Undirected Graphical Models

CPSC 440/550: Advanced Machine Learning

cs.ubc.ca/~dsuth/440/23w2

University of British Columbia, on unceded Musqueam land

2023-24 Winter Term 2 (Jan–Apr 2024)

Last Time

- DAG models factorize joint distribution into product of conditionals
  - Usually we assume conditionals depend on small number of "parents"
  - Most models we've seen can be represented as DAGs
  - Plate notation helps us do this efficiently

- D-separation allows us to test conditional independences based on a graph
  - Conditional independence follows if all undirected paths are "blocked"
  - Observed values in chain or parent block paths
  - Unobserved children (with no observed grandchildren) also blocks paths

Multivariate Gaussians as DAGs

- Remember the general multivariate Gaussian density:

$$
\begin{aligned}
p(x_1, \ldots, x_d) &\propto \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right) \\
&= \exp\left(-\frac{1}{2}\sum_{j=1}^{d}\sum_{j'=1}^{d}(x_j-\mu_j)(\boldsymbol{\Sigma}^{-1})_{jj'}(x_{j'}-\mu_{j'})\right) \\
&= \prod_{j=1}^{d}\left(e^{-\frac{1}{2}(\boldsymbol{\Sigma}^{-1})_{jj}(x_j-\mu_j)^2}\prod_{j'<j}e^{-(\boldsymbol{\Sigma}^{-1})_{jj'}(x_j-\mu_j)(x_{j'}-\mu_{j'})}\right)
\end{aligned}
$$

- x_j connects to every previous $x_{j'}$ where $(\boldsymbol{\Sigma}^{-1})_{jj'} \neq 0$
- If the precision $\boldsymbol{\Sigma}^{-1}$ is sparse, can imply conditional independence properties
- But this ordering is kind of unnatural; easier to think about without it...

Undirected Graphical Models (UGMs)

- Undirected graphical models (UGMs) are another popular graphical model class
  - Also called Markov random fields

- UGMs define joint distribution in terms of non-negative potential functions,

$$p(x_1, x_2, \ldots, x_d) \propto \prod_{c \in \mathcal{C}} \phi_c(x_c)$$

- Define a potential ϕ_c for each set c where we want to model a direct relationship

- The most common choice is a pairwise UGM,

$$p(x_1, x_2, \ldots, x_d) \propto \left(\prod_{j=1}^{d} \phi_j(x_j) \right) \left(\prod_{(j,j') \in \mathcal{E}} \psi_{jj'}(x_j, x_{j'}) \right)$$

  - This only has potentials on single variables (ϕ) and pairs of variables (ψ)
  - The "edge potentials" ψ are defined on edges of an undirected graph $\mathcal{E}$

Pairwise Undirected Graphical Models

- Pairwise undirected graphical models factorize probability using

$$p(x_1, x_2, \ldots, x_d) \propto \left(\prod_{j=1}^{d} \phi_j(x_j) \right) \left(\prod_{(j,j') \in \mathcal{E}} \psi_{jj'}(x_j, x_{j'}) \right)$$

- For example: multivariate Gaussians

$$\phi_j(x_j) = e^{-\frac{1}{2}(\mathbf{\Sigma}^{-1})_{jj}(x_j - \mu_j)^2} \quad \psi_{jj'}(x_j, x_{j'}) = e^{-(\mathbf{\Sigma}^{-1})_{jj'}(x_j - \mu_j)(x_{j'} - \mu_{j'})}$$

- Also Markov chains: edges only between adjacent nodes
- Ising model for $x_j \in \{-1, 1\}$ uses

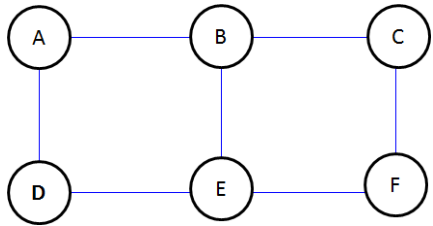
$$\phi_j(x_j) = \exp(x_j w_j) \quad \phi_{jj'}(x_j, x_{j'}) = \exp(x_j x_{j'} w_{jj'})$$

where w_i is the node weight and w_{ij} is the edge weight

  - If $w_{jj'} > 0$ it encourages neighbours to have same value ("attractive")
  - If $w_{jj'} < 0$ it encourages neighbours to have different values ("repulsive")

Conditional Independence in UGMs

- A UGM's independence properties are described by an undirected graph
  - For pairwise UGMs, the edges are given by the set of edges $\mathcal{E}$



- If you have 3-variable or higher-order potentials:
  - Add an edge (j, j') if j and j' are together in at least one c
- So these two factorizations have the same graph:

$$p(x_1.x_2, x_3) \propto \phi_{12}(x_1, x_2)\phi_{13}(x_1, x_3)\phi_{23}(x_2, x_3), \quad p(x_1, x_2, x_3) \propto \phi_{123}(x_1, x_3, x_3)$$

- UGM implies $A \perp\!\!\!\perp B \mid C$ if C separates all nodes in A from all nodes in B

Graphical LASSO

bonus!

- Conditional independence structure in Gaussians given by sparsity of $\mathbf{\Theta} = \mathbf{\Sigma}^{-1}$
- Popular way to estimate covariances adds L1 penalty to the precision:

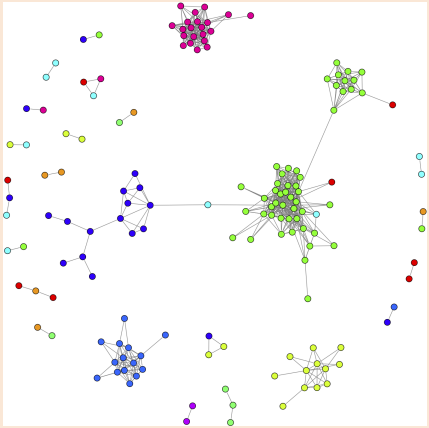
$$\arg\min_{\mathbf{\Theta}} \underbrace{\mathrm{Tr}(\mathbf{S\Theta}) - \log|\mathbf{\Theta}|}_{\text{MLE objective}} + \lambda \sum_{j=1}^{d} \sum_{j'=1}^{d} \left|\mathbf{\Theta}_{jj'}\right|$$

  - With specialized optimization algorithms, gives sparse off-diagonals of $\mathbf{\Theta}$
  - "Assume conditional independence unless there's good reason not to"
  - Learns a sparse graph for the UGM

Graphical LASSO Example

bonus!

- Graphical LASSO applied to stocks data:

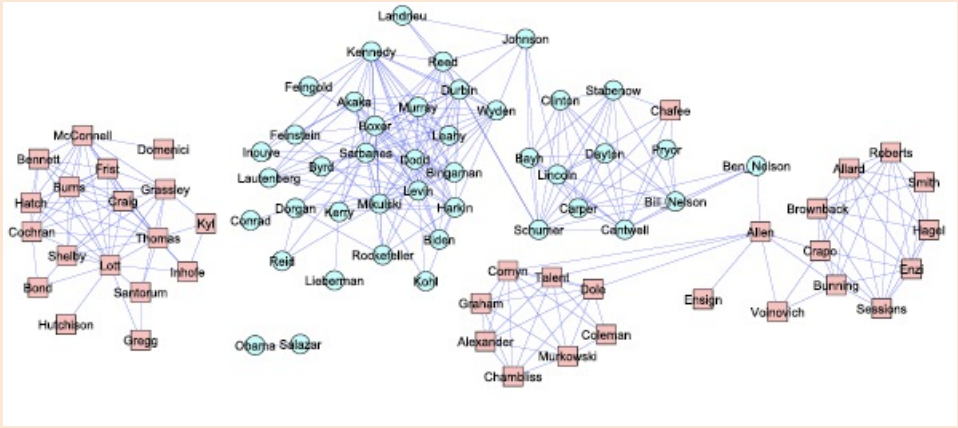


https://normaldeviate.wordpress.com/2012/09/17/high-dimensional-undirected-graphical-models

Graphical LASSO Example

bonus!

- Graphical LASSO applied to US senate voting data (Bush junior era):

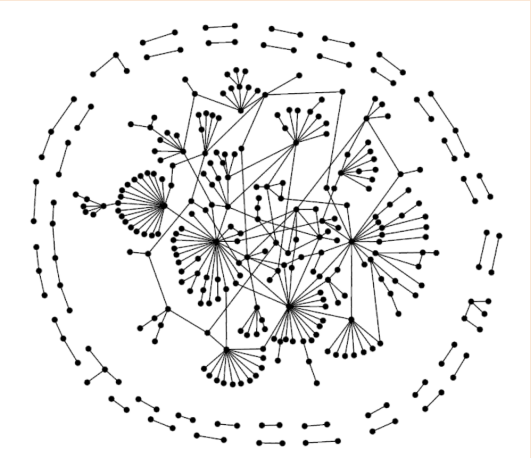


https://normaldeviate.wordpress.com/2012/09/17/high-dimensional-undirected-graphical-models

Graphical LASSO Example

bonus!

- Graphical LASSO applied to protein data:

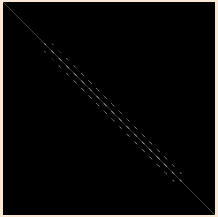


https://normaldeviate.wordpress.com/2012/09/17/high-dimensional-undirected-graphical-models

Graphical LASSO on Digits

bonus!

- Precision matrix from graphical LASSO applied to MNIST digits ($\lambda = 1/8$):

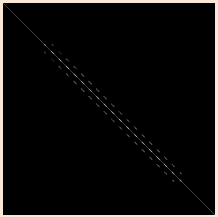


- To understand this picture, first the size of the precision matrix:
  - The images of digits, which are $m \times m$ matrices (m pixels by m pixels)
    - This gives $d = m^2$ elements of $x^{(i)}$, which we'll assume are in "column-major" order.
    - Frist m elements of $x^{(i)}$ are column 1, next m elements are columm 2, and so on.
  - The picture above, which is $d \times d$ so will thus be $m^2 \times m^2$

Graphical LASSO on Digits

bonus!

- Precision matrix from graphical LASSO applied to MNIST digits ($\lambda = 1/8$):



- So what are the non-zeroes in the precision matrix?
  1. The diagonals $\Theta_{j,j}$ (positive-definite matrices must have positive diagonals)
  2. The first off-diagonals $\Theta_{j,j+1}$ and $\Theta_{j+1,j}$
     - This represents the dependencies between adjacent pixels vertically
  3. The $(m+1)$ off-diagonals $\Theta_{j,j+m}$ and $\Theta_{j+m,j}$
     - This represents the dependencies between adjacent pixels horizontally
     - Because in "column-major" order, you go "right" a pixel every m indices

DAGs vs. UGMs

- Neither DAGs or UGMs are "more powerful" than the other
  - Any distribution can be written as a DAG, and as a UGM
  - But you might need to use a highly connected graph

- Set of independences in DAG cannot always be written as UGM (and vice versa)
  - UGMs cannot reflect independences in common child graph: $(x) \rightarrow (y) \leftarrow (z)$
  - DAGs cannot reflect independences in 4-node loop: $(x) - (y) - (z) - (x)$
  - Independences representable as both DAGs and UGMs are called decomposable.
    - An example is Markov chains: independences are same in DAG and UGM graphs

- DAGs are often used when it makes sense to work with conditionals, or we have an idea of causal directions

- UGMs are often used when there are no obvious directions (like MNIST), and are more often used when we want to add features to do supervised learning

Tractability of UGMs

- Without using $\propto$, a UGM probability would be

$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(x_c),$$

where Z is the constant that makes the probabilites sum up to 1

$$Z = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_d} \prod_{c \in \mathcal{C}} \phi_c(x_c) \quad \text{or} \quad Z = \int_{x_1} \int_{x_2} \cdots \int_{x_d} \prod_{c \in \mathcal{C}} \phi_c(x_c) \mathrm{d}x_d \mathrm{d}x_{d-1} \cdots \mathrm{d}x_1$$

- Whether you can compute Z (and do inference) depends on the choice of the ϕ_c:
  - Gaussian case: $\mathcal{O}(d^3)$ in general, but $\mathcal{O}(d)$ for forests (no loops)
  - Continuous non-Gaussian: usually requires approximate inference
  - Discrete case: $\#$P-hard in general, but $\mathcal{O}(dk^2)$ for forests (no loops)

Discrete DAGs vs. Discrete UGMs

- Common inference tasks in graphical models:
  1. Compute $p(x)$ for an assignment to the variables x
  2. Generate a sample x from the distribution
  3. Compute univariate marginals $p(x_j)$
  4. Compute decoding $\arg\max_x p(x)$
  5. Compute univariate conditional $p(x_j \mid x_{j'})$

- With discrete x_j, all of the above are easy in tree-structured graphs
  - For DAGs, a tree-structured graph has at most one parent
  - For UGMs, a tree-structured graph has no cycles

- With discrete x_j, the above may be harder for general graphs:
  - In DAGs the first two are easy, the others are NP-hard
  - In UGMs all of these are NP-hard

Inference in UGMs

- We're not going to cover this, but there are lots of bonus slides
- Gibbs sampling was invented to do approximate inference in UGMs
- Efficient exact inference is possible in graphs with low "treewidth"
  - Versions of the forward-backward algorithm we covered for Markov chains
  - But cost is exponential in treewidth

Outline

1. Undirected Graphical Models (UGMs)
2. Log-Linear Models
3. Conditional Random Fields
4. Hidden Markov Models

Vancouver Rain Data: DAG vs. UGM

- We previously considered the "Vancouver Rain" dataset:

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	...
Month 1	0	0	0	1	1	0	0	1	1	
Month 2	1	0	0	0	0	0	1	0	0	
Month 3	1	1	1	1	1	1	1	1	1	
Month 4	1	1	1	1	0	0	1	1	1	
Month 5	0	0	0	0	1	1	0	0	0	
Month 6	0	1	1	0	0	0	0	1	1	

- We previously fit this with a Markov chain under the DAG factorization:

$$p(x_1, x_2, \ldots, x_d) = p(x_1) \prod_{j=2}^{d} p(x_j \mid x_{j-1})$$

using tabular potentials (so learning was counting)

Vancouver Rain Data: DAG vs. UGM

- Consider fitting a Markov chain under a UGM factorization:

$$p(x_1, x_2, \ldots, x_d) \propto \left(\prod_{j=1}^{d} \phi_j(x_j)\right)\left(\prod_{j=2}^{d} \phi_{j,j-1}(x_j, x_{j-1})\right)$$

- We could use the following UGM parameterization (for $x_j \in \{-1, +1\}$):

$$\phi_j(x_j) = \exp(w_j x_j) \quad \phi_{ij}(x_i, x_j) = \exp(v_{ij} x_i x_j)$$

where w_j is a node weight, v_{ij} is an edge weight, and we're using Ising edges

  - The exponential function makes the potentials non-negative
    - We call this a log-linear model: logarithms of potentials are linear
  - Ising potentials can reflect how strongly neighbours are attracted/repulsed
  - For the rain data, we would expect $v_{ij} > 0$ (adjacent days likely to have same value)
  - For the rain data, it makes sense to tie w_j across j and v_{ij} across (i, j) values

Vancouver Rain Data: DAG vs. UGM

- Our log-linear model of the rain data under the Ising parameterization:

$$
\begin{aligned}
p(x_1, x_2, \ldots, x_d \mid w, v) &\propto \left(\prod_{j=1}^{d} \exp(w x_j) \right) \left(\prod_{j=2}^{d} \exp(v x_j x_{j-1}) \right) \\
&= \exp\left(\sum_{j=1}^{d} w x_j + \sum_{j=2}^{d} v x_j x_{j-1} \right) \\
&= \exp\left(w \sum_{j=1}^{d} x_j + v \sum_{j=2}^{d} x_j x_{j-1} \right) \\
&= \exp\left(\begin{bmatrix} w \\ v \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \sum_{j=1}^{d} x_j \\ \sum_{j=2}^{d} x_j x_{j-1} \end{bmatrix} \right).
\end{aligned}
$$

- This is an exponential family in canonical form!
  - NLL will be convex in terms of w and v; derivative of NLL has simple form
  - If we didn't tie parameters, we'd have a statistic for each time point

Learning Log-Linear Model for Vancouver Rain Data

- Canonical form: $p(x \mid w, v) \propto \exp\left(\begin{bmatrix} w \\ v \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \sum_{j=1}^{d} x_j \\ \sum_{j=2}^{d} x_j x_{j-1} \end{bmatrix}\right)$
  - Sufficient statistics $s_1(x) = \sum_{j=1}^{d} x_j$, $s_2(x) = \sum_{j=2}^{d} x_j x_{j-1}$
- We derived in general for canonical-form exponential families that

$$\nabla_\theta[-\log p(\mathbf{X} \mid \theta)] = -\sum_{i=1}^{n} s(x^{(i)}) + n\,\mathbb{E}[s(X) \mid \theta]$$

  - Can't solve analytically here... but we can just run gradient descent!
- We have $\mathbb{E}[s(X) \mid w, v] = \begin{bmatrix} \sum_{j=1}^{d} 2\,(p(x_j = 1 \mid w, v) - 1) \\ \sum_{j=2}^{d} (2p(x_j = x_{j-1} \mid w, v)) \end{bmatrix}$
  - Can compute all of these marginals with forward-backward
  - Could also compute $\log Z$ and use autodiff

Learning Log-Linear Models (in general)

- We often write log-linear UGMs in an exponential family form

$$p(x \mid w) = \frac{\exp\left(w^{\mathsf{T}} F(x)\right)}{Z(w)}$$

  where the feature functions $F(x)$ count the number of times we use each w_j
  - Examples of feature functions, and potentials for categoricals, in bonus slides

- Feature functions are just sufficient statistics, so

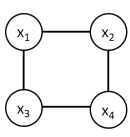
$$\nabla_w[-\log p(\mathbf{X} \mid w)] = -\sum_{i=1}^{n} F(x^i) + n\,\mathbb{E}[s(X) \mid w]$$

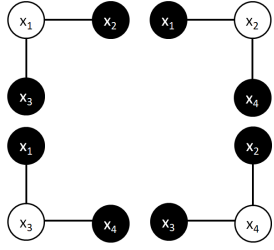
- Computing this requires inference, which is #P-hard in general graphs
  - So we need to consider approximations when learning

Approximate Learning: Pseudo-Likelihood

- A popular approximation to the NLL is pseudo-likelihood ("fast, convex, crude")
- Pseudo-likelihood turns learning into d single-variables problem (similar to DAGs):

$$p(x_1, x_2, \ldots, x_d) \approx \prod_{j=1}^{d} p(x_j \mid x_{\neg j}) = \prod_{j=1}^{d} p(x_j \mid x_{\text{nei}(j)})$$





Approximate Learning: Marginal Approximations

- Another way to approximate the NLL is with approximate inference
  1. Deterministic variational approximations of $\mathbb{E}[F(x)]$
     - Approximate p by a simpler q, and compute expectation for q
  2. Monte Carlo approximation of $\mathbb{E}[F_j(x)]$ given current parameters w:

$$\begin{aligned}\nabla f(w) &= -F(\mathbf{X}) + \mathbb{E}[F(x)] \\ &\approx -F(\mathbf{X}) + \underbrace{\frac{1}{t}\sum_{i=1}^{t} F(x^{(i)})}_{\text{Monte Carlo approx}}\end{aligned}$$

     based on samples from $p(x \mid w)$
     - Unfortunately, we usually can't sample efficiently...

Approximate Learning with MCMC Marginal Approximation

- An innefficient approach to using an MCMC approximation of gradient:
  1. At iteration t, we want to sample from $p(x \mid w^{(k)})$
     - Start from some $x^{(k,0)}$, sample $x^{(k,1)}$, sample $x^{(k,2)}$, etc from an MCMC chain for $w^{(k)}$
     - Treat the last sample $x^{(k,T)}$ from the Markov chain as a sample from $p(x \mid w^{(k)})$
  2. Update the parameters using $x^{(k,T)}$ to get a gradient estimate (sample size 1),

$$w^{(k+1)} = w^{(k)} + \alpha_k(F(\mathbf{X}) - F(x^{(k,T)}))$$

- If we run MCMC long enough, converges via standard SGD arguments
  - But have to run MCMC on each iteration of the SGD method

Younes Algorithm ("Persistent Contrastive Divergence")

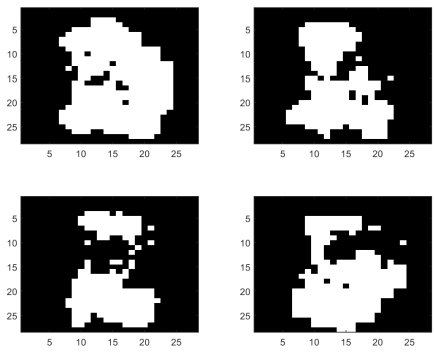
- Younes algorithm (also known as "persistent contrastive divergence"):
  1. At iteration k, we want to sample from $p(x \mid w^k)$
     - Set $x^{(k,0)} = x^{(k-1,T)}$, sample $x^{(k,1)}$, sample $x^{(k,2)}$, and so on
     - Treat the last sample $x^{(k,T)}$ from the Markov chain as a sample from $p(x \mid w^{(k)})$
  2. Update the parameters using $x^{(k)}$ to get a gradient estimate,

$$w^{(k+1)} = w^{(k)} + \alpha_k(F(\mathbf{X}) - F(x^{(k,T)}))$$

- In Younes algorithm, you don't need to run the Markov chain to stationarity
  - Usually you only run MCMC for 1 or a small number of iterations
  - This gives a biased estimate, but is much faster than running MCMC to stationarity
  - With small-enough step-size, can show convergence

Pairwise UGM on MNIST Digits

- Samples from a lattice-structured pairwise UGM trained on MNIST:



- Training: 100k stochastic gradient w/ Gibbs sampling steps with $\alpha_t = 0.01$
- Samples are iteration 100k of Gibbs sampling with fixed w

Outline

1. Undirected Graphical Models (UGMs)
2. Log-Linear Models
3. Conditional Random Fields
4. Hidden Markov Models

Motivation: Rain Data with Month Information

- Our Ising UGM model for the rain data with tied parameters was

$$p(y_1, y_2, \dots, y_k \mid w, v) \propto \exp\left(\sum_{c=1}^{k} w y_c + \sum_{c=2}^{k} v y_c y_{c-1}\right);$$

  we switched variable names from x_j to y_c (but model is same)

- First term reflects that "not rain" is more likely
- Second term reflects that consecutive days are more likely to be the same
  - This model is equivalent to a Markov chain model
- But the model doesn't know that some months are less rainy

- We can add features that reflect the month (or other information)
  - Multi-label supervised learning, but modeling dependence in labels y_c
  - Adding fixed features to a UGM is also called a conditional random field (CRF)

Conditional Random Field (CRF) for Rain Data

- A CRF model of rain data, conditioned on 12 "one of k" month features x_j,

$$p(y_1, y_2, \ldots, y_k \mid x, w_0, w, v) \propto \exp\left(\sum_{c=1}^{k} w_0 y_c + \sum_{c=2}^{k} v y_c y_{c-1} + \sum_{c=1}^{k} y_c w^\top x\right)$$

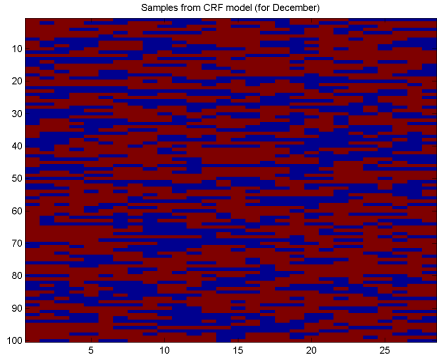
- The potentials in this model over the random variables y_c are

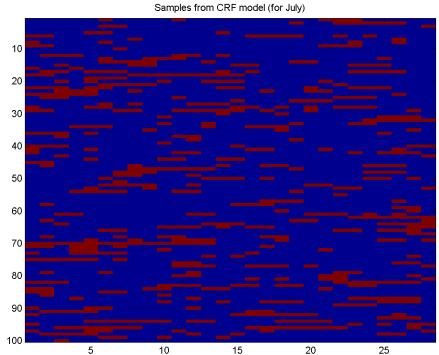
$$\phi_i(y_i) = \exp\left(w_0 y_i + y_i w^\top x\right), \quad \phi_{ij}(y_i, y_j) = \exp(v y_i y_j)$$

- If we draw the UGM over y_c variables we get a chain structure
  - So inference can be done using forward-backward
  - And it's still log-linear so the NLL will be convex
    - Gradient descent finds global optimum jointly with respect to w_0, w, and v

Rain Data with Month Information

- Samples from CRF conditioned on x being December (left) and July (right):





- Conditional NLL is 16.21, compared to Markov chain which gets NLL 16.81.
  - Mark has Matlab (:/) code for this and a variety of other UGM models: `https://www.cs.ubc.ca/~schmidtm/Software/UGM.html`

Conditional Random Fields (General Case)

- We often write the likelihood for general CRFs in the form

$$p(y \mid x, w) = \frac{1}{Z(x, w)} \exp(w^\mathsf{T} F(x, y))$$

for some parameters w and features $F(x, y)$

- The NLL is convex; for a single (x, y) it's

$$-\log p(y \mid x, w) = -w^\mathsf{T} F(x, y) + \log Z(x, w)$$

and the gradient is

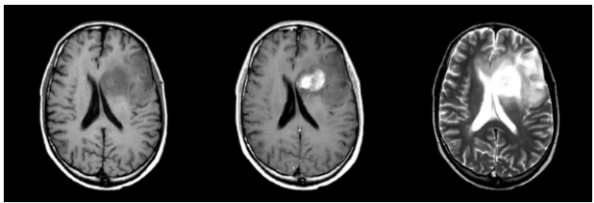
$$-\nabla \log p(y \mid x, w) = -F(x, y) + \mathop{\mathbb{E}}_{y|x,w}[F(x, y)]$$

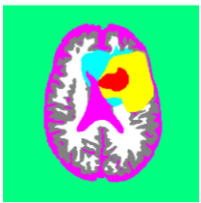
This requires inference for each value of x in training data

- For rain data, need to do run forward-backward 12 times
- If each example has its own features, need to run it n times
- Can make sense to use stochastic gradient if n is large

Motivation: Image Segmentation

- Task: identification of tumours in multi-modal MRI





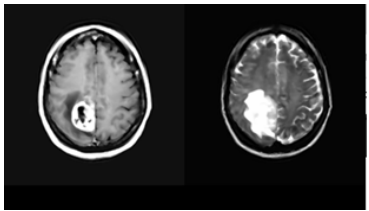
- Applications:
  - Radiation therapy target planning, quantifying treatment response
  - Mining growth patterns, image-guided surgery
- Challenges:
  - Variety of tumor appearances, similarity to normal tissue
  - "You are never going to solve this problem"

Segmentation with Label Dependencies

- After a lot pre-processing and feature engineering (convolutions, priors, etc.), final system used logistic regression to label each pixel as "tumour" or not

$$p(y_c \mid x_c) = \frac{1}{1+\exp(-2y_c w^T x_c)} = \frac{\exp(y_c w^T x_c)}{\exp(w^T x_c)+\exp(-w^T x_c)}$$

- Gives a high "pixel-level" accuracy, but sometimes gives silly results:



- Classifying each pixel independently misses dependence in labels $y^{(i)}$:
  - We prefer neighbouring voxels to have the same value

Segmentation with Label Dependencies

- With independent logistic, conditional distribution over all labels in one image is

$$p(y_1, y_2, \ldots, y_k \mid x_1, x_2, \ldots, x_k) = \prod_{c=1}^{k} \frac{\exp(y_c w^\mathsf{T} x_c)}{\exp(w^\mathsf{T} x_c) + \exp(-w^\mathsf{T} x_c)}$$
$$\propto \exp\left(\sum_{c=1}^{d} y_c w^\mathsf{T} x_c\right)$$

  Here x_c is the feature vector for position c in the image

- We can view this as a log-linear UGM with no edges,

$$\phi_c(y_c) = \exp(y_c w^\mathsf{T} x_c)$$

  Given the x_c, there is no dependence between the y_c

Segmentation with Label Dependencies

- Adding an Ising-like term to model dependencies between y_c gives

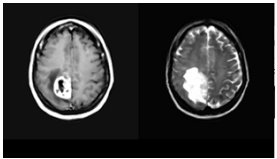
$$p(y_1, y_2, \ldots, y_k \mid x_1, x_2, \ldots, x_k) \propto \exp\left(\sum_{c=1}^{k} y_c w^\top x_c + \sum_{(c,c')\in\mathcal{E}} y_c y_{c'} v\right)$$

- Now we have the same "good" logistic regression model,
  but v controls how strongly we want neighbours to be the same

- We can run gradient descent to jointly optimize w and v (convex NLL)
  - So we find the optimal joint logistic regression and Ising model

Conditional Random Fields for Segmentation

- Recall the performance with the independent classifier:



- The pairwise CRF better modelled the "guilt by association":
  - Trained with pseudo-likelihood, constraining $v \geq 0$
  - Decoding with "graph cuts" (bonus slides)

(Using edge features $x_{cc'}$ too (bonus slides), and different λ on edges)

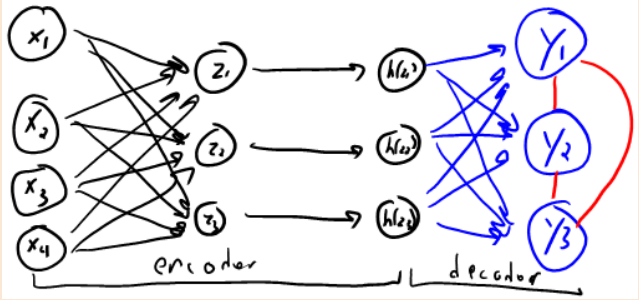
Combining Neural Networks and UGMs

bonus!

- Instead of fixed features, you could use a neural network:

$$p(y \mid x) \propto \exp\left(\sum_{c=1}^{k} y_c v^\top h(W^3 h(W^2(W^1 x_c))) + \sum_{(c,c')\in\mathcal{E}} u y_c y_{c'}\right).$$

or you could have an encode-decode model spit out potentials of a UGM:

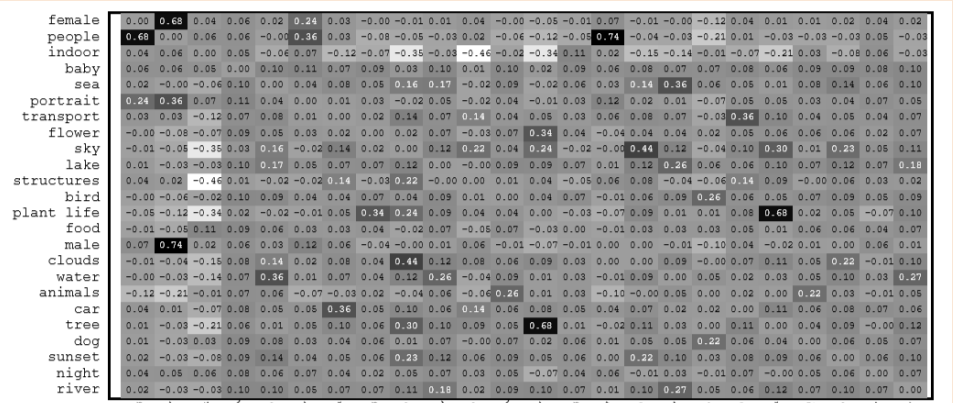


- These are sometimes called conditional neural fields or deep structured models

Multi-Label Classification

bonus!

- Learned dependencies on a mult-label image classification dataset:



Combining fully-convolutional nets with CRFs

bonus!

- DeepLab used a fully-connected pairwise UGM on top layer of FCN:

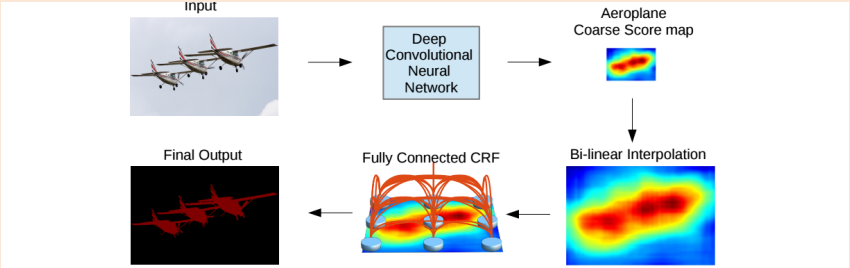


Fig. 1: Model Illustration. A Deep Convolutional Neural Network such as VGG-16 or ResNet-101 is employed in a fully convolutional fashion, using atrous convolution to reduce the degree of signal downsampling (from 32x down 8x). A bilinear interpolation stage enlarges the feature maps to the original image resolution. A fully connected CRF is then applied to refine the segmentation result and better capture the object boundaries.

https://arxiv.org/pdf/1606.00915.pdf

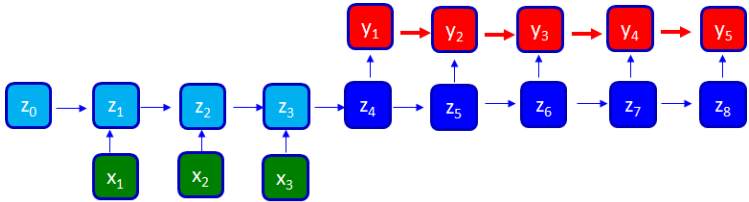
- Most recent iteration of the model removed the UGM
- Still really helps if you don't have tons of training data (Bae, . . . , Sutherland, IJCAI-23)

Do we need UGMs in Neural Networks?

- Recall that encode-decode hidden layers already capture label dependencies
  - So do we need a UGM to explicitly model label dependencies in output layer?
- Factor 1: data size (big vs. small)
  - With a small dataset, it could be helpful to have direct dependencies in model
  - With a large dataset, the hidden layers should reflect dependencies
- Factor 2: how you evaluate the model (individual parts or full decoding)
  - If you measure "pixel level" or "word level" error, UGMs may not help
  - If you measure "whole image" or "whole sentence" error, UGMs may help
    - For example, inference can discourage unlikely joint labellings

Combining RNNs and Graphical Models

- An example where we use explicit label dependencies is language translation:



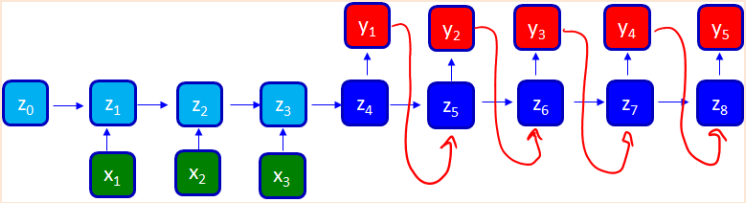
- Above model has usual deterministic edges, and DAG edges on labels
- Can use Viterbi decoding to find best translation in this model
  - Taking into account probability of seeing neighbouring words

- But there is not much information in the DAG part of the model
  - Only modeling dependencies between adjacent words
- What we really want is to have the label we output affect the hidden state
  - So that the encoding reflects previously-output words

Combining RNNs and Graphical Models

bonus!

- In order for the hidden states to depend on the output, we have this monstrosity:

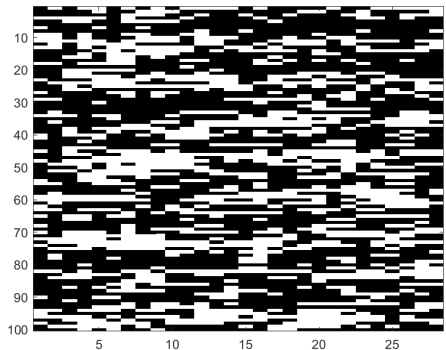


- This can still be written as a Markov chain, but we cannot do Viterbi decoding
  - Problem is that the hidden states in decoder become random variables
  - So the state at each time has discrete and continuous parts (cannot be enumerated)

- To do decoding in this thing, we typically use beam search
  - Heuristic algorithm that maintains "k best decodings up to time t"
    - Can be arbitrarily bad, but works if decoding is obvious as we go forward in time
  - The type of edge and decoding strategy is also common with transformers

Back to the Rain Data

- "Vancouver Rain" data:



- We used homogeneous Markov chains to model between-day dependence

Back to the Rain Data

- Before, we used a conditional random field to depend on the month
- We could alternately try to learn the clusters using a mixture model
  - But mixture of independents wouldn't capture dependencies within cluster
- A mixture of Markov chains could capture direct dependence *and* clusters,

$$p(x_1, x_2, \dots, x_d) = \sum_{c=1}^{k} p(z = c) \underbrace{p(x_1 \mid z = c)p(x_2 \mid x_1, z = c) \cdots p(x_d \mid x_{d-1}, z = c)}_{\text{Markov chain for cluster } c}$$

- Cluster z chooses which homogeneous Markov chain parameters to use.
  - We could learn that some months are more likely to have rain (like winter months)
  - Can do inference by running forward-backward on each mixture; fit model with EM

Comparison of Models on Rain Data

- Independent (homogeneous) Bernoulli:
  - Average NLL: 18.97 (1 parameter)
- Independent Bernoullis:
  - Average NLL: 18.95, (28 parmaeters)
- Mixture of Bernoullis ($k = 10$, five random restarts of EM):
  - Average NLL: 17.06 ($10 + 10 \times 28 = 290$ parameters)
- Homogeneous Markov chain:
  - Average NLL: 16.81 (3 parameters)
- Mixture of Markov chains ($k = 10$, five random restarts of EM):
  - Average NLL: 16.53 ($10 + 10 \times 3 = 40$ parameters)
  - Parameters of one of the clusters (possibly modeling summer months):

$$
\begin{aligned}
&p(z = 5) = 0.14 \\
&p(x_1 = \text{“rain”} \mid z = 5) = 0.22 && \text{(instead of usual 37\%)} \\
&p(x_j = \text{“rain”} \mid x_{j-1} = \text{“rain”}, z = 5) = 0.49 && \text{(instead of usual 65\%)} \\
&p(x_j = \text{“rain”} \mid x_{j-1} = \text{“not rain”}, z = 5) = 0.11 && \text{(instead of usual 35\%)}
\end{aligned}
$$

Back to the Rain Data

- The rain data is artificially divided into months

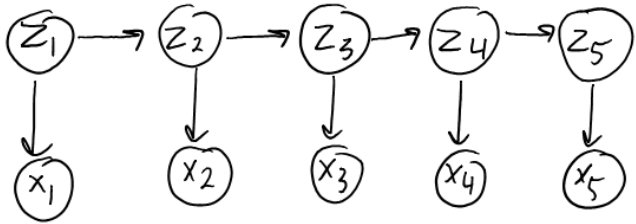
- We previously discussed viewing rain data as one very long sequence ($n = 1$)
- We could apply homogeneous Markov chains due to parameter tying
  - But a mixture doesn't make sense when $n = 1$

- What we want: different "parts" of the sequence come from different clusters
  - We transition from "summer" cluster to "fall" cluster at some time j

- One way to address this is with a "hidden" Markov model (HMM):
  - Instead of examples being assigned to clusters, days are assigned to clusters
  - Have a Markov dependency between cluster values of adjacent days

Hidden Markov Models

- Hidden Markov models have each x_j depend on a hidden Markov chain

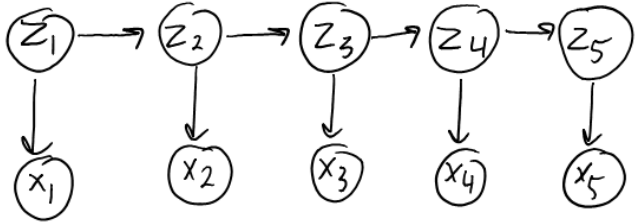


$$p(x_1, x_2, \ldots, x_d, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j)$$

- We're going to learn clusters z_j and the hidden dynamics between days
  - Hidden cluster z_j could be "summer" or "winter" (we're learning the clusters)
  - Transition probability $p(z_j \mid z_{j-1})$ is probability of staying in "summer"
    - Initial probability $p(z_1)$ is probability of starting chain in "summer"
  - Emission probability $p(x_j \mid z_j)$ is probability of `rain` during "summer"

Hidden Markov Models

- Hidden Markov models have each x_j depend on a hidden Markov chain



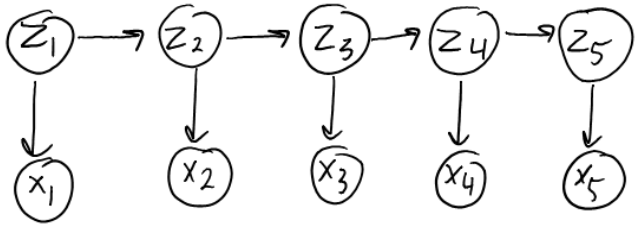
$$p(x_1, x_2, \ldots, x_d, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j)$$

- You observe the x_j values but don't see the z_j values
  - There is a "hidden" Markov chain, whose state determines the cluster at each time

- HMMs generalize both Markov chains and mixture of categoricals
  - Both models are obtained under appropriate parameters

Hidden Markov Models

- Hidden Markov models have each x_j depend on a hidden Markov chain.

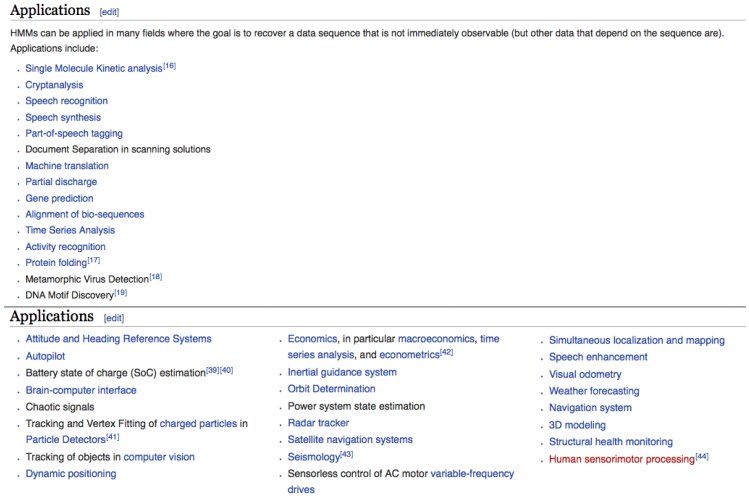


$$p(x_1, x_2, \ldots, x_d, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j)$$

- Note that the x_j can be continuous even with discrete clusters z_j
  - Data could come from a mixture of Gaussians, with cluster changing in time
- If the z_j are continuous it's often called a state-space model
  - If everything is Gaussian, it leads to Kalman filtering
  - Keywords for non-Gaussian: unscented Kalman filter and particle filter

Applications of HMMs and Kalman Filters

- HMMs variants are probably the most-used time-series model



Applications [edit]

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable (but other data that depend on the sequence are). Applications include:

- Single Molecule Kinetic analysis[16]
- Cryptanalysis
- Speech recognition
- Speech synthesis
- Part-of-speech tagging
- Document Separation in scanning solutions
- Machine translation
- Partial discharge
- Gene prediction
- Alignment of bio-sequences
- Time Series Analysis
- Activity recognition
- Protein folding[17]
- Metamorphic Virus Detection[18]
- DNA Motif Discovery[19]

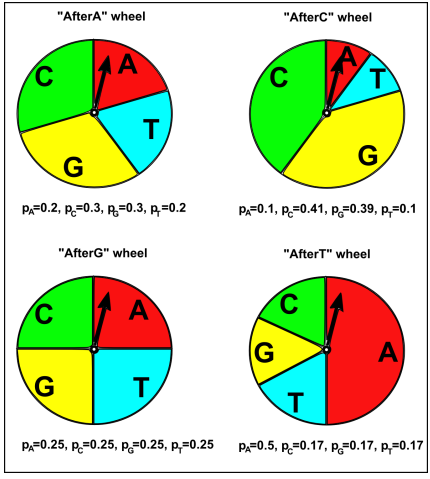
Applications [edit]

- Attitude and Heading Reference Systems
- Autopilot
- Battery state of charge (SoC) estimation[39][40]
- Brain-computer interface
- Chaotic signals
- Tracking and Vertex Fitting of charged particles in Particle Detectors[41]
- Tracking of objects in computer vision
- Dynamic positioning
- Economics, in particular macroeconomics, time series analysis, and econometrics[42]
- Inertial guidance system
- Orbit Determination
- Power system state estimation
- Radar tracker
- Satellite navigation systems
- Seismology[43]
- Sensorless control of AC motor variable-frequency drives
- Simultaneous localization and mapping
- Speech enhancement
- Visual odometry
- Weather forecasting
- Navigation system
- 3D modeling
- Structural health monitoring
- Human sensorimotor processing[44]

Also includes chain-structured conditional random fields

Example: Modeling DNA Sequences

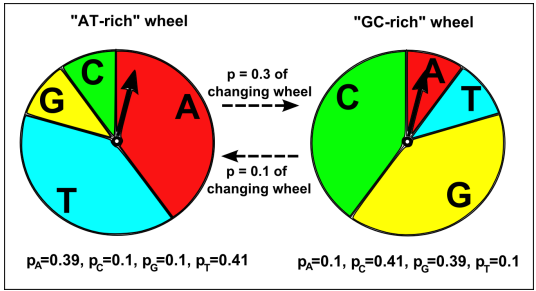
- Previously: Markov chain for DNA sequences:



https://www.tes.com/lessons/WE5E9RncBhieAQ/dna

Example: Modeling DNA Sequences

- Hidden Markov model (HMM) for DNA sequences (two hidden clusters):



- This is a (hidden) state transition diagram
  - Can reflect that probabilities are different in different regions
  - The actual regions are not given, but instead are nuisance variables handled by EM

- A better model might use a hidden and visible Markov chain
  - With 2 hidden clusters, you would have 8 "probability wheels" (4 per cluster)
  - Would have "treewidth 2", so inference would be tractable

Inference and Learning in HMMs

- Given observed features x_j, likelihood of a joint z_j assignment is

$$p(z_1, z_2, \dots z_d \mid x_1, x_2, \dots, x_d) \propto p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j)$$

- We can do inference with forward-backward by converting to potentials:

$$\begin{aligned} \phi_1(z_1) &= p(z_1)p(x_1 \mid z_1) \\ \phi_j(z_j) &= p(x_j \mid z_j) \qquad\qquad (j > 1) \\ \phi_{j,j-1}(z_j, z_{j-1}) &= p(z_j \mid z_{j-1}) \end{aligned}$$

- Marginals from forward-backward are used to update parameters in EM
  - In this setting EM is called the "Baum-Welch" algorithm
  - As with other mixture models, learning with EM is sensitive to initialization

Who is Guarding Who?

bonus!

- There is a lot of data on scoring/offense of NBA basketball players
  - Every point and assist is recorded, more scoring gives more wins and $$$
- But how do we measure defense ("stopping people from scoring")?
  - We need to know who each player is guarding (which isn't recorded)

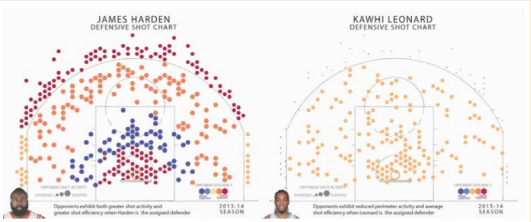


Figure 2a. Graphical depiction of a defender's volume (size) and disruption scores (color). Kawhi Leonard tends to suppress shots on the perimeter. More comparisons are provided in the Appendix.

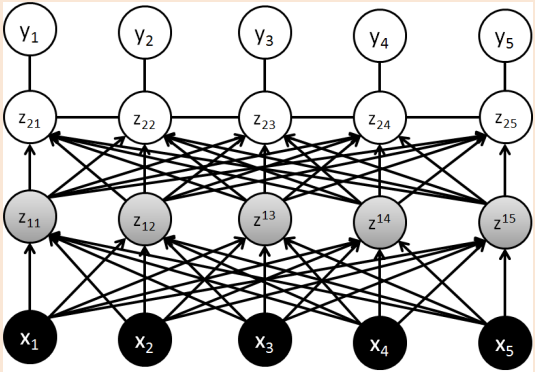
http://www.lukebornn.com/papers/franks_ssac_2015.pdf

- HMMs can be used to model who is guarding who over time
  - `https://www.youtube.com/watch?v=JvNkZdZJBt4`

Neural Networks with Latent-Dynamics

bonus!

- Could have (undirected) HMM parameters come out of a neural network:
  - Tries to model hidden dynamics across time



- Combines deep learning, mixture models, and graphical models
  - "Latent-dynamics model"
  - Previously achieved state of the art in several applications

Summary

- Undirected graphical models factorize probability into non-negative potentials
  - Also called "Markov random fields"
  - Gaussians are a special case, but can place potentials on any subset of variables
  - Checking independence is simple: is there a path in the (undirected) graph?
  - Exact inference is exponential in "treewidth" of graph
- Log-linear parameterization can be useful for learning
  - Need approximate inference as a subroutine inside the learning loop
- Conditional random fields add conditioning on other variables
  - Side information: month in the rain data
  - Consistency among outputs, like in image segmentation
- Hidden Markov models have Markov structure on latent states
  - EM to do inference

- Lots of bonus material today which were lectures in past years:
  - Graphical model inference
  - Topic models
  - Boltzmann machines

Automatic Differentiation (AD) vs. Inference

bonus!

- Deep structured model gradient combines neural/Markov gradients:
  1. Forward pass through neural network to get $\hat{y}_c$ predictions
  2. Forward message passing to compute normalizing constant
  3. Backwards message passing to compute marginals
  4. Backwards pass through neural network to get all gradients
- You could skip the last two steps if you use automatic differentiation

- But with approximate inference, AD may or may not work:
  - AD will work for iterative variational inference methods
    - But it takes way more memory than needed (needs to store all iterations)
  - AD is harder for Monte Carlo methods
    - Can't AD through sampling steps – but can use "reparamaterization trick" (later)

- Recent trend: run iterative variational method for a fixed number of iterations
  - AD can give gradient of result after this fixed number of iterations
  - "Train the inference you'll use at test time"

Example: Ising Model of Rain Data

bonus!

- E.g., for the rain data we could parameterize our node potentials using

$$\log(\phi_i(x_i)) = \begin{cases} w_1 & \text{no rain} \\ 0 & \text{rain} \end{cases}.$$

- Why do we only need 1 parameter?
  - Scaling $\phi_i(1)$ and $\phi(2)$ by constant doesn't change distribution.

- In general, we only need $(k-1)$ parameters for a k-state variable.
  - But if we're using regularization we may want to use k anyways (symmetry).

Example: Ising Model of Rain Data

bonus!

- The Ising parameterization of edge potentials,

$$\log(\phi_{ij}(x_i, x_j)) = \begin{cases} w_2 & x_i = x_j \\ 0 & x_i \neq x_j \end{cases}.$$

- Applying gradient descent gives MLE of

$$w = \begin{bmatrix} 0.16 \\ 0.85 \end{bmatrix}, \quad \phi_i = \begin{bmatrix} \exp(w_1) \\ \exp(0) \end{bmatrix} = \begin{bmatrix} 1.17 \\ 1 \end{bmatrix}, \quad \phi_{ij} = \begin{bmatrix} \exp(w_2) & \exp(0) \\ \exp(0) & \exp(w_2) \end{bmatrix} = \begin{bmatrix} 2.34 & 1 \\ 1 & 2.34 \end{bmatrix},$$

  preference towards no rain, and adjacent days being the same.
- Average NLL of 16.8 vs. 19.0 for independent model.

Full Model of Rain Data

bonus!

- We could alternately use fully expressive edge potentials

$$\log(\phi_{ij}(x_i, x_j)) = \begin{bmatrix} w_2 & w_3 \\ w_4 & w_5 \end{bmatrix},$$

  but these don't improve the likelihood much.

- We could fix one of these at 0 due to the normalization.
  - But we often don't do this when using regularization.

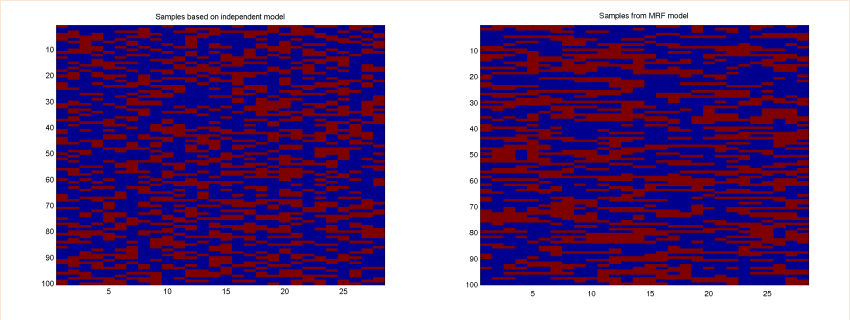
- We could also have special potentials for the boundaries.
  - Many language models are homogeneous, except for start/end of sentences.

Example: Ising Model of Rain Data

bonus!

Independent model vs. chain-UGM model with tied nodes and Ising tied edges:

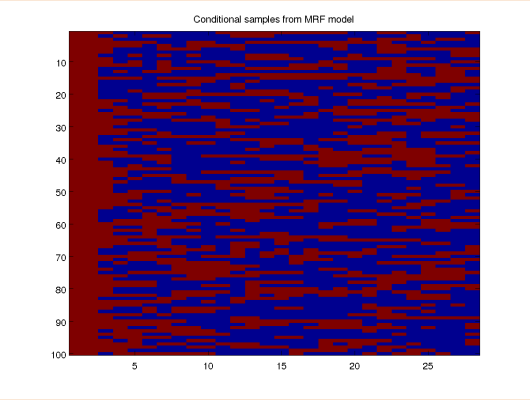
- For this dataset, using untied or general edges doesn't change likelihood much.



Example: Ising Model of Rain Data

bonus!

Samples from Ising chain-UGM model if it rains on the first day:



Example of Feature Function

bonus!

- Consider the 2-node 1-edge UGM (1)–(2), where each state has 2 values.
  - So we have potentials $\phi_1(x_1)$, $\phi_2(x_2)$, and $\phi_{12}(x_1, x_2)$ and want to have

$$w^{\mathsf{T}} F(x) = w_{1,x_1} + w_{2,x_2} + w_{1,2,x_1,x_2}.$$

- With no parameter tying and $x = \begin{bmatrix} 2 & 1 \end{bmatrix}$, our parameter vector and features are

$$w = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{2,1} \\ w_{2,2} \\ w_{1,2,1,1} \\ w_{1,2,1,2} \\ w_{1,2,2,1} \\ w_{1,2,2,2} \end{bmatrix}, \quad F(x) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

Example of Feature Function

bonus!

- If we instead had Ising potentials (just measuring whether $x_1 = x_2$) we would have

$$w^\mathsf{T} F(x) = w_{1,x_1} + w_{2,x_2} + w_{1,2,\text{same}},$$

  where $w_{1,2,\text{same}}$ is the parameter specifying how much we want $x_1 = x_2$.

- With no parameter tying and $x = \begin{bmatrix} 2 & 1 \end{bmatrix}$, our parameter vector and features are

$$w = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{2,1} \\ w_{2,2} \\ w_{1,2.\text{same}} \end{bmatrix}, \quad F(x) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

UGM Training Objective Function

bonus!

- With log-linear parameterization, NLL for IID training examples is

$$\begin{aligned} f(w) &= -\sum_{i=1}^{n} \log p(x^i \mid w) = -\sum_{i=1}^{n} \log \left(\frac{\exp(w^\mathsf{T} F(x^i))}{Z(w)} \right) \\ &= -\sum_{i=1}^{n} w^\mathsf{T} F(x^i) + \sum_{i=1}^{n} \log Z(w) \\ &= -w^\mathsf{T} F(\mathbf{X}) + n \log Z(w). \end{aligned}$$

  where the $F(\mathbf{X}) = \sum_i F(x^i)$ are called the sufficient statistics of the dataset.
  - Given sufficient statistics $F(\mathbf{X})$, we can throw out the examples x^i.

    (only go through data once)
- Function $f(w)$ is convex (it's linear plus a big log-sum-exp function).
  - But notice that Z depends on w

.

Log-Linear UGM Gradient

bonus!

- For 1 example x, we showed that NLL with log-linear parameterization is

$$f(w) = -w^\top F(\mathbf{X}) + \log Z(w).$$

- The partial derivative with respect to parameter w_j has a simple form

$$\begin{aligned}\nabla_{w_j} f(w) &= -F_j(\mathbf{X}) + \sum_x \frac{\exp(w^\top F(x))}{Z(w)} F_j(x) \\ &= -F_j(\mathbf{X}) + \sum_x p(x \mid w) F_j(x) \\ &= -F_j(\mathbf{X}) + \mathbb{E}[F_j(x)].\end{aligned}$$

- Observe that derivative of $\log(Z)$ is expected value of feature.

Segmentation with Label Dependencies

bonus!

- We got a bit more fancy and used edge features x^{ij},

$$p(y^1, y^2, \dots, y^d \mid x^1, x^2, \dots, x^d) = \frac{1}{Z} \exp\left(\sum_{i=1}^{d} y^i w^\top x^i + \sum_{(i,j) \in E} y^i y^j v^\top x^{ij} \right).$$

- For example, we could use $x^{ij} = 1/(1 + |x^i - x^j|)$.
  - Encourages y_i and y_j to be more similar if x^i and x^j are more similar.

- This is a pairwise UGM with

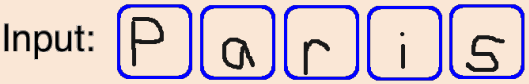
$$\phi_i(y^i) = \exp(y^i w^\top x^i), \quad \phi_{ij}(y^i, y^j) = \exp(y^i y^j v^\top x^{ij}),$$

so it didn't make inference any more complicated.

Modeling OCR Dependencies

bonus!

- What dependencies should we model for this problem?



  - $\phi(y_c, x_c)$: potential of individual letter given image.
  - $\phi(y_{c-1}, y_c)$: dependency between adjacent letters ('q-u').
  - $\phi(y_{c-1}, y_c, x_{c-1}, x_c)$: adjacent letters and image dependency.
  - $\phi_c(y_{c-1}, y_c)$: inhomogeneous dependency (French: 'e-r' ending).
  - $\phi_c(y_{c-2}, y_{c-1}, y_c)$: third-order and inhomogeneous (English: 'i-n-g' end).
  - $\phi(y \in \mathcal{D})$: is y in dictionary $\mathcal{D}$?

Tractability of Discriminative Models

bonus!

- Features can be very complicated, since we just condition on the x_c, .
- Given the x_c, tractability depends on the conditional UGM on the y_c.
  - Inference tasks will be fast or slow, depending on the y_c graph.
- Besides "low treewidth", some other cases where exact computation is possible:
  - Semi-Markov chains (allow dependence on time you spend in a state).
    - For example, in rain data the seasons will be approximately 3 months.
  - Context-free grammars (allows potentials on recursively-nested parts of sequence).
  - Sum-product networks (restrict potentials to allow exact computation).
  - "Dictionary" feature is non-Markov, but exact computation still easy.
- We can alternately use our previous approximations:
  1. Pseudo-likelihood (what we used).
  2. Monte Carlo approximate inference (eventually better but probably much slower).
  3. Variational approximate inference (fast, quality varies).

Structure Learning in UGMs

bonus!

- Recall that in Ising UGMs, our edge potentials have the form

$$\phi_{ij}(x_i, x_j) = \exp(w_{ij} x_i x_j).$$

- If we set $w_{ij} = 0$, it sets $\phi_{ij}(x_i, x_j) = 1$ for all x_i and x_j.
  - Potential just "multiplies by 1", which is equivalent to removing the edge.

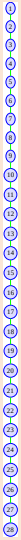
- L1-regularization of w_{ij} values performs structure learning in UGM.

- For general log-linear, each edge has multiple parameters $w_{i,j,s,s'}$.
  - In this case we can use "group L1-regularization" for structure learning.
    - Each group will be all parameters $w_{i,j,\cdot,\cdot}$ associated with an edge (i, j).

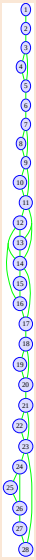
Structure Learning on Rain Data

bonus!

Large λ (and optimal tree):



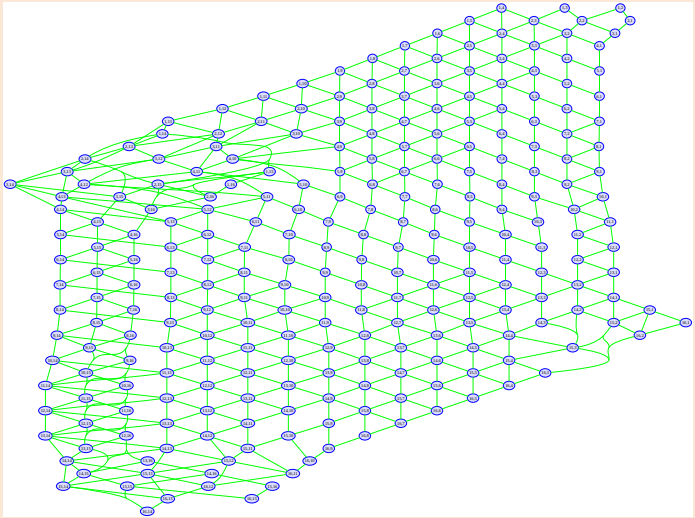
Small λ:



Structure Learning on USPS Digits

bonus!

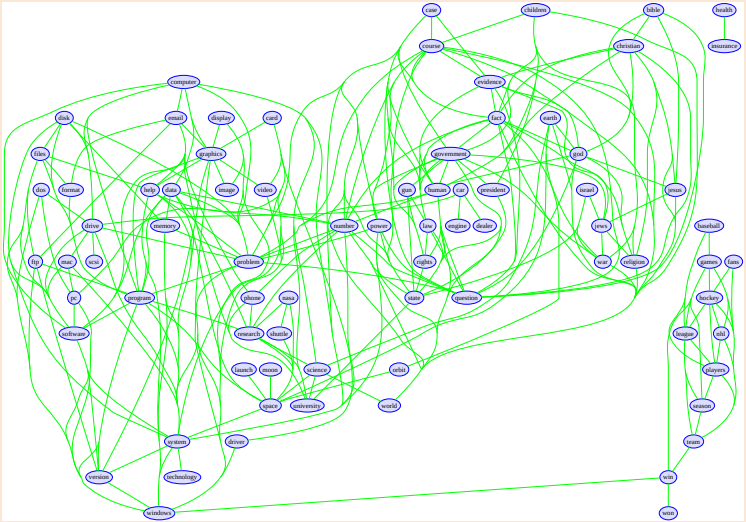
Structure learning of pairwise UGM with group-L1 on USPS digits:



Structure Learning on News Words

bonus!

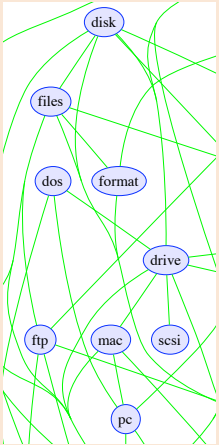
Group-L1 on newsgroups data:



Structure Learning on News Words

bonus!

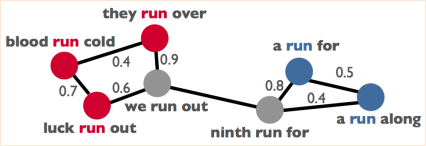
Group-L1 on newsgroups data:



Posterior Regularization

bonus!

- In some cases it might make sense to use posterior regularization:
  - Regularize the probabilities in the resulting model.
- Consider an NLP labeling task where
  - You have a small amount of labeled sentences.
  - You have a huge amount of unlabeled sentences.
- Maximize labeled likelihood, plus total-variation penalty on $p(y_c \mid x, w)$ values.
  - Give high regularization weights to words appearing in same trigrams:



http://jgillenw.com/conll2013-talk.pdf

- Useful for "out of vocabulary" words (words that don't appear in labeled data).
  - Has been replaced in recent by continuous word representations like word2vec.

Does Semi-Supervised Learning Make Sense?

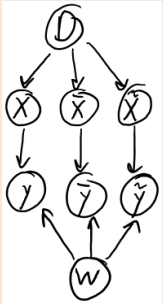
bonus!

- Should unlabeled examples always help supervised learning?
  - No!
- Consider choosing unlabeled features $\bar{x}^i$ uniformly at random.
  - Unlabeled examples collected in this way will not help.
  - By construction, distribution of $\bar{x}^i$ says nothing about $\bar{y}^i$.
- Example where SSL is not possible:
  - Try to detect food allergy by trying random combinations of food:
    - The actual random process isn't important, as long as it isn't affected by labels.
    - You can sample an infinite number of $\bar{x}^i$ values, but they says nothing about labels.
- Example where SSL is possible:
  - Trying to classify images as "cat" vs. "dog.":
    - Unlabeled data would need to be images of cats or dogs (not random images).
    - Unlabeled data contains information about what images of cats and dogs look like.
    - For example, there could be clusters or manifolds in the unlabeled images.

Does Semi-Supervised Learning Make Sense?

bonus!

- Let's assume our semi-supervised learning model is represented by this DAG:

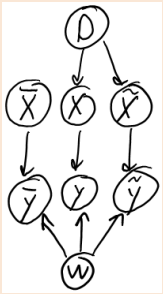


- Assume we observe $\{X, y, \tilde{X}\}$ and are interested in test labels $\tilde{y}$:
  - There is a dependency between y and $\tilde{y}$ because of path through w.
    - Parameter w is tied between training and test distributions.
  - There is a dependency between X and $\tilde{y}$ because of path through w (given y).
    - But note that there is also a second path through D and $\tilde{X}$.
  - There is a dependency between $\tilde{X}$ and $\tilde{y}$ because of path through D and $\tilde{X}$.
    - Unlabeled data helps because it tells us about data-generating distribution D.

Does Semi-Supervised Learning Make Sense?

bonus!

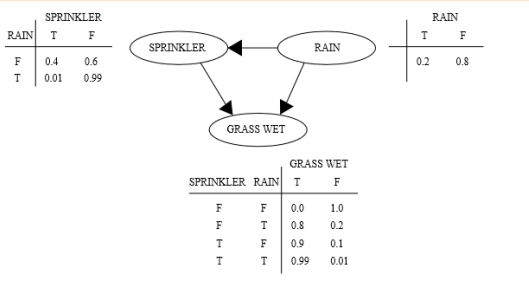
- Now consider generating $\bar{X}$ independent of D:



- Assume we observe $\{X, y, \bar{X}\}$ and are interested in test labels $\tilde{y}$:
  - Knowing X and y are useful for the same reasons as before.
  - But knowing $\bar{X}$ is not useful:
    - Without knowing $\bar{y}$, $\bar{X}$ is d-separated from $\tilde{y}$ (no dependence).

Tabular Parameterization Example

bonus!



https://en.wikipedia.org/wiki/Bayesian_network

Some quantities can be directly read from the tables:

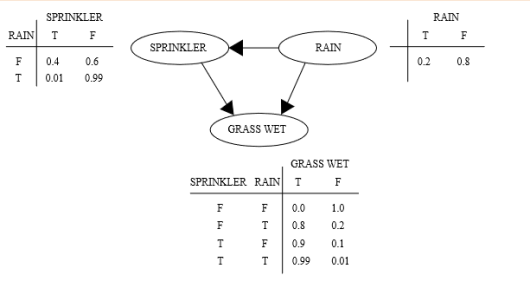
$$p(R = 1) = 0.2.$$

$$p(G = 1 \mid S = 0, R = 1) = 0.8.$$

Can calculate any probabilities using marginalization/product-rule/Bayes-rule (bonus).

Tabular Parameterization Example

bonus!



https://en.wikipedia.org/wiki/Bayesian_network

Can calculate any probabilities using marginalization/product-rule/Bayes-rule, for example:

$$p(G=1 \mid R=1) = p(G=1, S=0 \mid R=1) + p(G=1, S=1 \mid R=1) \quad \left(p(a \mid c) = \sum_b p(a, b \mid c) \right)$$

$$= p(G=1 \mid S=0, R=1)p(S=0 \mid R=1) + p(G=1 \mid S=1, R=1)p(S=1 \mid R=1)$$

$$= 0.8(0.99) + 0.99(0.01) = 0.81.$$

Dynamic Bayesian Networks

bonus!

- Dynamic Bayesian networks combine ideas from DAGs and Markov chains:
  - At each time, we have a set of variables x^t.
  - The initial x^0 comes from an "initial" DAG.
  - Given x^{t-1}, we generate x^t from a "transition" DAG.

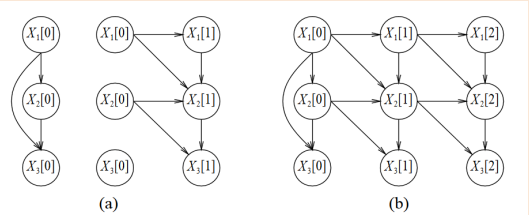


Figure 1: (a) A prior network and transition network defining a DPN for the attributes X_1, X_2, X_3. (b) The corresponding "unrolled" network.

https://www.cs.ubc.ca/~murphyk/Papers/dbnsem_uai98.pdf

- Can be used to model multiple variables over time.
  - Unconditional sampling is easy but inference may be hard.

Outline

5. Bonus material on inference
   - More UGMs
   - Treewidth
   - ICM
   - Block Inference
6. Bonus: Topic Models
7. Topic Models
8. Bonus: Restricted Boltzmann Machines

General Pairwise UGM

bonus!

- For general discrete x_i a generalization of Ising models is

$$p(x_1, x_2, \ldots, x_d) = \frac{1}{Z} \exp\left(\sum_{i=1}^{d} w_{i,x_i} + \sum_{(i,j)\in E} w_{i,j,x_i,x_j}\right),$$

  which can represent any "positive" pairwise UGM (meaning $p(x) > 0$ for all x).

- Interpretation of weights for this UGM:
  - If $w_{i,1} > w_{i,2}$ then we prefer $x_i = 1$ to $x_i = 2$.
  - If $w_{i,j,1,1} > w_{i,j,2,2}$ then we prefer $(x_i = 1, x_j = 1)$ to $(x_i = 2, x_j = 2)$.

- As before, we can use parameter tying:
  - We could use the same w_{i,x_i} for all positions i.
  - Ising model corresponds to a particular parameter tying of the w_{i,j,x_i,x_j}.

Label Propagation (Graph-Based Semi-Supervised) as a UGM

bonus!

- Consider modeling the probability of a vector of labels $\bar{y} \in \mathbb{R}^t$ using

$$p(\bar{y}^1, \bar{y}^2, \dots, \bar{y}^t) \propto \exp\left(-\sum_{i=1}^{n}\sum_{j=1}^{t} w_{ij}(y^i - \bar{y}^i)^2 - \frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{t} \bar{w}_{ij}(\bar{y}^i - \bar{y}^j)^2\right).$$

- Decoding in this model is the label propagation problem.

- This is a pairwise UGM:

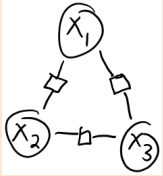
$$\phi_j(\bar{y}^j) = \exp\left(-\sum_{i=1}^{n} w_{ij}(y^i - \bar{y}^j)^2\right), \quad \phi_{ij}(\bar{y}^i, \bar{y}^j) = \exp\left(-\frac{1}{2}\bar{w}_{ij}(\bar{y}^i - \bar{y}^j)^2\right).$$

Factor Graphs

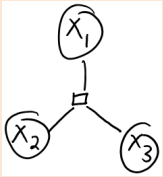
bonus!

- Factor graphs are a way to visualize UGMs that distinguishes different orders.
  - Use circles for variables, squares to represent dependencies.

- Factor graph of $p(x_1, x_2, x_3) \propto \phi_{12}(x_1, x_2)\phi_{13}(x_1, x_3)\phi_{23}(x_2, x_3)$:



- Factor graph of $p(x_1, x_2, x_3) \propto \phi_{123}(x_1, x_2, x_3)$:



Other Graphical Models

bonus!

- Factor graphs: we use a square between variables that appear in same factor.
  - Can distinguish between a 3-way factor and 3 pairwise factors.
- Chain-graphs: DAGs where each block can be a UGM.
- Ancestral-graph:
  - Generalization of DAGs that is closed under conditioning.
- Structural equation models (SEMs): generalization of DAGs that allows cycles.

Outline

5. Bonus material on inference
   - More UGMs
   - Treewidth
   - ICM
   - Block Inference
6. Bonus: Topic Models
7. Topic Models
8. Bonus: Restricted Boltzmann Machines

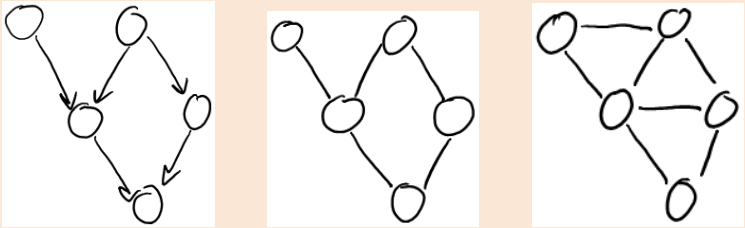
Moralization: Converting DAGs to UGMs

bonus!

- To address the NP-hard problems, DAGs and UGMs use same techniques.
- We'll focus on UGMs, but we can convert DAGs to UGMs:

$$p(x_1, x_2, \dots, x_d) = \prod_{j=1}^{d} p(x_j | x_{\text{pa}(j)}) = \prod_{j=1}^{d} \underbrace{\phi_j(x_j, x_{\text{pa}(j)})}_{=p(x_j|x_{\text{pa}(j)})},$$

  which is a UGM with $Z = 1$.
- Graphically: we drop directions and "marry" parents (moralization).



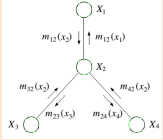
- May no longer see some independences, but doesn't change computational cost.

Easy Cases: Chains, Trees and Forests

bonus!

- The forward-backward algorithm still works for chain-structured UGMs:
  - We compute the forward messages M and the backwards messages V.
  - With both M and V we can [conditionally] decode/marginalize/sample.

- Belief propagation generalizes this to trees (undirected graphs with no cycles):
  - Pick an arbitrary node as the "root", and order the nodes going away from the root.
    - Pass messages starting from the "leaves" going towards the root.
  - "Root" is like the last node in a Markov chain.
    - Backtrack from root to leaves to do decoding/sampling.
    - Send messages from the root going to the leaves to compute all marginals.



https://www.quora.com/

Probabilistic-graphical-models-what-are-the-relationships-between-sum-product-algorithm-belief-propagation-and-junction-tree-

Easy Cases: Chains, Trees and Forests

bonus!

- Recall the CK equations in Markov chains:

$$M_c(x_c) = \sum_{x_p} p(x_c \mid x_p) M_p(x_p).$$

- For chain-structure UGMs we would have:

$$M_c(x_c) \propto \sum_{x_p} \phi(x_p)\phi(x_p, x_c) M_p(x_p).$$

- In tree-structured UGMs, parent p in the ordering may have multiple parents.
- Message coming from "neighbour" i that itself has neighbours j and k would be

$$M_{ic}(x_c) \propto \sum_{x_i} \phi_i(x_i)\phi_{ic}(x_i, x_c) M_{ji}(x_i) M_{ki}(x_i),$$

- Univariate marginals are proportional to $\phi_i(x_i)$ times all "incoming" messages.
  - The "forward" and "backward" Markov chain messages are a special case.
  - Replace $\sum_{x_i}$ with $\max_{x_i}$ for decoding.
    - "Sum-product" and "max-product" algorithms.

Exact Inference in UGMs

bonus!

- For general graphs, the cost of message passing depends on
  1. Graph structure.
  2. Variable order.
- To see the effect of the order, consider Markov chain inference with bad ordering:

$$
\begin{aligned}
p(x_5) &= \sum_{x_5}\sum_{x_4}\sum_{x_3}\sum_{x_2}\sum_{x_1} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_4) \\
&= \sum_{x_5}\sum_{x_1}\sum_{x_4}\sum_{x_3}\sum_{x_2} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_4) \\
&= \sum_{x_5}\sum_{x_1} p(x_1) \sum_{x_3}\sum_{x_4} p(x_4 \mid x_3)p(x_5 \mid x_4) \underbrace{\sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2)}_{M_{13}(x_1,x_3)}
\end{aligned}
$$

- So even though we have a chain, we have an M with k^2 values instead of k.
  - Increases cost to $O(dk^3)$ instead of $O(dk^2)$.
  - Inference can be exponentially more expensive with the wrong ordering.

Exact Inference in UGMs

bonus!

- For general graphs, the cost of message passing depends on
  1. Graph structure.
  2. Variable order.
- As a non-tree example, consider computing Z in a simple 4-node cycle:

$$
\begin{aligned}
Z &= \sum_{x_4}\sum_{x_3}\sum_{x_2}\sum_{x_1} \phi_{12}(x_1,x_2)\phi_{23}(x_2,x_3)\phi_{34}(x_3,x_4)\phi_{14}(x_1,x_4) \\
&= \sum_{x_4}\sum_{x_3} \phi_{34}(x_3,x_4) \sum_{x_2} \phi_{23}(x_2,x_3) \sum_{x_1} \phi_{12}(x_1,x_2)\phi_{14}(x_1,x_4) \\
&= \sum_{x_4}\sum_{x_3} \phi_{34}(x_3,x_4) \sum_{x_2} \phi_{23}(x_2,x_3) M_{24}(x_2,x_4) \\
&= \sum_{x_4}\sum_{x_3} \phi_{34}(x_3,x_4) M_{34}(x_3,x_4) = \sum_{x_4} M_4(x_4).
\end{aligned}
$$

- We again have an M with k^2 values instead of k.
  - We can do inference tasks with this graph, but it costs $O(dk^3)$ instead of $O(dk^2)$.

Variable Order and Treewidth

bonus!

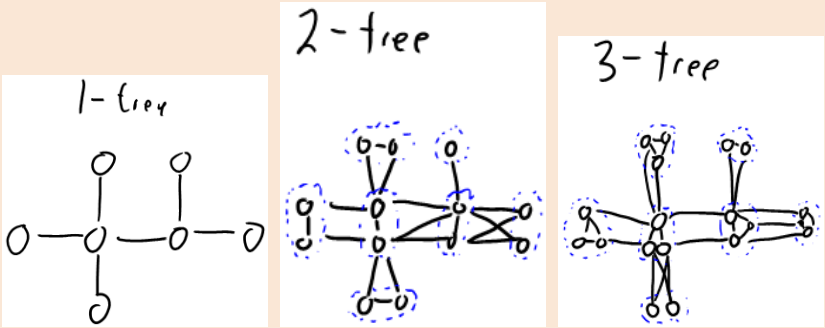
- Cost of message passing in general graphs is given by $O(dk^{\omega+1})$.
  - Here, ω is the number of dimensions of the largest message.
  - For trees, $\omega = 1$ so we get our usual cost of $O(dk^2)$.

- The minimum value of ω across orderings for a given graph is called treewidth.
  - In terms of graph: "minimum size of largest clique, minus 1, over all triangulations".
    - Also called "graph dimension" or "ω-tree".
  - Intuitively, you can think of low treewidth as being "close to a tree".
    - Trees have a treewidth of 1, and a single loop has a treewidth of 2.

Treewidth Examples

bonus!

- Examples of k-trees:

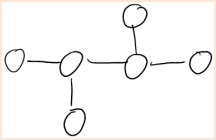


- 2-tree and 3-tree are trees if you use dotted circles to group nodes.

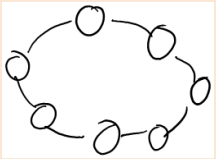
Treewidth Examples

bonus!

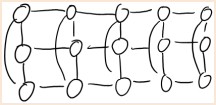
- Trees have $\omega = 1$, so with the right order inference costs $O(dk^2)$.



- A big loop has $\omega = 2$, so cost with the right ordering is $O(dk^3)$.



- The below grid-like structure has $\omega = 3$, so cost is $O(dk^4)$.



Variable Order and Treewidth

bonus!

- Junction trees generalize belief propagation to general graphs (requires ordering).
  - This is the algorithm that achieves the $O(dk^{\omega+1})$ runtime.
- Computing ω and the optimal ordering is NP-hard.
  - But various heuristic ordering methods exist.
- An m_1 by m_2 lattice has $\omega = \min\{m_1, m_2\}$.
  - So you can do exact inference on "wide chains" with Junction tree.
  - But for 28 by 28 MNIST digits it would cost $O(784 \cdot 2^{29})$.
- Some links if you want to read about treewidth:
  - https://www.win.tue.nl/~nikhil/courses/2015/2WO08/treewidth-erickson.pdf
  - https://math.mit.edu/~apost/courses/18.204-2016/18.204_Gerrod_Voigt_final_paper.pdf
- For some graphs $\omega = (d-1)$ so there is no gain over brute-force enumeration.
  - Many graphs have high treewidth so we need approximate inference.

Outline

5. Bonus material on inference
   - More UGMs
   - Treewidth
   - ICM
   - Block Inference
6. Bonus: Topic Models
7. Topic Models
8. Bonus: Restricted Boltzmann Machines

Iterated Conditional Mode (ICM)

bonus!

- The iterated conditional mode (ICM) algorithm for approximate decoding:
  - On each iteration k, choose a variable j_t.
  - Maximie the joint probability in terms of x_{j_t} (with other variables fixed),

$$x_j^{t+1} \in \arg\max_c p(x_1^t, \ldots, x_{j-1}^t, x_j = c, x_{j+1}^t, \ldots, x_d^t).$$

- Equivalently, iterations correspond to finding mode of conditional $p(x_j \mid x_{-j}^t)$,

$$x_j^{t+1} \in \arg\max_c p(x_j = c \mid x_{-j}^t),$$

where x_{-j} means "x_i for all i except x_j": $x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_d$.

ICM in Action

bonus!

- Start with some initial value: $x^0 = \begin{bmatrix} 2 & 2 & 3 & 1 \end{bmatrix}$.
- Select random j like $j = 3$.
- Set j to maximize $p(x_3 \mid x^0_{-3})$: $x^1 = \begin{bmatrix} 2 & 2 & 1 & 1 \end{bmatrix}$.
- Select random j like $j = 1$.
- Set j to maximize $p(x_1 \mid x^1_{-1})$: $x^2 = \begin{bmatrix} 3 & 2 & 1 & 1 \end{bmatrix}$.
- Select random j like $j = 2$.
- Set j to maximize $p(x_2 \mid x^2_{-2})$: $x^3 = \begin{bmatrix} 3 & 2 & 1 & 1 \end{bmatrix}$.
- ...
- Repeat until you can no longer improve by single-variable changes.
  - Intead of random, could cycle through the variables in order.
  - Or you could greedily choose the variable that increases the probability the most.

Optimality and Globalization of ICM

bonus!

- Does ICM find the global optimum?
- Decoding is usually non-convex, so doesn't find global optimum.
  - ICM is an approximate decoding method.
- There exist many globalization methods that can improve its performance:
  - Restarting with random initializations.
  - Global optimization methods:
    - Simulated annealing, genetic algorithms, ant colony optimization, GRASP, etc.

Using the Unnormalized Objective

bonus!

- How can you maximize $p(x)$ in terms of x_j if evaluating it is NP-hard?
- Let's define the unnormalized probability $\tilde{p}$ as

$$\tilde{p}(x) = \prod_{c \in \mathcal{C}} \phi_c(x_c).$$

- So the normalized probability is given by

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

- In UGMs evaluating Z is hard but evaluating $\tilde{p}(x)$ is easy.
- And for decoding we only need unnormalized probabilities,

$$\arg\max_x p(x) \equiv \arg\max_x \frac{\tilde{p}(x)}{Z} \equiv \arg\max_x \tilde{p}(x),$$

so we can decode based on $\tilde{p}$ without knowing Z.

ICM Iteration Cost

bonus!

- How much does ICM cost?

- Consider a pairwise UGM,

$$\tilde{p}(x) = \left(\prod_{j=1}^{d} \phi_j(x_j)\right)\left(\prod_{(i,j)\in E} \phi_{ij}(x_i, x_j)\right).$$

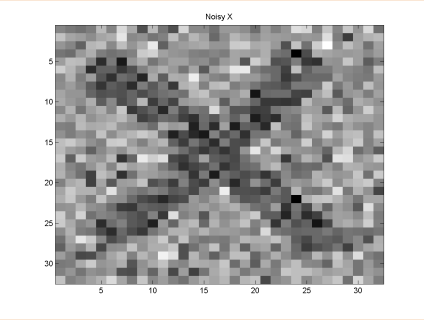
- Each ICM update would:
  1. Set $M_j(x_j = s)$ to product of terms in $\tilde{p}(x)$ involving x_j, with x_j set to s.
  2. Set x_j to the largest value of $M_j(x_j)$.

- The variable x_j has k values and appears in at most d factors here.
  - You can compute the k values of these d factors in $O(dk)$ to find the largest.
  - If you only have m nodes in "Markov blanket", this reduces to $O(mk)$.
    - We will define "Markov blanket" in a couple slides.

ICM in Action

bonus!

Consider using a UGM for binary image denoising:



We have

- Unary potentials ϕ_j for each position.
- Pairwise potentials ϕ_{ij} for neighbours on grid.
- Parameters are trained as CRF (later).

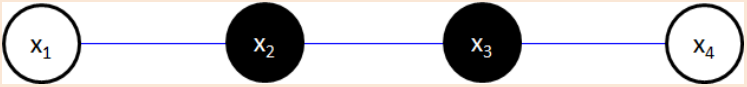
Goal is to produce a noise-free binary image (show video).

Digression: Closure of UGMs under Conditioning

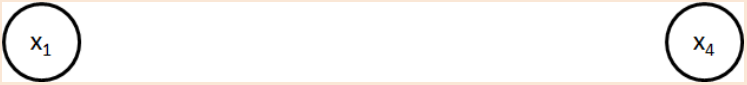
bonus!

- UGMs are closed under conditioning:
  - If $p(x)$ is a UGM, then $p(x_A \mid x_B)$ can be written as a UGM (for partition A and B).

- Conditioning on x_2 and x_3 in a chain,



  gives a UGM defined on x_1 and x_4 that is disconnected:



- Graphically, we "erase the black nodes and their edges".

- Notice that inference in the conditional UGM may be mucher easier.

Digression: Closure of UGMs under Conditioning

bonus!

- Mathematically, a 4-node pairwise UGM with a chain structure assumes

$$p(x_1, x_2, x_3, x_4) \propto \phi_1(x_1)\phi_2(x_2)\phi_3(x_3)\phi_4(x_4)\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4).$$

- Conditioning on x_2 and x_3 gives UGM over x_1 and x_4.

$$p(x_1, x_4 \mid x_2, x_3) = \frac{1}{Z'}\phi'_1(x_1)\phi'_4(x_4),$$

where new potentials "absorb" the shared potentials with observed nodes:

$$\phi'_1(x_1) = \phi_1(x_1)\phi_{12}(x_1, x_2), \quad \phi'_4(x_4) = \phi_4(x_4)\phi_{34}(x_3, x_4).$$

Conditioning in UGMs

bonus!

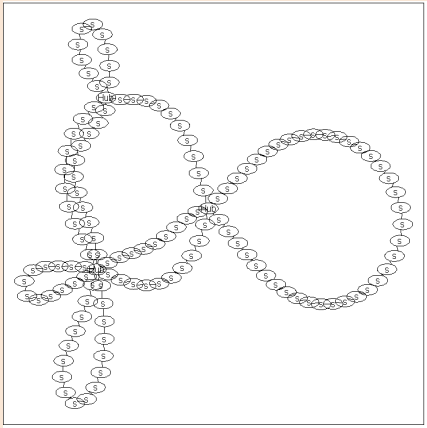
- Conditioning on x_2 and x_3 in 4-node chain-UGM gives

$$
\begin{aligned}
p(x_1, x_4 | x_2, x_3) &= \frac{p(x_1, x_2, x_3, x_4)}{p(x_2, x_3)} \\
&= \frac{\frac{1}{Z}\phi_1(x_1)\phi_2(x_2)\phi_3(x_3)\phi_4(x_4)\phi_1(x_1, x_2)\phi_2(x_2, x_3)\phi_3(x_3, x_4)}{\sum_{x_1', x_4'} \frac{1}{Z}\phi_1(x_1')\phi_2(x_2)\phi_3(x_3)\phi_4(x_4')\phi_1(x_1', x_2)\phi_2(x_2, x_3)\phi_3(x_3, x_4')} \\
&= \frac{\frac{1}{Z}\phi_1(x_1)\phi_2(x_2)\phi_3(x_3)\phi_4(x_4)\phi_1(x_1, x_2)\phi_2(x_2, x_3)\phi_3(x_3, x_4)}{\frac{1}{Z}\phi_2(x_2)\phi_3(x_3)\phi_2(x_2, x_3)\sum_{x_1', x_4'} \phi_1(x_1')\phi_4(x_4')\phi_1(x_1', x_2)\phi_3(x_3, x_4')} \\
&= \frac{\phi_1(x_1)\phi_4(x_4)\phi_1(x_1, x_2)\phi_3(x_3, x_4)}{\sum_{x_1', x_4'} \phi_1(x_1')\phi_4(x_4')\phi_1(x_1', x_2)\phi_3(x_3, x_4')} \\
&= \frac{\phi_1'(x_1)\phi_4'(x_4)}{\sum_{x_1', x_4'} \phi_1'(x_1')\phi_4'(x_4')}
\end{aligned}
$$

Simpler Inference in Conditional UGMs

bonus!

- Consider the following graph which could describe bus stops:



- If we condition on the "hubs", the graph forms a forest (and inference is easy).
  - Simpler inference after conditioning is used by many approximate inference methods.

Digression: Local Markov Property and Markov Blanket

bonus!

- Approximate inference methods often use conditional $p(x_j \mid x_{-j})$,
  - where x_{-j}^k means "x_i^k for all i except x_j^k": $x_1^k, x_2^k, \ldots, x_{j-1}^k, x_{j+1}^k, \ldots, x_d^k$.

- In UGMs, the conditional simplifies due to conditional independence,

$$p(x_j \mid x_{-j}) = p(x_j \mid x_{\text{nei}(j)}),$$

  this local Markov property means conditional only depends on neighbours.

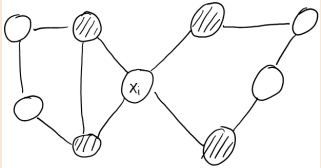
- We say that the neighbours of x_j are its "Markov blanket".

- Markov blanket is the set nodes that make you independent of all other nodes.

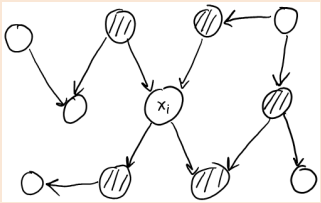
Digression: Local Markov Property and Markov Blanket

bonus!

- In UGMs the Markov blanket is the neighbours.



- Markov blanket in DAGs: parents, children, co-parents (parents of same children):



Outline

5. Bonus material on inference
   - More UGMs
   - Treewidth
   - ICM
   - Block Inference
6. Bonus: Topic Models
7. Topic Models
8. Bonus: Restricted Boltzmann Machines

Block-Structured Approximate Inference

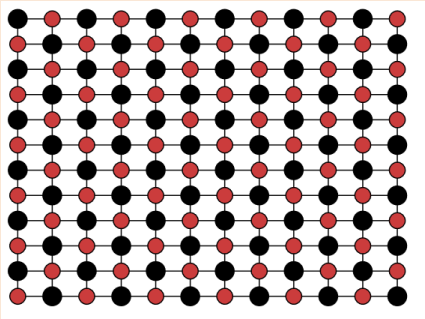
bonus!

- Basic approximate inference methods like ICM and Gibb sampling:
  - Update one x_j at a time.
  - Efficient because conditional UGM is 1 node.
- Better approximate inference methods use block updates:
  - Update a block of x_j values at once.
  - Efficient if conditional UGM allows exact inference.
- If we choose the blocks cleverly, this works substantially better.

Block-Structured Approximate Inference

bonus!

- Consider a lattice-structure and the following two blocks ("red-black ordering"):



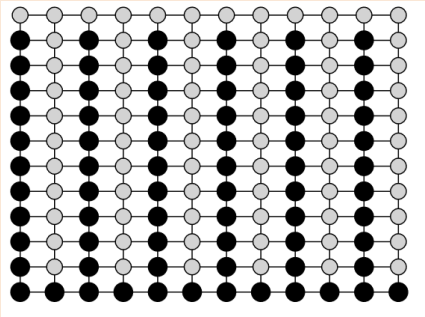
- Given black nodes, conditional UGM on red nodes is a disconnected graph.
  - "I can optimally update the red nodes given the black nodes" (and vice versa).
    - You update $d/2$ nodes at once for cost of this is $O(dk)$, and easy to parallelize.

- Minimum number of blocks to disconnect the graph is graph colouring.

Block-Structured Approximate Inference

bonus!

- We could also consider general forest-structured blocks:

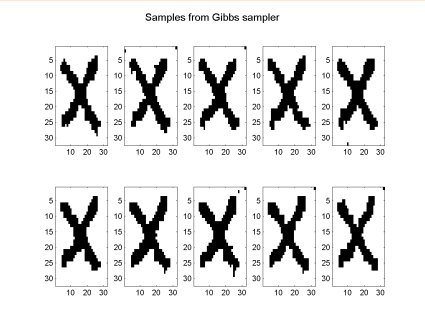


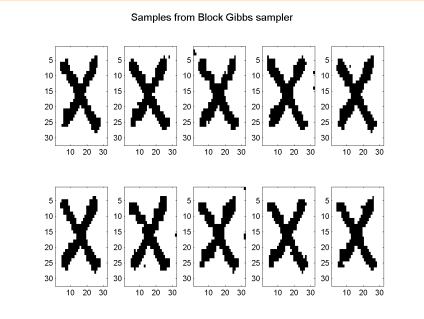
- We can still optimally update the black nodes given the gray nodes in $O(dk^2)$.
  - This works much better than "one at a time".

Block Gibbs Sampling in Action

bonus!

- Gibbs vs. tree-structured block-Gibbs samples:



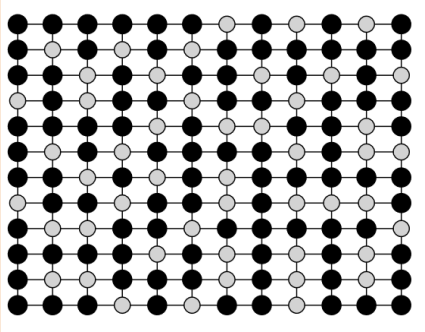


- With block sampling, the samples are far less correlated.
- We can also do tree-structured block ICM.
  - Harder to get stuck if you get to update entire trees.

Block-Structured Approximate Inference

bonus!

- Or we could define a new tree-structured block on each iteration:



- The above block updates around two thirds of the nodes optimally.

(Here we're updating the black nodes.)

Block ICM Based on Graph Cuts

bonus!

- Consider a binary pairwise UGM with "attractive" potentials,

$$\log \phi_{ij}(1,1) + \log \phi_{ij}(2,2) \geq \log \phi_{ij}(1,2) + \log \phi_{ij}(2,1).$$

- In words: "neighbours prefer to have similar states".

- In this setting exact decoding can be formulated as a max-flow/min-cut problem.
  - Can be solved in polynomial time.

- This is widely-used computer vision:
  - Want neighbouring pixels/super-pixels/regions to be more likely to get same label.

Graph Cut Example: "GrabCut"

bonus!

Figure 1: **Three examples of GrabCut .** The user drags a rectangle loosely around an object. The object is then extracted automatically.

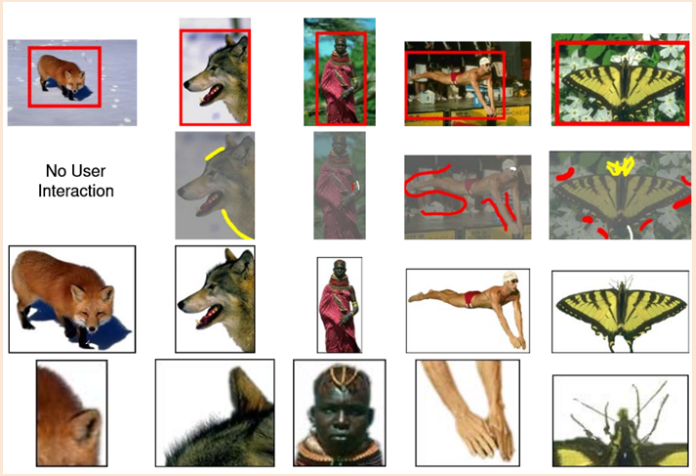
http://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf

1. User draws a box around the object they want to segment.
2. Fit Gaussian mixture model to pixels inside the box, and to pixels outside the box.
3. Construct a pairwise UGM using:
   - $\phi_i(x_i)$ set to GMM probability of pixel i being in class x_i.
   - $\phi_{ij}(x_i, x_j)$ set to Ising potential times RBF based on spatial/colour distance.
     - Use $w_{ij} > 0$ so the model is "attractive".
4. Perform exact decoding in the binary attractive model using graph cuts.

Graph Cut Example: "GrabCut"

bonus!

- GrabCut with extra user interaction:



http://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf

Alpha-Beta Swap and Alpha-Expansions: ICM with Graph Cuts

bonus!

- If we have more than 2 states, we can't use graph cuts.
- Alpha-beta swaps are an approximate decoding method for "pairwise attractive",

$$\log \phi_{ij}(\alpha, \alpha) + \log \phi_{ij}(\beta, \beta) \geq \log \phi_{ij}(\alpha, \beta) + \log \phi_{ij}(\beta, \alpha).$$

  - Each step choose an α and β, optimally "swaps" labels among these nodes.

- Alpha-expansions are another variation based on a slightly stronger assumption,

$$\log \phi_{ij}(\alpha, \alpha) + \log \phi_{ij}(\beta_1, \beta_2) \geq \log \phi_{ij}(\alpha, \beta_1) + \log \phi_{ij}(\beta_2, \alpha).$$

  - Steps choose label α, and consider replacing the label of any node not labeled α.

Alpha-Beta Swap and Alpha-Expansions: ICM with Graph Cuts

bonus!

- These don't find global optima in general, but make huge moves:

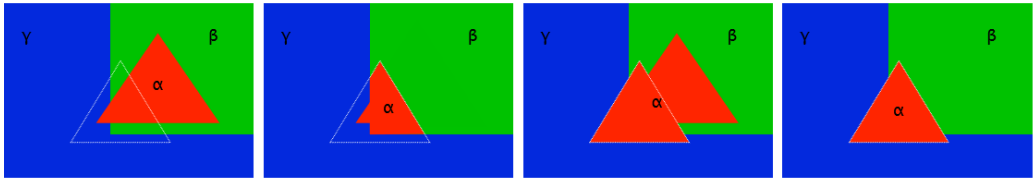


Figure 1: From left to right: Initial labeling, labeling after $\alpha\beta$-swap, labeling after α-expansion, labeling after α-expansion β-shrink. The optimal labeling of the α pixels is outlined by a white triangle, and is achieved from the initial labeling by one ~~α-expansion β-shrink move~~. ex-Swap move

- A somewhat-related MCMC method is the Swendson-Wang algorithm.

Example: Photomontage

bonus!

- Photomontage: combining different photos into one photo:

http://vision.middlebury.edu/MRF/pdf/MRF-PAMI.pdf

- Here, x_i corresponds to identity of original image at position i.

Example: Photomontage

bonus!

- Photomontage: combining different photos into one photo:

http://vision.middlebury.edu/MRF/pdf/MRF-PAMI.pdf

Outline

5 Bonus material on inference

6 Bonus: Topic Models

7 Topic Models

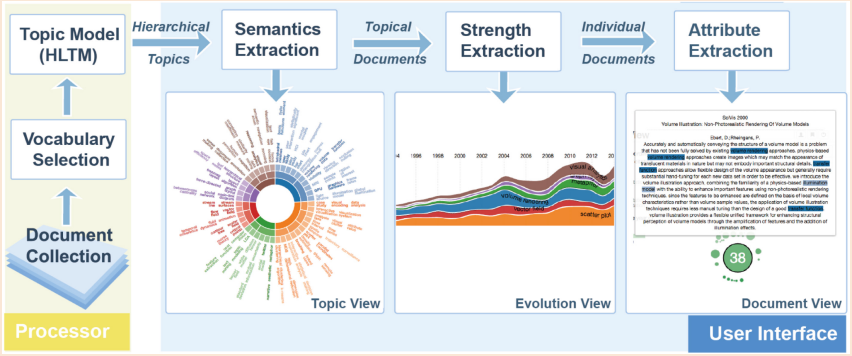
8 Bonus: Restricted Boltzmann Machines

Motivation for Topic Models

bonus!

We want a model of the hidden "factors" making up a set of documents.

- In this context, latent-factor models are called topic models.



https://www.sciencedirect.com/science/article/pii/S2468502X17300074

- "Topics" could be useful for things like searching for relevant documents.

Classic Approach: Latent Semantic Indexing

bonus!

- Classic methods are based on scores like TF-IDF:
  1. Term frequency: probability of a word occuring within a document.
     - E.g., 7% of words in document i are `the` and 2% of the words are `LeBron`.
  2. Document frequency: probability of a word occuring across documents.
     - E.g., 100% of documents contain `the` and 0.01% have `LeBron`.
  3. TF-IDF: measures like (term frequency)*log $1/$(document frequency).
     - Seeing `LeBron` tells you a lot about the document; seeing `the` tells you nothing.

- Many many many variations exist.

- TF-IDF features are very redundant.
  - Consider TF-IDF of `LeBron`, `Durant`, and `Giannis`.
  - High values of these typically just indicate topic of "basketball".
  - Basically a weighted bag of words.

- We want to find latent factors ("topics") like "basketball".

Modern Approach: Latent Dirichlet Allocation

bonus!

- Latent semantic indexing (LSI) topic model:
  1. Summarize each document by its TF-IDF values.
  2. Run a latent-factor model like PCA or NMF on the matrix.
  3. Treat the latent factors as the "topics".

- LSI has been largely replaced by latent Dirichlet allocation (LDA).
  - Hierarchical Bayesian model of all words in a document.
    - Still ignores word order.
    - Tries to explain all words in terms of topics.
  - The most cited ML paper in the 00s?

- LDA has several components; we'll build up to it by parts.
  - We'll assume all documents have d words and word order doesn't matter.

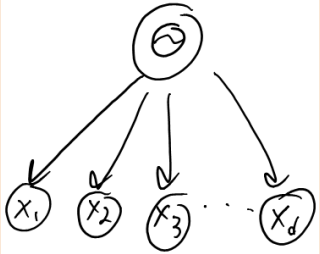
Model 1: Categorical Distribution of Words

bonus!

- Base model: each word x_j comes from the same categorical distribution.

$$p(x_j = \texttt{the}) = \theta_{\texttt{the}} \quad \text{where} \quad \theta_{\text{word}} \geq 0 \quad \text{and} \quad \sum_{\text{word}} \theta_{\text{word}} = 1.$$

- So to generate a document with d words:
  - Sample d words from the categorical distribution.

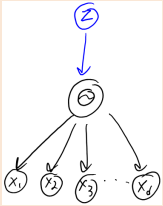


- Drawback: misses that documents are about different "topics."
  - We want the word distribution to depend on the "topics."

Model 2: Mixture of Categorical Distributions

bonus!

- To represent "topics", we'll use a mixture model.
  - Each mixture has its own categorical distribution over words.
    - E.g., the "basketball" mixture will have higher probability of `LeBron`.
- So to generate a document with d words:
  - Sample a topic z from a categorical distribution.
  - Sample d words from categorical distribution z.



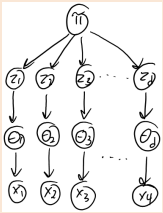
- Similar to a mixture of independent categorical distributions.
  - But we tie categorical distribution across the d variables, given cluster.
- Drawback: misses that documents may be about more than one topic.

Model 3: Multi-Topic Mixture of Categorical

bonus!

- Our third model introduces a new vector of "topic proportions" π.
  - Gives percentage of each topic that makes up the document.
    - E.g., 80% basketball and 20% politics.
  - Called probabilistic latent semantic indexing (PLSI).

- So to generate a document with d words given topic proportions π:
  - Sample d topics z_j from categorical distribution π.
  - Sample a word for each z_j from corresponding categorical distribution.

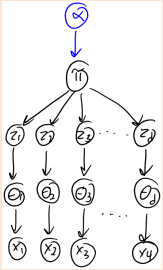


- Similar to HMM where each "time" has own cluster (but no Markov assumption).

Model 4: Latent Dirichlet Allocation

bonus!

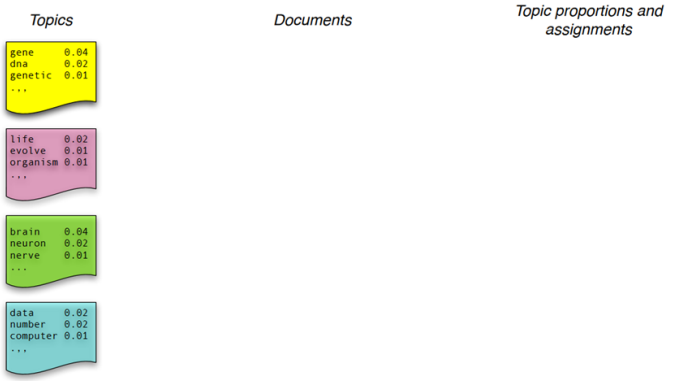
- Latent Dirichlet allocation (LDA) puts a prior on topic proportions.
  - Conjugate prior for categorical is Dirichlet distribution.
- So to generate a document with d words given Dirichlet prior:
  - Sample mixture proportions π from the Dirichlet prior.
  - Sample d topics z_j from categorical distribution π.
  - Sample a word for each z_j from corresponding categorical distribution.



- This is the generative model, typically used with MCMC or variational methods.

bonus!

Latent Dirichlet Allocation (LDA)



http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.

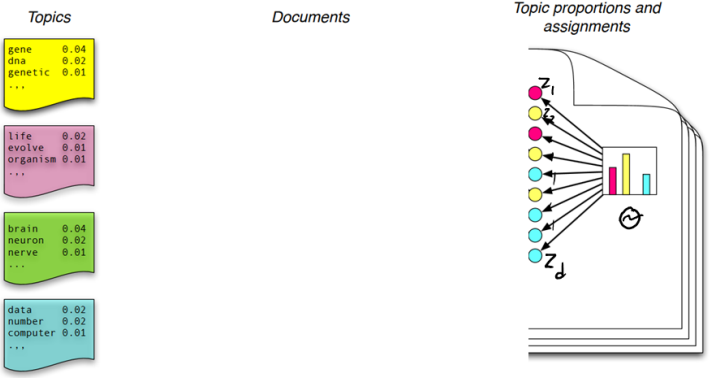
Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.
2. Sample 'd' topics z_j from θ.



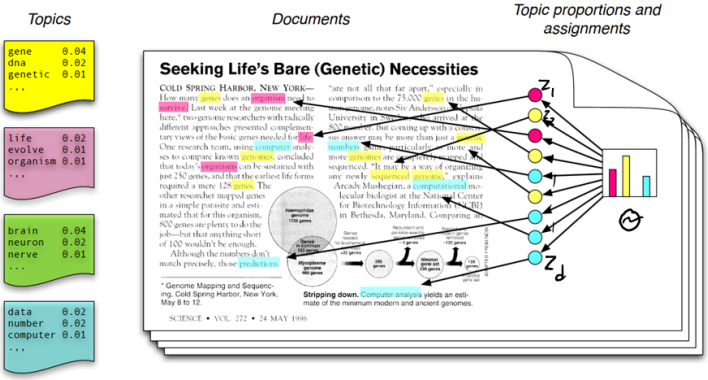
Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.
2. Sample 'd' topics z_j from θ.
3. For each z_j sample a word based on frequencies for topic.

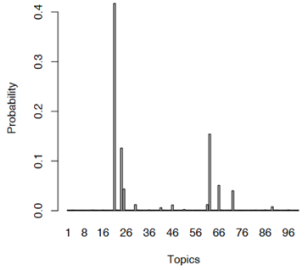


Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!



"Genetics"	"Evolution"	"Disease"	"Computers"
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

Figure 2: **Real inference with LDA.** We fit a 100-topic LDA model to 17,000 articles from the journal *Science*. At left is the inferred topic proportions for the example article in Figure 1. At right are the top 15 most frequent words from the most frequent topics found in this article.

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!

Figure 3: A topic model fit to the *Yale Law Journal*. Here there are twenty topics (the top eight are plotted). Each topic is illustrated with its top most frequent words. Each word's position along the x-axis denotes its specificity to the documents. For example "estate" in the first topic is more specific than "tax."

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!

Health topics in social media:

Non-Ailment Topics						
TV & Movies	**Games & Sports**	**School**	**Conversation**	**Family**	**Transportation**	**Music**
watch	killing	ugh	ill	mom	home	voice
watching	play	class	ok	shes	car	hear
tv	game	school	haha	dad	drive	feelin
killing	playing	read	ha	says	walk	lil
movie	win	test	fine	hes	bus	night
seen	boys	doing	yeah	sister	driving	bit
movies	games	finish	thanks	tell	trip	music
mr	fight	reading	hey	mum	ride	listening
watched	lost	teacher	thats	brother	leave	listen
hi	team	write	xd	thinks	house	sound

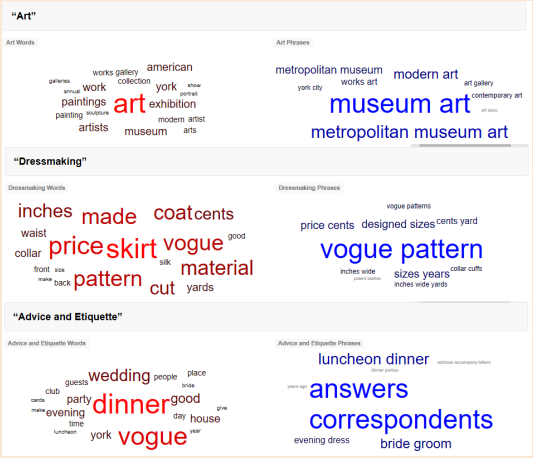
Ailments						
	Influenza-like Illness	**Insomnia & Sleep Issues**	**Diet & Exercise**	**Cancer & Serious Illness**	**Injuries & Pain**	**Dental Health**
General Words	better	night	body	cancer	hurts	dentist
	hope	bed	pounds	help	knee	appointment
	ill	body	gym	pray	ankle	doctors
	soon	ill	weight	awareness	hurt	tooth
	feel	tired	lost	diagnosed	neck	teeth
	feeling	work	workout	prayers	ouch	appt
	day	day	lose	died	leg	wisdom
	flu	hours	days	family	arm	eye
	thanks	asleep	legs	friend	fell	going
	xx	morning	week	shes	left	went
Symptoms	sick	sleep	sore	cancer	pain	infection
	sore	headache	throat	breast	sore	pain
	throat	fall	pain	lung	head	mouth
	fever	insomnia	aching	prostate	foot	ear
	cough	sleeping	stomach	sad	feet	sinus
Treatments	hospital	sleeping	exercise	surgery	massage	surgery
	surgery	pills	diet	hospital	brace	braces
	antibiotics	caffeine	dieting	treatment	physical	antibiotics
	fluids	pill	exercises	heart	therapy	eye
	paracetamol	tylenol	protein	transplant	crutches	hospital

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0103408

Latent Dirichlet Allocation Example

bonus!

Three topics in 100 years of "Vogue" fashion magazine:



http://dh.library.yale.edu/projects/vogue/topics/

Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.

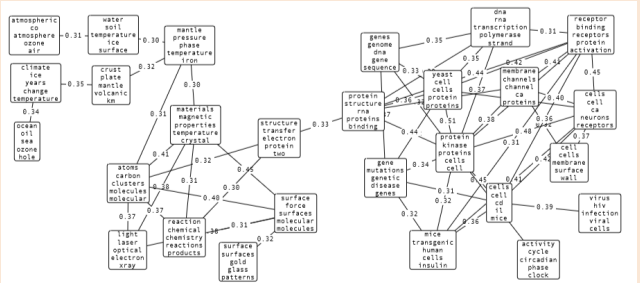


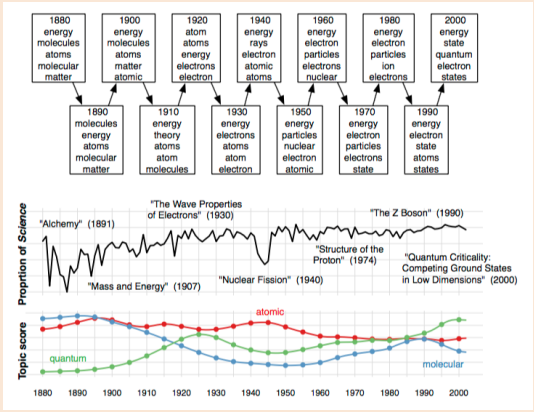
Figure 2: A portion of the topic graph learned from 15,744 OCR articles from *Science*. Each node represents a topic, and is labeled with the five most probable words from its distribution; edges are labeled with the correlation between topics.

http://people.ee.duke.edu/~lcarin/Blei2005CTM.pdf

Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.
  - Can be combined with Markov models to capture dependencies over time.



Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.
  - Can be combined with Markov models to capture dependencies over time.
  - Better word representations like "word2vec" (CPSC 340).
  - Now being applied beyond text, like "cancer mutation signatures":

http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1005657

Discussion of Topic Models

bonus!

- Topic models for analyzing musical keys:

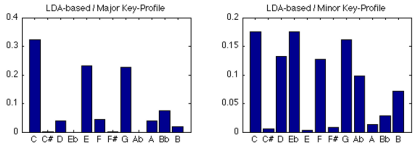


Figure 2: The C major and C minor key-profiles learned by our model, as encoded by the β matrix. Resulting key-profiles are obtained by transposition.

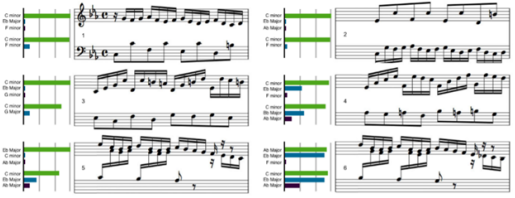


Figure 3: Key judgments for the first 6 measures of Bach's Prelude in C minor, WTC-II. Annotations for each measure show the top three keys (and relative strengths) chosen for each measure. The top set of three annotations are judgments from our LDA-based model; the bottom set of three are from human expert judgments [3].

Monte Carlo Methods for Topic Models

bonus!

- Nasty integrals in topic models:

Inference [edit]

See also: Dirichlet-multinomial distribution

Learning the various distributions (the set of topics, their associated word probabilities, the topic of each word, and the particular topic mixture of each document) is a problem of Bayesian inference. The original paper used a variational Bayes approximation of the posterior distribution;[1] alternative inference techniques use Gibbs sampling[6] and expectation propagation.[7]

Following is the derivation of the equations for collapsed Gibbs sampling, which means φs and θs will be integrated out. For simplicity, in this derivation the documents are all assumed to have the same length N. The derivation is equally valid if the document lengths vary.

According to the model, the total probability of the model is:

$$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{i=1}^{K} P(\varphi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j) P(W_{j,t}|\varphi_{Z_{j,t}}),$$

where the bold-font variables denote the vector version of the variables. First, $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$ need to be integrated out.

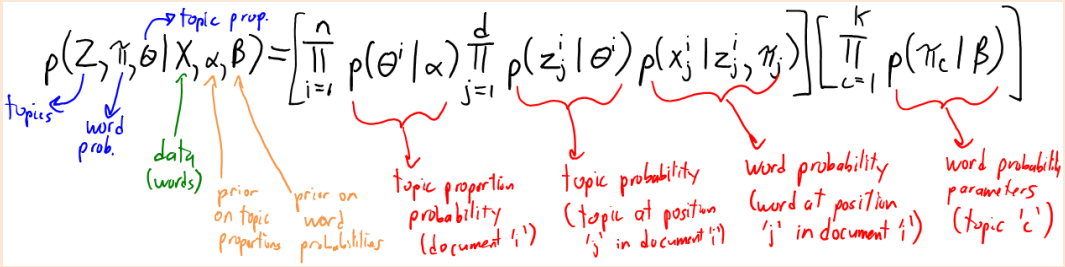
$$\begin{aligned} P(\boldsymbol{Z}, \boldsymbol{W}; \alpha, \beta) &= \int_{\boldsymbol{\theta}} \int_{\boldsymbol{\varphi}} P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta)\, d\boldsymbol{\varphi}\, d\boldsymbol{\theta} \\ &= \int_{\boldsymbol{\varphi}} \prod_{i=1}^{K} P(\varphi_i; \beta) \prod_{j=1}^{M} \prod_{t=1}^{N} P(W_{j,t} \mid \varphi_{Z_{j,t}})\, d\boldsymbol{\varphi} \int_{\boldsymbol{\theta}} \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t} \mid \theta_j)\, d\boldsymbol{\theta}. \end{aligned}$$

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Monte Carlo Methods for Topic Models

bonus!

- How do we actually *use* Monte Carlo for topic models?
- First we write out the posterior:



Monte Carlo Methods for Topic Models

bonus!

- How do we actually *use* Monte Carlo for topic models?
- First we generate samples from the posterior:
  - With Gibbs sampling we alternate between:
    - Sampling topics given word probabilities and topic proportions.
    - Sampling topic proportions given topics and prior parameters α.
    - Sampling word probabilities given topics, words, and prior parameters β.
  - Have a burn-in period, use thinning, try to monitor convergence, and so on.
- Then we use posterior samples to do inference:
  - Distribution of topic proportions for sample i is frequency in samples.
  - To see if words come from same topic, check frequency in samples.

Outline

5 Bonus material on inference

6 Bonus: Topic Models

7 Topic Models

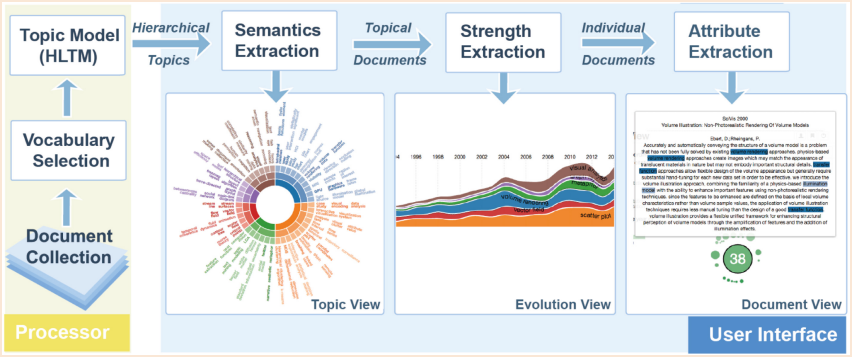
8 Bonus: Restricted Boltzmann Machines

Motivation for Topic Models

bonus!

We want a model of the hidden "factors" making up a set of documents.

- In this context, latent-factor models are called topic models.



https://www.sciencedirect.com/science/article/pii/S2468502X17300074

- "Topics" could be useful for things like searching for relevant documents.

Classic Approach: Latent Semantic Indexing

bonus!

- Classic methods are based on scores like TF-IDF:
  1. Term frequency: probability of a word occuring within a document.
     - E.g., 7% of words in document i are `the` and 2% of the words are `LeBron`.
  2. Document frequency: probability of a word occuring across documents.
     - E.g., 100% of documents contain `the` and 0.01% have `LeBron`.
  3. TF-IDF: measures like (term frequency)*log $1/$(document frequency).
     - Seeing `LeBron` tells you a lot about the document; seeing `the` tells you nothing.

- Many many many variations exist.

- TF-IDF features are very redundant.
  - Consider TF-IDF of `LeBron`, `Durant`, and `Giannis`.
  - High values of these typically just indicate topic of "basketball".
  - Basically a weighted bag of words.

- We want to find latent factors ("topics") like "basketball".

Modern Approach: Latent Dirichlet Allocation

bonus!

- Latent semantic indexing (LSI) topic model:
  1. Summarize each document by its TF-IDF values.
  2. Run a latent-factor model like PCA or NMF on the matrix.
  3. Treat the latent factors as the "topics".

- LSI has been largely replaced by latent Dirichlet allocation (LDA).
  - Hierarchical Bayesian model of all words in a document.
    - Still ignores word order.
    - Tries to explain all words in terms of topics.
  - The most cited ML paper in the 00s?

- LDA has several components; we'll build up to it by parts.
  - We'll assume all documents have d words and word order doesn't matter.

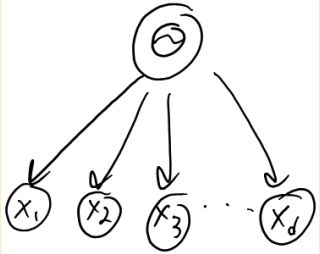
Model 1: Categorical Distribution of Words

bonus!

- Base model: each word x_j comes from the same categorical distribution.

$$p(x_j = \texttt{the}) = \theta_{\texttt{the}} \quad \text{where} \quad \theta_{\text{word}} \geq 0 \quad \text{and} \quad \sum_{\text{word}} \theta_{\text{word}} = 1.$$

- So to generate a document with d words:
  - Sample d words from the categorical distribution.

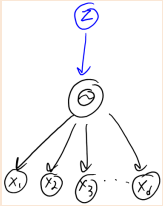


- Drawback: misses that documents are about different "topics."
  - We want the word distribution to depend on the "topics."

Model 2: Mixture of Categorical Distributions

bonus!

- To represent "topics", we'll use a mixture model.
  - Each mixture has its own categorical distribution over words.
    - E.g., the "basketball" mixture will have higher probability of `LeBron`.
- So to generate a document with d words:
  - Sample a topic z from a categorical distribution.
  - Sample d words from categorical distribution z.



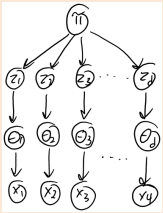
- Similar to a mixture of independent categorical distributions.
  - But we tie categorical distribution across the d variables, given cluster.
- Drawback: misses that documents may be about more than one topic.

Model 3: Multi-Topic Mixture of Categorical

bonus!

- Our third model introduces a new vector of "topic proportions" π.
  - Gives percentage of each topic that makes up the document.
    - E.g., 80% basketball and 20% politics.
  - Called probabilistic latent semantic indexing (PLSI).

- So to generate a document with d words given topic proportions π:
  - Sample d topics z_j from categorical distribution π.
  - Sample a word for each z_j from corresponding categorical distribution.

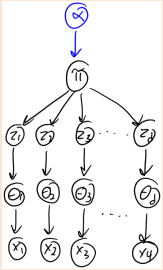


- Similar to HMM where each "time" has own cluster (but no Markov assumption).

Model 4: Latent Dirichlet Allocation

bonus!

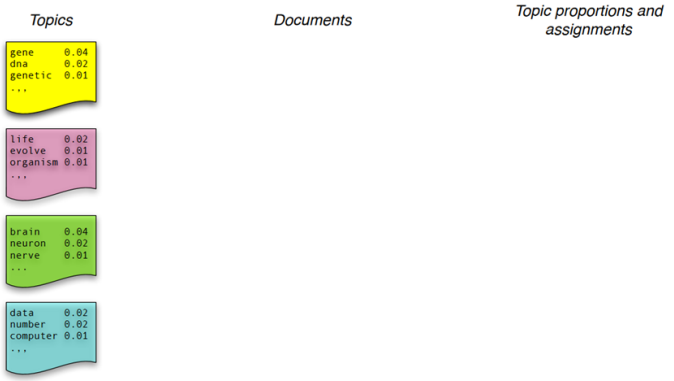
- Latent Dirichlet allocation (LDA) puts a prior on topic proportions.
  - Conjugate prior for categorical is Dirichlet distribution.
- So to generate a document with d words given Dirichlet prior:
  - Sample mixture proportions π from the Dirichlet prior.
  - Sample d topics z_j from categorical distribution π.
  - Sample a word for each z_j from corresponding categorical distribution.



- This is the generative model, typically used with MCMC or variational methods.

bonus!

Latent Dirichlet Allocation (LDA)



http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.

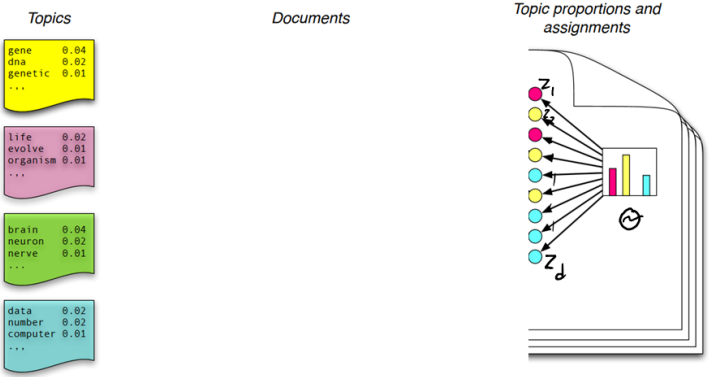
Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.
2. Sample 'd' topics z_j from θ.



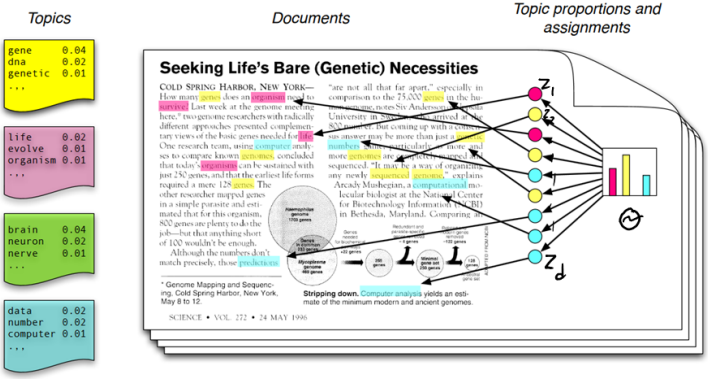
Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

bonus!

Latent Dirichlet Allocation (LDA)

1. Sample topic proportions θ from Dirichlet.
2. Sample 'd' topics z_j from θ.
3. For each z_j sample a word based on frequencies for topic.

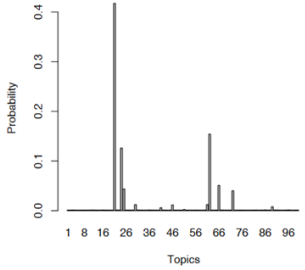


Each topic is like a "principal component" or "latent factor"

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!



"Genetics"	"Evolution"	"Disease"	"Computers"
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

Figure 2: **Real inference with LDA.** We fit a 100-topic LDA model to 17,000 articles from the journal *Science*. At left is the inferred topic proportions for the example article in Figure 1. At right are the top 15 most frequent words from the most frequent topics found in this article.

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!

Figure 3: A topic model fit to the *Yale Law Journal*. Here there are twenty topics (the top eight are plotted). Each topic is illustrated with its top most frequent words. Each word's position along the x-axis denotes its specificity to the documents. For example "estate" in the first topic is more specific than "tax."

http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Latent Dirichlet Allocation Example

bonus!

Health topics in social media:

Non-Ailment Topics						
TV & Movies	**Games & Sports**	**School**	**Conversation**	**Family**	**Transportation**	**Music**
watch	killing	ugh	ill	mom	home	voice
watching	play	class	ok	shes	car	hear
tv	game	school	haha	dad	drive	feelin
killing	playing	read	ha	says	walk	lil
movie	win	test	fine	hes	bus	night
seen	boys	doing	yeah	sister	driving	bit
movies	games	finish	thanks	tell	trip	music
mr	fight	reading	hey	mum	ride	listening
watched	lost	teacher	thats	brother	leave	listen
hi	team	write	xd	thinks	house	sound

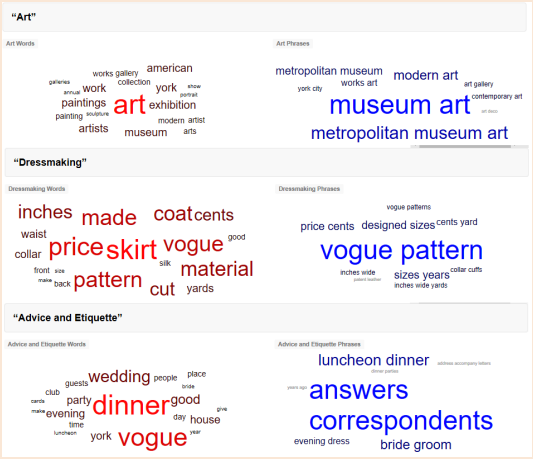
Ailments						
	Influenza-like Illness	**Insomnia & Sleep Issues**	**Diet & Exercise**	**Cancer & Serious Illness**	**Injuries & Pain**	**Dental Health**
General Words	better	night	body	cancer	hurts	dentist
	hope	bed	pounds	help	knee	appointment
	ill	body	gym	pray	ankle	doctors
	soon	ill	weight	awareness	hurt	tooth
	feel	tired	lost	diagnosed	neck	teeth
	feeling	work	workout	prayers	ouch	appt
	day	day	lose	died	leg	wisdom
	flu	hours	days	family	arm	eye
	thanks	asleep	legs	friend	fell	going
	xx	morning	week	shes	left	went
Symptoms	sick	sleep	sore	cancer	pain	infection
	sore	headache	throat	breast	sore	pain
	throat	fall	pain	lung	head	mouth
	fever	insomnia	aching	prostate	foot	ear
	cough	sleeping	stomach	sad	feet	sinus
Treatments	hospital	sleeping	exercise	surgery	massage	surgery
	surgery	pills	diet	hospital	brace	braces
	antibiotics	caffeine	dieting	treatment	physical	antibiotics
	fluids	pill	exercises	heart	therapy	eye
	paracetamol	tylenol	protein	transplant	crutches	hospital

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0103408

Latent Dirichlet Allocation Example

bonus!

Three topics in 100 years of "Vogue" fashion magazine:



http://dh.library.yale.edu/projects/vogue/topics/

Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.

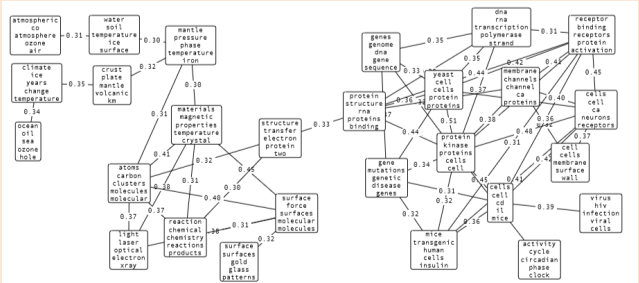


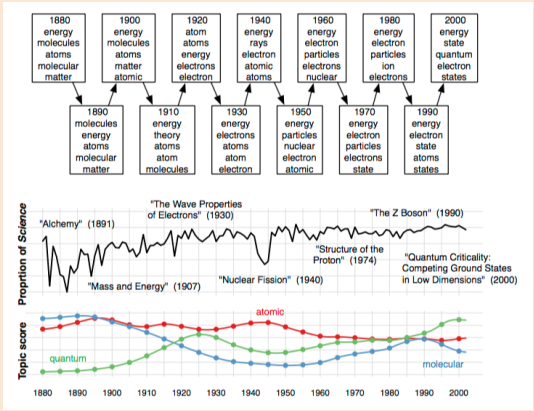
Figure 2: A portion of the topic graph learned from 15,744 OCR articles from *Science*. Each node represents a topic, and is labeled with the five most probable words from its distribution; edges are labeled with the correlation between topics.

http://people.ee.duke.edu/~lcarin/Blei2005CTM.pdf

Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.
  - Can be combined with Markov models to capture dependencies over time.



http://menome.com/wp/wp-content/uploads/2014/12/Blei2011.pdf

Discussion of Topic Models

bonus!

- There are *many* extensions of LDA:
  - We can put prior on the number of words (like Poisson).
  - Correlated and hierarchical topic models learn dependencies between topics.
  - Can be combined with Markov models to capture dependencies over time.
  - Better word representations like "word2vec" (CPSC 340).
  - Now being applied beyond text, like "cancer mutation signatures":

http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1005657

Discussion of Topic Models

bonus!

- Topic models for analyzing musical keys:

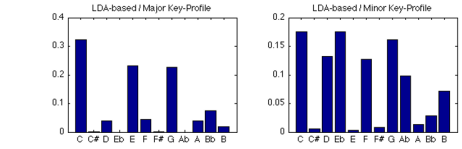


Figure 2: The C major and C minor key-profiles learned by our model, as encoded by the β matrix. Resulting key-profiles are obtained by transposition.

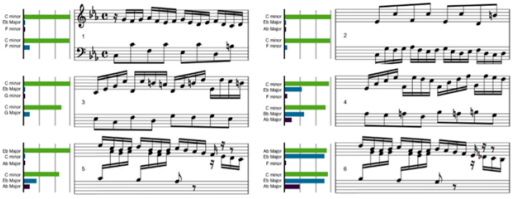


Figure 3: Key judgments for the first 6 measures of Bach's Prelude in C minor, WTC-II. Annotations for each measure show the top three keys (and relative strengths) chosen for each measure. The top set of three annotations are judgments from our LDA-based model; the bottom set of three are from human expert judgments [3].

Monte Carlo Methods for Topic Models

bonus!

- Nasty integrals in topic models:

Inference [edit]

See also: Dirichlet-multinomial distribution

Learning the various distributions (the set of topics, their associated word probabilities, the topic of each word, and the particular topic mixture of each document) is a problem of Bayesian inference. The original paper used a variational Bayes approximation of the posterior distribution;[1] alternative inference techniques use Gibbs sampling[6] and expectation propagation.[7]

Following is the derivation of the equations for collapsed Gibbs sampling, which means φs and θs will be integrated out. For simplicity, in this derivation the documents are all assumed to have the same length N. The derivation is equally valid if the document lengths vary.

According to the model, the total probability of the model is:

$$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{i=1}^{K} P(\varphi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j) P(W_{j,t}|\varphi_{Z_{j,t}}),$$

where the bold-font variables denote the vector version of the variables. First, $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$ need to be integrated out.

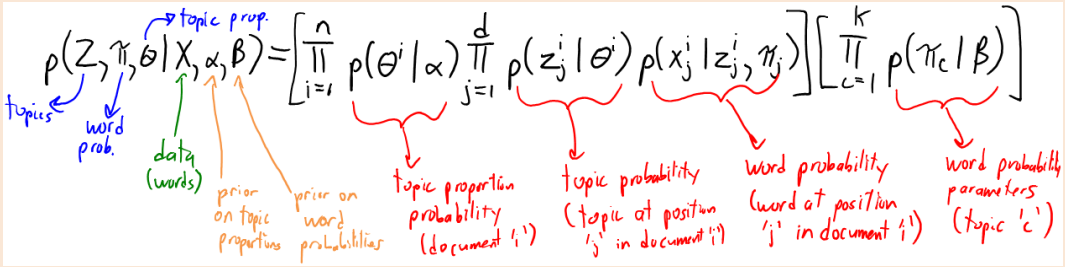
$$\begin{aligned} P(\boldsymbol{Z}, \boldsymbol{W}; \alpha, \beta) &= \int_{\boldsymbol{\theta}} \int_{\boldsymbol{\varphi}} P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta)\, d\boldsymbol{\varphi}\, d\boldsymbol{\theta} \\ &= \int_{\boldsymbol{\varphi}} \prod_{i=1}^{K} P(\varphi_i; \beta) \prod_{j=1}^{M} \prod_{t=1}^{N} P(W_{j,t} \mid \varphi_{Z_{j,t}})\, d\boldsymbol{\varphi} \int_{\boldsymbol{\theta}} \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t} \mid \theta_j)\, d\boldsymbol{\theta}. \end{aligned}$$

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Monte Carlo Methods for Topic Models

bonus!

- How do we actually *use* Monte Carlo for topic models?
- First we write out the posterior:



Monte Carlo Methods for Topic Models

bonus!

- How do we actually *use* Monte Carlo for topic models?
- First we generate samples from the posterior:
  - With Gibbs sampling we alternate between:
    - Sampling topics given word probabilities and topic proportions.
    - Sampling topic proportions given topics and prior parameters α.
    - Sampling word probabilities given topics, words, and prior parameters β.
  - Have a burn-in period, use thinning, try to monitor convergence, and so on.
- Then we use posterior samples to do inference:
  - Distribution of topic proportions for sample i is frequency in samples.
  - To see if words come from same topic, check frequency in samples.

Outline

5 Bonus material on inference

6 Bonus: Topic Models

7 Topic Models

8 Bonus: Restricted Boltzmann Machines

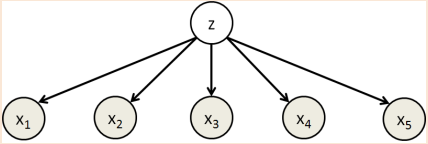
Mixture of Bernoullis Models

bonus!

- Recall the mixture of Bernoullis models:

$$p(x) = \sum_{c=1}^{k} p(z = c) \prod_{j=1}^{d} p(x_j \mid z = c).$$

- Given z, each variable x_j comes from a product of Bernoullis

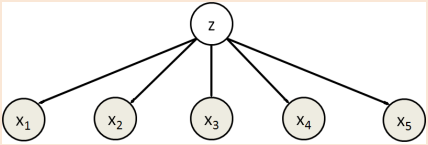


- This is enough to model *any* multivariate binary distribution.
  - But not an efficient representation: number of cluster might need to be huge.
    - Need to learn each cluster independently (no "shared" information across clusters).

Mixture of Independents as a UGM

bonus!

- The mixture of independents assumptions can be represented as a UGM:



  - "The x_j are independent given the cluster z".
- A log-linear parameterization for $x_j \in \{-1, +1\}$ and $z \in \{-1, +1\}$ could be

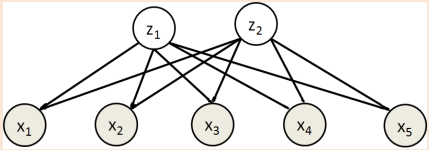
$$\phi_j(x_j) = \exp(w_j x_j), \quad \phi_z(z) = \exp(vz), \quad \phi_{j,z}(x_j, z) = \exp(w_j x_j z).$$

- We have three types of parameters:
  - Weight w_j in ϕ_j affects probability of $x_j = 1$ (independent of cluster).
  - Weight v in ϕ_z affecst probability that $z_j = 1$ (prior for cluster).
  - Weight w_j in $\phi_{j,z}$ affects probability that x_j and z are same.
    - Can encourage each binary variable to be same or different than "cluster sign".

"Double Clustering" Model

bonus!

- Now consider adding a second binary cluster variable:



  - "The x_j are independent given both cluster variables z_1 and z_2".
- A log-linear parameterization for $x_j \in \{-1, +1\}$ and $z_c \in \{-1, +1\}$ could be

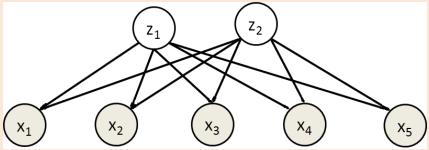
$$\phi_j(x_j) = \exp(w_j x_j), \quad \phi_c(z_c) = \exp(v_c z_c), \quad \phi_{j,c}(x_j, z_c) = \exp(w_{jc} x_j z)$$

- We have three types of parameters:
  - Weight w_j in ϕ_j affects probability of $x_j = 1$ (independent of cluster).
  - Weight v_c in ϕ_z affecst probability that $z_c = 1$ (prior for cluster variable).
  - Weight w_{jc} in $\phi_{j,z}$ affects probability that x_j and z_c are same.
    - Can encourage each binary variable to be same or different than "cluster variable".

"Double Clustering" Model

bonus!

- Now consider adding a second binary cluster variable:



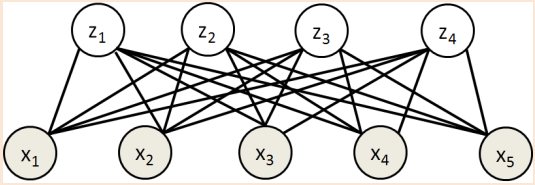
- Have we gained anything?
  - We have 4 clusters based on two hidden variables.
  - Each cluster shares parameters with 2 of the other clusters.

- Hope is to achieve some degree of composition
  - Don't need to re-learn basic things about the x_j in each cluster.
  - Maybe one hidden z_c models clusters, and another models correlations.
    - So that when you use both, you can capture both aspects.

Restricted Boltzmann Machines (RBMs)

bonus!

- Now consider adding two more binary latent variables:

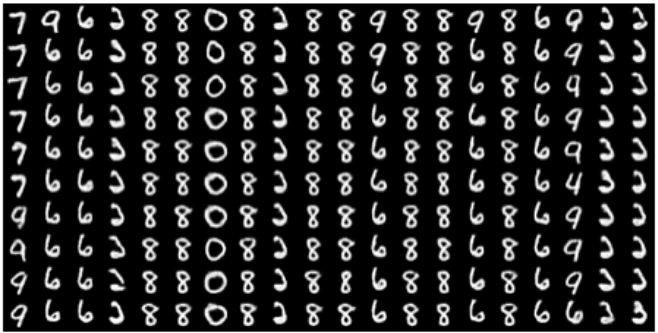


- Now we have 16 clusters, in general we'll have 2^k with k hidden binary nodes.
  - This discrete latent-factors give combinatorial number of mixtures.
    - You can think of each z_c as a "part" that can be included or not ("binary PCA").
- This is called a restricted Boltzmann machine (RBM).
  - A Boltzmann machine is a UGM with binary hidden variables.
- It is restricted because all edges are between "visible" x_j and "hidden" z_c.
  - If we know the x_j, then the z_c are independent.
  - If we know the z_c, then the x_j are independent.
  - Inference on both x and z is hard.
    - But we could alternate between Gibbs sampling of all x and all z variables.

Generating Digits with RBMs

bonus!

Here are the samples generated by the RBM after training. Each row represents a mini-batch of negative particles (samples from independent Gibbs chains). 1000 steps of Gibbs sampling were taken between each of those rows.

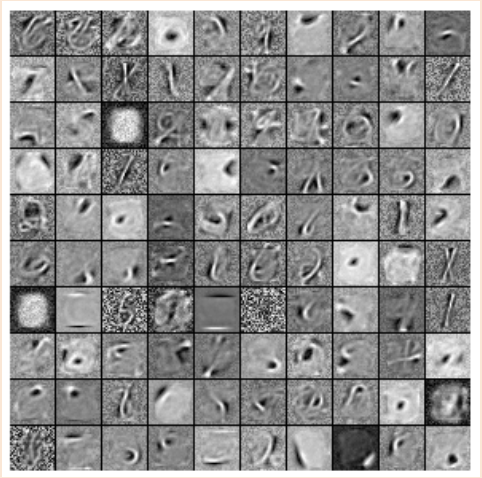


http://deeplearning.net/tutorial/rbm.html

Generating Digits with RBMs

bonus!

Visualizing each z_c's interaction parameters (w_{jc} for all j) as images:



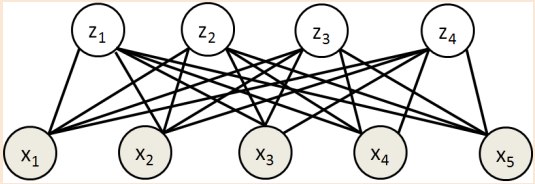
http://deeplearning.net/tutorial/rbm.html

Restricted Boltzmann Machines

bonus!

- The RBM graph structure leads to a joint distribution of the form

$$p(x,z) = \frac{1}{Z}\left(\prod_{j=1}^{d}\phi_j(x_j)\right)\left(\prod_{c=1}^{k}\phi_c(z_c)\right)\left(\prod_{j=1}^{d}\prod_{c=1}^{k}\phi_{jc}(x_j,z_c)\right).$$



- RBMs usually use a log-linear parameterization like

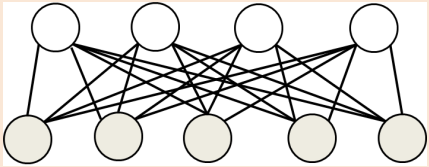
$$p(x,z) \propto \exp\left(\sum_{j=1}^{d} w_j x_j + \sum_{c=1}^{k} v_c z_c + \sum_{j=1}^{d}\sum_{c=1}^{k} w_{jc} x_j z_c\right),$$

for parameters w_j, v_c, and w_{jc} (variants exist for non-binary x_j).

Learning UGMs with Hidden Variables

bonus!

- For RBMs we have hidden variables:



- With hidden ("nuissance") variables z the observed likelihood has the form

$$p(x) = \sum_z p(x,z) = \sum_z \frac{\tilde{p}(x,z)}{Z}$$
$$= \frac{1}{Z} \underbrace{\sum_z \tilde{p}(x,z)}_{Z(x)} = \frac{Z(x)}{Z},$$

where $Z(x)$ is the partition function of the conditional UGM given x.

- $Z(x)$ is cheap in RBMs because the z are independent given x.

Learning UGMs with Hidden Variables

bonus!

- This gives an observed NLL of the form

$$-\log p(x) = -\log(Z(x)) + \log Z,$$

  where $Z(x)$ sums over hidden z values, and Z sums over z and x.

- The second term is convex but the first term is non-convex.
  - This is expected when we have hidden variables.

- With a log-linear parameterization, the gradient has the form

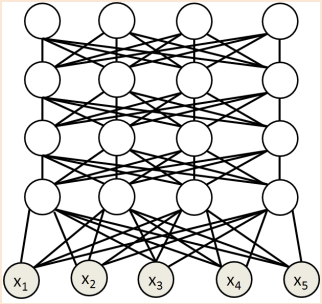
$$-\nabla \log p(x) = -\mathop{\mathbb{E}}_{z|x}[F(X,Z)] + \mathop{\mathbb{E}}_{z,x}[F(X,Z)].$$

- For RBMs, first term is cheap due to independence of z given x.
- We can approximate second term using block Gibbs sampling.
  - For other problems, you would also need to approximate first term.

Deep Boltzmann Machines

bonus!

- 15 years ago, a hot topic was "stacking RBMs", as in deep Boltzmann Machine:



- Part of the motivation for people to re-consider "deep" models.
- Model above allows block Gibbs sampling "by layer".
  - Variables in layer are conditionally independent given layer above and below.

Deep Boltzmann Machines

bonus!

- Performance of deep Boltzmann machine on NORB data:

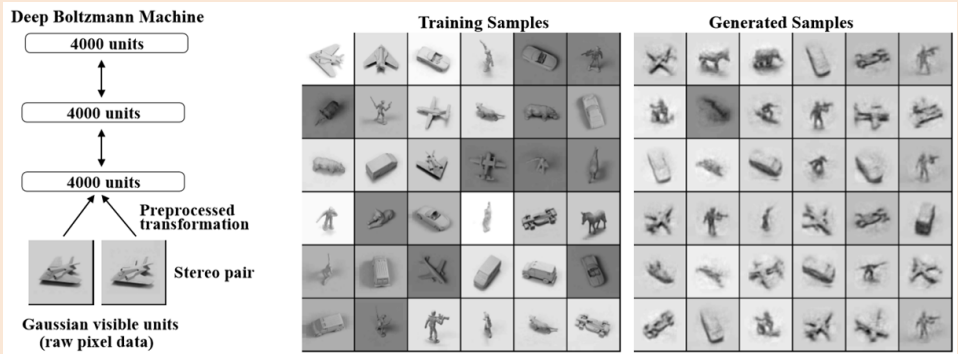


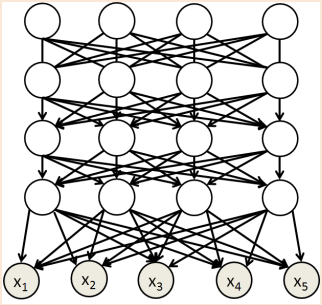
Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

http://www.cs.toronto.edu/~fritz/absps/dbm.pdf

Deep Belief Networks

bonus!

- There were also deep belief networks where RBM outputs DAG layers.



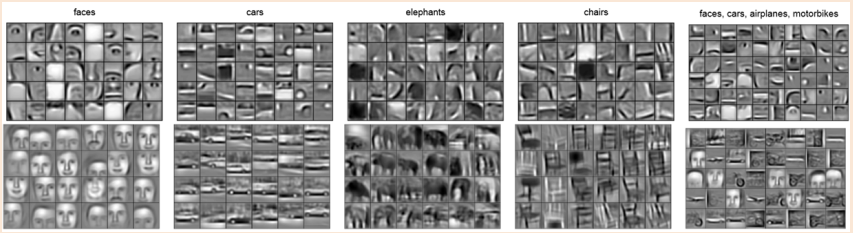
- More difficult to train and do inference due to explaining away.
- Though easier to sample using ancestral sampling.

Cool Pictures Motivation for Deep Learning

bonus!

- First layer of z_i in a convolutional deep belief network:

- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

- Many classes use these particular images to motivate deep neural networks
  - But they're not from a neural network: they're from a deep DAG model