Sampling methods

In practice, the probabilistic models that we use are often quite complex, and simple algorithms like variable elimination may be too slow for them. In fact, many interesting classes of models may not admit exact polynomial-time solutions at all, and for this reason, much research effort in machine learning is spent on developing algorithms that yield *approximate* solutions to the inference problem. This section begins our study of such algorithms.

There exist two main families of approximate algorithms: *variational* methods[1], which formulate inference as an optimization problem, as well as *sampling* methods, which produce answers by repeatedly generating random numbers from a distribution of interest.

Sampling methods can be used to perform both marginal and MAP inference queries; in addition, they can compute various interesting quantities, such as expectations $\mathbb{E}[f(X)]$ of random variables distributed according to a given probabilistic model. Sampling methods have historically been the main way of performing approximate inference, although over the past 15 years variational methods have emerged as viable (and often superior) alternatives.

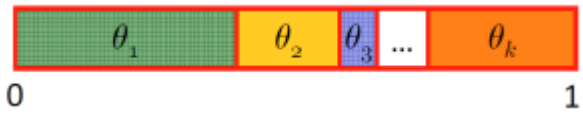
Sampling from a probability distribution

As a warm-up, let's think for a minute how we might sample from a multinomial distribution with k possible outcomes and associated probabilities $\theta_1, \ldots, \theta_k$.

Sampling, in general, is not an easy problem. Our computers can only generate samples from very simple distributions[2], such as the uniform distribution over $[0, 1]$. All sampling techniques involve calling some

kind of simple subroutine multiple times in a clever way.

In our case, we may reduce sampling from a multinomial variable to sampling a single uniform variable by subdividing a unit interval into k regions with region i having size θ_i. We then sample uniformly from $[0, 1]$ and return the value of the region in which our sample falls.



Reducing sampling from a multinomial distribution to sampling a uniform distribution in [0,1].

Sampling from directed graphical models

⊕

Our technique for sampling from multinomials naturally extends to Bayesian networks with multinomial variables, via a method called *ancestral* (or *forward*) sampling. Given a probability $p(x_1, \ldots, x_n)$ specified by a Bayes net, we sample variables in topological order. In other words, we start by sampling the variables with no parents; then we sample from the next generation by conditioning these variables' CPDs to values sampled at the first step. We proceed like this until the n variables have been sampled.

In our earlier model of a student's grade, we would first sample an exam difficulty d' and an intelligence level i'. Then, once we have samples, d', i' we generate a student grade g' from $p(g \mid d', i')$. At each step, we simply perform standard multinomial sampling.

A former CS228 student has created an interactive web simulation for visualizing Bayesian network forward sampling methods. Feel free to play around with it and, if you do, please submit any feedback or bugs through the Feedback button on the web app.

Monte Carlo estimation

Sampling from a distribution lets us perform many useful tasks, including marginal and MAP inference, as well as computing integrals of the form

$$\mathbb{E}_{x\sim p}[f(x)] = \sum_x f(x)p(x).$$

If $f(x)$ does not have a special structure that matches the Bayes net structure of p, this integral will be impossible to perform analytically; instead, we will approximate it using a large number of samples from p. Algorithms that construct solutions based on a large number of samples from a given distribution are referred to as Monte Carlo (MC) methods[3].

Monte Carlo integration is an important instantiation of the general Monte Carlo principle. This technique approximates a target expectation with

$$\mathbb{E}_{x\sim p}[f(x)] \approx \frac{1}{T}\sum_{t=1}^{T} f(x^t),$$

where $x^1, \ldots, x^T$ are samples drawn according to p.

It is easy to show that the expected value of I_T, the MC estimate, equals the true integral. We say that I_T is an unbiased estimator for $\mathbb{E}_{x\sim p}[f(x)]$. Moreover as $I_T \to \mathbb{E}_{x\sim p}[f(x)]$ as $T \to \infty$. Finally, we can show that the variance of I_T equals $\text{var}_P(f(x))/T$, which can be made arbitrarily small with T.

Rejection sampling

$\oplus$ A special case of Monte Carlo integration is rejection sampling. We may use it to compute the area of a region R by sampling in a larger region with a known area and recording the fraction of samples that falls within R.

For example, we may use rejection sampling to compute marginal probabilities of the form $p(x = x')$: we may write this probability as $\mathbb{E}_{x\sim p}[\mathbb{I}(x = x')]$ and then take the Monte Carlo approximation. This will amount to sampling many samples from p and keeping ones that are consistent with the value of the marginal.

Importance sampling

Unfortunately, this procedure is very wasteful. If $p(x = x')$ equals, say, 1%, then we will discard 99% of all samples.

A better way of computing such integrals is via an approach called *importance sampling*. The main idea is to sample from a distribution q (hopefully roughly proportional to $f \cdot p$), and then *reweight* the samples in a principled way, so that their sum still approximates the desired integral.

More formally, suppose we are interested in computing $\mathbb{E}_{x \sim p}[f(x)]$. We may rewrite this integral as

$$\begin{aligned}
\mathbb{E}_{x \sim p}[f(x)] &= \sum_x f(x) p(x) \\
&= \sum_x f(x) \frac{p(x)}{q(x)} q(x) \\
&= \mathbb{E}_{x \sim q}[f(x) w(x)] \\
&\approx \frac{1}{T} \sum_{t=1}^{T} f(x^t) w(x^t)
\end{aligned}$$

where $w(x) = \frac{p(x)}{q(x)}$. In other words, we may instead take samples from q and reweigh them with $w(x)$; the expected value of this Monte Carlo approximation will be the original integral.

Now the variance of this new estimator equals

$$\text{var}_{x \sim q}(f(x)w(x)) = \mathbb{E}_{x \sim q}[f^2(x)w^2(x)] - \mathbb{E}_{x \sim q}[f(x)w(x)]^2 \geq 0$$

Note that we can set the variance to zero by choosing $q(x) = \frac{|f(x)|p(x)}{\int |f(x)|p(x)dx}$; this means that if we can sample from this q (and evaluate the corresponding weight), all the Monte Carlo samples will be equal and correspond to the true value of our integral. Of course, sampling from such a q would be NP-hard in general, but this at least gives us an indication for what to strive for.

In the context of our previous example for computing $p(x = x') = \mathbb{E}_{z \sim p}[p(x'|z)]$, we may take q to be the uniform distribution and apply importance sampling as follows:

$$\begin{aligned} p(x = x') &= \mathbb{E}_{z \sim p}[p(x'|z)] \\ &= \mathbb{E}_{z \sim q}[p(x'|z)\frac{p(z)}{q(z)}] \\ &= \mathbb{E}_{z \sim q}[\frac{p(x', z)}{q(z)}] \\ &\approx \frac{1}{T}\sum_{t=1}^{T}\frac{p(z^t, x')}{q(z^t)} \end{aligned}$$

Unlike rejection sampling, this will use all the examples; if $p(z|x')$ is not too far from uniform, this will converge to the true probability after only a very small number of samples.

Normalized importance sampling

Unfortunately, unnormalized importance sampling is not suitable for estimating conditional probabilities.

$$P(x = x'|e = e') = \frac{P(x = x', e = e')}{P(e = e')}$$

Because we would estimate the numerator using one approximation and the denominator using another approximation, the errors in the approximations may compound. For example, if the numerator is an under-estimate and the denominator is an over-estimate, the final probability could be a severe under-estimate.

To avoid this issue, we first generate samples to approximate the denominator $P(e = e')$. The numerator can then be estimated by counting the weighted number of samples with values $x = x'$. Since the numerator and denominator are both approximated using the same samples, the potential issue of errors compounding is avoided.

The final form of normalized importance sampling is thus

$$P(x = x' | e = e') = \frac{P(x = x', e = e')}{P(e = e')} \qquad \approx \frac{\frac{1}{T}\sum_{t=1}^{T} \delta_{x'}(z^t) w(z^t)}{\frac{1}{T}\sum_{t=1}^{T} w(z^t)}$$

Markov chain Monte Carlo

Let us now turn our attention from computing expectations to performing marginal and MAP inference using sampling. We will solve these problems using a very powerful technique called Markov chain Monte Carlo[4] (MCMC).

Markov Chain

A key concept in MCMC is that of a *Markov chain*. A (discrete-time) Markov chain is a sequence of random variables $S_0, S_1, S_2, \ldots$ with each random variable $S_i \in \{1, 2, \ldots, d\}$ taking one of d possible values, intuitively representing the state of a system. The initial state is distributed according to a probability $P(S_0)$; all subsequent states are generated from a conditional probability distribution that depends only on the previous random state, i.e. S_i is distributed according to $P(S_i \mid S_{i-1})$.

The probability $P(S_i \mid S_{i-1})$ is the same at every step i; this means that the transition probabilities at any time in the entire process depend only on the given state and not on the history of how we got there. This is called the *Markov* assumption.

$\oplus$ It is very convenient to represent the transition probability as a $d \times d$ matrix

$$T_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j).$$

If the initial state S_0 is drawn from a vector probabilities p_0, we may represent the probability p_t of ending up in each state after t steps as

$$p_t = T^t p_0,$$

where T^t denotes matrix exponentiation (we apply the matrix operator t times).

The limit $\pi = \lim_{t \to \infty} p_t$ (when it exists) is called a *stationary distribution* of the Markov chain. We will construct below Markov chain with a stationary distribution π that exists and is the same for all p_0; we will refer to such π as *the* stationary distribution* of the chain.

A sufficent condition for a stationary distribution is called *detailed balance*:

$$\pi(x')T(x \mid x') = \pi(x)T(x' \mid x) \text{ for all } x$$

It is easy to show that such a π must form a stationary distribution (just sum both sides of the equation over x and simplify). However, the reverse may not hold and indeed it is possible to have MCMC without satisfying detailed balance.

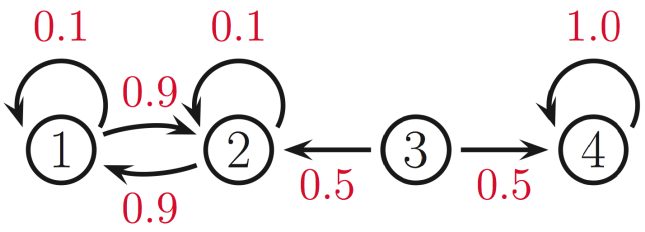
Existence of a stationary distribution

The high-level idea of MCMC will be to construct a Markov chain whose states will be joint assignments to the variables in the model and whose stationary distribution will equal the model probability p.

In order to construct such a chain, we first need to understand when stationary distributions exist. This turns out to be true under two sufficient conditions:

> *Irreducibility*: It is possible to get from any state x to any other state x' with probability > 0 in a finite number of steps.
>
> *Aperiodicity*: It is possible to return to any state at any time, i.e. there exists an n such that for all i and all $n' \geq n$, $P(s_{n'} = i \mid s_0 = i) > 0$.

The first condition is meant to prevent *absorbing states*, i.e. states from which we can never leave. In the example below, if we start in states $1, 2$, we will never reach state 4. Conversely, if we start in state 4, then we will never reach states 1,2. If we start the chain in the middle (in state 3), then clearly it cannot have a single limiting distribution.



A reducible Markov Chain over four states.

The second condition is necessary to rule out transition operators such as

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Note that this chain alternates forever between states 1 and 2 without ever settling in a stationary distribution.

Fact: An irreducible and aperiodic finite-state Markov chain has a stationary distribution.

In the context of continuous variables, the Markov chain must be *ergodic*, which is slightly stronger condition than the above (and which requires irreducibility and aperiodicity). For the sake of generality, we will simply require our Markov Chain to be ergodic.

Markov Chain Monte Carlo

As we said, the idea of MCMC algorithms is to construct a Markov chain over the assignments to a probability function p; the chain will have a stationary distribution equal to p itself; by running the chain for some number of time, we will thus sample from p.

At a high level, MCMC algorithms will have the following structure. They take as argument a transition operator T specifying a Markov chain whose stationary distribution is p, and an initial assignment x_0 to the variables of p. An MCMC algorithm then perform the following steps.

1. Run the Markov chain from x_0 for B *burn-in* steps.
2. Run the Markov chain from x_0 for N *sampling* steps and collect all the states that it visits.

Assuming B is sufficiently large, the latter collection of states will form samples from p. We may then use these samples for Monte Carlo integration (or in importance sampling). We may also use them to produce Monte Carlo estimates of marginal probabilities. Finally, we may take the sample with the highest probability and use it as an estimate of the mode (i.e. perform MAP inference).

Metropolis-Hastings algorithm

The Metropolis-Hastings (MH) algorithm is our first way to construct Markov chains within MCMC. The MH method constructs a transition operator $T(x' \mid x)$ from two components:

A transition kernel $Q(x' \mid x)$, specified by the user

An acceptance probability for moves proposed by Q, specified by the algorithm as

$$A(x' \mid x) = \min\left(1, \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)}\right).$$

At each step of the Markov chain, we choose a new point x' according to Q. Then, we either accept this proposed change (with probability α), or with probability $1 - \alpha$ we remain at our current state.

Notice that the acceptance probability encourages us to move towards more likely points in the distribution (imagine for example that Q is uniform); when Q suggests that we move into a low-probability region, we follow that move only a certain fraction of the time.

In practice, the distribution Q is taken to be something simple, like a Gaussian centered at x if we are dealing with continuous variables.

Given any Q the MH algorithm will ensure that P will be a stationary distribution of the resulting Markov Chain. More precisely, P will satisfy the detailed balance condition with respect to the MH Markov chain.

To see that, first observe that if $A(x' \mid x) < 1$, then $\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1$ and thus $A(x \mid x') = 1$. When $A(x' \mid x) < 1$, this lets us write:

$$\begin{aligned} A(x' \mid x) &= \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)} \\ P(x')Q(x \mid x')A(x \mid x') &= P(x)Q(x' \mid x)A(x' \mid x) \\ P(x')T(x \mid x') &= P(x)T(x' \mid x), \end{aligned}$$

which is simply the detailed balance condition. We used $T(x \mid x')$ to denote the full transition operator of MH (obtained by applying both Q and A). Thus, if the MH Markov chain is ergodic, its stationary distribution will be P.

Gibbs sampling

A widely-used special case of the Metropolis-Hastings methods is Gibbs sampling. Given an ordered set of variables $x_1, \ldots, x_n$ and a starting configuration $x^0 = (x_1^0, \ldots, x_n^0)$, consider the following procedure.

Repeat until convergence for $t = 1, 2, \ldots$:

> Set $x \leftarrow x^{t-1}$.
>
> For each variable x_i in the order we fixed:
>
> 1. Sample $x_i' \sim p(x_i \mid x_{-i})$
>
> 2. Update $x \leftarrow (x_1, \ldots, x_i', \ldots, x_n)$.
>
> Set $x^t \leftarrow x$

We use x_{-i} to denote all variables in x except x_i. It is often very easy to performing each sampling step, since we only need to condition x_i on its Markov blanket, which is typically small. Note that when we update x_i, we *immediately* use its new value for sampling other variables x_j.

Gibbs sampling can be seen as a special case of MH with proposal $Q(x_i', x_{-i} \mid x_i, x_{-i}) = P(x_i' \mid x_{-i})$. It is easy check that the acceptance probability simplifies to one.

Assuming the right transition operator, both Gibbs sampling and MH will eventually produce samples from their stationary distribution, which by construction is P.

There exist simple ways of ensuring that this will be the case

> To ensure irreducibility, the transition operator Q with MH should be able to potentially move to every state. In the case of Gibbs sampling, we would like to make sure that every x_i' can get sampled from $p(x_i \mid x_{-i}^t)$.
>
> To ensure aperiodicity, it is enough to let the chain transition stay in its state with some probability.

In practice, it is not difficult to ensure these requirements are met.

Running time of MCMC

A key parameter to this algorithm in the number of burn-in steps B. Intuitively, this corresponds to the number of steps needed to converge to our limit (stationary) distribution. This is called the *mixing time* of the Markov chain[5]. Unfortunately, this time may vary dramatically, and may sometimes take essentially forever. For example, if the transition matrix is

$$T = \begin{bmatrix} 1-\epsilon & \epsilon \\ \epsilon & 1-\epsilon \end{bmatrix},$$

then for small ϵ it will take a very long time to reach the stationary distribution, which is close to $(0.5, 0.5)$. At each step, we will stay in the same state with overwhelming probability; very rarely, we will transition to another state, and then stay there for a very long time. The average of these states will converge to $(0.5, 0.5)$, but the convergence will be very slow.

This problem will also occur with complicated distributions that have two distinct and narrow modes; with high probability, the algorithm will sample from a given mode for a very long time. These examples are indications that sampling is a hard problem in general, and MCMC does not give us a free lunch. Nonetheless, for many real-world distributions, sampling will produce very useful solutions.

Another, perhaps more important problem, is that we may not know when to end the burn-in period, even if it is theoretically not too long. There exist many heuristics to determine whether a Markov chain has *mixed*; however, typically these heuristics involve plotting certain quantities and estimating them by eye; even the quantitative measures are not significantly more reliable than this approach.

In summary, even though MCMC is able to sample from the right distribution (which in turn can be used to solve any inference problem), doing so may sometimes require a very long time, and there is no easy way to judge the amount of computation that we need to spend to find a good solution.