MCMC-MH: Learning an Optimal Proposal Function

Josh Hellerstein

In this paper we explore improvements to the Markov Chain Monte Carlo (MCMC)- Metropolis–Hastings (MH) algorithm. Particularly, we seek to learn an optimal proposal function, such that we can more efficiently sample from an arbitrary probability distribution. We highlight the challenges of optimally learning the proposal function, as our primary objective is to make the learned MH algorithm more efficient (in an empirical sense) compared to the unlearned MH algorithm. We then propose a method for optimizing the proposal distribution, and show its optimality with respect to hand-crafting proposal distribution parameters. The software for this project is available on Github.

Introduction. Markov Chain Monte Carlo (MCMC) methods are a class of approximate inference algorithms aimed at sampling from a probability distribution which is hard to directly sample from (usually, because of an intractable state space). One of the these algorithms is the Metropolis-Hastings algorithm, which aims to efficiently explore the state-space. The algorithm does this by constructing a markov chain over a set of states, with a Hamiltonian (transition likelihoods) proportional to the true steady-state distribution we're interested in sampling from.

Although the MCMC-MH algorithm has been around for a couple of decades, there has been recent interest in improving the algorithm's efficiency though adaptive proposal methods, and gradient methods (Hamiltonian MCMC-MH). The problem with adaptive methods are that they usually have to break the standard MCMC assumptions, and don't have the same stationary distribution guarantees. The Hamiltonian/gradient MCMC-MH method attempts to reduce the correlation between successive sampled states, but with poorly initialized hyper-parameters, fail to sample efficiently.

Concretely, at a high level, our goal is to learning-augment the MCMC-MH algorithm so that:

1. We efficiently sample/explore the distribution, well enough to learn its sufficient statistics.
2. Our new algorithm is less sensitive to initial conditions/parameters, and we have to tweak less about the algorithm (hyper-parameters) to achieve a good result.
3. We don't need domain expertise to arrive at a good result.

MCMC Problem Setup. Let's say we want to estimate some unknown/intractable distribution $\pi(x)$, over some large number of states x. This occurs in many practical applications, for instance if you want to estimate the most

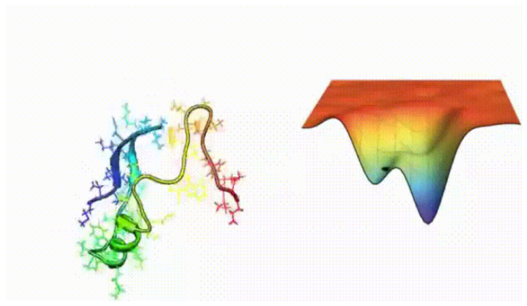

Fig. 1. Protein Folding For example, we might want to estimate a probability distribution over all possible configurations of a protein, to find the mode of the distribution.

likely configuration of a protein, you would need to compute $\arg\max \pi(x)$. This can be hard if there are many states i.e. many possible ways for the molecules in a protein to be arranged (angles between them for instance). Instead of computing $\pi(x)$ directly, we can set up a Markov Chain which has a stationary distribution $\pi(x)$, and random walk on this Markov Chain.

The Metropolis-Hastings Algorithm. One can show that if you walk on a Markov Chain given by some likelihoods $p(x'|x)$ according to MH, you will achieve a stationary distribution that asymptotically approaches $\pi(x)$.

While there are derivations of why the Acceptance probability in MH guarantees convergence to $\pi(x)$ (if your $p(x'|x)$ forms an irreducible, aperiodic, and ergodic Markov Chain), once central piece of intuition is that our acceptance probability ratios are proportional to our stationary probability ratios.

$$\frac{\alpha(x'|x)}{\alpha(x|x')} \propto \frac{\pi(x')}{\pi(x)} = \frac{p(x'|x)}{p(x|x')}$$

So the input to our MH algorithm that uniquely defines the random walk can just be the likelihoods $p(x'|x)$ (and it will still converge to our distribution $\pi(x)$). These are often much easier to compute "what's the likelihood of transitioning to state x' from x?" versus "what's the overall likelihood of being in state x?"

We also note that the MH algorithm uses some proposal function $q(x'|x)$ which we use to select a candidate for which state to actually transition to. Note that this "proposal" factor is corrected for in the acceptance probability ratio, as to maintain the "detailed balance" constraints of the Markov Chain.

Algorithm 1 Metropolis-Hastings algorithm

```
Initialize $x^{(0)} \sim q(x)$
for iteration $i = 1, 2, \ldots$ do
    Propose: $x^{cand} \sim q(x^{(i)} | x^{(i-1)})$
    Acceptance Probability:
        $\alpha(x^{cand} | x^{(i-1)}) = \min \left\{1, \frac{q(x^{(i-1)} | x^{cand}) \pi(x^{cand})}{q(x^{cand} | x^{(i-1)}) \pi(x^{(i-1)})}\right\}$
    $u \sim$ Uniform $(u; 0, 1)$
    if $u < \alpha$ then
        Accept the proposal: $x^{(i)} \leftarrow x^{cand}$
    else
        Reject the proposal: $x^{(i)} \leftarrow x^{(i-1)}$
    end if
end for
```

Fig. 2. MH Algorithm The algorithm performs a random walk on a Markov Chain.

What does this means for our optimization question? How can we select a good proposal function $q(x|x)$ as to efficiently explore the distribution? We can't just adapt $q(x'|x)$ on the fly, as the acceptance function would change over time, meaning we'd be solving for a "changing" or "moving" stationary distribution – breaking some of our assumptions about a unique distribution for our Markov Chain. Further, it would be hard to learn the proposal function from our likelihoods $p(x'|x)$ because it would require explicitly computing them for some subset of our states x (because our states are intractable). How should we choose which subset of x to compute them over? Also our states x often come in many different representations (vectors, functions, sets) and it's unclear how we could universally exploit structure using machine learning.

The Solution. The way we will frame this as a learning problem, is to treat this problem as a hyper-parameter optimization problem. We need to use a model which can operate on only a few samples, so that our learning is more optimal than simply running the MCMC-MH algorithm to convergence. Gaussian Processes used in "Bayesian hyper-parameter optimization" are primed for this type of scenario.

Procedure for estimating $q(x, \theta)$:

1. Random init θ, x_0
2. Run MH for L iterations with $q(x, \theta)$
3. Score according to some metric $M() \to \mathbb{R}$
4. Determine a new θ to try next using a Gaussian Process, maximizing an acquisition function (such as "expected improvement"). This is a standard procedure.
5. Repeat K times

After you estimate a good $\hat{\theta}$, you can run MH until convergence with $q(x, \hat{\theta})$.

We want to make sure that $L * K$ is relatively small compared to the number of iterations it takes for convergence. We also want to choose a sufficient metric $M()$ which will accurately reflect the success of a proposal function.

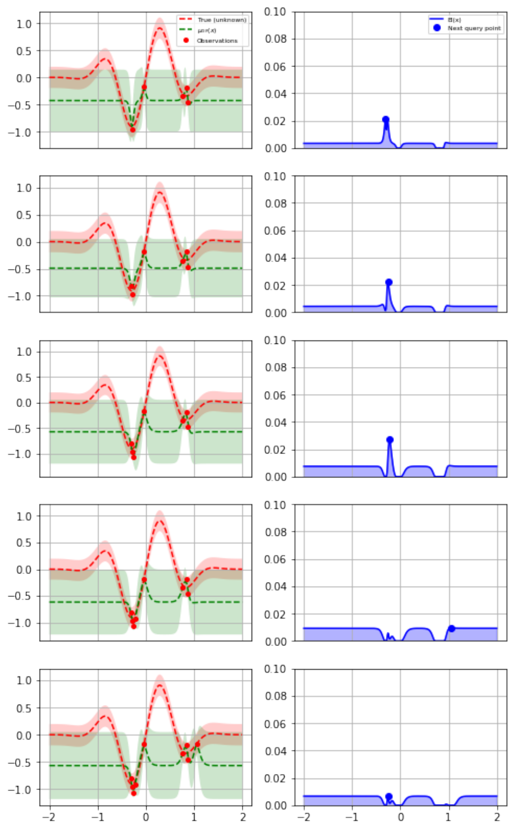

Fig. 3. Gaussian Process An example of iterated Sampling of a hyper-parameter θ for $q(x, \theta)$ scored on some metric function $M() \to \mathbb{R}$

Gaussian Processes treat the domain of our continuous metric function $M()$ a multivariate Gaussian distribution, with some kernel function (co-variance function). This allows us to estimate a the minimum/maximum of $M()$ using only a few samples.

For our implementation (described below) we try 3 different metric functions $M()$ to see which one is better at estimating a good θ.

For a sampled trajectory L, we try 3 metric functions:

1. Max-Probability(L)
2. Mean-Probability(L) / Variance-Probability(L)
3. $\sum_{x \to x' \in L} proposal - distance(x, x') * \frac{\pi(x')}{\pi(x)}$

A Toy MCMC-MH Problem: Cipher Decryption. We set up a toy problem to run MCMC-MH on. It is intractable, so we can get a good feel for how well our optimization technique works.

Problem: Decode a text document that's encoded with some substitution cipher, mapping each letter of the alphabet to a different one.

States: Alphabet mappings i.e. (a,b,c...z) $\to$ (f,e,h ... u). In code this is simply given as a string i.e.

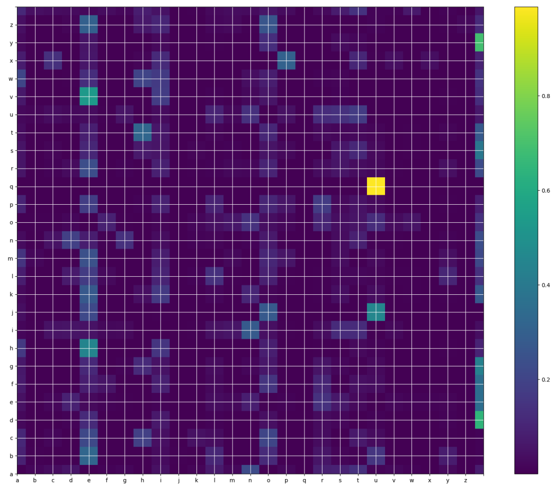

Fig. 4. Letter Transition Likelihood Our likelihood function that we use as $\pi(x)$ is based on the probability of a letter transition matrix i.e. a->a, a->b, a->c, from the book *War and Peace*

"$dujhgfoexpavimnytrlzqcwbsk$". Note that there are 26! possible states (permutations), so we have an intractable example problem.

Likelihood Function: We compute the log-likelihood of a given state by summing all the log-likelihoods of "letter transitions." We build a letter transition matrix using the famous book (in English) *War and Peace* to estimate letter transition likelihoods. The idea is that the state (a cipher) is more likely if it corresponds to a higher likelihood of letters being positioned next to each other.

True Document:
it was the best of times it was the worst of times it was the age of wisdom it was the age of foolishness it was the epoch of belief it was the epoch of incredulity it was the season of light it was the season of darkness it was the spring of hope it was the winter of despair we had everything before us we had nothing before us we were all going direct to heaven we were all going direct the other way in short the period was so far like the present period that some of its noisiest authorities insisted on its being received for good or for evil in the superlative degree of comparison only

Encrypted/ciphered Document:
wu cvt usl bltu gm uwylt wu cvt usl cgztu gm uwylt wu cvt usl vjl gm cwtigy wu cvt usl vjl gm mgghwtsaltt wu cvt usl ldgks gm blhwlm wu cvt usl ldgks gm wakzlixhwup wu cvt usl tlvtga gm hwjsu wu cvt usl tlvtga gm ivzqaltt wu cvt usl tdzwaj gm sgdl wu cvt usl cwaulz gm iltdvwz cl svi lelzpuswaj blmgzl xt cl svi aguswaj blmgzl xt cl clzl vhh jgwaj iwzlku ug slvela cl clzl vhh jgwaj iwzlku usl guslz cvp wa tsgzu usl dlzwgi cvt tg mvz hwql usl dzltlau dlzwgi usvu tgyl gm wut agwtwltu vxusgzwuwlt watwtuli ga wut blwaj zlklweli mgz jggi gz mgz lewh wa usl txdlzhvuwel iljzll gm kgydvzwtga gahp

Proposal Procedure ($q(x,\theta)$)**:**

1. N-swaps $\sim exp^{-\theta}$
2. Swap 2 letters of the current cipher x N-swaps times.

Thus our proposal function is tunable (to number of letter swaps we do), and our objective is to learn θ.

Toy Problem: Implementation Specs.

- **Language:** Python3
- **Computer:** Dell XPS 4 core i7, Ubuntu 18.04.
- **Gaussian Process Optimizer:** Scikit-optimize function, "gp-minimize()", default settings.
- **MH:** Custom Implementation
- **Metric Functions:** Custom Implementation

Toy Problem: Implementation Constants.

- **Cap on runtime:** Stop the algorithm after 200,000 iterations (rarely reaches this, but to bound our runtime).
- Ciphers are randomized for every run, even on trajectories L's we use to learn θ.
- θ **Search Range:** $(0.01, 20)$
- L**, length of path:** 500
- K**, number of runs of the GP to determine optimal** θ**:** 10

Results. As stated in the intro, we will determine which Metric ultimately works best (and if the optimization procedure learns anything) by looking at how long it takes to get a sufficient statistic. In this toy problem case, the "sufficient statistic" we will use is how many MH iterations it takes to find the mode of our stationary distribution, in other words, **how many iterations until it successfully decrypts the text**.

Below is the estimated θ values that go into the proposal function evaluated for the 3 different metric functions (see previous section for which corresponds to which metric function). Each result is calculated by running the proposal function θ estimation procedure 10 times and taking the mean.

Metric	Mean $\hat{\theta}$
1	1.8
2	0.94
3	1.25

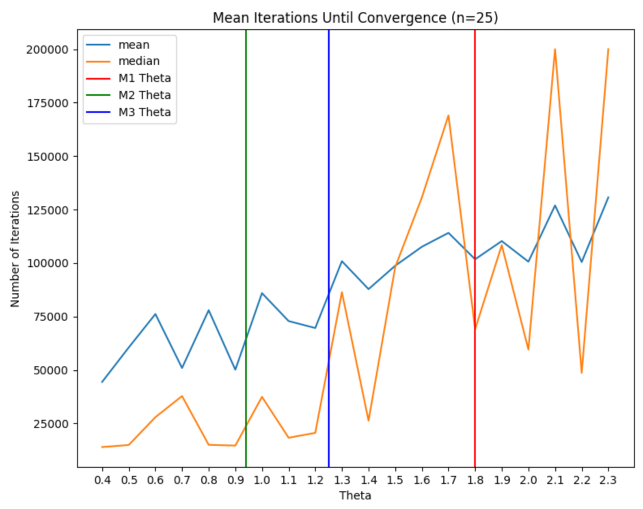

Fig. 5. Iterations Until Convergence We plot the number of iterations until we reach the mode of the distribution for different values of θ. M1, M2, and M3 are each of the metric function's estimations of θ

To determine which $\hat{\theta}$ is truly optimal, we run the MH algorithm to completion (until we decrypt the document/find the mode) for a proposal function with θ in range (0.4, 0.5, 0.6 ... 2.3) $q(x,\theta)$. For each of these experiments of θ, we run it to completion 25 times and take the mean and median, because number of iterations until completion is highly randomized (random ciphers, random initialization, etc...).

From the results above, we see that the median of each of the 25 trials tends to be much lower than the mean, indicating that we have strong outliers on the high end. This is likely because we have a very discrete problem (only stops when we reach the mode exactly), and there are likely some times when we just don't terminate, because we got "unlucky" with the random swaps.

Another trend we notice is that both the mean and median indicate it's preferable to have a low θ parameter, meaning the problem setup is better suited to "1-letter swap only" for the proposal distribution (because our θ parameters is in an exponential distribution). This is likely a domain specific issue, because for most other applications, there isn't a "clear answer" - it's more along the line of "what sigma for our Gaussian should we choose?".

Discussion and Future Perspectives. Overall, we determine that the best metric function for optimizing the proposal distribution $q(x,\theta)$ is **Metric 2: Mean-Probability(L)/Variance-Probability(L)**. The motivation for this metric is to stay in an area where there is a high probability with low variance (similar to a *"Sharpe Ratio"*, a concept in financial applications). It makes sense that this is the best metric for finding the mode of the distribution - we're intentionally optimizing for regions which get stuck / stay in high probability, with little movement. However, it's not clear that this is the best metric for determining every statistic about our stationary distribution $\pi(x)$.

In the case that we're looking for other metrics about our distribution aside from the mode (mean, variance, other moments, etc...) then we should use **Metric 3:**, as it corresponds to big moves that also correspond to good points with good stationary probability, indicating it's meaningfully selecting an optimal parameter of our proposal distribution.

Irrespective of our metric function, we can conclude that our optimization procedure (using a Gaussian Process to find a good $q(x,\theta)$ worked much better than a random initialization of θ. Our GP could have chosen values of θ anywhere in the interval $(0.01, 20)$ but was able to hone in / learn something about our constructed Markov Chain, simply by observing 10 trajectories of length 500 iterations - significantly smaller than the total number of iterations until convergence.

One of the limitations of the experimentation process was the ability to pin down low variance results, because of the randomized aspects (randomized ciphers, every time) to the MH algorithm. To solve this, we made sure to take many samples, to hopefully provide a more true estimate of the performance.

Software Availability. The code for this report is available on Github:
https://github.com/jhell96/Metropolis-Hastings-Augmented-Proposal