Lecture 4: Hidden Markov Models & Monte-Carlo Methods

Thibault Randrianarisoa

University of Toronto, Winter 2026

January 27, 2026

Statistical Sciences
UNIVERSITY OF TORONTO

Table of contents

1. **Hidden Markov Models**
   - Basic Definitions
   - Forward/Backward Algorithm
   - Viterbi Algorithm
2. **Monte-Carlo Methods**
   - Ancestral Sampling
   - Simple Monte Carlo
   - Importance Sampling
   - Rejection Sampling

Hidden Markov Models (HMMs)

Sequential data

We generally assume data are i.i.d, however this may be a poor assumption:

- Sequential data is common in time-series modelling (e.g. stock prices, speech, video analysis) or ordered (e.g. textual data, gene sequences).
- Recall the general joint factorization via the chain rule

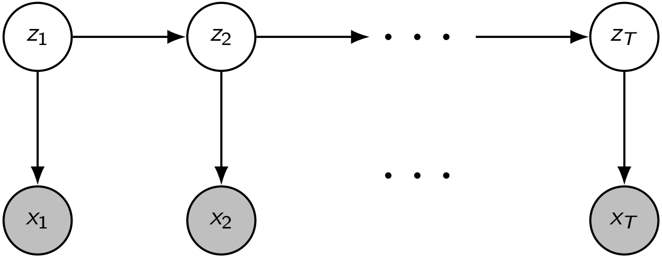
$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1) \quad \text{where } p(x_1 | x_0) = p(x_1).$$

- But this quickly becomes intractable for high-dimensional data: each factor requires exponentially many parameters to specify as a function of T.
- So we can make the simplifying assumption that our data can be modeled as a first-order Markov chain

$$p(x_t | x_{1:(t-1)}) = p(x_t | x_{t-1})$$

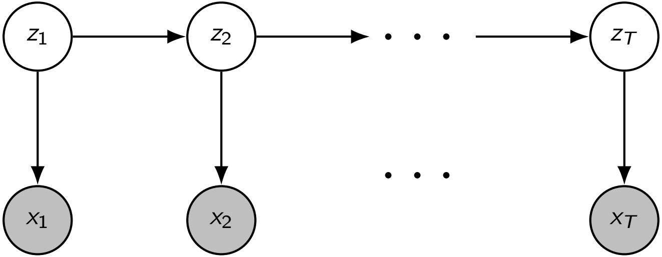
Sequential data

- In certain cases, Markov chain assumption is also restrictive.
- The state of our variables is not fully observed. Hence, we introduce Hidden Markov Models



Hidden Markov Models (HMMs)

- HMMs hide the temporal dependence by keeping it in an unobserved state.
- No assumptions on the temporal dependence of observations is made.
- For each observation x_t, we associate a corresponding unobserved hidden/latent variable z_t



- The joint distribution of the model becomes

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{t-1}) \prod_{t=1}^{T} p(x_t|z_t)$$

Hidden Markov Models (HMMs)

In HMMs the observations are not limited by a Markov assumption of any order.

Assuming we have homogeneous model (i.e., $p(z_t|z_{t-1})$ and $p(x_t|z_t)$ do not depend on t), we only have to know three sets of distributions:

1. **Initial distribution**: $\pi(i) = p(z_1 = i)$. The probability of the first hidden variable being in state i; often denoted $\pi \in \mathbb{R}^K$.
2. **Transition distribution**: $A_{ij} = p(z_{t+1} = j|z_t = i) \quad i \in \{1, ..., K\}$. The probability of moving from hidden state i to hidden state j; $A \in \mathbb{R}^{K \times K}$.
3. **Emission probability**: $p(x_t = j|z_t = i)$. The probability of an observed random variable x_t given the state of the hidden variable that "emitted" it. (Often think about x_t as discrete but all that follows also works in the continuous case)

HMMs: Objectives

We consider the following objectives:

1. Compute the probability of a latent sequence given an observation sequence. That is, we want to be able to compute $p(z_{1:t}|x_{1:t})$. This is achieved with the **Forward-Backward algorithm**.

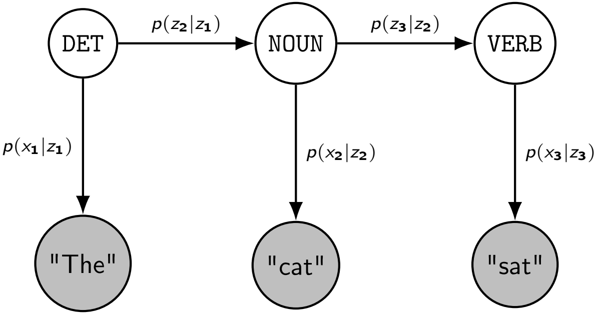
2. Infer the most likely sequence of hidden states. That is, we want to be able to compute

$$z^* = \underset{z_{\mathbf{1}:T}}{\operatorname{argmax}}\, p(z_{1:T}|x_{1:T}).$$

This is achieved using the **Viterbi algorithm**.

Example: Part-of-Speech Tagging

- **Goal**: Assign the correct grammatical tag (e.g., Noun, Verb) to each word in a sentence.
- **Hidden States (z_t)**: POS tags (e.g., `DET`, `NOUN`, `VERB`, `ADJ`).
- **Observations (x_t)**: The words in the sentence (e.g., "The", "cat", "sat").



- **Transition**: $p(\texttt{NOUN}|\texttt{DET})$ (Likelihood of Noun following Determinant).
- **Emission**: $p(\text{"cat"}|\texttt{NOUN})$ (Likelihood of "cat" being a Noun).

Forward algorithm

- The goal is to recursively compute the **filtered marginals**,

$$\alpha_t(j) = p(z_t = j | x_{1:t}).$$

- Assuming that we know the initial $p(z_1)$, transition $p(z_t|z_{t-1})$, and emission $p(x_t|z_t)$ probabilities for all $1 \leqslant t \leqslant T$.

- This is a step in the **forward-backward algorithm**.

Forward algorithm has two steps

- **Prediction step**: compute the one-step-ahead predictive density:

$$\begin{aligned}
p(z_t = j | x_{1:(t-1)}) &= \sum_{i=1}^{K} p(z_{t-1} = i, z_t = j | x_{1:(t-1)}) \\
&= \sum_{i=1}^{K} p(z_t = j | z_{t-1} = i, x_{1:(t-1)}) p(z_{t-1} = i | x_{1:(t-1)}) \\
&= \sum_{i=1}^{K} p(z_t = j | z_{t-1} = i) p(z_{t-1} = i | x_{1:(t-1)}) = \sum_{i=1}^{K} A_{ij} \alpha_{t-1}(i) = (A^\top \alpha_{t-1})_j
\end{aligned}$$

- **Update step**: Denoting $\lambda_t(j) = p(x_t | z_t = j)$ (here x_t is fixed and $\lambda_t \in \mathbb{R}^K$)

$$\begin{aligned}
\alpha_t(j) &= p(z_t = j | x_{1:t}) = p(z_t = j | x_{1:(t-1)}, x_t) \propto p(z_t = j, x_t | x_{1:(t-1)}) \\
&= p(x_t | z_t = j, x_{1:(t-1)}) p(z_t = j | x_{1:(t-1)}) \\
&= p(x_t | z_t = j) p(z_t = j | x_{1:(t-1)}) = \lambda_t(j) p(z_t = j | x_{1:(t-1)})
\end{aligned}$$

Using matrix notation: $\alpha_t \propto \lambda_t \odot (A^\top \alpha_{t-1})$ [$\odot$ is the Hadamard (entrywise) product]

Forward-Backward algorithm

This task of hidden state inference breaks down into the following:

- **Filtering**: compute posterior over current hidden state, $p(z_t|x_{1:t})$.
- **Prediction**: compute posterior over future hidden state, $p(z_{t+k}|x_{1:t})$.
- **Smoothing**: compute posterior over past hidden state, $p(z_t|x_{1:T}) \quad 1 \leqslant t < T$.

The probability of interest, $p(z_t|x_{1:T})$ is computed using a forward and backward recursion

- **Forward Recursion**: $\alpha_t(j) = p(z_t = j|x_{1:t})$ [this was computed earlier]
- **Backward Recursion**: $\beta_t(j) := p(x_{(t+1):T}|z_t = j)$

We assume that we know the initial $\pi(j) = p(z_1 = j)$, transition $A_{ij} = p(z_t = j|z_{t-1} = i)$, and emission $\lambda_t(j) = p(x_t|z_t = j)$ probabilities for all t.

Forward-Backward algorithm

We can break the chain into two parts, the past and the future, by conditioning on z_t:

- We have

$$\begin{aligned}
\gamma_t &:= p(z_t|x_{1:T}) \propto p(z_t, x_{1:T}) = p(z_t, x_{1:t}, x_{(t+1):T}) \\
&= p(z_t, x_{1:t})p(x_{(t+1):T}|z_t, x_{1:t}) = p(z_t, x_{1:t})p(x_{(t+1):T}|z_t) \\
&\propto p(z_t|x_{1:t})p(x_{(t+1):T}|z_t) \\
&= (\text{Forward Recursion})(\text{Backward Recursion})
\end{aligned}$$

- Here we use the conditional independence $x_{(t+1):T} \perp x_{1:t}|z_t$.
- We know how to perform forward recursion from the previous part.

Backward recursion

In the backward pass,

$$\begin{aligned}
\beta_t(i) = p(x_{(t+1):T}|z_t = i) &= \sum_{j=1}^{K} p(z_{t+1} = j, x_{t+1}, x_{(t+2):T}|z_t = i) \\
&= \sum_j p(x_{(t+2):T}|z_{t+1} = j, x_{t+1}, z_t = i)p(x_{t+1}|z_{t+1} = j, z_t = i)p(z_{t+1} = j|z_t = i) \\
&= \sum_j p(x_{(t+2):T}|z_{t+1} = j)p(x_{t+1}|z_{t+1} = j)p(z_{t+1} = j|z_t = i) \\
&= \sum_j \beta_{t+1}(j)\lambda_{t+1}(j)A_{ij}
\end{aligned}$$

In vector notation $\beta_t = A(\lambda_{t+1} \odot \beta_{t+1})$, where $\beta_T(i) = 1$.

Forward-backward algorithm $[\gamma_t(j) = p(z_t = j|x_{1:T})]$

Once we have the forward and the backward steps complete, we can compute $\gamma_t \propto \alpha_t \odot \beta_t$.

Viterbi algorithm

- The Viterbi algorithm (Viterbi 1967) is used to compute the most probable sequence.

$$\hat{z} = \arg\max_{z_{\mathbf{1}:T}} p(z_{1:T} | x_{1:T})$$

- Since this is MAP inference, we might think of replacing sum-operators with max-operators, just like we did in sum-product and max-product.

- Viterbi algorithm is a specialized version of max-product: the forward pass uses max-product, and the backward pass uses a traceback procedure to recover the most probable path.

Viterbi algorithm

- Define δ_t via

$$\delta_t(j) = \max_{z_1,\dots,z_{t-1}} p(z_{1:(t-1)}, z_t = j, x_{1:t})$$

which is the probability of ending up in state j at time t, by taking the most probable path.

- We notice that

$$\begin{aligned}
\delta_t(j) &= \max_{z_1,\dots,z_{t-1}} p(z_{1:(t-1)}, z_t = j, x_{1:(t-1)}, x_t) \\
&= \max_{z_1,\dots,z_{t-1}} p(z_{1:(t-1)}, x_{1:(t-1)}) p(z_t = j | z_{t-1}) p(x_t | z_t = j) \\
&= \max_i \max_{z_1,\dots,z_{t-2}} p(z_{1:(t-2)}, z_{t-1} = i, x_{1:(t-1)}) p(z_t = j | z_{t-1} = i) p(x_t | z_t = j) \\
&= \max_i \delta_{t-1}(i) A_{ij} \lambda_t(j)
\end{aligned}$$

- Keep track of the most likely previous state: $\theta_t(j) = \arg\max_i \delta_{t-1}(i) A_{ij} \lambda_t(j)$.

Viterbi algorithm

- Initialize the algorithm with

$$\delta_1(j) = p(z_1 = j, x_1) = \pi_j \lambda_1(j).$$

- and terminate with

$$z_T^* = \arg\max_i \delta_T(i)$$

- Then, we compute the most probable sequence of states using traceback:

$$z_t^* = \theta_{t+1}(z_{t+1}^*)$$

Summary: HMMs

- HMMs hide the temporal dependence by keeping it in the unobserved state.
- No assumptions on the temporal dependence of observations is made.
- Forward-backward algorithm can be used to find "beliefs".
- Viterbi algorithm can be used to do MAP.

Monte-Carlo Methods

Sampling

- A sample from a distribution $p(x)$ is a single realization x whose probability distribution is $p(x)$. Here, x can be high-dimensional.

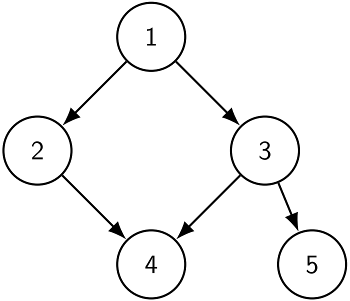
- Assumption: The density from which we sample, $p(x)$, can be evaluated to within a multiplicative constant. That is, we have $\tilde{p}(x)$ such that

$$p(x) = \frac{\tilde{p}(x)}{Z}.$$

Warm up: Ancestral Sampling

- Given a DAGM, and the ability to sample from each of its factors given its parents, we can sample from the joint distribution over all the nodes by **ancestral sampling**.
- Start with nodes that have no parents. Sample them from the corresponding marginal distributions.
- At each step, sample from any conditional distribution that you haven't visited yet, whose parents have all been sampled.

Ancestral Sampling: example



- The distribution graph factorizes according to the DAG

$$
\begin{aligned}
p(x_1, \ldots, x_5) &= \prod_{i}^{5} p(x_i | \text{parents}(x_i)) \\
&= p(x_1) p(x_2|x_1) p(x_3|x_1) p(x_4|x_2, x_3) p(x_5|x_3)
\end{aligned}
$$

- Start by sampling from $p(x_1)$.
- Then sample from $p(x_2|x_1)$ and $p(x_3|x_1)$.
- Then sample from $p(x_4|x_2, x_3)$.
- Finally, sample from $p(x_5|x_3)$.

Main objectives of sampling

Use Monte Carlo methods to solve one or both of the following problems.

- **Problem 1**: Generate samples $\{x^{(r)}\}_{r=1}^{R}$ from $p(x)$.
- **Problem 2**: To estimate expectations of functions, $\phi(x)$, under this distribution $p(x)$

$$\Phi = \mathbb{E}_{x \sim p(x)}[\phi(x)] = \int \phi(x) p(x) dx$$

The function ϕ is called a *test function*.

Example

Examples of test functions $\phi(x)$:

- the **mean** of a function $f(x)$ under $p(x)$ by finding the expectation of the function $\phi_1(x) = f(x)$.
- the **variance** of f under $p(x)$ by finding the expectations of the functions $\phi_1(x) = f(x)$ and $\phi_2(x) = f(x)^2$

$$\begin{aligned}
\phi_1(x) = f(x) &\Rightarrow \Phi_1 = \mathbb{E}_{x \sim p(x)}[\phi_1(x)] \\
\phi_2(x) = f(x)^2 &\Rightarrow \Phi_2 = \mathbb{E}_{x \sim p(x)}[\phi_2(x)] \\
&\Rightarrow \mathrm{var}(f(x)) = \Phi_2 - (\Phi_1)^2
\end{aligned}$$

Estimation problem

We start with the estimation problem using simple Monte Carlo:

- **Simple Monte Carlo**: Given $\{x^{(r)}\}_{r=1}^{R} \sim p(x)$ we can estimate the expectation $\mathbb{E}_{x \sim p(x)}[\phi(x)]$ using the estimator $\hat{\Phi}$:

$$\Phi := \mathbb{E}_{x \sim p(x)}[\phi(x)] \approx \frac{1}{R} \sum_{r=1}^{R} \phi(x^{(r)}) := \hat{\Phi}$$

- The fact that $\hat{\Phi}$ is a consistent estimator of Φ follows from the Law of Large Numbers (LLN).

Basic properties of Monte Carlo estimation

- **Unbiasedness**: If the vectors $\{x^{(r)}\}_{r=1}^{R}$ are all generated (independently or not) from $p(x)$, then the expectation of $\hat{\Phi}$ is Φ. Indeed,

$$\begin{aligned}
\mathbb{E}[\hat{\Phi}] &= \mathbb{E}\left[\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\right] = \frac{1}{R}\sum_{r=1}^{R}\mathbb{E}[\phi(x^{(r)})] \\
&= \frac{1}{R}\sum_{r=1}^{R}\mathbb{E}_{x\sim p(x)}[\phi(x)] = \frac{R}{R}\mathbb{E}_{x\sim p(x)}[\phi(x)] \\
&= \Phi
\end{aligned}$$

Simple properties of Monte Carlo estimation

- **Variance**: As the number of samples of R increases, the variance of $\hat{\Phi}$ will decrease with rate $\frac{1}{R}$ if the samples are independent

$$\begin{aligned}\mathrm{var}[\hat{\Phi}] &= \mathrm{var}\left[\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\right] = \frac{1}{R^2}\mathrm{var}\left[\sum_{r=1}^{R}\phi(x^{(r)})\right] \\ &= \frac{1}{R^2}\sum_{r=1}^{R}\mathrm{var}\left[\phi(x^{(r)})\right] = \frac{R}{R^2}\mathrm{var}[\phi(x)] = \frac{1}{R}\mathrm{var}[\phi(x)]\end{aligned}$$

Accuracy of the Monte Carlo estimate depends on R and on the variance of ϕ.

Normalizing constant

- Assume we know the density $p(x)$ up to a multiplicative constant

$$p(x) = \frac{\tilde{p}(x)}{Z}$$

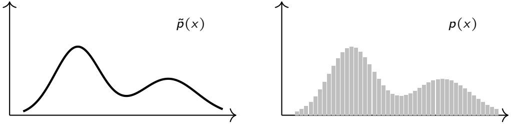
- There are two difficulties:
  - We do not generally know the normalizing constant, Z. Computing

    $$Z = \int \tilde{p}(x) dx$$

    requires a high-dimensional integral or sum.
  - Even if we did know Z, the problem of drawing samples from $p(x)$ is still a challenging one, especially in high-dimensional spaces.

Bad Idea: Lattice Discretization

Suppose we want to sample from $p(x)$ for which $\tilde{p}(x)$ is given in figure (a).

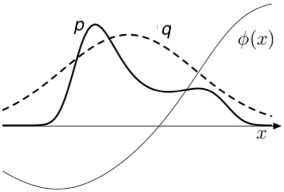


(a) (b)

- How to compute Z?
- We could discretize the variable x and sample from the discrete distribution.
- In figure (b) there are 50 uniformly spaced points in one dimension. If our system had, $D = 1000$ dimensions say, then the corresponding number of points would be $50^D = 50^{1000}$. Thus, the cost is exponential in dimension!

Estimation tool: Importance Sampling

Importance sampling: to estimate the expectation of a function $\phi(x)$.



- The density from which we wish to draw samples can be evaluated up to normalizing constant. As before, we have $p(x) = \tilde{p}(x)/Z$.
- There is a simpler density, $q(x)$ from which it is easy to sample from and easy to evaluate up to normalizing constant (i.e. $\tilde{q}(x)$)

$$q(x) = \frac{\tilde{q}(x)}{Z_q}$$

Estimation tool: Importance Sampling

- In importance sampling, we generate R samples from $q(x)$

$$\{x^{(r)}\}_{r=1}^{R} \sim q(x)$$

- If these points were samples from $p(x)$ then we could estimate Φ by

$$\Phi = \mathbb{E}_{x\sim p(x)}[\phi(x)] \approx \frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)}) = \hat{\Phi}$$

That is, we could use a simple Monte Carlo estimator.

- But we sampled from q. We need to correct this!
- Values of x where $q(x)$ is greater than $p(x)$ will be over-represented in this estimator, and points where $q(x)$ is less than $p(x)$ will be under-represented. Thus, we introduce weights.

Estimation tool: Importance Sampling

- Introduce weights: $\tilde{w}_r = \frac{\tilde{p}(x^{(r)})}{\tilde{q}(x^{(r)})} = \frac{Z_p p(x^{(r)})}{Z_q q(x^{(r)})}$ and notice that

$$\frac{1}{R}\sum_{r=1}^{R} \tilde{w}_r \approx \mathbb{E}_{x\sim q(x)}\left[\frac{\tilde{p}(x)}{\tilde{q}(x)}\right] = \frac{Z_p}{Z_q}\int \frac{p(x)}{q(x)} q(x)dx = \frac{Z_p}{Z_q}$$

- Finally, we rewrite our estimator under q

$$\Phi = \int \phi(x)p(x)dx = \int \phi(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\frac{p(x^{(r)})}{q(x^{(r)})} = (*)$$

- However, the estimator relies on p. It can only rely on $\tilde{p}$ and $\tilde{q}$.

$$(*) = \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\frac{\tilde{p}(x^{(r)})}{\tilde{q}(x^{(r)})} = \frac{Z_q}{Z_p}\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\tilde{w}_r$$

$$\approx \frac{\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\tilde{w}_r}{\frac{1}{R}\sum_{r=1}^{R}\tilde{w}_r} = \sum_{r=1}^{R}\phi(x^{(r)})\cdot w_r = \hat{\Phi}_{iw}$$

where $w_r = \frac{\tilde{w}_r}{\sum_{r=1}^{R}\tilde{w}_r}$ and $\hat{\Phi}_{iw}$ is our importance weighted estimator.

Rejection Sampling: Intuition

Key Intuition:
To sample uniformly from a complex domain D, we can sample uniformly from a larger, simpler domain C and keep only the realizations that fall within D.

Example: Unit Disk

- **Target** D**:** Unit disk $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leqslant 1\}$.
- **Proposal** C**:** Square $[-1, 1] \times [-1, 1]$.

Procedure:

1. Sample $(U, V) \sim \mathcal{U}([-1, 1]^2)$.
2. **Accept** if $X^2 + Y^2 \leqslant 1$ (Blue).
3. **Reject** otherwise (Gray).

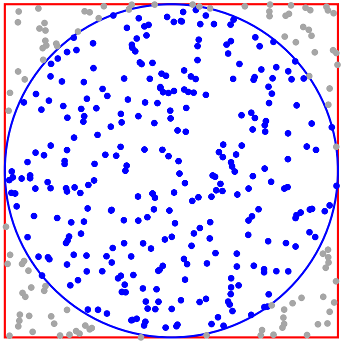


Illustration of rejection sampling for the unit disk.

Sampling tool: Rejection sampling

- We want expectations under $p(x) = \frac{p^*(x)}{Z_p}$.

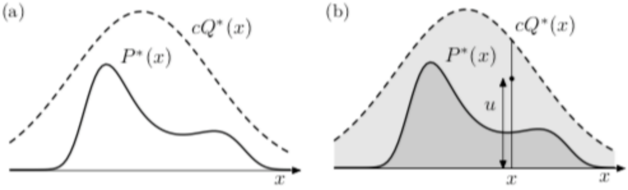
- Assume that we have a simpler proposal density $q(x)$ which we can evaluate (within a multiplicative factor Z_q, as before), and from which we can generate samples, i.e.

$$q^*(x) = Z_q \cdot q(x)$$

- Further assume that we know the value of a constant c such that

$$\forall x, \qquad cq^*(x) > p^*(x)$$

Sampling tool: Rejection sampling



The procedure is as follows:

1. Generate two random numbers.
   1. x is generated from $q(x)$.
   2. u is generated uniformly from the interval $[0, cq^*(x)]$.
2. Accept or reject the sample x by comparing the value of u with $p^*(x)$
   1. If $u > p^*(x)$, then x is rejected.
   2. Otherwise x is accepted; x is added to our set of samples $\{x^{(r)}\}$.

Why does rejection sampling work?

(i) $x \sim q(x)$, (ii) $u|x \sim \mathrm{Unif}[0, cq^*(x)]$, (iii) accept x if $u \leqslant p^*(x)$.

- Note: $\mathbb{P}(u \leqslant p^*(x)|x) = \frac{p^*(x)}{cq^*(x)}$ (remember we assume $p^*(x) < cq^*(x)$).
- $\forall A \subseteq \mathcal{X}: \quad \mathbb{P}_{x\sim p}(x \in A) = \int_A p(x)dx = \int \mathbf{1}_{\{x\in A\}} p(x)dx = \mathbb{E}_{x\sim p}[\mathbf{1}_{\{x\in A\}}]$.
- Law of total expectation $\mathbb{E}[\mathbb{E}[Z|\mathcal{H}]] = \mathbb{E}Z$

This gives:

$$\begin{aligned}
\mathbb{P}_{x\sim q}(x \in A | u \leqslant p^*(x)) &= \frac{\mathbb{P}_{x\sim q}(x \in A, u \leqslant p^*(x))}{\mathbb{E}_{x\sim q}[\mathbb{P}(u \leqslant p^*(x)|x)]} \\
&= \mathbb{E}_{x\sim q}\left[\mathbf{1}_{\{x\in A\}}\mathbb{P}(u \leqslant p^*(x)|x)\right] / \mathbb{E}_{x\sim q}\left[\frac{p^*(x)}{cq^*(x)}\right] \\
&= \mathbb{E}_{x\sim q}\left[\mathbf{1}_{\{x\in A\}}\frac{p^*(x)}{cq^*(x)}\right] \Big/ \frac{Z_p}{cZ_q} = \mathbb{P}_{x\sim p}(x \in A)\frac{Z_p}{cZ_q}\Big/\frac{Z_p}{cZ_q} \\
&= \mathbb{P}_{x\sim p}(x \in A).
\end{aligned}$$

Rejection sampling in many dimensions

- In high-dimensional problems, the requirement that $cq^*(x) \geqslant p^*(x)$ will force c to be huge, so acceptances will be very rare.

- Finding such a value of c may be difficult too, since we don't know where the modes of p^* are located nor how high they are.

- In general c grows exponentially with the dimensionality, so the acceptance rate is expected to be exponentially small in dimension

$$\text{acceptance rate} = \frac{\text{area under } p^*}{\text{area under } cp^*} = \frac{Z_p}{cZ_q}$$

Summary

- Estimating expectations is an important problem, which is in general hard. We learned 3 sampling-based tools for this task:
  - Simple Monte Carlo
  - Importance Sampling
  - Rejection Sampling

- Next lecture, we will learn more refined techniques.