Speeding up Gradient-Based Inference and Learning in deep/recurrent Bayes Nets with Continuous Latent Variables

(by a neural network equivalence)

Preprint: http://arxiv.org/abs/1306.0733

Diederik P. (Durk) Kingma
Ph.D. Candidate
www.dpkingma.com
Universiteit van Amsterdam

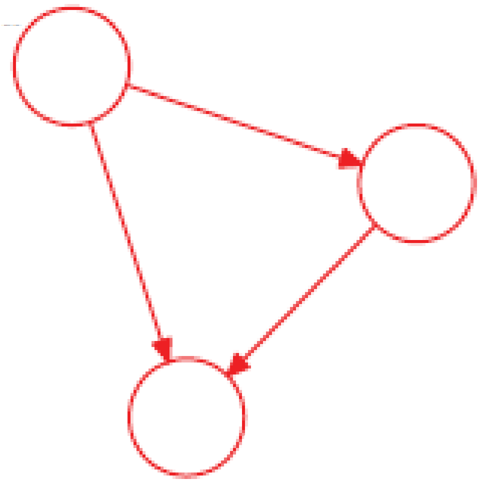
Bayesian networks

- Intro: Bayes nets with continuous latent vars
  - Example: generative model of MNIST
- Inference/learning with EM and HMC
  - Fast, except in deep/recurrent bayes nets
- Trick: Auxiliary form
  - More efficient inference/learning in an auxiliary latent space

Bayesian networks
(with continuous latent variables)

- Joint p.d.f. = product of prior/conditional p.d.f.'s

$$p(a, b, c) = p(c|a, b)p(b|a)p(a)$$



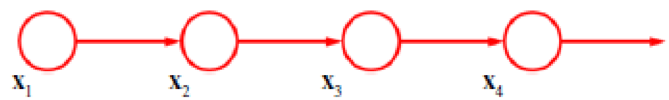
- We can have use conditional p.d.f.'s and any DAG graph structure

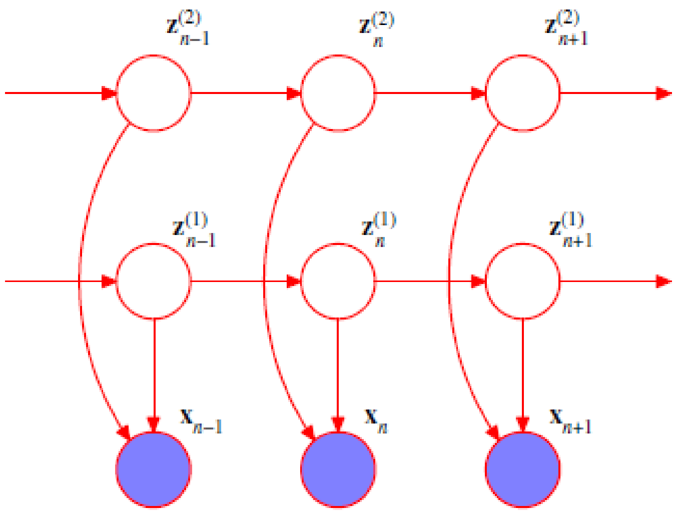
  => Natural way to use prior knowledge

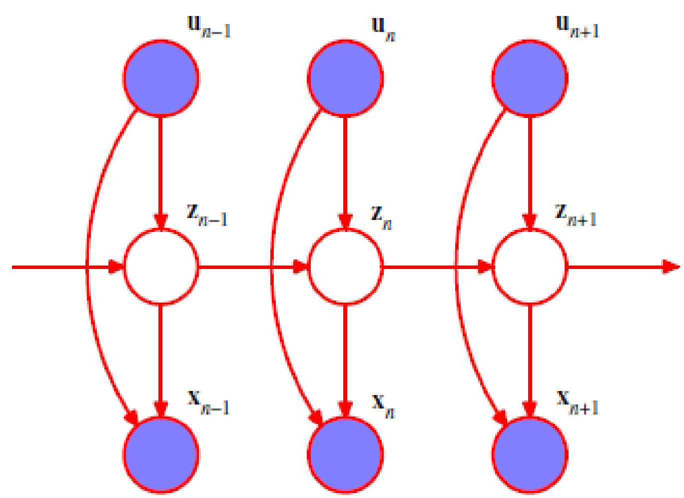
- Countless ML models are actually Bayes nets:
  - Generative models (PCA, ICA), probabilistic classification/regression models (neural nets, etc.), LDA, nonlinear dynamical systems, etc

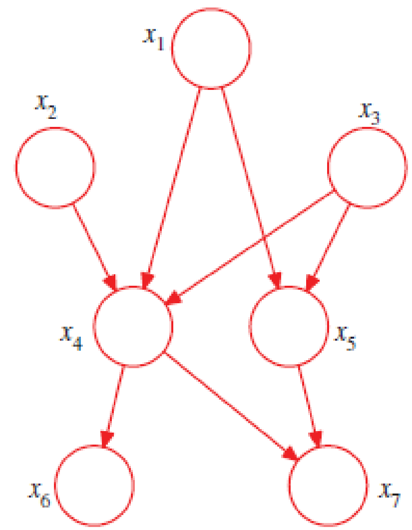
- Inference fast in general case **but slow in general case**

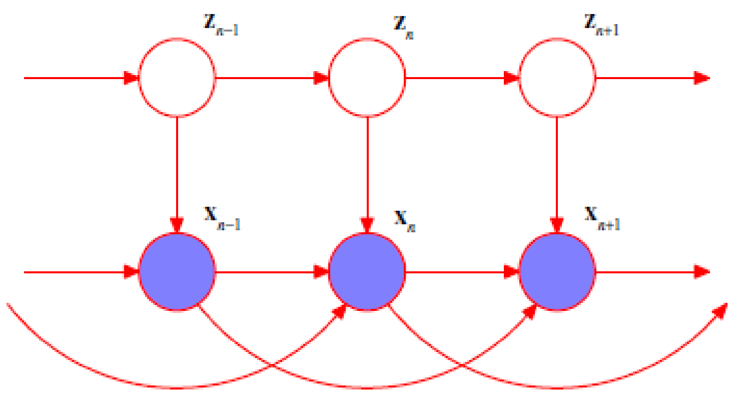
Bayesian networks with cont. latent variables

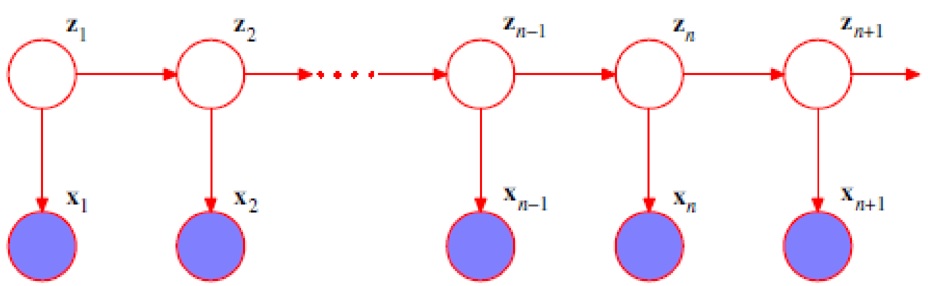












Reasonably fast algorithm to train them all?

Inference in Bayes nets (with continuous latent variables)

- Joint p.d.f.:

$$p(\mathbf{x}, \mathbf{z}) = \prod_j p_{\boldsymbol{\theta}}(\mathbf{x}_j | \mathbf{pa}_j) \prod_j p_{\boldsymbol{\theta}}(\mathbf{z}_j | \mathbf{pa}_j)$$

$$\log p(\mathbf{x}, \mathbf{z}) = \sum_j \log p_{\boldsymbol{\theta}}(\mathbf{x}_j | \mathbf{pa}_j) + \sum_j \log p_{\boldsymbol{\theta}}(\mathbf{z}_j | \mathbf{pa}_j)$$

- We' re going to use Hybrid Monte Carlo (HMC)
  - A method for sampling *z*|*x* using first-order gradients of log p(*z*|*x*)
- Gradients of p(z|x) are easy:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) / p_{\boldsymbol{\theta}}(\mathbf{x})$$

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) \propto p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$$

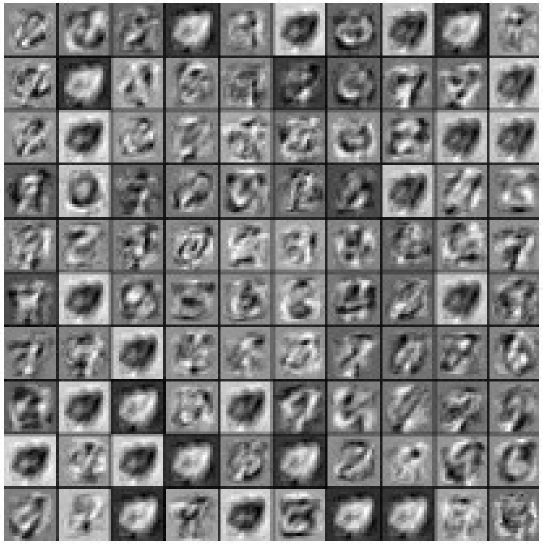
$$\nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \nabla_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$$

Learning in any Bayes net with continuous variables

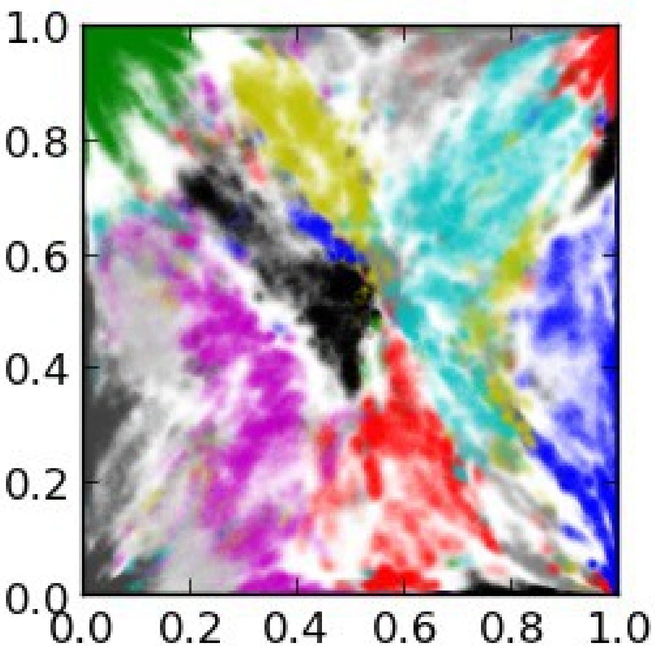
- MCMC-EM
  - E-step: generate samples $\mathbf{z}^l \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$
    - with gradient-based sampler, e.g. HMC
  - M-step: maximize w.r.t. $\boldsymbol{\theta}$:

    $$\mathbb{E}_{\mathbf{z}|\mathbf{x}}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})\right] \approx \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}^l) \text{ with } \mathbf{z}^l \sim p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$$

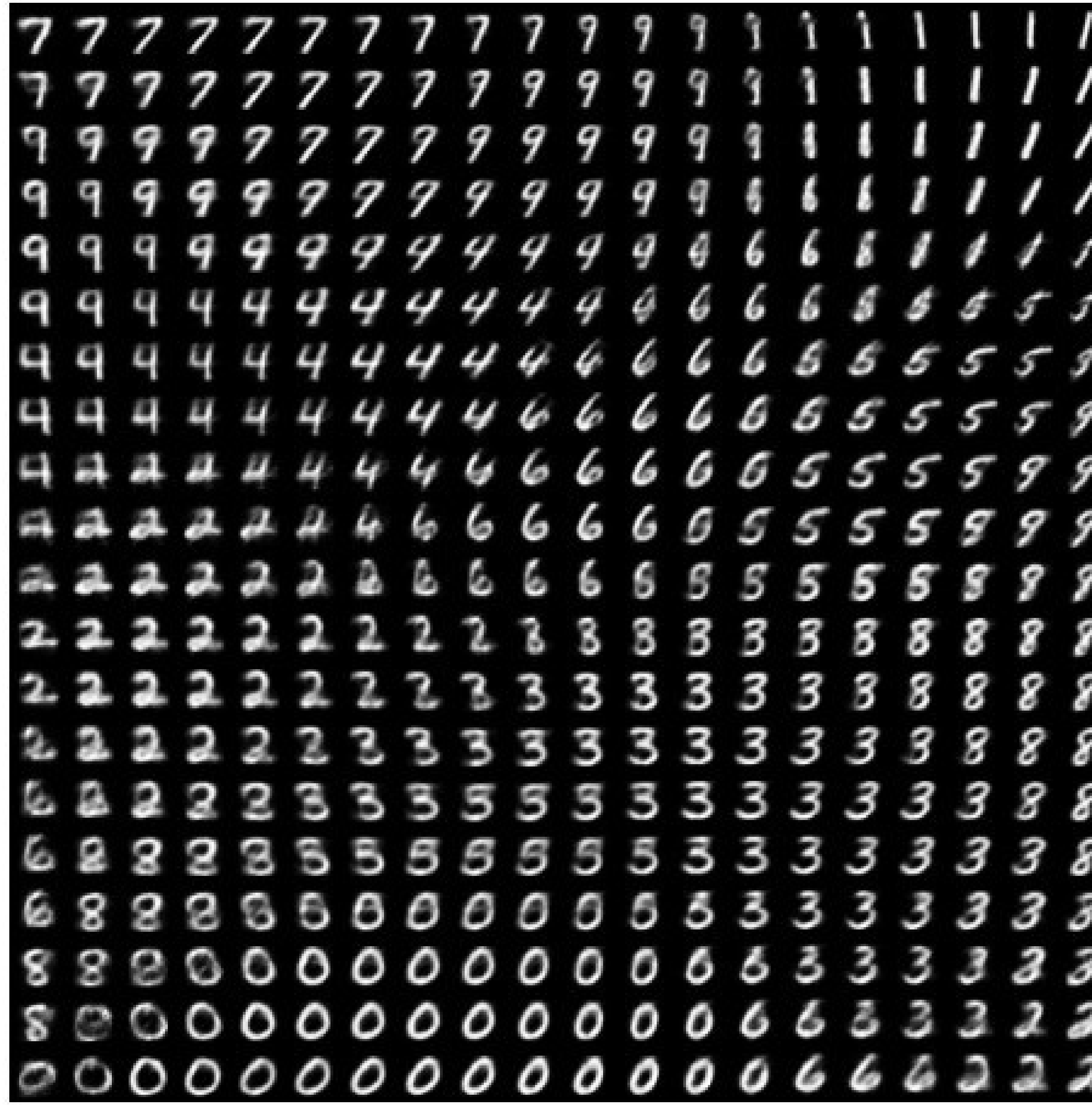
    with gradient descent, e.g. Adagrad



Basis functions



z ~ p(z|x)

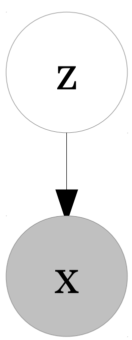


2-D Latent space (transformed through square using inverse CDF of gaussian), projected to observed space

Example: simple generative MNIST model

$\mathbf{z} =$ 2-D continuous latent variable

$\mathbf{x} =$ 768-D multivariate binary variable



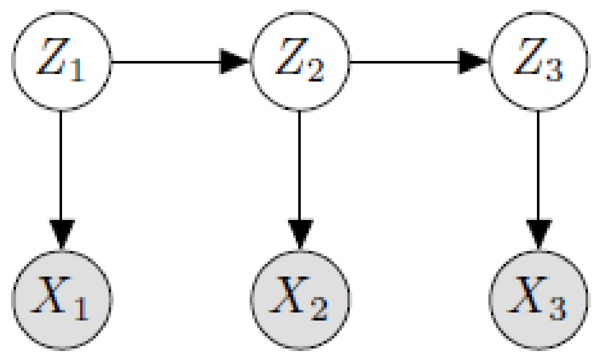
$p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(0, I)$
$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) =$ 2-layer feed-forward NN + binary softmax

Data: MNIST digits (no labels)

Training: EM using HMC and Adagrad

Problem: HMC mixes slowly for deep or recurrent Bayes nets

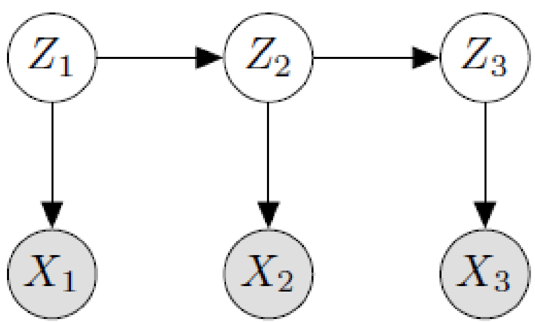
$$\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}_1) + \sum_{j=1}^{3} \log p_{\boldsymbol{\theta}}(\mathbf{x}_j|\mathbf{z}_j) + \sum_{j=1}^{2} \log p_{\boldsymbol{\theta}}(\mathbf{z}_{j+1}|\mathbf{z}_j)$$

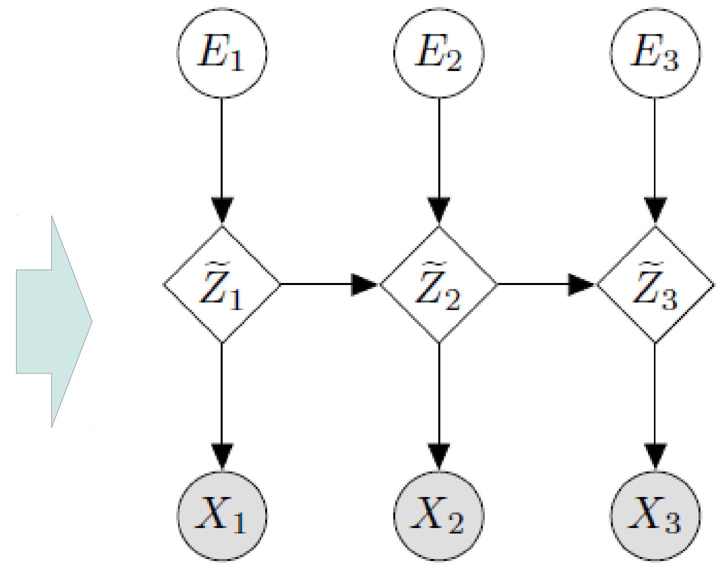


Momentum updates for each $\mathbf{z}_j$ using first-order gradients
so: each $\mathbf{z}_j$'s new values are only affected by variables in it's Markov blanket

e.g. updates of $\mathbf{z}_1$ only indirectly influenced by $\mathbf{x}_2$ and $\mathbf{x}_3$!
$\nabla_{\mathbf{z}_1} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = \nabla_{\mathbf{z}_1} \log p_{\boldsymbol{\theta}}(\mathbf{x}_1|\mathbf{z}_1) + \nabla_{\mathbf{z}_1} \log p_{\boldsymbol{\theta}}(\mathbf{z}_2|\mathbf{z}_1)$

Auxiliary form: equivalent neural net with injected noise





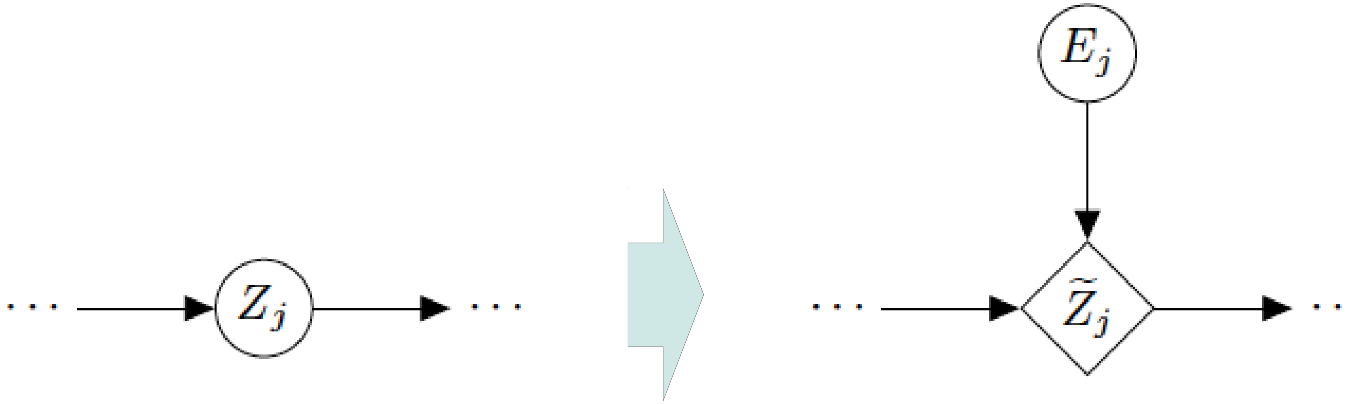
$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \widetilde{\mathbf{z}}, \boldsymbol{\epsilon})$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}, \widetilde{\mathbf{z}}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \widetilde{\mathbf{z}}, \boldsymbol{\epsilon})\, d\boldsymbol{\epsilon}$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \widetilde{\mathbf{z}}, \boldsymbol{\epsilon})\, d\widetilde{\mathbf{z}}$$

where $p_{\boldsymbol{\theta}}(\mathbf{x}, \widetilde{\mathbf{z}})$ is equal in distribution to $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$

Advantage: inference using $p_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})$ is *lightning fast*

Auxiliary form



$p_{\boldsymbol{\theta}}(\mathbf{z}_j|\mathbf{pa}_j)$

Choose $p(\boldsymbol{\epsilon}_j)$ and $g_j(.)$
where $\widetilde{\mathbf{z}}_j = g_j(\mathbf{pa}_j, \boldsymbol{\epsilon}_j, \boldsymbol{\theta})$

such that: $p_{\boldsymbol{\theta}}(\mathbf{z}_j|\mathbf{pa}_j)$ is equal in distribution to $p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j)$
$= \mathbb{E}_{\boldsymbol{\epsilon}}\left[p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j, \boldsymbol{\epsilon}_j)\right]$

using: $p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j, \boldsymbol{\epsilon}_j) = \delta(\widetilde{\mathbf{z}}_j - g(.))$

e.g.: $\boldsymbol{\epsilon}_j \sim \mathcal{U}(0,1)$ and $g(.) =$ inverse CDF

Example: Gaussian conditional with nonlinear dependency of mean

Original form:

$p_{\boldsymbol{\theta}}(\mathbf{z}_j|\mathbf{pa}_j) = \mathcal{N}(\mathbf{z}_j; f_j(\mathbf{pa}, \boldsymbol{\theta}), \sigma^2 I)$

Auxiliary form:

$\boldsymbol{\epsilon}_j \sim \mathcal{N}(0, I)$

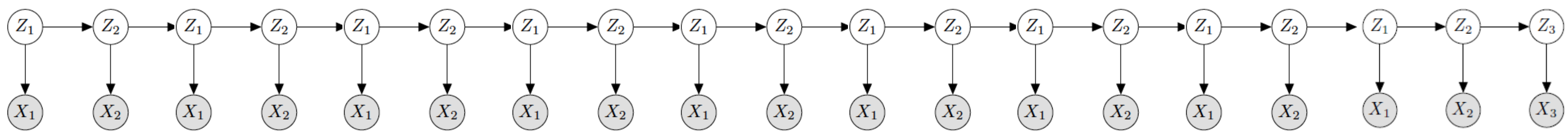
$\widetilde{\mathbf{z}}_j = g_j(.) = f_j(\mathbf{pa}_j, \boldsymbol{\theta}) + \sigma\boldsymbol{\epsilon}_j$

Valid since:

$p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j, \boldsymbol{\epsilon}_j) = \delta(\widetilde{\mathbf{z}}_j - (f(\mathbf{pa}_j, \boldsymbol{\theta}) + \sigma\boldsymbol{\epsilon}_j))$

$p_{\boldsymbol{\theta}}(\mathbf{z}_j|\mathbf{pa}_j) = p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j) = \mathbb{E}_{\boldsymbol{\epsilon}_j}\left[p_{\boldsymbol{\theta}}(\widetilde{\mathbf{z}}_j|\mathbf{pa}_j, \boldsymbol{\epsilon}_j)\right]$

Experiment: DBN



DBN with $j \in \{1 \ldots 20\}$

$\forall j : \mathbf{z}_j \in \mathcal{R}^{10}$
$\forall j : \mathbf{x}_j \in \mathcal{R}^{1}$

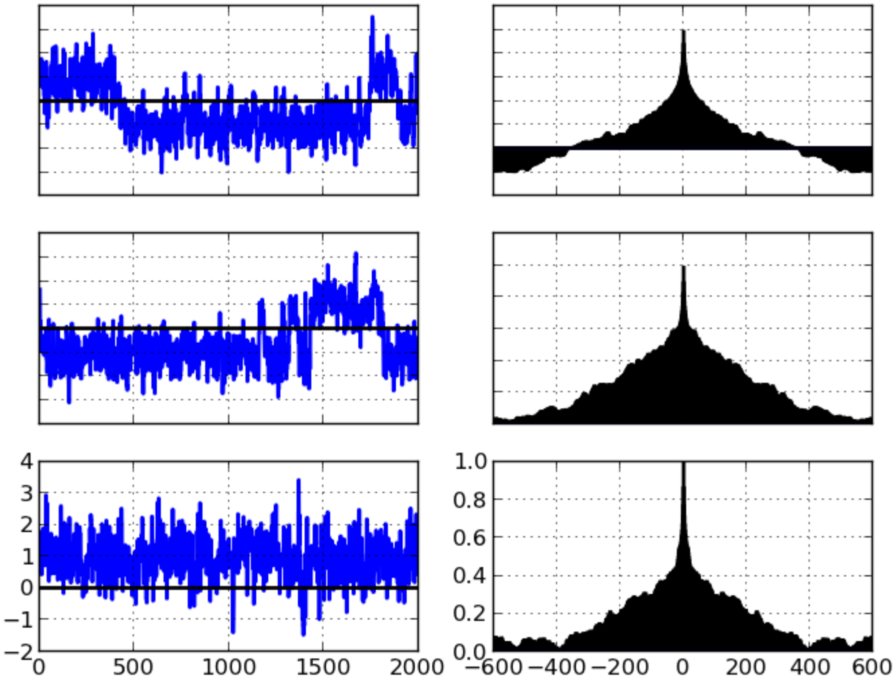
$$\mathrm{p}_{\boldsymbol{\theta}}(X_{j,i} = 1 | \mathbf{z}_j) = sigmoid(\mathbf{w}_i^T \mathbf{z}_j + b_i)$$

$$p_{\boldsymbol{\theta}}(\mathbf{z}_{j+1} | \mathbf{z}_j) = \mathcal{N}(tanh(W_z \mathbf{z}_j + \mathbf{b_j}), \sigma_{\mathbf{z}} \mathbf{I})$$

Experiment: DBN

Original form

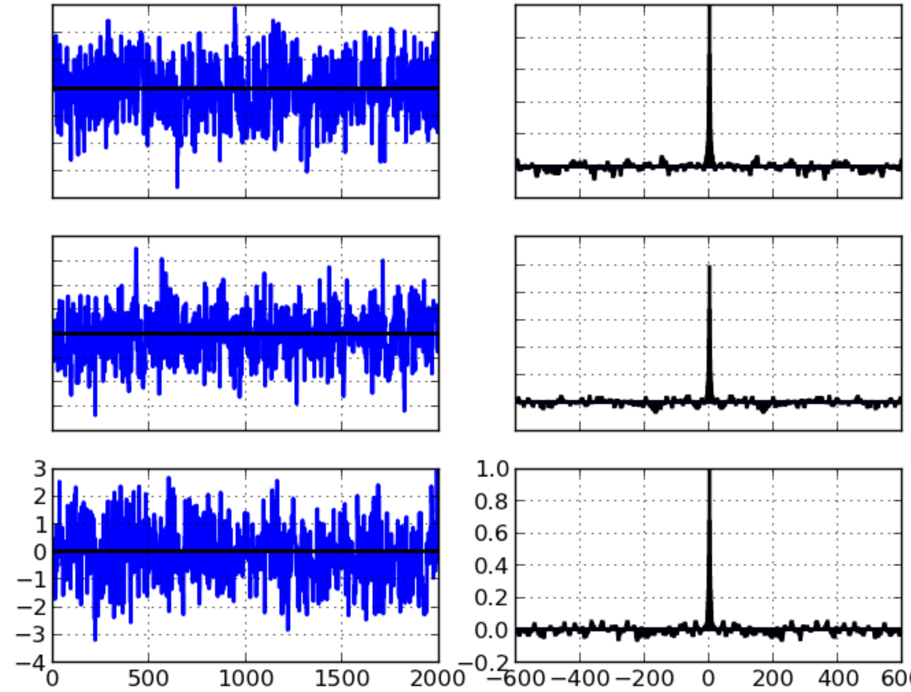
Samples   Autocorrelation



Terribly slow mixing

Auxiliary form

Samples   Autocorrelation



Fast mixing

Extra: EM-free Learning

- In Auxiliary form: all latent random variables are root nodes
- MC Likelihood:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}}\left[p(\mathbf{x}|\mathbf{z}\right] \approx \frac{1}{L}\sum_{l=1}^{L} p(\mathbf{x}|\mathbf{z}^l) \, \mathrm{with} \, \mathbf{z}^l \sim (\mathbf{z})$$

  - Only works if all z's are root nodes of the graph and p(z) is not affected by 'w'
  - Dropout: MC likelihood with L=1

Conclusion: auxiliary form

- Auxiliary form:
  - Fast mixing => Fast inference and learning
- Very broadly applicable
  (e.g. inverse CDF method)
- http://arxiv.org/abs/1306.0733
  Link at: www.dpkingma.com
- Next step: applications!