CPSC 540: Machine Learning

Deep Graphical Models,
Recurrent Neural Networks

Winter 2016

Admin

- Assignment 5:
  - Due on Tuesday (standard late day sequence applies).
  - For Q2, don't use *inv(C)* and set *errorDet=inf.*
- Project:
  - Due date moved again to April 26 (so that undergrads can graduate).
    - With some late "days" possible.
    - Submission instructions will be posted on Piazza next week.
  - Graduate students graduating in May must submit by April 21.
- Help session Monday, no more tutorials.
- Lecture may go long today.

Outline

1. Variational Inference
2. Unsupervised Deep Learning
3. Recurrent Neural Networks
4. What’s next?

Undirected Graphical Models

- Undirected graphical models (UGMs) for density estimation use:

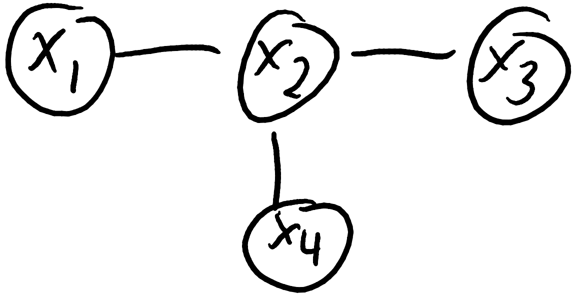
$$p(x) = \frac{\prod_{c \in C} \phi_c(x_c)}{Z}$$

Non-negative "potential" when variables 'c' have the values x_c.

Normalizing constant that makes sum to 1.

- The conditional independencies summarized by undirected graph:
  - Edge between node 'i' and 'j' if they appear together in at least one 'c'.

$$p(x) = \frac{\phi_{12}(x_1,x_2)\,\phi_{23}(x_2,x_3)\,\phi_{24}(x_2,x_4)\,\phi_{34}(x_3,x_4)}{Z} \Rightarrow$$



Conditional Random Fields

- Last time we considered conditional random fields (CRFs):

$$p(Y \mid X) = \frac{\prod_{c \in C} \phi_c(X, Y_c)}{Z(X)}$$

Non-negative "potential" for variables 'c' to have values Y_c given X.

Normalizing constant with this specific X.

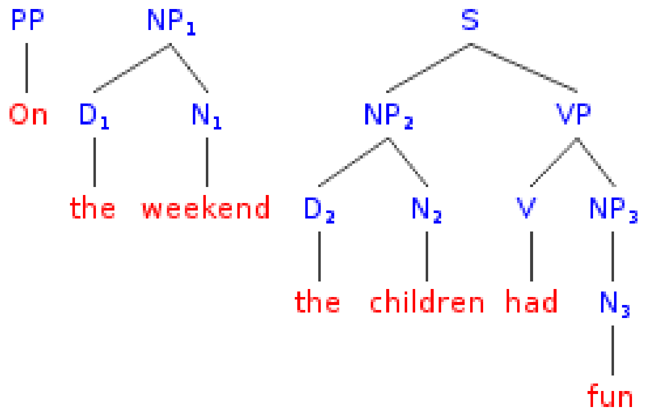
- CRFs model conditional probability as a UGM.
  - No need to model X.
  - Independence properties given by UGM on Y variables.
- Usually, we use a log-linear parameterization:

$$\phi_c(X, Y_c) = \exp(w_c f(X, Y_c)).$$

$$p(Y \mid X) \propto \exp(w^T F(X, Y))$$

CRFs for Part-of-Speech Tagging

- Part of speech tagging task: label sentence type for each word.



- Can get close to state of the art with CRFs.
  - Features for each word and adjacent words: $[F(X,Y)]_j = I[X_6 = \text{"had"}, Y_6 = \text{"V"}]$
    - These don't add edges to graph.

$[F(X,Y)]_{j'} = I[X_5 = \text{"children"}, Y_6 = \text{"V"}]$

  - Features on transitions between labels: $[F(X,Y)]_{j''} = I[Y_5 = \text{"V"}, Y_6 = \text{"V"}]$
  - Handles new test words ("OOV") by context.

Causes $Y_5 - Y_6$ edge

http://english.stackexchange.com/questions/93989/building-a-phrase-structure-of-on-the-weekend

Difficulty of Fitting CRFs

- CRF NLL requires involves normalizing constant Z(X):

$$p(Y \mid X, w) = \frac{\exp(w^T F(X,Y))}{Z(X)} \qquad -\log p(Y \mid X, w) = -w^T F(X,Y) + \log(Z(X))$$

  - Different than DAGs where Z=1.
- Gradient of NLL has special form and requires inference:

$$-\nabla_f \log p(Y \mid X, w) = -F_f(X,Y) + E[F_f(X,Y)]$$

- So optimizing NLL needs Z and marginals (“inference”).
- But exact inference is hard for general graphs.
  - Also hard for Bayesian statistics.

Monte Carlo vs. Variational Inference

- Two main strategies for approximate inference:
  1. Monte Carlo methods:
     - Approximate p(x) with empirical distribution over samples:

$$p(x) \approx \frac{1}{n}\sum_{i=1}^{n} I[x^i = x]$$

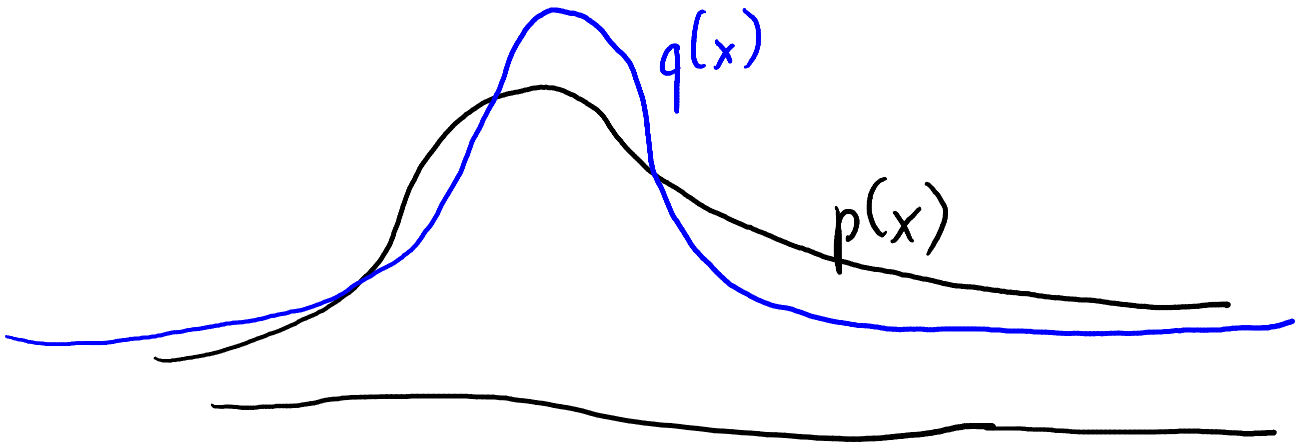
     - Turns inference into sampling.
  2. Variational methods:
     - Approximate p(x) with "closest" distribution 'q' from a tractable family:

$$p(x) \approx q(x)$$

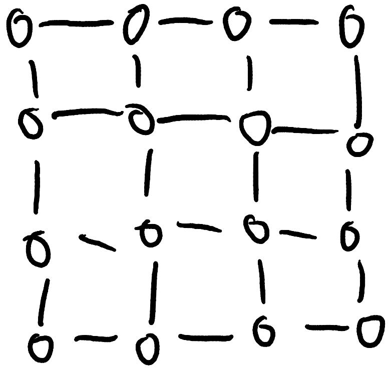
     - Could use Gaussian, independent Bernoulli, tree-structured graphical model:
       – Or mixtures of these simple distributions.
     - Turns inference into optimization.

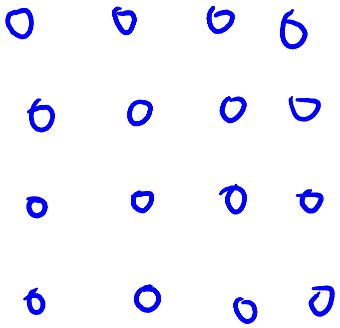
Variational Inference Illustration

- Approximate non-Gaussian ‘p’ by Gaussian ‘q’:

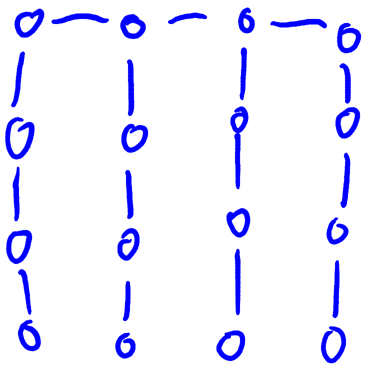


- Approximate non-tree UGM by independent distribution:





Better:
Tree-structured 'q':



Laplace Approximation

- Simple variational method is Laplace approximation:
  - Find 'x' that maximizes p(x):

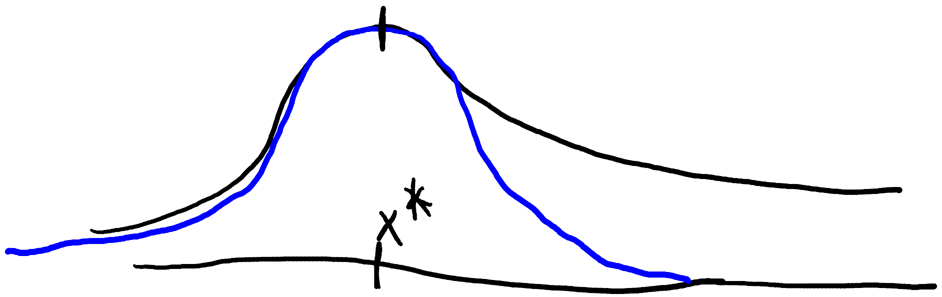
    $f(\cdot) = -\log p(x)$ $\qquad \min_x f(x)$

  - Choose 'q' so that –log q(x) and –log p(x) have same Taylor expansion at x^*:

We want $-\log q(x) = f(x^*) + \underbrace{\nabla f(x^*)}_{=0}{}^T(x - x^*) + \frac{1}{2}(x - x^*)^T \nabla^2 f(x^*)(x - x^*)$

$$= f(x^*) + \frac{1}{2}(x - x^*)^T \nabla^2 f(x)(x - x^*)$$

So $q \sim N(x^*, \nabla^2 f(x)^{-1})$



Minimizing Reverse KL Divergence

- Most common variational method:
  - Minimize (reverse) KL divergence between q and p:

$$KL(q \,||\, p) = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

  - KL divergence is common measure of similarity between distributions.
  - Only needs unnormalized distribution and gives lower bound on log(Z):

$$KL(q \,||\, p) = \sum_x q(x) \log q(x) - \sum_x q(x) \log(\tilde{p}(x)) + \sum_x q(x) \log(Z)$$

$$= \sum_x q(x) \log \frac{q(x)}{\hat{p}(x)} + \log(Z)$$

Mean Field and Variational Bayes

- As an example, consider minimize KL with independent ‘q’:

$$q(x) = \prod_{j=1}^{d} q_j(x_j)$$

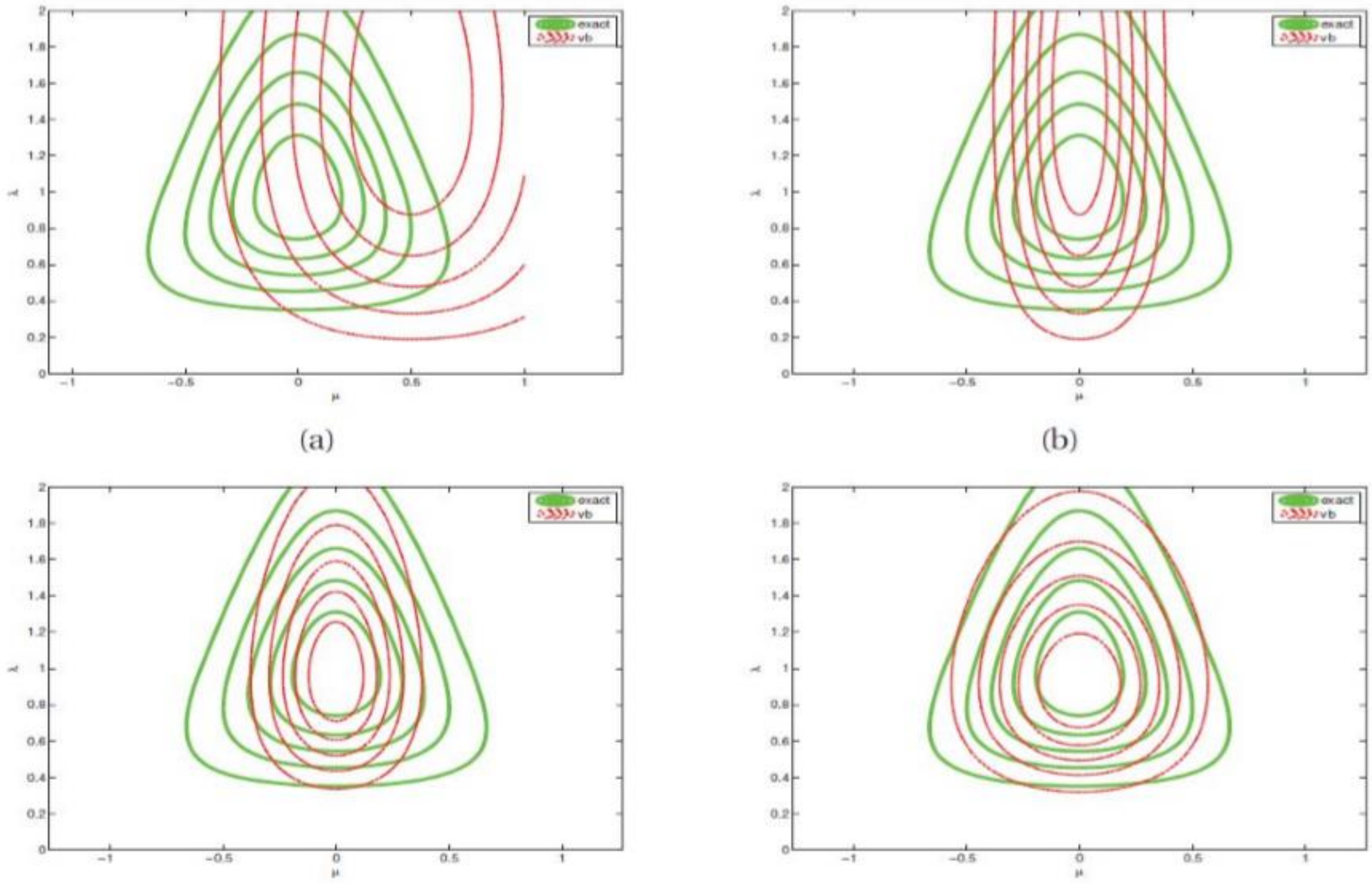
- Optimization of functional ‘q_j’ yields:

$$\log q_j(x_j) = E_{-q_j}[\log \tilde{p}(x)] + \text{const.}$$

Update only depends on Markov blanket

- Applying this update is called:
  - Mean field method (graphical models).
  - Variational Bayes (Bayesian inference).

Variational Bayes in Action



Loopy Belief Propagation

- Other main variational method is loopy belief propagation:
  - Does not require ‘q’ to be a probability, just requires “local consistency”:
    - Expectations of neighbouring nodes agree.
  - Locally minimizes KL, typically gives better marginal approximations.
  - Only has closed-form for Gaussian/discrete UGMs:
    - Can approximating non-Gaussian/discrete using “expectation propagation”.
  - Not convex and does not give bound on Z.
    - TRBP variant is convex and gives upper bound on Z.

Variational Methods Discussion

- Monte Carlo vs. variational methods:
  - Variational methods are typically more complicated.
  - Variational methods are not consistent:
    - ‘q’ does not converge to ‘p’.
  - But variational typically gives better approximation for same time.
    - Although MCMC is easier to parallelize.
  - Variational methods typically have similar cost to MAP.

- Related approach is convex relaxations:
  - Approximate non-convex decoding by convex optimization.

- Combinations of variational inference and stochastic methods:
  - Stochastic variational inference: use stochastic gradient to speed up variational methods.
  - Variational MCMC: use Metropolis-Hastings where variational ‘q’ sometimes makes proposals.

Outline

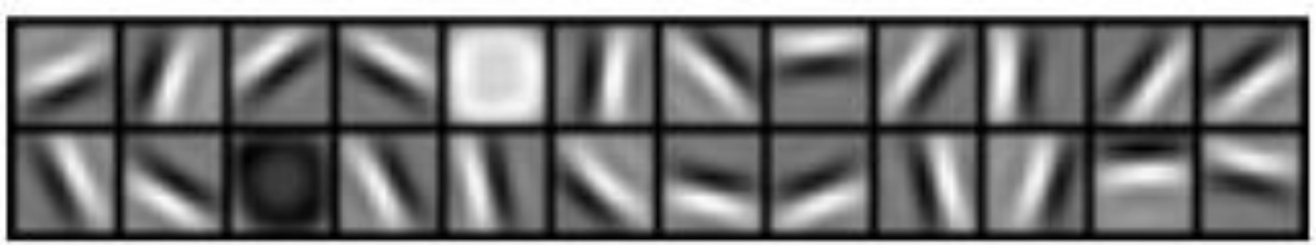
1. Variational Inference
2. Unsupervised Deep Learning
3. Recurrent Neural Networks
4. What’s next?

Deep Density Estimation

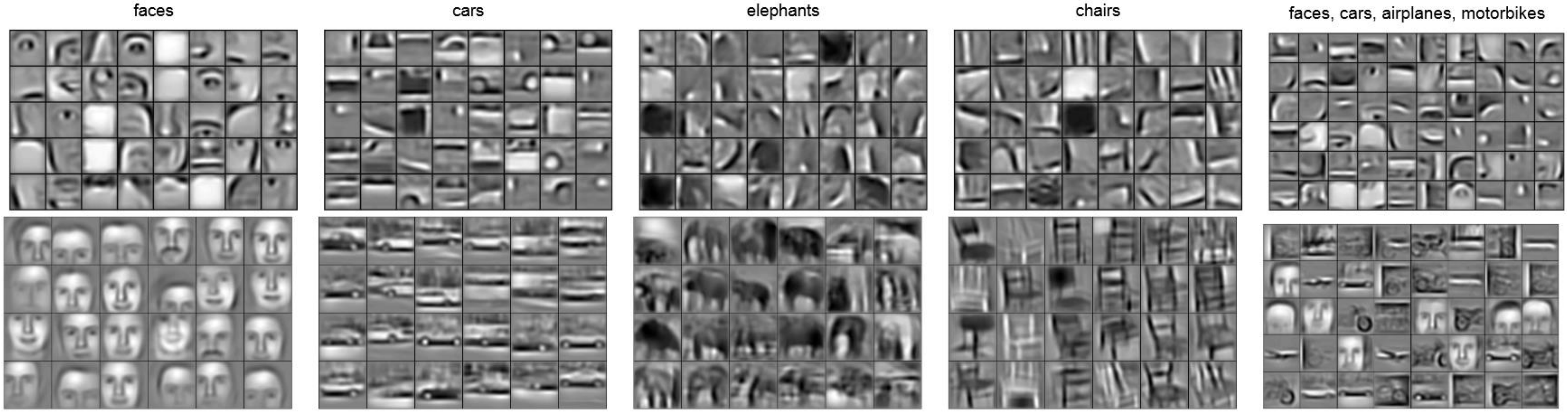
- We've previously discussed supervised deep learning.
  - And autoencoders as a form of unsupervised learning.
- Does it make sense to talk about deep density estimation?
- Standard argument:
  - Human learning seems to be mostly unsupervised.
  - Could we learn unsupervised models with much less data?
- Deep belief networks started deep learning movement (2006).
  - First non-convolutional deep network that people got working.

Cool Picture Motivation 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

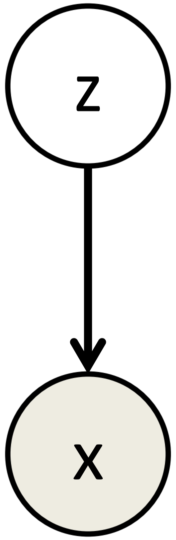
Mixture of Independent Models

- Recall the basic mixture model:

$$p(x) = \sum_{c=1}^{k} p(z=c)\, p(x \mid z=c)$$

categorical simple.

- Interpretation of joint p(x,z) as a graphical model:
  - Data ‘x’ comes from some “nice” distribution given cluster ‘z’.

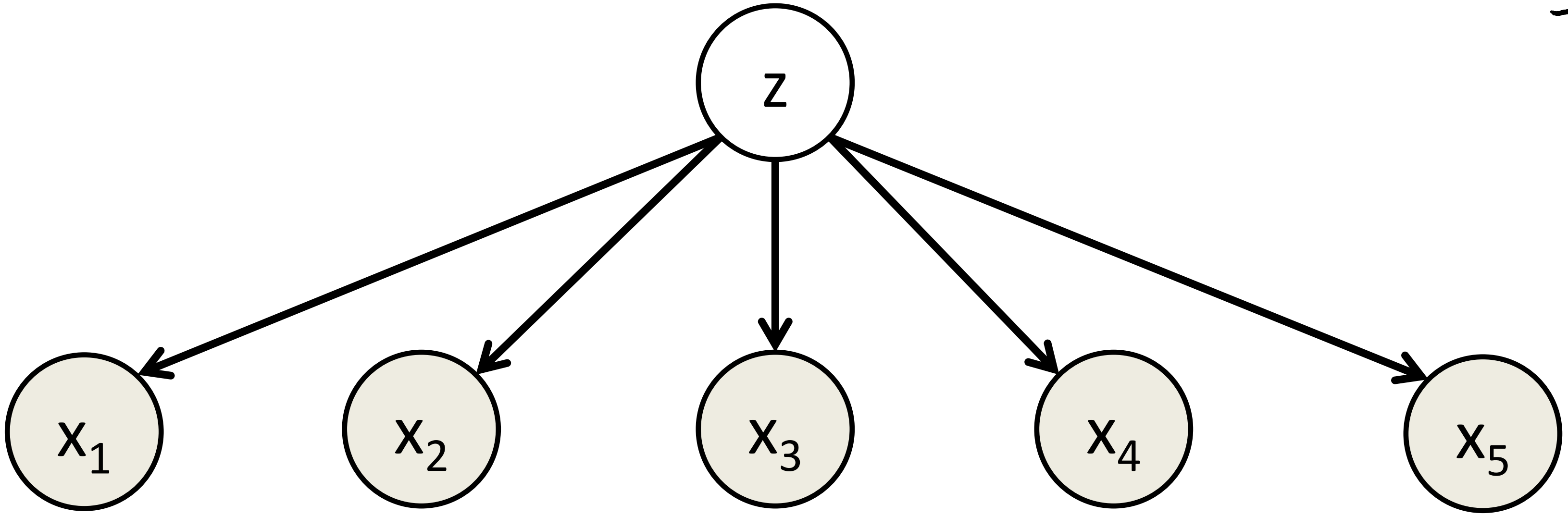


Mixture of Independent Models

- Recall the mixture of independent models:

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

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



This is enough to model <u>any</u> distribution.

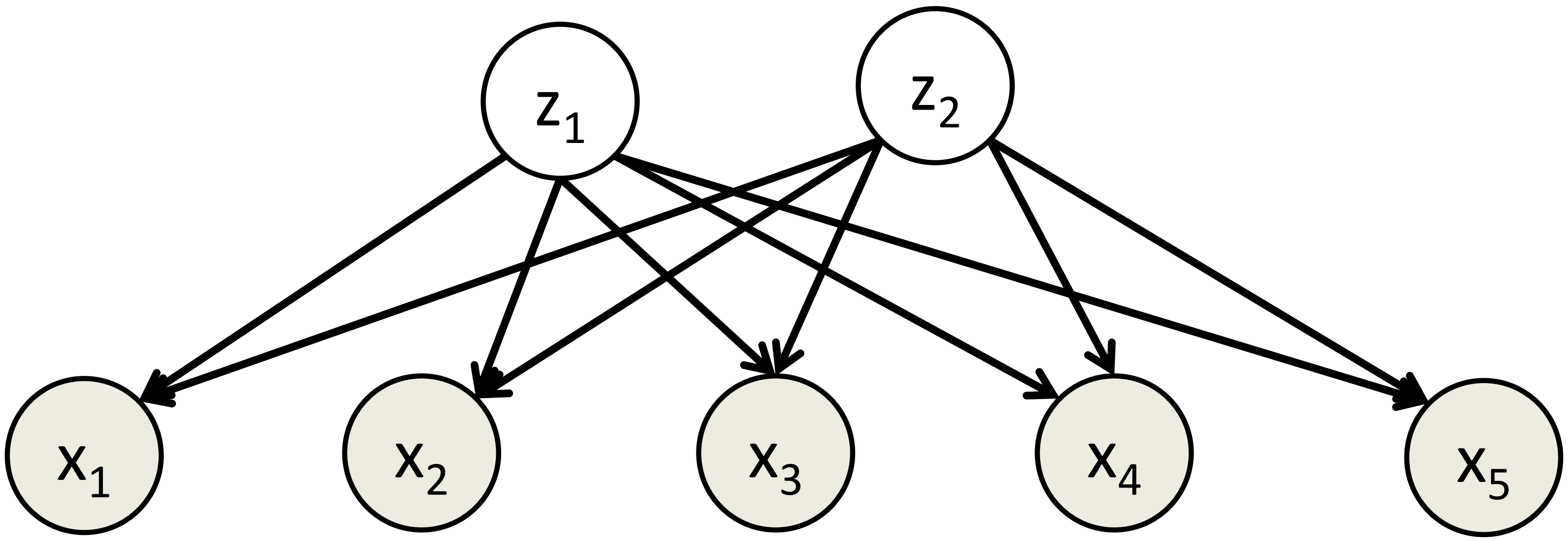
— Just need to know cluster of x^i and distribution of x_j given z^i

— But <u>not efficient</u> representation

— <u>Number of clusters</u> might be <u>huge</u>!

Latent DAG Model

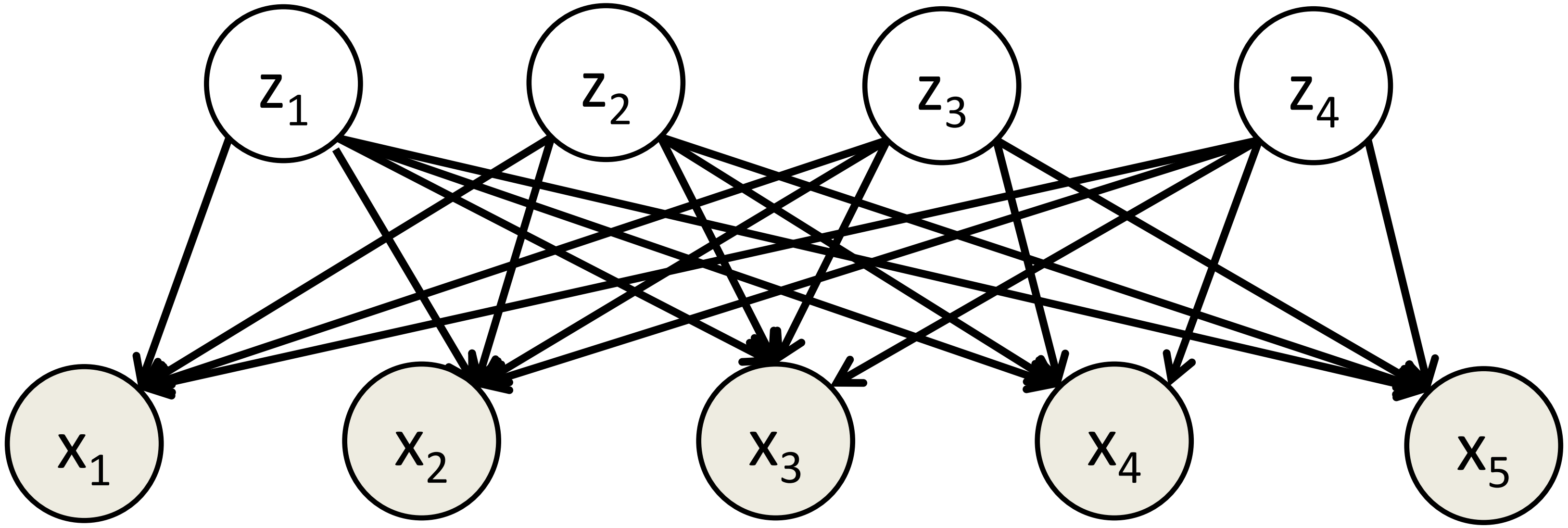
- Consider the following model with binary z_1 and z_2:



- 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 nodes.
  - We have combinatorial number of mixtures.
  - Let's 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'.
    - We '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:

Given top layer:
- Sampling is easy.
- Inference is easy.

But, given bottom layer:
- Everything is hard.
- Combinatorial "explaining away".

Common training method:
- Greedy layerwise as RBM.

Abstract concept

↓

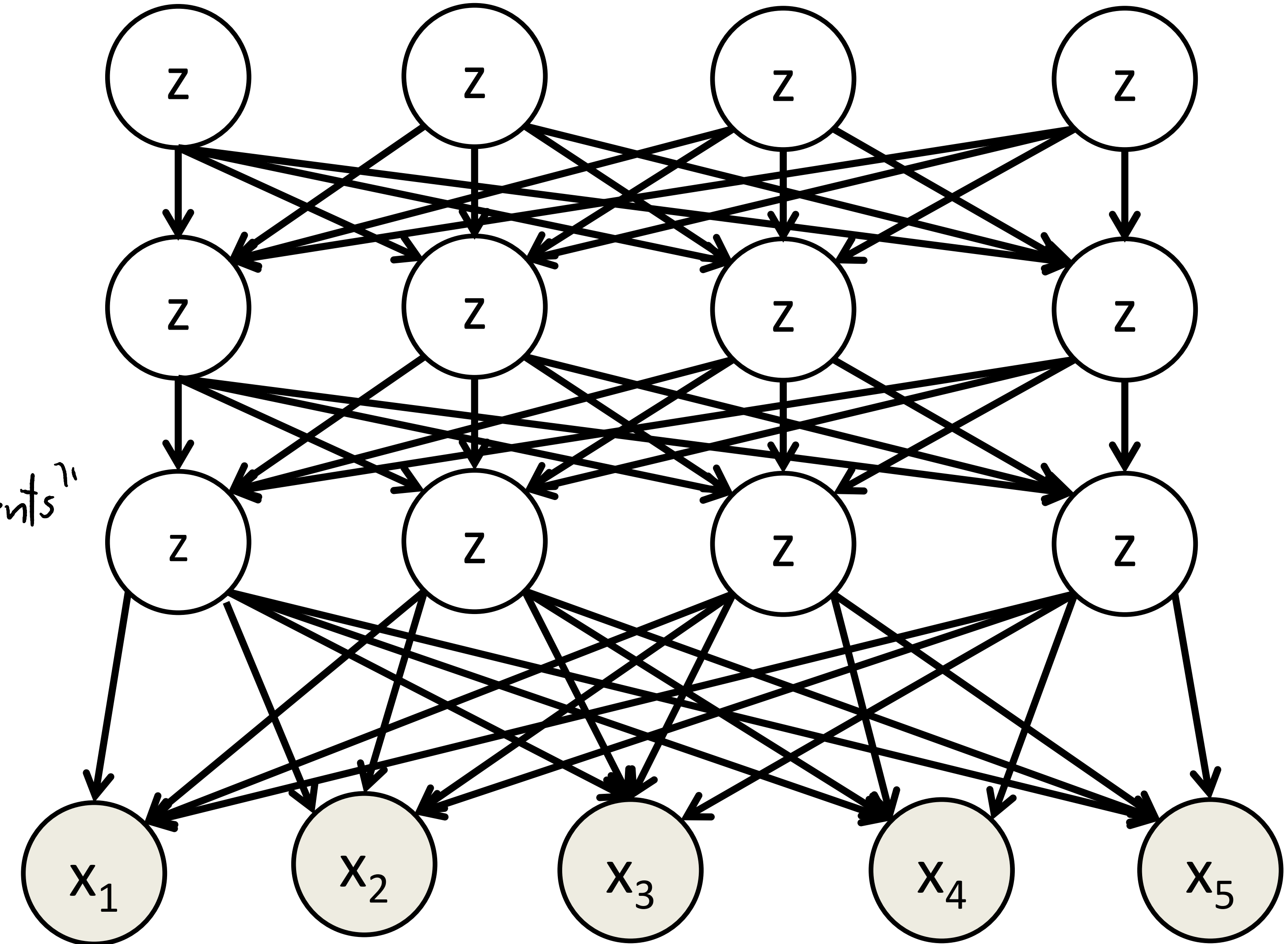
"Parts"

↓

"Basic ingredients"

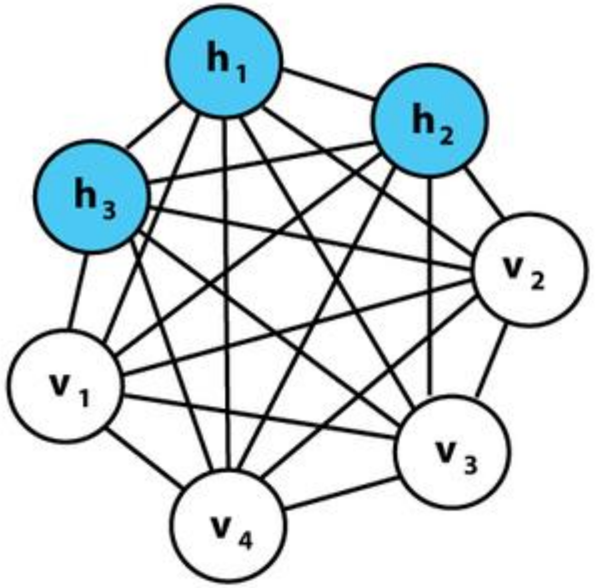
↓

Data



Boltzmann Machine

- Boltzmann machines are UGMs with binary latent variables:

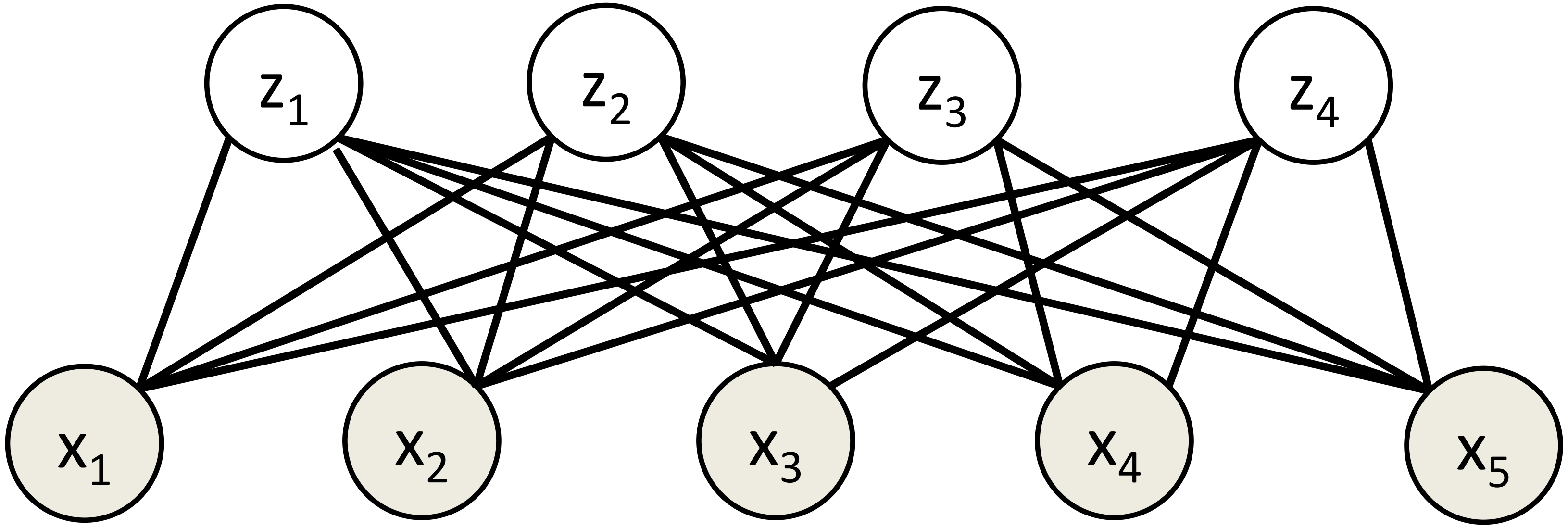


- Yet another latent-variable model for density estimation.
  - Hidden variables again give a combinatorial latent representation.
- Hard to do anything in this model, even if you know all the 'h'.

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

Restricted Boltzmann Machine

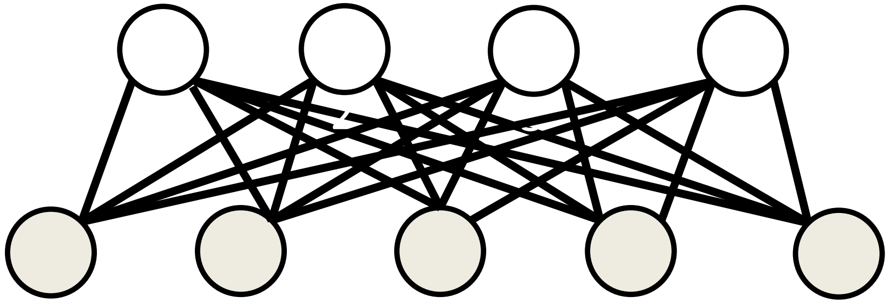
- By restricting graph structure, some things are easier:
  - Restricted Boltzmann machines (RBMs): edges only between the x_j and z_c.



  - Given visible x, decoding/inference/sampling of z is easy:
    - Block Gibbs sampling is just sampling each z_j independently.
  - Given hidden h, decoding/inference/sampling of x is easy (independent).
    - Block Gibbs sampling is just sampling each x_j independently.

Restricted Boltzmann Machine

- Restricted Boltzmann machines (RBMs):



$$p(x) = \sum_h p(x,h) = \sum_h \frac{\tilde{p}(x,h)}{\sum_{h,x} \tilde{p}(x,h)} = \frac{\sum_h \tilde{p}(x,h)}{\sum_{h,x} \tilde{p}(x,h)} = \frac{Z(x)}{Z}$$

$$-\log p(x) = \underbrace{-\log(Z(x))}_{\text{Now easy to compute but still non-convex}} + \underbrace{\log(Z)}_{}.$$

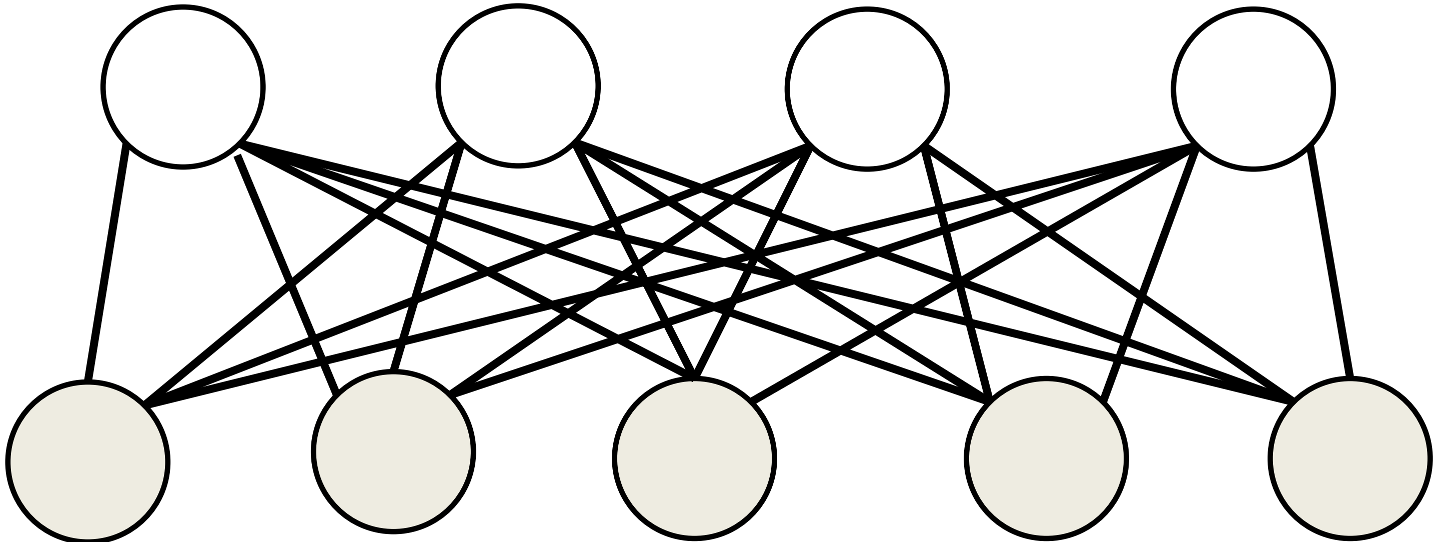
Convex but still <u>hard</u> to compute

Standard training:

- "Persistent contrastive divergence".
- One step of <u>block</u> Gibbs sampling to approximate $\log(Z)$, then one gradient step.

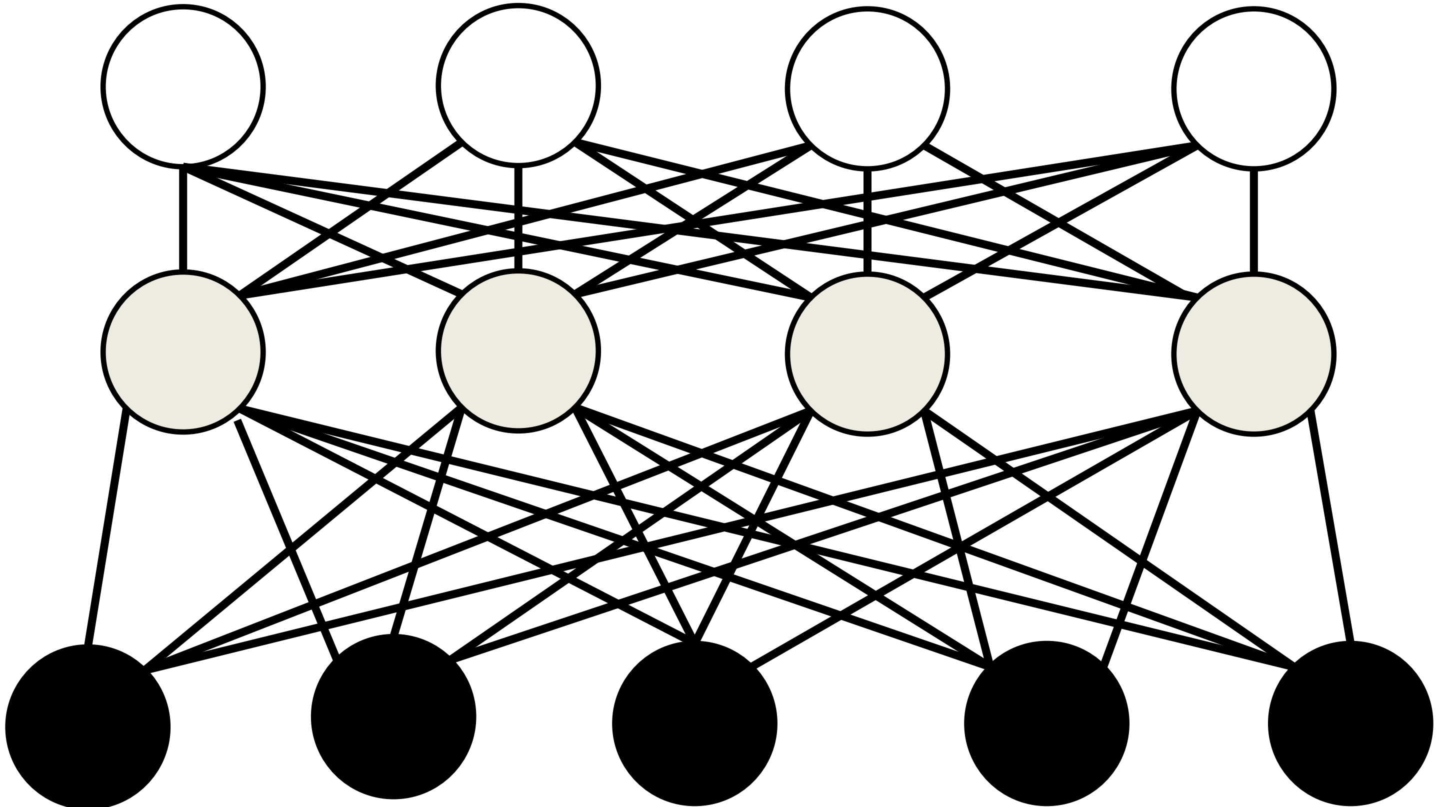
Greedy Layerwise Training of Stacked RBMs

- Step 1: train an RBM.



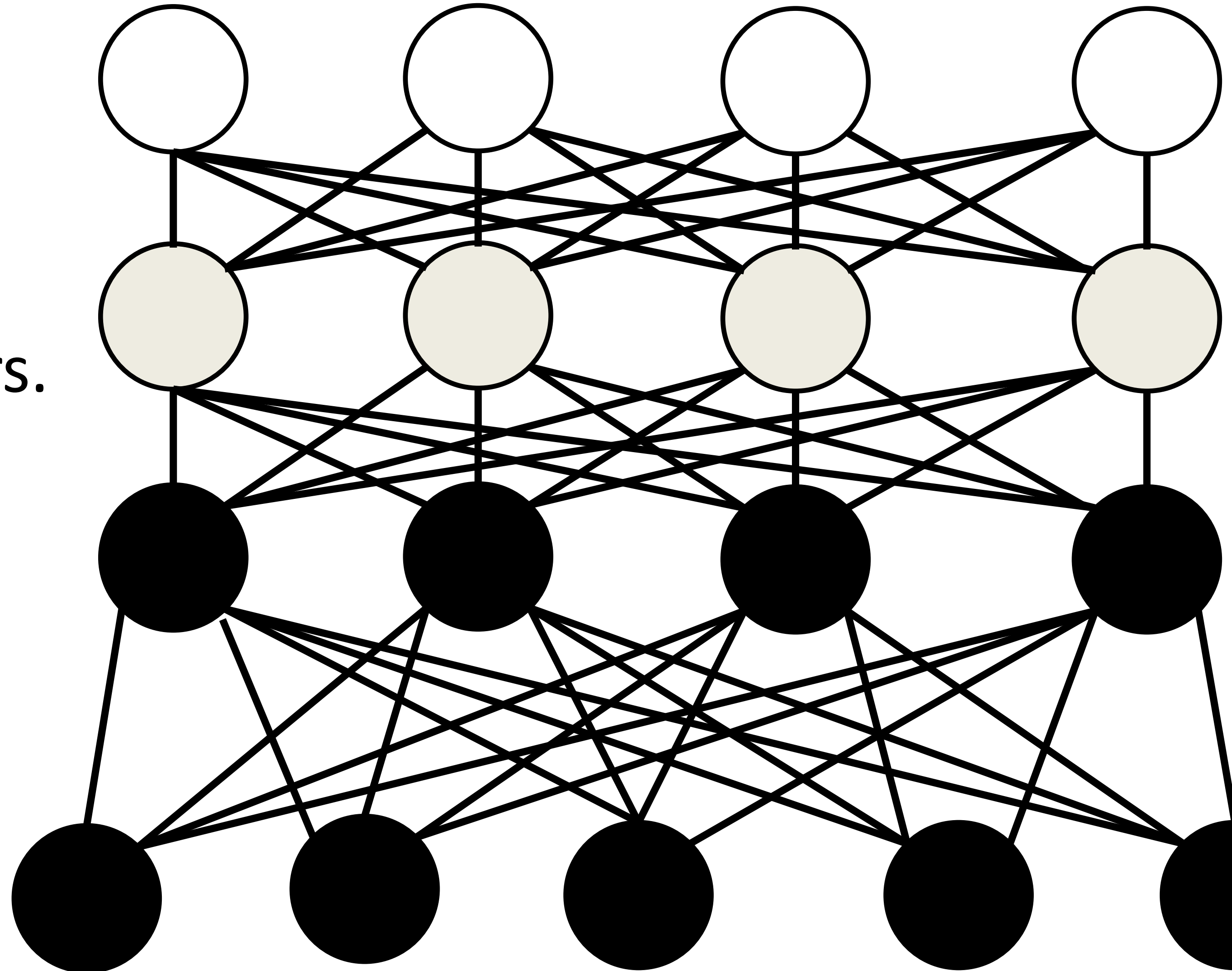
Greedy Layerwise Training of Stacked RBMs

- Step 1: train an RBM.
- Step 2:
  - Fix first hidden layer values.
  - Train an RBM.



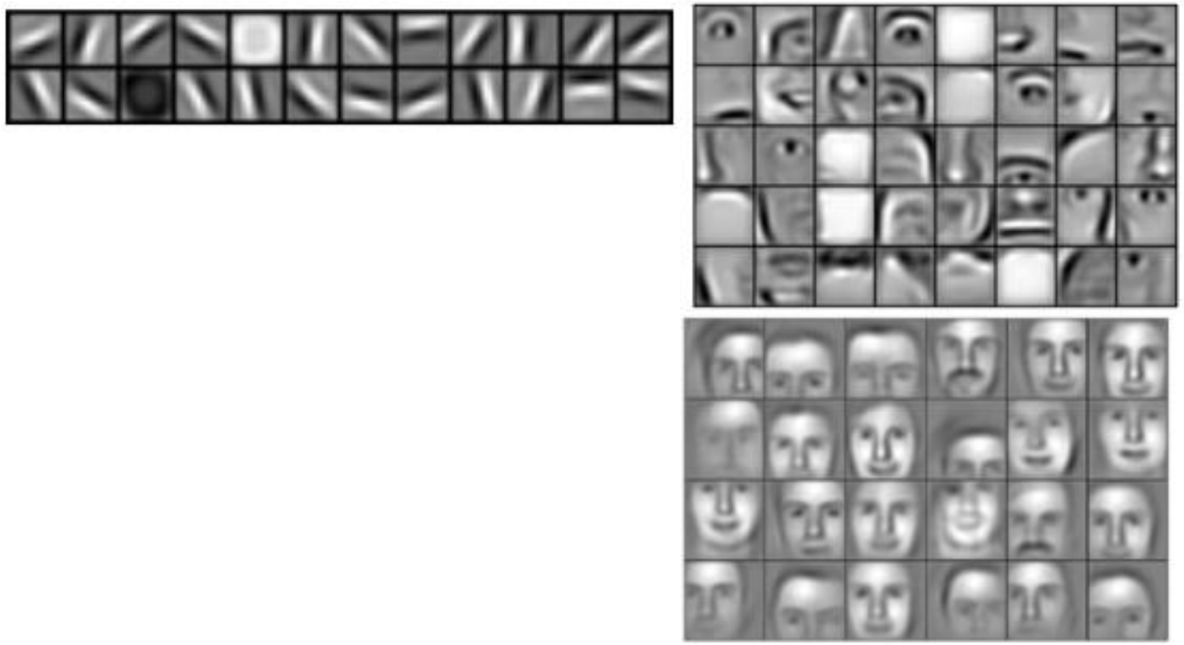
Greedy Layerwise Training of Stacked RBMs

- Step 1: train an RBM.
- Step 2:
  - Fix first hidden layer.
  - Train an RBM.
- Continue to add more layers.

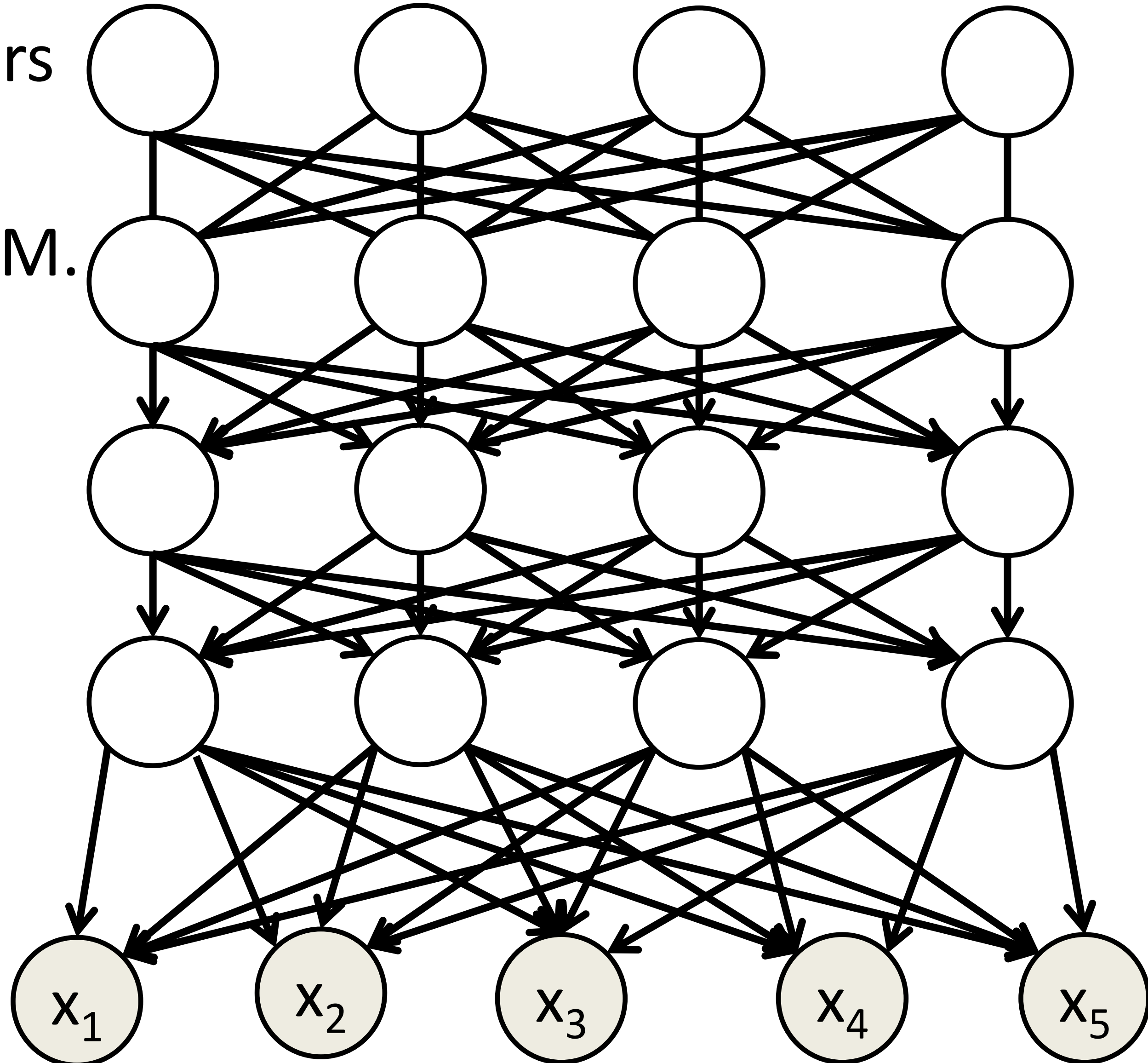


Deep Belief Networks

- Now treat stacked RBM parameters as parameters of deep belief net.
- Usually the last layer is kept as RBM.



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

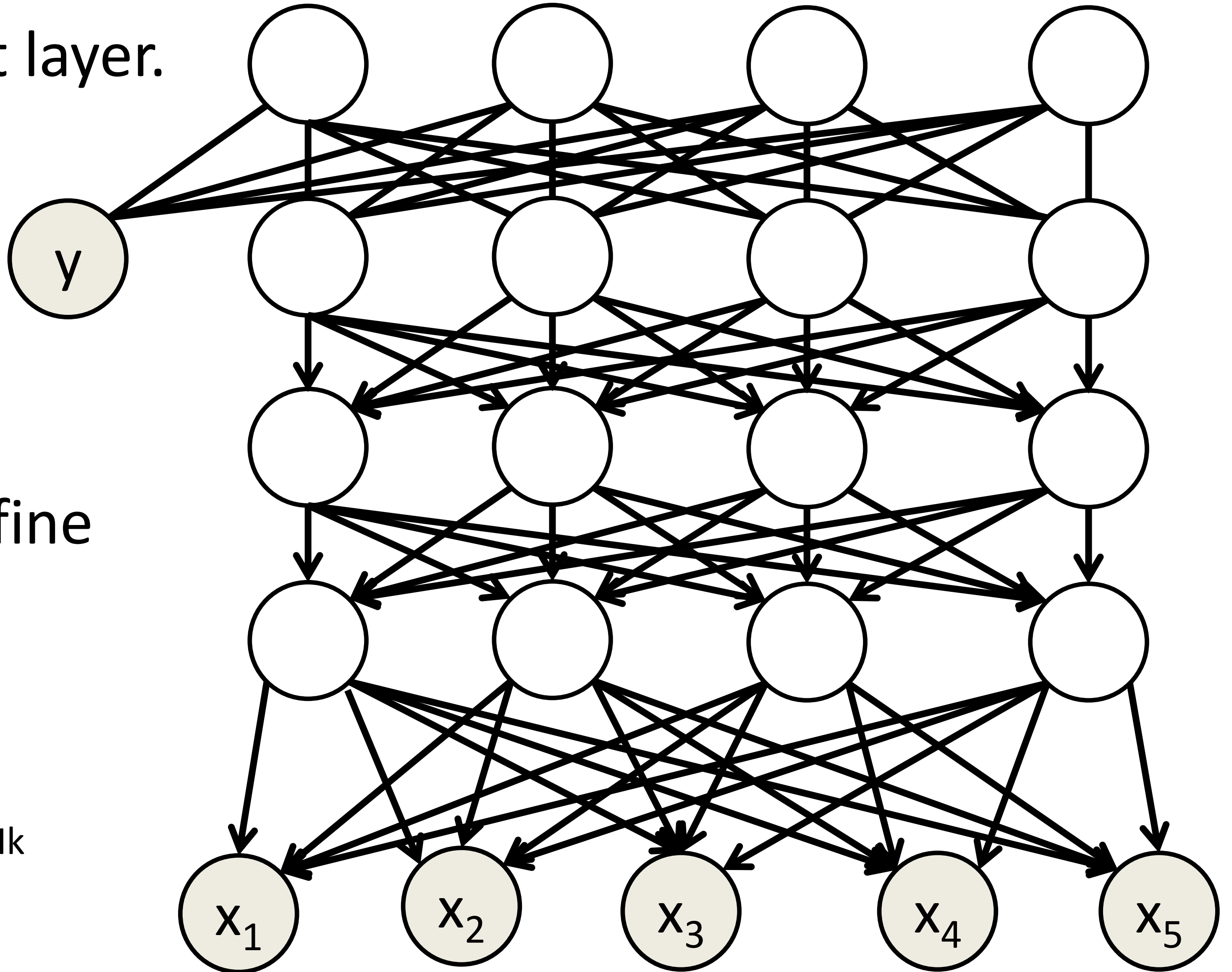


Deep Belief Networks

- Can add a class label to last layer.

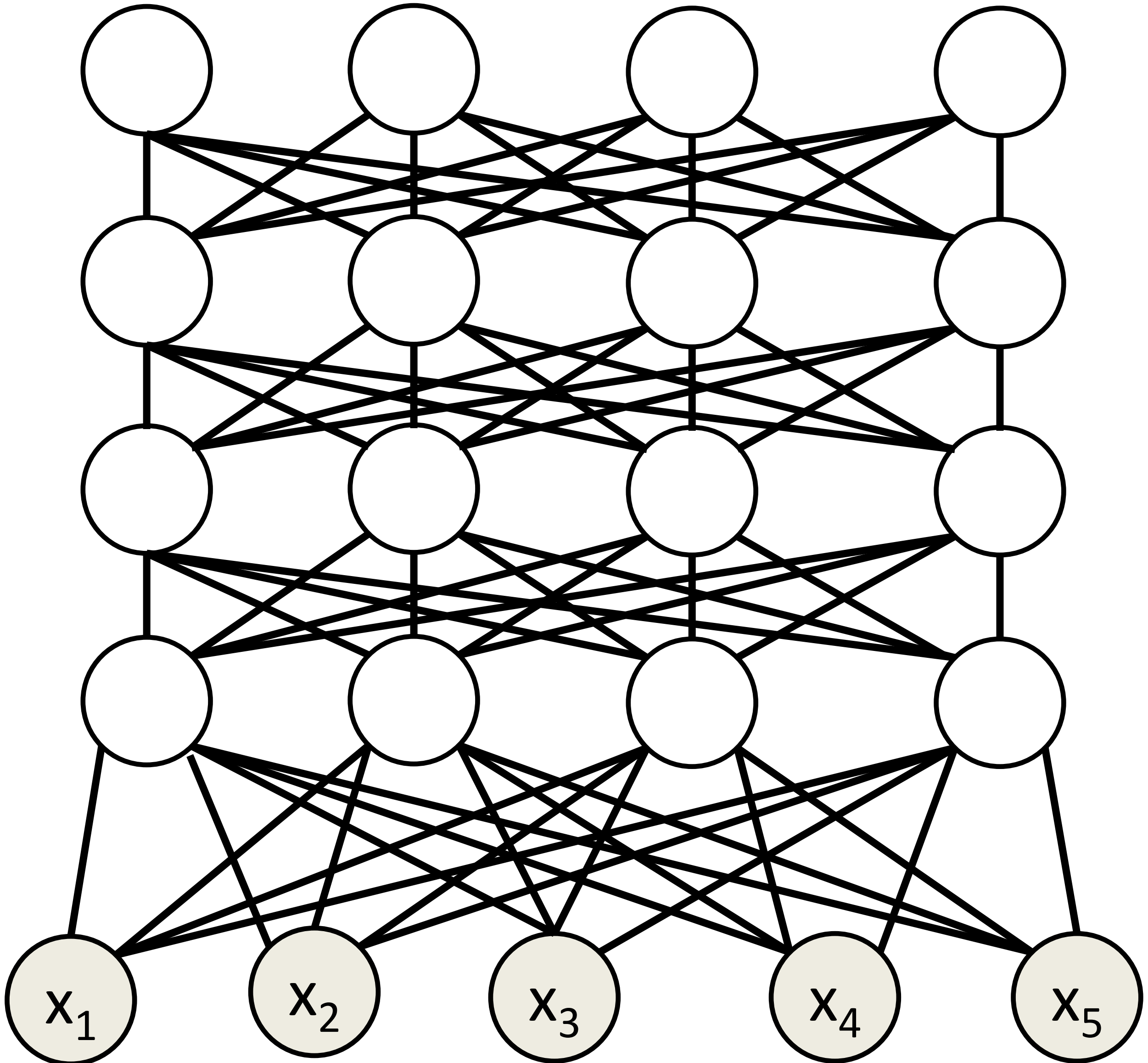
- Can use "fine-tuning" as feedforward network to refine weights.

https://www.youtube.com/watch?v=KuPai0ogiHk



Deep Boltzmann Machines

- Deep Boltzmann machines:
  - Just keep as undirected model.
  - Sampling is a nicer:
    - No explaining away within layer.
    - Variables in layer are independent given variables in layers above and below.
- More recent generative models:
  - Variational autoencoder.
    - Variational ‘q’ parameters are output of neural network.
  - Generative adversarial networks.
    - Adds discriminative model that tries to tell if samples come from model.
  - Bayesian dark knowledge.
    - Represent posterior by neural net.



Deep Boltzmann Machines

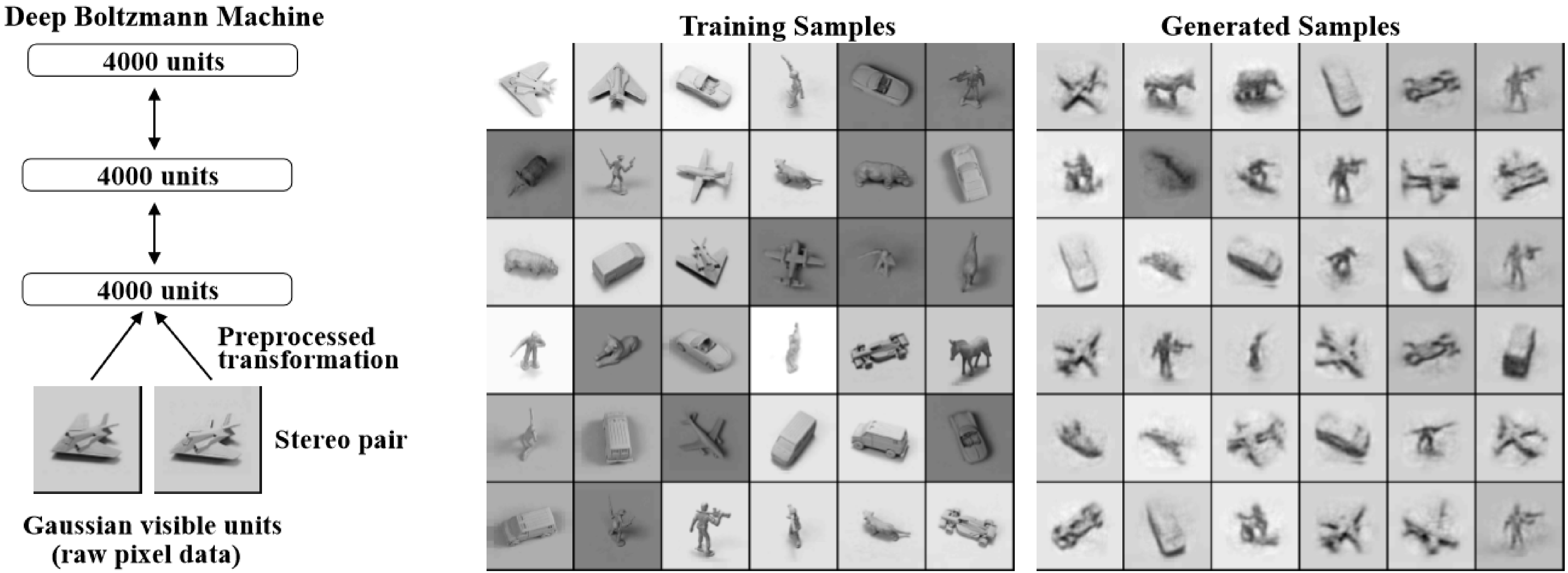


Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

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

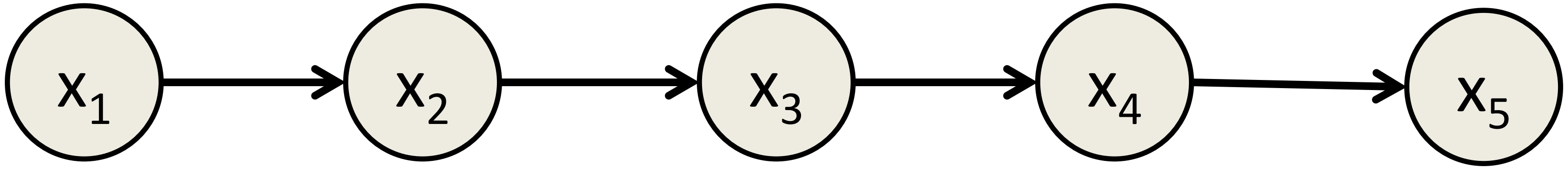
Outline

1. Variational Inference
2. Unsupervised Deep Learning
3. Recurrent Neural Networks
4. What's next?

This section takes a lot from these sources:
http://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf
https://ift6266h15.files.wordpress.com/2015/04/21_rnn.pdf

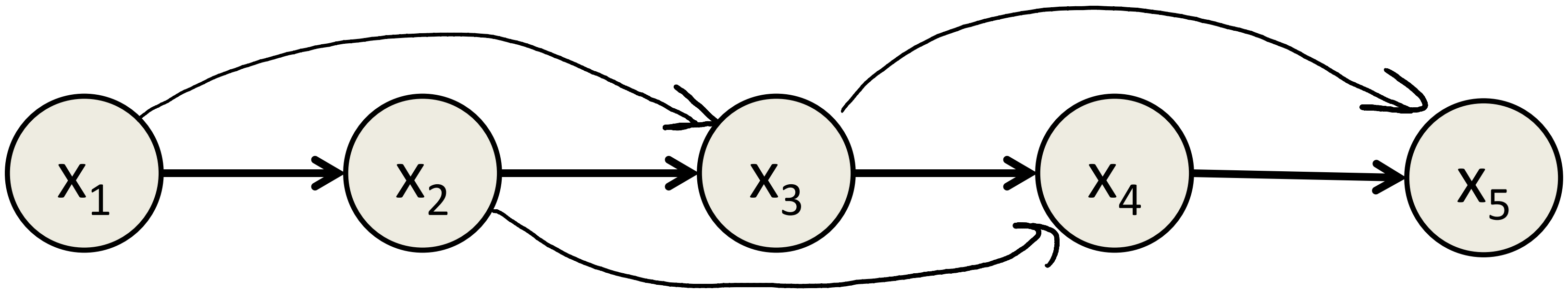
Motivation: Sequence Modeling

- We want to predict the next words in a sequence:
  - "I am studying to become a [?????????????????????????????]".
- Simple idea: supervised learning to predict the next word.
  - Applying it repeatedly to generate the sequence.
- Simple approaches:
  - Markov chain:



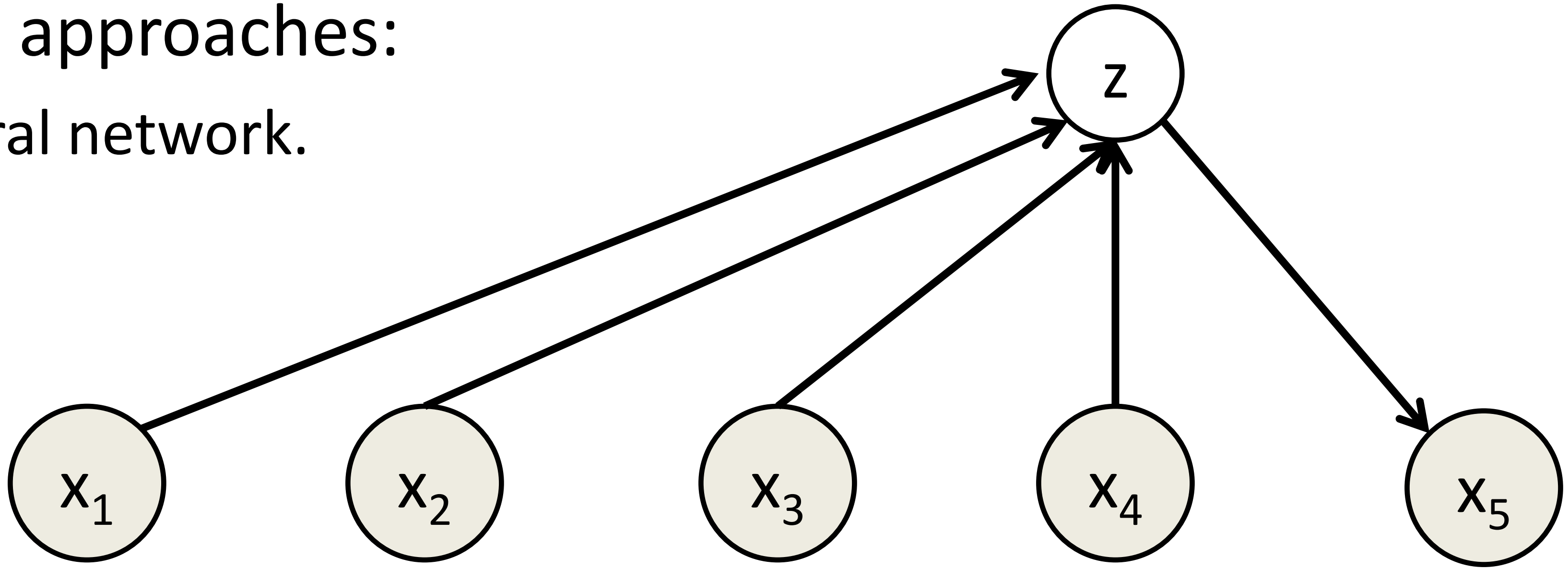
Motivation: Sequence Modeling

- We want to predict the next words in a sequence:
  - "I am studying to become a [???????????????????????????]".
- Simple idea: supervised learning to predict the next word.
  - Applying it repeatedly to generate the sequence.
- Simple approaches:
  - Higher-order Markov chain:



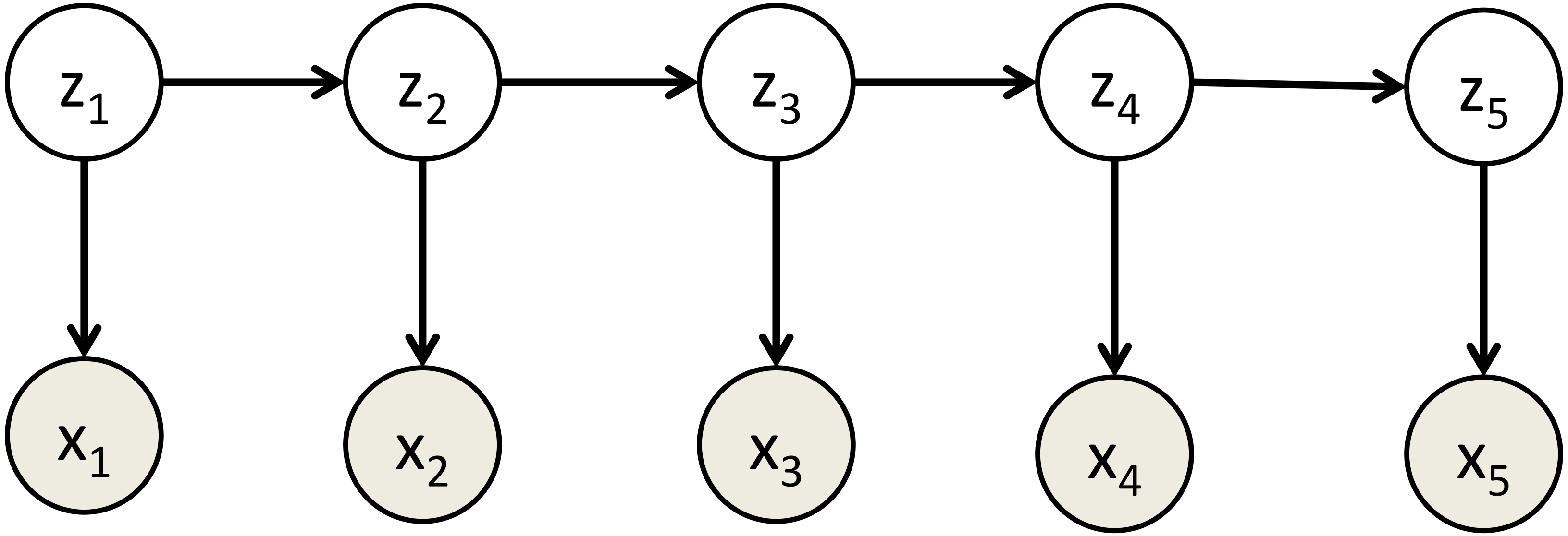
Motivation: Sequence Modeling

- We want to predict the next words in a sequence:
  - "I am studying to become a [???????????????????????????]".
- Simple idea: supervised learning to predict the next word.
  - Applying it repeatedly to generate the sequence.
- Simple approaches:
  - Neural network.



State-Space Models

- Problem with simple approaches:
  - All information about previous decision must be summarized by x_t.
  - We 'forget' why we predicted x_t when we got to predict x_{t+1}.
- More complex dynamics possible with state-space models:
  - Add hidden states with their own dynamics.

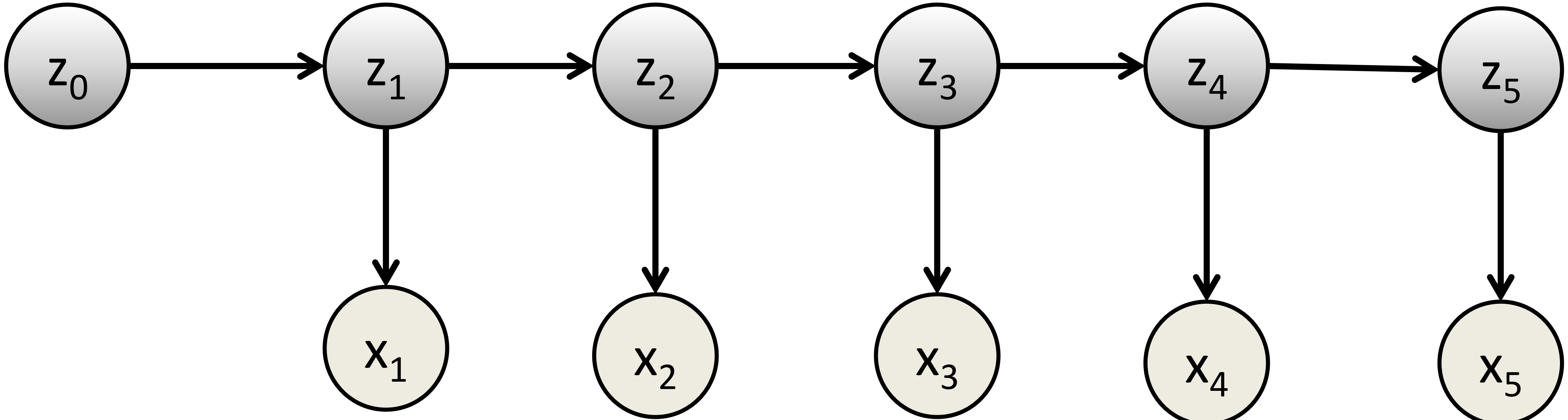


Challenges of State-Space Models

- Problem 1: inference only has closed-form when.
  - Markov blanket of each node must be conjugate to node.
  - Only 2 cases: Gaussian z *and* x (Kalman filter) or Discrete z (HMMs).
  - Otherwise, need to use approximate inference:
    - Most common is sequential Monte Carlo (also known as particle filters).
- Problem 2: memory is very limited.
  - You have to choose a z_t at time 't'.
    - More complicated dynamics but still need to compress information into a state.
- Want (deep) hidden representation with combinatorial structure.
  - Obvious solution: have multiple hidden z_t at time 't', as we did before.
    - But now inference becomes hard.

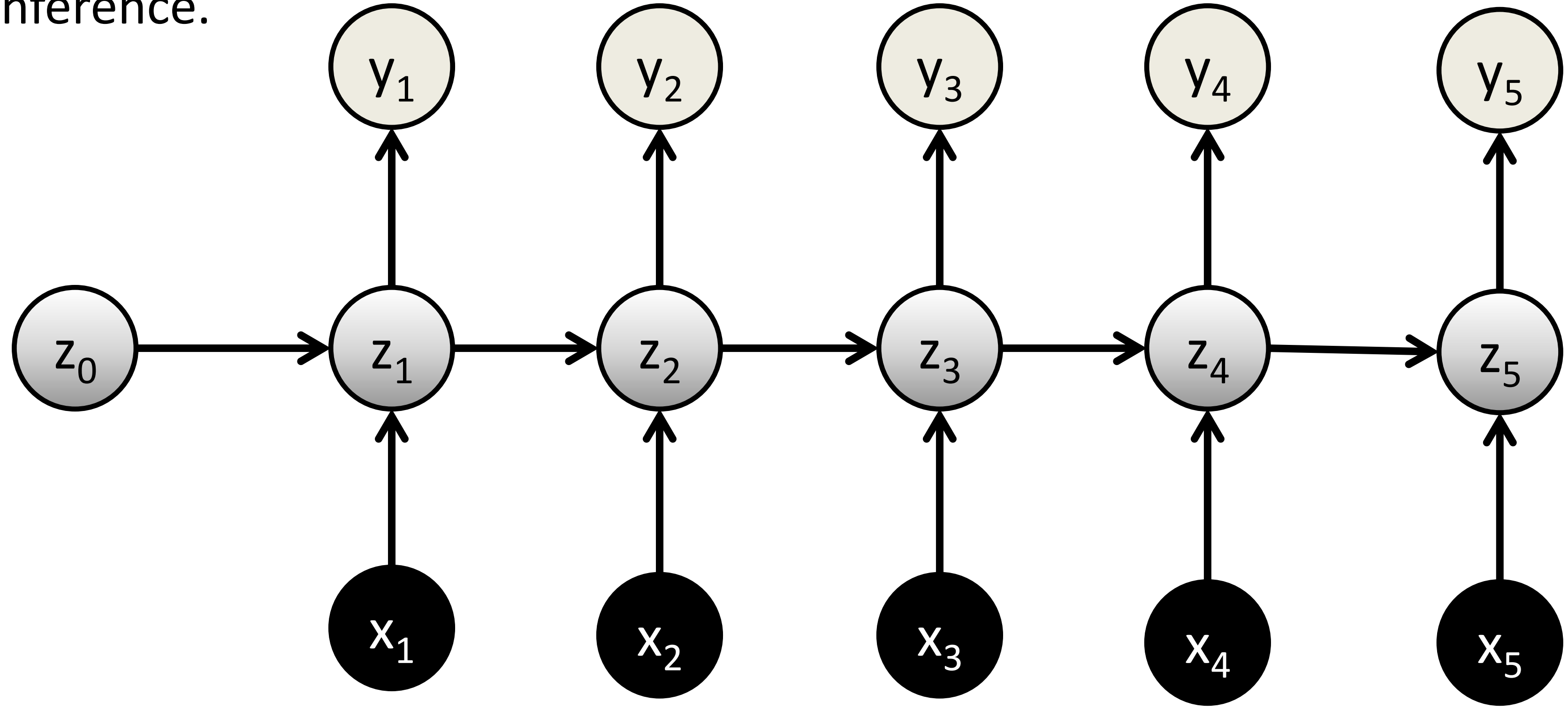
Recurrent Neural Networks

- Obvious solution (same as for mixtures):
  - Have multiple hidden z_t at time 't', as we did before.
    - But now inference becomes hard.
- Recurrent neural networks (RNNs) give solution to inference:
  - At time 't', hidden units are deterministic transformations of time 't-1'.
  - Basically turns the problem into a big and structured neural network.



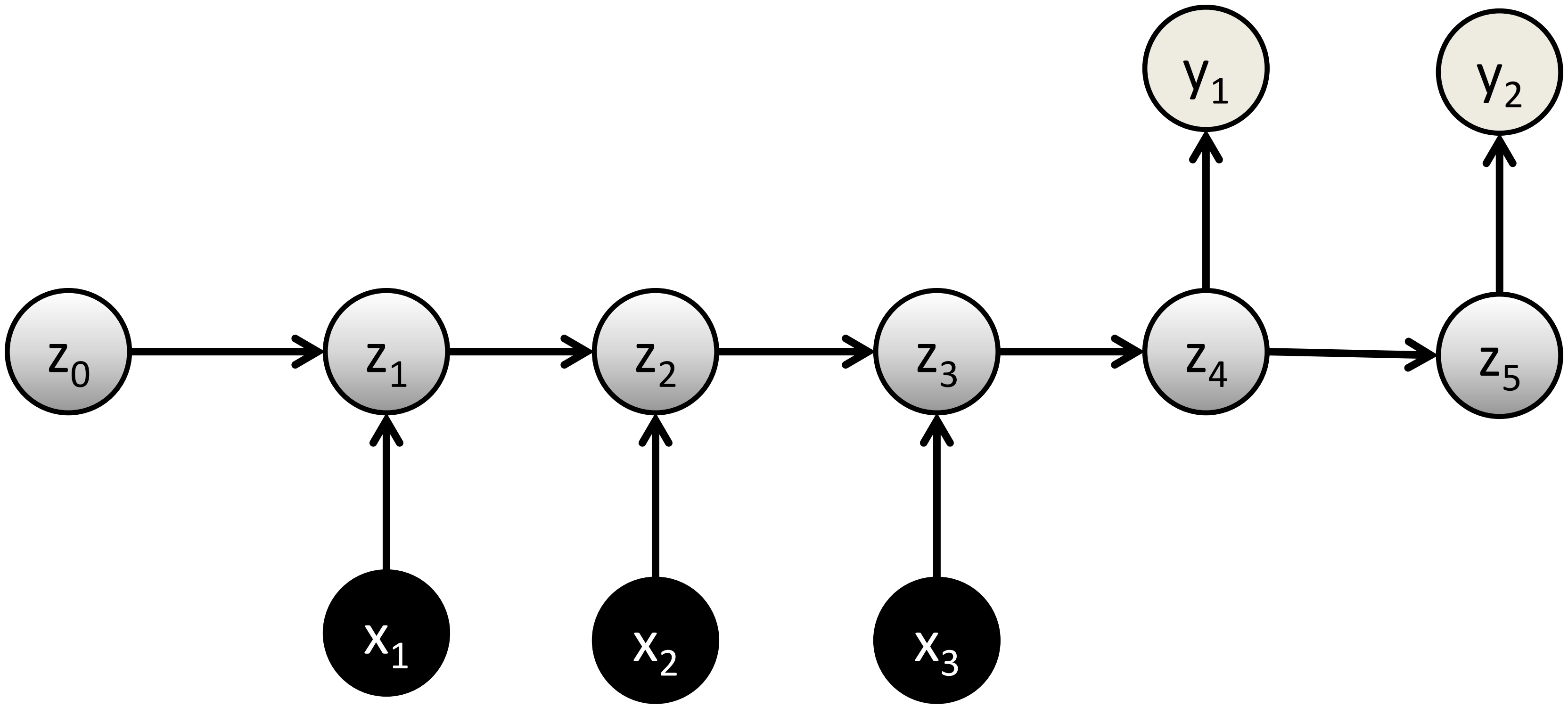
Recurrent Neural Networks

- RNNs can be used to translate input sequence to output sequence:
  - Similar to latent-dynamics model from last time (a bit less powerful).
  - But deterministic transforms means hidden ‘z’ can be really complicated.
    - But with easy inference.



Recurrent Neural Networks for Sequence

- An interesting variation on this for sequences of different lengths:
  - Translate from French sentence ‘x’ to English sentence ‘y’.
  - Turn video frames into a sentence.



Discussion of Recurrent Neural Networks

- Train using stochastic gradient: gradient by backpropagation.
- Similar challenges/heuristics to training deep neural networks:
    - "Exploding/vanishing gradient", initialization is important, slow progress, etc.
- Interesting variations:
    - Skip connections: connections from older 'z_t' to current hidden state.
    - Bi-directional RNNs: feedforward from past and future.
    - Recursive neural networks: consider sequences through non-chain data.

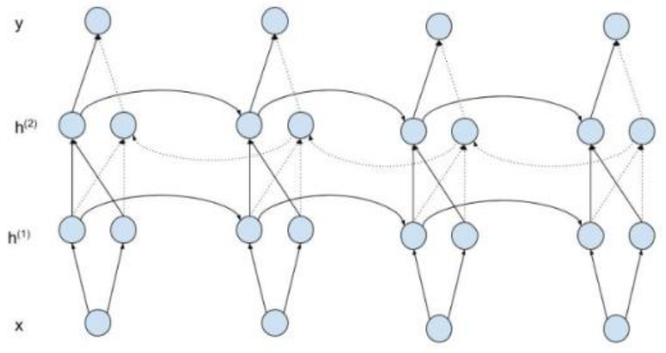
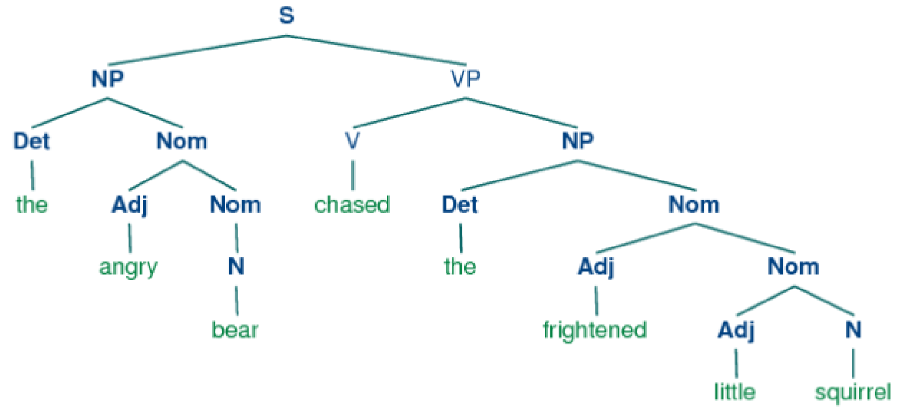


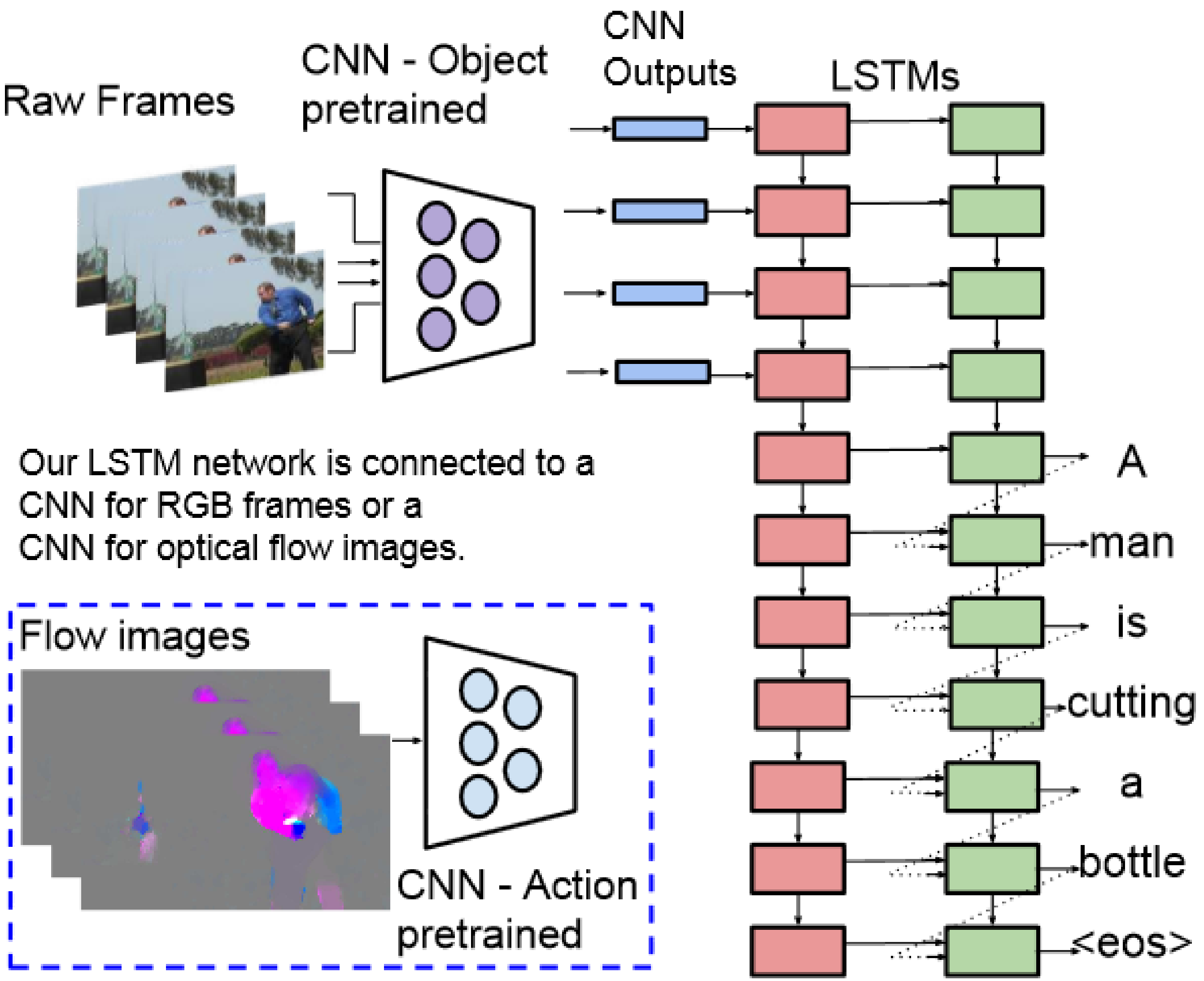
Figure 2: A deep bi-directional RNN with 2 stakced layers



Long Short Term Memory (LSTM)

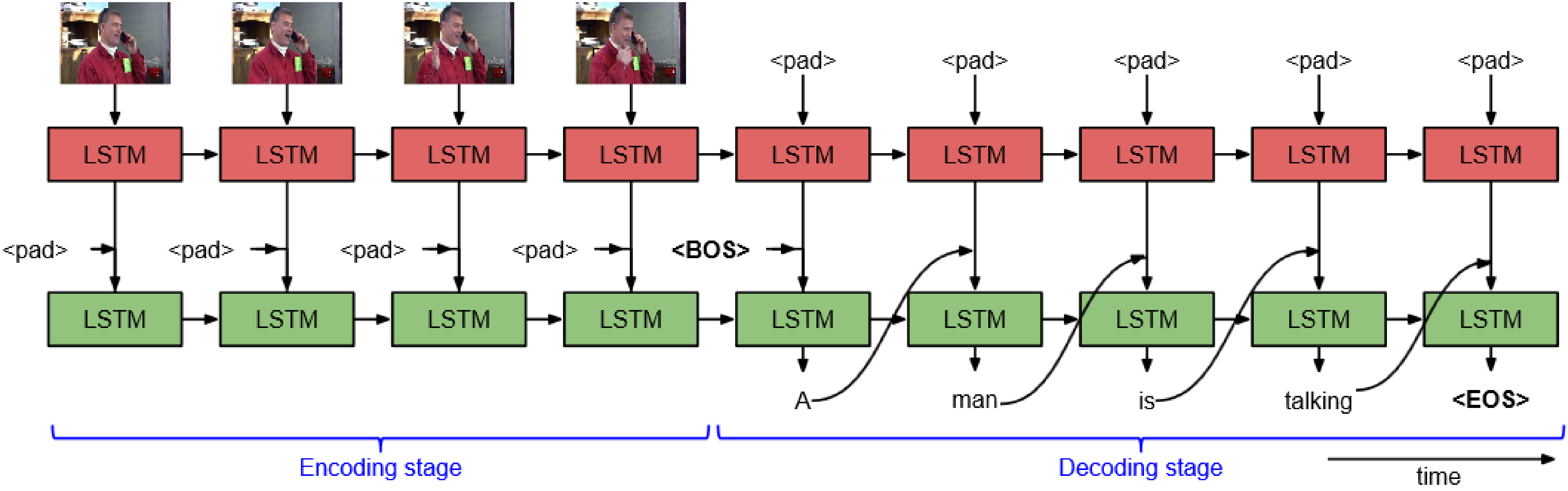
- Long short term memory (LSTM) models are special case of RNNs:
  - Designed so that model can remember things for a long time.
- LSTMs are the analogy of convolutional neural networks for RNNs:
  - The trick that makes them work in applications.
- LSTMs are getting impressive performance in various settings:
  - Cursive handwriting recognition.
    - https://www.youtube.com/watch?v=mLxsbWAYIpw
  - Speech recognition.
  - Machine translation.
  - Image and video captioning.

LSTMs for Video Captioning



http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Venugopalan_Sequence_to_Sequence_ICCV_2015_paper.pdf

LSTMs for Video Captioning



http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Venugopalan_Sequence_to_Sequence_ICCV_2015_paper.pdf

LSTMs for Video Captioning

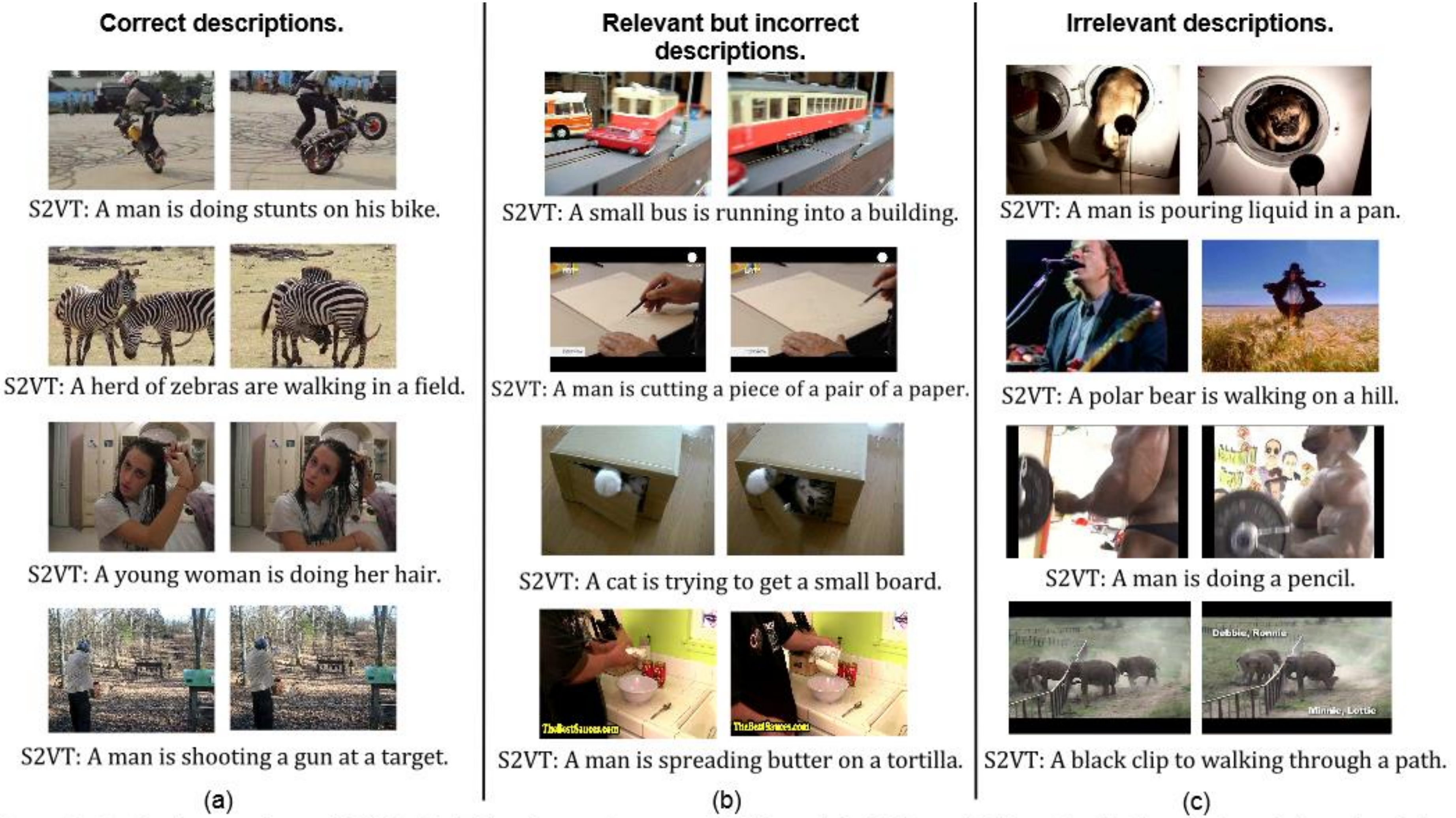


Figure 3. Qualitative results on MSVD YouTube dataset from our S2VT model (RGB on VGG net). (a) Correct descriptions involving different objects and actions for several videos. (b) Relevant but incorrect descriptions. (c) Descriptions that are irrelevant to the event in the video.

http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Venugopalan_Sequence_to_Sequence_ICCV_2015_paper.pdf

Long Short Term Memory

- In addition to usual hidden values 'z', LSTMs have memory cells 'c':
  - Purpose of memory cells is to remember things for a long time.
- Pieces of LSTM model:
  - Forget function: should we keep or forget value in a memory cell?
  - Candidate value: new value based on inputs.
  - Input function: should we take the new value?
  - Output function: should we output a value?
- Three of the above are "gate" functions:
  - Binary variables, which are approximated by sigmoids.

Vanilla RNN vs. LSTM

Vanilla Recurrent Neural Network (RNN) has a recurrence of the form

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

Previous layer, same time.
Same layer, previous time.

memory vector c_t^l. At each time step the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms. The precise form of the update is as follows:

Input
Forget
Output
Candidate

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

Cell $\rightarrow$ $c_t^l = f \odot c_{t-1}^l + i \odot g$

Output $\rightarrow$ $h_t^l = o \odot \tanh(c_t^l)$

Forget times old memory.
Input times candidate.
Output times current memory

Here, the sigmoid function sigm and tanh are applied element-wise, and W^l is a $[4n \times 2n]$ matrix.

LSTM Structure

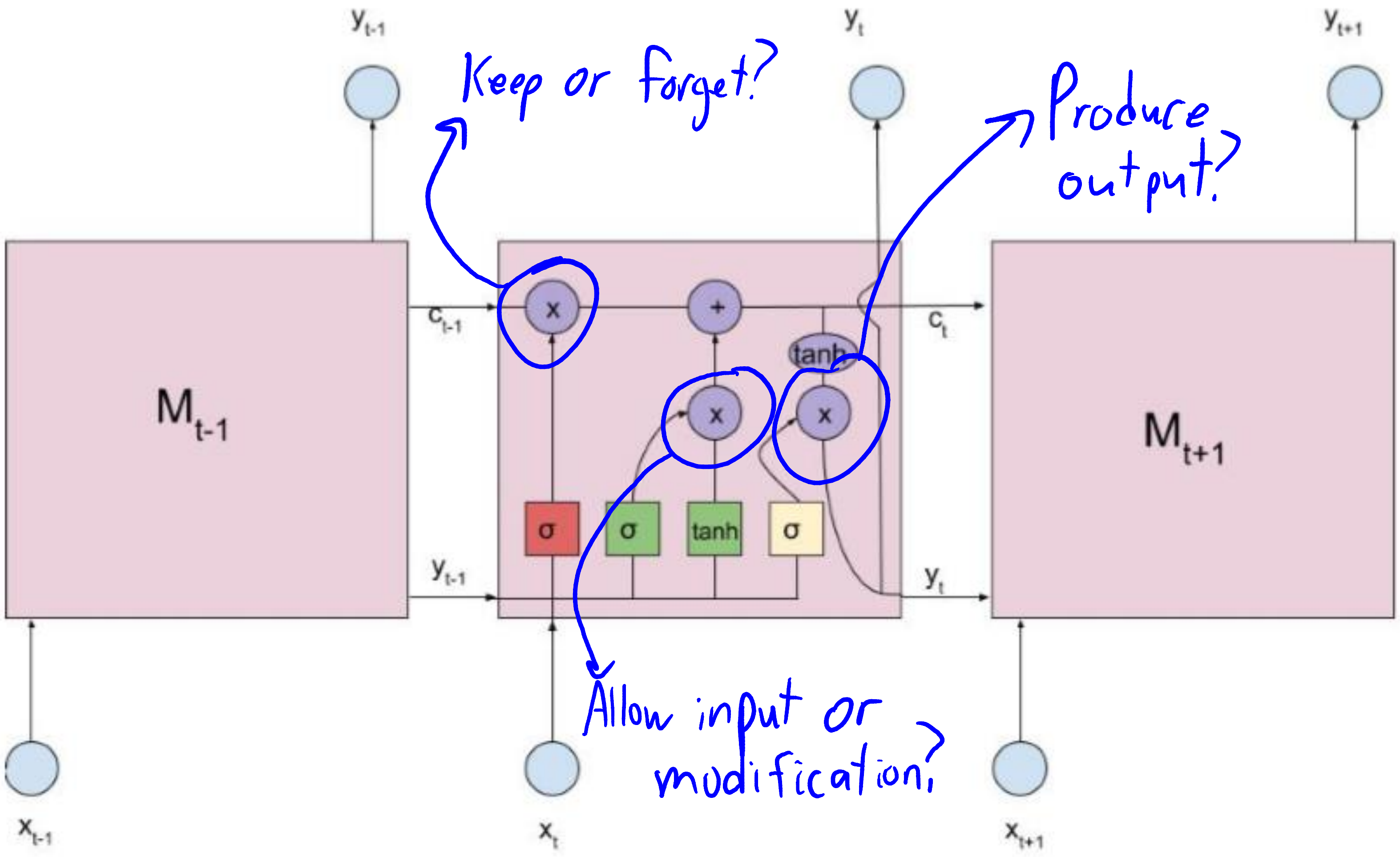


Figure 6: A close look at LSTM structure

Beyond LSTMs

- Many interesting recent variations on readable/writeable memory:
  - Memory networks and neural Turing machines.

Here is an example of what the system can do. After having been trained, it was fed the following short story containing key events in JRR Tolkien's Lord of the Rings:

Bilbo travelled to the cave.
Gollum dropped the ring there.
Bilbo took the ring.
Bilbo went back to the Shire.
Bilbo left the ring there.
Frodo got the ring.
Frodo journeyed to Mount-Doom.
Frodo dropped the ring there.
Sauron died.
Frodo went back to the Shire.
Bilbo travelled to the Grey-havens.
The End.

After seeing this text, the system was asked a few questions, to which it provided the following answers:

Q: Where is the ring?
A: Mount-Doom
Q: Where is Bilbo now?
A: Grey-havens
Q: Where is Frodo now?
A: Shire

It's probably one of the few technical papers that cite "Lord of the Rings".

https://www.facebook.com/FBAIResearch/posts/362517620591864

Outline

1. Variational Inference
2. Unsupervised Deep Learning
3. Recurrent Neural Networks
4. What’s next?

My Original Plan

- CPSC 340:

1. Data representation/summarization.
2. Supervised learning (counting/distances)
3. Unsupervised learning (counting/distances)
4. Supervised learning (linear models).
5. Unsupervised learning (latent-factor).
6. Deep Learning.
7. Sequences, time-series, and graphs.

- CPSC 540:

1. Linear models.
2. Large-Scale Learning.
3. Density Estimation (latent-factor).
4. Graphical Models.
5. Deep Learning.
6. Bayesian Methods.
7. Causal, active, and online learning.
8. Reinforcement learning.
9. Learning theory.

Topics we didn't cover

- For a preview of the red topics, see the last lecture of CPSC 340:
  - http://www.cs.ubc.ca/~schmidtm/Courses/340-F15/L35.pdf
- Other major topics we didn't cover:
  - Topic models (latent Dirichlet allocation).
  - Source separation (independent component analysis).
  - Relational models (Markov logic networks).
  - Sub-modularity (discrete version of convexity).
  - Spectral methods (consistent HMMs).

Machine Learning Reading Group

- If you want to keep going over the summer, join the MLRG:
  - http://www.cs.ubc.ca/labs/lci/mlrg

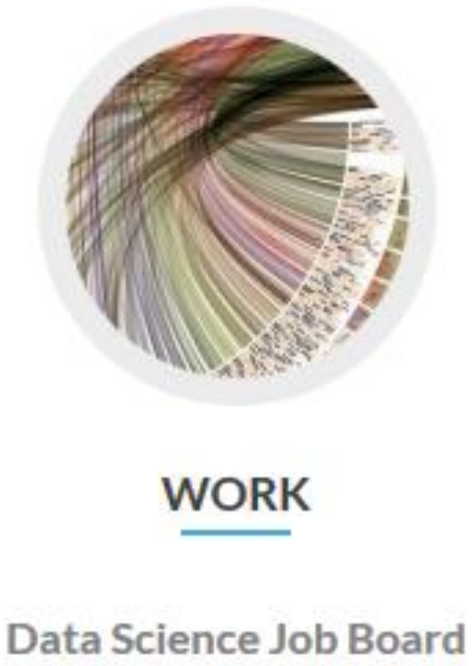
- Previous topics:
  - Summer 2015: graphical models.
  - Fall 2015: convex optimization.
  - Winter 2016: Bayesian learning.
- Future topics:
  - Summer 2016: undecided.
  - Fall 2016: deep learning.
  - Winter 2017: reinforcement learning.

Next Year

- CPSC 340 may require multivariate calculus.
  - Some material will be moved to that course.
- CPSC 5xx Courses (very tentative, check back in summer):
  - Optimization?
  - Game theory?
  - 2 ML courses?
  - Vision with deep learning emphasis?
  - Learning theory?
  - Approximate dynamic programming (= reinforcement learning)?
- Courses from other departments:
  - STAT 560/561 (~ Stats version of this material).
  - Advanced Bayesian stats (Alexandre Bouchard-Côté).
  - ML for biostatistics (Sara Mostafavi).
  - EECE 592: deep learning and reinforcement learning.

Data Science Job Board

- Many local companies are looking for people with CPSC 540 skills.
- If you are looking for local jobs, go here and make a profile.
  - http://makedatasense.ca/jobs



- Thank you for your patience, I'm still learning to teach!