Graphical Models in Computer Vision

Andreas Geiger

Max Planck Institute for Intelligent Systems
Perceiving Systems

May 2, 2016
## Syllabus

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.04.2016</td>
<td>Introduction</td>
</tr>
<tr>
<td>18.04.2016</td>
<td>Graphical Models 1</td>
</tr>
<tr>
<td>25.04.2016</td>
<td>Graphical Models 2 (Sand 6/7)</td>
</tr>
<tr>
<td>02.05.2016</td>
<td>Graphical Models 3</td>
</tr>
<tr>
<td>09.05.2016</td>
<td>Graphical Models 4</td>
</tr>
<tr>
<td>23.05.2016</td>
<td>Body Models 1</td>
</tr>
<tr>
<td>30.05.2016</td>
<td>Body Models 2</td>
</tr>
<tr>
<td>06.06.2016</td>
<td>Body Models 3</td>
</tr>
<tr>
<td>13.06.2016</td>
<td>Body Models 4</td>
</tr>
<tr>
<td>20.06.2016</td>
<td>Stereo</td>
</tr>
<tr>
<td>27.06.2016</td>
<td>Optical Flow</td>
</tr>
<tr>
<td>04.07.2016</td>
<td>Segmentation</td>
</tr>
<tr>
<td>11.07.2016</td>
<td>Object Detection 1</td>
</tr>
<tr>
<td>18.07.2016</td>
<td>Object Detection 2</td>
</tr>
</tbody>
</table>
Todays topic

- Recap
  - Belief Networks
  - Markov Networks & Markov Random Fields
  - Filter View
  - Factor Graphs
  - Belief Propagation on Trees

- Approximate Inference
  - Loopy Belief Propagation on General Graphs
  - Sampling
Belief Networks

A belief network is a distribution of the form

\[ p(x_1, \ldots, x_D) = \prod_{i=1}^{D} p(x_i \mid pa(x_i)) \]

where \( pa(x) \) denotes the parental variables of \( x \)
Markov Networks & Markov Random Fields

Markov Network

For a set of variables $\mathcal{X} = \{x_1, \ldots, x_D\}$ a Markov network is defined as a product of potentials over the maximal cliques $\mathcal{X}_c$ of the graph $\mathcal{G}$

$$p(x_1, \ldots, x_D) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(\mathcal{X}_c)$$

$$p(a, b, c) = \frac{1}{Z} \phi_{ac}(a, c)\phi_{bc}(b, c)$$
Filter View

- Each graph describes a family of probability distributions
- Extremes:
  - Fully connected, no constraints, all $p$ pass
  - no connections, only product of marginals may pass
Factor Graphs

- Now consider we introduce an extra node (a square) for each factor:

(a) Markov Network
(b) Factor graph representation of \( \phi(a, b, c) \)
(c) Factor graph representation of \( \phi(a, b)\phi(b, c)\phi(c, a) \)

- Both factor graphs have the same Markov network \((b, c) \Rightarrow (a)\)
Factor Graphs

**Factor Graph**

Given a function

\[ f(x_1, \ldots, x_n) = \prod_i \psi_i(x_i) \]

the factor graph (FG) has a node (represented by a square) for each factor \( \psi_i(x_i) \) and a variable node (represented by a circle) for each variable \( x_j \).

When used to represent a distribution

\[ p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_i \psi_i(x_i) \]

a normalization constant \( Z \) is assumed.
Belief Propagation on a Chain

\[ p(a, b, c, d) = f_1(a, b)f_2(b, c)f_3(c, d)f_4(d) \]

\[ p(a, b, c) = \sum_d p(a, b, c, d) \]

\[ = f_1(a, b)f_2(b, c) \sum_d f_3(c, d)f_4(d) \]

\[ = f_1(a, b)f_2(b, c) \sum_c f_3(c, d)f_4(d) \]

\[ p(a, b) = \sum_c p(a, b, c) = f_1(a, b) \sum_c f_2(b, c)\mu_{d\rightarrow c}(c) \]

\[ = f_1(a, b) \sum_c f_2(b, c)\mu_{d\rightarrow c}(c) \]

\[ = f_1(a, b) \sum_c f_2(b, c)\mu_{c\rightarrow b}(b) \]
Belief Propagation on a Tree

- Idea: compute messages
Belief Propagation: Finding **Marginals**

**Sum-Product Algorithm for Trees**

1. Initialize messages
2. Iterate from leaves of the tree to target variable:
   - Factor-to-variable messages ("sum-product")
     \[
     \mu_{f \rightarrow x}(x) = \sum_{x' \in \mathcal{X}_f \setminus x} \phi_f(x') \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y)
     \]
   - Variable-to-factor messages (at target ⇒ marginal!)
     \[
     \mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x)
     \]

- $\mathcal{X}_f$: Variables that connect to factor $f$
- $\text{ne}(x)$: Factors that connect to variable $x$
- If all marginals are desired: 1) leaves $\rightarrow$ root 2) root $\rightarrow$ leaves
Belief Propagation: Find **Most Likely State (MAP)**

**Max-Product Algorithm for Trees**

1. Initialize messages
2. Iterate from leaves of the tree to target variable:
   - Factor-to-variable messages ("max-product")
     \[ \mu_{f \rightarrow x}(x) = \max_{X_f \setminus x} \phi_f(X_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \rightarrow f}(y) \]
   - Variable-to-factor messages (at target \( \Rightarrow \) **most likely state**)!
     \[ \mu_{x \rightarrow f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \rightarrow x}(x) \]

- \( X_f \): Variables that connect to factor \( f \)
- \( ne(x) \): Factors that connect to variable \( x \)
- If all states are of interest: 1) leaves \( \rightarrow \) root  2) root \( \rightarrow \) leaves
Fantastic, this is all very nice!

BUT ...
What if the graph is not singly connected?

\[ p(a, b, c, d) = f_1(a, b)f_2(b, c)f_3(c, d)f_4(d, a) \]
What if the graph is not singly connected?

\[
p(a, b, c, d) = f_1(a, b)f_2(b, c)f_3(c, d)f_4(d, a)
\]

\[
p(a, b, c) = \sum_d p(a, b, c, d) = f_1(a, b)f_2(b, c) \sum_d f_3(c, d)f_4(d, a)
\]

\[
p(a, b) = \sum_c p(a, b, c) = f_1(a, b) \sum_c f_2(b, c) \mu_{d \rightarrow a,c}(a, c)
\]

\[
p(a) = \sum_b p(a, b) = \sum_b f_1(a, b) \mu_{c \rightarrow a,b}(a, b)
\]

2D messages now ⇒ simply buy more RAM and wait a bit longer?
What if the graph gets bigger?

\[ p(\text{all}) = f_1(a, b)f_2(b, c)f_3(a, d)f_4(b, e)f_5(c, g)f_6(d, e) \]
\[ f_7(e, g)f_8(d, h)f_9(e, i)f_{10}(g, j)f_{11}(h, i)f_{12}(i, j) \]
What if the graph gets bigger?

\[ p(\text{all}) = f_1(a, b)f_2(b, c)f_3(a, d)f_4(b, e)f_5(c, g)f_6(d, e) \]
\[ f_7(e, g)f_8(d, h)f_9(e, i)f_{10}(g, j)f_{11}(h, i)f_{12}(i, j) \]

\[ p(\text{all}\setminus\{j\}) = f_1(a, b)f_2(b, c)f_3(a, d)f_4(b, e)f_5(c, g)f_6(d, e) \]
\[ f_7(e, g)f_8(d, h)f_9(e, i)f_{11}(h, i)\mu_j\rightarrow i, g(i, g) \]

\[ p(\text{all}\setminus\{i, j\}) = f_1(a, b)f_2(b, c)f_3(a, d)f_4(b, e)f_5(c, g)f_6(d, e) \]
\[ f_7(e, g)f_8(d, h)\mu_{i}\rightarrow e, h, g(e, h, g) \]

3D messages now \(\Rightarrow\) this is getting intractable!
How can we handle general loopy graphs?

**Loopy Belief Propagation**

- Messages are well defined for loopy graphs:
  
  \[ \mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x) \]
  
  \[ \mu_{f \rightarrow x}(x) = \sum_{\mathcal{X}_f \setminus x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y) \]

- Simply apply them to loopy graphs as well
- We loose exactness (⇒ approximate inference)
- No guarantee of convergence [Yedida et al. 2004]
- But often works astonishingly well in practice
- Same algorithm works for trees (exact) as well as for loopy graphs (approximate) ⇒ Programming exercise
Loopy Belief Propagation

Outline of the algorithm:

- Initialize messages to fixed value (e.g., uniform distribution)
- Perform message updates in fixed or random order
- After convergence: Calculate approximate marginals
- Note: LBP does not always converge
- There exist converging variants: TRW-S [Kolmogorov, PAMI 2006]
Loopy Belief Propagation

Which message passing schedule?

- Random or fixed order
- Popular choice:
  1. Factors $\rightarrow$ variables
  2. Variables $\rightarrow$ factors
  3. Repeat for $N$ iterations
- Can be run in parallel as factor graph is bipartite:

![Factor Graph Diagram]
Loopy Belief Propagation

Sum-Product Belief Propagation

- Goal: Compute marginals of distribution
- Multiplying many double-precision numbers is not a good idea
- Better use log messages $\lambda(x) = \log \mu(x)$:
  - Factor-to-variable messages:
    $$\mu_{f \rightarrow x}(x) = \sum_{x' \in \mathcal{X}_f \setminus x} \phi_f(x') \prod_{y \in \mathcal{X}_f \setminus x} \mu_{y \rightarrow f}(y)$$
    $$\lambda_{f \rightarrow x}(x) = \log \left( \sum_{x' \in \mathcal{X}_f \setminus x} \phi_f(x') \exp \left\{ \sum_{y \in \text{ne}(f)} \lambda_{y \rightarrow f}(y) \right\} \right) \quad (1)$$
  - Variable-to-factor messages:
    $$\mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x)$$
    $$\lambda_{x \rightarrow f}(x) = \sum_{g \in \{\text{ne}(x) \setminus f\}} \lambda_{g \rightarrow x}(x) \quad (2)$$

- $\sum_{x' \in \mathcal{X}_f \setminus x}$: Summation over all states in $\mathcal{X}_f \setminus x$
- $\sum_{y \in \text{ne}(f)}$: Summation over all incoming messages
- To avoid numbers from getting too large, normalize $\lambda_{x \rightarrow f}(x)$ after the message update (Eq. 2), for example by subtracting its mean
Loopy Belief Propagation

Max-Product/Sum Belief Propagation

- Goal: Find most likely state (MAP state)
- Very similar to sum-product, only factor-to-variable message changes
- As before, we better use log messages \( \lambda(x) = \log \mu(x) \):
  - Factor-to-variable messages:
    \[
    \begin{align*}
    \mu_{f \rightarrow x}(x) &= \max_{x \setminus f} \left[ \phi_f(\mathcal{X}_f) \prod_{y \in \mathcal{X}_f \setminus x} \mu_{y \rightarrow f}(y) \right] \\
    \lambda_{f \rightarrow x}(x) &= \max_{x \setminus f} \left[ \log \phi_f(\mathcal{X}_f) + \sum_{y \in \text{ne}(f)} \lambda_{y \rightarrow f}(y) \right] 
    \end{align*}
    \]
  - Variable-to-factor messages:
    \[
    \begin{align*}
    \mu_{x \rightarrow f}(x) &= \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x) \\
    \lambda_{x \rightarrow f}(x) &= \sum_{g \in \{\text{ne}(x) \setminus f\}} \lambda_{g \rightarrow x}(x)
    \end{align*}
    \]
- \( \max_{x \setminus f} \) : Maximization over all states in \( \mathcal{X}_f \setminus x \)
- \( \sum_{y \in \text{ne}(f)} \) : Summation over all incoming messages
- To avoid numbers from getting too large, normalize \( \lambda_{x \rightarrow f}(x) \) after the message update (Eq. 2), for example by subtracting its mean
Loopy Belief Propagation

Unary and Pairwise Factor-to-Variable Messages
Factor-to-variable messages simplify as follows if you only consider unary or pairwise factors. Variable-to-factor messages don’t simplify.

▶ Sum-Product Belief Propagation:
  ▶ Unary factor $\phi_f(x)$:
    $$\lambda_{f\rightarrow x}(x) = \log \phi_f(x)$$  \hspace{1cm} (1)
  ▶ Pairwise factor $\phi_f(x, y)$:
    $$\lambda_{f\rightarrow x}(x) = \log \left( \sum_y \phi_f(x, y) \exp \{ \lambda_{y\rightarrow f}(y) \} \right)$$  \hspace{1cm} (1)

▶ Max-Product Belief Propagation:
  ▶ Unary factor $\phi_f(x)$:
    $$\lambda_{f\rightarrow x}(x) = \log \phi_f(x)$$  \hspace{1cm} (3)
  ▶ Pairwise factor $\phi_f(x, y)$:
    $$\lambda_{f\rightarrow x}(x) = \max_y \left[ \log \phi_f(x, y) + \lambda_{y\rightarrow f}(y) \right]$$  \hspace{1cm} (3)

Note: The sum/max here run over all states of variable $y$!
Loopy Belief Propagation

Let’s implement this now! Which data structures to use?

- A vector variables containing the #labels each variable can take
- A vector factors; each factor contains:
  - The variable id or id’s of the variables it is connected to
  - A vector or matrix storing the factor values for all states
- A vector of factor-to-variable messages \((\lambda_{f \rightarrow x})\)
- A vector of variable-to-factor messages \((\lambda_{x \rightarrow f})\)
- Each message contains:
  - The id’s of the involved variables, factors and input messages it depends on for enabling quick updates according to the formulas on the previous slide
  - The message log values themselves (a vector, length: #labels)
- variables and factors are the inputs to the algorithm
- messages are computed by the algorithm
Loopy Belief Propagation

**Belief Propagation Algorithm** (handles both cases)

- **Input:** variables and factors
- **Allocate all messages**
- **Initialize the message log values to 0** (=uniform distribution)
- **For** $N = 10$ iterations do
  - Update all factor-to-variable messages (Eq. 1 or Eq. 3)
  - Update all variable-to-factor messages (Eq. 2)
  - Normalize all variable-to-factor messages:
    $$\mu_{x \rightarrow f}(x) \leftarrow \mu_{x \rightarrow f}(x) - \text{mean}(\mu_{x \rightarrow f}(x))$$
- **Read off** marginal or MAP state at each variable:

\[
\lambda(x) = \sum_{g \in \{\text{ne}(x)\}} \lambda_{g \rightarrow x}(x)
\]
\[
p(x) = \exp\{\lambda(x)\} / \sum_{x} \exp \{\lambda(x)\}
\]

\[
x^* = \arg\max_{x} \sum_{g \in \{\text{ne}(x)\}} \lambda_{g \rightarrow x}(x)
\]
Imagine ...
Denoising a Binary Image

Can we recover the original image from the noisy observation?

Let us model this using a MRF!

- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unary potentials: $\psi_1(x_1), \ldots, \psi_{100}(x_{100})$
- $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i$
- Log representation: $\psi_i(x_i) = \log f_i(x_i)$

$$p(x) = \frac{1}{Z} \prod_i f_i(x_i) = \frac{1}{Z} \exp \{ \sum_i \psi_i(x_i) \}$$
Denoising a Binary Image

What will be the outcome of MAP inference with unary factors only?

- Maximizing a MRF with unary factors only is equivalent to maximizing each factor individually (no dependencies)
- Thus the result equals the observation
Denoising a Binary Image

What can we do?

- Let us look at the clean image again!
- What prior knowledge do we have about this image?
- Smoothness! (Neighboring pixels tend to have the same label)
- Really? How many neighbors share / don’t share their label?
- $10 \times 10 \times 2 - 20 = 180$ neighborhood relationships in total
- $34 \times$ label transition $\Rightarrow 146 \times$ same label
Denoising a Binary Image

Introducing a Smoothness Prior

Log representation:

\[
p(x) \propto \exp \left\{ \sum_{i=1}^{100} \psi_i(x_i) + \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}
\]

Variables: \(x_1, \ldots, x_{100} \in \{0, 1\}\)

Unary potentials: \(\psi_i(x_i) = [x_i = o_i]\) with pixel observation \(o_i \in \{0, 1\}\)

Pairwise potentials: \(\psi_{ij}(x_i, x_j) = \alpha \cdot [x_i = x_j]\)

Parameter \(\alpha\) controls the strength of the smoothing / prior
Ising Model

Ising Model (1924)

- Statistical mechanics
- Mathematical model of ferromagnetism
- Magnetic dipole moments of atomic spins
- Two states: +1 and -1, arranged in lattice
- Allows identification of phase transitions

Ernst Ising (1900-1998)

- Studies in Göttingen, Bonn, Hamburg
- Investigated simple chain model
- Grid model solved in 1944 by Osanger
- School teacher (Caputh, Berlin)
- Escaped to US (Bradley University, Illinois)
Denoising a Binary Image

What will the MAP result look like?

- Programming exercise
- Play with smoothness parameters $\alpha$
- How to set $\alpha$ in a principled fashion?
- Learn from training data! ⇒ Next week ...
- Next: Approximate inference via sampling
So far:

- We learned about one particular deterministic approximation
- There are other deterministic techniques (overview at end of lecture)
- There is also another way of approaching approximate inference:

**Sampling**

**Deterministic Approximation**
- Approximate the model or inference procedure
- Retrieve a determ. solution to this approximation

**Stochastic Approximation**
- Use the true model / target distribution of interest
- Draw samples to approximate this distribution
Motivation: Sampling

Many statistical problems involve solving analytically intractable integrals (for example in Bayesian inference with continuous variables and non-conjugate priors). Typical problems that can be solved with sampling:

- **Normalization:** \( p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \)
- **Marginalization:** \( p(x|y) = \int p(x, z|y)dz \)
- **Maximization:** \( x^* = \arg\max_x p(x|y) \) (no integral here)
- **Expectation:** \( E_p(f(x)) = \int f(x)p(x)dx \)

Examples for functions \( f(x) \) in the latter case:

- The expectation: \( \int xp(x)dx \)
- The variance: \( \int x^2p(x)dx - (\int xp(x)dx)^2 \)
- The expected risk: \( \int \text{risk}(x)p(x)dx \)
Monte Carlo Approximation

The more samples we draw, the better the approximation:

\[
\frac{1}{N} \sum_{i=1}^{N} f(x_i) \xrightarrow{N \to \infty} \int f(x)p(x)\,dx
\]

The estimate is unbiased and will almost surely converge to the right value by the strong law of large numbers.

Difficulties: Obtaining uncorrelated samples for fast convergence.
Basic Sampling Strategies

- For most (multivariate) standard distributions there exist good sampling algorithms that you can just call in Python/MATLAB
- Uniform, Gaussian, Poisson, Dirichlet, Discrete
- But those are usually not the distributions we are interested in
- Our distributions specified by a graphical model are more complex
So how to sample?
Let’s look at the simple univariate case first
Discrete Case

- Assume distribution: \( p(x) = \begin{cases} 
0.6 & x = 1 \\
0.1 & x = 2 \\
0.3 & x = 3 
\end{cases} \)

- Calculate cumulant: \( c(y) = \sum_{x \leq y} p(x) = \begin{cases} 
0.6 & y = 1 \\
0.7 & y = 2 \\
1.0 & y = 3 
\end{cases} \)

- Draw \( u \sim [0, 1] \) using pseudo-random number generator
- Find \( y \) such that: \( c(y - 1) < u \leq c(y) \)
- Return state \( y \) as sample from \( p \)
Continuous Case

- Similar to the discrete case
- Compute the **cumulant** function:
  \[
  c(y) = \int_{-\infty}^{y} p(x)dx
  \]
- Sample \( u \sim [0, 1] \Rightarrow \) compute \( x = c^{-1}(u) \)
- The integral \( c(y) \) can be computed analytically or numerically

For example: \( p(x) = \begin{cases} 
  \exp(-x) & 0 \leq x, \\
  0 & \text{else}
\end{cases} \)
Overview: Sampling Methods

- Inverse Transform
- Ancestral Sampling
- Rejection Sampling
- Importance Sampling
- Slice Sampling
- Markov Chain Monte Carlo
  - Metropolis-Hastings
  - Gibbs Sampling
  - Hybrid Monte Carlo

- Do I need to know them all?
- Yes! Most efficient technique depends on model/application
- Today “only” the ones in red ;}
Rejection Sampling
Rejection Sampling

- Suppose a \( p(x) \) such that direct sampling is not tractable
- Furthermore assume we can evaluate \( p(x) \) up to a constant (e.g., Markov Networks!):
  \[
p(x) = \frac{1}{Z} \tilde{p}(x) = \frac{1}{Z} \prod_c \phi_c(x_c)
\]

- Sample from a proposal distribution \( q(x) \)
- Choose \( q(\cdot) \) which we can easily sample and a \( k \) exists with
  \[
k \cdot q(x) \geq \tilde{p}(x) \quad \forall x
\]
Rejection Sampling

- Sample two random variables:
  1. $z_0 \sim q(x)$
  2. $u \sim [0, kq(z_0)]$ uniform
- Reject sample $z_0$ if $u_0 > \tilde{p}(z_0)$

$z_0$ from $q$ is accepted with probability $\tilde{p}(z)/kq(z)$

$$p(\text{accept}) = \int \frac{\tilde{p}(z)}{kq(z)} q(z)dz = \frac{1}{k} \int \tilde{p}(z)dz$$

- $k = 1$ and $q(x) = p(x) \Rightarrow p(\text{accept}) = 1$
- But often: $p(\text{accept} \mid x) = \prod_{i=1}^{D} p(\text{accept} \mid x_i) = \mathcal{O}(\gamma^D)$
Rejection Sampling

Robot Localization Example

- You bought a vacuum robot for your living room (1 × 1 m)
- For proper cleaning, the robot needs to localize itself
- No prior knowledge on location: \( \mathbf{x} \sim U([0, 1] \times [0, 1]) \)
- Independent measurements: \( d_i | \mathbf{x} \sim \mathcal{N}(\| \mathbf{x} - \mathbf{e}_i \|, \sigma^2) \)

\[
p(\mathbf{x}|d_1, d_2, d_3, d_4) \propto p(\mathbf{x}) p(d_1|\mathbf{x}) p(d_2|\mathbf{x}) p(d_3|\mathbf{x}) p(d_4|\mathbf{x}) \\
\propto [0 \leq x_1, x_2 \leq 1] \\
\times \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{4} [\| \mathbf{x} - \mathbf{e}_i \| - d_i]^2 \right)
\]
Rejection Sampling

Robot Localization Example

- The maximum of the unnormalized posterior is 1
- Thus we can choose: \( q(x) = [0 \leq x_1, x_2 \leq 1] \)
Metropolis-Hastings Sampling
Metropolis-Hastings Sampling

Markov Chain

- Discrete random process with Markov property:

\[ P(x_i|x_{i-1}, \ldots, x_1) = P(x_i|x_{i-1}) = P(x'|x) \]

Markov Chain Monte Carlo (MCMC)

- We want to sample from \( p(x) = \frac{1}{Z} \tilde{p}(x) \) with \( Z \) unknown
- Idea: Establish a Markov chain with transition kernel \( T(x' \mid x) \) and with stationary distribution \( p(x) \):

\[ p(x') = \int_X T(x' \mid x) p(x) dx \]

- Task: Find \( T(x' \mid x) \) such that \( p(x) \) is its stationary distribution!
Metropolis-Hastings Sampling

Metropolis-Hastings

- Initialize $x$ and specify proposal distribution $q(x'|x)$
- Sample $x'$ from $q(x'|x)$ and accept with probability

$$A(x', x) = \min \left(1, \frac{p(x') q(x|x')}{p(x) q(x'|x)}\right) = \min \left(1, \frac{\tilde{p}(x') q(x|x')}{\tilde{p}(x) q(x'|x)}\right)$$

- If accepted: $x \leftarrow x'$
- If not accepted: stay at $x$
- Iterate (sample again)
Example: 2D Gaussian

- 150 proposal steps, 43 are rejected (red)
Why does it work?

- Remember the acceptance probability:

\[ A(x', x) = \min \left( 1, \frac{p(x') q(x|x')}{p(x) q(x'|x)} \right) \]

- Let us write down the transition kernel \( T(x'|x) \)
  i.e., the probability to transition the state from \( x \) to \( x' \):

\[
T(x'|x) = q(x'|x) A(x', x) \\
+ \delta(x' - x) \int q(\tilde{x}|x) [1 - A(\tilde{x}|x)] \, d\tilde{x}
\]
Why does it work?

\[
\int T(x'|x)p(x)dx = \int \min\{p(x)q(x'|x), p(x')q(x|x')\}dx \\
+ \int p(x')q(\tilde{x}|x')[1 - A(\tilde{x}|x')]d\tilde{x} \\
= \int \min\{p(x)q(x'|x), p(x')q(x|x')\}dx \\
+ p(x') \int q(\tilde{x}|x')d\tilde{x} \\
- \int p(x')q(\tilde{x}|x')A(\tilde{x}|x')d\tilde{x} \\
= \int \min\{p(x)q(x'|x), p(x')q(x|x')\}dx \\
+ p(x') \\
- \int \min\{p(x')q(\tilde{x}|x'), p(\tilde{x})q(x'|\tilde{x})\}d\tilde{x} \\
= p(x')
\]
Why does it work?

Other requirements that need to be fulfilled:

- **Irreducibility**: Any state \( x' \) can be reached by any other state \( x \) in a finite number of steps.

- **Aperiodicity**: The occurrence of states is not restricted to periodic events (any state may occur at any time).
Example: Irreducibility

- $q(x'|x)$ needs to be able to bridge the gap
Metropolis-Hastings Sampling

Robot Localization Example

- Now inferring 2 variables: location $x$ and sensor noise $\sigma$
- Uniform prior on location: $x \sim \mathcal{U}([0, 1] \times [0, 1])$
- Uniform prior on sensor noise: $\sigma \sim \mathcal{U}(0.01, 0.5)$
- Measurements depend on $\sigma$: $d_i|x, \sigma \sim \mathcal{N}(\|x-e_i\|, \sigma^2)$

$$p(x, \sigma|d_1, \ldots, d_{16}) \propto p(x)p(\sigma)p(d_1|x, \sigma) \cdots p(d_{16}|x, \sigma)$$
$$\propto [0 \leq x_1, x_2 \leq 1] \times [0.01 \leq \sigma \leq 0.5]$$
$$\times \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{16} [\|x-e_i\|-d_i]^2 \right) \frac{1}{(2\pi \sigma^2)^8}$$

Diagram: Graph representing the relationships between $x$, $\sigma$, and measurements $d_1$ to $d_{16}$.
Metropolis-Hastings Sampling

Robot Localization Example

- 2465 rejected
- Markov chain
- 500 accepted
Gibbs Sampling

Special case of MH Sampling:

- Cyclic MH kernel that updates one variable at a time
- Sample directly from the full conditional distribution

\[ q(x'|x) = p(x_k|x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_D) \]

- Samples get accepted with probability 1 (exercise)
- But: conditionals must be easy to sample from!
- Danger of slow convergence and non-irreducibility:
Approximate Inference Overview

- **Deterministic Inference**
  - Junction Tree (not approximate but intractable)
  - Loopy Belief Propagation
  - Variational Approximation
  - Expectation Propagation
  - Mean field
  - Gradient Descent
  - ...

- **Sampling**
  - Rejection Sampling
  - Slice Sampling
  - Metropolis-Hastings Sampling
  - Gibbs Sampling
  - ...

Recap
Loopy Belief Propagation
Sampling
Next Time ...

- Learning
- And after that: Computer Vision, finally!
- No more toy examples, but real stuff - promised ;)

Recap
Loopy Belief Propagation
Sampling