Day 5

Deep Generative Models

Jun Zhu
dcszj@mail.tsinghua.edu.cn
Department of Computer Science and Technology
Tsinghua University

Tsinghua Deep Learning Summer School, Beijing, 2019

A bit about the Instructor

- Jun Zhu, Professor, Depart. of Computer Science & Technology. I received my Ph.D. in DCST of Tsinghua University in 2009. My research interests include statistical machine learning, Bayesian nonparametrics, and data mining
- I did post-doc at the Machine Learning Department in CMU with Prof. Eric P. Xing. Before that I was invited to visit CMU for twice. I was also invited to visit Stanford for joint research (with Prof. Li Fei-Fei)
- 2015: Adjunct Professor at CMU

- Published 100+ research papers on the top-tier ML conferences and journals, including JMLR, TPAMI, ICML, NIPS, etc.
- Served as Area Chairs for ICML, NIPS, UAI, AAAI, IJCAI; Associate Editor for PAMI, AI Journal
- Research is supported by National 973, NSFC, "Tsinghua 221 Basic Research Plan for Young Talents".
- Homepage: http://ml.cs.tsinghua.edu.cn/~jun

Outline

- Review of Probability and Statistics
  - MLE
- Generative Models
- Deep Generative Models
  - VAE
  - GAN
- ZhuSuan
  - Programming library
- Applications

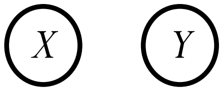
Basics of Probabilities and MLE

Independence

- Independent random variables:

$$P(X, Y) = P(X)P(Y)$$

$$P(X|Y) = P(X)$$



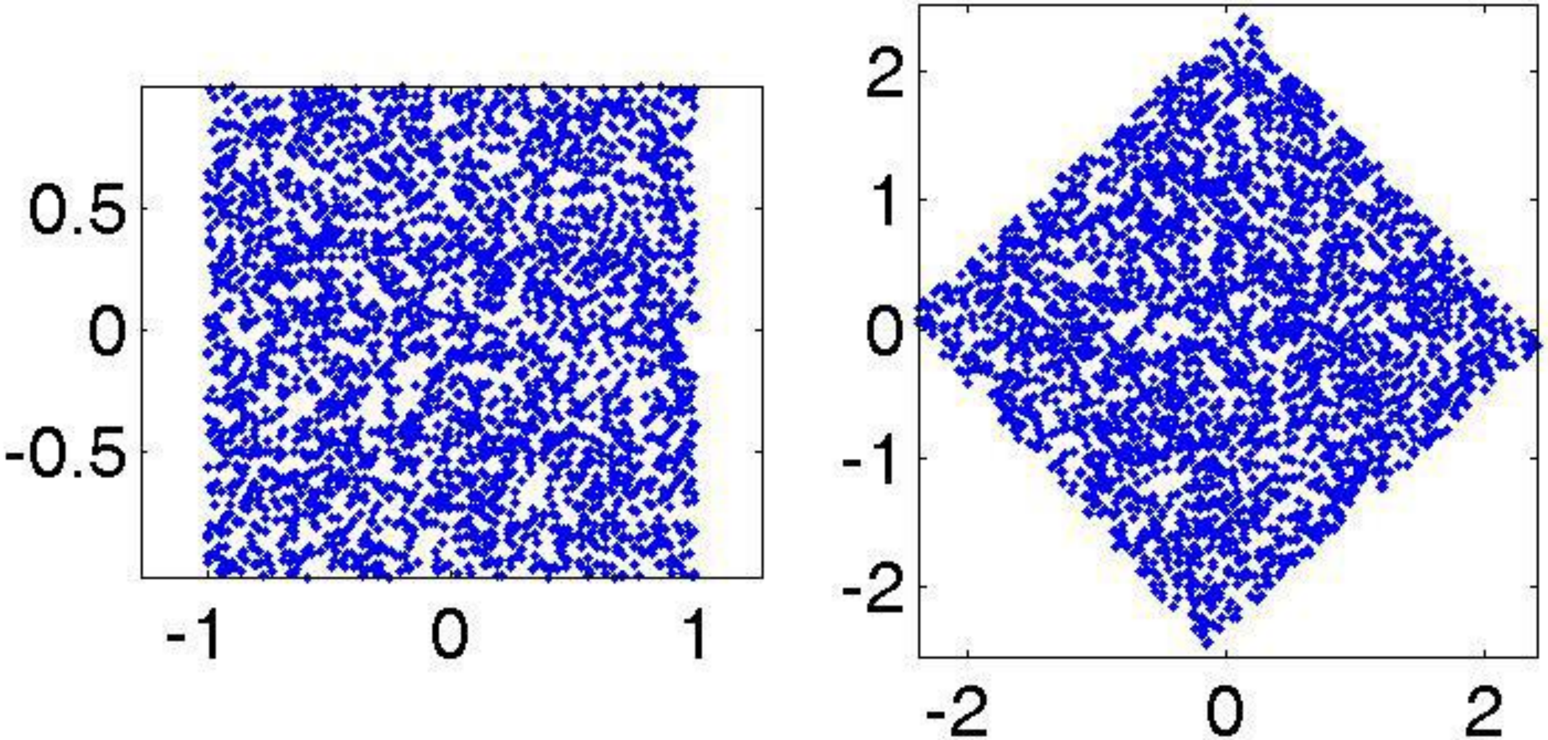
  - Y and X don't contain information about each other

    Observing Y doesn't help predicting X

    Observing X doesn't help predicting Y

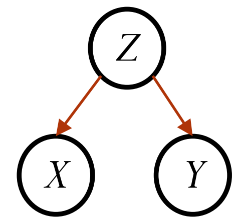
- Examples:
  - Independent:
    - winning on roulette this week and next week
  - Dependent:
    - Russian roulette

Dependent / Independent?



Conditional Independence

- Conditionally independent:



$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$

  - knowing Z makes X and Y independent

- Examples:

London taxi drivers: A survey has pointed out a positive and significant correlation between the number of accidents and wearing coats. They concluded that coats could hinder movements of drivers and be the cause of accidents. A new law was prepared to prohibit drivers from wearing coats when driving.

Finally another study pointed out that people wear coats when it rains…

Maximum Likelihood Estimation (MLE)

Flipping a Coin

- What's the probability that a coin will fall with a head up (if flipped)?
- Let us flip it a few times to estimate the probability

The estimated probability is: 3/5 "frequency of heads"

Questions:

The estimated probability is: 3/5 "frequency of heads"

- Why frequency of heads?
- How good is this estimation?

Question (1)

- Why frequency of heads?
  - Frequency of heads is exactly the Maximum Likelihood Estimator for this problem

  - MLE has nice properties

    (interpretation, statistical guarantees, simple)

MLE for Bernoulli Distribution

Data, D =

$$D = \{X_i\}_{i=1}^n, \; X_i \in \{\mathrm{H}, \mathrm{T}\}$$

$$P(Head) = \theta \qquad P(Tail) = 1 - \theta$$

- Flips are i.i.d:
  - **Independent** events that are **identically distributed** according to Bernoulli distribution
- **MLE**: choose θ that maximizes the probability of observed data

Maximum Likelihood Estimation (MLE)

- MLE: choose θ that maximizes the probability of observed data

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

$$= \arg\max_{\theta} \prod_{i=1}^{n} P(X_i|\theta)$$ Independent draws

$$= \arg\max_{\theta} \prod_{i:X_i=H} \theta \prod_{i:X_i=T} (1-\theta)$$ Identically distributed

$$= \arg\max_{\theta} \theta^{N_H} (1-\theta)^{N_T}$$

Maximum Likelihood Estimation (MLE)

- MLE: choose θ that maximizes the probability of observed data

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

$$= \arg\max_{\theta} \theta^{N_H}(1-\theta)^{N_T}$$

- Solution?

$$\hat{\theta}_{MLE} = \frac{N_H}{N_H + N_T}$$

  - Exactly the "**Frequency of heads**"

Question (2)

- How good is the MLE estimation?

$$\hat{\theta}_{MLE} = \frac{N_H}{N_H + N_T}$$

  - Is it biased?

How many flips do I need?

- I flipped the coins 5 times: 3 heads, 2 tails

$$\hat{\theta}_{MLE} = \frac{3}{5}$$

- What if I flipped 30 heads and 20 tails?

$$\hat{\theta}_{MLE} = \frac{30}{50}$$

- Which estimator should we trust more?

A Simple Bound

- Let $\theta^\star$ be the true parameter. For n data points, and

$$\hat{\theta}_{MLE} = \frac{N_H}{N_H + N_T}$$

- Then, for any ε>0 , we have the Hoeffding's Inequality:

$$P(|\hat{\theta} - \theta^\star| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$$

Probably Approximately Correct (PAC) Learning

- I want to know the coin parameter θ, within ε=0.1 error with probability at least 1-δ (e.g., 0.95)
- How many flips do I need?

$$P(|\hat{\theta} - \theta^{\star}| \geq \epsilon) \leq 2e^{-2n\epsilon^2} \leq \delta$$

- Sample complexity:

$$n \geq \frac{\ln(2/\delta)}{2\epsilon^2}$$

Deep Generative Models (DGMs)

Why generative models?

"What I cannot create, I do not understand."

—Richard Feynman

Why generative models?

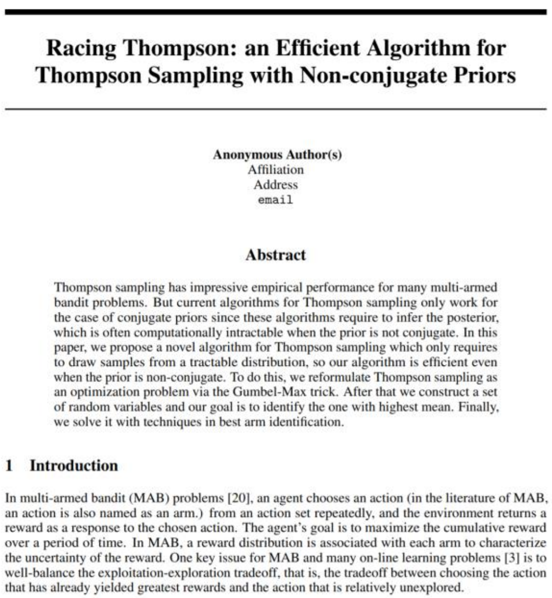
- Leverage unlabeled datasets, which are often much larger than labeled ones
  - Unsupervised learning
  - Semi-supervised learning

- Conditional generative models
  - Speech synthesis: Text ⇒ Speech
  - Machine Translation: French ⇒ English
  - Image captioning: Image ⇒ Text

Generative models are everywhere …

- A simple unigram language model
  - Observations (e.g., bag-of-words)

$$\mathrm{x} = \{x_1, \ldots, x_d\}$$



Racing Thompson: an Efficient Algorithm for Thompson Sampling with Non-conjugate Priors

Anonymous Author(s)
Affiliation
Address
email

Abstract

Thompson sampling has impressive empirical performance for many multi-armed bandit problems. But current algorithms for Thompson sampling only work for the case of conjugate priors since these algorithms require to infer the posterior, which is often computationally intractable when the prior is not conjugate. In this paper, we propose a novel algorithm for Thompson sampling which only requires to draw samples from a tractable distribution, so our algorithm is efficient even when the prior is non-conjugate. To do this, we reformulate Thompson sampling as an optimization problem via the Gumbel-Max trick. After that we construct a set of random variables and our goal is to identify the one with highest mean. Finally, we solve it with techniques in best arm identification.

1 Introduction

In multi-armed bandit (MAB) problems [20], an agent chooses an action (in the literature of MAB, an action is also named as an arm.) from an action set repeatedly, and the environment returns a reward as a response to the chosen action. The agent's goal is to maximize the cumulative reward over a period of time. In MAB, a reward distribution is associated with each arm to characterize the uncertainty of the reward. One key issue for MAB and many on-line learning problems [3] is to well-balance the exploitation-exploration tradeoff, that is, the tradeoff between choosing the action that has already yielded greatest rewards and the action that is relatively unexplored.

Term	D1	D2
game	1	0
decision	0	0
theory	2	0
probability	0	3
analysis	0	2
…		

Generative models are everywhere …

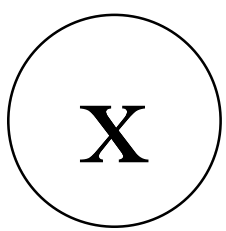
- A simple unigram language model
  - Observations (e.g., bag-of-words)

$$\mathrm{x} = \{x_1, \dots, x_d\}$$

  - Joint distribution (likelihood)

$$p(\mathrm{x};\theta) = \prod p(x_i\,;\theta)$$

  - Graphical representation (parameters ignored)



Generative models are everywhere …

- Learn a simple generative model
  - Given a set of observations

$$\mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_N\}$$

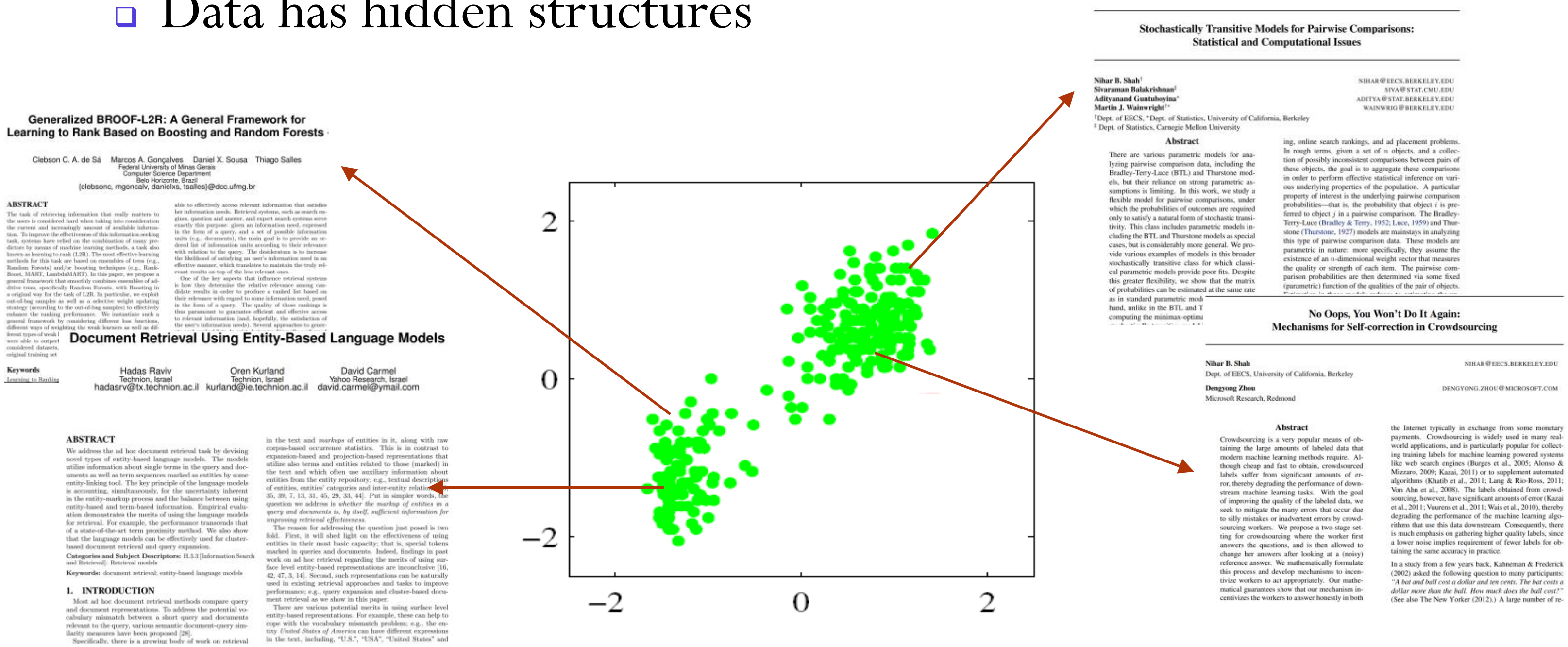
  - Maximize the log-likelihood

$$\max_{\theta} \log p(\mathrm{X}; \theta) = \sum \log p(\mathrm{x}_\mathrm{i}\, ; \theta)$$

  - Simple closed-form solutions:
    - count frequency for discrete or empirical mean/variance for Gaussian distribution

Generative models are everywhere …

- A fully-observed model is not sufficient
  - Data has hidden structures



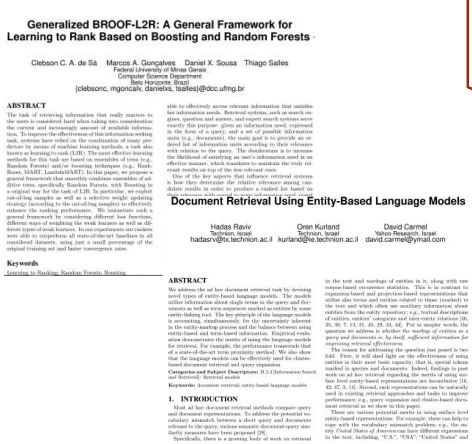
  - A simple distribution is not sufficient

Generative models are everywhere …

- Mixture model --- a simple generative model with hidden factors
  - Separate the data into different groups

T1=“SIGIR”

retrieval 0.05
text 0.02
learning 0.01
sports 0.01
…



T2=“ICML”

learning 0.03
theory 0.02
algorithm 0.01
deep 0.01
…

Generative models are everywhere …

- Mixture model --- a simple generative model with hidden factors
  - Graphical model representation

Generalized BROOF-L2R: A General Framework for Learning to Rank Based on Boosting and Random Forests

Clebson C. A. de Sá, Marcos A. Gonçalves, Daniel X. Sousa, Thiago Salles
Federal University of Minas Gerais
Computer Science Department
Belo Horizonte, Brazil
{clebsonc, mgoncalv, danielxs, tsalles}@dcc.ufmg.br

Document Retrieval Using Entity-Based Language Models

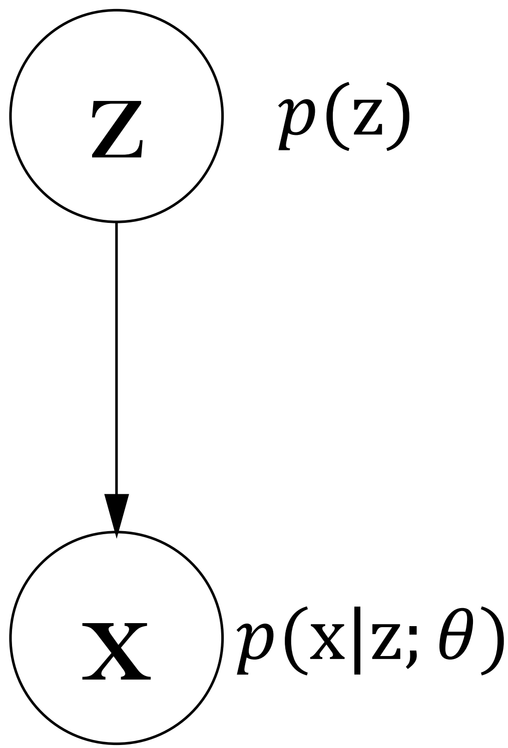
Hadas Raviv, Technion, Israel, hadasrv@tx.technion.ac.il
Oren Kurland, Technion, Israel, kurland@ie.technion.ac.il
David Carmel, Yahoo Research, Israel, david.carmel@ymail.com

Stochastically Transitive Models for Pairwise Comparisons: Statistical and Computational Issues

Nihar B. Shah, Sivaraman Balakrishnan, Adityanand Guntuboyina, Martin J. Wainwright

No Oops, You Won't Do It Again: Mechanisms for Self-correction in Crowdsourcing

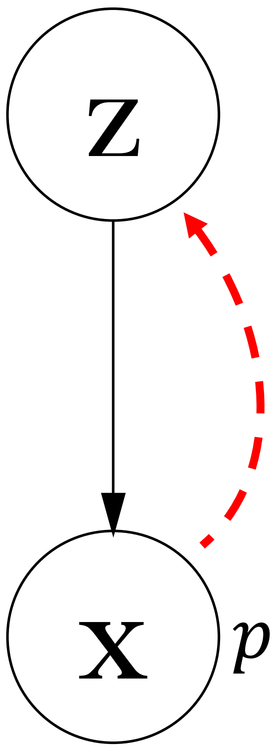
Nihar B. Shah, Dengyong Zhou



$$p(\mathrm{x},\mathrm{z}) = p(\mathrm{z})p(\mathrm{x}|\mathrm{z};\theta)$$

Generative models are everywhere …

- Mixture model --- a simple generative model with hidden factors
  - Infer the latent Z:



$p(\mathrm{z})$

$p(\mathrm{x}|\mathrm{z};\theta)$

Bayes' Rule:

$$p(\mathrm{z}|\mathrm{x}) = \frac{p(\mathrm{x},\mathrm{z})}{p(\mathrm{x})}$$

$$\propto p(\mathrm{z})p(\mathrm{x}|\mathrm{z};\theta)$$

No Oops, You Won't Do It Again: Mechanisms for Self-correction in Crowdsourcing

Nihar B. Shah NIHAR@EECS.BERKELEY.EDU
Dept. of EECS, University of California, Berkeley

Dengyong Zhou DENGYONG.ZHOU@MICROSOFT.COM
Microsoft Research, Redmond

Abstract

Crowdsourcing is a very popular means of obtaining the large amounts of labeled data that modern machine learning methods require. Although cheap and fast to obtain, crowdsourced labels suffer from significant amounts of error, thereby degrading the performance of downstream machine learning tasks. With the goal of improving the quality of the labeled data, we seek to mitigate the many errors that occur due to silly mistakes or inadvertent errors by crowdsourcing workers. We propose a two-stage setting for crowdsourcing where the worker first answers the questions, and is then allowed to change her answers after looking at a (noisy) reference answer. We mathematically formulate this process and develop mechanisms to incentivize workers to act appropriately. Our mathematical guarantees show that our mechanism incentivizes the workers to answer honestly in both

the Internet typically in exchange from some monetary payments. Crowdsourcing is widely used in many real-world applications, and is particularly popular for collecting training labels for machine learning powered systems like web search engines (Burges et al., 2005; Alonso & Mizzaro, 2009; Kazai, 2011) or to supplement automated algorithms (Khatib et al., 2011; Lang & Rio-Ross, 2011; Von Ahn et al., 2008). The labels obtained from crowdsourcing, however, have significant amounts of error (Kazai et al., 2011; Vuurens et al., 2011; Wais et al., 2010), thereby degrading the performance of the machine learning algorithms that use this data downstream. Consequently, there is much emphasis on gathering higher quality labels, since a lower noise implies requirement of fewer labels for obtaining the same accuracy in practice.

In a study from a few years back, Kahneman & Frederick (2002) asked the following question to many participants: *"A bat and ball cost a dollar and ten cents. The bat costs a dollar more than the ball. How much does the ball cost?"* (See also The New Yorker (2012).) A large number of re-

Generative models are everywhere …

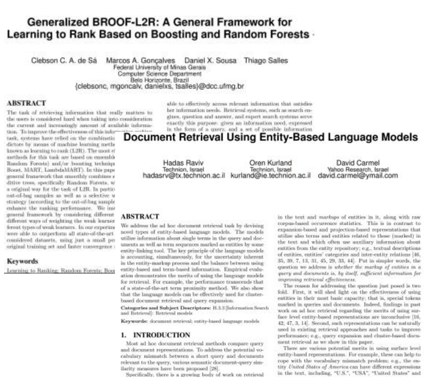
- Mixture model --- a simple generative model with hidden factors
  - EM algorithm to learn the unknown language models

E-step: Infer the hidden Z

M-step: Update the parameters

T1="SIGIR"

T2="ICML"


Generalized BROOF-L2R: A General Framework for Learning to Rank Based on Boosting and Random Forests

Document Retrieval Using Entity-Based Language Models


Stochastically Transitive Models for Pairwise Comparisons: Statistical and Computational Issues

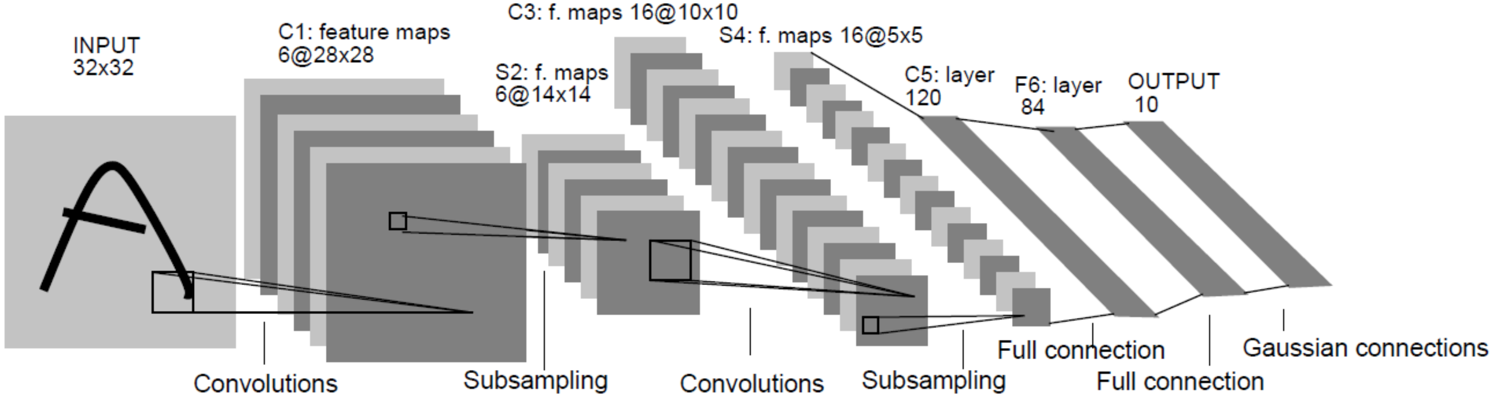
No Oops, You Won't Do It Again: Mechanisms for Self-correction in Crowdsourcing

retrieval	0.05
text	0.02
learning	0.01
sports	0.01
…	

learning	0.03
theory	0.02
algorithm	0.01
deep	0.01
…	

Discriminative Deep Learning

- Learn a deep NN to map an input to output



- Gradient back-propagation
- Dropout
- Activation functions:
  - rectified linear

Generative Modeling

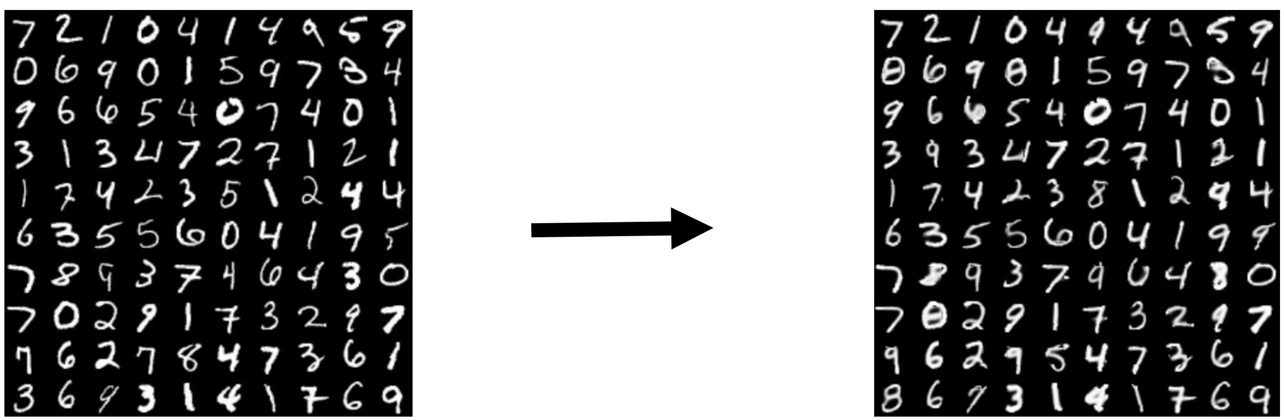
- Have training examples

$$x \sim p_{\text{data}}(x)$$

- Want a model that can draw samples:

$$x' \sim p_{\text{model}}(x)$$

  - where $p_{\text{model}}(x) \approx p_{\text{data}}(x)$



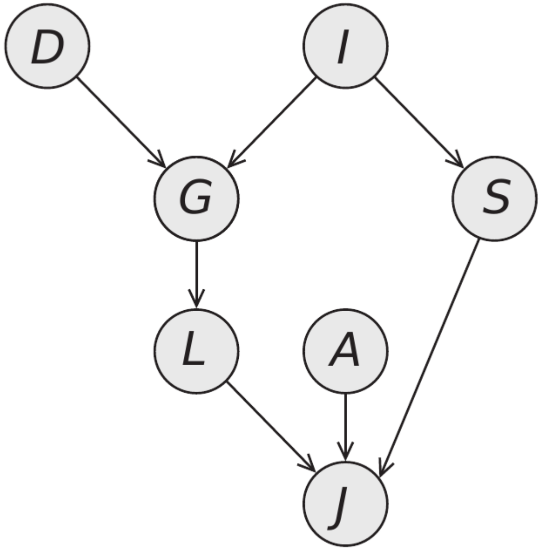
$x \sim p_{\text{data}}(x)$ $\qquad$ $x' \sim p_{\text{model}}(x)$

Two ways to build deep generative models

- **Traditional one**
  - Hierarchical Bayesian methods
- **More modern one**
  - Deep generative models

Hierarchical Bayesian Modeling

- Build a hierarchy throng distributions in analytical forms



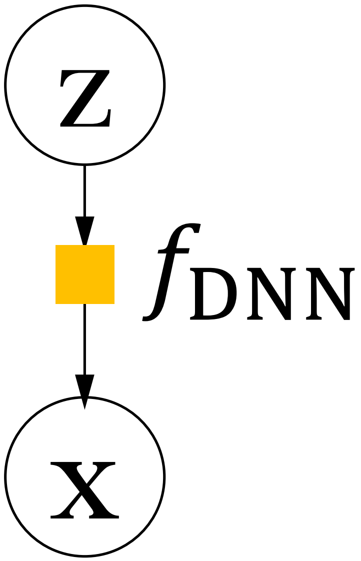
$$P(D, I, G, S, L, A, J) = P(D)P(I)P(G|D, I)P(S|I) \cdot P(A)\,P(L|G)P(J|L, A, S)$$

Simple, Local Factors: a conditional probability distribution

Deep Generative Models

- **More flexible by using differential function mapping between random variables**

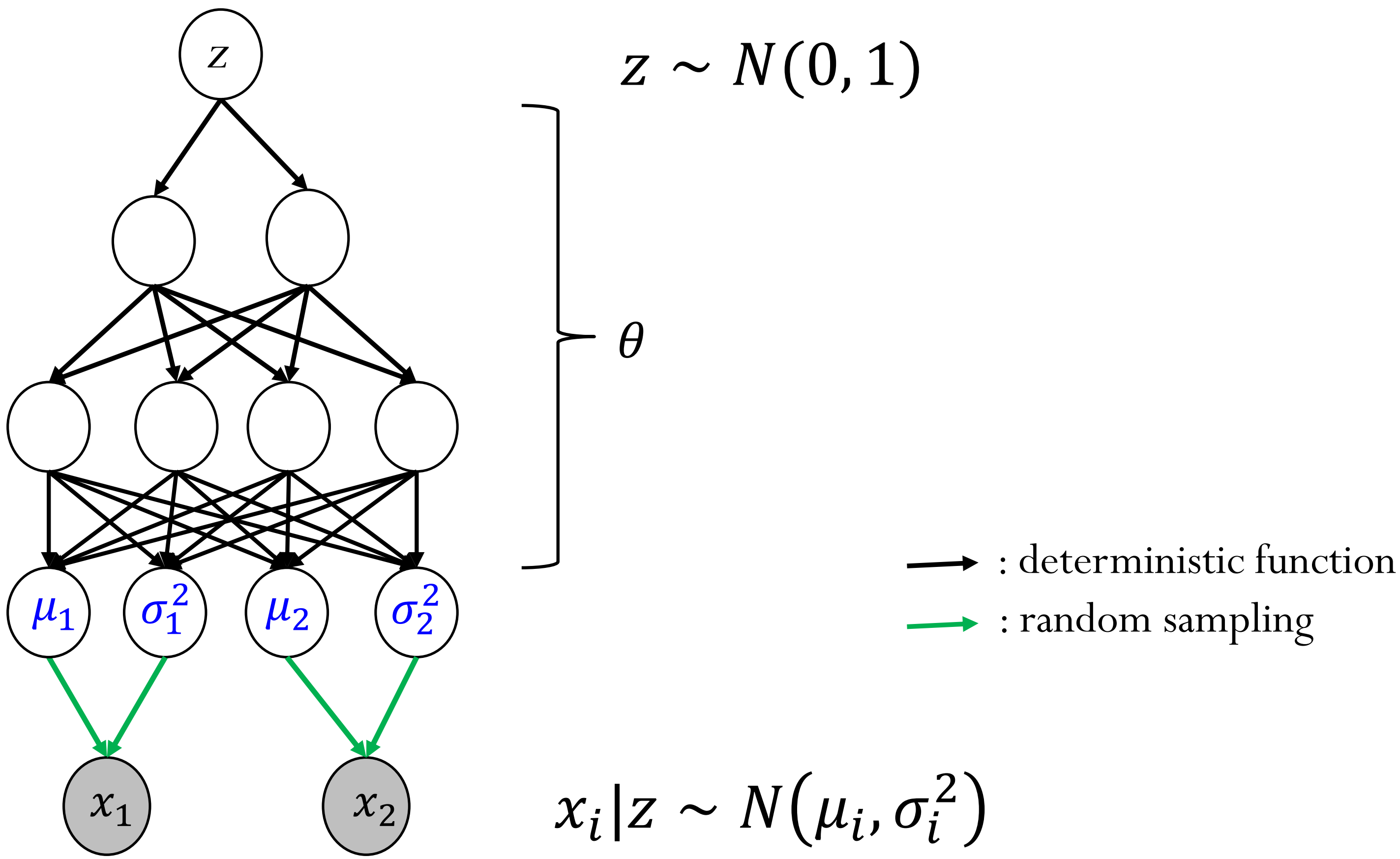
- DGMs learn a function transform with deep neural networks



Cause => Disease
Topics => Docs
Objects => Images
Words => Phonemes

An example with MLP

- 1D latent variable z; 2D observation x
- **Idea**: NN + Gaussian (or Bernoulli) with a diagonal covariance

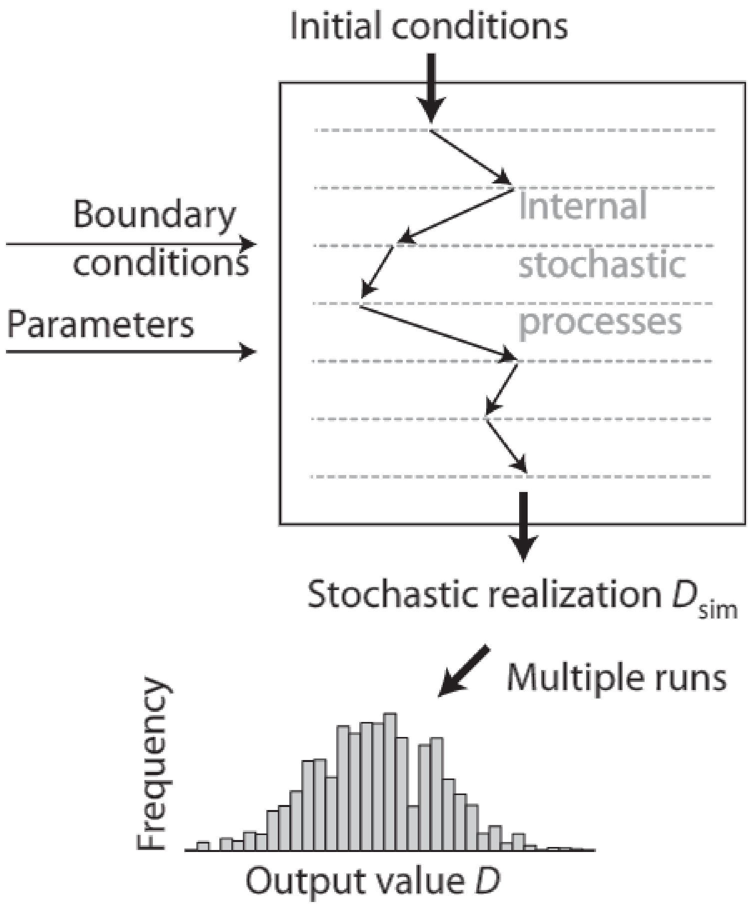


$$z \sim N(0,1)$$

$$x_i|z \sim N\left(\mu_i, \sigma_i^2\right)$$

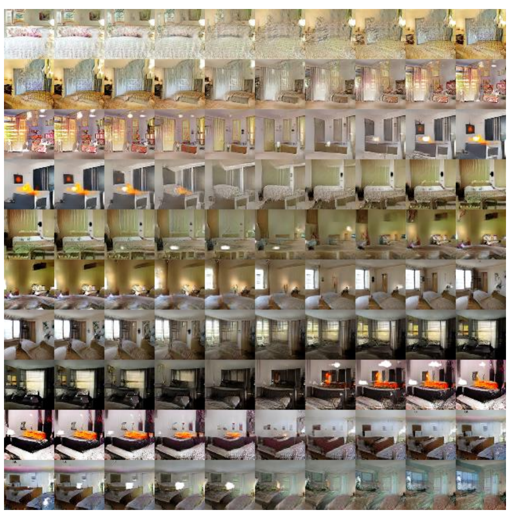
Implicit Deep Generative Models

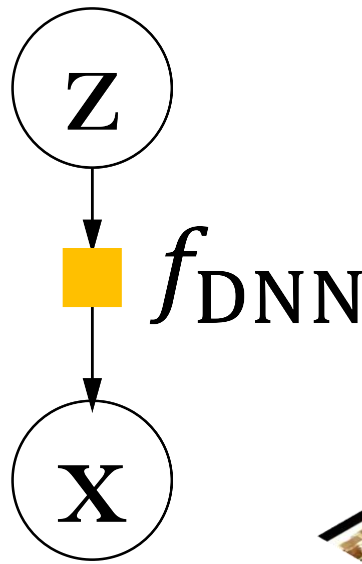
- Generate data with a stochastic process whose likelihood function is not explicitly specified (Hartig et al., 2011)

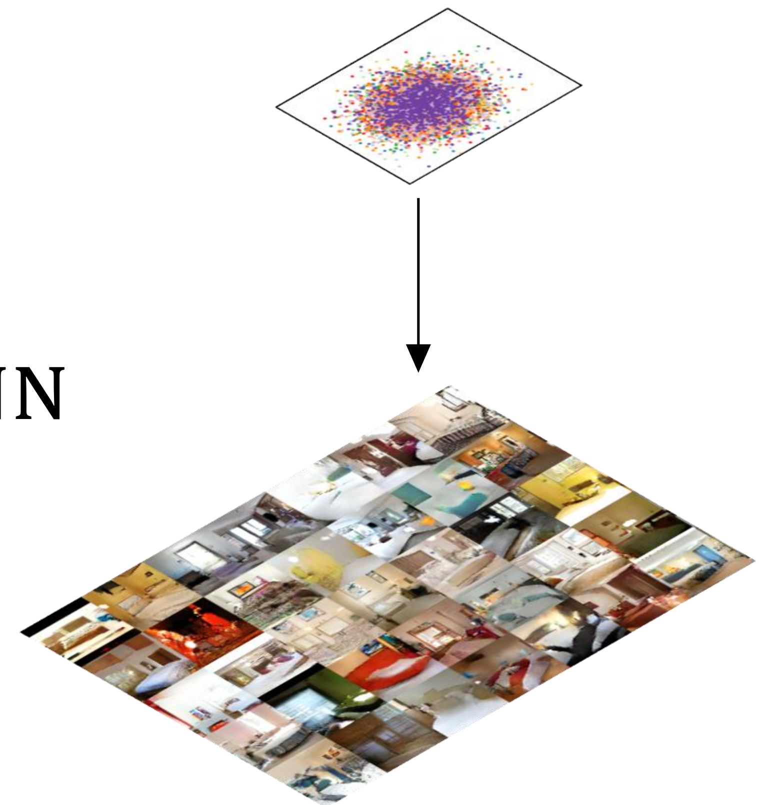


Deep Generative Models

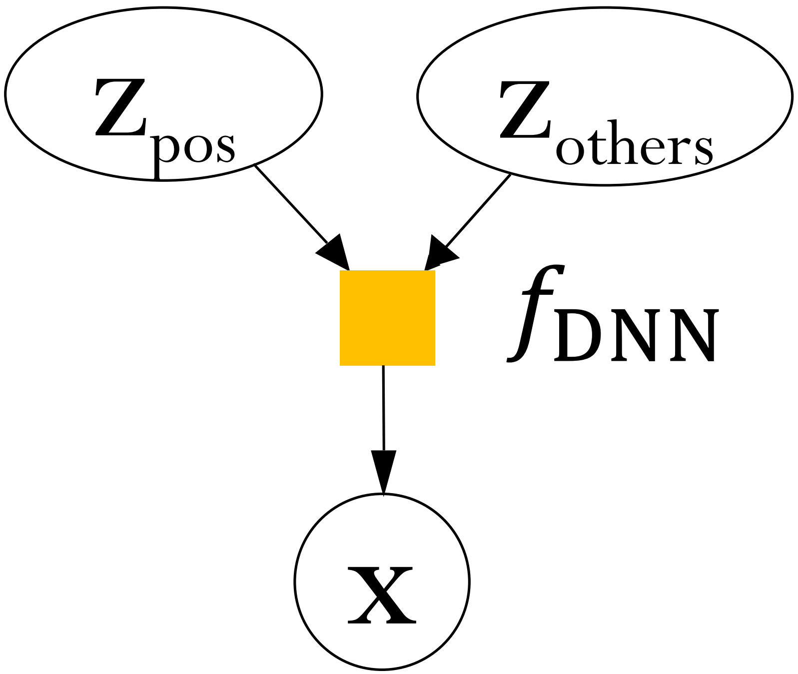
[Image Generation:
Generative Adversarial Nets,
Goodfellow13 & Radford15]

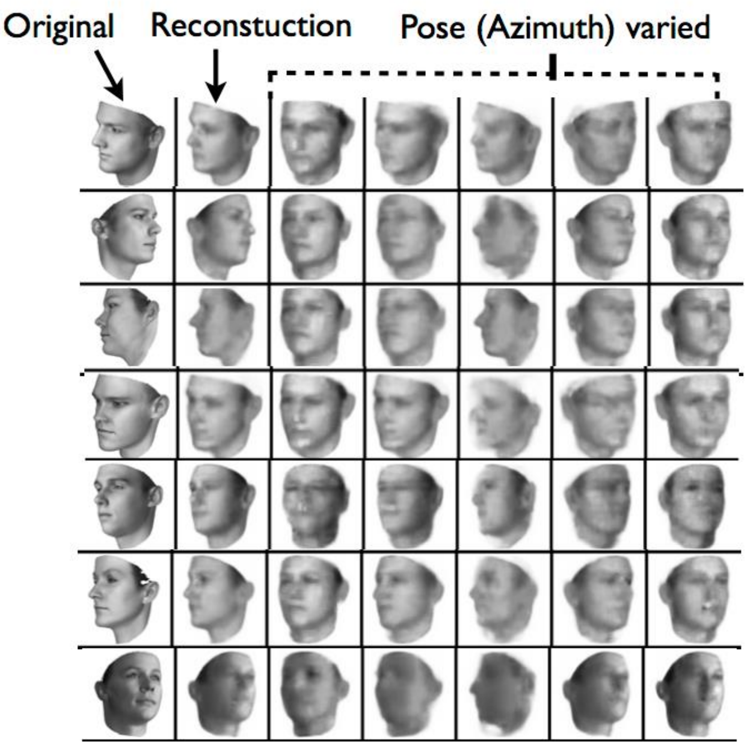






Deep Generative Models



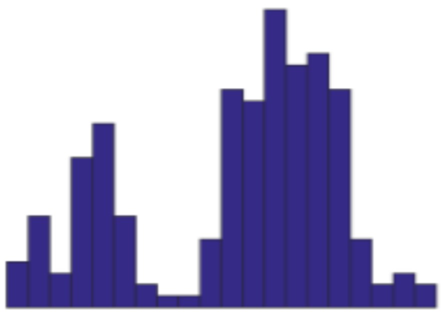


[Image Understanding: Variational Autoencoders, Kingma13 & Tejas15 & Eslami16]

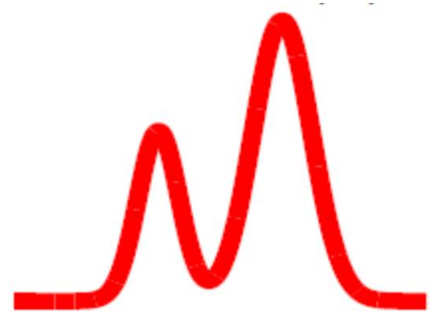
Learning Deep Generative Models

- **Given** a set D of unlabeled samples, learn the unknown parameters (or a distribution)

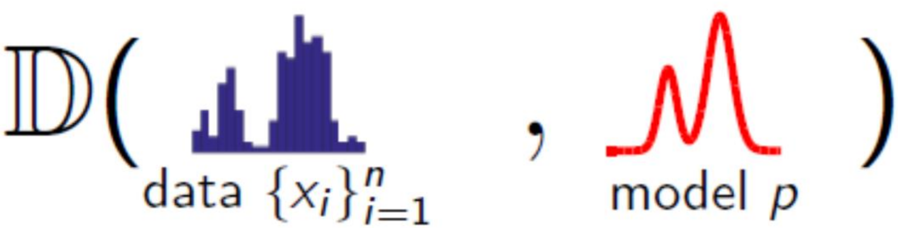
data $D=\{\mathrm{x_i}\}$



models $p(\mathbf{x}|\theta)$



- Find a model that minimizes



Learning Deep Generative Models

- Maximum likelihood estimation (MLE):

$$\hat{\theta} = \operatorname{argmax} p(D|\theta)$$

  - has an explicit likelihood model
- Minimax objective (e.g., GAN)
  - A two-player game to reach equilibrium
  - A three-player game for semi-supervised learning (NIPS'17)
- Moment-matching:
  - draw samples from p: $\widehat{D} = \{\mathrm{y}_i\}_{i=1}^{M}$, where $\mathrm{y}_i \sim p(\mathrm{x}|\theta)$
  - Kernel MMD (NIPS'16):
    - rich enough to distinguish any two distributions in certain RKHS
  - PMD (NIPS'17)

Variational Bayes

- Consider the log-likelihood of ***a single example***

$$\log p(\mathrm{x};\theta) = \log \int p(\mathrm{z},\mathrm{x};\theta) dz$$

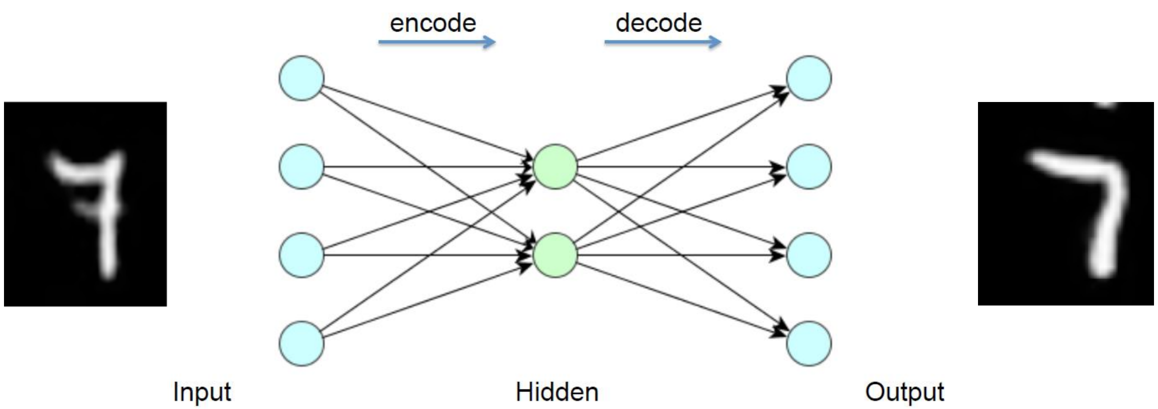
- Log-integral/sum is annoying to handle directly
- Derive a variational lower bound $L(\theta,\phi,\mathrm{x})$

$$\log p(\mathrm{x};\theta) = L(\theta,\phi,\mathrm{x}) + \mathrm{KL}(q(\mathrm{z}|\mathrm{x};\phi)\|p(\mathrm{z}|\mathrm{x};\theta))$$

$$\begin{aligned} L(\theta,\phi,\mathrm{x}) &= \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{z},\mathrm{x};\theta) - \log q(\mathrm{z}|\mathrm{x};\phi)] \\ &= \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|\mathrm{z};\theta) + \log p(\mathrm{z};\theta) - \log q(\mathrm{z}|\mathrm{x};\phi)] \\ &= \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|\mathrm{z};\theta)] - \mathrm{KL}(q(\mathrm{z}|\mathrm{x};\phi)\|p(\mathrm{z};\theta)) \end{aligned}$$

reconstruction term **prior regularization**

Recap: Auto-Encoder

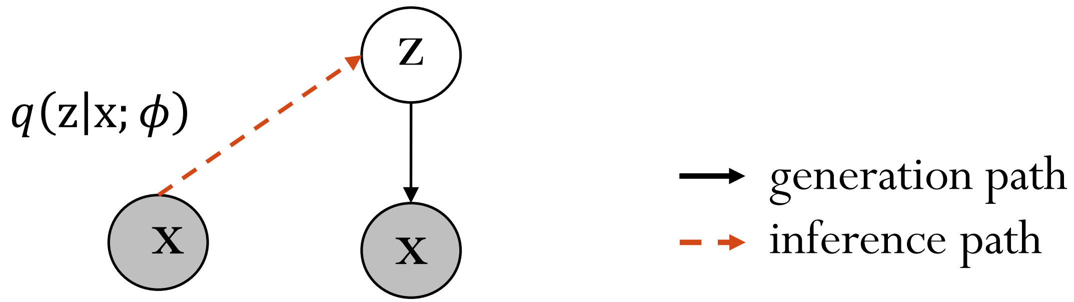


- Encoder: $\mathrm{h} = s(W\mathrm{x} + b)$
- Decoder: $\mathrm{x}' = s(W'\mathrm{h} + b')$
- Training: minimize the reconstruction error (e.g., square loss, cross-entropy loss)
- Denoising AE: randomly corrupted inputs are restored to learn more robust features

Auto-Encoding Variational Bayes (AEVB)

- What's unique in AEVB is that ***the variational distribution is parameterized by a deep neural network***

$$q(\mathrm{z}|\mathrm{x};\phi) \approx p(\mathrm{z}|\mathrm{x};\theta)$$



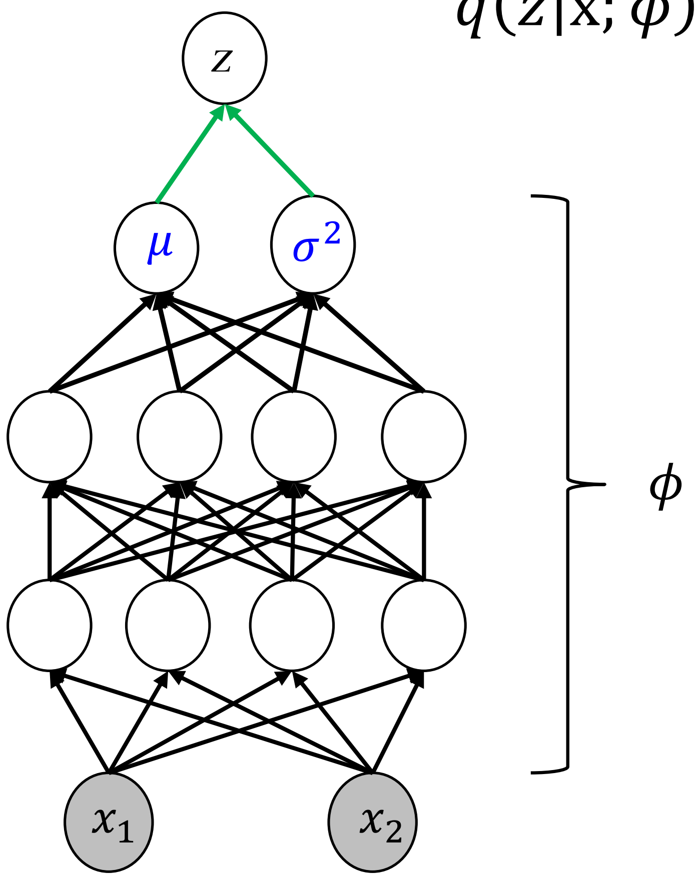
  - We call it an inference (recognition, encoder) network or a Q-network
  - All the parameters are learned jointly via SGD with variance reduction

[Kingma & Welling, ICLR 2013; Rezende et al., ICML 2014]

The Encoder Network

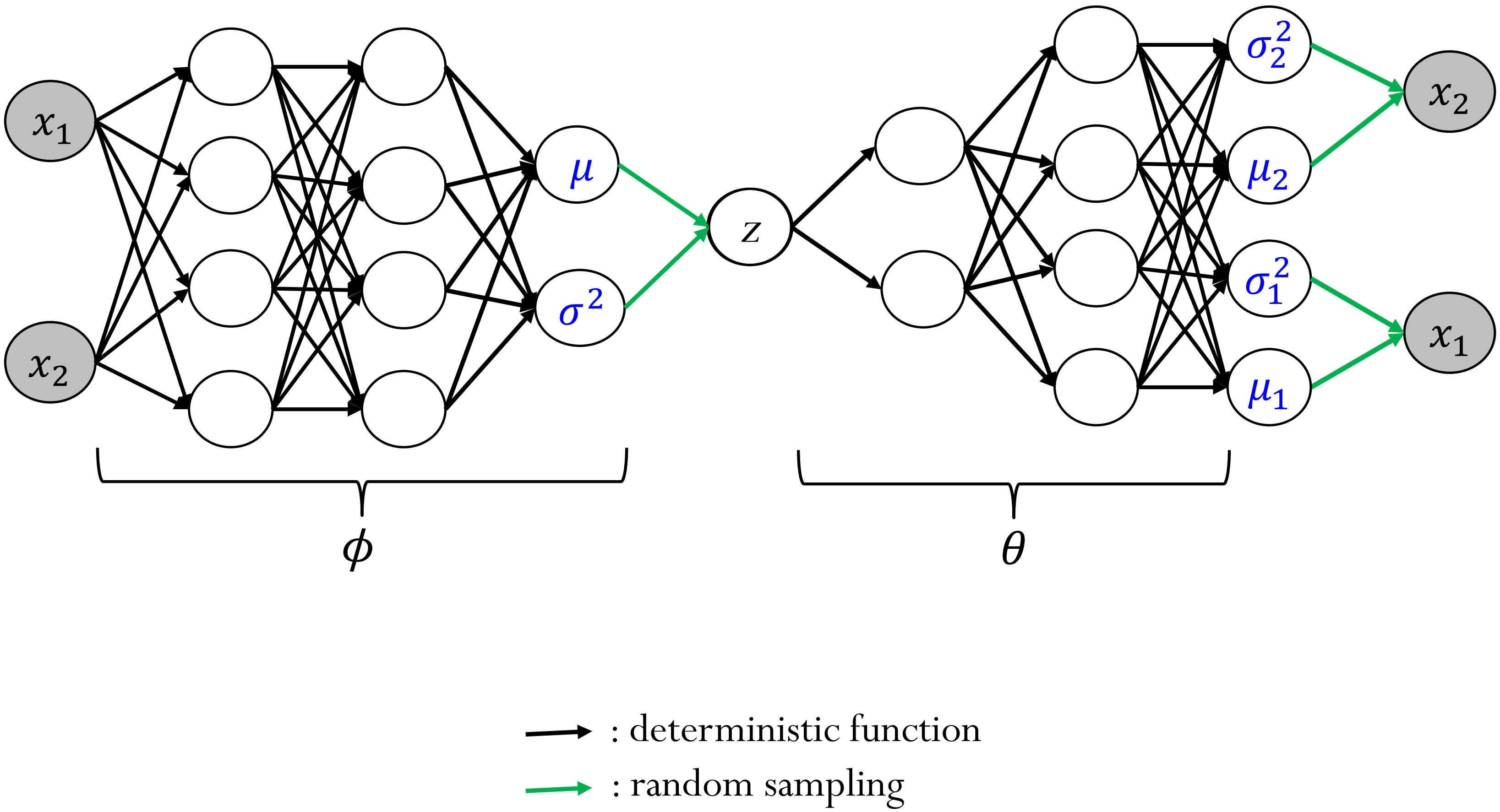
- A feedforward NN + Gaussian

$$q(z|\mathrm{x};\phi) = N(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$$



The Complete Auto-encoder

- The Q-P network architecture:



Stochastic Variational Inference

- Variational lower-bound for ***a single example***

$$L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|\mathrm{z};\theta)] - \mathrm{KL}(q(\mathrm{z}|\mathrm{x};\phi)\|p(\mathrm{z};\theta))$$

- Variational lower-bound for ***a set of examples***

$$L(\theta, \phi, D) = \sum_i \mathbf{E}_{q(\mathrm{z}_i|\mathrm{x}_i;\phi)}[\log p(\mathrm{x}_i|\mathrm{z}_i;\theta)] - \mathrm{KL}(q(\mathrm{z}_i|\mathrm{x}_i;\phi)\|p(\mathrm{z}_i;\theta))$$

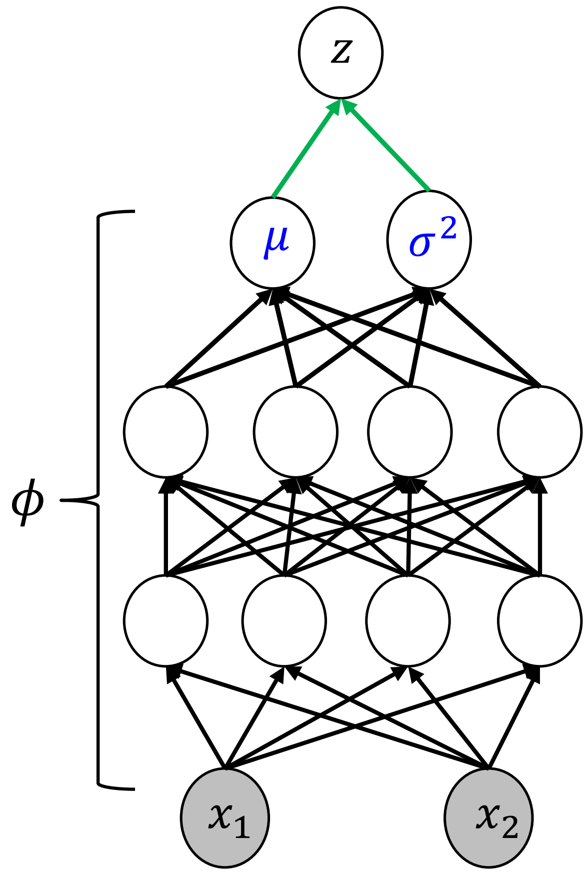
  - Use stochastic gradient methods to handle large datasets
  - Random mini-batch
    - for each i, infer the posterior $q(\mathrm{z}_i|\mathrm{x}_i;\phi)$; As we parameterize as a neural network, this in fact optimizes ϕ

- *However, calculating the expectation and its gradients is non-trivial, often intractable*

Example with Gaussian Distributions

- Use $N(0,1)$ as prior for z; $q(\mathrm{z}|\mathrm{x};\phi)$ is Gaussian with parameters $(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$ determined by NN
  - The KL-divergence

$$-\mathrm{KL}\big(q(z|\mathrm{x};\phi)\big\|p(z;\theta)\big) = \frac{1}{2}\left(1 + \log\sigma^2 - \mu^2 - \sigma^2\right)$$



  - **Exercise**: finish the derivation

Example with Gaussian Distributions

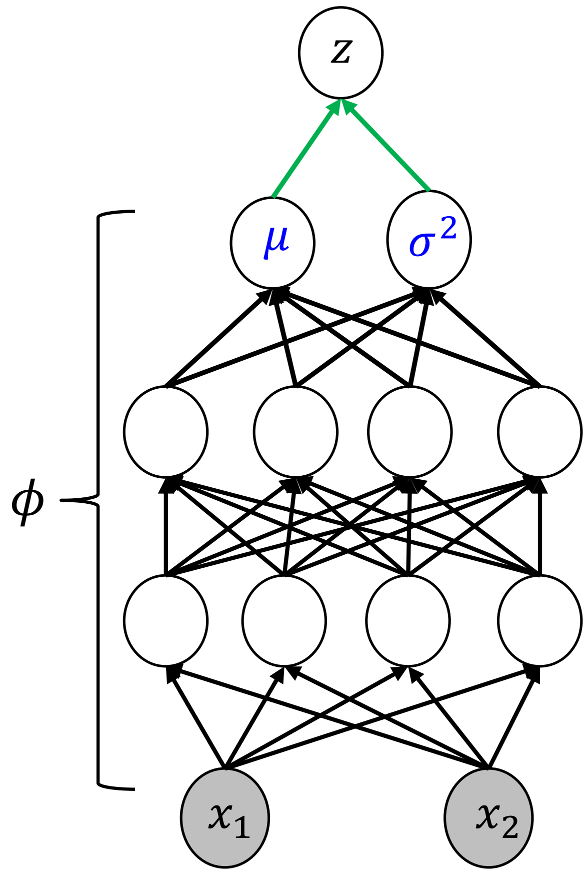
- Use $N(0,1)$ as prior for z; $q(\mathrm{z}|\mathrm{x};\phi)$ is Gaussian with parameters $(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$ determined by NN
  - The expected log-likelihood

$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|z;\theta)]$$

  - If the likelihood is Gaussian

$$-\log p(\mathrm{x}_i|z_i) = \sum_j \frac{1}{2}\log\sigma_j^2 + \frac{(x_{ij}-\mu_{xi})^2}{2\sigma_j^2}$$

  - *The expectation is still hard to compute because of nonlinearity functions*



Example with Gaussian Distributions

- Use $N(0,1)$ as prior for z; $q(\mathrm{z}|\mathrm{x};\phi)$ is Gaussian with parameters $(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$ determined by NN
  - The expected log-likelihood

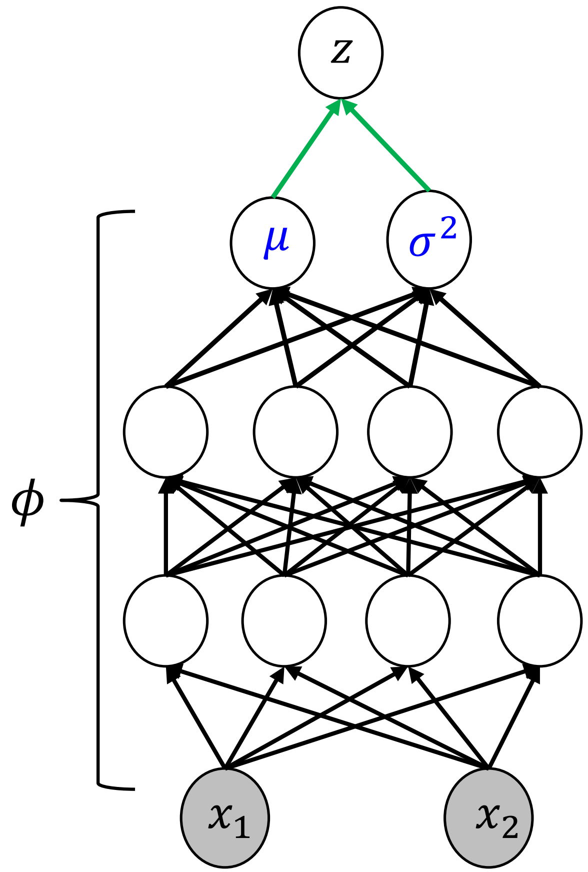
$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|z;\theta)]$$

  - Approximate via Monte Carlo methods

$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|\mathrm{z};\theta)] \approx \frac{1}{L}\sum_k \log p\left(\mathrm{x}|\mathrm{z}^{(k)}\right)$$

$$z^{(k)} \sim q(z|\mathrm{x};\phi)$$

  - An unbiased estimator



Example with Gaussian Distributions

- The KL-regularization term (closed-form):

$$-\mathrm{KL}(q(z|\mathrm{x};\phi)\|p(z;\theta)) = \frac{1}{2}(1 + \log \sigma^2 - \mu^2 - \sigma^2)$$

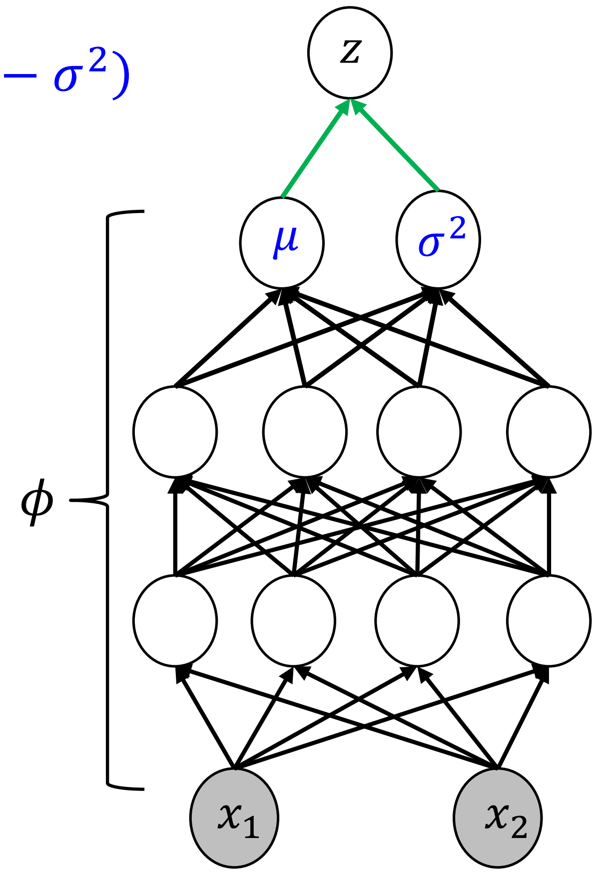
  - Easy to calculate gradient

- The expected log-likelihood term (MC estimate)

$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|\mathrm{z};\theta)] \approx \frac{1}{L}\sum_{k} \log p\left(x\middle|\mathrm{z}^{(k)}\right)$$

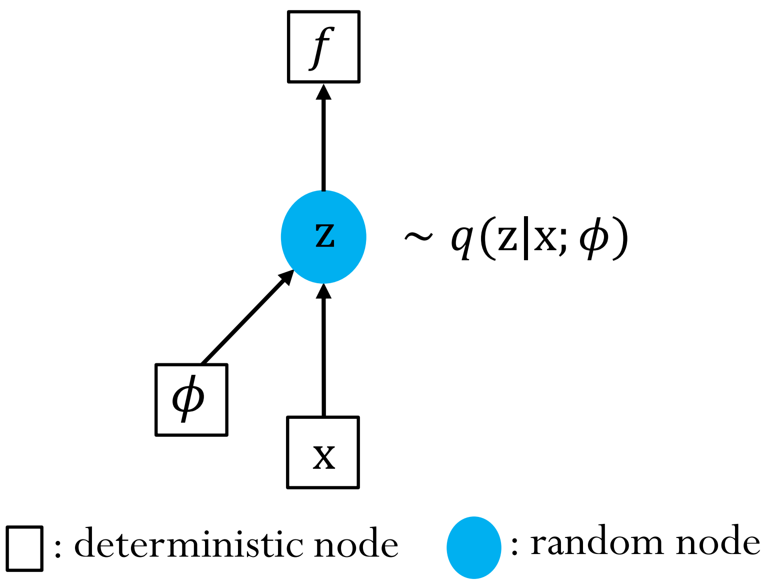
$$z^{(k)} \sim q(z|\mathrm{x};\phi)$$

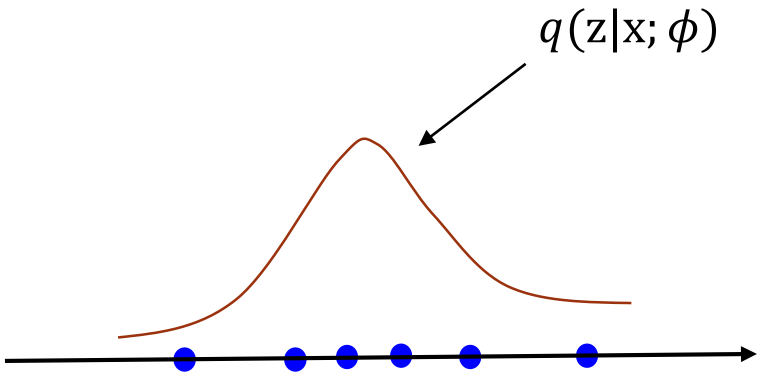
  - Gradient needs back-propagation!
  - *However, $z^{(k)}$ is a random variable, we can't take gradient over a randomly drawn number*



Reparameterization Trick

- Backpropagation not possible through random sampling





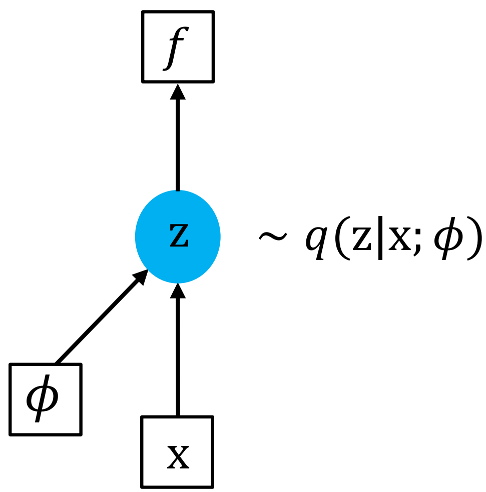
$\{-1.5, -0.5, 0.3, 0.6, 1.5, \ldots\}$

$$z^{(k)} \sim N\left(\mu(\mathrm{x},\phi), \sigma^2(\mathrm{x},\phi)\right)$$

Cannot back-propagate through a randomly drawn number

Reparameterization Trick

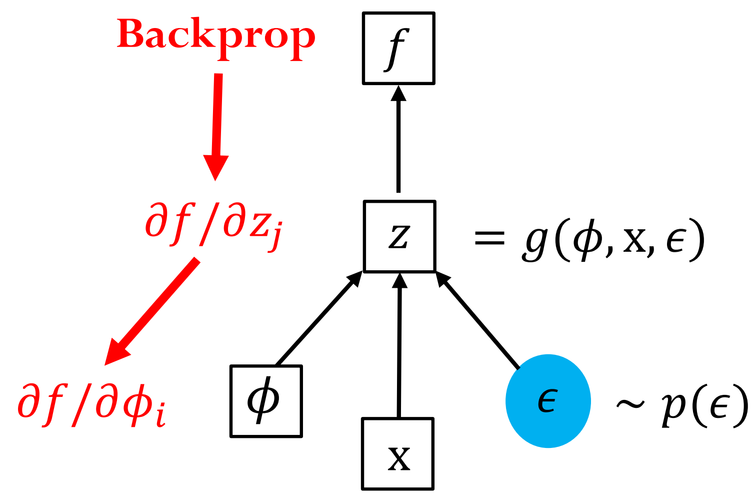
- Backpropagation not possible through random sampling



: deterministic node : random node

$$z^{(k)} \sim N\big(\mu(\mathrm{x},\phi), \sigma^2(\mathrm{x},\phi)\big)$$

Cannot back-propagate through a randomly drawn number



$$\epsilon^{(k)} \sim N(0,1)$$

$$z^{(k)} = \mu(\mathrm{x},\phi) + \sigma(\mathrm{x},\phi) \cdot \epsilon^{(k)}$$

Z has the same distribution, but now can back-prop

Separate into a deterministic part and noise

The General Form

- The VAE bound

$$L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{z}, \mathrm{x}; \theta) - \log q(\mathrm{z}|\mathrm{x}; \phi)]$$

$$= \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}\left[\log \frac{p(\mathrm{z}, \mathrm{x}; \theta)}{q(\mathrm{z}|\mathrm{x}; \phi)}\right]$$

- Monte Carlo estimate:

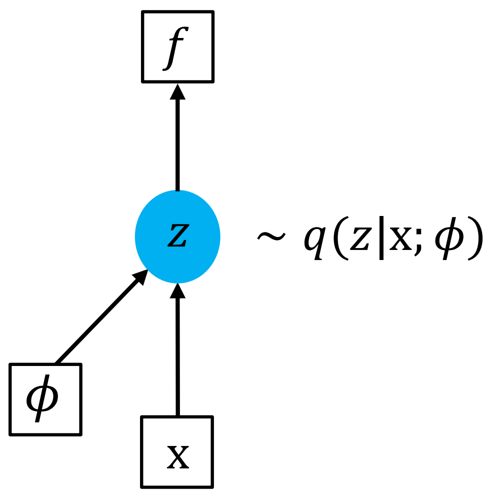
$$L(\theta, \phi, \mathrm{x}) \approx \frac{1}{L}\sum_k \log \frac{p\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)}{q\left(\mathrm{z}^{(k)}\middle|\mathrm{x}; \phi\right)}$$

$$z^{(k)} \sim q(z|\mathrm{x}; \phi)$$

  - Again, we cannot back-prop through the randomly drawn numbers

Reparameterization Trick

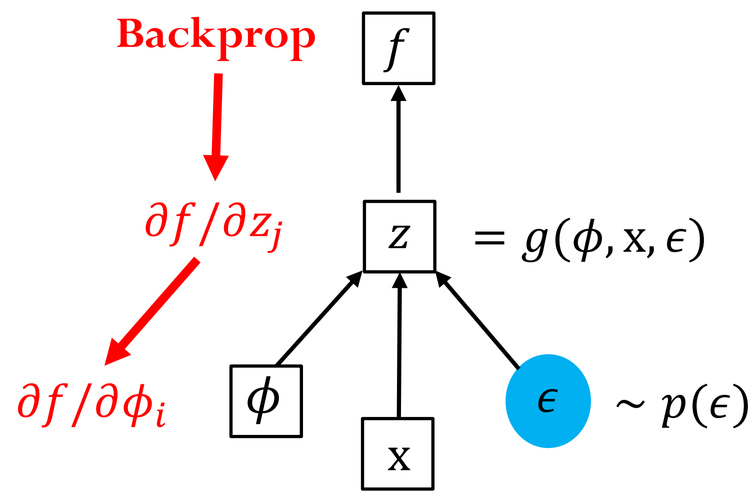
- Backpropagation not possible through random sampling



: deterministic node : random node

$$z^{(k)} \sim q(z|\mathrm{x};\phi)$$

Cannot back-propagate through a randomly drawn number



$$\epsilon^{(k)} \sim p(\epsilon)$$

$$z^{(k)} = g\left(\phi, x, \epsilon^{(k)}\right)$$

Z has the same distribution, but now can back-prop
Separate into a deterministic part and noise

Reparam-Trick Summary

- The VAE bound

$$L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}\left[\log \frac{p(\mathrm{z}, \mathrm{x}; \theta)}{q(\mathrm{z}|\mathrm{x}; \phi)}\right]$$

  - Reparameterized as

$$L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{p(\epsilon)}\left[\log \frac{p(g(\mathrm{x}, \epsilon, \phi), \mathrm{x}; \theta)}{q(g(\mathrm{x}, \epsilon, \phi)|\mathrm{x}; \phi)}\right]$$

    - where ϵ is a simple distribution (e.g., standard normal) and g is a deep NN

- The gradients are

$$\nabla_\theta L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{p(\epsilon)}\left[\nabla_\theta \log \frac{p(g(\mathrm{x}, \epsilon, \phi), \mathrm{x}; \theta)}{q(g(\mathrm{x}, \epsilon, \phi)|\mathrm{x}; \phi)}\right]$$

  - Back-prop is applied over the deep NN
  - Similar for ϕ

Importance Weighted Auto-Encoder (IWAE)

- The VAE lower bound of log-likelihood

$$L(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}\left[\log \frac{p(\mathrm{z}, \mathrm{x}; \theta)}{q(\mathrm{z}|\mathrm{x}; \phi)}\right]$$

- A better variational lower bound (IWAE)

$$L_K(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}\left[\log\left(\frac{1}{K}\sum_{k=1:K} \frac{p\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)}{q\left(\mathrm{z}^{(k)}\middle|\mathrm{x}; \phi\right)}\right)\right]$$

where $z^{(k)} \sim q(z|\mathrm{x}; \phi)$

  - This is a lower-bound of the log-likelihood
  - When K=1, recovers the VAE bound
  - When $K = \infty$, recovers the log-likelihood
  - A monotonic sequence:

$$L_K(\theta, \phi, \mathrm{x}) \leq L_{K+1}(\theta, \phi, \mathrm{x}), \qquad \forall \theta, \phi, \mathrm{x}$$

[Burda et al., arXiv, 2015]

Reparametrization Trick

- The IWAE bound:

$$L_K(\theta, \phi, \mathrm{x}) = \mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}\left[\log\left(\frac{1}{K}\sum_{k=1:K} w\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)\right)\right]$$

where $z^{(k)} \sim q(z|\mathrm{x};\phi)$ $\quad w\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta, \phi\right) = \dfrac{p\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)}{q\left(\mathrm{z}^{(k)}\middle|\mathrm{x}; \phi\right)}$

- Reparameterization form:

$$L_K(\theta, \phi, \mathrm{x}) = \mathbf{E}_{p(\epsilon)}\left[\log\left(\frac{1}{K}\sum_{k=1:K} w\left(g(\epsilon^{(k)}, \mathrm{x}, \phi), \mathrm{x}; \theta\right)\right)\right]$$

where $\epsilon^{(k)} \sim p(\epsilon)$

  - The gradient can be calculated as in VAE

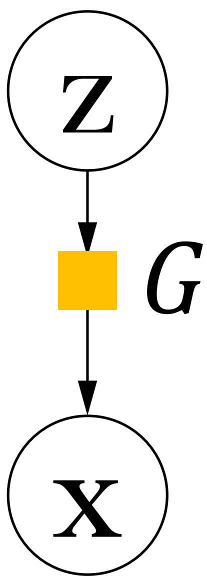
Generative Adversarial Networks (GAN)

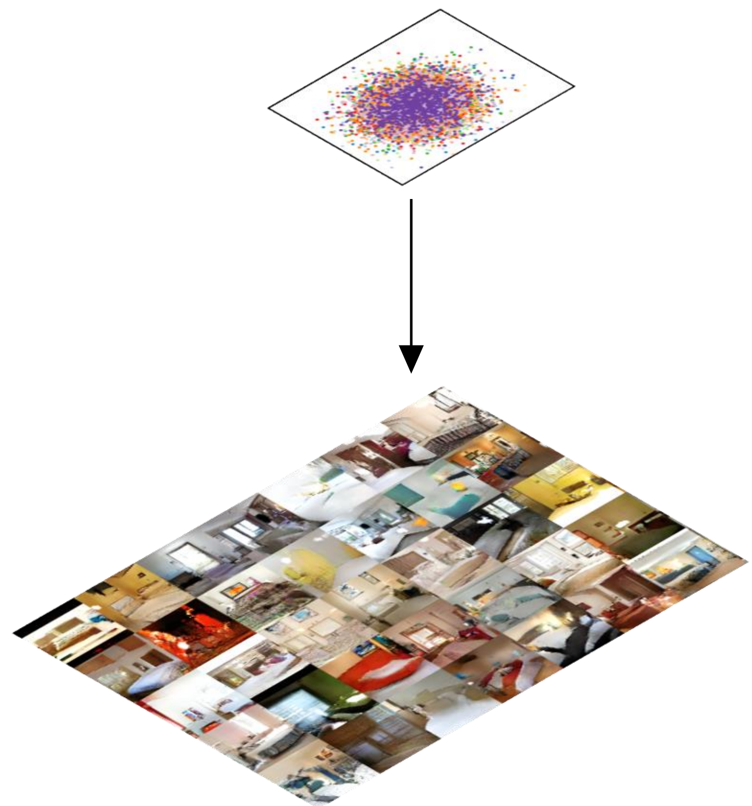
- A game between two players:
  - A discriminator *D*
  - A generator *G*
- *D* tries to discriminate between:
  - A sample from the data distribution.
  - And a sample from the generator *G*.
- *G* tries to "trick" *D* by generating samples that are hard for *D* to distinguish from data.

[Goodfellow et al., NIPS 2014]

GAN – architecture

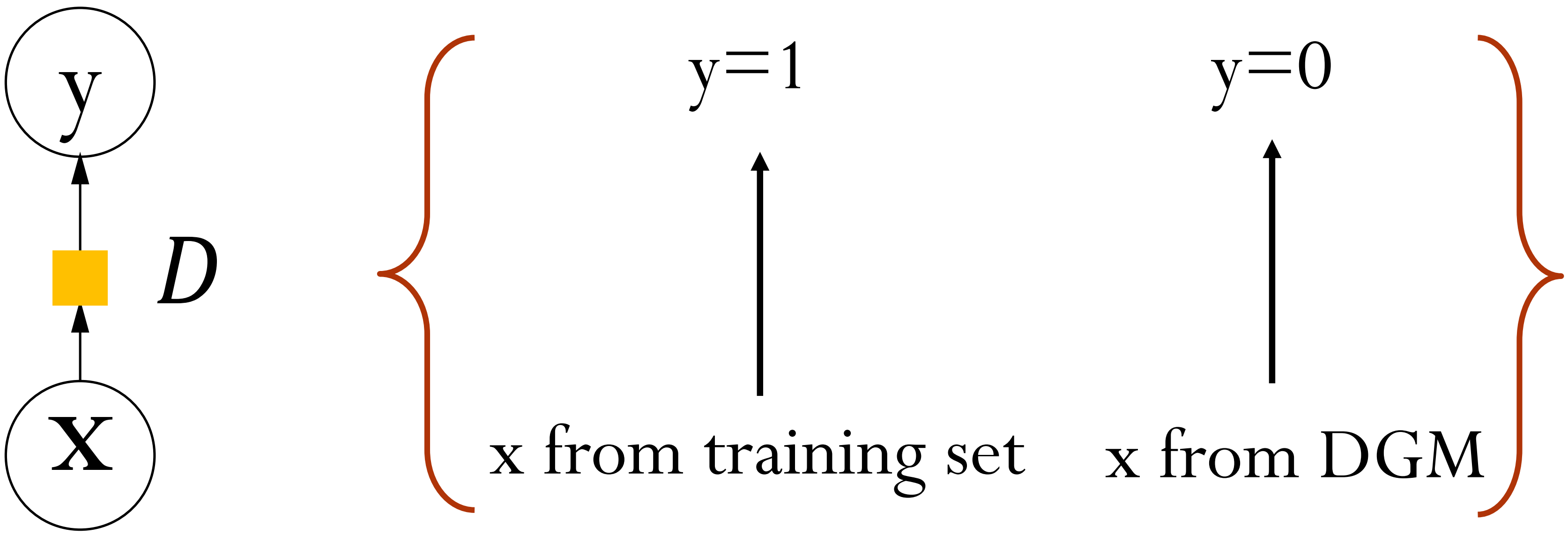
- The generator-network G is a DGM
  - It generates samples from random noise





GAN – architecture

- The discriminator-network D is a binary classifier
  - It aims to assign the correct label to both training samples and the samples from G



  - This is a supervised learning task (binary classification)!

GAN - objective

- The discriminator-network D is a binary classifier
  - It aims to assign the correct label to both training samples and the samples from G
- Maximum likelihood estimation (MLE) is the natural choice!

$$\max_{D} \mathbf{E}_{p_{\text{data}}(\mathrm{x})}[\log D(\mathrm{x})] + \mathbf{E}_{p(\mathrm{z})}\left[\log\left(1 - D\left(G(\mathrm{z})\right)\right)\right]$$

x from training set

G(z) from generator

  - $D(\mathrm{x}) = p(\mathrm{y}=1 \mid \mathrm{x})$
  - aka. cross-entropy loss minimization

GAN - objective

- The generator aims to fool the discriminator D
  - Generated samples should be identified as "real" by D
  - Maximize the likelihood of being real:

$$\max_G \mathbf{E}_{p(\mathrm{z})}\left[\log\left(D\left(G(\mathrm{z})\right)\right)\right]$$

$G(\mathrm{z})$ is a sample from generator

  - Or minimize the likelihood of being fake:

$$\min_G \mathbf{E}_{p(\mathrm{z})}\left[\log\left(1 - D\left(G(\mathrm{z})\right)\right)\right]$$

GAN – objective

- Minimax objective function

$$\min_G \max_D \ \mathbf{E}_{p_{\text{data}}(\text{x})}[\log D(\text{x})] + \mathbf{E}_{p(\text{z})}\left[\log\left(1 - D\left(G(\text{z})\right)\right)\right]$$

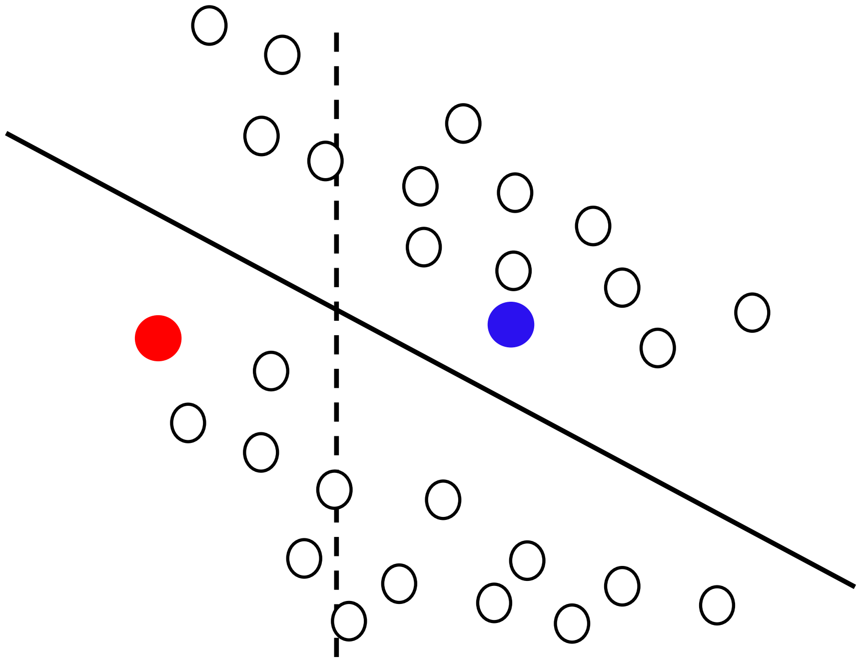
  - Optimal strategy of the discriminator for any $p_{\text{model}}(\text{x})$ is

$$D(\text{x}) = \frac{p_{\text{data}}(\text{x})}{p_{\text{data}}(\text{x}) + p_{\text{model}}(\text{x})}$$

  - Assume infinite data, infinite model capacity, direct updating generator's distribution
    - Unique global optimum
    - Optimum corresponds to data distribution

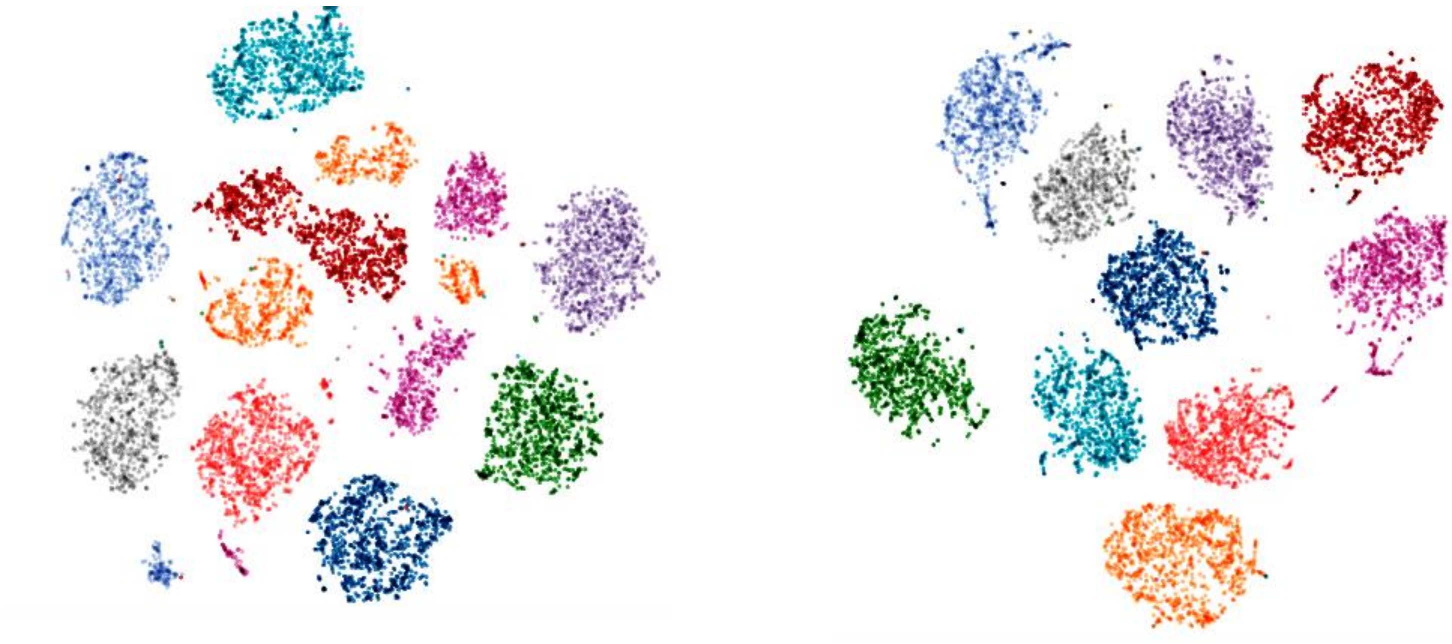
Semi-supervised Learning

- A toy example



Representation Matters

- t-SNE embedding of learned representations by different DGM models on CIFAR10



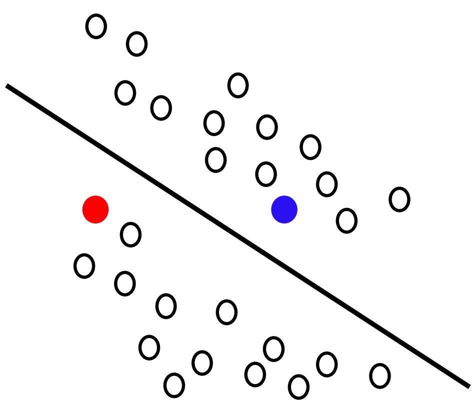
Triple Generative Adversarial Nets

- A minimax game for semi-supervised learning
  - GAN is for unsupervised learning $p_{\text{model}}(\mathrm{x}) = p_{\text{data}}(\mathrm{x})$
  - We aim to learn the joint distribution $p_{\text{model}}(\mathrm{x},\mathrm{y}) = p_{\text{data}}(\mathrm{x},\mathrm{y})$
- We need three players
  - factorization form with conditionals

$$p(x,y) = p(x)p(y|x)$$
$$= p(y)p(x|y)$$

A classifier

A class-conditional generator

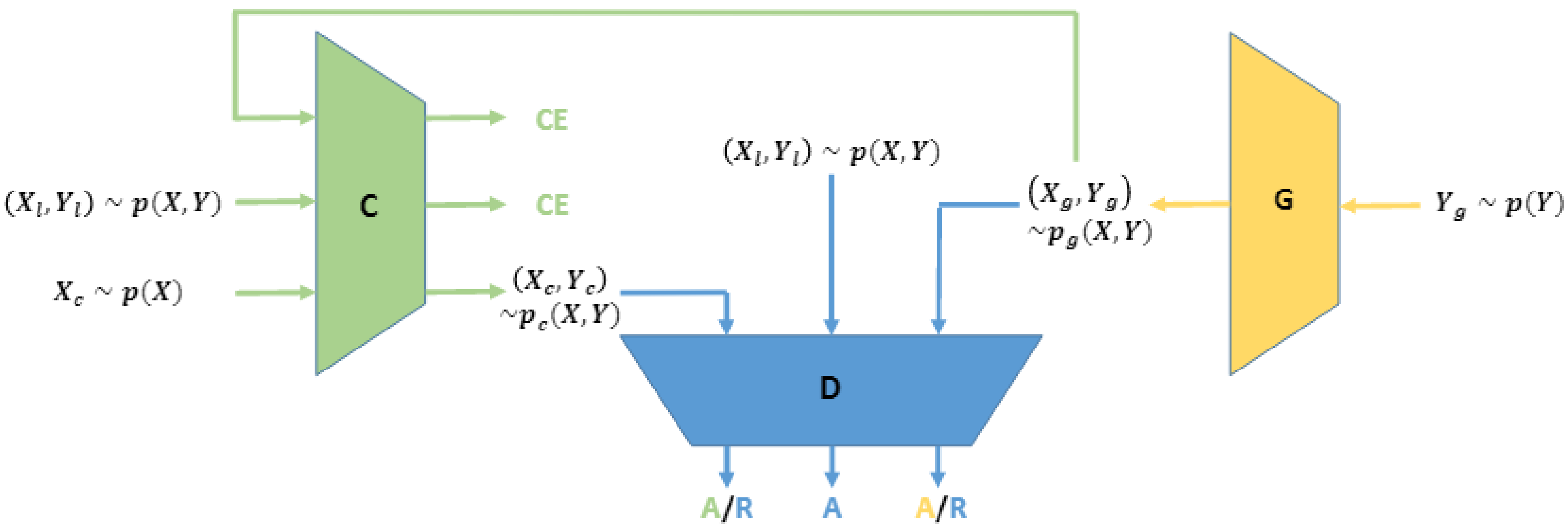


  - Two generators to generate (x,y)
  - A discriminator to distinguish fake (x,y)

[Li et al., NIPS 2017]

Triple-GAN

◆ The network architecture



- Both C and G are generators
- D is the discriminator
- CE: cross-entropy loss for learning classifier

[Li et al., NIPS 2017]

A minimax game

- The optimization problem

$$\min_{C,G}\max_{D} U(C, G, D) = E_p[\log D(x, y)] + \alpha E_{p_c}[\log(1 - D(x, y))] + (1 - \alpha) E_{p_g}[\log(1 - D(x, y))]$$

  - The hyper-parameter *a* is often set at 1/2
  - The standard supervised loss can be incorporated

$$\min_{C,G}\max_{D} \tilde{U}(C, G, D) = U(C, G, D) + E_p[-\log p_c(y|x)]$$

[Li et al., NIPS 2017]

Major theoretical results

Theorem

The equilibrium of $\tilde{U}(C, G, D)$ is achieved if and only if $p(x, y) = p_g(x, y) = p_c(x, y)$ with $D^*_{C,G}(x, y) = \frac{1}{2}$ and the optimum value is $-\log 4$.

Lemma

For any fixed C and G, the optimal discriminator D is:

$$D^*_{C,G}(x, y) = \frac{p(x, y)}{p(x, y) + p_\alpha(x, y)},$$

where $p_\alpha(x, y) := (1 - \alpha)p_g(x, y) + \alpha p_c(x, y)$.

Some Practical Tricks for SSL

- Pseudo discriminative loss: using $(x, y) \sim p_g(x, y)$ as labeled data to train C
  - Explicit loss, equivalent to $KL(p_g(x, y)||p_c(x, y))$
  - Complementary to the implicit regularization by D

- Collapsing to the empirical distribution $p(x, y)$
  - Sample $(x, y) \sim p_c(x, y)$ as true data for D
  - Biased solution: target shifting towards $p_c(x, y)$
- Unlabeled data loss on C
  - Confidence (Springenberg [2015])
  - Consistence (Laine and Aila [2016])

Some Results

- Semi-supervised classification

Table 1: Error rates (%) on partially labeled MNIST, SHVN and CIFAR10 datasets. The results with † are trained with more than 500,000 extra unlabeled data on SVHN.

Algorithm	MNIST $n = 100$	SVHN $n = 1000$	CIFAR10 $n = 4000$
M1+M2 [11]	3.33 ($\pm$0.14)	36.02 ($\pm$0.10)	
VAT [18]	2.33		24.63
Ladder [23]	1.06 ($\pm$0.37)		20.40 ($\pm$0.47)
Conv-Ladder [23]	**0.89** ($\pm$0.50)		
ADGM [17]	0.96 ($\pm$0.02)	22.86 †	
SDGM [17]	1.32 ($\pm$0.07)	16.61($\pm$0.24)†	
MMCVA [15]	1.24 ($\pm$0.54)	**4.95** ($\pm$0.18) †	
CatGAN [26]	1.39 ($\pm$0.28)		19.58 ($\pm$0.58)
Improved-GAN [25]	0.93 ($\pm$0.07)	8.11 ($\pm$1.3)	18.63 ($\pm$2.32)
ALI [5]		7.3	18.3
Triple-GAN **(ours)**	**0.91** ($\pm$0.58)	**5.77**($\pm$0.17)	**16.99** ($\pm$0.36)

Some Results

- Class-conditional generation

(d) Dog (e) Horse (f) Ship

Some Results

- Disentangle class and style

Figure: Same y for each row. Same z for each column.

Some Results

- Latent space interpolation on MNIST

Some Results

- Latent space interpolation on SVHN

Some Results

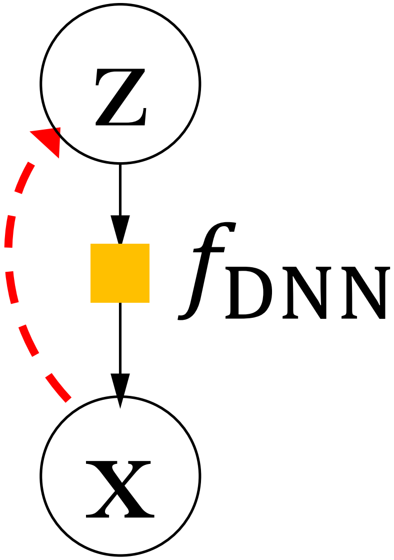
- Latent space interpolation on CIFAR10

ZhuSuan

http://zhusuan.readthedocs.io

Bayesian inference

$$p(\mathrm{z}|\mathrm{x}) = \frac{p(\mathrm{x}, \mathrm{z})}{p(\mathrm{x})}$$



Given Disease, what is the Cause?

Given Object, what are the Components?

Given Docs, what are the Topics?

Find Cause of Disease

Extract Topics from Docs

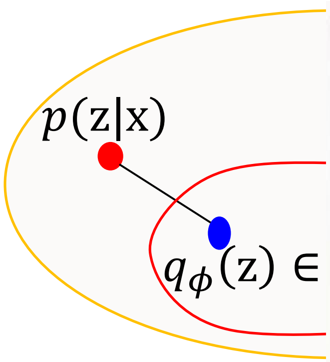
Identify Objects from Images

Recognize Words in Speeches

Inference in Old Days

Variational Inference

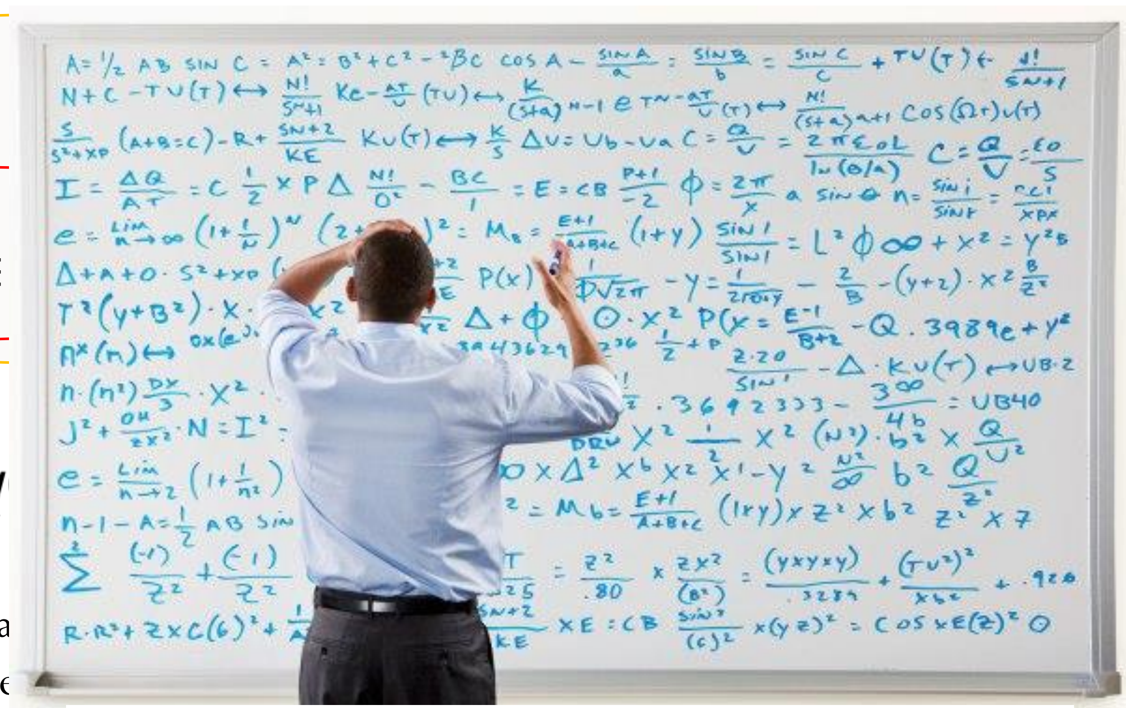
(Too much math!!!)



$\min_{\phi} \mathrm{KL}(q$

Varia

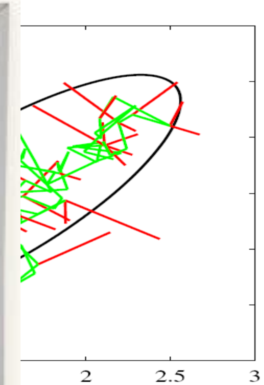
poste



Extremely painful for a single model!

MCMC

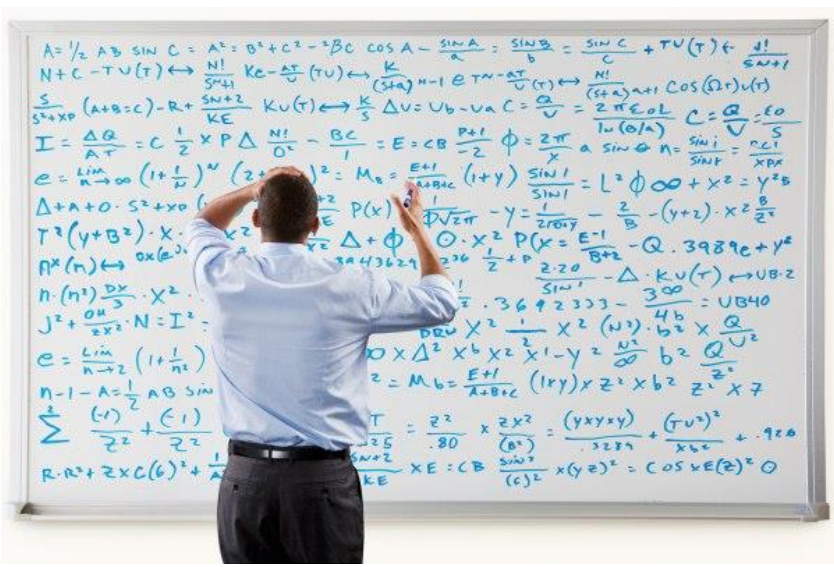
(Many dynamics!!!)

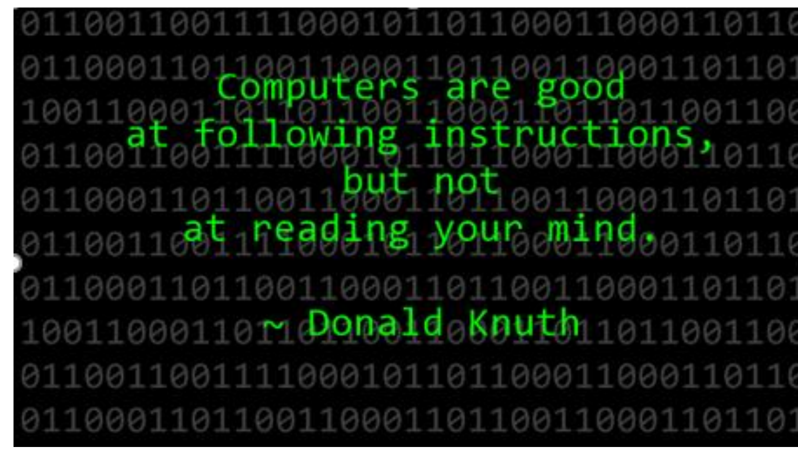


$\nabla_q[p^\top G^{-1}p]\mathrm{d}t + \mathcal{N}(0, 2CG\mathrm{d}t)$

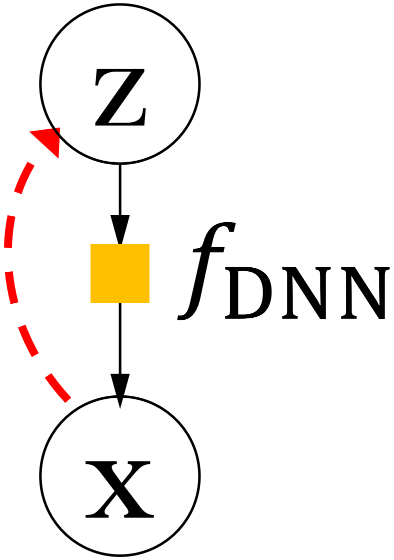
Inference in ZhuSuan

Turn painful math deviations into Easy and Intuitive (Probabilistic) Programming





ZhuSuan

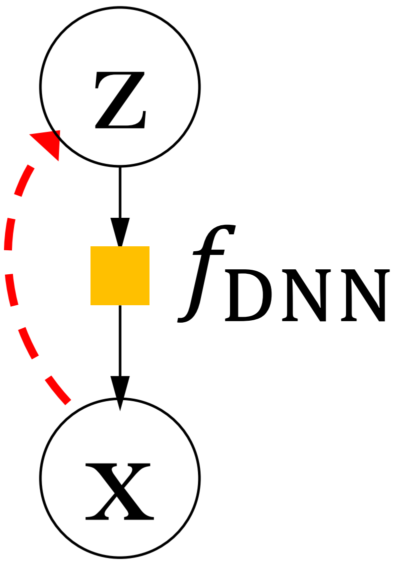


ZhuSuan is a python library for generative models, built upon TensorFlow.

Unlike existing DL libraries, which are mainly for supervised tasks, ZhuSuan is featured for:

- its deep root into Bayesian Inference
- supporting various kinds of generative models: traditional hierarchical Bayesian models & recent deep generative models.

ZhuSuan

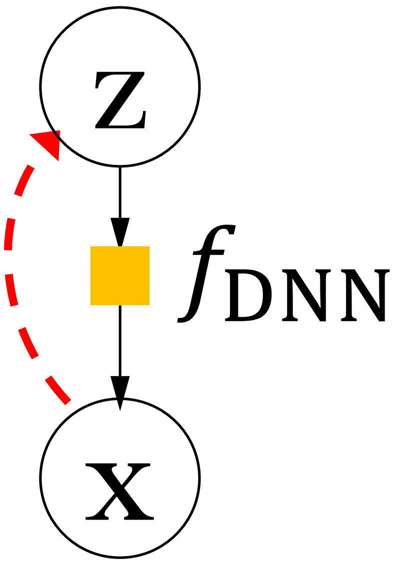


With ZhuSuan, users can enjoy

- powerful fitting and multi-GPU training of deep learning

- while at the same time they can use generative models to
  - ✓ model the complex world
  - ✓ exploit unlabeled data
  - ✓ deal with uncertainty by performing principled Bayesian inference
  - ✓ generate new samples

Model Primitives: BayesianNet

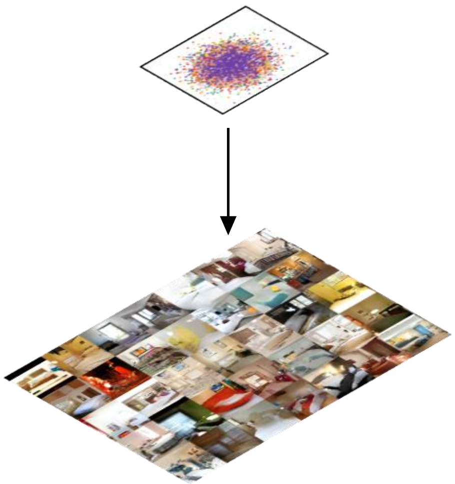


- A DAG representing a Bayesian Network
- Two types of nodes:
  - Deterministic nodes: Can be composed of any Tensorflow operations.
  - Stochastic nodes: Use StochasticTensor's from ZhuSuan's library.
- Start a BayesianNet environment

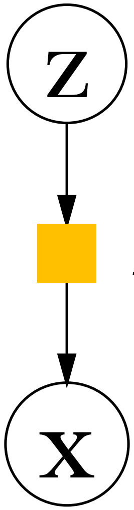
```
import zhusuan as zs
with zs.BayesianNet() as model:
    # build the model
```

Example: Variational Autoencoders

ZHUSUAN



$$
\begin{aligned}
z &\sim \mathrm{N}(z|0, I) \\
x_{logits} &= f_{NN}(z) \\
x &\sim \mathrm{Bernoulli}(x|\mathrm{sigmoid}(x_{logits}))
\end{aligned}
$$



f_{DNN}: $2L$ MLP

```
import tensorflow as tf
from tensorflow.contrib import layers
import zhusuan as zs

with zs.BayesianNet() as model:
    z_mean = tf.zeros([n, n_z])
    z_logstd = tf.zeros([n, n_z])
    z = zs.Normal('z', z_mean, z_logstd)
    h = layers.fully_connected(z, 500)
    x_logits = layers.fully_connected(
        h, n_x, activation_fn=None)
    x = zs.Bernoulli('x', x_logits)
```

Variational Inference in ZhuSuan

ZHUSUAN

```
with zs.BayesianNet() as variational:
    # build variational ...
qz_samples, log_qz = variational.query(
    ‘z’, outputs=True, local_log_prob=True)
```

- Build variational posterior as BayesianNet

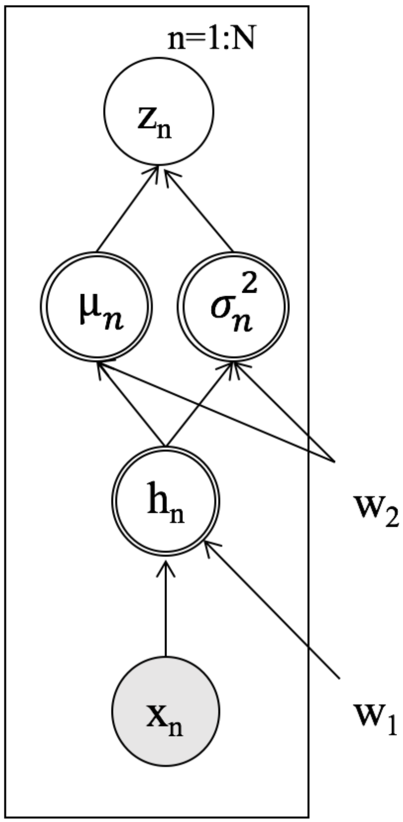
- Call variational objectives

```
lower_bound = zs.sgvb(log_joint, observed={‘x’: x},
                      latent={‘z’: [qz_samples, log_qz]})
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
run_op = optimizer.minimize(-lower_bound)
With tf.Session() as sess:
    for iter in range(iters):
        sess.run(run_op)
```

- Run gradient descent!

Example: Variational Autoencoders



Structured variational posterior as a BayesianNet ALSO.

```
import tensorflow as tf
from tensorflow.contrib import layers
import zhusuan as zs

with zs.BayesianNet() as variational:
    x = tf.placeholder([None, n_x], tf.float32)
    h = layers.fully_connected(x, 500)
    z_mean = layers.fully_connected(
        h, n_z, activation_fn=None)
    z_logstd = layers.fully_connected(
        h, n_z, activation_fn=None)
    z = zs.Normal('z', z_mean, z_logstd)
```

Variational Inference Algorithms

ZHUSUAN

ZhuSuan supports a broad class of variational objectives, ranging from widely used evidence lower bounds to recent state-of-arts.

Works for continuous latent variables

- `zs.sgvb`: Stochastic gradient variational Bayes.
- `zs.iwae`: Importance weighted lower bounds.

Works for both continuous and discrete latent variables

- `zs.nvil`: Variance reduced score function estimator/REINFORCE.
- `zs.vimco`: Variance reduced multi-sample score function estimator.

* This is like optimization algorithms (SGD, momentum, Adam, etc.) in deep learning software. Users need not dive into the technical details of these algorithms because ZhuSuan provides easy-to-use APIs for users to directly try on their generative models.

HMC like a TensorFlow optimizer

ZHUSUAN

```
# like creating the variable to optimize over.
z = tf.Variable(0.)
```

- Create the variable to store samples

```
# like optimizer = tf.train.AdamOptimizer(…)
hmc = zs.HMC(step_size=1e-3, n_leapfrogs=10)
```

- Initialize HMC

```
# like optimize_op = optimizer.minimize(…)
sample_op, hmc_info = hmc.sample(
  log_joint, observed={'x': x}, latent={'z': z})
```

- Call sample() method to return a sample operation

```
with tf.Session() as sess:
    for iter in range(iters):
        # like sess.run(optimize_op)
        _ = sess.run(sample_op)
```

- Run the sample operation like an optimizer!

Applications: ZhuSuan as a Research Platform

ZHUSUAN

ZhuSuan is featured for both Bayesian Statistics and Deep Learning.
State-of-the-Art models can be found in ZhuSuan's examples.

- Bayesian Logistic Regression
- Bayesian Neural Nets for Multivariate Regression
- (Convolutional) Variational Autoencoders (VAE)
- Semi-supervised learning for images with VAEs
- Deep Sigmoid Belief Networks
- Generative Adversarial Networks (GAN)
- Gaussian processes (GPs)
- Topic Models
- More to come …

ZhuSuan: GitHub Page

ZHUSUAN

thu-ml / **zhusuan**

Watch 107 | Star 1,103 | Fork 219

Contributions & Stars Welcome!

Code | Issues 5 | Pull requests 4 | Projects 0 | Insights

Branch: master ▾ zhusuan / docs / **index.rst**

Jiaxin Change README to markdown for displaying the logo.

1 contributor

`86 lines (63 sloc) | 2.49 KB`

Welcome to ZhuSuan

ZhuSuan is a python probabilistic programming library for
advantages of Bayesian methods and deep learning. ZhuSu
which are mainly designed for deterministic neural network
primitives and algorithms for building probabilistic models
algorithms include:

- Variational inference with programmable variational pc
  (SGVB, REINFORCE, VIMCO, etc.).
- Importance sampling for learning and evaluating models, with programmable proposals.
- Hamiltonian Monte Carlo (HMC) with parallel chains, and optional automatic parameter tuning.

README.md

ZhuSuan

ZhuSuan is a python library for **Generative Models**, built upon Tensorflow. Unlike existing deep learning libraries, which are mainly designed for supervised tasks, ZhuSuan is featured for its deep root into Bayesian Inference, thus supporting various kinds of generative models: both the traditional **hierarchical Bayesian models** and recent **deep generative models**.

With ZhuSuan, users can enjoy powerful fitting and multi-GPU training of deep learning, while at the same time they can use generative models to model the complex world, exploit unlabeled data and deal with uncertainty by performing principled Bayesian inference.

Supported Inference

(Stochastic) Variational Inference (VI & SVI)

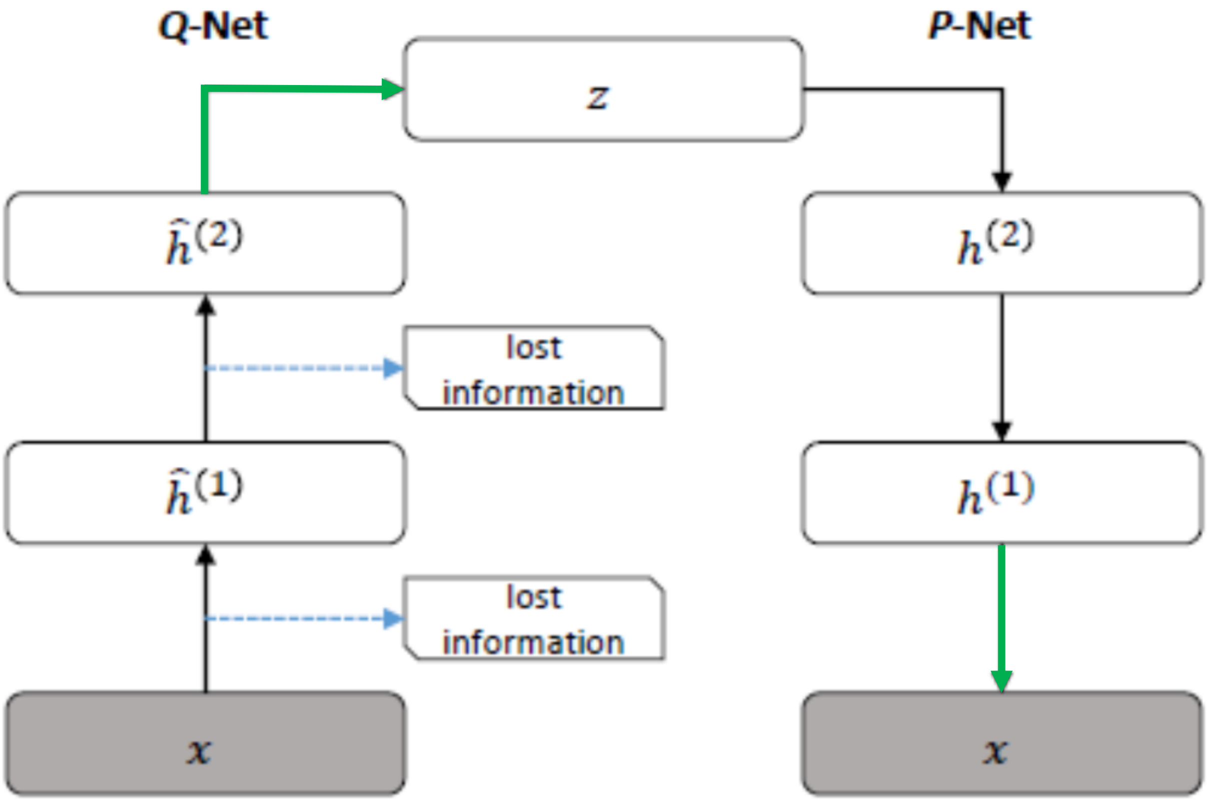
- Kinds of variational posteriors we support:
  - **Mean-field** posterior: Fully-factorized.
  - **Structured** posterior: With user specified dependencies.
- Variational objectives we support:
  - **SGVB**: Stochastic gradient variational Bayes
  - **IWAE**: Importance weighted objectives
  - **NVIL**: Score function estimator with variance reduction
  - **VIMCO**: Multi-sample score function estimator with variance

github.com/thu-ml/zhusuan

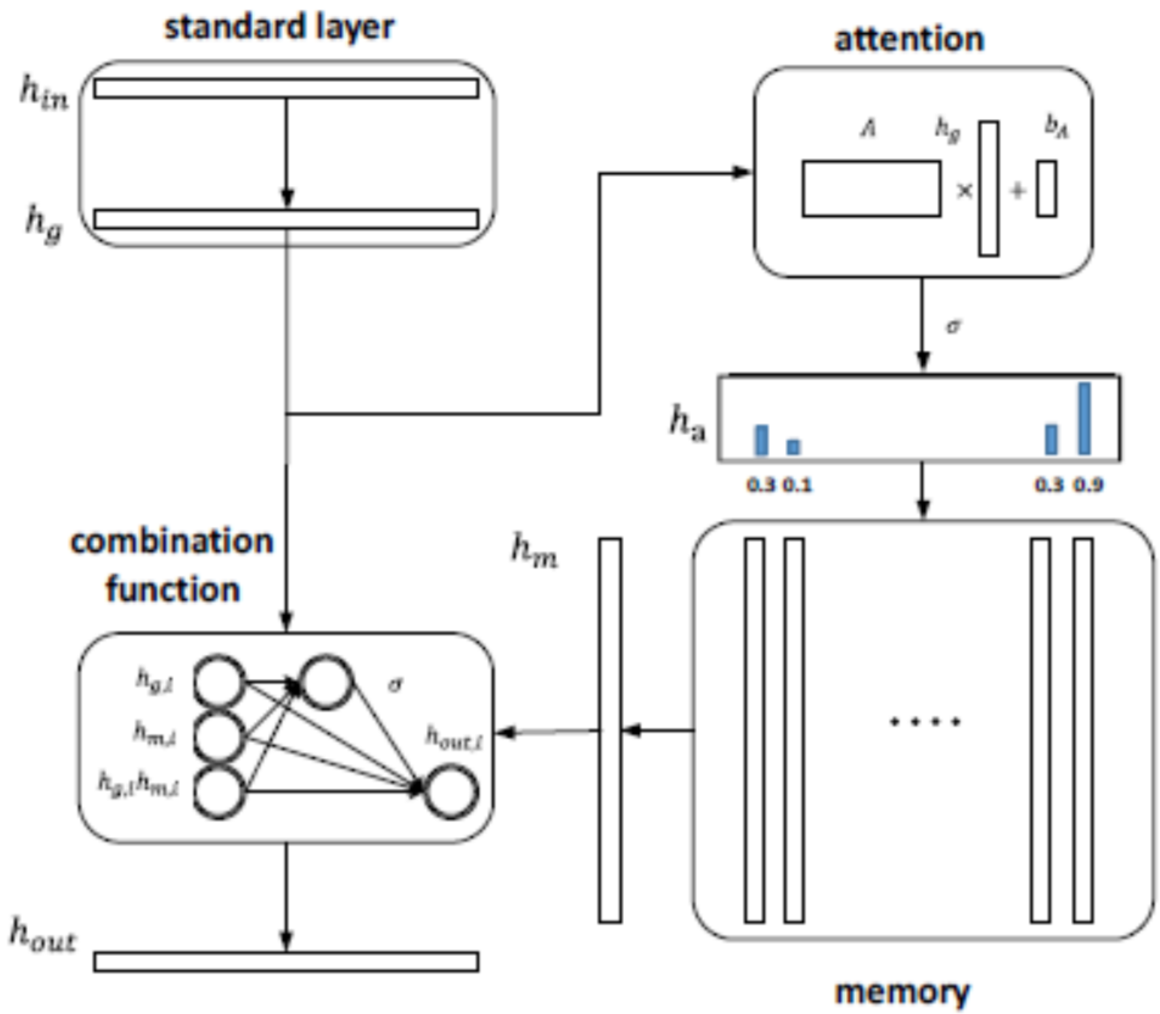
More Applications

Symmetric Q-P Network

- **Problem**: detail information is lost during abstraction



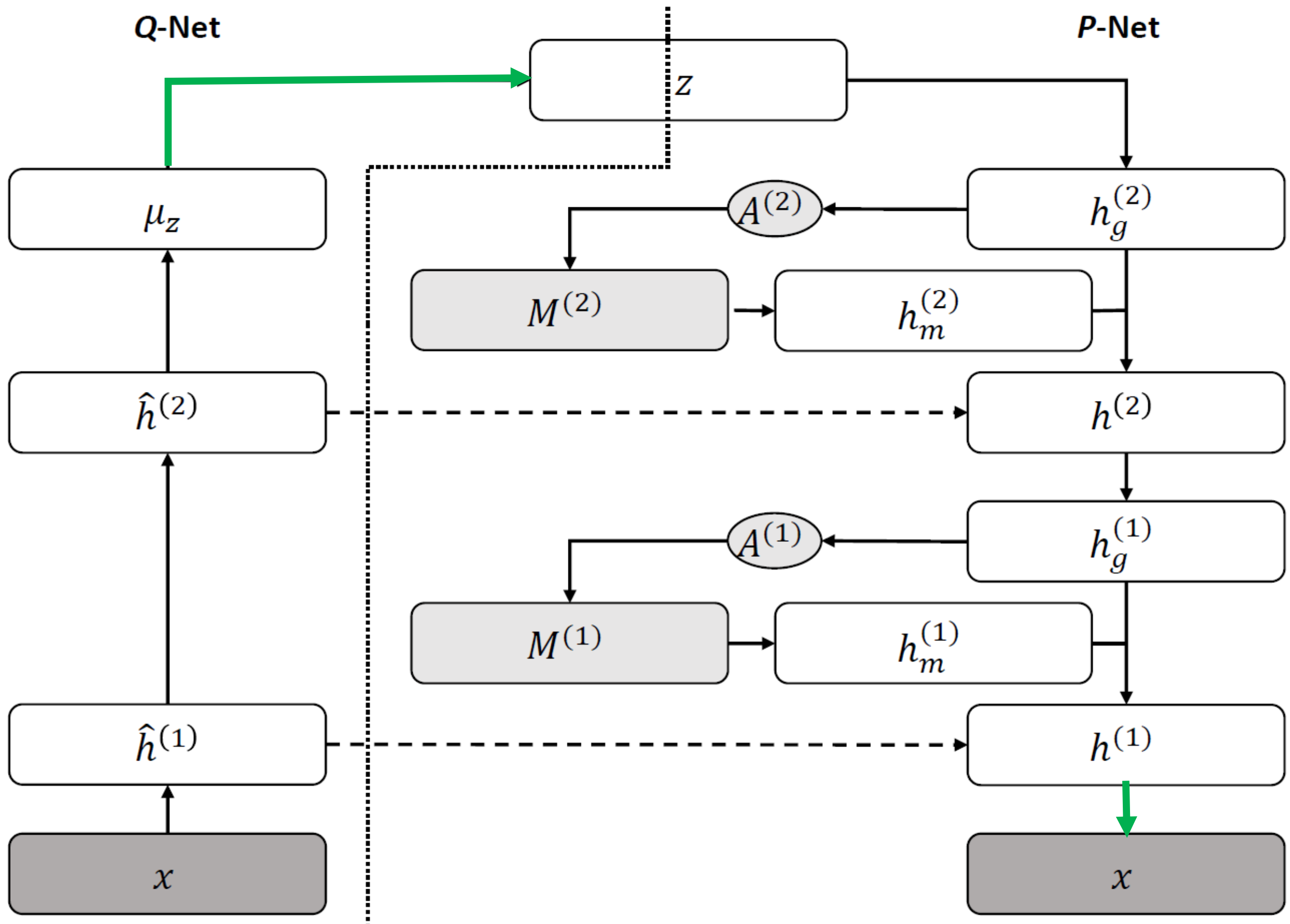
A Layer with Memory and Attention



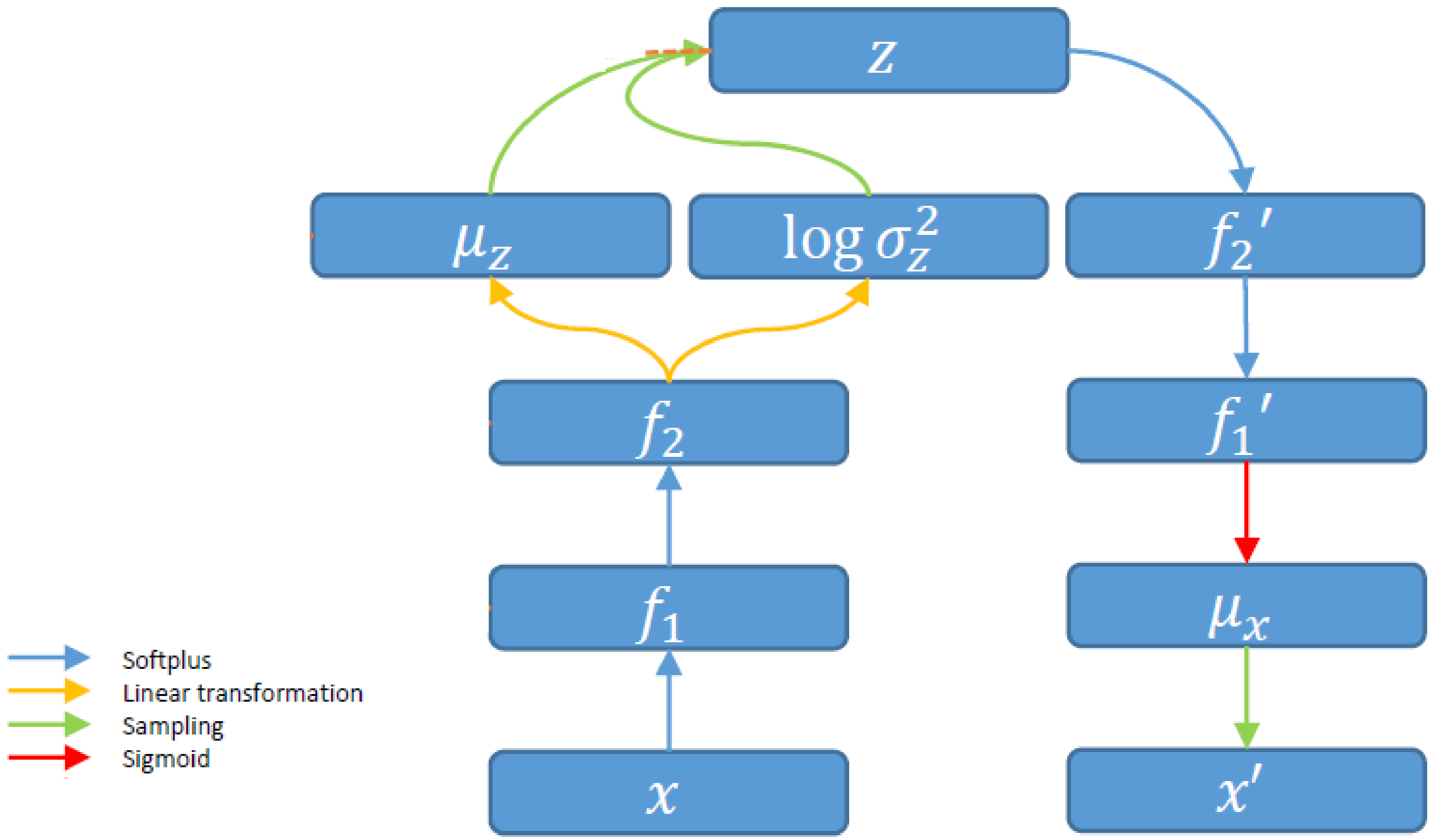
[Li, et al., ICML 2016]

A Stacked Deep Model with Memory

- Asymmetric architecture

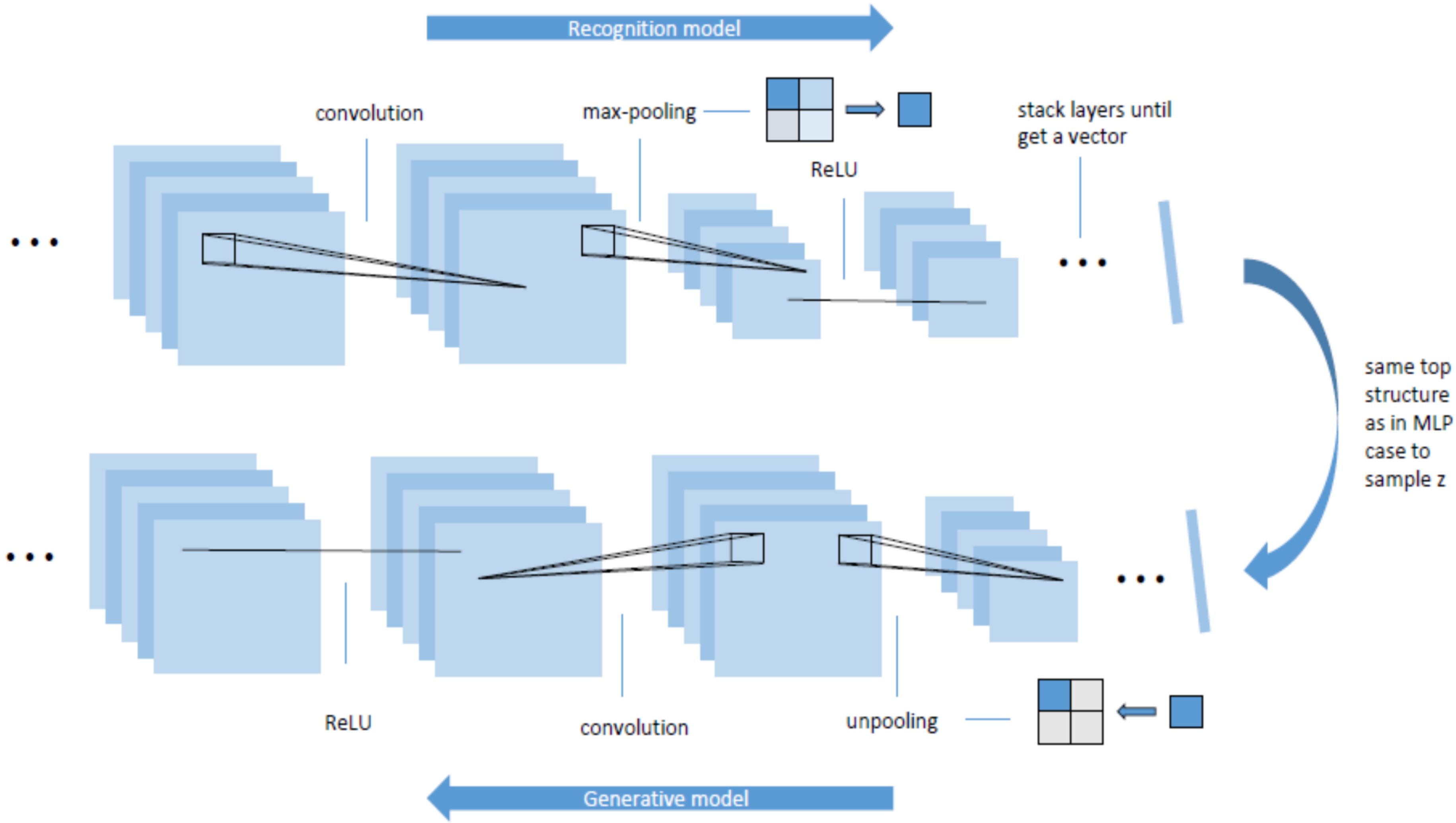


2-Layer MLP: Q-P network architecture



*Same as in Auto-Encoding Variational Bayes (VA) [Kingma & Welling, 2014]

5-Layer CNN: Q-P network architecture



Some Results

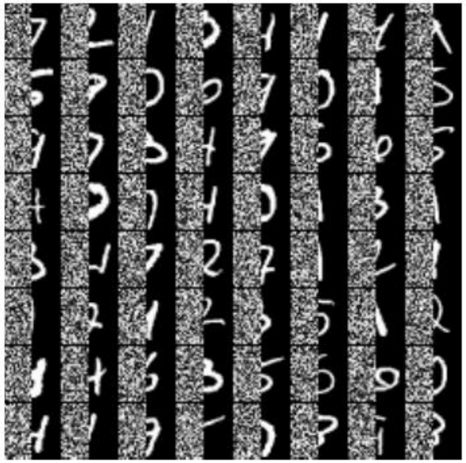
- Density estimation

Models	MNIST	OCR-letters
VAE	-85.69	-30.09
MEM-VAE(ours)	**-84.41**	**-29.09**
IWAE-5	-84.43	-28.69
MEM-IWAE-5(ours)	**-83.26**	**-27.65**
IWAE-50	-83.58	-27.60
MEM-IWAE-50(ours)	**-82.84**	**-26.90**

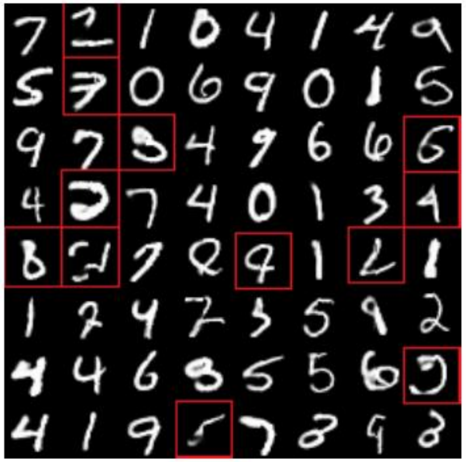
  - Better than symmetric VAE networks
  - Comparable with state-of-the-art with much fewer parameters

Missing Value Imputation

(a) Data



(b) Noisy data



(c) Results of VAE

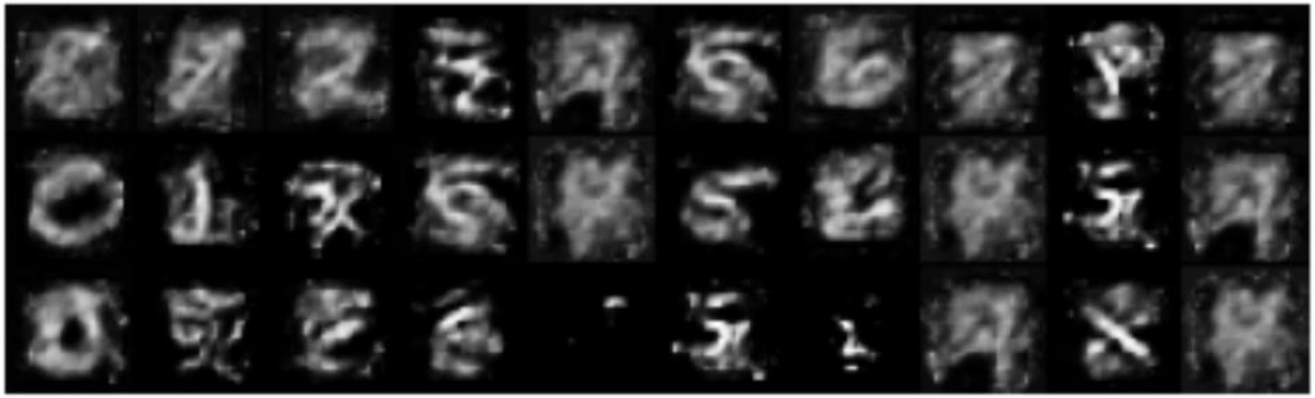
(d) Results of MEM-VAE

Learnt Memory Slots

- Average preference over classes of the first 3 slots:

"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
0.27	0.82	0.33	0.11	0.34	0.15	0.49	0.27	0.09	0.28
0.24	0.09	0.06	0.11	0.30	0.13	0.12	0.27	0.09	0.21
0.18	0.05	0.06	0.11	0.07	0.07	0.05	0.11	0.09	0.18

- Corresponding images:



References

- Kingma, D. P. and Welling, M. *Auto-encoding variational Bayes*. In ICLR, 2013.
- Rezende, D. J., Mohamed, S., and Wierstra, D. *Stochastic backpropagation and approximate inference in deep generative models*. In ICML, 2014.
- Burda, Y., Grosse, R., and Salakhutdinov, R. *Importance weighted autoencoders*. In arXiv:1509.00519, 2015.
- Goodfellow, I. J., Abadie, J. P., Mirza, M., Xu, B., Farley, D. W., S.ozair, Courville, A., and Bengio, Y. *Generative adversarial nets*. In NIPS, 2014
- Bengio, Y., Thlhodeau-Laufer, E., Alain, G., and Yosinski, J. *Deep generative stochastic networks trainable by backprop*. In ICML, 2014.
- Li, C., Zhu, J., and Zhang, B. *Learning to generate with memory*. In ICML, 2016

Thanks!

ZhuSuan: A Library for Bayesian Deep Learning. J. Shi, J. Chen, J. Zhu, S. Sun, Y. Luo, Y. Gu, Y. Zhou. arXiv preprint, arXiv:1709.05870 , 2017

Online Documents: http://zhusuan.readthedocs.io/