$\LaTeX$ command declarations here.

In [1]:

```
from __future__ import division

# scientific
%matplotlib inline
from matplotlib import pyplot as plt;
import matplotlib as mpl;
import numpy as np;
import sklearn as skl;
import sklearn.datasets;
import sklearn.cluster;
import sklearn.mixture;

# ipython
import IPython;

# python
import os;
import random;

######################################################

# image processing
import PIL;

# trim and scale images
def trim(im, percent=100):
    print("trim:", percent);
    bg = PIL.Image.new(im.mode, im.size, im.getpixel((0,0)))
    diff = PIL.ImageChops.difference(im, bg)
    diff = PIL.ImageChops.add(diff, diff, 2.0, -100)
    bbox = diff.getbbox()
    if bbox:
        x = im.crop(bbox)
        return x.resize(((x.size[0]*percent)//100, (x.size[1]*percent)//1
 PIL.Image.ANTIALIAS);


######################################################

# daft (rendering PGMs)
import daft;

# set to FALSE to load PGMs from static images
RENDER_PGMS = True;

# decorator for pgm rendering
def pgm_render(pgm_func):
    def render_func(path, percent=100, render=None, *args, **kwargs):
        print("render_func:", percent);
        # render
        render = render if (render is not None) else RENDER_PGMS;

        if render:
            print("rendering");
            # render
            pgm = pgm_func(*args, **kwargs);
            pgm.render();
```

```
            pgm.figure.savefig(path, dpi=300);

            # trim
            img = trim(PIL.Image.open(path), percent);
            img.save(path, 'PNG');
        else:
            print("not rendering");

        # error
        if not os.path.isfile(path):
            raise Exception("Error:  Graphical model image %s not found.
  You may need to set RENDER_PGMS=True." % path);

        # display
        return IPython.display.Image(filename=path);#trim(PIL.Image.open
(path), percent);

    return render_func;

######################################################
```

EECS 545: Machine Learning

Lecture 18: Inference & Applications of PGMs

- Instructor: **Jacob Abernethy**
- Date: March 23, 2016

Lecture Exposition: Benjamin Bray

References

- **[MLAPP]** Murphy, Kevin. *Machine Learning: A Probabilistic Perspective* (https://mitpress.mit.edu/books/machine-learning-0). 2012.
- **[PRML]** Bishop, Christopher. *Pattern Recognition and Machine Learning* (http://research.microsoft.com/en-us/um/people/cmbishop/prml/). 2006.
- **[Koller & Friedman 2009]** Koller, Daphne and Nir Friedman. *Probabilistic Graphical Models* (https://mitpress.mit.edu/books/probabilistic-graphical-models)

Review: Hidden Markov Models

> Uses material from **[MLAPP]**

Hidden Markov Models

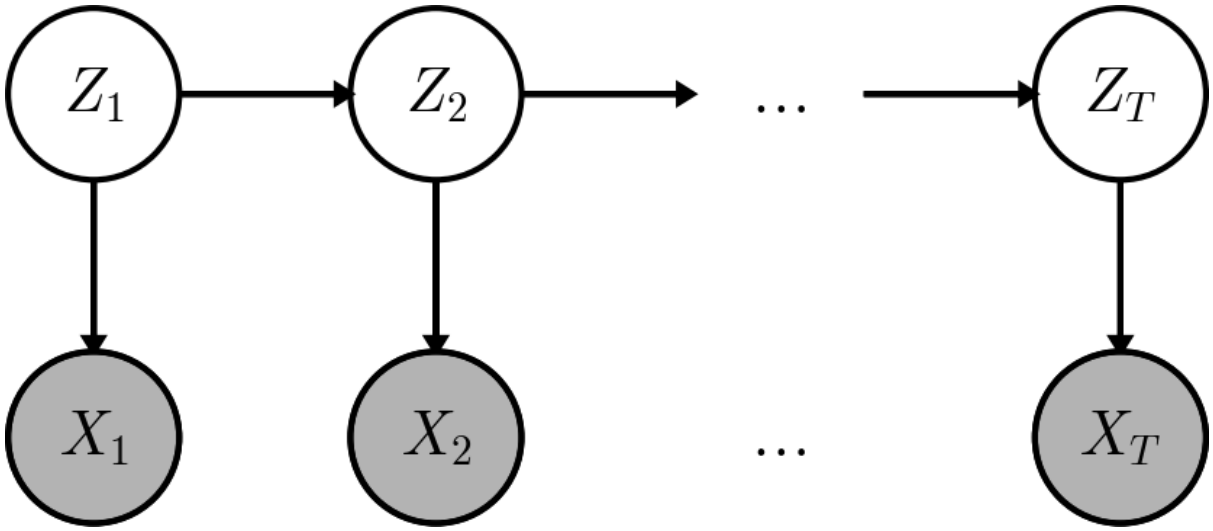
Noisy observations X_k generated from *discrete* hidden Markov chain Z_k.

$$P(\mathbf{X}, \mathbf{Z}) = P(Z_1)P(X_1 \mid Z_1)\prod_{k=2}^{T} P(Z_k \mid Z_{k-1})P(X_k \mid Z_k)$$

In [31]:
```
@pgm_render
def pgm_hmm():
    pgm = daft.PGM([7, 7], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("Z1", r"$Z_1$", 1, 3.5))
    pgm.add_node(daft.Node("Z2", r"$Z_2$", 2, 3.5))
    pgm.add_node(daft.Node("Z3", r"$\dots$", 3, 3.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("Z4", r"$Z_T$", 4, 3.5))

    pgm.add_node(daft.Node("x1", r"$X_1$", 1, 2.5, observed=True))
    pgm.add_node(daft.Node("x2", r"$X_2$", 2, 2.5, observed=True))
    pgm.add_node(daft.Node("x3", r"$\dots$", 3, 2.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("x4", r"$X_T$", 4, 2.5, observed=True))


    # Add in the edges.
    pgm.add_edge("Z1", "Z2", head_length=0.08)
    pgm.add_edge("Z2", "Z3", head_length=0.08)
    pgm.add_edge("Z3", "Z4", head_length=0.08)

    pgm.add_edge("Z1", "x1", head_length=0.08)
    pgm.add_edge("Z2", "x2", head_length=0.08)
    pgm.add_edge("Z4", "x4", head_length=0.08)

    return pgm;
```

In [32]:
```
%%capture
pgm_hmm("images/pgm/hmm.png")
```

Out[32]:



HMM: Parameters

For a Hidden Markov Model with N hidden states and M observed states, there are three *row-stochastic* parameters $\theta = (A, B, \pi)$,

- Transition matrix $A \in \mathbb{R}^{N \times N}$

$$A_{ij} = p(Z_t = j | Z_{t-1} = i)$$

- Emission matrix $B \in \mathbb{R}^{N \times M}$

$$B_{jk} = p(X_t = k | Z_t = j)$$

- Initial distribution $\pi \in \mathbb{R}^N$,

$$\pi_j = p(Z_1 = j)$$

HMM: Filtering Problem

Filtering means to compute the current *belief state* $p(z_t | x_1, \ldots, x_t, \theta)$.

$$p(z_t | x_1, \ldots, x_t) = \frac{p(x_1, \ldots, x_t, z_t)}{p(x_1, \ldots, x_t)}$$

- Given observations $x_{1:t}$ so far, infer z_t.
- *Example*: Estimate robot position given previous sensor readings.

> Solved by the **forward algorithm**.

How do we infer values of hidden variables?

- One of the most challenging part of HMMs is to try to "predict" what are the values of the hidden variables z_t, having observed all the $x_1, \ldots, x_T$.
- Computing $p(z_t \mid \mathcal{X})$ is known on *smoothing*. More on this soon.
- But it turns out that this probability can be computed from two other quantities:
  - $p(x_1, \ldots, x_t, z_t)$, which we are going to label $\alpha_t(z_t)$
  - $p(x_{t+1}, \ldots, x_T | z_t)$, which we are going to label $\beta_t(z_t)$

HMM: Forward Algorithm

The **forward algorithm** computes $\alpha_t(z_t) \equiv p(x_1, \ldots, x_t, z_t)$.

> Looks kinda hard to compute this probability, there are a lot of variables! Maybe it can be computed recursively? **Yes!**

$$
\begin{aligned}
\alpha_t(z_t) &= \sum_{z_{t-1}} p(x_1, \ldots, x_t, z_{t-1}, z_t) \\
&= \sum_{z_{t-1}} p(x_1, \ldots, x_{t-1}, z_{t-1}) p(z_t | z_{t-1}) p(x_t | z_t) \\
&= p(x_t | z_t) \sum_{z_{t-1}} \alpha_{t-1}(z_{t-1}) p(z_t | z_{t-1}) \\
&= B_{z_t, x_t} \sum_{z_{t-1}} \alpha_{t-1}(z_{t-1}) A_{z_{t-1}, z_t}
\end{aligned}
$$

> Recursion starts from the *front* of the chain. (Suggestion: you need to work out the above recursion on your own, it's a really important exercise)

HMM: Backward Algorithm

The **backward algorithm** computes $\beta_t(z_t) \equiv p(x_{t+1}, \ldots, x_T | z_t)$,

> Again, maybe we can try to compute this recursively?

$$
\begin{aligned}
\beta(z_t) &= \sum_{z_{t+1}} p(x_{t+1}, \dots, x_T, z_{t+1} | z_t) \\
&= \sum_{z_{t+1}} p(z_{t+1}|z_t) p(x_{t+1}, \dots, x_T | z_{t+1}, z_t) \\
(\text{since } x_{t+1:T} \perp z_t \mid z_{t+1}) \quad &= \sum_{z_{t+1}} p(z_{t+1}|z_t) p(x_{t+1}, \dots, x_T | z_{t+1}) \\
(\text{since } x_{t+2:T} \perp x_{t+1} \mid z_{t+1}) \quad &= \sum_{z_{t+1}} p(z_{t+1}|z_t) p(x_{t+1}|z_{t+1}) p(x_{t+2}, \dots, x_T | z_{t+1}) \\
&= \sum_{z_{t+1}} p(z_{t+1}|z_t) p(x_{t+1}|z_{t+1}) \beta_{t+1}(z_{t+1}) \\
&= \sum_{z_{t+1}} A_{z_t, z_{t+1}} B_{z_{t+1}, x_{t+1}} \beta_{t+1}(z_{t+1})
\end{aligned}
$$

> Recursion starts from the *back* of the chain. (Suggestion: you need to work out the above recursion on your own, it's a really important exercise)

HMM: Smoothing Problem

Compute $p(z_t|\mathcal{X})$ offline, given all observations.

- Retroactively infer z_t. (Hindsight!)

We can break the chain into two parts, the *past* and *future*:

$$
\begin{aligned}
p(z_t|\mathcal{X}) &= p(x_{1:t}, z_t, x_{t+1:T}) \frac{1}{p(\mathcal{X})} \\
&= p(x_{1:t}, z_t) p(x_{t+1:T} | x_{1:t}, z_t) \frac{1}{p(\mathcal{X})} \\
&= p(x_{1:t}, z_t) p(x_{t+1:T} | z_t) \frac{1}{p(\mathcal{X})}
\end{aligned}
$$

Exercise: How did we get that last line?

HMM: Smoothing Problem

Overall, the **smoothing problem** is to compute

$$\begin{aligned}\gamma_t(j) \equiv p(z_t = j \mid \mathcal{X}) &= \frac{p(\mathcal{X} \mid z_t = j)p(z_t = j)}{p(\mathcal{X})} \\ &= \frac{p(\mathcal{X}_{1:t} \mid z_t = j)p(\mathcal{X}_{t+1:T}|z_t = j)p(z_t = j)}{p(\mathcal{X})} \\ &= \frac{\alpha_t(j)\beta_t(j)}{p(\mathcal{X})} = \frac{\alpha_t(j)\beta_t(j)}{\sum_k \alpha_t(k)\beta_t(k)}\end{aligned}$$

It is an easy exercise to check that, for any t, $p(\mathcal{X}) = \sum_k \alpha_t(k)\beta_t(k)$.

We solve this via the **forward-backward algorithm**, where

- $\alpha_t(z_t) \equiv p(x_1, \dots, x_t, z_t)$ is found with the **forward algorithm**
- $\beta_t(z_t) \equiv p(x_{t+1}, \dots, x_T|z_t)$ is found with the **backward algorithm**

HMM: Decoding Problem

Decoding computes the most probable state sequence, given observations.

$$\mathbf{z}^* = \arg\max_{z_1,\dots,z_T} p(z_1, \dots, z_T|x_1, \dots, x_T, \theta)$$

> The decoding problem is solved by the **Viterbi algorithm**, which uses dynamic programming. See **[PRML]** or **[MLAPP]** for more details.

Viterbi Algorithm

- Viterbi is another recursive procedure for computing the most likely sequence of hidden states.
- We define $V_t(z_t)$ to be the probability of the most likely sequence of states up to time t *that ended in state* z_t, given observed data
- These $V(z_t)$ values satisfies the recursion:

$$\begin{aligned}V_1(j) &= p(x_1 \mid z_1 = j) = B_{x_1,j} \\ V_{t+1}(j) &= p(x_{t+1} \mid z_{t+1} = j) \max_k \left\{ V_t(k) p(z_{t+1} = j \mid z_t = k) \right\} \\ &= B_{x_{t+1},j} \max_k \left\{ V_t(k) A_{j,k} \right\}\end{aligned}$$

- Can be computed very quickly!

HMM: Part-of-Speech Tagging

In English, some words can have multiple parts of speech. For instance,

- Business is going **well** (*Adverb*).
- All is **well** with us (*Adjective*).
- **Well**, who would have thought he could do it? (*Interjection*)
- The **well** was drilled fifty meters deep. (*Noun*)
- Tears **well** up in my eyes. (*Verb*)

(Example taken from [here](http://english.stackexchange.com/questions/46277/what-word-can-fulfill-the-most-parts-of-speech).)

HMM: Part-of-Speech Tagging

We can use a Hidden Markov Model to **disambiguate** the part of speech using context clues!

- Hidden states z_t are parts-of-speech
- Observed states x_t are words

Certain sequences of POS tags are unlikely. This allows us to infer the correct tags!

EM for HMMs: The Baum-Welch Algorithm

HMM: Learning Problem

It is usually necessary to **learn** the model parameters $\theta = (A, B, \pi)$ from data.

- Given observations $\mathcal{X} = \{x_1, \ldots, x_T\}$
- Given model dimensions N and M
- Find parameters, i.e. the matrices A & B, and the vector π that best fit the data.

> The learning problem is solved by the **Baum-Welch algorithm**, a special case of expectation maximization.

Recall: Expectation-Maximization

E-Step: Write down an expression for

$$Q(\theta_t, \theta) = E_q[\log p(\mathcal{X}, Z|\theta)] \qquad \text{where } q = q(\cdot|\theta_t)$$

M-Step: Maximize the auxiliary function,

$$\theta_{t+1} = \arg\max_{\theta} Q(\theta_t, \theta)$$

Recall $q_t(Z) = p(Z|\mathcal{X}, \theta_t)$

HMM: Complete-Data Log-Likelihood

The joint likelihood of the hidden and observed states is

$$\begin{aligned}\log p(x_{1:T}, z_{1:T}|\theta) &= \log\left[p(z_1|\pi)\prod_{t=2}^{T} p(z_t|z_{t-1}, A)\prod_{t=1}^{T} p(x_t|z_t, B)\right] \\ &= \log p(z_1|\pi) + \sum_{t=2}^{T}\log p(z_t|z_{t-1}, A) \\ &\quad + \sum_{t=1}^{T}\log p(x_t|z_t, B)\end{aligned}$$

HMM: Complete-Data Log-Likelihood

Each term of the complete-data log-likelihood is:

$$\log p(z_1|\pi) = \sum_{j=1}^{N}\mathbb{I}(z_t = j)\log\pi_j$$

$$\log p(z_t|z_{t-1}, A) = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{I}(z_{t-1} = i)\mathbb{I}(z_t = j)\log A_{ij}$$

$$\log p(x_t|z_t, B) = \sum_{j=1}^{N}\mathbb{I}(z_t = j)\log B_{j,x_t}$$

HMM: Expected Complete Likelihood

The expected complete likelihood $Q(\theta_t, \theta)$ is

$$\begin{aligned} Q(\theta_t, \theta) &= E_q[\log p(\mathcal{X}, Z|\theta)] \\ &= E_q[\log p(z_1|\pi)] + E_q\left[\sum_{t=2}^{T} \log p(z_t|z_{t-1}, A)\right] \\ &\quad + E_q\left[\sum_{t=1}^{T} \log p(x_t|z_t, B)\right] \end{aligned}$$

HMM: Expected Complete Likelihood

Fixing $t > 1$, and taking expectations with respect to $q(Z) = p(Z|\mathcal{X}, \theta)$,

$$E_q[\log p(z_1|\pi)] = \sum_{j=1}^{N} q(z_1 = j) \log \pi_j$$

$$E_q[\log p(z_t|z_{t-1}, A)] = \sum_{i=1}^{N}\sum_{j=1}^{N} q(z_{t-1} = i, z_t = j) \log A_{ij}$$

$$E_q[\log p(x_t|z_t, B)] = \sum_{j=1}^{N} q(z_t = j) \log B_{j,x_t}$$

HMM: Baum-Welch

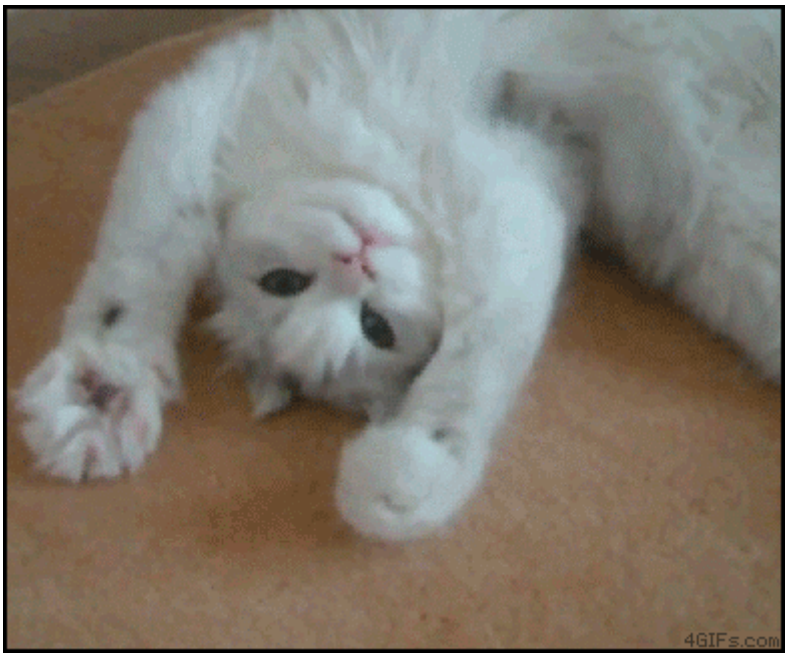
- The **E-Step** consists of computing the q terms from the previous slide, which can all be computed using the **forward-backward algorithm**!
- The **M-Step** consists of normalizing the expected transition and emission counts
  - Similar to MLE for complete data
  - Requires some careful calculations. See details on next slide.

You must compute the key quantities for $q()$ using the forward-backward algorithm. In the E step you can assume the parameters $\theta = (A, B, \pi)$, so we'll drop dependence on θ.

1. First note that $q(z_t = j) = p(z_t = j \mid \mathcal{X}, \theta) = \gamma_t(j)$ which we computed above.
2. We then compute $q(z_{t-1} = i, z_t = j) = p(z_{t-1} = i, z_t = j | \mathcal{X}, \theta)$.

$$
\begin{aligned}
p(z_{t-1}, z_t | \mathcal{X}, \theta) &= \frac{p(\mathcal{X} \mid z_t, z_{t-1}) p(z_t, z_{t-1})}{p(\mathcal{X})} \\
&= \frac{p(\mathcal{X}_{1:t-1} \mid z_t, z_{t-1}) p(\mathcal{X}_{t:T} | z_t, z_{t-1}) p(z_t, z_{t-1})}{p(\mathcal{X})} \\
&= \frac{p(\mathcal{X}_{1:t-1} \mid z_{t-1}) p(\mathcal{X}_{t:T} | z_t) p(z_t, z_{t-1})}{p(\mathcal{X})} \\
&= \frac{p(\mathcal{X}_{1:t-1}, z_{t-1}) p(x_t \mid z_t) p(\mathcal{X}_{t+1:T} | z_t) p(z_t, z_{t-1})}{p(z_{t-1}) p(\mathcal{X})} \\
&= \frac{\alpha_{t-1}(z_{t-1}) \beta_t(z_t) p(z_t \mid z_{t-1}) p(x_t \mid z_t)}{p(\mathcal{X})} \\
&= \frac{\alpha_{t-1}(z_{t-1}) \beta_t(z_t) A_{z_{t-1}, z_t} B_{z_t, x_t}}{\sum_k \alpha_t(k) \beta_t(k)}
\end{aligned}
$$

Break Time!



Graphical Models: Wrap-Up

We've just barely scratched the surface of what **probabilistic graphical models** have to offer.

- Bayesian Networks
- Mixture Models
- Hidden Markov Models
- Latent Variable Models & Expectation-Maximization

This lecture will be a grab-bag of some of the things we missed.

> Accordingly, we won't test you on the details.

Graphical Models: Related Courses

These courses are being offered next semester:

- **IOE 591**, Bayesian Data Analysis (https://d1b10bmlvqabco.cloudfront.net/attach/iizaobacnl55nb/hku8ttycli2/ilxshlclzteu/IOE591_flyer.pdf)
- **EECS 592**, Advanced Artificial Intelligence

The following courses have been offered in the past:

- **STATS 601**, Analysis of Multivariate and Categorical Data
- **EECS 598**, Probabilistic Graphical Models in Vision and Beyond (http://web.eecs.umich.edu/~jjcorso/t/598W15/)
- **STATS 700**, Probabilistic Graphical Models (http://dept.stat.lsa.umich.edu/~xuanlong/courses/stat700-f09/)

Graphical Models: Other Resources

- **[Koller & Friedman 2009]** Koller, Daphne and Nir Friedman. *Probabilistic Graphical Models* (https://mitpress.mit.edu/books/probabilistic-graphical-models)
- **[Wainwright & Jordan 2008]** Wainwright, Martin, and Michael Jordan. Graphical Models, Exponential Families, and Variational Inference (https://www.eecs.berkeley.edu/~wainwrig/Papers/WaiJor08_FTML.pdf). 2008
- **Coursera,** Probabilistic Graphical Models (https://www.coursera.org/course/pgm)

Outline

- Undirected Graphical Models
  - Markov Random Fields
- Summary of Exact Inference
- Approximate Inference
  - Gibbs Sampling
  - Variational Inference
- Applications
  - Latent Dirichlet Allocation

Undirected Graphical Models

> Uses material from **[MLAPP]** Chapter 19 and **[Koller & Friedman 2009]** Chapter 4

> Also known as **Markov Random Field**

Markov Random Fields: Motivation

So far, we have talked only about **directed models**.

- Directed edges represent an underlying generative process.
- Formalizes independence properties between variables

In some cases, **undirected models** are more natural.

- In many applications, it's hard to assign a direction to the interactions between variables.
- Offer a different, sometimes simpler, perspective

Markov Random Fields: Motivation

Consider the problem of modeling pixels in an image.

- One random variable X_{ij} per pixel (i, j) in the image.
- Intensity values of neighboring pixels are *locally* correlated

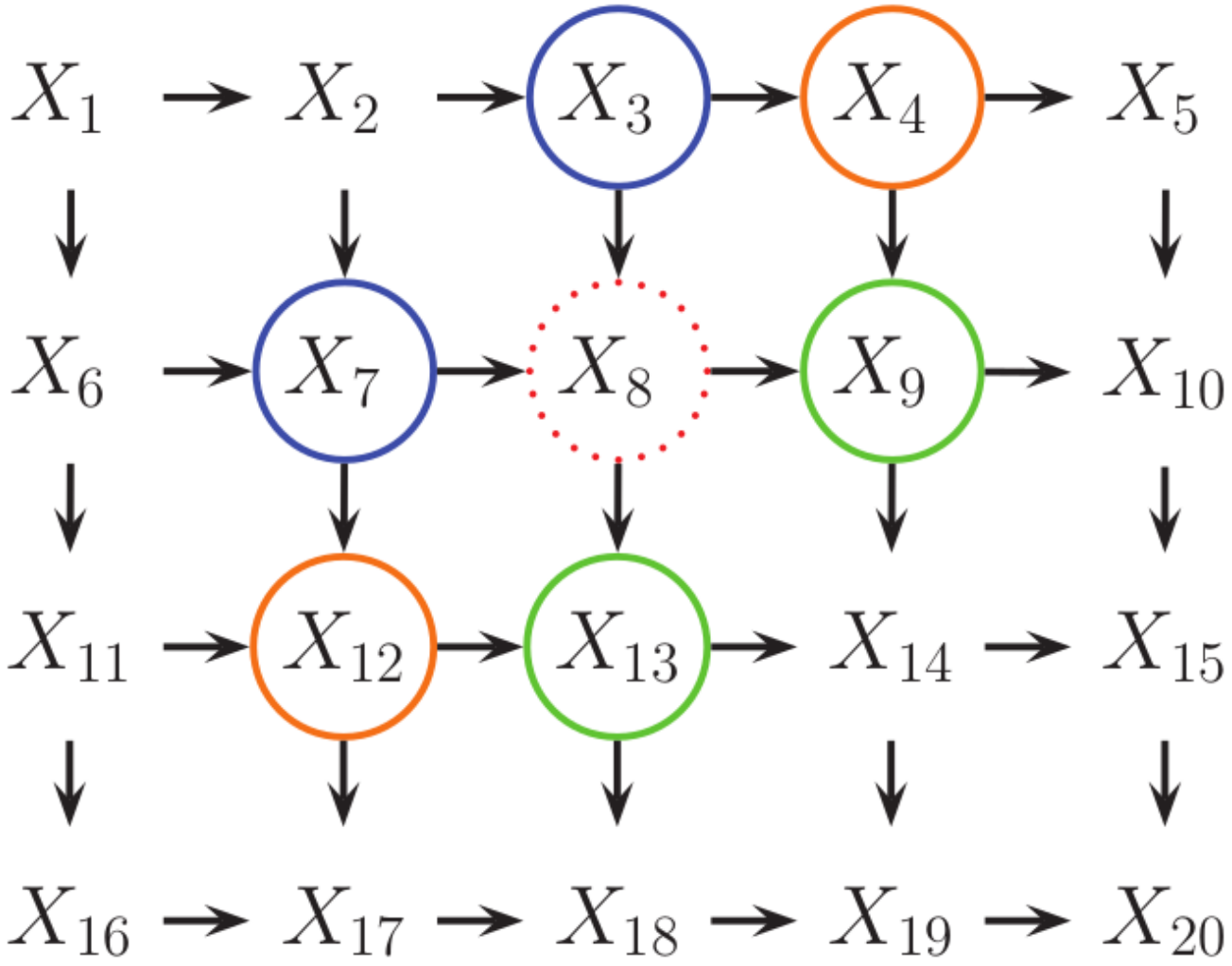
How can we represent this situation as a Bayesian network?

- Which directions do the arrows go?
- Which independence properties are desirable?

Markov Random Fields: Motivation

We could arbitrarily add directed edges, but this leads to strange independence structures.

- Below, X_8 is independent of all other variables given the colored nodes.

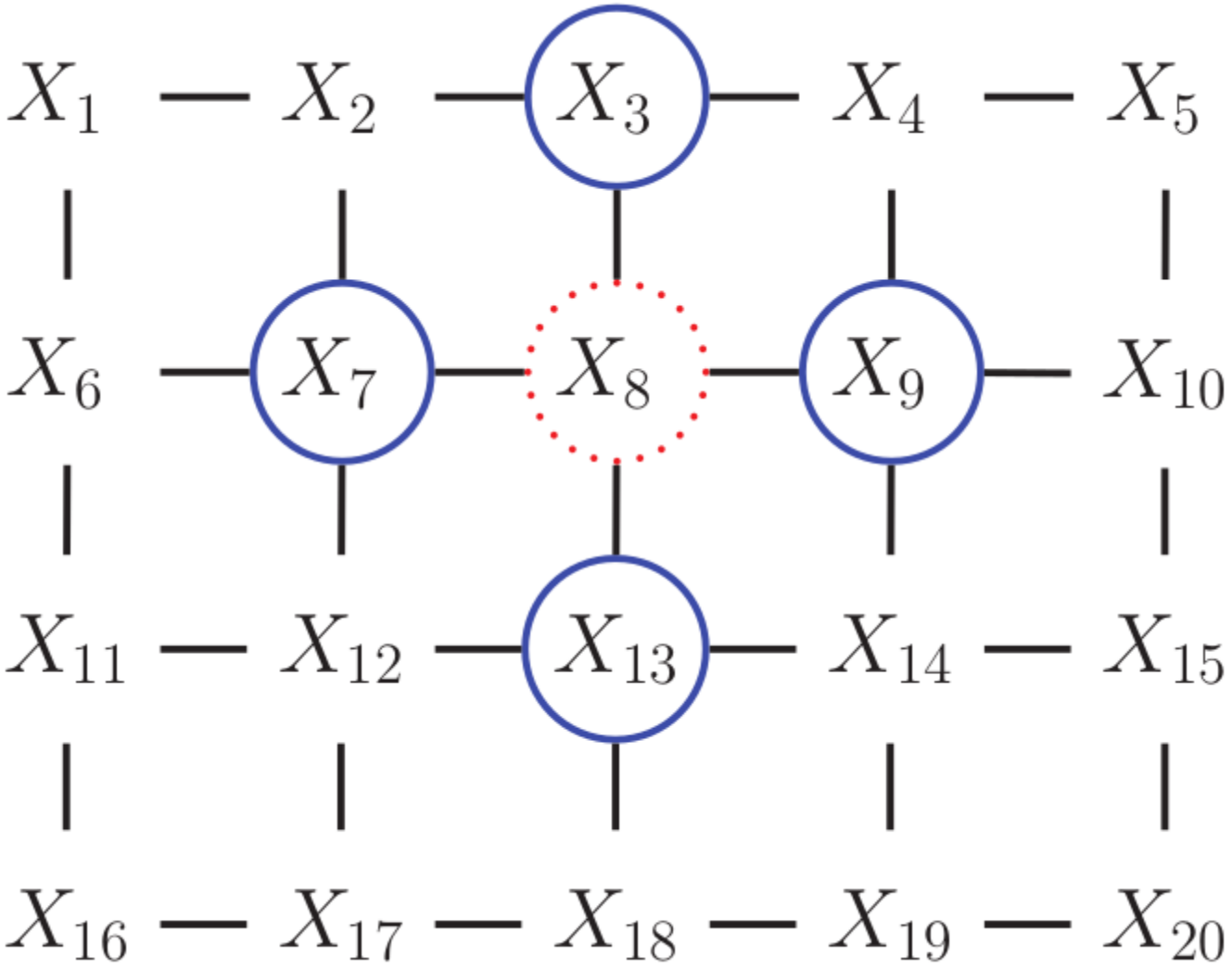


(Figure 19.1a from **[MLAPP]**)

Markov Random Fields: Motivation

Since images have no natural "directionality", why bother with undirected edges?

- Now, X_8 is independent of all other variables given its immediate neighbors.



(Figure 19.1b from **[MLAPP]**)

Markov Random Fields: Parameterization

Directed models come with a natural topological ordering.

- Applying the chain rule yields a factorization.
- It is only necessary to store **conditional probability tables**

Undirected models have no natural ordering!

- Instead, we describe the model in terms of **factors** describing the local interactions between variables.

Markov Random Fields: Factors

Let $\mathbf{D}$ be a set of random variables. A **factor** is a function $\phi : \mathbf{Val}(\mathbf{D}) \mapsto \mathbb{R}$.

- where $\mathbf{Val}(\mathbf{D})$ is the set of possible configurations of the variables.
- where $\mathbf{D}$ is called the *scope* of ϕ

> A factor assigns a real number to every possible configuration of the variables $\mathbf{D}$

Markov Random Fields: Factors

Probability distributions are *normalized* factors.

- $P(X_1, X_2, X_3)$ is a factor on $\mathbf{D} = (X_1, X_2, X_3)$
- $P(X|Y)$ is a factor on $\mathbf{D} = (X, Y)$

Other factors need not be normalized. For example, $\phi(A, B)$ assigns high *affinity* to configurations where A and B agree:

A	B	$\phi(A, B)$
0	0	100
0	1	-50
1	0	-50
1	1	100

Markov Random Fields: Parameterization

Given a set of factors $\mathbf{\Phi} = (\phi_1(\mathbf{D}_1), \phi_2(\mathbf{D}_2), \ldots, \phi_K(\mathbf{D}_K))$, we can define a **Gibbs Distribution**,

$$\begin{aligned} P_{\mathbf{\Phi}}(X_1, \ldots, X_n) &= \frac{1}{Z}\phi_1(\mathbf{D}_1)\phi_2(\mathbf{D}_2)\cdots\phi_K(\mathbf{D}_K) \\ &= \frac{1}{Z}\exp\left[\sum_{k=1}^{K}\log\phi_k(\mathbf{D}_k)\right] \end{aligned}$$

where Z is a normalization constant. Sometimes log-potentials are referred to as **energies** associated with each configuration of variables.

> This model was stolen from physicists :)

Markov Random Fields: Texture Modeling

Suppose we have a **pairwise Markov Random Field** over binary pixels.

- *Binary:* Each pixel is either -1 (black) or 1 (white)
- *Pairwise:* Only model interactions between directly neighboring pixels.

Define the **energy function**, summing over all edges (i, j):

$$U(\mathcal{X}) = -\beta \sum_{(i,j)} x_i x_j \qquad (\beta > 0)$$

Markov Random Fields: Texture Modeling

Then, the probability of an image is

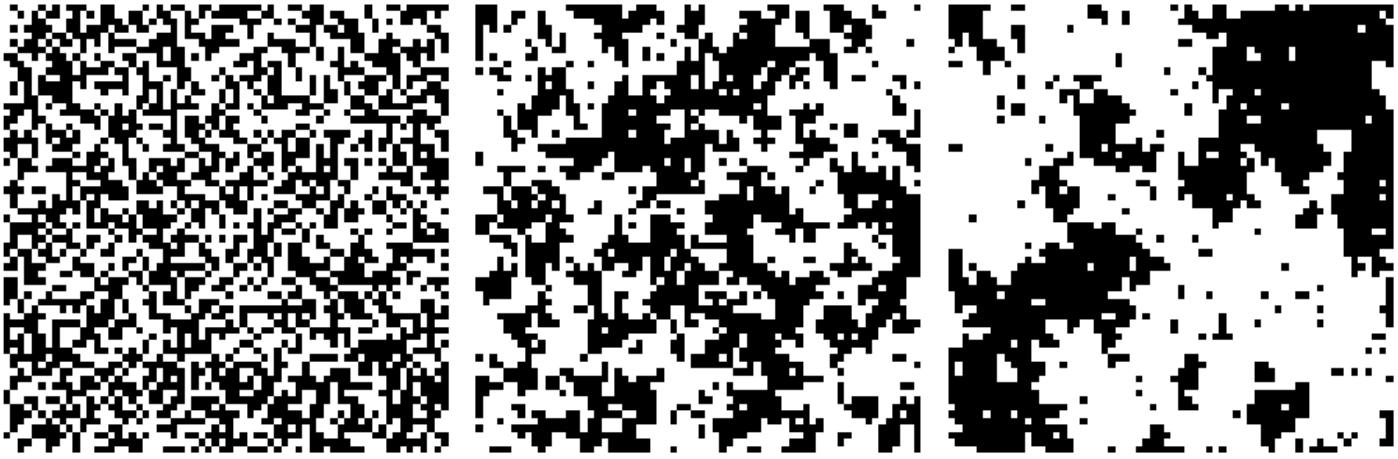
$$P(\mathcal{X}) = \frac{1}{Z} \exp[-U(\mathcal{X})] = \frac{1}{Z} \exp\left[-\beta \sum_{(i,j)} x_i x_j\right]$$

The *affinity parameter* $\beta > 0$ determines how much we want neighboring pixels to agree with each other.

- Low energy = High probability = Neighbors Agree
- High energy = Low probability = Neighbors Disagree

Markov Random Fields: Texture Modeling

Using Gibbs Sampling (later!) we can generate image samples! (low β)



Markov Random Fields: Texture Modeling

Using Gibbs Sampling (later!) we can generate image samples! (high β)

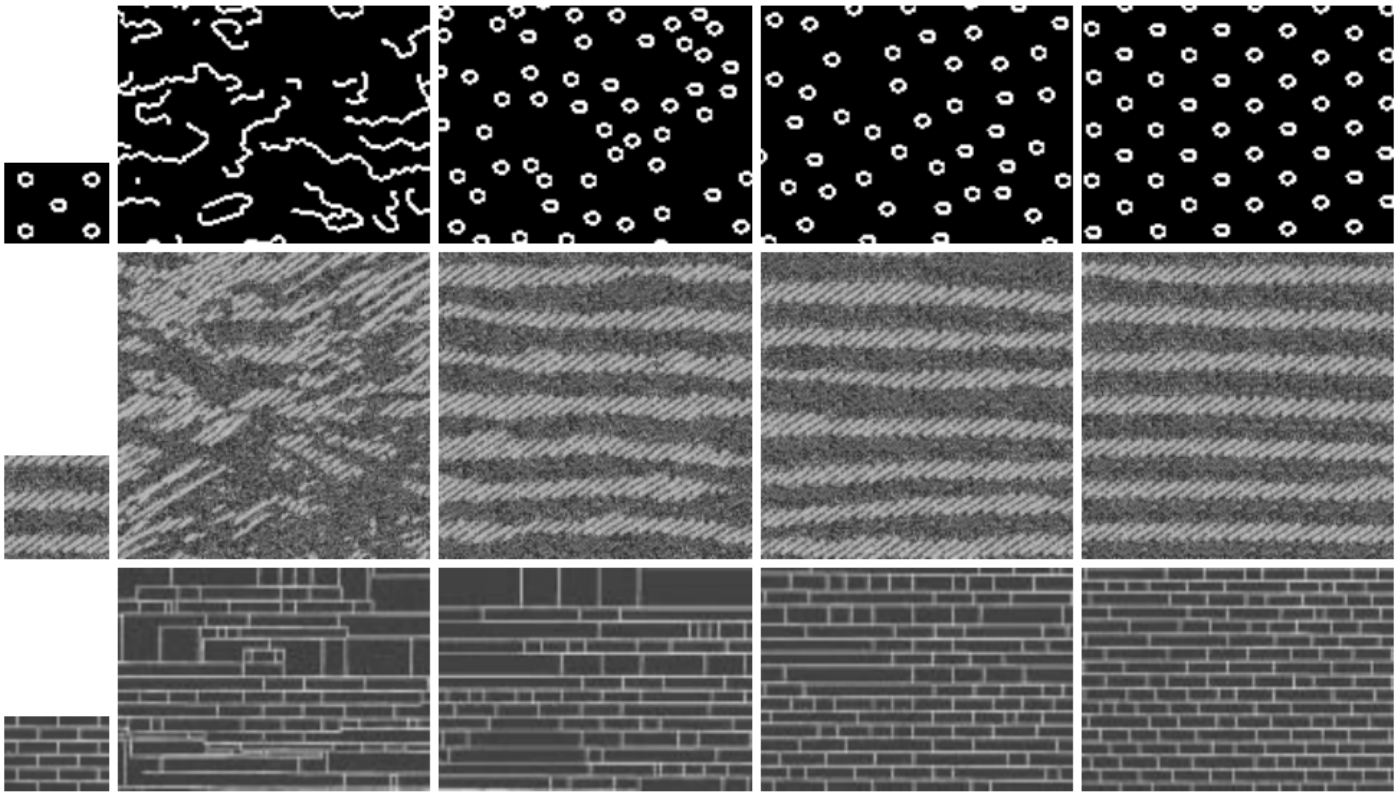
Markov Random Fields: Texture Modeling

We can also model more complex interactions between pixels.

- Force vertical neighbors to agree, horizontal neighbors to disagree.
- Rather than taking direct neighbors, look at a *window* around each pixel.
- Learn different β values for different regions of the image.

Markov Random Fields: Texture Synthesis

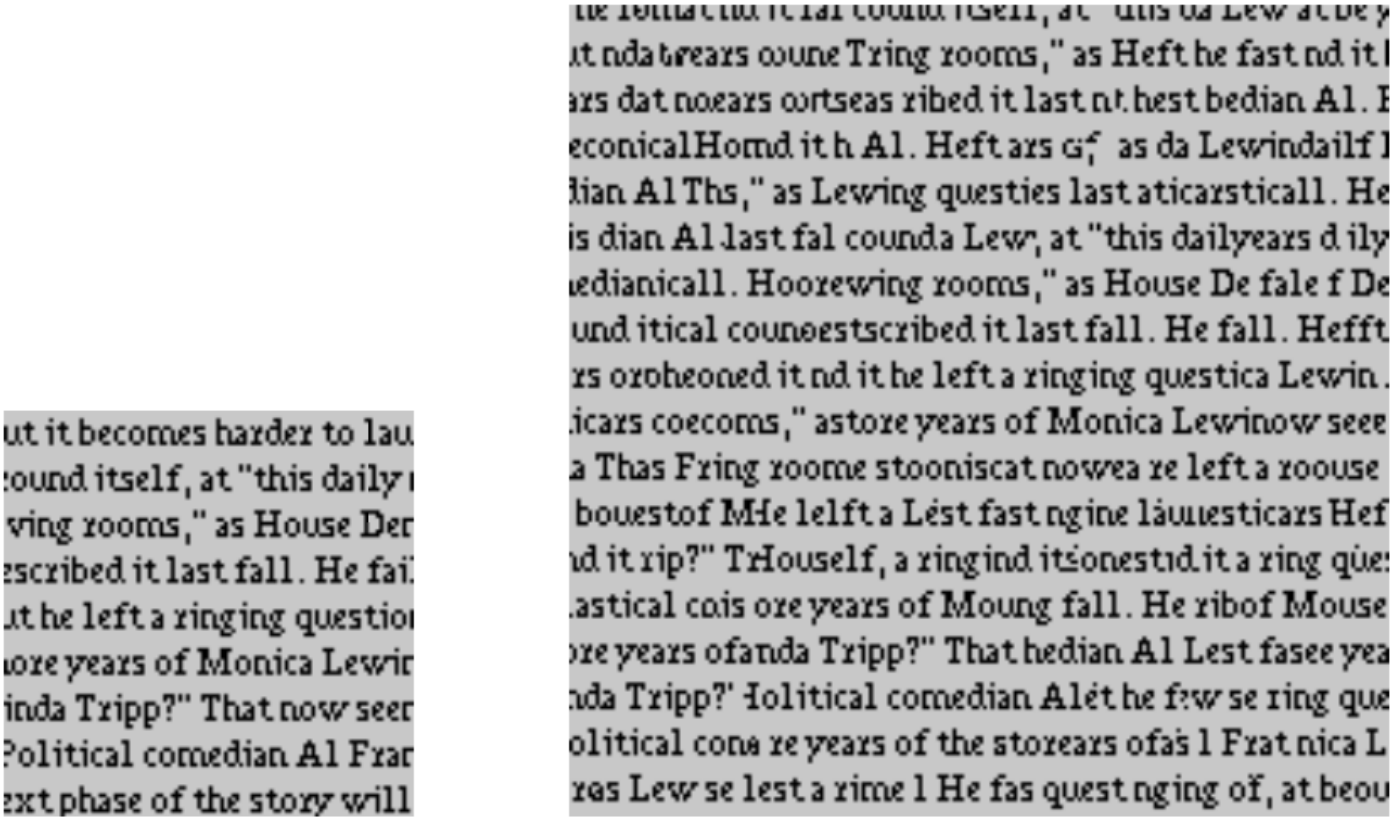
We can learn to sample large textures from a small starting seed:



(Example taken from Efros & Leung, [Texture Synthesis by Non-Parametric Sampling](http://ieeexplore.ieee.org.proxy.lib.umich.edu/stamp/stamp.jsp?tp=&arnumber=790383))

Markov Random Fields: Texture Synthesis

The same algorithm can even be applied to text!



(Example taken from Efros & Leung, [Texture Synthesis by Non-Parametric Sampling](http://ieeexplore.ieee.org.proxy.lib.umich.edu/stamp/stamp.jsp?tp=&arnumber=790383))

Undirected Models: Other Applications

Undirected models are incredibly useful for probabilistic modeling.

- Image Processing
  - Segmentation, Denoising
- Natural Language Processing
  - Named entity recognition
  - Parsing
  - Computational Humor
- Maximum entropy models (related to exponential family)
- Conditional random fields (discriminative model)
- Deep learning

Inference in Graphical Models

> Uses material from **[MLAPP]** Chapters 20, 21

Inference in Graphical Models

Inference: Estimate hidden variables Z from observed variables X.

$$P(Z|X,\theta) = \frac{P(X,Z|\theta)}{P(X|\theta)}$$

- Denominator $P(X|\theta)$ is sometimes called the probability of the **evidence**.
- Occasionally we care only about a subset of the hidden variables, and marginalize out the rest.

Inference: Examples

We have already described the inference procedure for simple models:

- Expectation Maximization
- Mixture Models
- Hidden Markov Models

> Given an arbitrary directed or undirected graph, can we systematically perform inference?

Exact Inference

> Uses material from **[MLAPP]** Chapter 20

Exact Inference

We can recover $P(Z|X,\theta)$ exactly when the hidden variables Z are **discrete**.

- Belief Propagation
- Variable Elimination
- Junction Tree

Many exact inference algorithms can be seen as a generalization of the **fowards-backwards algorithm** for HMMs.

> Unfortunately, they have undesirable time and space complexity properties.

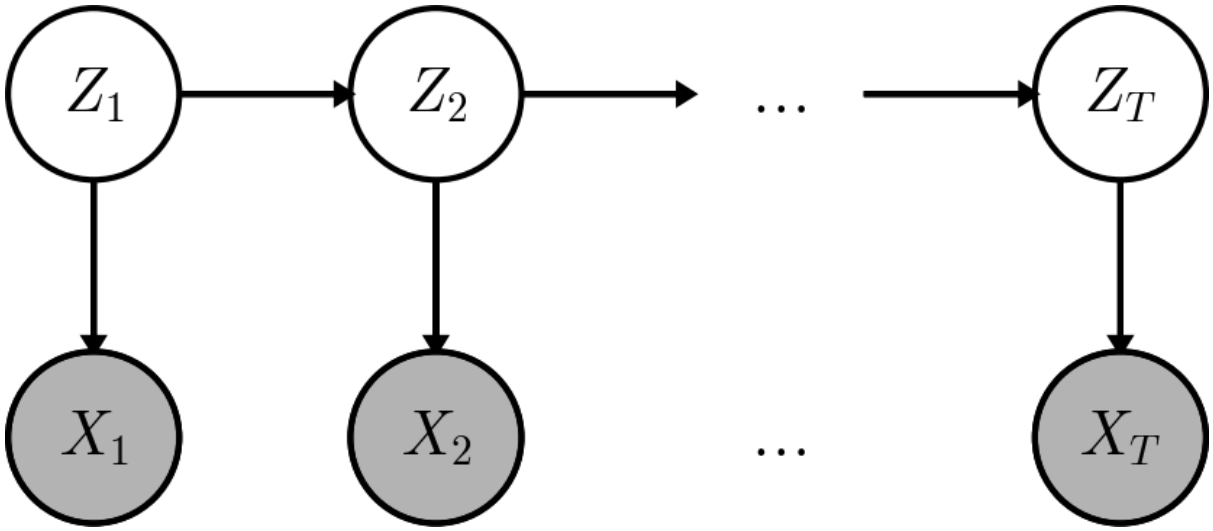
Exact Inference: HMMs

Recall that a **Hidden Markov Model** is a chain-structured directed model:

In [2]:
```
@pgm_render
def pgm_hmm():
    pgm = daft.PGM([7, 7], origin=[0, 0])

    # Nodes
    pgm.add_node(daft.Node("Z1", r"$Z_1$", 1, 3.5))
    pgm.add_node(daft.Node("Z2", r"$Z_2$", 2, 3.5))
    pgm.add_node(daft.Node("Z3", r"$\dots$", 3, 3.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("Z4", r"$Z_T$", 4, 3.5))

    pgm.add_node(daft.Node("x1", r"$X_1$", 1, 2.5, observed=True))
    pgm.add_node(daft.Node("x2", r"$X_2$", 2, 2.5, observed=True))
    pgm.add_node(daft.Node("x3", r"$\dots$", 3, 2.5, plot_params={'ec':'n

    pgm.add_node(daft.Node("x4", r"$X_T$", 4, 2.5, observed=True))


    # Add in the edges.
    pgm.add_edge("Z1", "Z2", head_length=0.08)
    pgm.add_edge("Z2", "Z3", head_length=0.08)
    pgm.add_edge("Z3", "Z4", head_length=0.08)

    pgm.add_edge("Z1", "x1", head_length=0.08)
    pgm.add_edge("Z2", "x2", head_length=0.08)
    pgm.add_edge("Z4", "x4", head_length=0.08)

    return pgm;
```

In [3]:
```
%%capture
pgm_hmm("images/pgm/hmm.png")
```

Out[3]:



Exact Inference: HMMs

We used the **fowards-backwards algorithm** to find $P(Z_t|\mathcal{X}, \theta)$.

- We split the graph at Z_t
- Performed the **foward algorithm** to propagate information from the start state *forwards* to Z_t
- Performed the **backward algorithm** to propagate information *backwards* to Z_t

Belief Propagation: From Chains to Trees

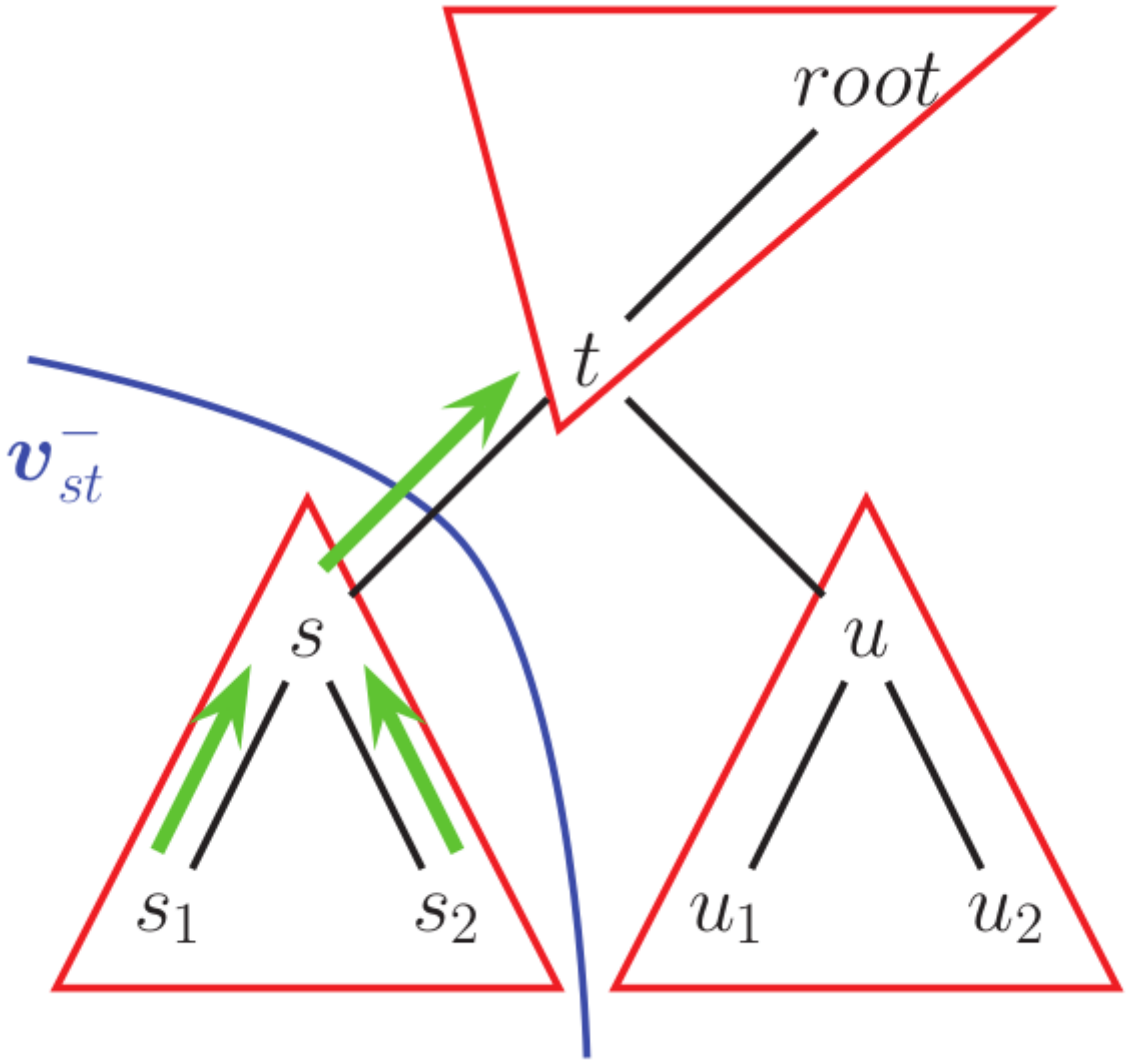
The **Belief Propagation** algorithm generalizes the forwards-backwards algorithm from chains to tree-structured graphs.

- *Idea:* Every path from the root node to a leaf node is a chain!

> Let's do some hand-waving...

Belief Propagation: Collect-to-Root Phase

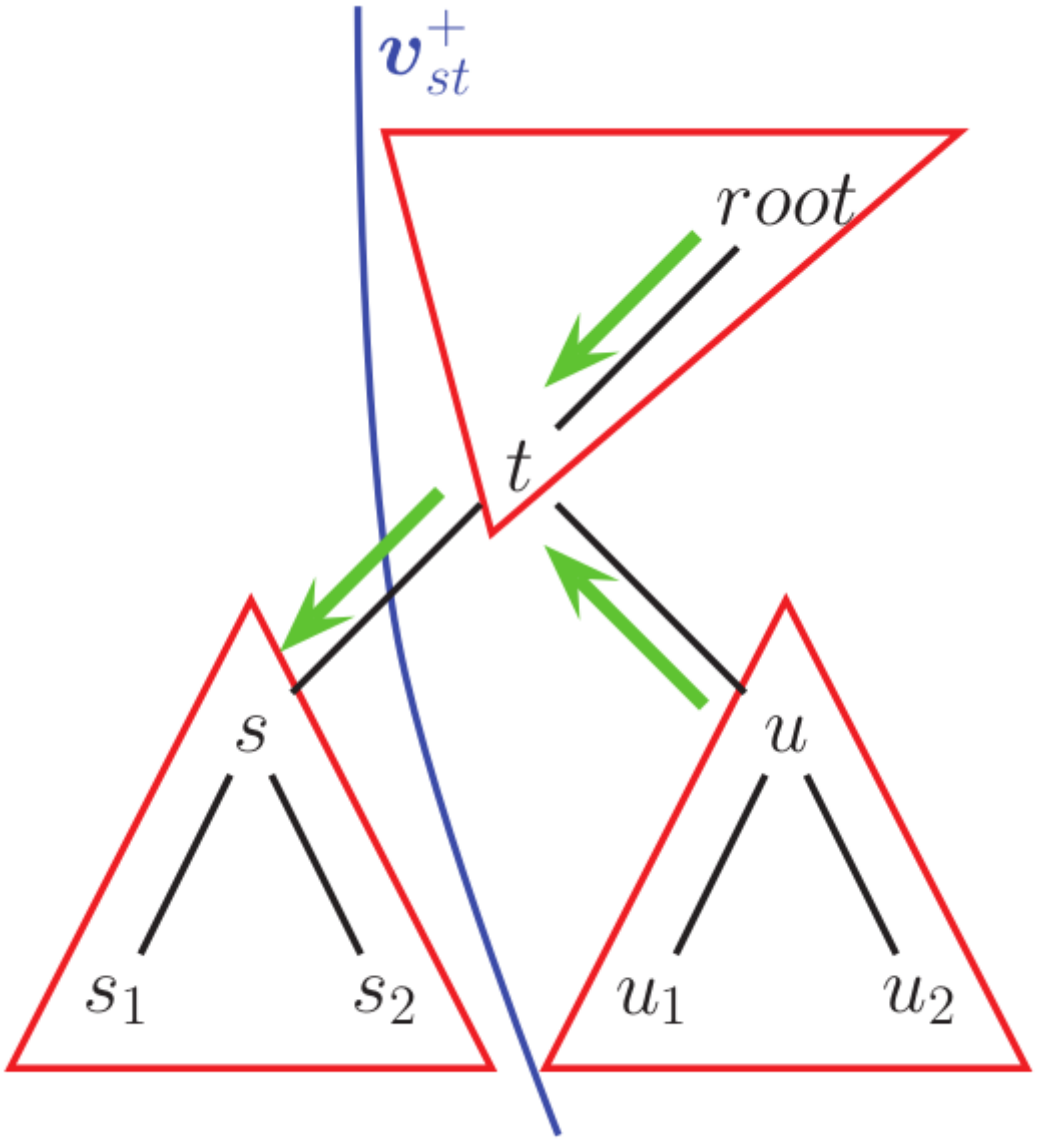
Much like before, we split the tree at node t and propagate evidence upwards:



(Figure 20.1a from **[MLAPP]**)

Belief Propagation: Distribute-from-Root Phase

Next, we propagate the collected evidence back down the tree:



(Figure 20.1b from **[MLAPP]**)

Junction Tree: From Trees to Graphs

The **Junction Tree** algorithm generalizes Belief Propagation from trees to arbitary graphs.

> See **[MLAPP]** or **[Koller & Friedman 2009]** for details.

Exact Inference: Summary

Omitting some slides from lecture -- but still available on notes

Exact inference is possible in discrete graphical models.

- Unfortunately, algorithms to do so are ugly and complicated.
- Moreover, exact inference is NP-Hard, so these algorithms are very inefficient.
- Incapable of dealing with general, continuous distributions.

Fortunately, **approximate inference** works well in practice.

Approximate Inference

> Uses material from **[MLAPP]** Chapter 21 and MLSS 2009 (http://videolectures.net/site/normal_dl/tag=50736/mlss09uk_murray_mcmc.pdf)

Approximate Inference: Intro

Exact inference is too difficult, so we turn to **approximate inference**.

- Theory is lacking--no guarantees about accuracy or runtime
- However, they seem to work well in practice.
- Capable of handling continuous variables!

The two most common methods are

- **Markov Chain Monte Carlo:** Estimate $P(Z|X,\theta)$ by sampling from it!
- **Variational Inference:** Cast inference as a deterministic optimization problem.

Integration is Hard

In general, we want to find $p(X|Y)$ for some variables X and Y. Using Bayes' Rule,

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\int_Y p(x|y)p(y)\,dy}$$

Usually, the hard part is evaluating the normalization integral,

$$\int_Y p(x|y)p(y)\,dy$$

> **Key Point:** Many inference problems can be reduced to the problem of *evaluating an integral*.

Monte Carlo Integration

Suppose we wish to evaluate $\int_X f(x)p(x)\,dx$, where

- $f(x)$ is an arbitrary function
- $p(x)$ is a probability distribution

We can express this as an expectation, with $X \sim p$,

$$\int_{X\sim p} f(x)p(x)\,dx = E_p[f(X)]$$

Monte Carlo Integration

We can approximate any expectation by averaging over samples $X_1, \ldots, X_N \overset{iid}{\sim} p(X)$,

$$\int_X f(x)p(x)\,dx = E_p[f(X)] \approx \bar{f}_N = \frac{1}{N}\sum_{n=1}^{N} f(x_n)$$

> **Key Point:** We can reduce any integration problem to a *sampling problem*.

Example: Dumb Approximation of π

We can express π as an integral over the unit circle C, which has area $\pi r^2 = \pi$.

$$\pi = \int_{\mathbb{R}^2} \mathbb{I}_C(x)\, dx = \int_{[-1,1]\times[-1,1]} \mathbb{I}_C(x)\, dx$$

Let p be the uniform distribution on the square of size 2.

$$p(x) = \begin{cases} \frac{1}{4} & x \in [-1,1] \times [-1,1] \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$\pi = 4 \int_{[-1,1]\times[-1,1]} \mathbb{I}_C(x) p(x)\, dx \approx E_p[4 \cdot \mathbb{I}_C(X)]$$

Example: Dumb Approximation of π

Therefore, we can estimate π by drawing samples uniformly from the square $[-1,1] \times [-1,1]$ and computing the fraction of samples that fall within the unit circle C!

> This method is **not ideal** and requires tons of samples.

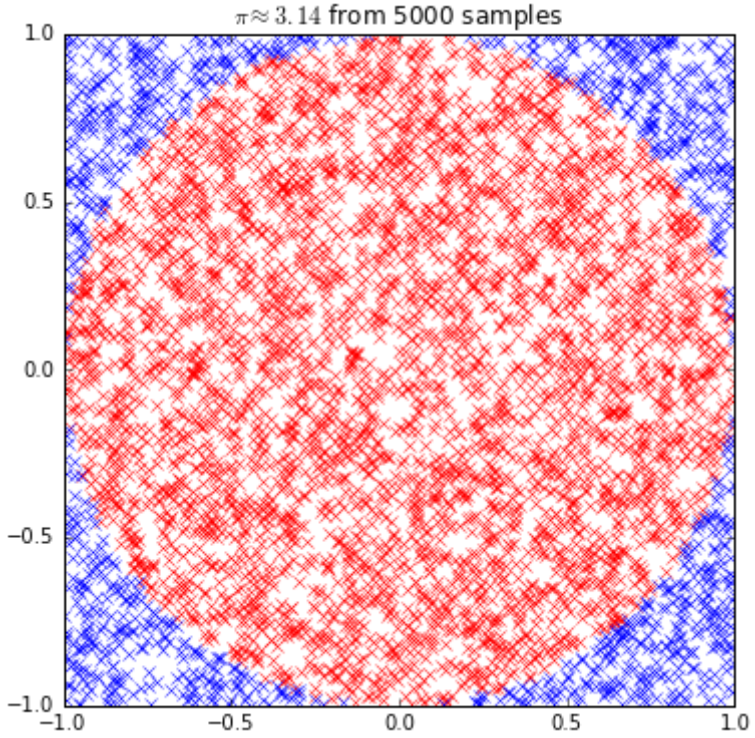
$$\frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

Example: Dumb Approximation of π

In [4]:
```
def estimate_pi(n):
    # draw samples
    samples = np.random.random((2,n)) * 2 - 1;
    # separate samples inside/outside circle
    r = np.sum(samples**2, axis=0);
    inside = samples[:,r < 1];
    outside = samples[:,r >= 1];
    # estimate pi
    return inside.shape[1] / n * 4;

def estimate_pi_plot(n):
    # draw samples
    samples = np.random.random((2,n)) * 2 - 1;
    # separate samples inside/outside circle
    r = np.sum(samples**2, axis=0);
    inside = samples[:,r < 1];
    outside = samples[:,r >= 1];
    # estimate pi
    pi_estimate = inside.shape[1] / n * 4;
    print("pi estimated to be %.5f" % pi_estimate)

    # plot
    plt.figure(figsize=(6,6))
    plt.plot(inside[0,:], inside[1,:], "rx");
    plt.plot(outside[0,:], outside[1,:], "bx");
    plt.axis("equal");
    plt.title(r"$\pi \approx %0.2f$ from %d samples" % (pi_estimate,n));
```

In [5]:
```
estimate_pi_plot(5000)
```

```
pi estimated to be 3.14320
```



Monte Carlo Integration: Sampling

We can compute any integral $\int_X f(x)p(x)\,dx$ provided that we know how to sample from p. Different methods have been devloped to sample from arbitrary distributions:

- Sampling from Inverse CDF
- Rejection Sampling
- Importance Sampling
- Particle Filtering

The most popular is **Markov Chain Monte Carlo**, which beats out the other methods in high-dimensional spaces.

Markov Chain Monte Carlo

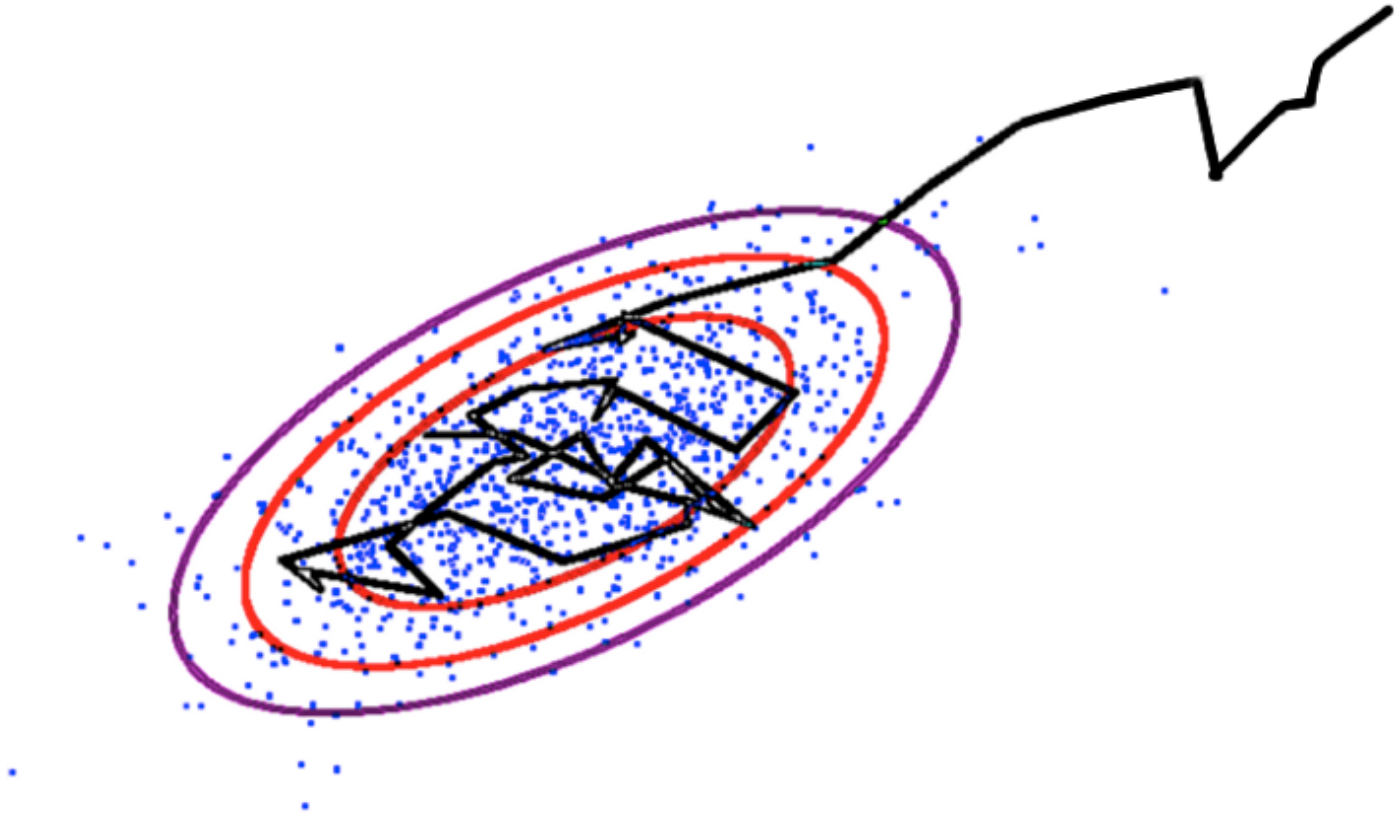
Goal: Draw samples from $P(X_1, \ldots, X_N)$.

- Usually, N is large, so P is high-dimensional.

Idea: Construct a Markov Chain on the state space $\mathcal{X}$ whose *stationary distribution* is the target density.

> We will perform a (biased) random walk on the state space, such that the fraction of time we spend at each state $(x_1, \ldots, x_N)$ is proportional to $p(x_1, \ldots, x_N)$

Markov Chain Monte Carlo



(Figure taken from [MLSS 2009 Slides](http://videolectures.net/site/normal_dl/tag=50736/mlss09uk_murray_mcmc.pdf))

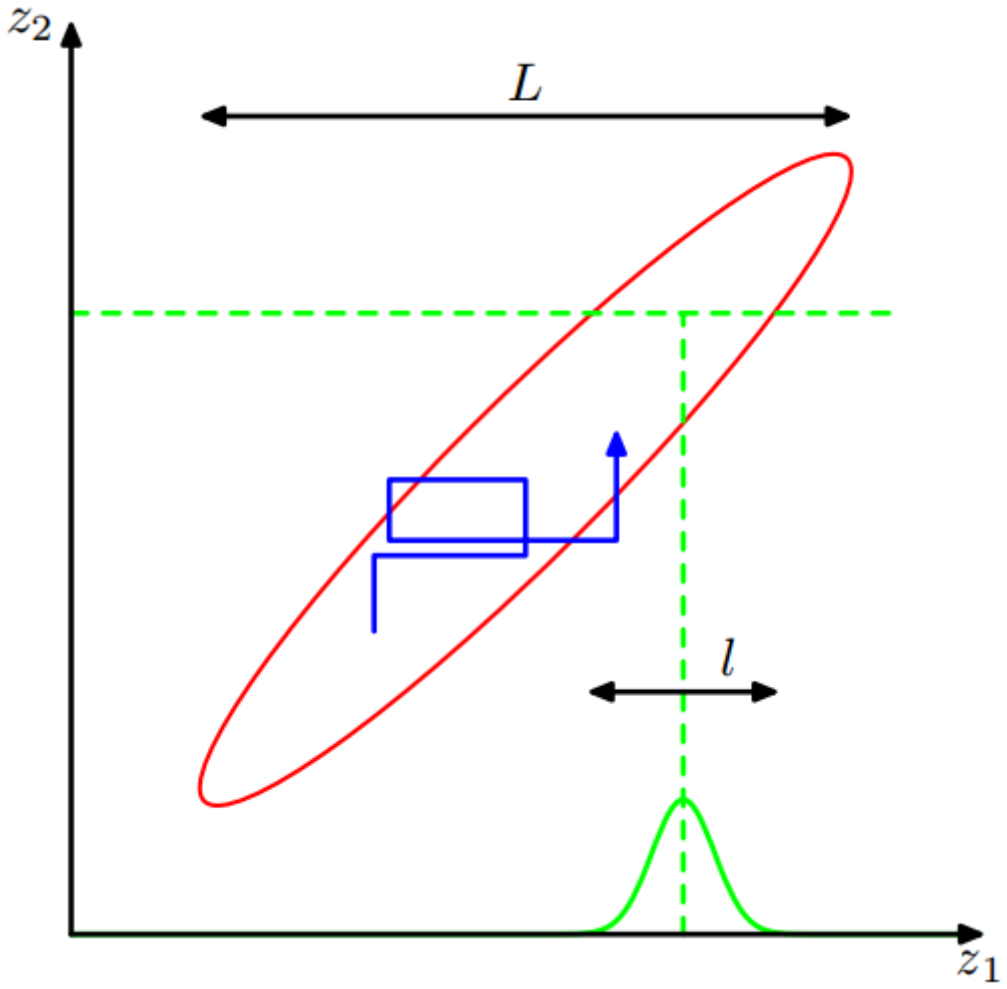
MCMC: Gibbs Sampling

Sample each variable in turn, conditioned on all the others.

- Assumes the **full conditionals** $p(x_k|\mathbf{x}_{-k})$ are known

1. Initialize $\mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_N^0)$ randomly.
2. For $k \in 1, \ldots, N$ in a random order,
   A. Set $\mathbf{x}^{t+1} = \mathbf{x}^t$
   B. Resample $x_k^{t+1} \sim p(x_k \mid \mathbf{x}_{-k}^t)$

MCMC: Gibbs Sampling



(Figure 11.11 from **[PRML]**)

MCMC: Summary

1. Inference can often be reduced to an integration problem.
2. Integration problems can be expressed as an expection.
3. Expectations can be approximated by averaging over samples.
4. **Inference can be reduced to a sampling problem.**

MCMC: Summary

Advantages:

- Easy to implement
- Works well in high dimensions
- Guaranteed to converge to the true distribution

Disadvantages:

- The Markov Chain may not **mix** fast enough
- Need lots of samples
- Convergence rate unknown

Application: Latent Dirichlet Allocation

> This was originally a homework problem, but you lucked out :) See **[Blei, Ng, and Jordan 2003]** (http://ai.stanford.edu/~ang/papers/nips01-lda.pdf) for details.

Latent Dirichlet Allocation

Latent Dirichlet Allocation models documents as mixtures of latent *topics*.

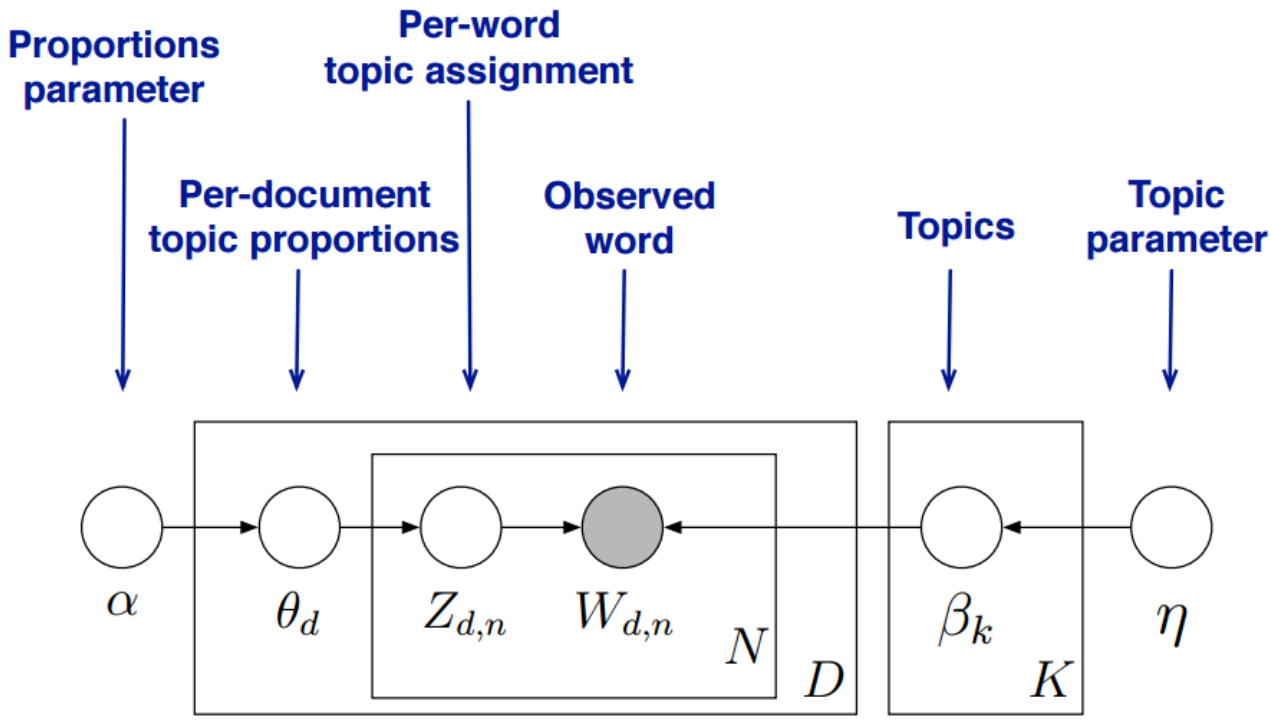
- Makes the **bag-of-words** assumption.
- A **topic** is a probability distribution over words.
- Is a **mixed-membership** mixture model. Each document expresses more than one topic with different *proportions*.

LDA: Generative Process

Given a set $\beta = (\beta_1, \ldots, \beta_K)$ of K topics and a fixed vocabulary, generate documents in the following way:

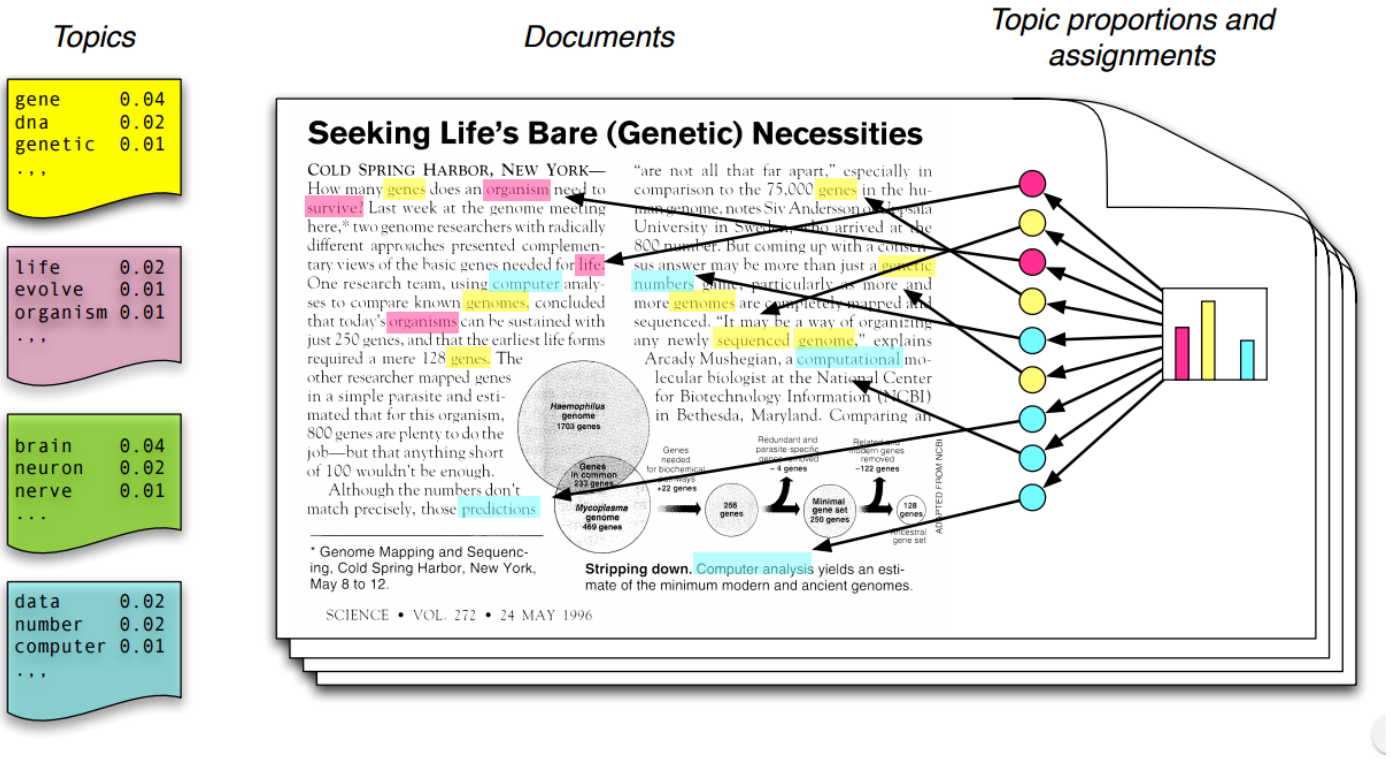
- For document $d = 1, 2, \ldots$
  - Choose topic proportions $\theta_d \sim \text{Dirichlet}(\alpha)$
  - For word position $n = 1, 2, \ldots$,
    - Sample topic $z_{dn} \sim \text{Categorical}(\theta_d)$
    - Sample word $w_{dn} \sim \text{Categorical}(\beta_{z_{dn}})$

LDA: Graphical Model



(Taken from [MLSS 2012 Slides](http://yosinski.com/mlss12/media/slides/MLSS-2012-Blei-Probabilistic-Topic-Models.pdf) by David Blei)

LDA: Intuitive View



(Taken from [MLSS 2012 Slides](http://yosinski.com/mlss12/media/slides/MLSS-2012-Blei-Probabilistic-Topic-Models.pdf) by David Blei)

LDA: Example of Learned Topics

human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

(Taken from [MLSS 2012 Slides](http://yosinski.com/mlss12/media/slides/MLSS-2012-Blei-Probabilistic-Topic-Models.pdf) by David Blei)

LDA: Inference

Latent Dirichlet Allocation is arguably the simplest model that requires approximate inference. It is often used as the "hello world" example when new inference techniques are introduced.

- Mean field variational inference
- Expectation propagation
- Collapsed Gibbs sampling
- Distributed sampling
- Collapsed variational inference
- Online variational inference
- Factorization based inference

Latent Dirichlet Allocation has **many, many** applications to areas other than text modeling.