[Monte Carlo Simulation for estimators: An Introduction](https://www.youtube.com/watch?v=5nM5e2_1OQ0)

Sampling examples: generative model, or computing an expectation of a function

Sampling is discrete due to a finite sample size n, while integrals are not

**Monte Carlo Methods**

Las Vegas algorithms always return precisely the correct answer, but they take a lot of time

Monte Carlo provides approximations with some amount of error, depending on how many resources were expended

Monte Carlo methods are preferred when having a precise deterministic approach is intractable

**Sampling and Monte Carlo Methods**

We can sample from a distribution and use these samples to estimate some desired quantity

*Why Sampling?*

Sampling is good for when there is a costly but tractable sum/integral, which can be approximated well enough by samples

Sampling is also used for intractable sums/integrals, or when we want sampling as our goal (we want the model to sample from the training distribution)

*Basics of Monte Carlo Sampling*

If a sum or integral is intractable, we can use Monte Carlo sampling by viewing it as an

expectation under some distribution and approximate the expectation w/ some average

Let s be the sum or integral to approximate rewritten as an expectation with p as the probability distribution (for a sum) or probability density (for an integral)



We can approximate s by drawing n samples x1 .. xn from p and forming the empirical average



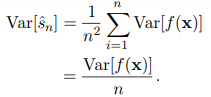
The estimator s is unbiased since



By the law of large numbers, if the samples xi are [iid](https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables) the average converges to the expected:

if the variance is bounded

The variance decreases and converges to 0 as n increases, as long as Var[f(xi)] < ∞



This also provides a way to figure out the amount of expected error of the Monte Carlo approx.

We compute both the empirical average of f(x(i)) and the empirical variance and divide the estimated variance by n to obtain an estimator of the variance of the totality s^n

The central limit theorem states that the distribution of s^ converges to normal with a mean of s and variance of Var[f(x)]/n, allowing us to estimate confidence intervals about the mean

However, this requires sampling from p(x) directly which is not always possible, so we may have to resort to importance sampling or Monte Carlo Markov chains

**Main idea - if we sample from the actual distribution p(x), we can get good estimates with a large sample size n**

**Importance Sampling**

We do not know which part of the integrand should play the role of p(x), and which should play the role of f(x) whose expected value is being estimated

There is no unique decomposition since p(x)f(x) can always be written as:

where we sample from q w/ an average of pf/q

Sometimes we wish to calculate an expectation given p and f, which gives us a reasonable decomposition

However, this decomposition may not be optimal in terms of the number of samples needed and accuracy

The optimal q\* can be derived easily, however; q\* corresponds to what is called optimal importance sampling

Any Monte Carlo estimator



can be transformed into an importance sampling estimator

Since q is where we are sampling from, and pf/q is the avg

The expected value of the estimator does not depend on q



The variance, however, is sensitive to q



The minimum variance occurs when q

where Z is the normalization constant chosen so q\*(x) integrates to 1

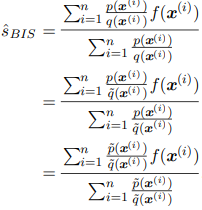
Good importance sampling distributions put more weight f(x) where the integrand is larger

When f(x) does not change sign, Var[sq] = 0 meaning a single sample is sufficient when the optimal distribution is used — however this is impractical since we don't have access to it

* Substituting pf/Z simplifies Var[sq] to Z/n
* This is true because if f(x) has a constant sign, then it probably diverges due to the higher weights given by f which means z -> ∞ and the variance simplifies to 1/∞

Another approach is called importance biased sampling, which does not require a normalized q or p

The estimator is given by:

where p~ and q~ are unnormalized forms of p and q and xi are samples from q

This estimator is biased since E[sBIS] ≠ s unless n -> ∞ (asymptotically unbiased)

A poor choice of q can make the efficiency of Monte Carlo methods much worse

If there are samples of q for which the followingis large, the variance of the estimator can get large:



This can happen when q(x) is tiny, and p and f are not small enough to cancel it

q is chosen to be a simple distribution in order to sample easily; when q is simple and x has a high dimensionality, then q matches p(x)|f(x)| poorly, which can cause q >> pf which results in an underestimation, or q << pf, which results in a huge overestimation

This is common when the dimensionality of xn is huge, since the dynamic range of joint probabilities of the vector elements of x is wide

Importance sampling is useful in many deep learning algorithms

* It accelerates training for neural networks w/ a large # of outputs
* Provides an estimate for the log-likelihood used to train a variational autoencoder
* Can be used to improve the estimate of gradient of the cost function for models like classifiers where difficult examples exclusively can reduce the variance of the gradient

**Main idea - various importance sampling methods, with different intermediate distributions q, have different standard deviations despite having the same 𝝁; these standard deviations change how many samples are needed to estimate the distribution well**

[**Markov Chain Monte Carlo Methods**](https://www.youtube.com/watch?v=uvYTGEZQTEs)

Sometimes, there is no tractable way for drawing exact samples from the distribution pmodel(x) or from a low variance importance sampling distribution q(x)

This most often happens when pmodel(x) is an undirected model

We introduce a Markov chain to approximately sample from pmodel(x)

Markov chain Monte Carlo methods are the family of algorithms which use Markov chains

The general guarantees for MCMC techniques apply when the probability of any given state is nonzero, which motivates the use of an EBM, where:



Consider a joint distribution over two variables p(a, b)

In order to sample a, we must draw from p(a | b) and in order to sample b we must draw from p(b | a), which seems to be an intractable chicken-egg problem

Directed models avoid this since their graph is directed and acyclic

**Ancestral sampling** samples each of the variables in topological order, conditioning on the parents of each variable which have been guaranteed to be sampled

* Efficient, single pass method of obtaining a sample

In an EBM, we avoid the chicken-egg problem by sampling from a Markov chain

We start with a state **x** as an arbitrary value, which we then update such that it becomes a very nearly fair sample over time

A markov chain is defined by a random state **x** and a transition distribution T(**x'** | **x**) specifying the probability of a random update yielding **x'** from **x**

If **x** has countably many states, represent this state as just a positive integer x, with different integer values mapping back to different states **x**

Suppose we have infinitely many Markov chains running in parallel

If we have qt(x) where qt is some arbitrary distribution and t is the # of timesteps elapsed, we want it to eventually converge to p(x) based on all the Markov chain steps

We can describe the probability distribution q using a vector v with



where x corresponds to the mapped state to integer

The probability of a single Markov chain's state landing in x' is given by:



Represent the effect of T using a matrix A



We can now see how the entire distribution over all Markov chains running in parallel shifts as we apply an update



We can think of the process as exponentiating the matrix A:



Each of the columns of A represents a probability matrix; it is a **stochastic matrix**

If there is a nonzero probability of transferring from any state x to any other state x', the largest eigenvalue is real and equals 1

The [eigenvalues](https://www.youtube.com/watch?v=PFDu9oVAE-g&t=111s) are exponentiated over time:



All eigenvalues therefore decay to 0, and A is guaranteed to have only one eigenvector w/ eigenvalue 1

The process converges to a **stationary distribution** or **equilibrium distribution** in which:

(since 1 is the only eigenvalue, and A is eigendecomposed)

This holds for every additional step; basically, at some point after many repeated steps, the states stop changing

To be stationary, v must be an eigenvector with corresponding eigenvalue 1, so that once we have reached equilibrium, repeated applications of the transition distribution do not change the *distribution* over states of various Markov chains (although each state changes, of course)

* Two similar states will have the same distribution, but may sample diff. values from them

If we have chosen T correctly, the stationary distribution q will be equal to p, which we wish to sample from

Properties of markov chains with countable states can be generalized to continuous variables

A Markov chain w/ transition operator T will converge to a fixed point described by the equation:

Expectation over dist. T where all states **x** drawn from q to yield **x'**

When **x** is continuous, it corresponds to an integral

Running the Markov chain until it reaches equilibrium is known as **"burning in"** the chain

After it has been burned in, infinitely many samples can be drawn

They are identically distributed, but any few successive samples will be highly correlated with each other; we can return only every n samples to prevent successive correlation bias

* Highly correlated b/c similar states (which tend to be subsequent) have similar T distribs

Markov chains are expensive since it takes a lot of time to transition from one sample to another reasonably decorrelated sample at equilibrium

To get truly independent samples, you can run Markov chains in parallel

Practitioners usually use as many Markov chains as the size of the minibatch (around 100)

We do not know in advance the **mixing time** - time needed to reach equilibrium

We know that the chain mixes when At has lost all eigenvalues from A except for the unique eigenvalue of 1, so the magnitude of the second largest eigenvalue determines the mixing time

In practice, we cannot represent the Markov chain in terms of a matrix since the number of states our model can visit is so large it is infeasible to represent v, A, or eigenvalues of A

**Main idea - Using an EBM Markov chain as a way to learn a distribution (when we do not have access to p or q), we update the state x to x' based on a distribution T, and after many repeated steps, this converges to a stationary distribution; a good T must be used in this case**

[**Gibb's Sampling**](http://youtube.com/watch?v=QaojSzk7Hpw)

We can draw samples from q by repeatedly updating x ← x' ~ T(**x'** | **x**)

How do we ensure q(**x**) is a good distribution?

* Derive T with a given learned pmodel (with sampling from EBMs)
* Directly parametrize T and learn it so its stationary distribution implicitly defines pmodel

We use Markov chains to draw samples from EBMs defining pmodel(x)

We want the q(**x**) for Markov chains to be pmodel(x), and we therefore must choose a good T(**x'**|**x**)

To achieve this, we can use **Gibbs' Sampling**, where we sample from T by selecting one variable xi and sampling it from pmodel conditioned on its neighbors in the undirected graph G defining the EBM

We can sample several variables at once as long as they are conditionally independent given their neighbors

* Conditional independence occurs when two events have a correlation but there is no causation between them (their shared parent event is the form of causation)

Gibbs Sampling approaches which update multiple variables at once are called **block Gibbs' sampling**

**Main idea - an EBM in the form of a Markov chains lets us analytically derive T based off the graph structure and operations, or alternatively, we can parametrize and learn T. The first approach can be accomplished through Gibbs' sampling in an undirected model where we sample variables 1 at a time from the EBM conditioned on their neighbors, and update them in the graph**

**The Challenge of Mixing Between Separate Modes**

MCMC methods tend to **mix** poorly

Ideally, successive samples from the Markov chain would visit different regions in x space proportional to the region's probability

However, in high dimensional cases, successive samples lay in the same region and are very correlated, something known as **slow mixing**

* This can be interpreted as noisy gradient descent w.r.t the state of the chain (the random variables being sampled)

The chain tends to take very small steps, with E(xt) lower or approximately equal to Ext-1

Eventually, after many small steps, the chain reaches a new mode with low energy and tends to take "random walks" around that mode and stays there unless it finds a rare escape route

* e-E(x) means that the probability is slightly increasing each step, visiting exclusively high probability regions
* The high probabliity region where one variable is conditioned on the others means that one variable will be updated only in a very small range

In Gibb's sampling, the number of steps required to travel between 2 modes is dependent on the energy barrier between them

If the energy barrier is high energy (low probability region), it is exponentially less likely for one mode to change into the other

This creates a problem when two high probability modes are separated by low probability energy barriers, especially when each Gibbs' sampling step must update only a small subset of variables whose values are conditioned/dependent on other variables (the only case this doesn't happen is when variables are conditionally independent)

Fig 17.1

Shows how a model with 2 independent variables mixes well

Also shows how a model with 2 highly correlated variables does not mix well since the update on one variable must be conditioned on the other and as a result makes it difficult for the chain to move away from the starting point

Another figure shows 2 separated modes with extremely slow mixing as a result of it being difficult to change modes while altering only one variable at a time (due to conditioning/dependencies and high correlation, you cannot change multiple variables at once)

Consider an EBM over 2 variables a and b, which are both binary with a sign (either -1 or 1)

If E(a, b) = -wab where w is a large positive number, the model expresses a strong belief that a and b have the same sign

The conditional distribution over b of the model is given by P(a = 1 | b = 1) = sig(w), where if w is large, the sigmoid saturates and the probability of assigning b to 1 is close to 1

According to pmodel(a | b), both models should have the same signs meaning Gibbs' Sampling rarely flips the signs of these variables

This problem can sometimes be resolved by finding groups of highly dependent units and updating all of them simultaneously

However, when the dependencies are complicated, it is intractable to draw a sample from this group (this is why we used Markov chains in the first place, since Pmodel is intractable)

In models with latent variables defining p(x, h), we draw samples of x by sampling between pmodel(x | h) and from pmodel(h | x)

From the point of view of mixing rapidly, we want pmodel(h | x) to have a very high entropy

However, from the point of view of learning a representation, we want h to encode info about x in order to reconstruct it well, implying that h and x should have a lot of mutual information

We often learn generative models which encode x into h but do not mix very well, especially with Boltzmann machines where the Markov chain struggles with sharp distributions

Fig 17.2

We see Gibbs' sampling being applied to a deep Boltzmann machine

All this could make MCMC methods less useful when the distribution of interest has a manifold structure with a separate manifold for each class; in this case, the distribution is concentrated around multiple modes separated by regions of high energy

* This usually happens in classification tasks where MCMC methods converge slowly due to poor mixing