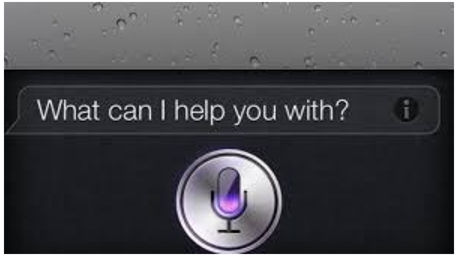
Random Projections for Probabilistic Inference

Stefano Ermon

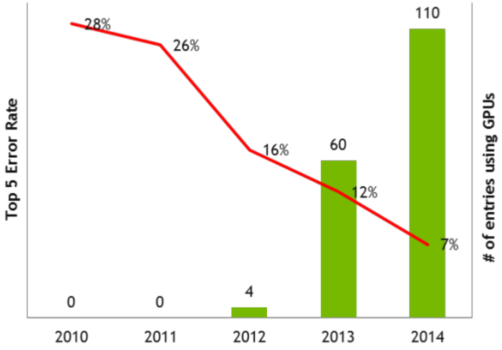
STANFORD UNIVERSITY

Recent progress in Artificial Intelligence

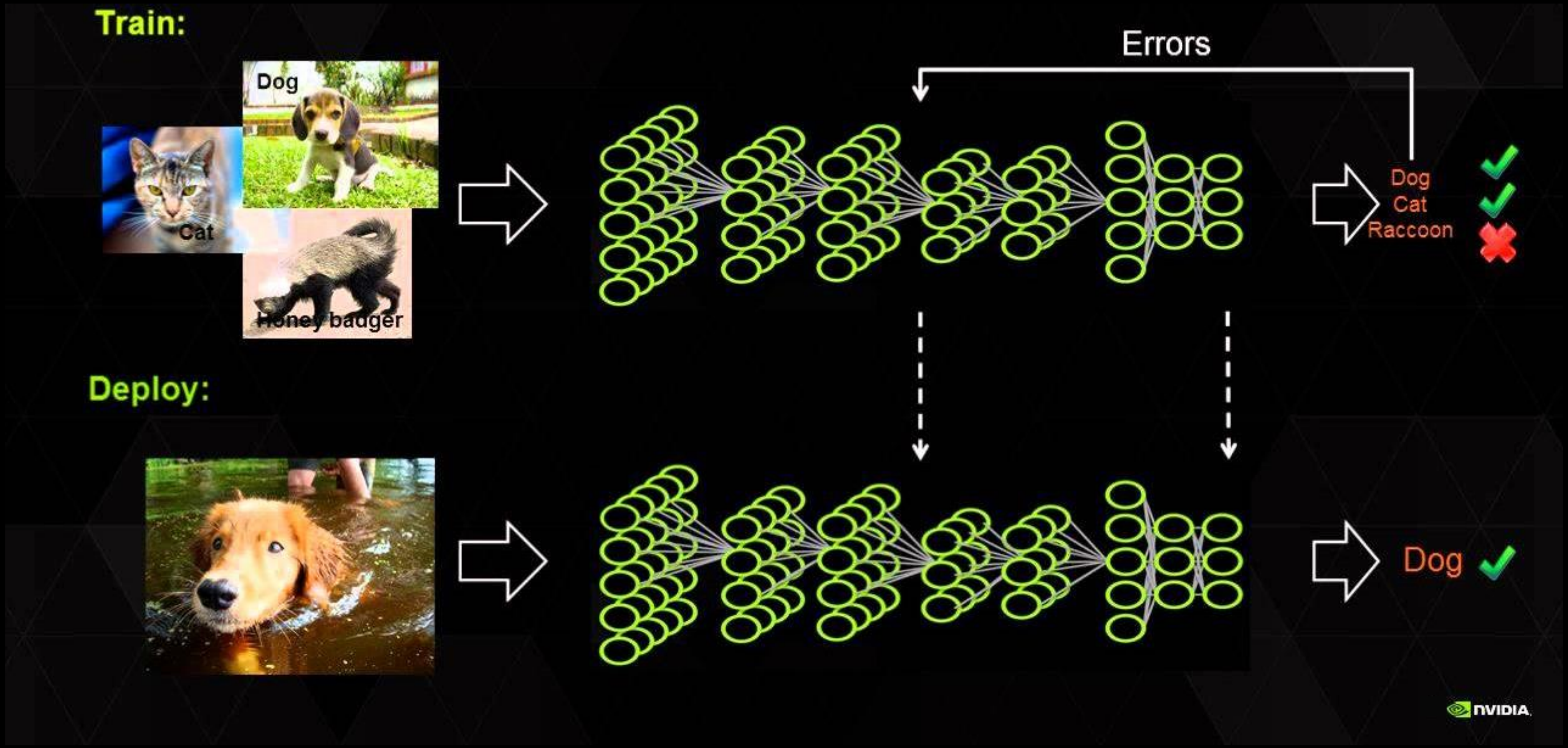


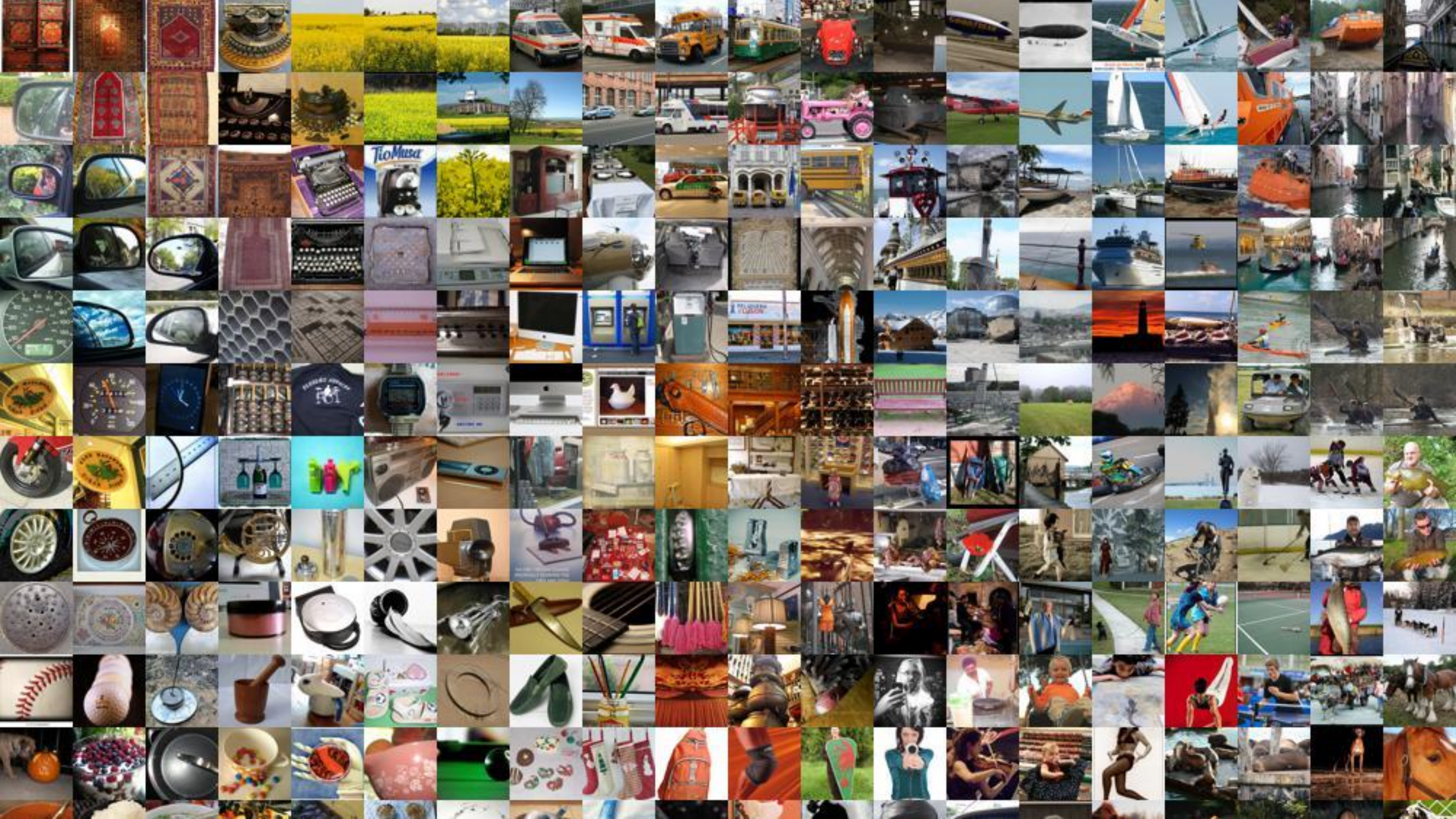
Skype Translator

IMAGENET



Supervised Learning

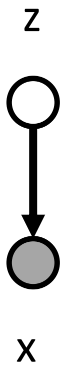



TioMusa

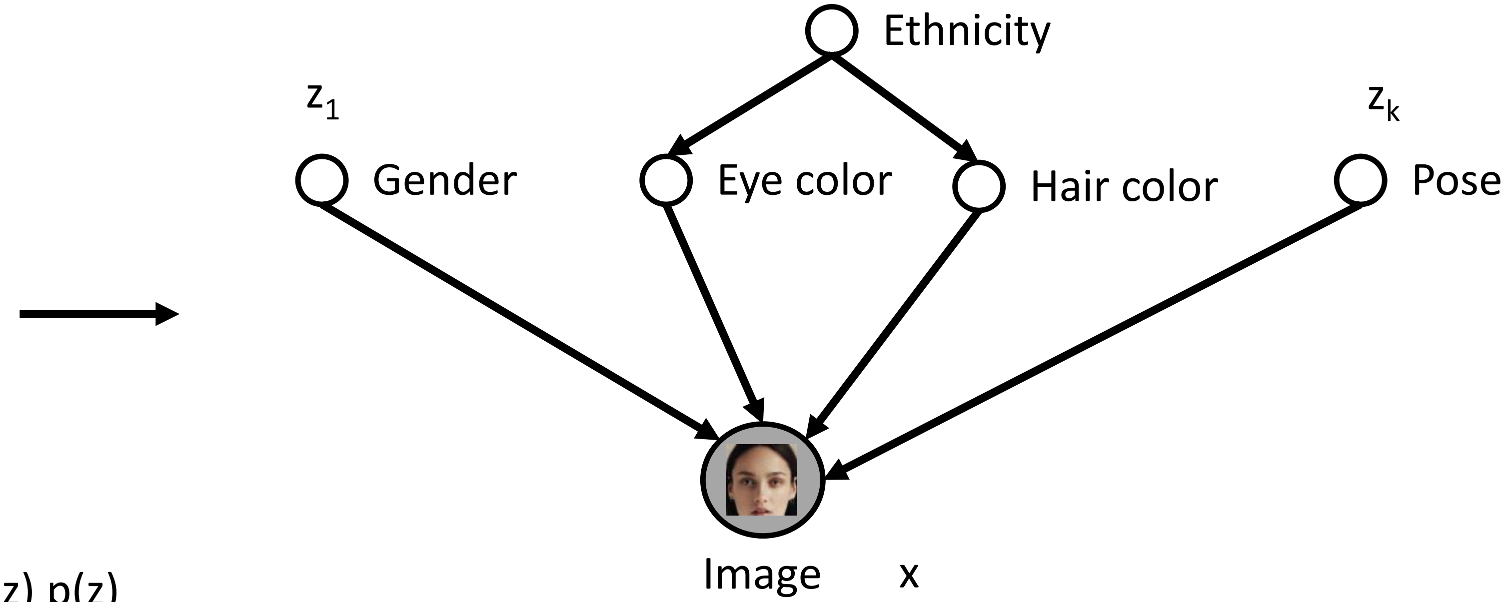
200k photos of celebrities

$\sim p(x)$

Latent Variable Model



$p(x,z) = p(x|z)\, p(z)$



A

B

C

D

E

F

G

H

I

J

K

L

True/sample? 

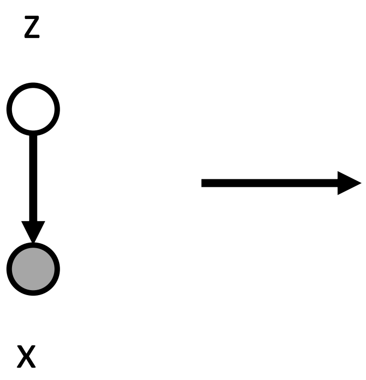
True/sample? 

“Seeds”

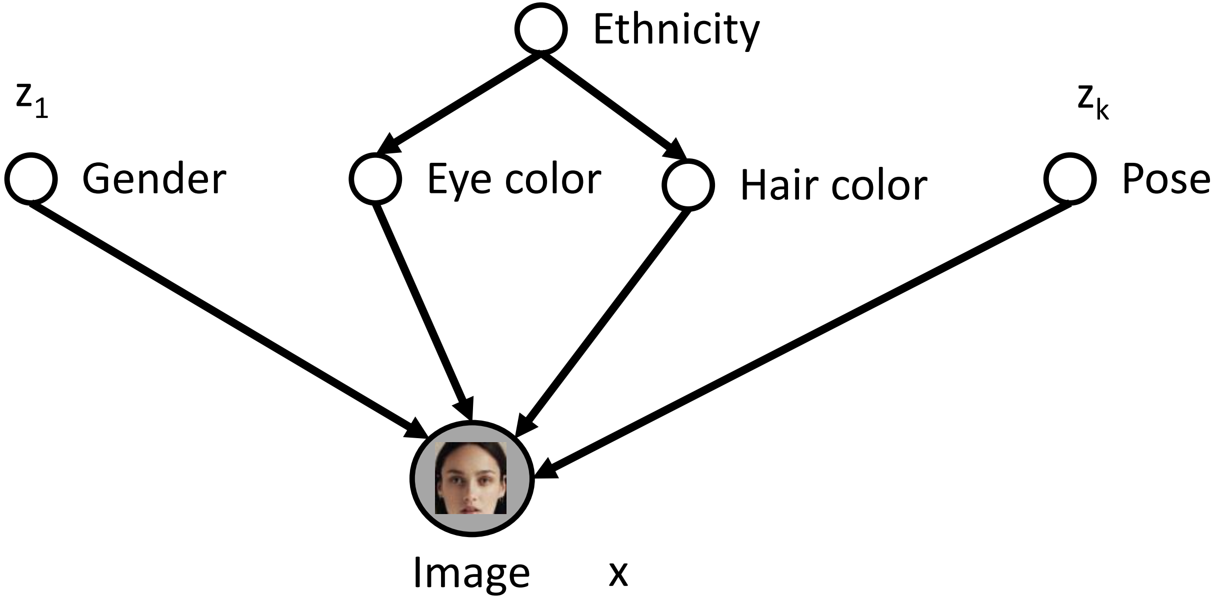
A

B

Learning Latent Variable Models



$p(x,z) = p(x|z)\ p(z)$



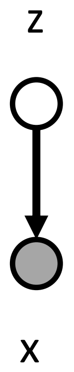
Choose parameters to maximize the (log) likelihood of the data:

$$max_{\theta} \log p(x;\theta) = max_{\theta} \log \sum_{z} p(x,z;\theta)$$

Learning Latent Variable Models

Approach: maximize the (log) likelihood of the data:

$$\log \sum_z p(x,z) = \log \sum_z q(z) \frac{p(x,z)}{q(z)}$$

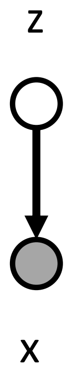


Learning Latent Variable Models

Approach: maximize the (log) likelihood of the data:

$$\log \sum_z p(x,z) = \log \sum_z q(z) \frac{p(x,z)}{q(z)}$$

$$\log \sum_z p(x,z) \geq \sum_z q(z) \log \frac{p(x,z)}{q(z)}$$



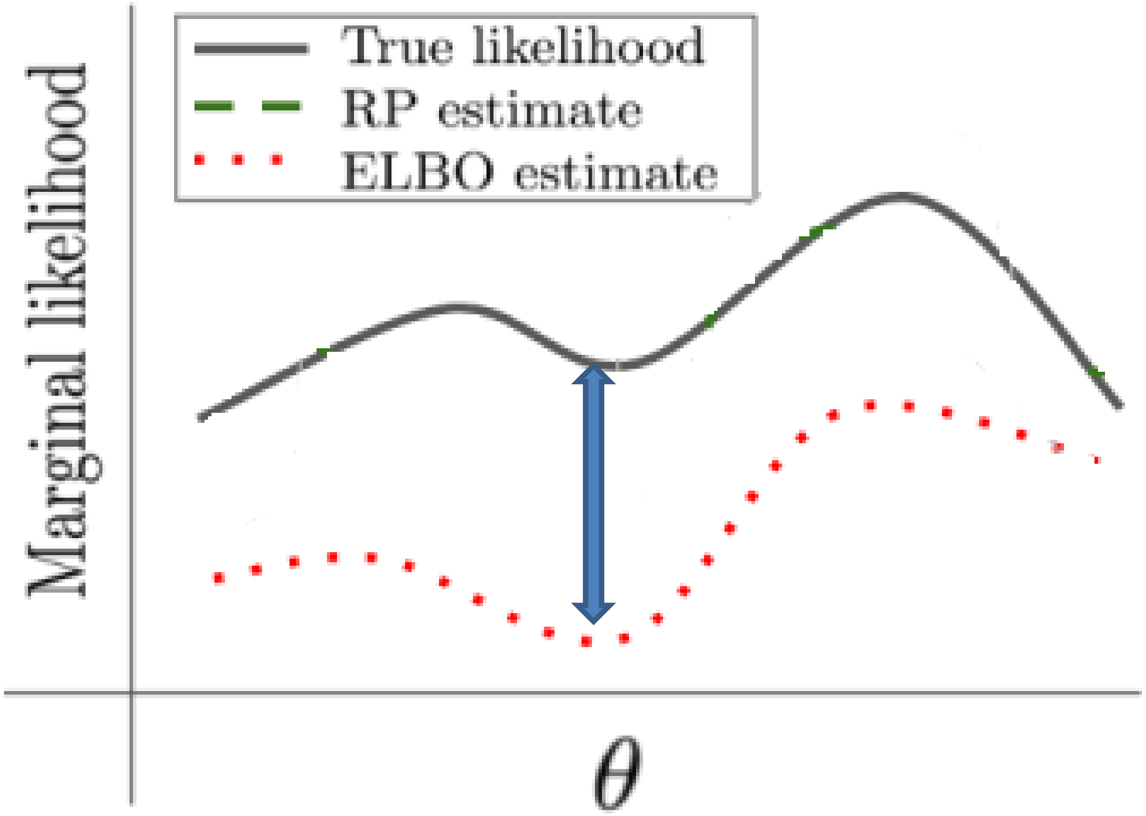
Learning Latent Variable Models

Approach: maximize the (log) likelihood of the data:

$$\log \sum_z p(x,z) = \log \sum_z q(z) \frac{p(x,z)}{q(z)}$$

$$\log \sum_z p(x,z) \geq \sum_z q(z) \log \frac{p(x,z)}{q(z)}$$

Idea: choose q to be simple


True likelihood
RP estimate
ELBO estimate
Marginal likelihood
θ

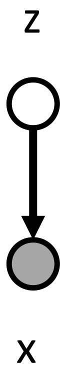
Learning Latent Variable Models

Approach: maximize the (log) likelihood of the data:

$$\log \sum_z p(x,z) = \log \sum_z q(z) \frac{p(x,z)}{q(z)}$$

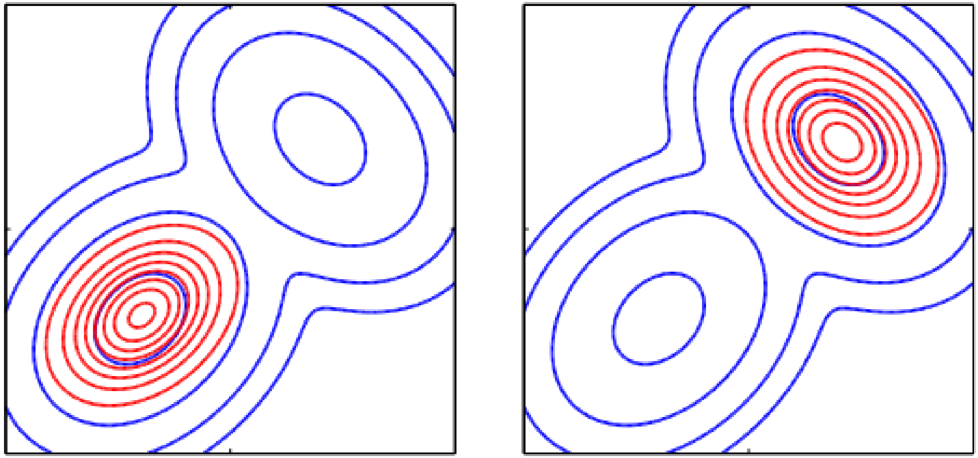
$$\log \sum_z p(x,z) \geq \sum_z q(z) \log \frac{p(x,z)}{q(z)}$$

$$\log \sum_z p(x,z) = \sum_z q(z) \log \frac{p(x,z)}{q(z)} \boxed{+ KL(q, p(z|x))}$$



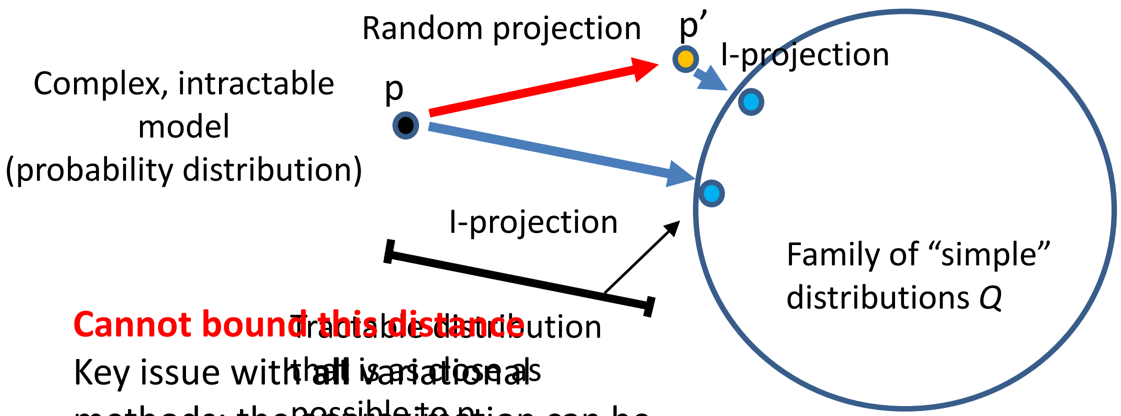
Variational Inference

Approximate a complex distribution ***p*** using a simpler (tractable) distribution ***q****
(e.g., a Gaussian distribution)



p=Blue, q^*=Red (two equivalently good solutions!)

Variational Methods and Random Projections



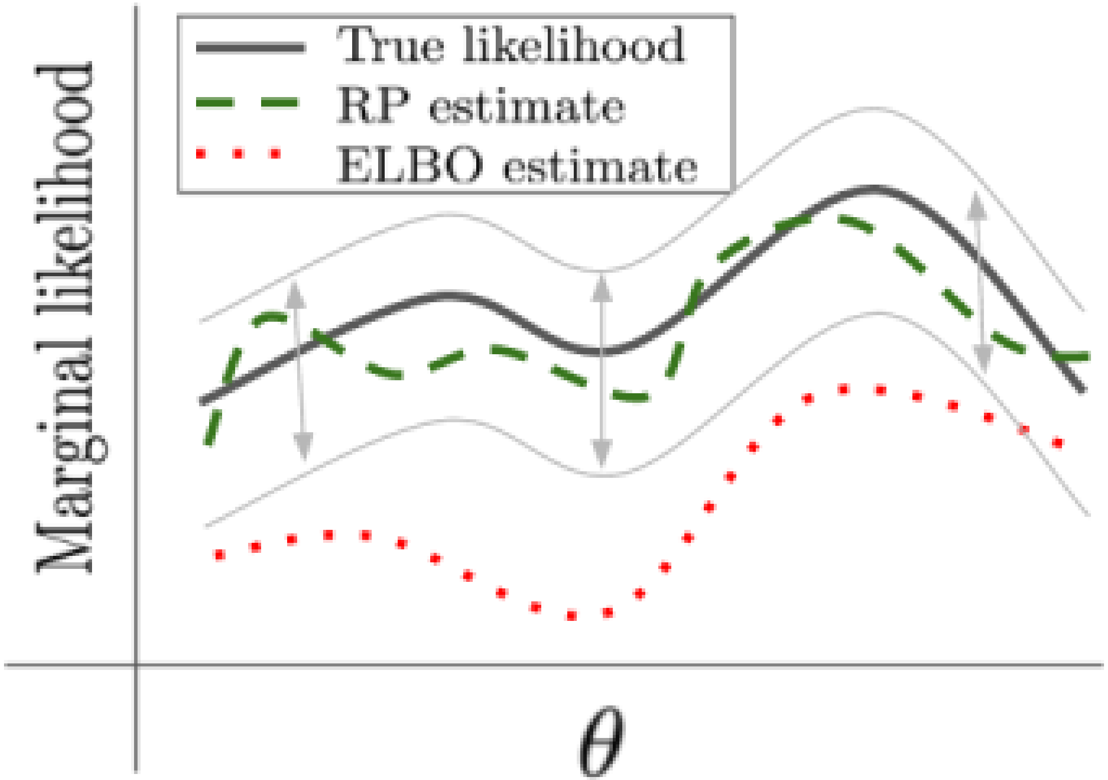
Key issue with **all** variational methods: the approximation can be arbitrarily bad

Taking a random projection of *p*:

1. Properties of p are preserved
2. p' is "simpler" and therefore "closer" to *Q*

Idea: take a random projection first, then an I-projection [AISTATS-16, NIPS-16]

Main result (informal): ***good approximation with high probability***



True likelihood
RP estimate
ELBO estimate
Marginal likelihood
θ

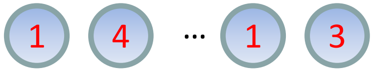
Outline

1. Introduction ✓
2. Probabilistic inference by hashing and optimization
   I. Formalism
   II. Random Projections
3. Combining random projections and I-projections
4. Other classes of random projections
5. Conclusions

Probabilistic Inference in High Dimensions

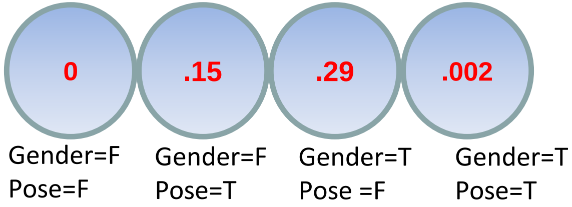
- We are given
  - a set of 2^n configurations (= assignments of z vars)
  - non-negative weights *w*
    - *from Bayes Net, factor graph, weighted CNF..*
- Goal: compute **total weight**

2^n configurations



$$\sum_z p(x,z) = \sum_z w(z)$$

- *Example 1*: n=2 binary variables, sum over 4 configurations



P[Face]

$= \sum_{x=\{T,F\}} \sum_{y=\{T,F\}}$ P[Gender=x, Pose=y, Image=Face]

$= 0 + 0.15 + 0.29 + 0.002 = 0.442$

- *Example 2*: n=100 variables, sum over $2^{100} \approx 10^{30}$ terms (**curse of dimensionality**)

$$w(z) \in \{0,1\}$$

How Might One Count?

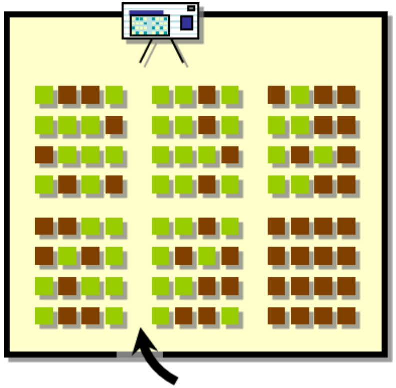
Problem characteristics:

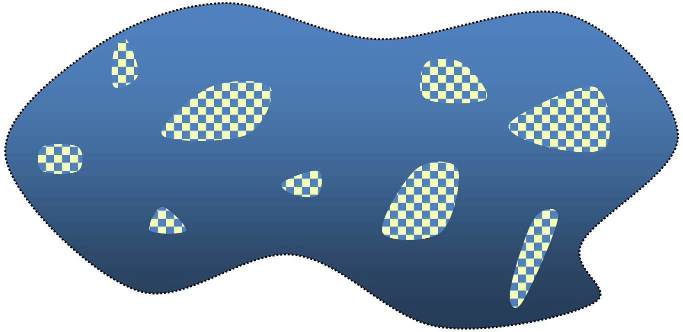
- Space naturally divided into rows, columns, sections, …
- Many seats empty
- Uneven distribution of people (e.g. more near door, aisles, front, etc.)

Analogy: How many people are present in the hall?

From Counting People to Marginal Inference

– Auditorium : search space
– Seats : 2^n truth assignments of the z variables
– Occupied seats : assignments with non-zero weight

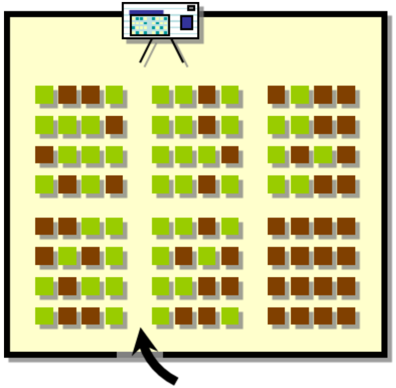


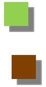


: occupied seats (47) = assignments with non-zero weight

: empty seats (49)

#1: Brute-Force Counting





Idea:

- Go through every seat
- If occupied, increment counter

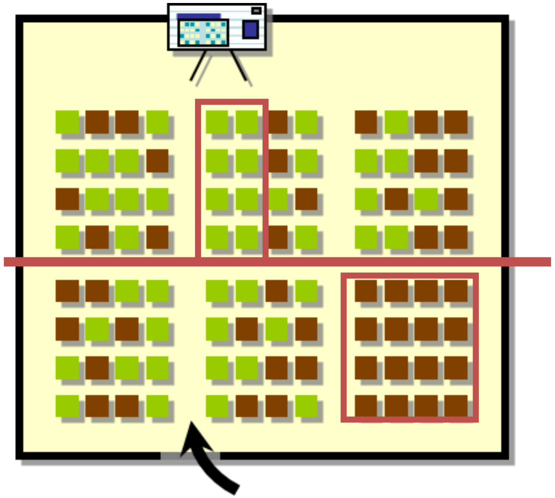
Advantage:

- Simplicity, accuracy

Drawback:

- Scalability

#2: Branch-and-Bound (DPLL-style)



Idea:

- Split space into sections
  e.g. front/back, left/right/ctr, …
- Use smart detection of full/empty sections
- Add up all partial counts

Advantage:

- Relatively faster, exact

Drawback:

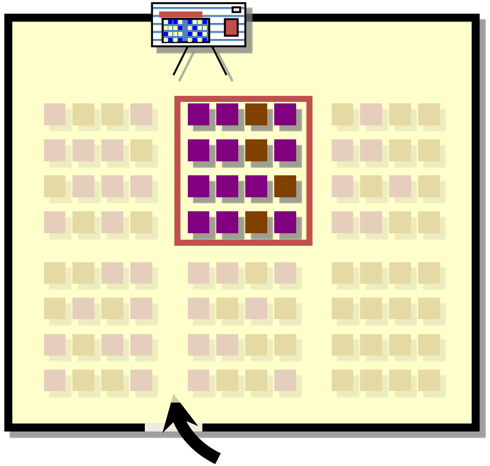
- Still "accounts for" every single person present: need extremely fine granularity
- Scalability

Framework used in DPLL-based systematic exact counters
e.g. Cachet [Sang-et]

Related to compilation approaches [Darwiche et. al]

Approximate model counting?

#3: Estimation By Sampling -- Naïve



Idea:

- Randomly select a region
- Count within this region
- Scale up appropriately

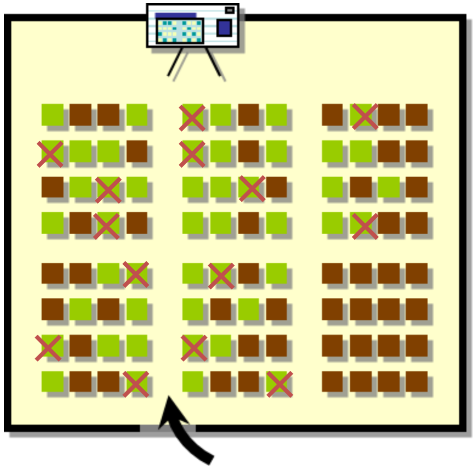
Advantage:

- Quite fast

Drawback:

- Robustness: can easily under- or over-estimate
- Scalability in sparse spaces: e.g. 10^{60} solutions out of 10^{300} means need region much larger than 10^{240} to “hit” any solutions

Let's Try Something Different …



A Distributed Coin-Flipping Strategy
(Intuition)

Idea:

Everyone starts with a hand up

– Everyone tosses a coin
– If heads, keep hand up,
  if tails, bring hand down
– Repeat till no one hand is up

Return $2^{\#(\text{rounds})}$

Does this work?

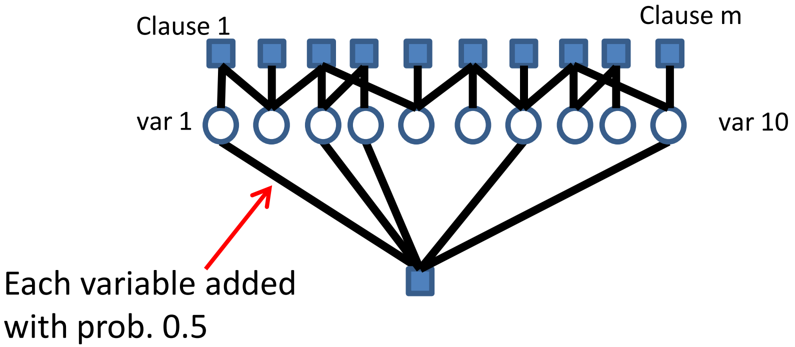
- On average, Yes!
- With M people present, need roughly $\log_2 M$ rounds for a unique hand to survive

Making the Intuitive Idea Concrete

- How can we make each configuration “flip” a coin?
- How do we transform the average behavior into a robust method with **provable correctness guarantees**?
- Many approaches based on this idea (originated from theoretical work due to [Stockmeyer-83, Valiant-Vazirani-86, etc.]):
  - Mbound, XorSample [Gomes et al-2007]
  - WISH, PAWS [Ermon et al-2013]
  - ApproxMC, UniWit,UniGen [Chakraborty et al-2014,2016]
  - Achilioptas et al UAI-15 (error correcting codes)
  - Belle et al. at UAI-15 (SMT solvers)

Random parity constraints

- XOR/parity constraints:
  - *Example:* $a \oplus b \oplus c \oplus d = 1$ satisfied if an odd number of a,b,c,d are set to **1**



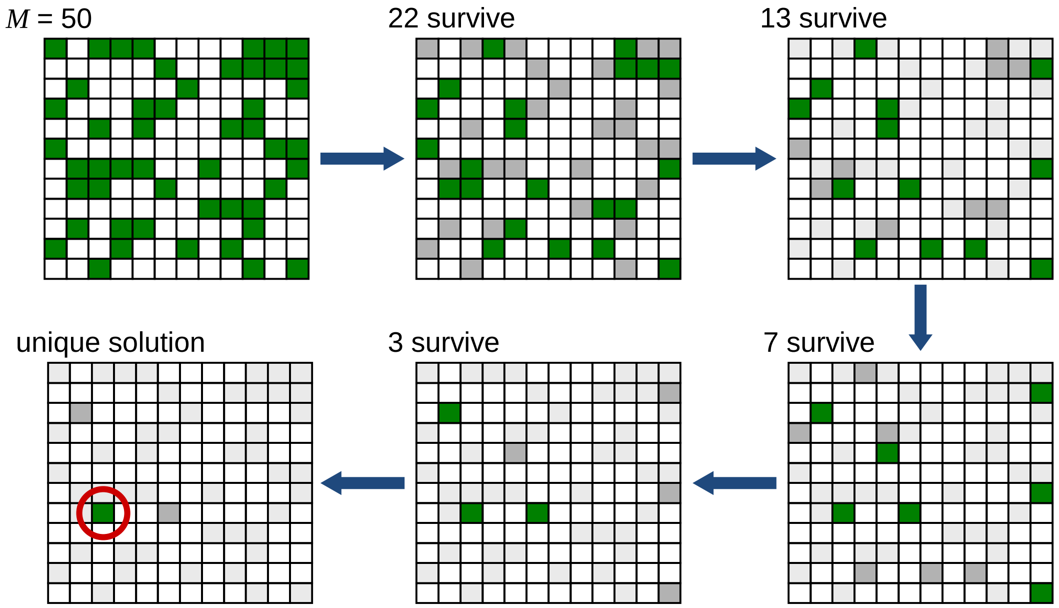
Randomly generated parity constraint X

$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = \mathbf{1}$$

- Each solution satisfies this random constraint with probability ½
- Pairwise independence: For every two configurations ***A*** and ***B***, "***A*** satisfies ***X***" and "***B*** satisfies ***X***" are independent events

The Desired Effect

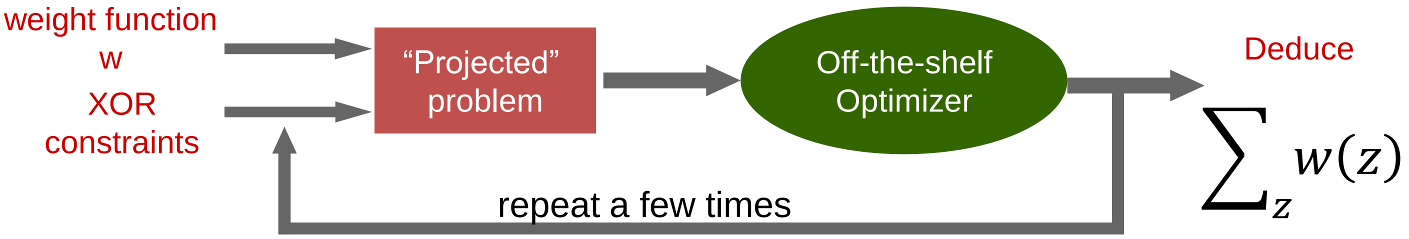
If each XOR cut the solution space roughly in half, would get down to a unique solution in roughly $\log_2 M$ steps!

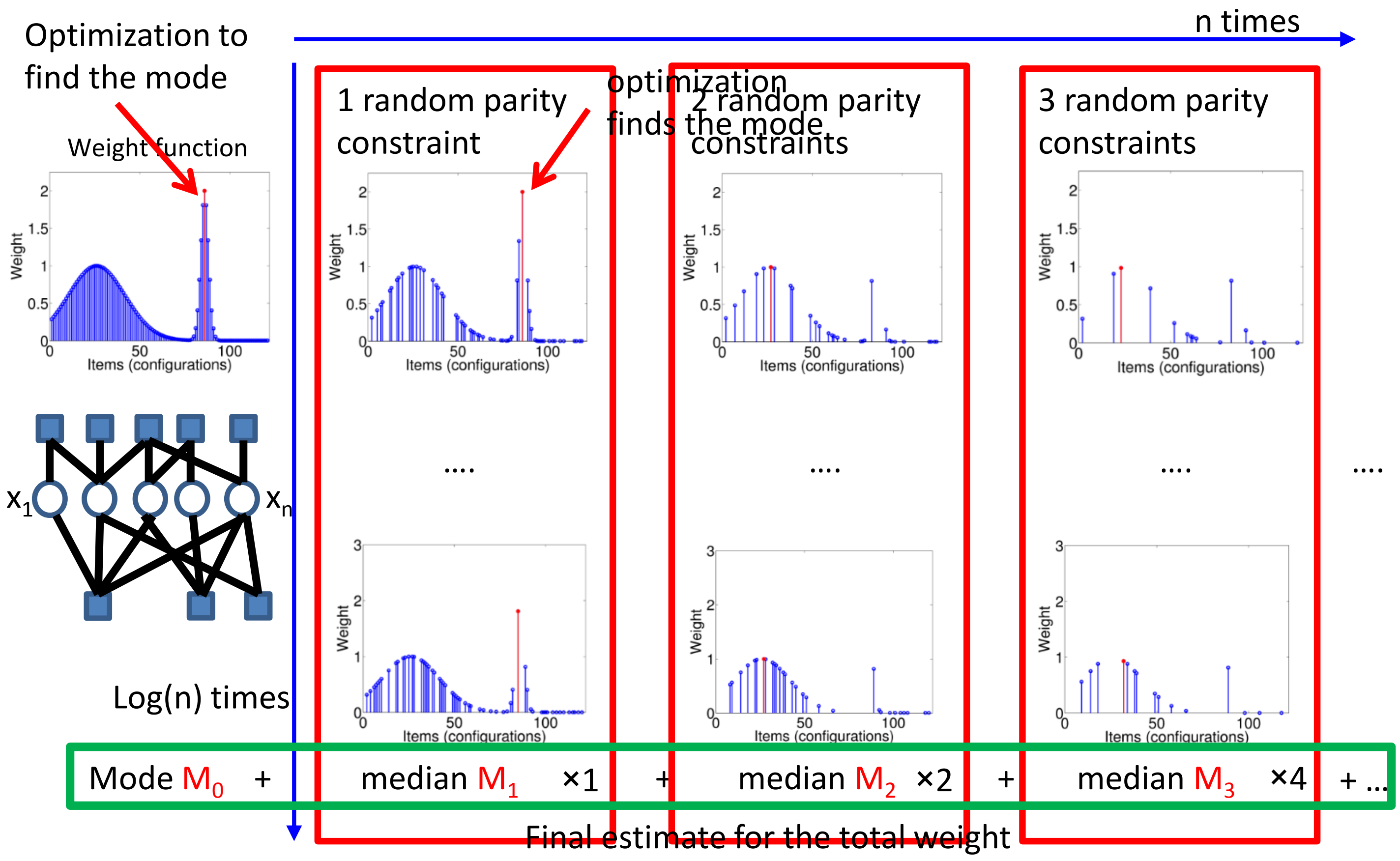


Hashing for Weighted Problems

Given a **weight function** $w(z) = p(x, z)$

1. Add some XOR constraints to w to get w'
   (this reduces the degrees of freedom)
2. Find MAX-weight assignment of w'
3. Conclude "something" about the **total weight** $\sum_z w(z)$





Optimization to find the mode
n times
1 random parity constraint
optimization finds the mode
2 random parity constraints
3 random parity constraints
Weight function
Weight
Items (configurations)
x_1
x_n
....
Log(n) times
Mode M_0 + median M_1 ×1 + median M_2 ×2 + median M_3 ×4 + ...
Final estimate for the total weight

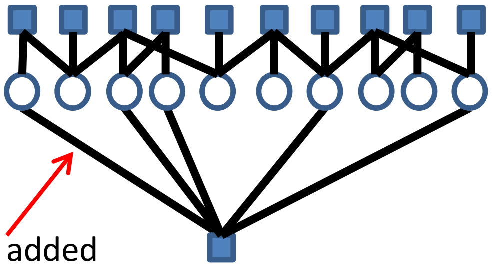
Accuracy Guarantees

Result (stated informally):

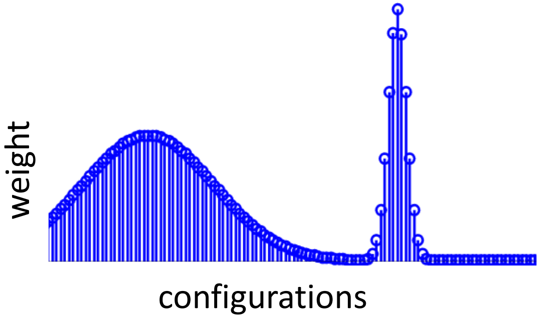
With high probability WISH (**W**eighted-**S**ums-**H**ashing) computes a constant factor approximation to the total weight and it requires solving $\theta(n \log n)$ **optimization instances.**

Hashing as a random projection

- XOR/parity constraints:
  - *Example:* $a \oplus b \oplus c \oplus d$ = 1 satisfied if an odd number of a,b,c,d are set to **1**



Each variable added with prob. 0.5

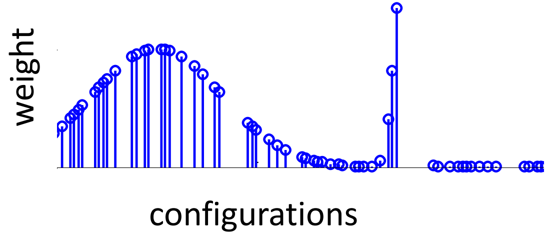


Randomly generated parity constraint X

$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = \mathbf{1}$$

Set weight to zero if constraint is not satisfied

Random projection



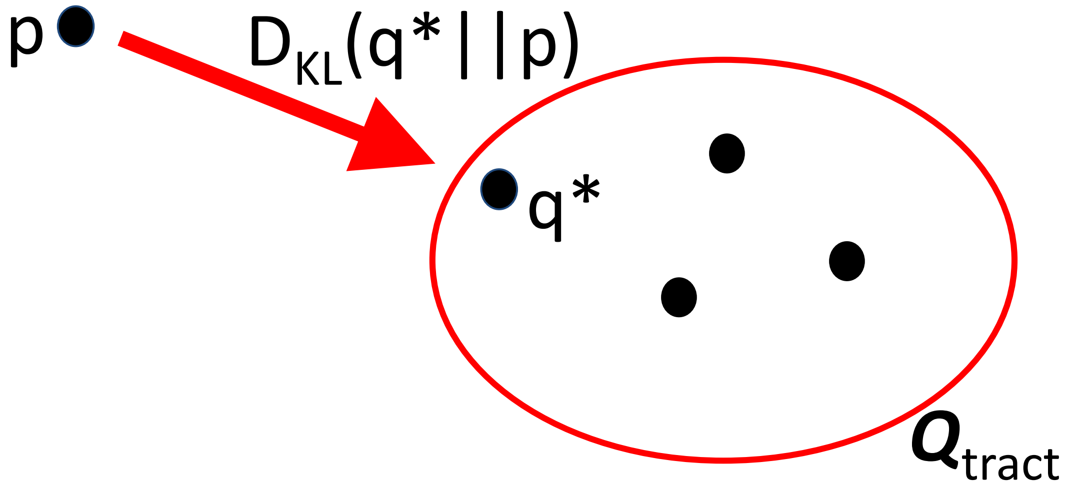
This random projection:

1. "**simplifies**" the model
2. preserves its "key properties"

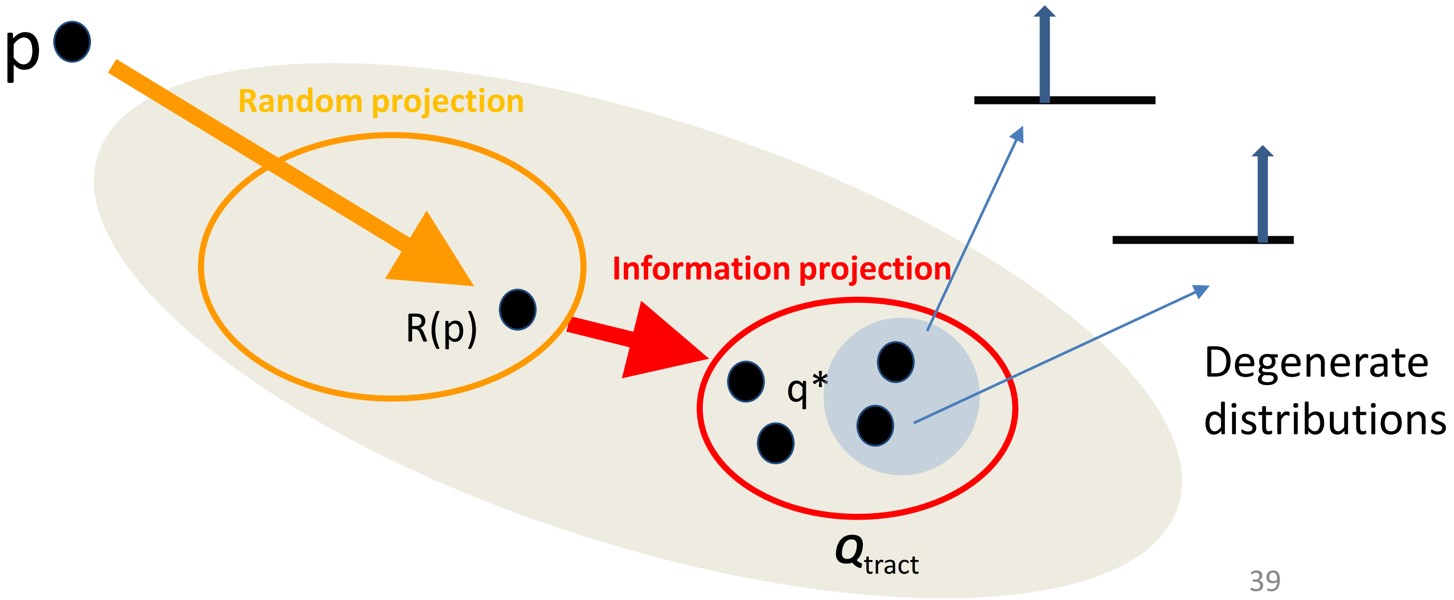
Outline

1. Introduction ✓
2. Probabilistic inference by hashing and optimization ✓
   I. Formalism
   II. Random Projections
3. Combining random projections and I-projections
4. Other classes of random projections
5. Conclusions

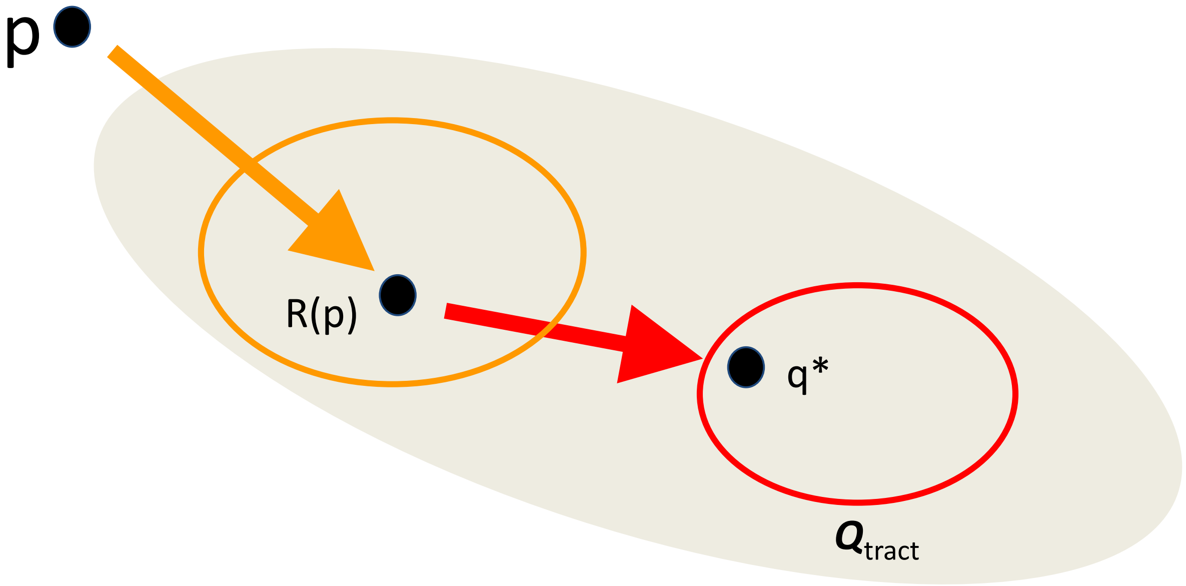
Variational Inference



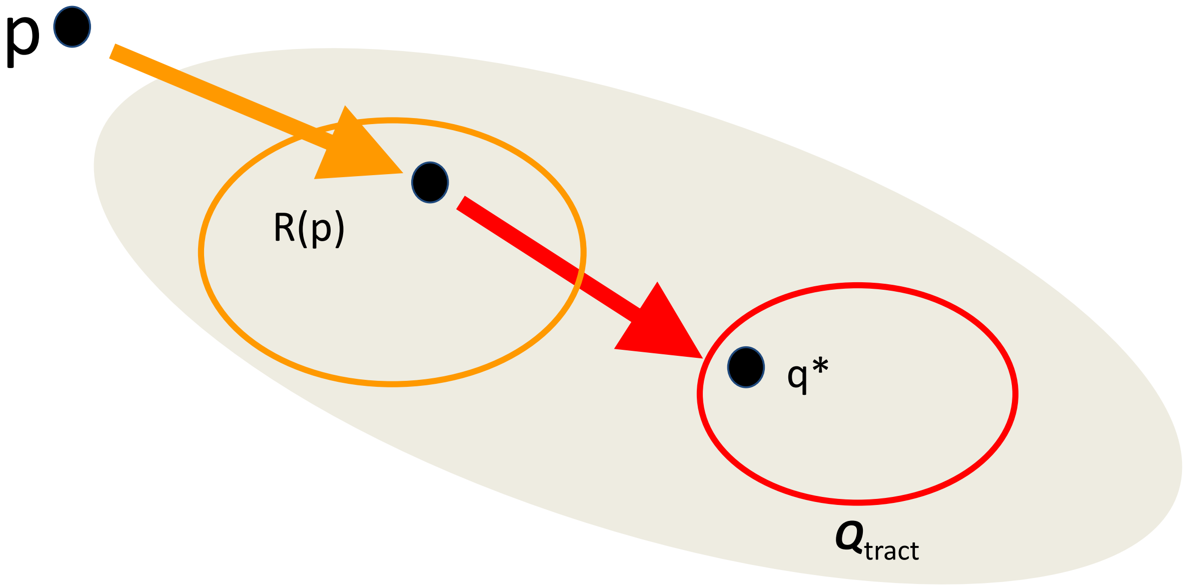
Variational Inference with Random Projections



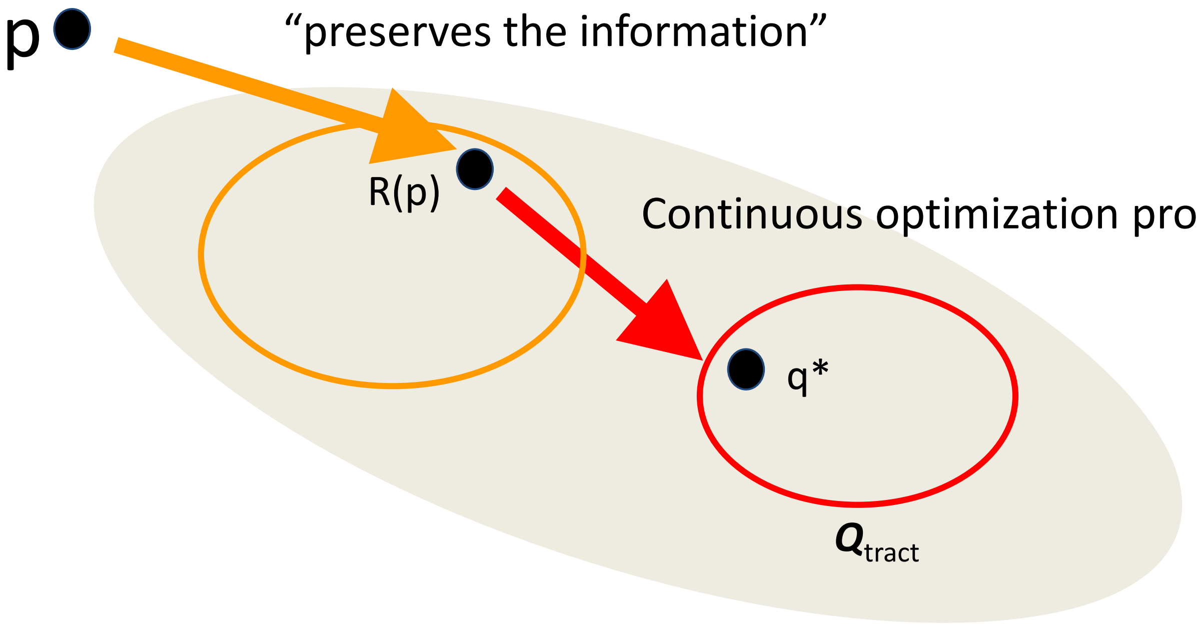
Variational Inference with Random Projections



Variational Inference with Random Projections

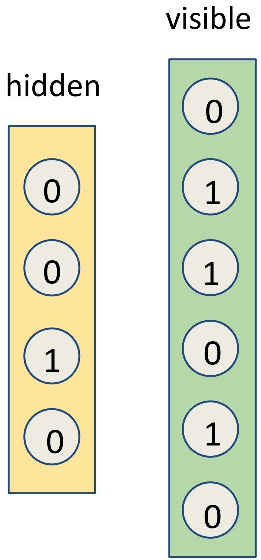


Variational Inference with Random Projections



Result (informal statement): after the random projection (e.g., using the right number of XORs), the resulting distribution can be **well approximated** using standard variational inference (with accuracy guarantees)

Algorithm: Mean Field with Random Projections



Fully factored distribution:

$$q(z) = q(z_1)\, q(z_2) \ldots q(z_n)$$

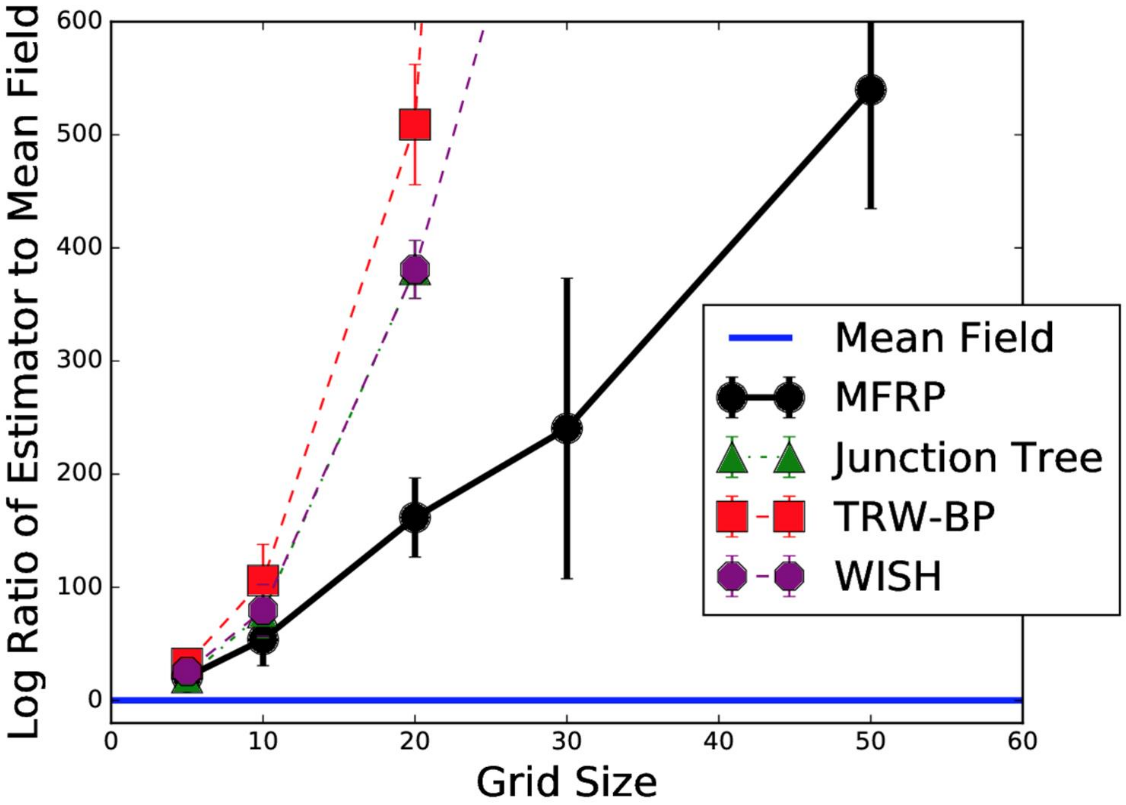
Randomly-projected mean field:

Variational objective is

- coordinate-wise concave
- has fewer variables

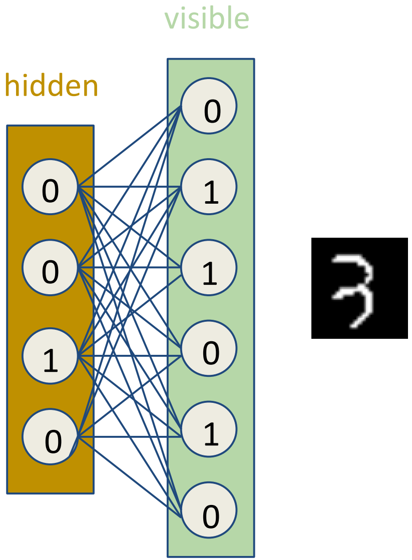
<u>If</u> we find **global optimum**, then accuracy guarantees

Ising Model



Log Ratio of Estimator to Mean Field
Grid Size
0
100
200
300
400
500
600
0
10
20
30
40
50
60
Mean Field
MFRP
Junction Tree
TRW-BP
WISH

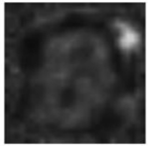
Boltzmann Machines



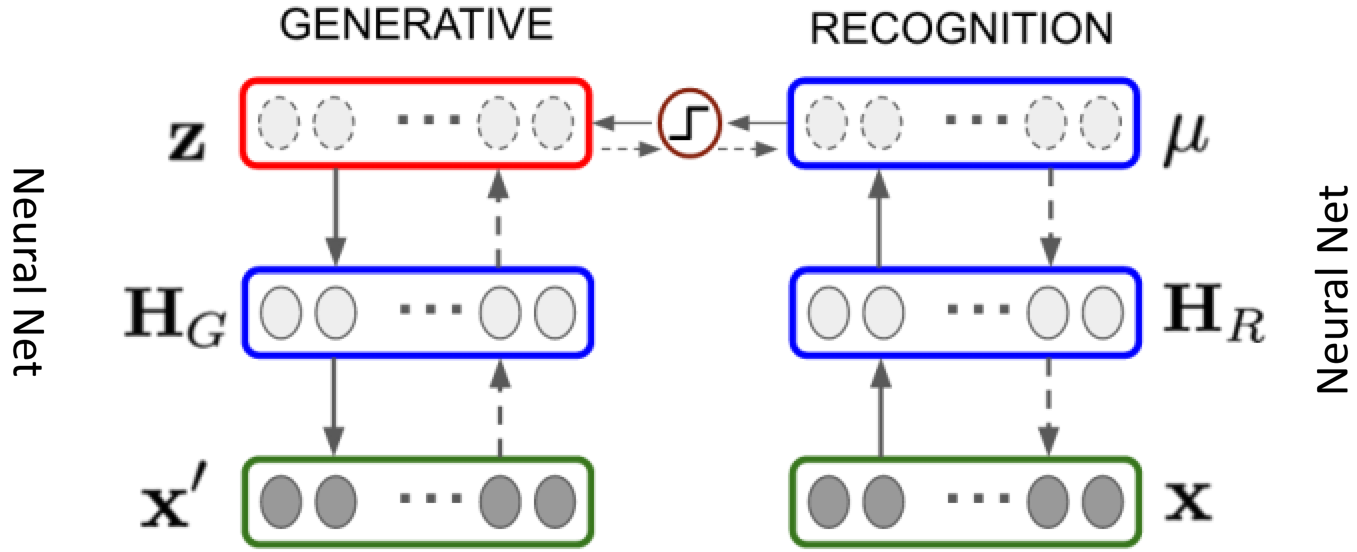
Discrete graphical model:

p(h, v) = f(h, v) / Z

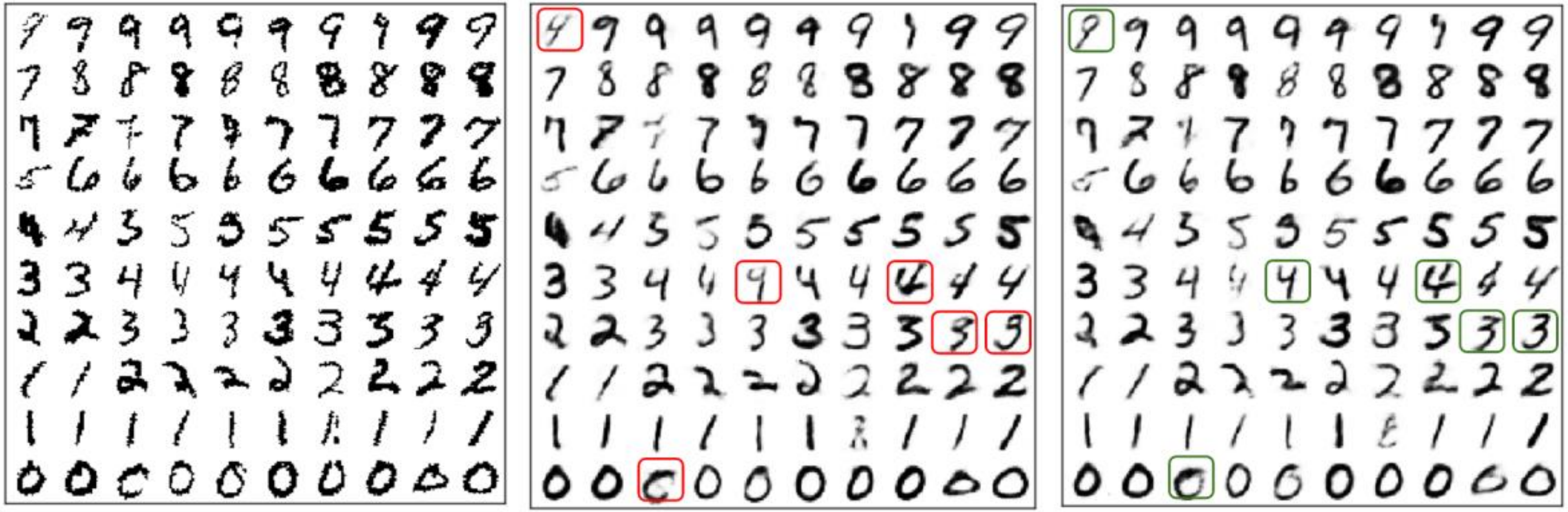
RBMs

No. Hidden Nodes	100	100	100	200	200	200
CD-k	1	5	15	5	15	25
MF $\log Z$	501	348	297	203	293	279
MFRP $\log Z$	501	**433**	**342**	**380**	**313**	**295**
MF μ						
MFRP μ						

Variational Autoencoders



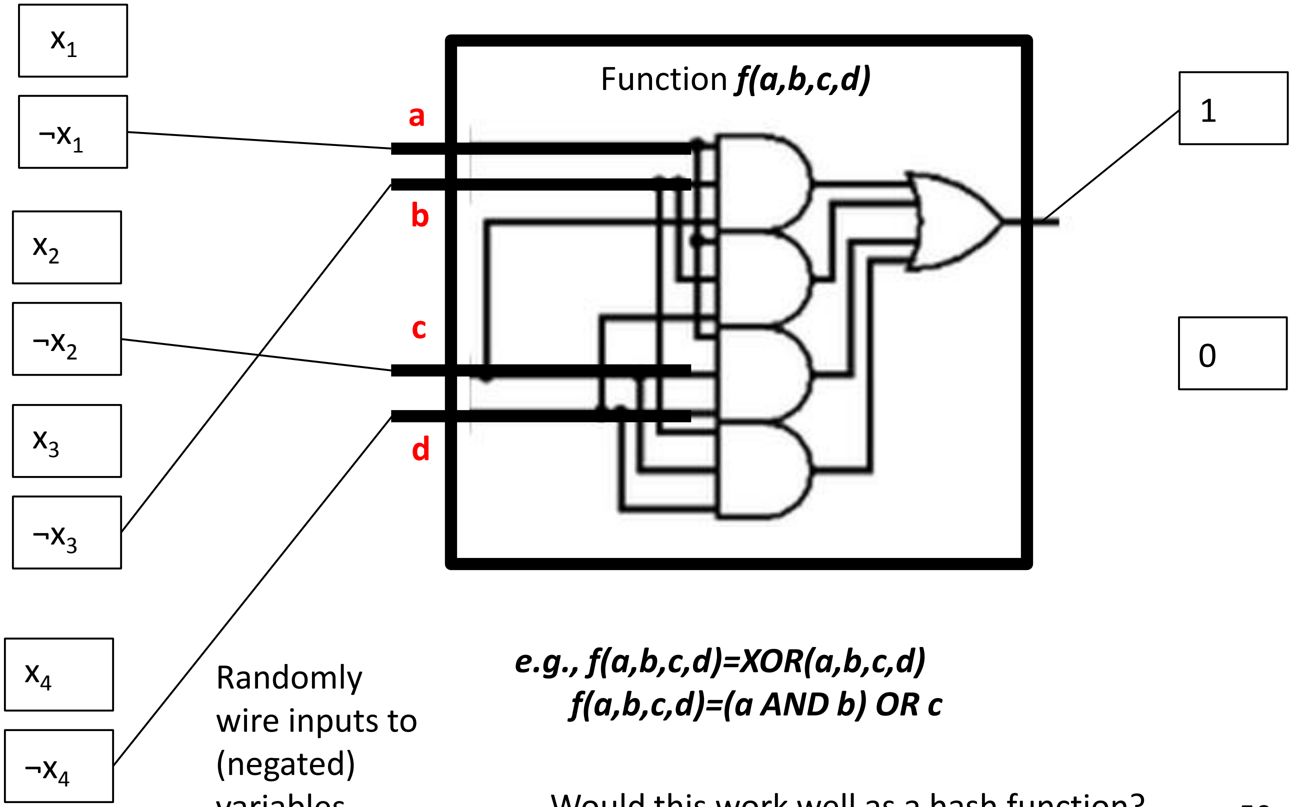
Denoising results



Outline

1. Introduction ✓
2. Probabilistic inference by hashing and optimization ✓
   I. Formalism
   II. Random Projections
3. Combining random projections and I-projections ✓
4. Other classes of random projections

Random Constraints for Probabilistic Inference



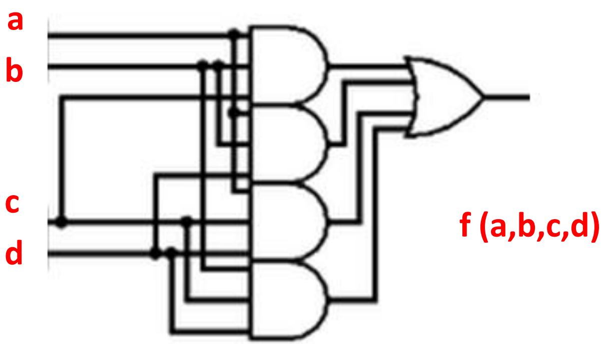
Randomly wire inputs to (negated) variables

e.g., f(a,b,c,d)=XOR(a,b,c,d)

f(a,b,c,d)=(a AND b) OR c

Would this work well as a hash function?

Noise sensitivity

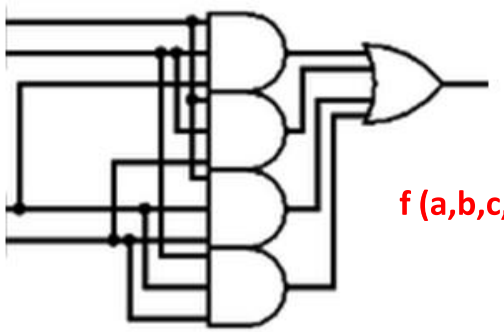


Performance of this family of hash functions depends on the **noise sensitivity** of **f (a,b,c,d)**

- Given a random input x, e.g. **x**=(0, 1,1,0)
- Randomly flip each bit of x with probability p → **x'**=(1, 1,1,0)
- How likely is that f(x) = f(x')?

Result (informal statement): The more noise sensitive a function f is, the better the corresponding family of hash functions (with random wiring) behaves. [ICML-16]

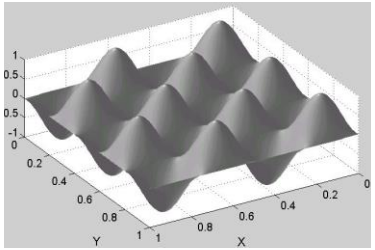
Random Constraints for Probabilistic Inference and Model Counting

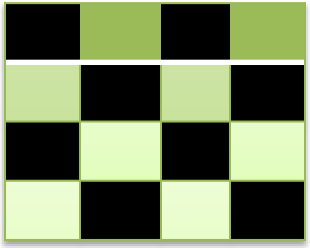


f (a,b,c,d)

- Noise sensitivity can be computed in *closed form* from the Fourier spectrum of **f.** Known for many common functions!
- Intuitively, the more "oscillatory" **f** is, the more noise sensitive it is

2-D sine wave





2-D parity function

Experimental results

INSTANCE	GT	LB_{trib}	LB_{xor}	t_{trib}	t_{xor}
LS10	24	18	19	**1**	209
LS11	33	25	24	**28**	623
LS12	—	32	29	**34**	658
LS13	—	33	34	**3**	74
LS14	—	36	34	**12**	761
LS15	—	**39**	34	**51**	829
20RDR45	—	29	29	**1**	523
23RDR45	—	**27**	10	**7**	800

Tribes: One or two orders of magnitude faster!

$$(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_4 \wedge \neg x_5) \vee (\neg x_6 \wedge x_7)$$

Conclusions

- Random projections and hashing have been extremely useful in Information Retrieval and Machine Learning
- Can also be applied to more complex objects, like probability distributions
- New algorithms with improved guarantees and practical performance
- Exciting ML applications: learning generative models of data