π is called the stationary distribution *of the chain, and is independent of t and the initial state.*

The motivation for calling π the stationary distribution is that if π is the limiting distribution for a chain defined by transition matrix P, it satisfies $\pi P = \pi$; hence it is stationary over time. Another immediate consequence of the fact that $\pi P = \pi$ is that the stationary distribution, π, is is a (left) eigenvector of the transition matrix of the chain, P, with eigenvalue 1.

Obviously the MCMC for evaluating Go moves does not satisfy these conditions—since there is no way of transitioning from a winning final state to a losing final state. The pagerank MCMC does satisfy this, because of the small probability of jumping to a random webpage. The weather example also satisfies this, since there is some probability of eventually having any weather, no matter the weather today.

4 Using MCMC to Solve Hard Problems

Up to now, we have been considering Markov Chains that model natural processes. We will now see how to use MCMCs to solve hard problems that do not inherently have any randomness. We will design our own probabilistic process to help us solve the problem, even though the problem does not necessarily have any inherent stochastic process.

RECIPE FOR SUCCESS: Given a hard search problem over a HUGE space:

1. Define a random walk/Markov Chain over the search space, such that the transition function makes the chain have a stationary distribution, π, that places a higher probability on "good" solutions to our search problem.

2. Run MCMC.

3. Wait/hope that the MCMC finds a good solution.

In the homework, we see a concrete realization of this recipe of success for the hard problem of finding a short traveling salesman tour of the national parks in the US.

Below we see another extremely cute concrete realization, due to Persi Diaconis (see the links on the course webpage for his much more thorough/eloquent description of this):

Example 4.1 Decrypting Substitution Ciphers: Suppose we are given a document that has been encrypted by substituting each letter for a strange symbol. We will apply the above recipe to reveal this mapping. The states of the markov chain will correspond to mappings between the symbols, and the letters. To define the transition function, we need some sense of how to score a given mapping is. Almost anything will work (e.g. defining the score to be the fraction of translated/decrypted words that appear in an english dictionary, or defining the score to be the probability of seeing the given sequence of decrypted characters, where this is calculated as the product of all consecutive PAIRS of characters, using the pair-wise probabilities based on real text—for example, if the current mapping gives a translation of 'bakjb ska en...', we compute the score as $\Pr[ba] \cdot \Pr[ak] \cdot \Pr[ki] \cdot \Pr[ib] \dots$, where $\Pr[ba]$ is an estimate of the probability that a follows b in typical text). Given a score function, we can define transitions as follows: given a mapping, consider switching the image of 2 random symbols: compute the score of the new mapping and the old one: if the new mapping has a better score, keep it, otherwise, keep the new score with some (small) probability, proportional to how much worse the score is. This transition function ensures that the stationary distribution of the chain puts much higher weight on better mappings.

Note that we need to sometimes make 'backwards progress'—i.e. keep a worse scoring mapping, in order to ensure that we satisfy the first condition of the Theorem; if we don't do this, our chain might get stuck at a local optima. Amazingly, the above chain seems to decrypt this sort of cipher in a few thousand iterations.

5 Mixing Time

MCMC is magical and solves many problems very well. The main issue, which might be obvious at this point, is that it might not be clear how long we need to wait until we are close to the stationary distribution: ie how long do we need to run the MCMC before we think we are drawing a sample from something resembling the stationary distribution. Namely how large does t need to be to ensure that $D(t, s) \approx \pi$?

Mathematically, the "time until we are close to π" is known as the *mixing time* of the chain, and is defined as the minimum time t s.t. no matter where we start (i.e. for all states s), the distance between $D(t, s)$ and π is at most $1/4$. Formally:

$$mixingTime = \min\{t : \max_s \|D(t, s) - \pi\|_{TV} \leq 1/4\}.$$

[Note: one sometimes sees this definition with the constant $1/4$ replaced by $1/e$.]

5.1 Eigenvalues and Power Iteration

It is often hard to estimate the mixing time of a given chain—particularly for those chains whose transition matrix is too large to write out completely. Nevertheless, it is both practically and conceptually useful to relate the mixing time of a Markov process to several concepts that we have already seen.

Consider the distributions of a Markov chain with transition matrix P after $0, 1, 2, \ldots$ timesteps, given an initial state (or distribution) represented by the vector v: namely, we have $v, vP, vP^2, vP^3, \ldots$. This sequence looks exactly like the sequence of calculations that we performed in the context of the "Power Iteration Algorithm" for computing the maximum eigenvalue of a matrix that we saw in lecture 8. The only difference is that for this sequence, we are given an explicit starting vector, v; in contrast, in the Power Iteration Algorithm, we selected a vector uniformly at random, and then repeatedly multiplied it by the matrix in question.

In the analysis of the power iteration algorithm, we saw that the ratio of the second largest eigenvalue of a matrix, λ_{max-1} to the largest, λ_{max}, determined the rate at which we could expect the vector to converge to the largest eigenvector. Similarly, we could hope that in most settings, the ratio between the largest eigenvalue of a transition matrix of a Markov chain (which will be 1), and the second largest eigenvalue, will roughly correspond to the rate at which vP^t converges to π, i.e. the mixing time. While we can't argue that this correspondence exactly holds, because the initial distribution v is not chosen randomly—in many special cases we can relate this ratio of eigenvalues to the mixing time. (You will explore this connection in this week's mini-project....)