# Numerical Methods with FHE: Monte Carlo Sampling

Jordan Cotler

## 1 Review of Markov Chain Monte Carlo

Suppose we have a probability distribution with probability density $p(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$. To sample from $p(\mathbf{x})$, we will use the Metropolis-Hastings algorithm.[1] First, we choose an initial value $\mathbf{x}_0$. Now we build up a sequence $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ...\}$ as follows. For $i = 0, 1, 2, ...,$

1. Sample $\Delta\mathbf{x}$ from the uniform distribution on the ball with radius $c$, centered at the origin. We say that our step size is $c$.

2. Sample a number $r$ uniformly from 0 to 1. If $r < p(\mathbf{x}_i + \Delta\mathbf{x})/p(\mathbf{x}_i)$, let $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}$. Otherwise, let $\mathbf{x}_{i+1} = \mathbf{x}_i$.

For an appropriate step size $c$, if we build up a sufficiently large sequence $\{\mathbf{x}_i\}_{i=1}^N$, then this sequence behaves as a collection of points randomly sampled from $p(\mathbf{x})$.

There are many variations on this theme. One comment is that often $p(\mathbf{x})$ is more naturally written in the form $p(\mathbf{x}) = e^{-S(\mathbf{x})}$. Then in Step 2 above, it is more natural to compute the inequality $-\log(r) > S(\mathbf{x}_i + \Delta\mathbf{x}) - S(\mathbf{x}_i)$, although of course this is equivalent to the inequality above by taking the minus logarithm of both sides (and using the fact that on $(0, \infty)$, the function $-\log(\cdot)$ is strictly monotonically decreasing).

## 2 Proposed FHE Application

Suppose that Alice has a secret probability distribution $p(\mathbf{x})$ that she wants to sample. We imagine that Bob has computational resources which Alice wants to use,

[1] The probability distribution $p(\mathbf{x})$ need not be normalized for most applications, but we will ignore this issue for now since it is easy to deal with.

but Alice does not want Bob to know either (i) the probability distributions or (ii) the samples. Further suppose that $p(\mathbf{x})$ can be written as

$$p(\mathbf{x}) = g(h(\mathbf{a}, \mathbf{x})) \tag{1}$$

where $\mathbf{a} = (a_1, ..., a_n)$, $h(\mathbf{a}, \mathbf{x})$ is a function that depends polynomially on the $a_i$'s but can otherwise have arbitrary dependence on $\mathbf{x}$, and $g : (0, \infty) \to \mathbb{R}$ is strictly monotonic. (For instance, we could have $g(y) = e^{-y}$ which is reminiscent of the previous section above.)

The functions $g$ and $h$ are known to both Alice and Bob, but the parameters $\mathbf{a}$ are only known to Alice. This is the sense in which only Alice knows the probability distribution. The procedure is as follows: Alice encrypts $\mathbf{a}$ using FHE, and sends the encrypted version to Bob. Alice also generates and encrypts an initial sample $\mathbf{x}_0$, and sends it to Bob. (Alternatively, Bob could generate this initial sample, unencrypted.) Then for $i = 0, 1, 2, ...,$

1. Bob samples $\Delta\mathbf{x}$ from the uniform distribution on the ball with radius $c$, centered at the origin. (Here, $c$ is pre-decided, and unencrypted.)

2. Bob computes samples a number $r$ uniformly from 0 to 1, and then computes $s_i = \pm \operatorname{sgn}\left((h(\mathbf{x}_i + \Delta\mathbf{x}) - h(\mathbf{x}_i)) - g^{-1}(r)\right)$, which is an encrypted value. We choose "+" if $g$ is monotonically increasing, and "−" if $g$ is monotonically decreasing. Then $\mathbf{x}_{i+1} = \mathbf{x}_i + (s_i + 1)\,\Delta\mathbf{x}$, which is also an encrypted value.

If the error becomes too large after an intermediate step, then Alice may have to decrypt and encrypt the current value of $\mathbf{x}_i$ as well as the parameters $\mathbf{a}$.

# 3 Aside: Garbled Circuits and Monte Carlo Sampling

A different cryptographic setting is as follows. Suppose that Alice has some secrete probability distribution $p(\mathbf{x})$ that Bob wants to sample from. As above, we may suppose that form of the function $p(\mathbf{x}) = g(h(\mathbf{a}, \mathbf{x}))$ is known to both Alice and Bob, but only Alice knows the parameters $\mathbf{a}$. Then Alice could encrypt a low-depth arithmetic circuit $C$ which maps $C : \mathbf{x}_i \mapsto \mathbf{x}_{i+1}$, much in the same way as the procedure above. In this case, Bob can sample from the unknown distribution. For technological reasons, it is important that $C$ be a low depth arithmetic circuit, since the known garbled circuits protocols have similar difficulties as the known FHE protocols (even though both classes of protocols rely on different methods).

There are many interesting and obvious applications of garbled circuit Monte Carlo sampling. Here, let me give an example which is less immediately apparent. There are many dynamical programming algorithms, with applications in AI and finance, which entail the Monte Carlo sampling of some policy distributions $\pi_i(\mathbf{x})$. For instance, in Monte Carlo tree sampling, or in dynamical programming applications of options pricing. Indeed, it is conceivable that these policy distributions would be proprietary.

# 4 Parameterization of the Probability Distribution in Function Space

For practical reasons, we have considered probability distributions of the form $p(\mathbf{x}) = g(h(\mathbf{a}, \mathbf{x}))$ where $h(\mathbf{a}, \mathbf{x})$ is a function that depends polynomially on the $a_i$'s but can otherwise have arbitrary dependence on $\mathbf{x}$, and $g : (0, \infty) \to \mathbb{R}$ is strictly monotonic. As mentioned before, in many applications it is natural to choose $g(y) = e^{-y}$.

Here, we enumerate some useful forms for the function $h(\mathbf{a}, \mathbf{x})$:

1. **Polynomial.** Consider $h(\mathbf{a}, \mathbf{x})$ of the form

$$h(\mathbf{a}, \mathbf{x}) = \sum_{i_1, \ldots, i_d = 1}^{K} a_{i_1 \cdots i_d} \, x_1^{i_1} \cdots x_d^{i_d} \tag{2}$$

   where $\mathbf{a} \in \mathbb{R}^{\binom{K+d}{K}}$ so that each component of $\mathbf{a}$ is an $a_{i_1 \cdots i_d}$.

2. **Laurent series.** Similarly to the polyonomial case, we consider $h(\mathbf{a}, \mathbf{x})$ of the form

$$h(\mathbf{a}, \mathbf{x}) = \sum_{i_1, \ldots, i_d = -K}^{K} a_{i_1 \cdots i_d} \, x_1^{i_1} \cdots x_d^{i_d} \tag{3}$$

   where $\mathbf{a} \in \mathbb{R}^{\binom{2K-1+d}{2K-1}}$ so that each component of $\mathbf{a}$ is an $a_{i_1 \cdots i_d}$.

3. **Quotients of polynomials.** In a similar vein, we may consider

$$h(\mathbf{a}, \mathbf{x}) = \frac{\sum_{i_1, \ldots, i_d = 1}^{K} a^{(\mathrm{num})}_{i_1 \cdots i_d} \, x_1^{i_1} \cdots x_d^{i_d}}{\sum_{i_1, \ldots, i_d = 1}^{K} a^{(\mathrm{denom})}_{i_1 \cdots i_d} \, x_1^{i_1} \cdots x_d^{i_d}} \tag{4}$$

   where $\mathbf{a} = (\mathbf{a}^{\mathrm{num}}, \mathbf{a}^{\mathrm{denom}}) \in \mathbb{R}^{\binom{K+d}{K}^2}$.

4. **Fourier series.** Suppose, for concreteness, that our distribution is one-dimensional (i.e., $\mathbf{x} = x \in \mathbb{R}$), and defined on the compact interval $[0, 2\pi]$. Then we might let

$$h(\mathbf{a}, \mathbf{x}) = a_0 + \sum_{k=1}^{K} \left( a_{2k-1} \cos(kx) + a_{2k} \sin(kx) \right) \tag{5}$$

where $\mathbf{a} = (a_0, a_1, ..., a_{2K}) \in \mathbb{R}^{2K+1}$. This is easy to generalize to other compact regions, and the higher-dimensional analog is also straightforward.

5. **Wavelets.** Instead of doing Fourier expansions, one can also do one of many wavelet expansions.

6. **Kernels.** Suppose we are interested in sampling *functions*, such as Gaussian processes. This is different than previously considered cases above. Then we may have

$$p[\mathbf{h}(\mathbf{x})] \propto \exp\left( -\frac{1}{2} \int d^d\mathbf{x}\, d^d\mathbf{y}\, h(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) h(\mathbf{y}) \right) . \tag{6}$$

So in this context, it appears natural to parameterize the kernel $K(\mathbf{x}, \mathbf{y})$. For instance, if $K(\mathbf{x}, \mathbf{y}) = K(|\mathbf{x} - \mathbf{y}|)$, i.e. it is translation-invariant, then it may be natural to parameterize $K(|\mathbf{x} - \mathbf{y}|)$ as a quotient of polynomials in Fourier space (i.e., after Fourier-transforming with respect to $|\mathbf{x} - \mathbf{y}|$). For instance, if $K(\mathbf{x}, \mathbf{y}) = (\nabla^2_{\mathbf{x}} + m^2)\, \delta^{(d)}(\mathbf{x} - \mathbf{y})$, then in Fourier space it becomes $\mathbf{k}^2 + m^2$.

# 5 Agent-based Stochastic Simulations

Suppose Alice wants to perform a secret, stochastic agent-based simulation on Bob's computers. We sketch how to do this with FHE. For a detail and excellent review of stochastic agent-based simulations, see [3].

Suppose the system has $M$ kinds of agents. The number of each agent at a fixed time is stored as an element of the vector $\vec{v}$. We require two classes of functions: (i) $\alpha_i = \alpha_i(\vec{v})$ for $i = 1, ..., R$ are polynomials in the elements of $\vec{v}$, where $\alpha_i$ gives the rate for the $i$th reaction to occur (e.g., there are a total of $R$ possible reactions), and (ii) $\vec{r}_i$ for $i = 1, ..., R$ are vectors such that $\vec{v} \to \vec{v} + \vec{r}_i$ implements the $i$th reaction. Then the procedure is as follows: Alice gives encrypts the initial configuration $\vec{v}_0$ and gives it to Bob. Bob stores $(t_0 = 0, \vec{v}_0)$ as the configuration of the system at the initial time. Then, for $i = 1, 2, ...,$

1. Bob independently samples two numbers $r_1, r_2$ uniformly at random between 0 and 1. The numbers are known to Bob, and thus unencrypted.

2. Bob computes $\alpha_0 = \sum_{j=1}^{R} \alpha_j(\vec{v}_i)$, and lets $t_{i+1} = t_i + \frac{1}{\alpha_0}\log(1/r_1)$. These numbers are encrypted.

3. Bob computes the vector $\vec{\beta} = \big(\alpha_1(\vec{v}_i), \cdots, \alpha_R(\vec{v}_i)\big)/\alpha_0$. Then Bob computes the new vector
$$\vec{\gamma} = \big(\sum_{j=1}^{1}\beta_j\,,\,\sum_{j=1}^{2}\beta_j\,,\,...\,,\,\sum_{j=1}^{R}\beta_j\big)\,, \tag{7}$$
which is a vector of accumulating partial sums. Both $\vec{\beta}$ and $\vec{\gamma}$ are encrypted.

4. Bob computes
$$\vec{\delta} = \frac{1}{2}\,\mathrm{sgn} \odot \big(\vec{\gamma} - (r_2, r_2, ..., r_2)\big) + (1/2, 1/2, ..., 1/2) \tag{8}$$
where "$\odot$" designates a component-wise application of the sgn function. Here, $\vec{\delta}$ is encrypted.

5. Bob now computes the encrypted vector
$$\vec{\varepsilon} = (\delta_1, \delta_2 - \delta_1, \delta_3 - \delta_2, ..., \delta_R - \delta_{R-1})\,. \tag{9}$$

6. Finally, Bob computes the encrypted vector
$$\vec{v}_{i+1} = \vec{v}_i + \sum_{j=1}^{R}\varepsilon_j\,\vec{r}_j \tag{10}$$
and stores $(t_{i_1}, \vec{v}_{i+1})$ as the next time step of the system.

It may be the case that even a single time step of agent-based stochastic simulations are outside the realm of current practical FHE implementations. In particular, even a single time step may make the FHE error blow up. Nonetheless, encrypted stochastic simulations of this kind may find great application in the future.

# References


[1] Gilad-Bachrach, Ran, et al. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy." *International Conference on Machine Learning*. 2016.

[2] Chen, Hao, Kim Laine, and Rachel Player. "Simple encrypted arithmetic library-SEAL v2.3.0." 2017.

[3] Erban, Radek, Jonathan Chapman, and Philip Maini. "A practical guide to stochastic simulations of reaction-diffusion processes." *arXiv:0704.1908* (2007).