Metropolis-Hastings and MCMC

Scott Hwang[1] and Aditya Mittal[2]

March 2021

1 Introduction

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method that allows us to estimate probability distributions by reframing the context of a Markov chain. Instead of using a Markov chain to learn about a distribution, we reverse the role, using a distribution to model a Markov chain, designing it such that the its equilibrium state *estimates* our model distribution. Here we provide context and a brief introduction to the algorithm, as well as a few applications to illustrate it's efficacy.

2 Markov Chains[1]

A *Markov chain* is a model of transition states whose probabilities depend only on the last occurring state. Here is an example of a Markov chain transition matrix.

$$M = \begin{bmatrix} 0.8 & 0.4 \\ 0.2 & 0.6 \end{bmatrix}$$

Entry $M_{(1,2)}$ represents that given an element in the second state (column number), there is a probability of 0.4 that it exists in the first state (row number) after one iteration. Multiplying this matrix by an initial state matrix gives the new distribution of states after the transition probabilities are applied.

$$M \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0.8a + 0.4b \\ 0.2a + 0.6b \end{bmatrix}$$

An important property of a Markov chain M is that there exists some vector v where applying the transition probabilities yields the same vector. This is called the *equilibrium state.*

$$Mv = v$$

Here is the equilibrium state v for M.

$$v = \begin{bmatrix} 0.667 \\ 0.333 \end{bmatrix}$$

$$Mv = \begin{bmatrix} 0.8 & 0.4 \\ 0.2 & 0.6 \end{bmatrix} \begin{bmatrix} 0.667 \\ 0.333 \end{bmatrix} = \begin{bmatrix} 0.667 \\ 0.333 \end{bmatrix}$$

Taking n iterations of M is modeled by taking M to the nth power. Over many iterations of the Markov chain, the distribution of states will be the same as the equilibrium state regardless of initial distribution vector p.

$$\lim_{n\to\infty} M^n p = v$$

This property of approaching an equilibrium state over many iterations regardless of the initial distribution will be important later. Another property of this equilibrium state is that it can be used to estimate the distribution of states at any random moment when no other information is known. That is, in this example, we can estimate that if this transition matrix is applied over some amount of iterations, we can estimate that 0.667 will be in state 1 and 0.333 will be in state 2 at any arbitrarily chosen time. Therefore, we can use the equilibrium state of the Markov Chain matrix in essence to sample from a distribution, which we will see later.

3 Monte Carlo Simulations[2]

A *Monte Carlo simulation* is a method of utilizing random sampling to approximate otherwise difficult to calculate numerical values. This is equivalent to unbiased sampling in other experiments: we take a small, random proportion of the greater population, expecting to model and derive behavior from that small sample set without needing to go through every single data point. But, this doesn't mean we can't use more data. In fact, the more samples we take, the more accurate of an approximation we'll get. For example, we can estimate π effectively using a Monte Carlo simulation by playing a game of darts.

Say we are playing a game of blind darts, randomly throwing darts at a circular dartboard placed against its square backboard. What is the probability we hit the dartboard? With some basic geometry, this can be solved fairly easily: $\frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$. However, this assumes that we know what π is to begin with. So we can actually reverse the question, and rewrite our equation to get that $\pi = 4 \cdot \frac{\text{Area of Circle}}{\text{Area of Square}}$, which we can estimate with our random darts, counting how many land in the circle and dividing that by total darts thrown to estimate our proportion of areas. This will give us an approximation for π.

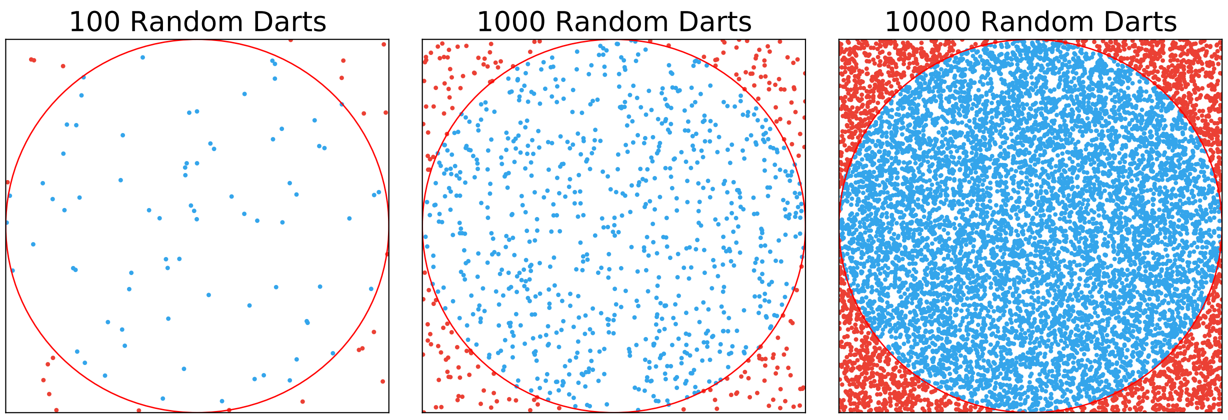


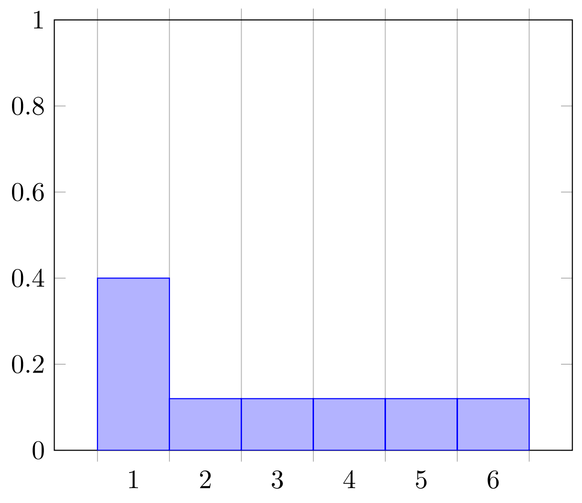
Figure 1: A circular dartboard with a square backboard and radius r

	Trial 1	**Trial 2**	**Trial 3**
Darts in Circle	77	800	7854
Total Darts	100	1000	10000
$\pi \approx$	3.08	3.2	3.1416

As you can see in the table, the more samples we took (i.e. more random darts thrown), the better our approximation for π became. This is the essence of a Monte Carlo simulation: using random sampling, to calculate otherwise computationally hard numbers using the Law of Large Numbers to estimate to an arbitrary degree of precision.

4 Markov Chain Monte Carlo (MCMC)[1]

Here is the general goal for MCMC sampling: We want to sample from a distribution. What this means is that we want to essentially be able to take random values from a set of values, which can be discrete or continuous, and we want these random values to appear with a likelihood defined by the probability function $P(x)$. As an example, say you have a weighted die that lands on the value 1 with probability 0.4, and all other numbers equally. When the die is thrown, that is the equivalent of taking a sample. As the process occurs infinitely many times, the distribution can be plotted.



4.1 Dependent Sampling

MCMC is a method of dependent sampling. Dependent sampling takes the previous sample into account when generating the next sample from the function, as opposed to independent sampling, where each sample is simply taken at random from the distribution. This usually means that the next sample is close to the previous sample, so the process can be thought of as "walking" along the distribution. Before moving on, think about how Markov chains work: the probability of the next state is dependent on the last state (sample) taken. One drawback of this is that dependent sampling is more inefficient, taking larger sample sizes to model a distribution. So then why use dependent sampling? Sometimes independent sampling is just not possible, for reasons beyond the scope of this project. Here is an example:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This is Bayes' Theorem. To briefly explain, it updates a conditional probability $P(A|B)$ given knowledge of prior (unconditional) probabilities $P(A)$ and $P(B)$. Without explaining why, it is often the case that $P(B)$ (the normalizing constant) is not easily calculable in higher dimensions, nor can it be obtained through independent sampling. MCMC is a common method to sample from $P(B)$.

4.2 How it works

The general idea is to find a Markov chain whose stationary distribution (equilibrium state v) is the desired distribution. There are different methods of obtaining this Markov chain, including Hamiltonian Monte Carlo and the Metropolis-Hastings algorithm.

5 The Metropolis-Hastings Algorithm (MH)[2]

Metropolis-Hastings follows a fairly simple set of instructions to find a distribution of $P(x)$ even when we don't know all the details of it. We do this by defining $\pi(x)$ that is *proportional* to $P(x)$ that we can sample from. The reason why we do this is because even if $\pi(x) = c \cdot P(x)$, it will still have the same relative probabilities (or densities for a continuous function) that mimic $P(x)$. From there, we can iterate and perform our "random walk" along the distribution until we find the steady state. Here's how it works:

1. Give some arbitrary initial $x_0 \rightarrow$ This is where we start our "walk"

2. For $t = 1, 2, 3, ..., n$ with n as our number of "steps" in our walk

   (a) Sample a new state y from $Q(y|x)$. Here y is our "proposed" state for where we will walk to x_{t+1}

(b) Compute $A = \min(1, \frac{\pi(y)}{\pi(x_t)})$

(c) With probability A, "walk" from $x_{t+1} = y$, and with probability $1 - A$ set $x_{t+1} = x_t$

This may seem like a lot, but once broken down, it's fairly straightforward. Let's start with what $\pi(x)$ and $Q(y|x)$ is. $\pi(x)$ is our function proportional to $P(x)$, so we use it as a "goodness" function: if $\pi(x)$ is really high (i.e. likely to occur), that means $P(x)$ is also really high for its distribution. So when we are deciding where we are going to walk, $\pi(x)$ tells us if we should walk in that direction. $Q(y|x)$ is the Markov chain of this Markov chain Monte Carlo method. Given a position x, Q tells us how "near" y is to x in terms of how probable it is to walk to it. This is our dependent sampling function, proposing a new state to look at given a current one, just like how Markov chains work. A, the acceptance probability, is just that: if our proposed position is better than our current one, we are guaranteed going to go there. If it is not better, we *might* go there, but only proportional to how much worse it is. However, the A presented is not the *true* MH algorithm. The A shown is for the Metropolis algorithm. Hastings improved it for a wider range of Q distributions, redefining $A = \min(1, \frac{\pi(y)Q(x_t|y)}{\pi(x_t)Q(y|x_t)})$, but many times the Q terms divide out if it's a symmetric distribution.

That's it! Note, however, we never explicitly defined a Markov chain with transition states. We just gave rules for transitioning from one state to another, but not necessarily listed all transitions. The longer we let our algorithm run and "walk" along the distribution, the more spots it will have visited, and the closer it will converge to our distribution $P(x)$. Here's an interesting example of MH.

5.1 Code Breaking with Metropolis-Hastings

We've been given a 1-to-1 ciphertext, where every letter (including space) has been replaced with another random letter. How can we find the key to this cipher? We can't brute force the search as there are 27! different keys—that's on the order of 10^{28} possibilities. Instead, we can perform our random walk along the different keys until we find a solution that matches English. Here's how we do it following the previous instructions:

1. Give some arbitrary initial key x_0
2. For $t = 1, 2, 3, ..., n$ with n as our number of "steps" in our walk
   (a) Propose a new key y by randomly swapping 2 letters in our original key x_t.
   (b) Compute $A = \min(1, \frac{\pi(y)}{\pi(x_t)})$ to see if one resembles English more.
   (c) With probability A, accept our new key and set $x_{t+1} = y$, and with probability $1 - A$ set $x_{t+1} = x_t$

We were able to generate a simple Q by proposing new keys by swapping 2 letters, but what about $\pi(x)$? This is where some of the human element of this algorithm comes along: just as we designed Q as simply as we wanted, we can define π in another very simple way. Using a text that we know for a fact is English (i.e. the book *A Tale of Two Cities*), we can generate a transition matrix that tells us the conditional probabilities of any letter following another. The closer our key decrypts to English text, the higher the probability π should output, giving us a very distinct way to measure how "good" our key is to decrypting to English. Running this enough times should eventually converge to English.

```
  1. YS LXYS N ONZZIF KXYWX Y SII AIBRFI GI LXI XNCOHI LRKNFO GU XNCO...
 31. PT LOPT N ANZZER DOPFO P TEE GEBURE CE LOE ONSAHE LUDNRA CK ONSA...
 61. IS OTIS L FLDDER PTICT I SEE AEGURE VE OTE TLNFME OUPLRF VY TLNF...
 91. IS FLIS O DOGGER BLICL I SEE UETARE HE FLE LONDME FABORD HY LOND...
     ⋮
151. IS THIS A DAGGER WHICH I SEE BEFORE ME THE HANDLE TOWARD MY HAND...
```

Everything after this will be that same sentence, indicating we've found the equilibrium distribution that most closely matches to English, and hence decoded the message.