CPSC 540: Machine Learning

Latent Graphical Models

Mark Schmidt

University of British Columbia

Winter 2018

Last Time: Learning Log-Linear UGMs

- We discussed log-linear parameterization of UGMs,

$$\phi_j(s) = \exp(w_{j,s}), \quad \phi_{jk}(s, s') = \exp(w_{j,k,s,s'}), \quad \phi_{jkl}(s, s', s'') = \exp(w_{j,k,l,s,s',s''}).$$

- The likelihood of an example x given parameter w is given by

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

and the feature functions $F(x)$ count the number of times we use each w_j.

- This leads to a convex NLL of the form

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

Log-Linear UGM Gradient

- Gradient in log-linear UGM with respect to parameter w_j is (bonus)

$$\nabla_{w_j} f(w) = -F_j(x) + \mathbb{E}[F_j(x)],$$

and observe that derivative of $\log(Z)$ is expected value of feature.

- Example of $\phi_{10,3} = \exp(w_{10,3})$ (potential that feature 10 is in state 3).

- Averaging over n examples, the gradient with no parameter tieing is given by

$$\nabla_{w_{10,3}} f(w) = \underbrace{-\frac{1}{n}\left[\sum_{i=1}^{n} I[x_{10}^i = 3]\right]}_{\text{frequency in data}} + \underbrace{p(x_{10} = 3)}_{\text{model "frequency"}}.$$

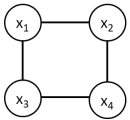
  - So if $\nabla_{w_{10,3}} = 0$, probabilities match frequencies in training data.
  - At MLE, you match the frequencies of all the potentials in the training data.
- But computing gradient requires inference (computing marginals like $p(x_{10} = 3)$).

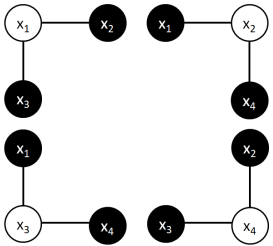
Approximate Learning: Alternate Objectives

- One way to avoid code of inference is to change the objective:
  - Pseudo-likelihood (fast, convex, and crude):

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

which turns learning into d single-variable problems (similar to DAGs).





  - Structured SVMs: generalization of SVMs that only requires decoding (later).

Approximate Learning: Approximate Marginals

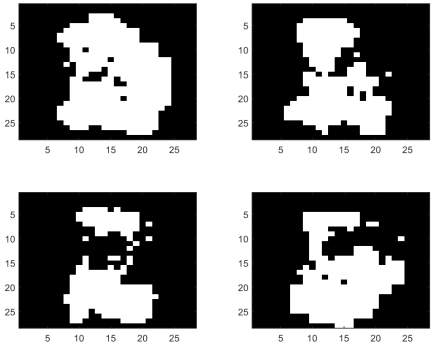
- Alternately, we can use approximate inference to use NLL:
  - Monte Carlo approximation of $\mathbb{E}[F_j(x)]$ given current parameters w:

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

    - Simple method: generate lots of samples to approximate gradient given w, then update w (many samples per iteration).
    - Younes algorithm: alternate between steps of Gibbs sampling and stochastic gradient, using 1 sample per iteration ("persistent contrastive divergence" in deep learning). (SG updates w, Gibbs updates x)
  - Deterministic variational approximations of $\mathbb{E}[F_j(x)]$ can alternately be used (later).

Pairwise UGM on MNIST Digits

- Samples from a lattice-structured pairwise UGM:



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

Digression: Structure Learning in UGMs

- 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 wet $w_{ij} = 0$, it sets $\phi_{ij}(x_i, x_j) = 1$ for all x_i and x_j.
  - This is equivalent to removing the edge.

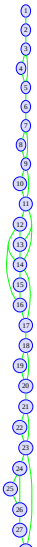
- L1-regularization of w_{ij} values performs structure learning in UGM.

- 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.

Structure Learning on Rain Data

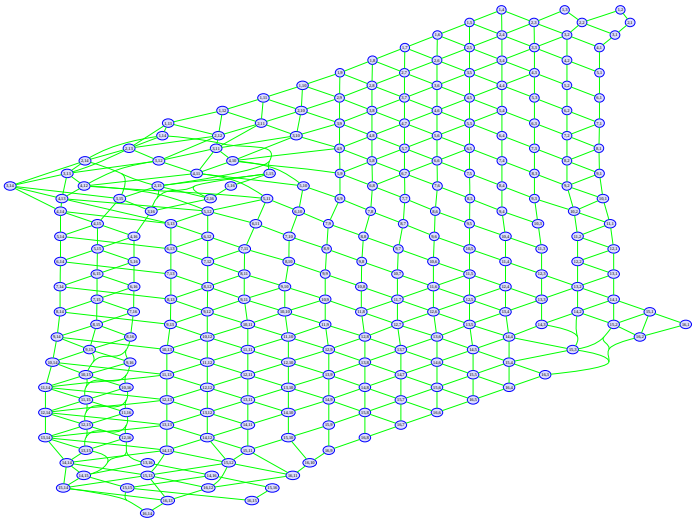
Large λ (and optimal tree):

Small λ:



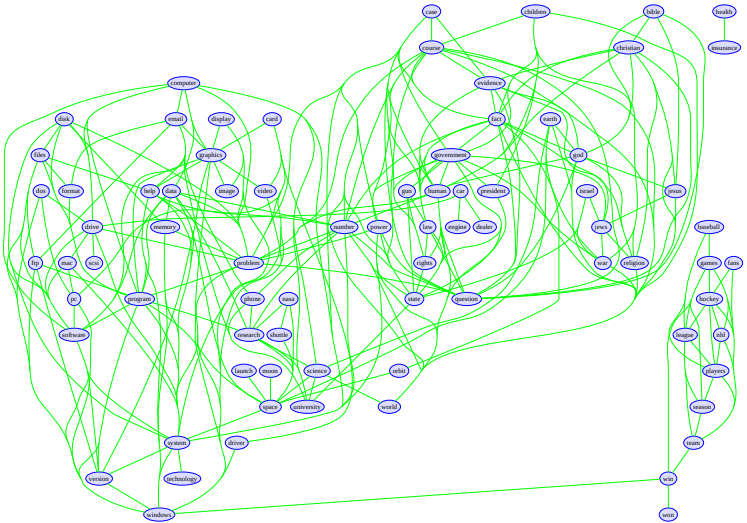
Structure Learning on USPS Digits

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



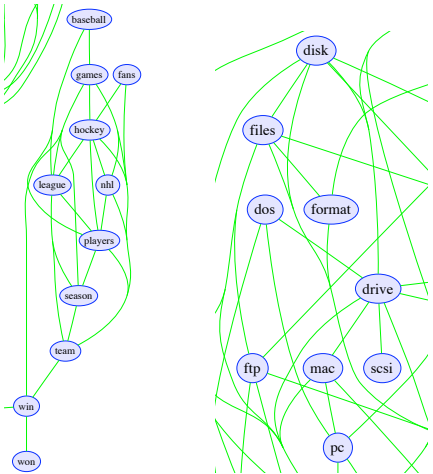
Structure Learning on News Words

Group-L1 on newsgroups data:



Structure Learning on News Words

Group-L1 on newsgroups data:



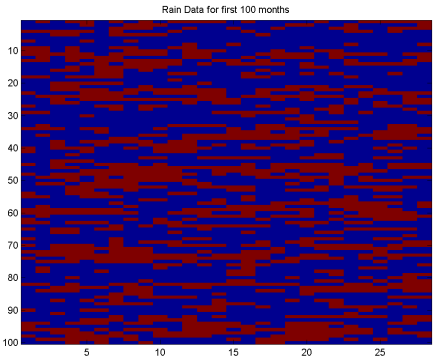
Outline

Where we are where we're going...

- Last n lectures: four topics related to density estimation:
  1. Mixture models can model clusters in the data.
  2. Latent-factor models consider interacting hidden factors in the data.
  3. Graphical models can model direct dependencies between variables.
  4. Approximate inference is needed when probabilities are too complicated
- Each has many applications, but they're limited/boring on their own.
- But by combining them we get very powerful models.
  - Next time we'll start combining them with supervised learning tricks from 340.

Back to the Rain Data

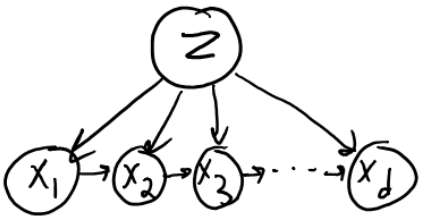
- We previously considered the "Vancouver Rain" data:



- We said that a homogeneous Markov chain is a good model:
  - Captures direct dependency between adjcaent days.

Back to the Rain Data

- But doesn't it rain less in the summer?
- There are hidden clusters in the data not captured by the Markov chain.
  - But mixture of independent models are inefficient at representing direct dependency.
- Mixture of Markov chains could capture direct dependence *and* clusters.



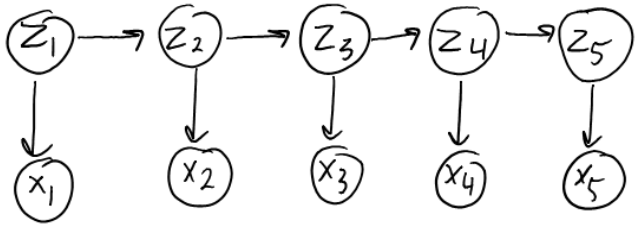
- Cluster z chooses which homogeneous Markov chain parameters to use.
  - We could learn that we're more likely to have rain in winter.
  - Graph has treewidth of 2: exact inference and EM will be cheap.

Back to the Rain Data

- The rain data is artificially divideded into months.
- Consider viewing rain data as one very long sequence ($n = 1$).
- This isn't an issue for homogeneous Markov chains due to parameter tieing.
- But a mixture doesn't make sense when $n = 1$.
- We want different parts of sequence to come from different clusters.
- One way to address this:
  - Let each day have it's own cluster.
  - Have a Markov dependency between cluster values of adjacent days.

Hidden Markov Models

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

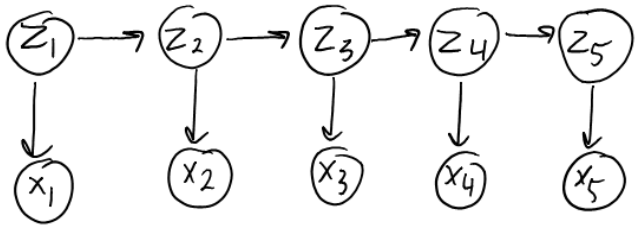


$$p(x_1, x_2, \ldots, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j).$$

- We're going to learn clusters z_j and the hidden dynamics.
  - 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".
  - 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 hidden Markov chain.



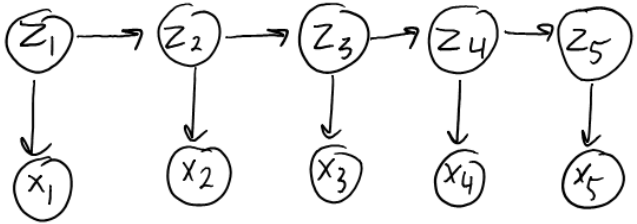
$$p(x_1, x_2, \ldots, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j).$$

- Inference is easy in this model: it's a tree.

- Learning with EM is also easy due to chain-structured z_j dependence:
  - Convert to UGM, conditioning on x_j gives a chain, run forward-backward.

Hidden Markov Models

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



- Note that the x_j can be continuous even with discrete clusters z_j.

- 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.

- Variants of HMMs are probably the most-used time-series model...

Applications of HMMs and Kalman Filters

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]

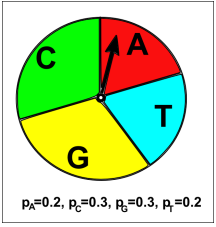
- Attitude and Heading Reference Systems
- Autopilot
- Battery state of charge (SoC) estimation[39][40]
- Brain-computer interface
- Chaotic signals
- Tracking and Vertex Fitting of charged particles in Particle Detectors[41]
- Tracking of objects in computer vision
- Dynamic positioning
- Economics, 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]

Example: Modeling DNA Sequences

- A nice demo of independent vs. Markov vs. HMMs for DNA sequences:
  - http://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/chapter10.html

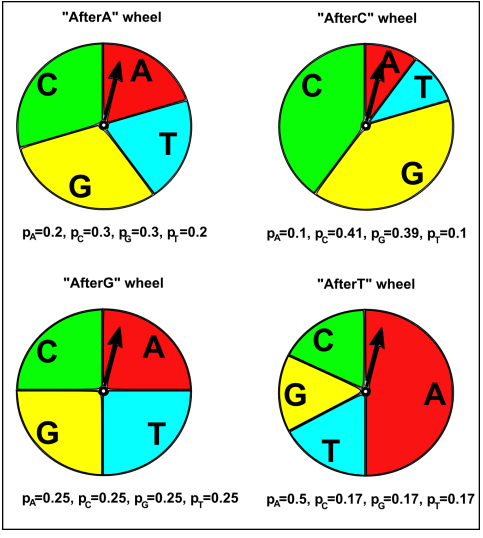
https://www.tes.com/lessons/WE5E9RncBhieAQ/dna

- Independent model for elements of sequence:



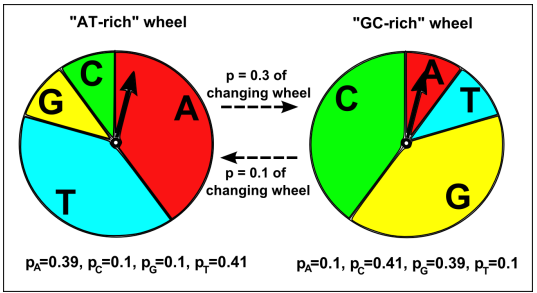
Example: Modeling DNA Sequences

- Markov model for elements of sequence (dependence on previous symbol):



Example: Modeling DNA Sequences

- Hidden Markov model (HMM) for elements of sequence:



- Can reflect that probabilities are different in different regions.

- You probably get a better model by consider hidden Markov and visible Markov.
  - Would have treewidth 2.

Who is Guarding Who?

- There is a lot of data on offense of NBA basketball players.
  - Every point and assist is recorded, more scoring gives more wins and $$$.
- But how do we measure defense?
  - We need to know who each player is guarding.

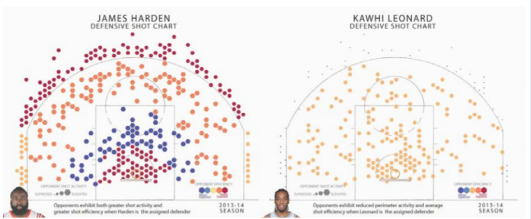


Figure 2a. Graphical depiction of a defender's volume (size) and disruption scores (color). Kawhi Leonard tends to suppress shots on the perimeter. More comparisons are provided in the Appendix.

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

Outline

“THE REVOLUTION WILL NOT BE SUPERVISED” PROMISES FACEBOOK’S YANN LECUN IN KICKOFF AI SEMINAR

POSTED MARCH 6TH, 2018

« PRESS ROOM Facebook Twitter Print

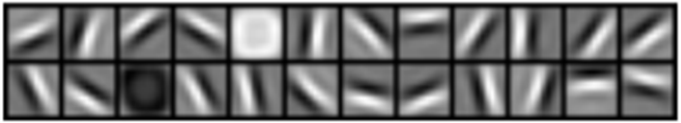
http://engineering.nyu.edu/news/2018/03/06/revolution-will-not-be-supervised-promises-facebooks-yann-lecun-kickoff-ai-seminar

Deep Density Estimation

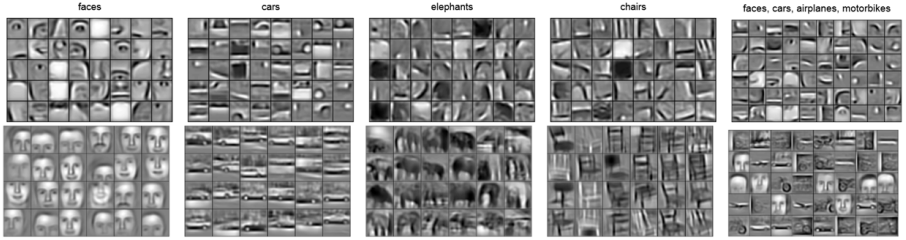
- In 340 we discussed deep learning methods for supervised learning.
- Does it make sense to talk about deep unsupervised learning?
- Standard argument:
  - Human learning seems to be mostly unsupervised.
  - Supervision gives limited feedback: bits in a class label vs. an image.
  - Could we learn unsupervised models with much less data?
- Deep belief networks started modern deep learning movement (2006).

Cool Pictures Motviation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



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



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

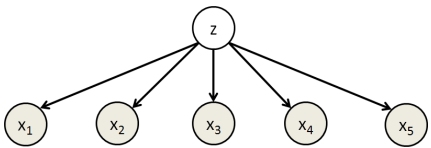
- Many classes use these partcular images to motivate deep neural networks.
  - But they're not from a neural network: they're from a deep belief network.

Mixture of Independent Models

- Recall the mixture of independent models:

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

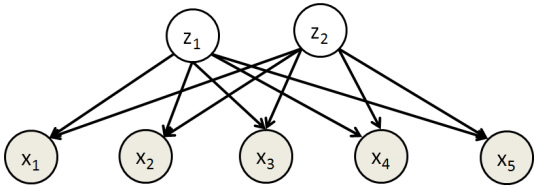
- Given z, each variable x_j comes from some "nice" distribution.



- This is enough to model *any* distribution.
  - Just need to know cluster of example x and distribution of x_j given z.
  - But not an efficient representation: number of cluster might need to be huge.

Latent DAG Model

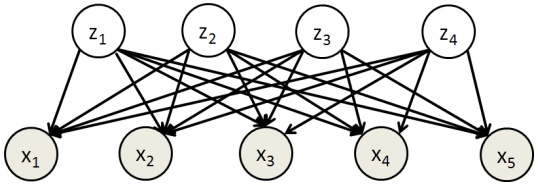
- Consider the following model with binary z_1 and z_2:



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

Latent DAG Model

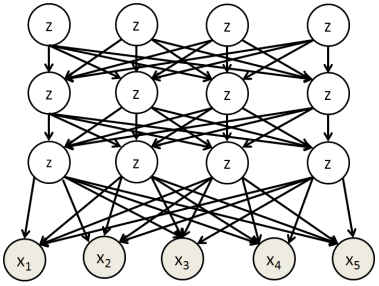
- Consider the following model:



- 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.
  - We'll assume $p(x_j \mid z_1, z_2, z_3, z_4)$ is a linear model (Gaussian, logistic, etc.).
    - Distributed representation where x is made of parts z.
    - With d visible x_j and k hidden z_j, we only have dk parameters.

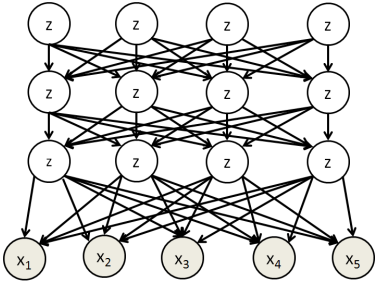
Deep Belief Networks

- Deep belief networks add more binary hidden layers:



- Data is at the bottom.
- First hidden layer could be "basic ingredients".
- Second hidden layer could be general "parts".
- Third hidden layer could be "abstract concept".

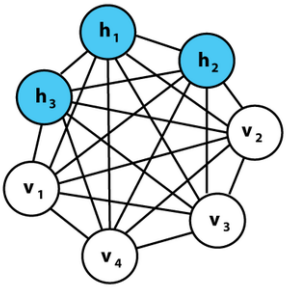
Deep Belief Networks



- If we were conditioning on *top* layer:
  - Sampling would be easy.
- But we're conditioning on the *bottom* layer:
  - Everything is hard.
  - There is combinatorial "explaining away".
- Common training method:
  - Greedy "layerwise" training as a restricted Boltzmann machine.

Boltzmann Machines

- Boltzmann machines are UGMs with binary latent variables:

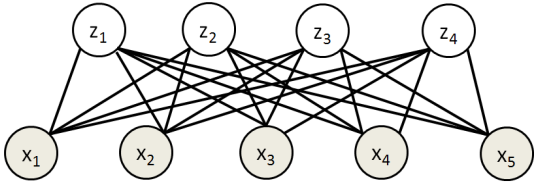


https://en.wikipedia.org/wiki/Boltzmann_machine

- Yet another latent-variable model for density estimation.
  - Hidden variables again give a combinatorial latent representation.
- Hard to do anything in this model, even if you know all the z.

Restricted Boltzmann Machine

- By restricting graph structure, some things get easier:
  - Restricted Boltzmann machines (RBMs): edges only between the x_j and z_c.



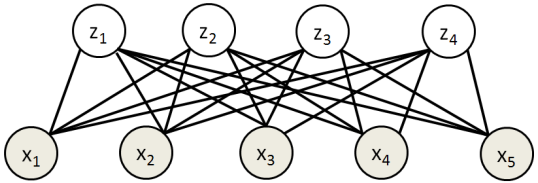
- Bipartite structure allows block Gibbs sampling given one type of variable:
  - Conditional UGM is disconnected.

- Given visible x, we can sample each z_c independently.
- Given hidden z, we can sample each x_j independently.

Restricted Boltzmann Machines

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

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



- RBMs usually use a log-linear parameterization like

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

for parameters w_j, v_c, and w_{jc}.

Learning UGMs with Hidden Variables

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

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

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

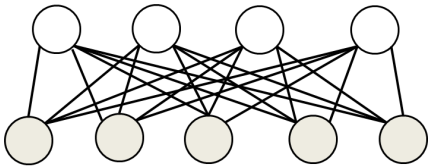
- This gives an observed NLL of the form

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

- The second term is convex but the first term is non-convex.
  - In RBMs, $Z(x)$ is cheap due to independence of z given x.
  - For other problems we'll need to approximate $Z(x)$ and Z.

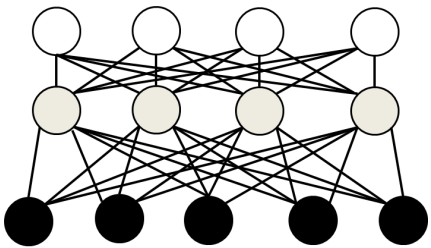
Greedy Layerwise Training of Stacked RBMs

- Step 1: Train an RBM (alternating between block Gibbs and stochastic gradient)



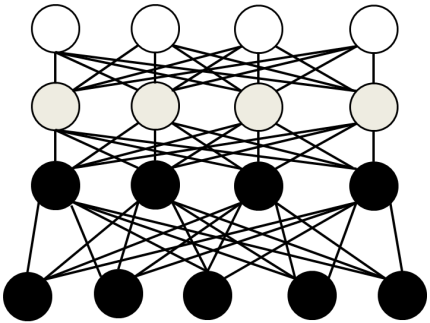
Greedy Layerwise Training of Stacked RBMs

- Step 1: Train an RBM (alternating between block Gibbs and stochastic gradient)
- Step 2:
  - Fix first hidden layer values.
  - Train an RBM.



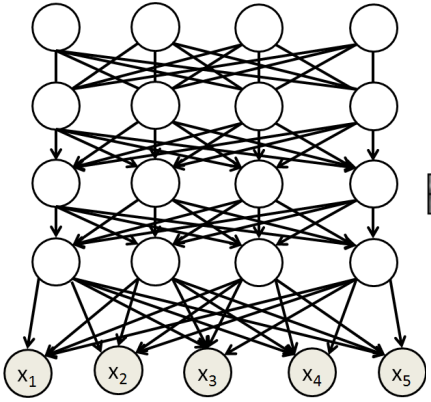
Greedy Layerwise Training of Stacked RBMs

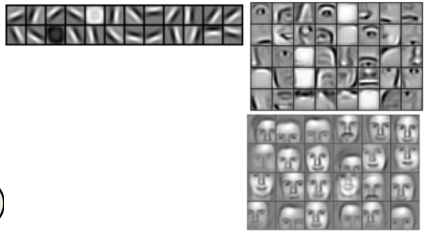
- Step 1: Train an RBM (alternating between block Gibbs and stochastic gradient)
- Step 2:
  - Fix first hidden layer values.
  - Train an RBM.
- Step 3:
  - Fix second hidden layer values.
  - Train an RBM.



Deep Belief Networks

- Keep top as an RBM.
- For the other layers, use DAG parameters that implement block sampling.
  - Can sample by running block Gibbs on top layer for a while, then ancestral sampling.

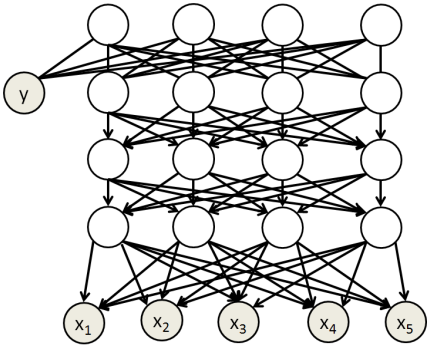




http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

Deep Belief Networks

- Can add a class label to last layer.



- Can use "fine-tuning" as a feedforward neural network to refine weights.
  - `https://www.youtube.com/watch?v=KuPai0ogiHk`

Summary

- Approximate UGM learning:
  1. Change objective function: pseudolikelihood and structured SVMs.
  2. Approximate marginals: Monte Carlo or variational methods.
- Structure learning in UGMs with [group] L1-regularization.
- Hidden Markov models model time-series with latent factors.
  - Tons of applications, typically more realistic than Markov models.
- Boltzmann machines are UGMs with binary hidden variables.
  - Restricted Boltzmann machines only allow connections between hidden/visible.
- Deep belief networks and Boltzmann machines have layers of hidden variables.
- Next time: we'll use these tools for supervised learning.

Log-Linear UGM Gradient

- We showed that NLL with log-linear parameterization is

$$f(w) = -w^T F(x) + \log Z(w).$$

- The gradient with respect to parameter w_j has a simple form

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