STA 414/2104:
Probabilistic Learning and Reasoning

Week 4: Message Passing / Monte Carlo Methods

Piotr Zwiernik

University of Toronto

Overview

1. TrueSkill latent variable models
2. Message passing
   - Sum-product algorithm
   - Loopy Belief Propagation
3. Monte Carlo Methods
   - Ancestral sampling
   - Basic Monte Carlo
   - Importance sampling

TrueSkill latent variable models

Latent variables

- What to do when a variable z is unobserved?
- If we never condition on z when in the inference problem, then we can just integrate it out.
- However, in certain cases, we are interested in the latent variables themselves, e.g. the clustering problems.
- More on latent variables when we cover Gaussian mixtures.

The TrueSkill latent variable model

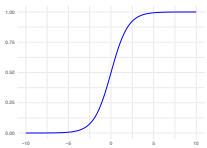
- **TrueSkill** model is a player ranking system for competitive games.
- The goal is to infer the skill of players in a competitive game, based on observing who beats who.
- In the TrueSkill model, each player has a fixed level of skill, denoted z_i.
- We initially don't know anything about anyone's skill, but we assume everyone's skill is independent (e.g. an independent Gaussian prior).
- We never get to observe the players' skills directly, which makes this a latent variable model.

TrueSkill model

- We observe the outcome of a series of matches between different players.
- For each game, the probability that player i beats player j is given by

$$p(i \text{ beats } j) = \sigma(z_i - z_j)$$

where sigma is the logistic function: $\sigma(y) = \frac{1}{1+\exp(-y)}$.



- We can write the entire joint likelihood of a set of players and games as:

$$p(z_1, z_2, \ldots z_N, \text{game 1, game 2, .. game T})$$
$$= \left[\prod_{i=1}^{N} p(z_i)\right] \left[\prod_{\text{games}} p(\text{i beats j}|z_i, z_j)\right]$$

Posterior

- Given the outcome of some matches, the players' skills are no longer independent, even if they've never played each other.
- Computing the posterior over even two players' skills requires integrating over all the other players' skills:

$$p(z_1, z_2 | \text{game 1, game 2, ... game T})$$
$$= \int \cdots \int p(z_1, z_2, z_3 \ldots z_N | x) dz_3 \ldots dz_N$$

- **Message passing** can be used to compute posteriors!
- More on this model in Assignment 2.

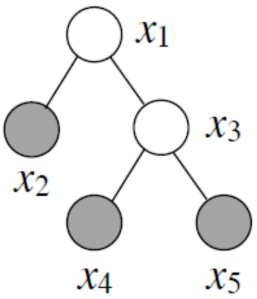
Message passing

Variable Elimination Order and Trees

Last week: we can do exact inference by variable elimination: i.e. to compute $p(x_F|x_E)$, we can marginalize $p(x_F, x_R|x_E)$ over every variable in x_R.

- The computational cost depends on the graph, and the elimination ordering.
- Determining the optimal elimination ordering is hard.
- The resulting marginalization might be still be unreasonably costly.
- For trees any elimination ordering that goes from the leaves inwards towards any root will be optimal.

Inference in Trees (graphs with no cycles)

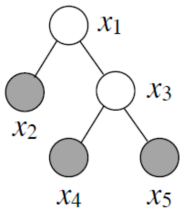


- A graph is $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices (nodes) and $\mathcal{E}$ the set of edges; $\mathcal{V} = \{1, \ldots, n\}$.
- For $i, j \in \mathcal{V}$, we have $(i, j) \in \mathcal{E}$ if there is an edge between the nodes i and j.
- For a node in graph $i \in \mathcal{V}$, $N(i)$ denotes the neighbors of i, i.e. $N(i) = \{j : (i, j) \in \mathcal{E}\}$.
- The nodes in x_E are shaded.

The joint distribution in the corresponding MRF is

$$p(x_1, x_2, \ldots, x_n) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \psi(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j).$$

Example: Inference in Trees



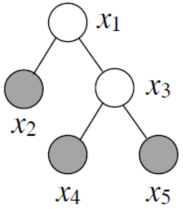
- The joint distribution is $p(x) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \psi(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j)$.
- Want to compute $p(x_3|x_E)$, $\quad x_E = (\bar{x}_2, \bar{x}_4, \bar{x}_5)$, $x_R = x_1$.
- We have $p(x_3|x_E) \propto p(x_3, x_E)$.

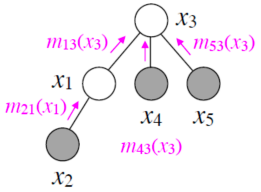
(meaning that $p(x_3|x_E) = \frac{p(x_3,x_E)}{\sum_{x_3'} p(x_3',x_E)}$, $Z^E = \sum_{x_3'} p(x_3', x_E)$)

$$p(x_3|x_E) = \frac{1}{Z^E} \sum_{x_1} \psi_1(x_1)\psi_2(\bar{x}_2)\psi_3(x_3)\psi_4(\bar{x}_4)\psi_5(\bar{x}_5)\psi_{12}(x_1, \bar{x}_2)\psi_{13}(x_1, x_3)\psi_{34}(x_3, \bar{x}_4)\psi_{35}(x_3, \bar{x}_5).$$

We write the variable elimination algorithm revealing additional structure.

Inference in Trees





$$
\begin{aligned}
p(x_3|x_E) &= \frac{1}{Z^E}\sum_{x_1}\psi_1(x_1)\psi_2(\bar{x}_2)\psi_3(x_3)\psi_4(\bar{x}_4)\psi_5(\bar{x}_5)\psi_{12}(x_1,\bar{x}_2)\psi_{13}(x_1,x_3)\psi_{34}(x_3,\bar{x}_4)\psi_{35}(x_3,\bar{x}_5) \\
&= \frac{1}{Z^E}\underbrace{\psi_4(\bar{x}_4)\psi_{34}(x_3,\bar{x}_4)}_{m_{43}(x_3)}\underbrace{\psi_5(\bar{x}_5)\psi_{35}(x_3,\bar{x}_5)}_{m_{53}(x_3)}\psi_3(x_3)\sum_{x_1}\psi_1(x_1)\psi_{13}(x_1,x_3)\underbrace{\psi_2(\bar{x}_2)\psi_{12}(x_1,\bar{x}_2)}_{m_{21}(x_1)} \\
&= \frac{1}{Z^E}m_{43}(x_3)m_{53}(x_3)\psi_3(x_3)\underbrace{\sum_{x_1}\psi_1(x_1)\psi_{13}(x_1,x_3)m_{21}(x_1)}_{m_{13}(x_3)} \\
&= \frac{1}{Z^E}\psi_3(x_3)m_{43}(x_3)m_{53}(x_3)m_{13}(x_3) = \frac{\psi_3(x_3)m_{43}(x_3)m_{53}(x_3)m_{13}(x_3)}{\sum_{x_3'}\psi_3(x_3')m_{43}(x_3')m_{53}(x_3')m_{13}(x_3')}
\end{aligned}
$$

Sum-product algorithm

Perform variable elimination from leaves to root. Belief propagation is a message-passing between neighboring vertices of the graph.

- If x_j unobserved, the message sent from variable j to $i \in N(j)$ is

$$m_{j \to i}(x_i) = \sum_{x_j} \psi_j(x_j) \psi_{ij}(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{k \to j}(x_j)$$

- If x_j is observed, the message is

$$m_{j \to i}(x_i) = \psi_j(\bar{x}_j) \psi_{ij}(x_i, \bar{x}_j) \prod_{k \in N(j) \setminus i} m_{k \to j}(\bar{x}_j)$$

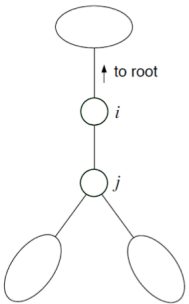
- Once the message passing stage is complete, we can compute our beliefs as

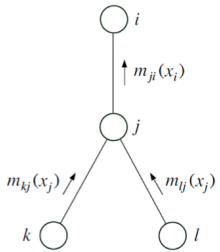
$$b(x_i) \;=\; p(x_i | x_E) \;\propto\; \psi_i(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i).$$

Message Passing on Trees

The message sent from variable j to $i \in N(j)$ is

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\setminus i} m_{k\to j}(x_j)$$





Each message $m_{j\to i}(x_i)$ is a vector with one value for each state of x_i.

Belief Propagation on Trees

Belief Propagation Algorithm on Trees

Step 1 Choose root r arbitrarily

Step 2 Pass messages from leafs to r

Step 3 Pass messages from r to leafs

Step 4 Compute beliefs (marginals)

These two passes are sufficient on trees!

$\forall (i,j)$ compute $m_{i\to j}(x_j)$ and $m_{j\to i}(x_i)$.

$$b(x_i) = p(x_i|x_E) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i),\ \forall_i$$

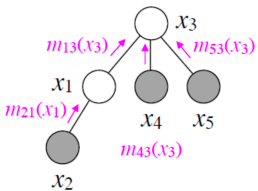
One can compute them in two steps:

- Compute unnormalized beliefs $\tilde{b}(x_i) = \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i)$
- Normalize them $b(x_i) = \tilde{b}(x_i) / \sum_{x_i'} \tilde{b}(x_i')$.

Inference in Trees: Compute $p(x_3|\bar{x}_2, \bar{x}_4, \bar{x}_5)$ and $p(x_1|\bar{x}_2, \bar{x}_4, \bar{x}_5)$

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\setminus i} m_{k\to j}(x_j)$$

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$



- $m_{5\to 3}(x_3) = \psi_5(\bar{x}_5)\psi_{35}(x_3, \bar{x}_5)$
- $m_{2\to 1}(x_1) = \psi_2(\bar{x}_2)\psi_{12}(x_1, \bar{x}_2)$ x_2, x_4, x_5 are observed
- $m_{4\to 3}(x_3) = \psi_4(\bar{x}_4)\psi_{34}(x_3, \bar{x}_4)$
- $m_{1\to 3}(x_3) = \sum_{x_1} \psi_1(x_1)\psi_{13}(x_1, x_3)m_{2\to 1}(x_1)$
- $m_{3\to 1}(x_1) = \sum_{x_3} \psi_3(x_3)\psi_{13}(x_1, x_3)m_{4\to 3}(x_3)m_{5\to 3}(x_3)$
- $b(x_1) \propto \psi_1(x_1)m_{2\to 1}(x_1)m_{3\to 1}(x_1)$
- $b(x_3) \propto \psi_3(x_3)m_{1\to 3}(x_3)m_{4\to 3}(x_3)m_{5\to 3}(x_3)$

Loopy Belief Propagation

- What if the graph (MRF) is not a tree? (e.g. TrueSkill model)
- Keep passing messages until convergence.
- This is called **Loopy Belief Propagation**.
- This is like when someone starts a rumour and then hears the same rumour from someone else, making them more certain it's true.
- We won't get the exact marginals, but an approximation.
- But turns out it is still very useful!

Although these ideas are general, we focus on the pairwise graphical models.

Loopy Belief Propagation

- Initialize all messages uniformly:

$$m_{i \to j}(x_j) \;=\; (1/k, \ldots, 1/k)$$

  where k is the number of states x_j can take.
- Keep running BP updates until it "converges":

$$m_{j \to i}(x_i) = \sum_{x_j} \psi_j(x_j) \psi_{ij}(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{k \to j}(x_j)$$

  and (sometimes) normalized for stability.
- It will generally not converge, but often works fine.
- Compute beliefs $b(x_i) \propto \psi_i(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i)$.

With no theoretical guarantees, this algorithm is still very useful in practice.

Max-product algorithm

- MAP inference: Suppose that instead of marginalizing out x_R we are interested in the most likely configuration $\hat{x} = \arg\max p(x)$.
- For MAP inference, we maximize over x_j instead of summing over them. This is called **max-product BP** with updates

$$m_{j\to i}(x_i) = \max_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\setminus i} m_{k\to j}(x_j)$$

- After BP algorithm converges, the beliefs are **max-marginals**

$$\hat{b}(x_i) = \max_{x_{\setminus i}} p(x_i, x_{\setminus i}) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$

- MAP inference: take $\hat{x}_i := \arg\max_{x_i} \hat{b}(x_i)$ for all $i \notin E$.

Summary

- Loopy Belief Propagation is very useful in practice, without much theoretical guarantee (other than trees).
  - It multiplies the same potentials multiple times. It is often over-confident.
  - It can oscillate, but this is generally ok.
  - Often works better if we normalize messages, and use momentum in the updates.
- The algorithm we learned is called **sum-product BP**. If we are interested in MAP inference, we can maximize over x_j instead of summing over them. This is called **max-product BP**.

Monte Carlo Methods

Overview

- Ancestral Sampling
- Simple Monte Carlo
- Importance Sampling
- Rejection Sampling

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

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

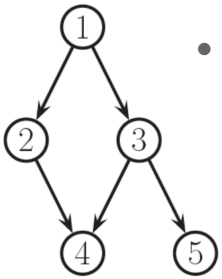
- e.g. consider an Ising model with fixed values for its parameters

$$p(x) \propto \tilde{p}(x) = \exp\left\{\sum_i b_i x_i + \sum_{i<j} J_{ij} x_i x_j\right\}$$

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

$$p(x_{1,\ldots,5}) = \prod_{i}^{5} p(x_i|\text{parents}(x_i))$$
$$= p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2,x_3)p(x_5|x_3)$$

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

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

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 $\phi_2(x) = f(x)^2$

$$\phi_1(x) = f(x) \Rightarrow \Phi_1 = \mathop{\mathbb{E}}_{x \sim p(x)}[\phi_1(x)]$$
$$\phi_2(x) = f(x)^2 \Rightarrow \Phi_2 = \mathop{\mathbb{E}}_{x \sim p(x)}[\phi_2(x)]$$
$$\Rightarrow \text{var}(f(x)) = \Phi_2 - (\Phi_1)^2$$

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 $\mathop{\mathbb{E}}_{x\sim p(x)}[\phi(x)]$ using the estimator $\hat{\Phi}$:

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

- 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 generated independently from $p(x)$, then the expectation of $\hat{\Phi}$ is Φ. Indeed,

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

Simple properties of Monte Carlo estimation

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

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

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

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

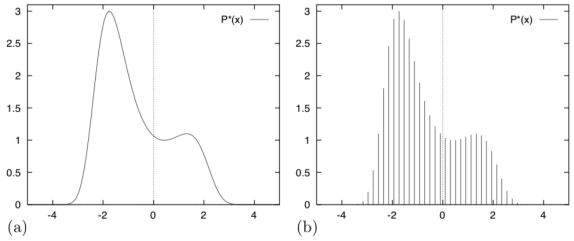
- There are two difficulties:
  - We do not generally know the normalizing constant, Z. Computing

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

    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

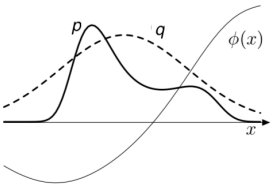
Suppose we want to sample from $p(x)$ for which $\tilde{p}(x)$ is given in figure (a).



- 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

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

  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.

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

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

- Finally, we rewrite our estimator under q

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

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

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

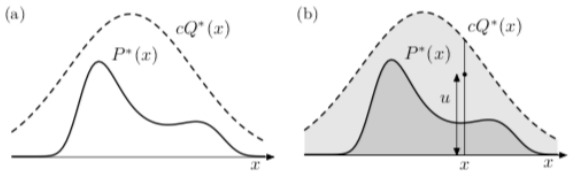
$$\approx \frac{\frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)})\cdot\tilde{w}_r}{\frac{1}{R}\sum_{r=1}^{R}\tilde{w}_r} = \sum_{r=1}^{R}\phi(x^{(r)})\cdot w_r = \hat{\Phi}_{iw}$$

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

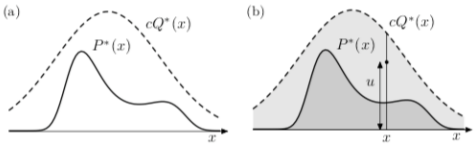
Sampling tool: Rejection sampling

- We want expectations under $p(x) = \tilde{p}(x)/Z$.
- 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. $\tilde{q}(x) = Z_q \cdot q(x)$.
- Further assume that we know the value of a constant c such that

$$c\tilde{q}(x) > \tilde{p}(x) \quad \forall x$$



Sampling tool: Rejection sampling



The procedure is as follows:

1. Generate two random numbers.
   1.1 x is generated from $q(x)$.
   1.2 u is generated uniformly from the interval $[0, c\tilde{q}(x)]$ (see figure (b) above: book's notation $P^* = \tilde{p}$, $Q^* = \tilde{q}$).
2. Accept or reject the sample x by comparing the value of u with $\tilde{p}(x)$
   2.1 If $u > \tilde{p}(x)$, then x is rejected
   2.2 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) $u|x \sim \text{Unif}[0, c\tilde{q}(x)]$, (iii) accept x if $u \leq \tilde{p}(x)$.

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

This gives:

$$\begin{aligned}
\mathbb{P}_{x\sim q}\big(x \in A | u \leq \tilde{p}(x)\big) =& \mathbb{P}_{x\sim q}(x \in A, u \leq \tilde{p}(x))/\mathbb{E}_{x\sim q}[\mathbb{P}(u \leq \tilde{p}(x)|x)] \\
=& \mathbb{E}_{x\sim q}[\mathbf{1}_{\{x\in A\}}\mathbb{P}(u \leq \tilde{p}(x)|x)]/\mathbb{E}_{x\sim q}[\frac{\tilde{p}(x)}{c\tilde{q}(x)}] \\
=& \mathbb{E}_{x\sim q}[\mathbf{1}_{\{x\in A\}}\frac{\tilde{p}(x)}{c\tilde{q}(x)}]/\frac{Z_p}{cZ_q} = \mathbb{P}_{x\sim p}(x \in A)\frac{Z_p}{cZ_q}/\frac{Z_p}{cZ_q} \\
=& \mathbb{P}_{x\sim p}(x \in A)
\end{aligned}$$

Rejection sampling in many dimensions

- In high-dimensional problems, the requirement that $c\tilde{q}(x) \geq \tilde{p}(x)$ 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 $\tilde{p}$ 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

$$\text{acceptance rate} = \frac{\text{area under } \tilde{p}}{\text{area under } c\tilde{q}} = \frac{Z_p}{cZ_q}$$

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.