[70240413 Statistical Machine Learning, Spring, 2017]

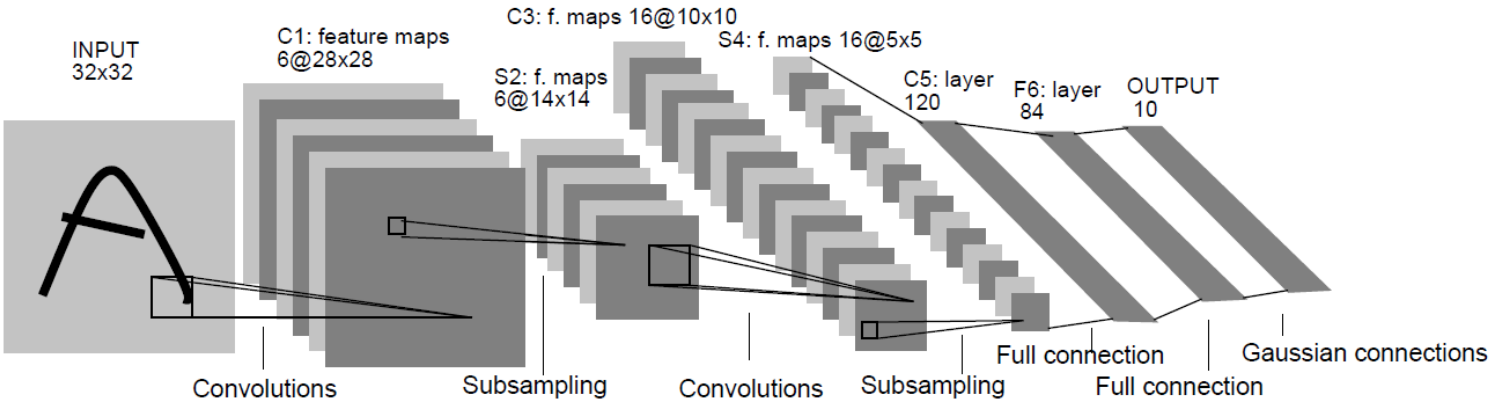
Deep Generative Models

Jun Zhu
dcszj@mail.tsinghua.edu.cn
http://bigml.cs.tsinghua.edu.cn/~jun
State Key Lab of Intelligent Technology & Systems
Tsinghua University

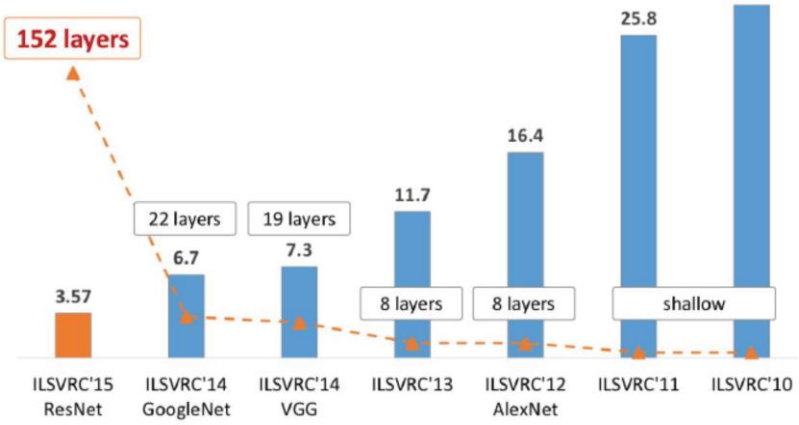
April 18, 2017

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

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

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

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

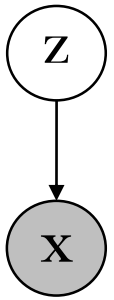
- Many recent advances on VAE, GAN, etc

Probabilistic Generative Models

- **Assumption**: data is described by some factors, which are often hidden

$$z \sim p(z|\alpha)$$

$$x \sim p(x|z,\beta)$$

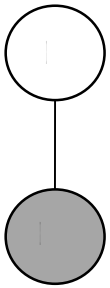


  - **Inference**: infer the posterior distribution

$$p(z|x;\theta) \propto p(z|\alpha)p(x|z,\beta)$$

- Undirected random fields

$$z, x \sim p(z, x|\theta)$$

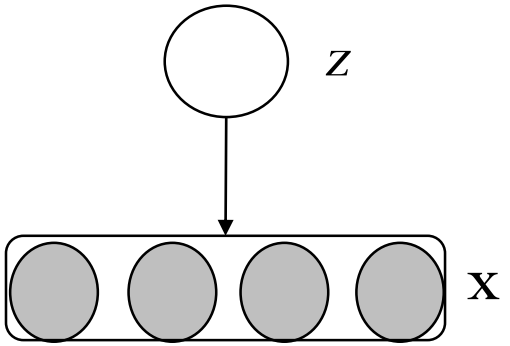


Examples

- **Latent-class** mixture models

$$p(z|\alpha) = \text{Mult}(\alpha)$$

$$p(\text{x}|z,\beta) = N(\mu_z, \Sigma_z)$$

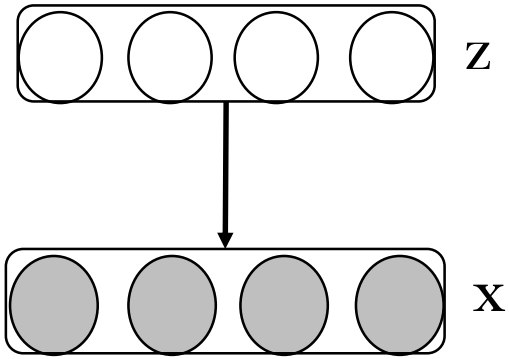


- **Latent-feature** factor analysis models

$$p(\mathbf{z}|\alpha) = N(\mu_0, \Sigma_0)$$

$$p(\text{x}|\mathbf{z},\beta) = N(W\mathbf{z}, \Sigma)$$

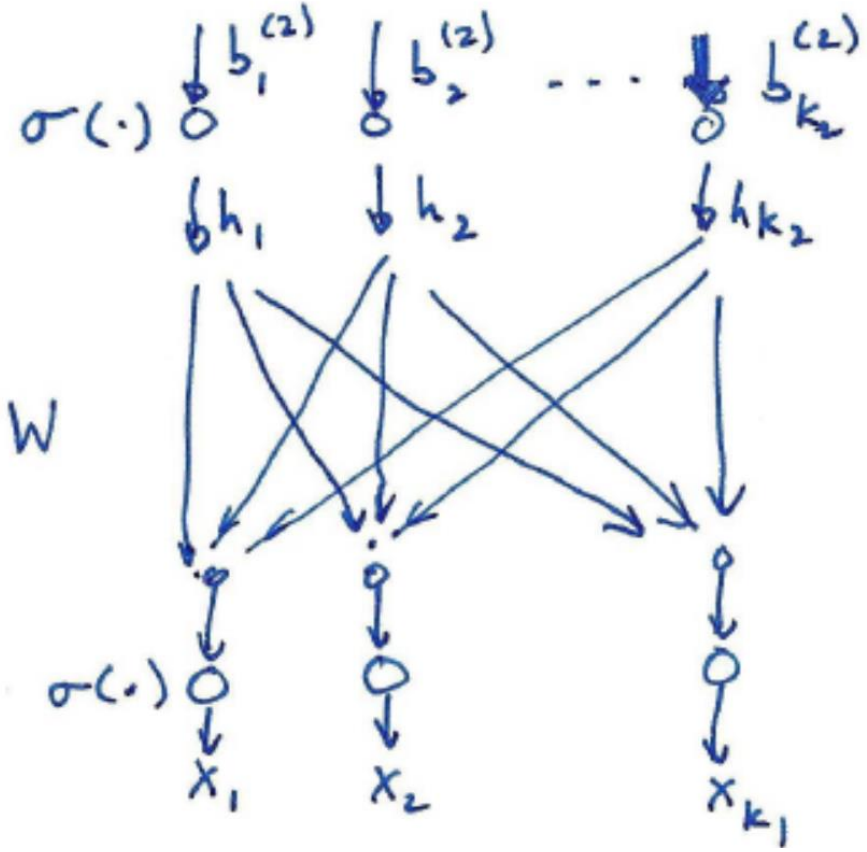
  - W is a loading matrix



- **Building blocks for deep models**

Examples

- Sigmoid belief network



$$h_k \sim \text{Bern}(\sigma(b_k^{(2)})), \;\; k = 1, \ldots, K_2$$

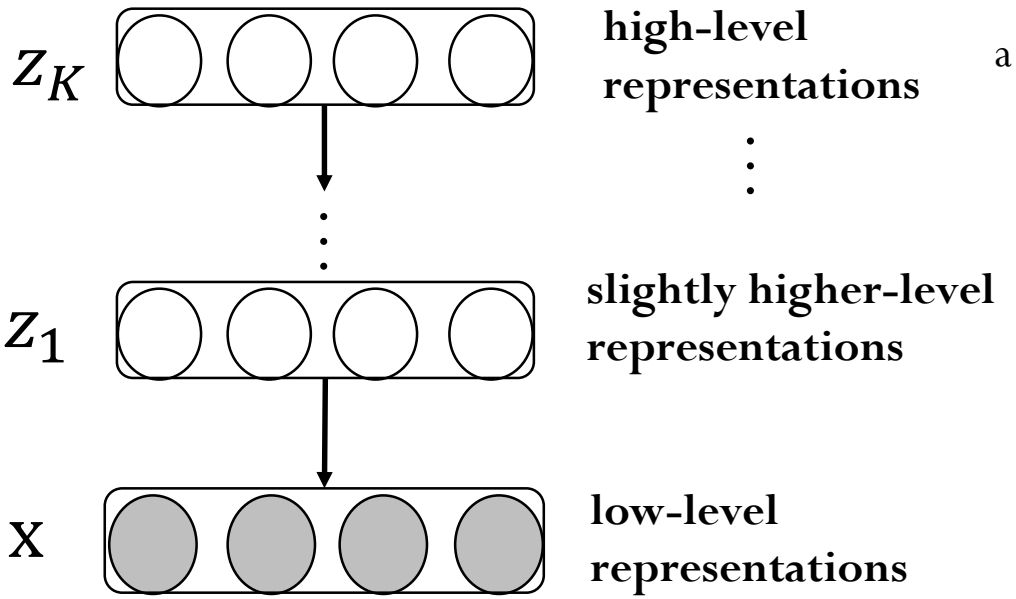
$$\boldsymbol{z} = \mathbf{W}\boldsymbol{h} + \boldsymbol{b}^{(1)}$$

$$x_j \sim \text{Bern}(\sigma(z_j)), \;\; j = 1, \ldots, K_1$$

[Slide courtesy: Larry Carin]

Deep Generative Models

- Multi-layer ***latent-feature*** representations with nonlinear transformations



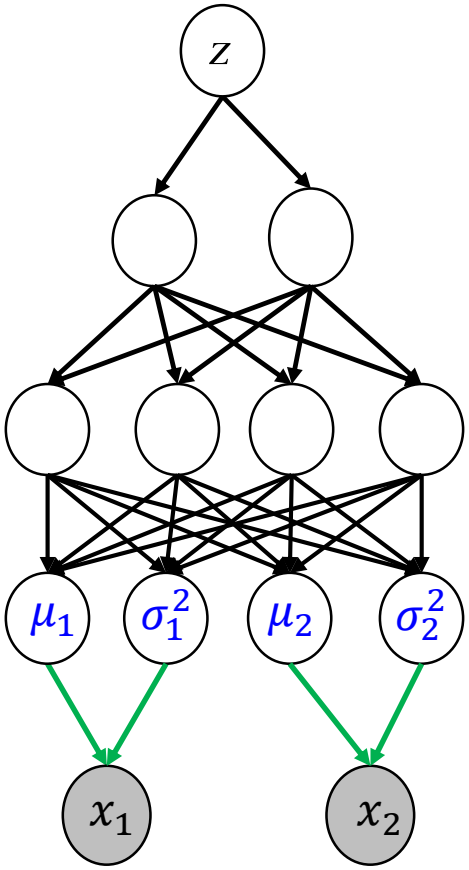
a horse sitting on the grass

$$z \sim p(z|\alpha) \qquad x \sim p(x|z,\beta)$$

- Many variants by combining different building blocks

An example with MLP

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



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

θ

: deterministic function

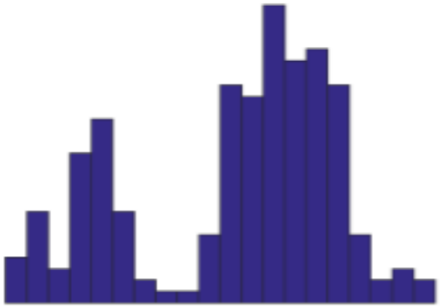
: random sampling

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

Learning Deep Generative Models

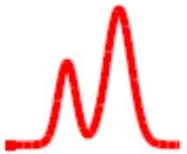
- **Given** a set *D* of unlabeled samples, learn the unknown parameters (or a distribution)

data $D=\{x_i\}$



models $p(x|\theta)$

- Find a model that minimizes

$$\mathbb{D}\Big(\underbrace{\text{  }}_{\text{data } \{x_i\}_{i=1}^n}, \underbrace{\text{  }}_{\text{model } p}\Big)$$

Learning Deep Generative Models

- Maximum likelihood estimation (MLE):

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

- Moment-matching:
  - draw samples from p: $\hat{D} = \{\mathrm{y}_i\}_{i=1}^{M}$, where $\mathrm{y}_i \sim p(\mathrm{x}|\theta)$
  - Kernel MMD:

$$\mathcal{L}_{MMD^2} = \left\| \frac{1}{N}\sum_{i=1}^{N}\phi(x_i) - \frac{1}{M}\sum_{j=1}^{M}\phi(y_i) \right\|_{\mathcal{H}}^2$$

    - rich enough to distinguish any two distributions in certain RKHS
- Minimax objective (e.g., GAN)
- Bayesian inference:
  - infer the posterior distribution of parameters

$$p(\theta|D) \propto p_0(\theta)p(D|\theta)$$

Variational Bayes

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

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

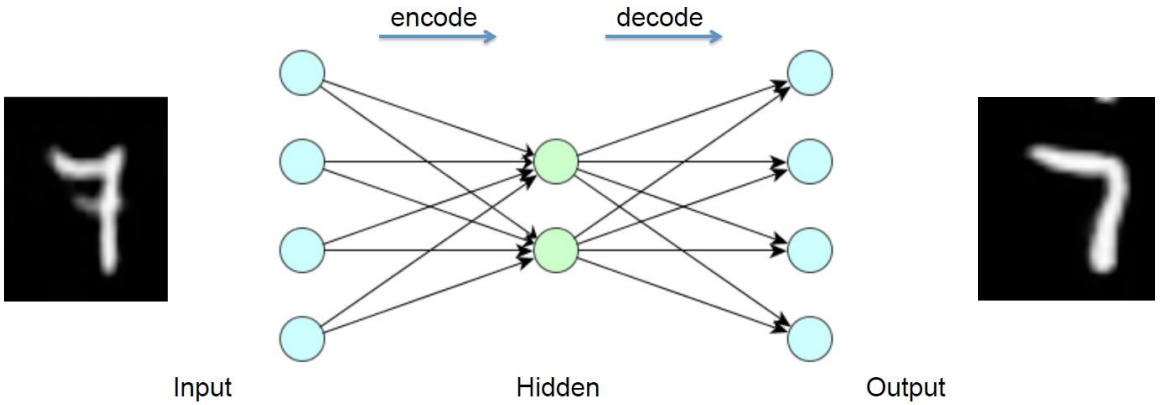
- Log-integral/sum is annoying to handle directly
- Derive a variational lower bound $L(\theta,\phi,\mathrm{x})$

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

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

reconstruction term **prior regularization**

Recap: Auto-Encoder

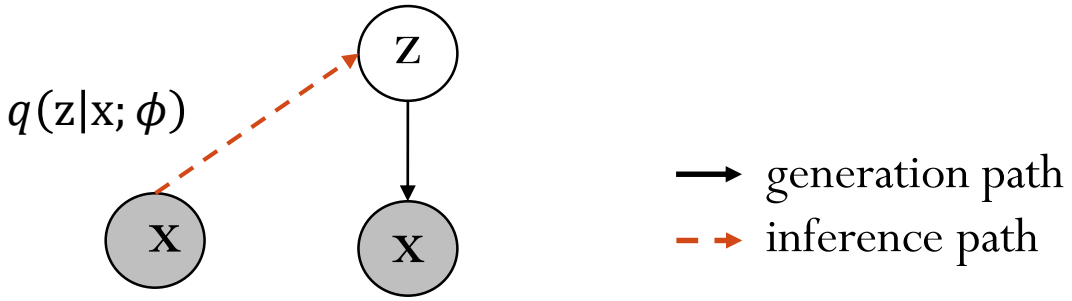


- Encoder: $\mathrm{h} = s(W\mathrm{x} + b)$
- Decoder: $\mathrm{x}' = s(W'\mathrm{h} + b')$
- Training: minimize the reconstruction error (e.g., square loss, cross-entropy loss)
- Denoising AE: randomly corrupted inputs are restored to learn more robust features

Auto-Encoding Variational Bayes (AEVB)

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

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



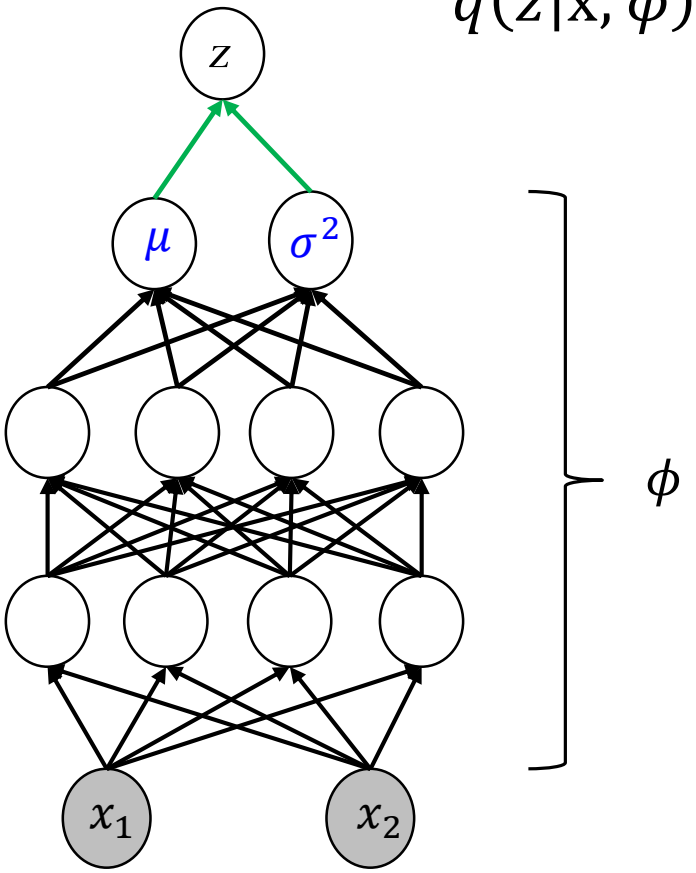
- We call it an inference (recognition, encoder) network or a Q-network
- All the parameters are learned jointly via SGD with variance reduction

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

The Encoder Network

- A feedforward NN + Gaussian

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

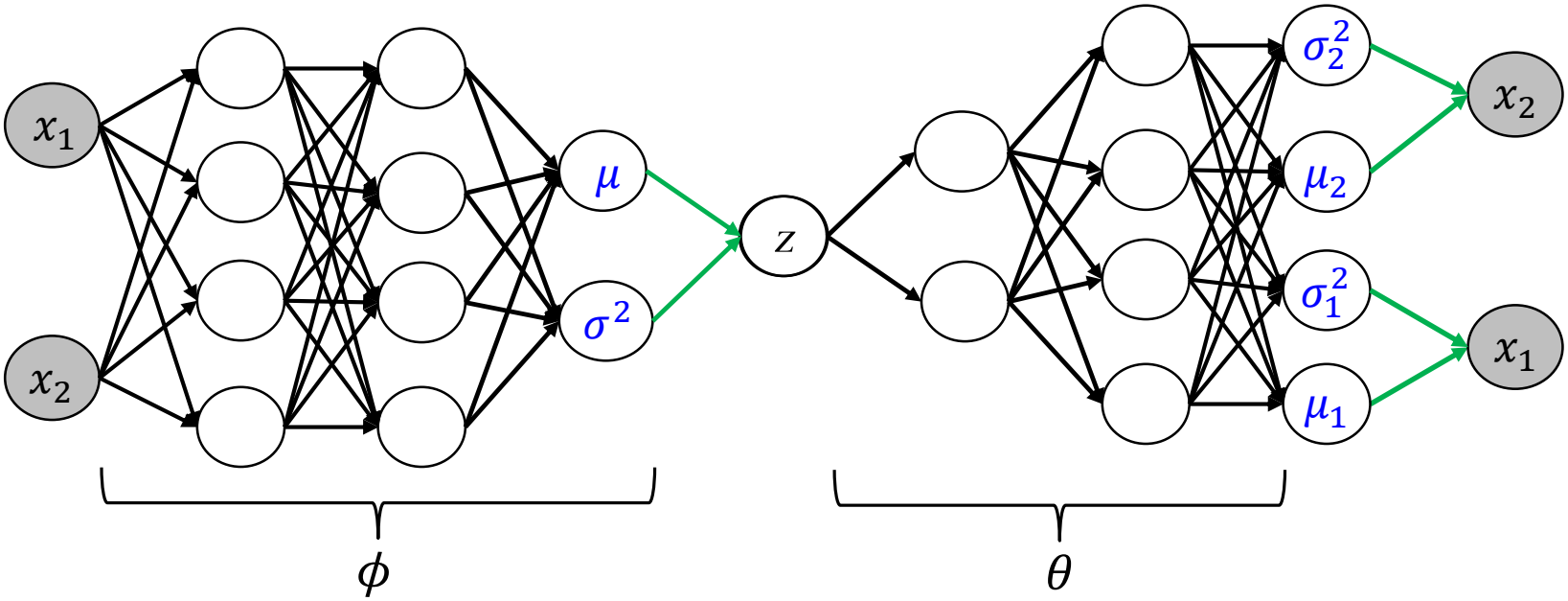


: deterministic function

: random sampling

The Complete Auto-encoder

- The Q-P network architecture:



: deterministic function

: random sampling

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***

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

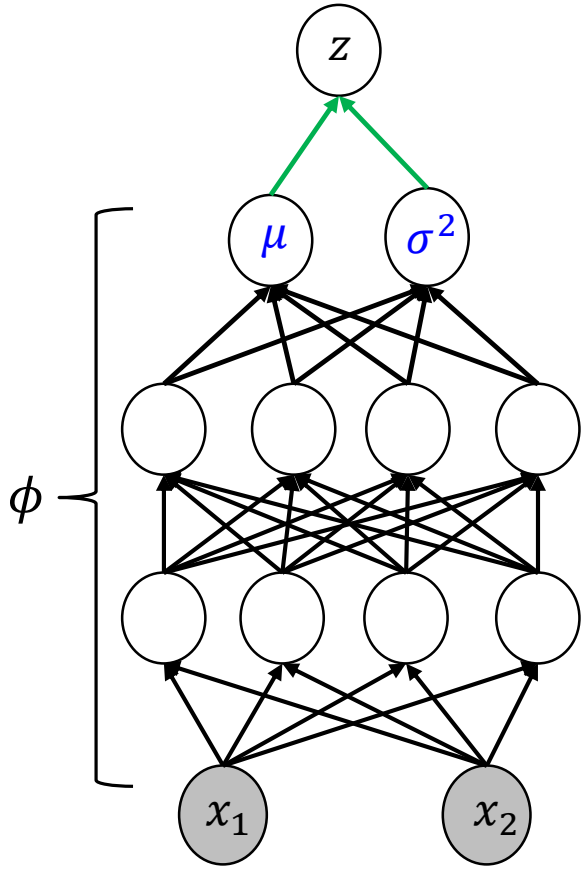
  - Use stochastic gradient methods to handle large datasets
  - Random mini-batch
    - for each i, infer the posterior $q(\mathrm{z}_i|\mathrm{x}_i;\phi)$; As we parameterize as a neural network, this in fact optimizes ϕ

- *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

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



  - **Exercise**: finish the derivation

Example with Gaussian Distributions

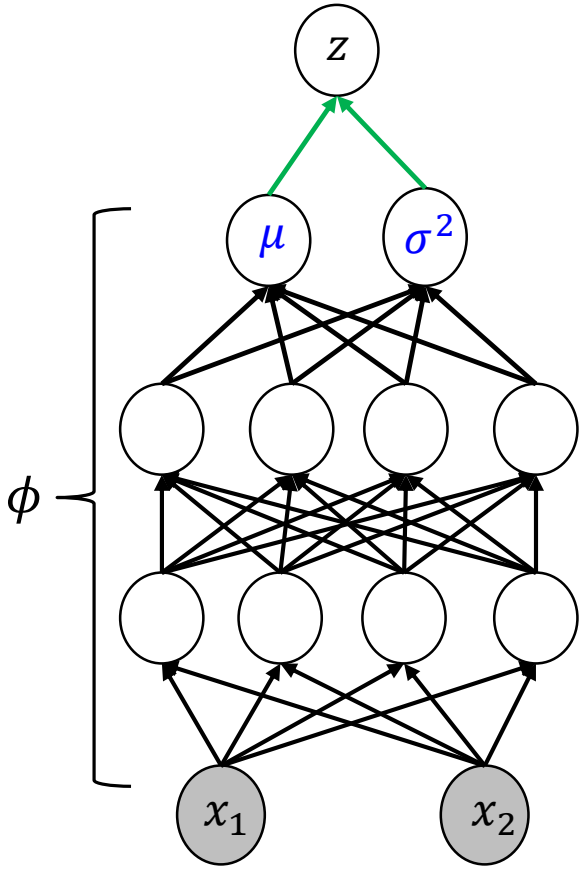
- Use $N(0,1)$ as prior for z; $q(\mathrm{z}|\mathrm{x};\phi)$ is Gaussian with parameters $(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$ determined by NN
  - The expected log-likelihood

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

  - 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

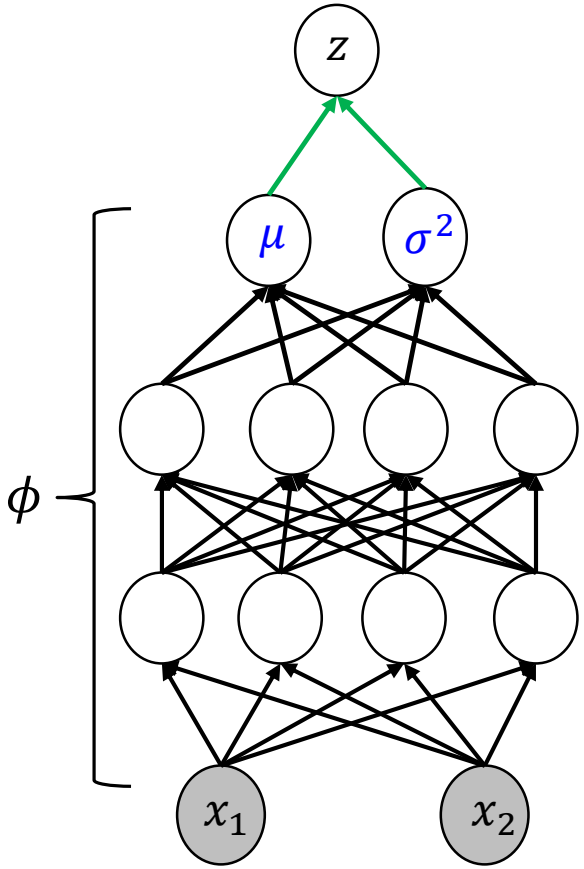
$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|z;\theta)]$$

  - 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):

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

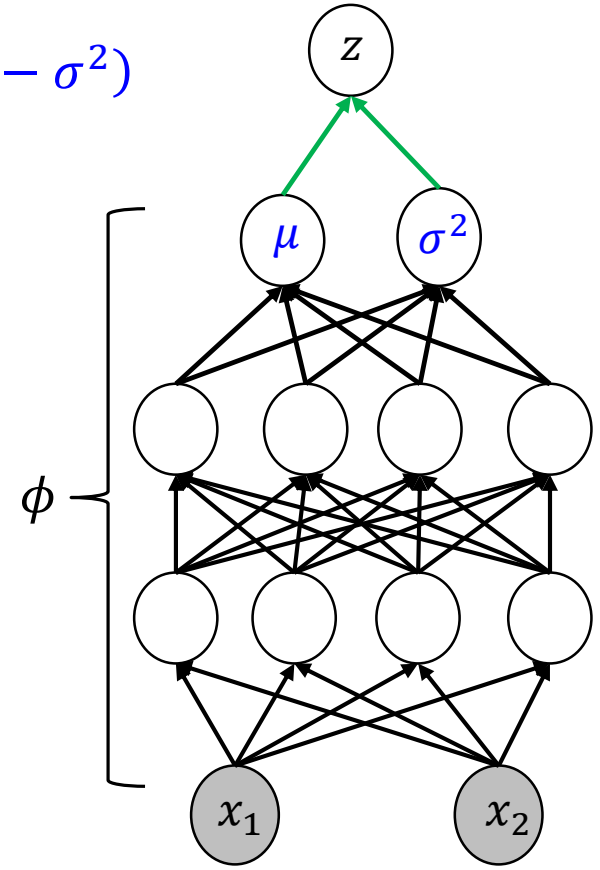
  - Easy to calculate gradient

- The expected log-likelihood term (MC estimate)

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

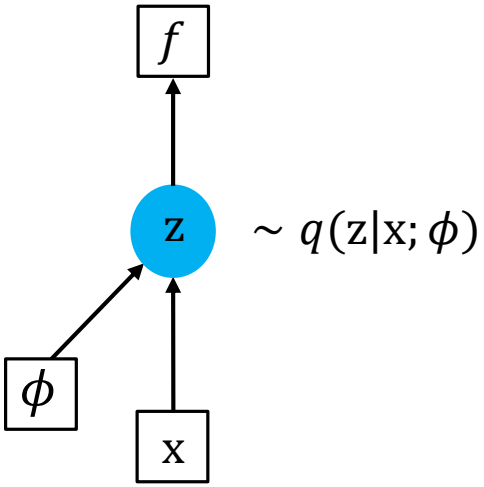
$$z^{(k)} \sim q(z|\mathrm{x};\phi)$$

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

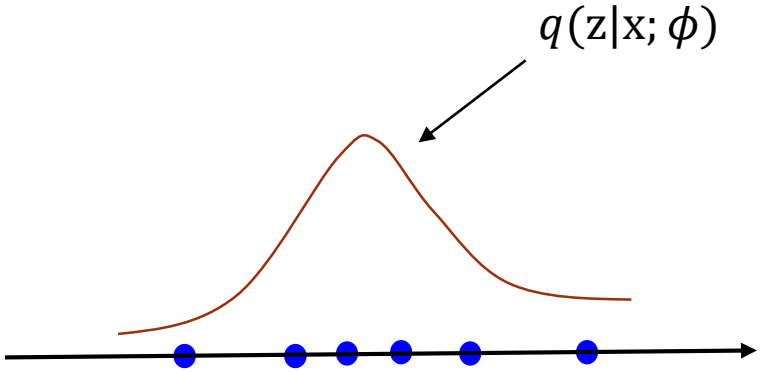


Reparameterization Trick

- Backpropagation not possible through random sampling



□ : deterministic node ● : random node



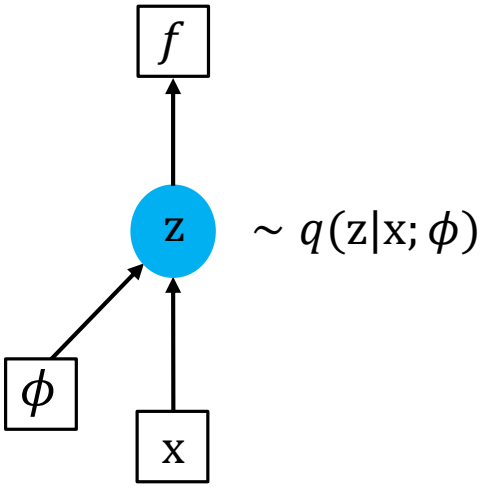
$$z^{(k)} \sim N\left(\mu(\mathrm{x},\phi), \sigma^2(\mathrm{x},\phi)\right)$$

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

Cannot back-propagate through a randomly drawn number

Reparameterization Trick

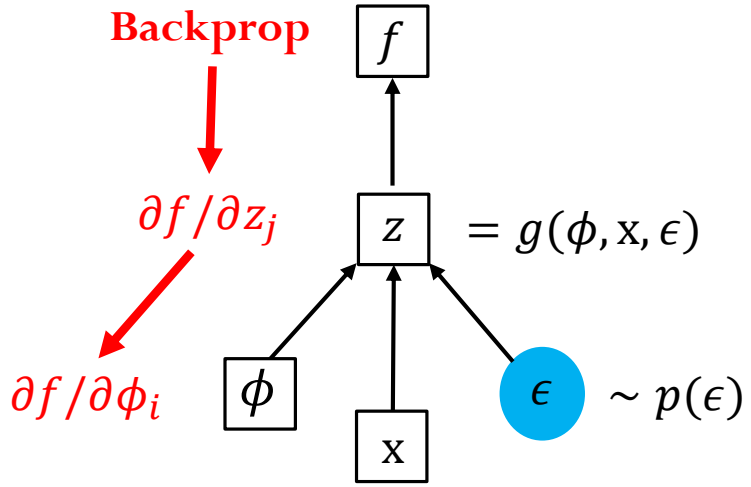
- Backpropagation not possible through random sampling



□ : deterministic node ● : random node

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

Cannot back-propagate through a randomly drawn number



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

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

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:

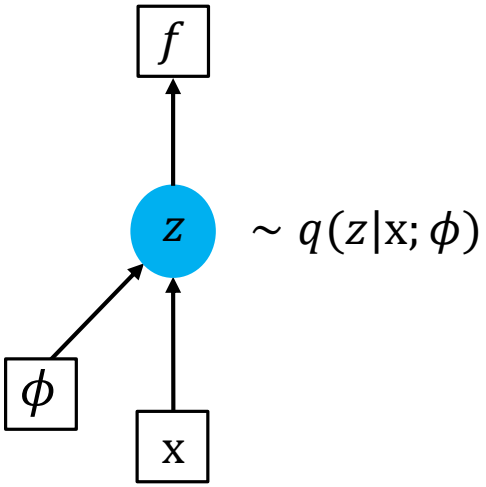
$$L(\theta, \phi, \mathrm{x}) \approx \frac{1}{L}\sum_{k} \log \frac{p\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)}{q\left(\mathrm{z}^{(k)}|\mathrm{x}; \phi\right)}$$

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

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

Reparameterization Trick

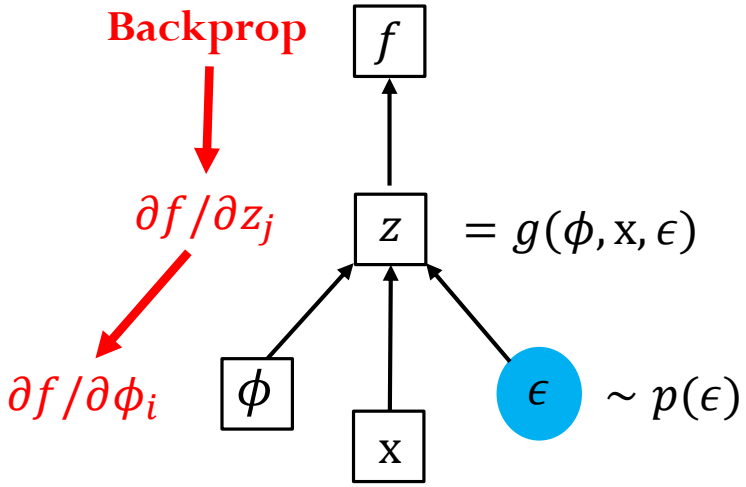
- Backpropagation not possible through random sampling



□ : 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)

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

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:

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

Homework (*): prove these properties

[Burda et al., arXiv, 2015]

Reparametrization Trick

- The IWAE bound:

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

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

- Reparameterization form:

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

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

  - The gradient can be calculated as in VAE

Generative Adversarial Networks (GAN)

- A game between two players:
  - A discriminator *D*
  - A generator *G*

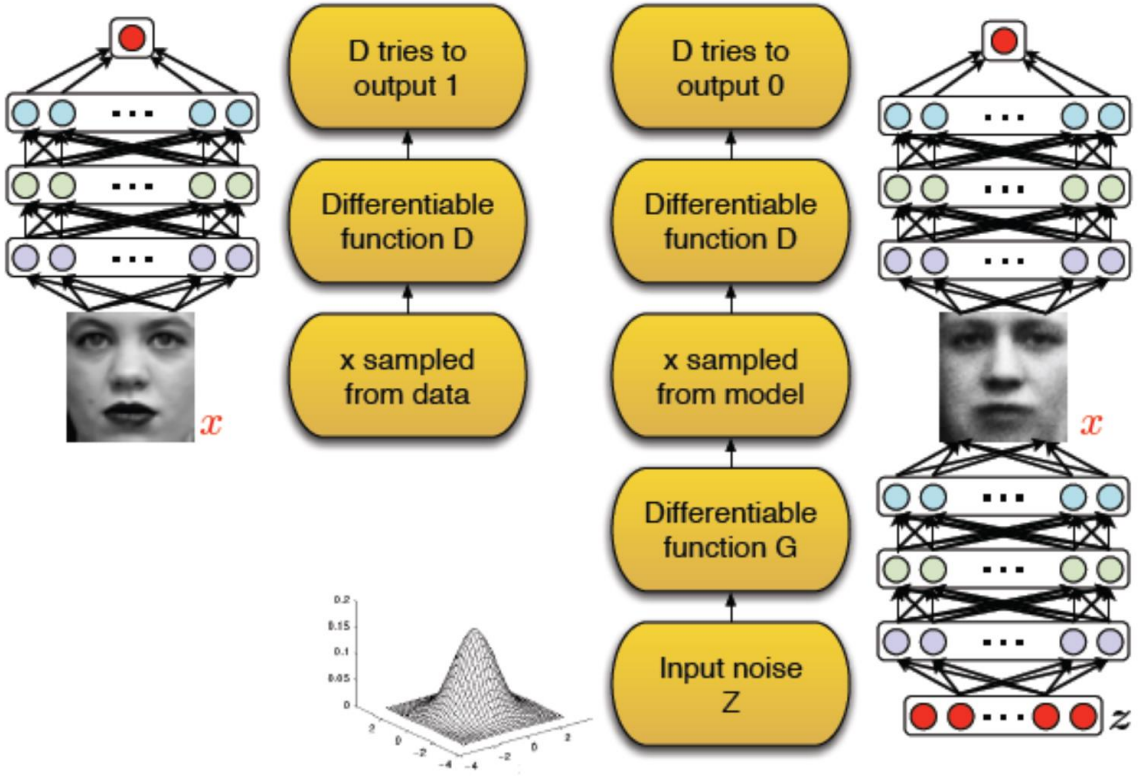
- *D* tries to discriminate between:
  - A sample from the data distribution.
  - And a sample from the generator *G*.

- *G* tries to "trick" *D* by generating samples that are hard for *D* to distinguish from data.

[Goodfellow et al., NIPS 2014]

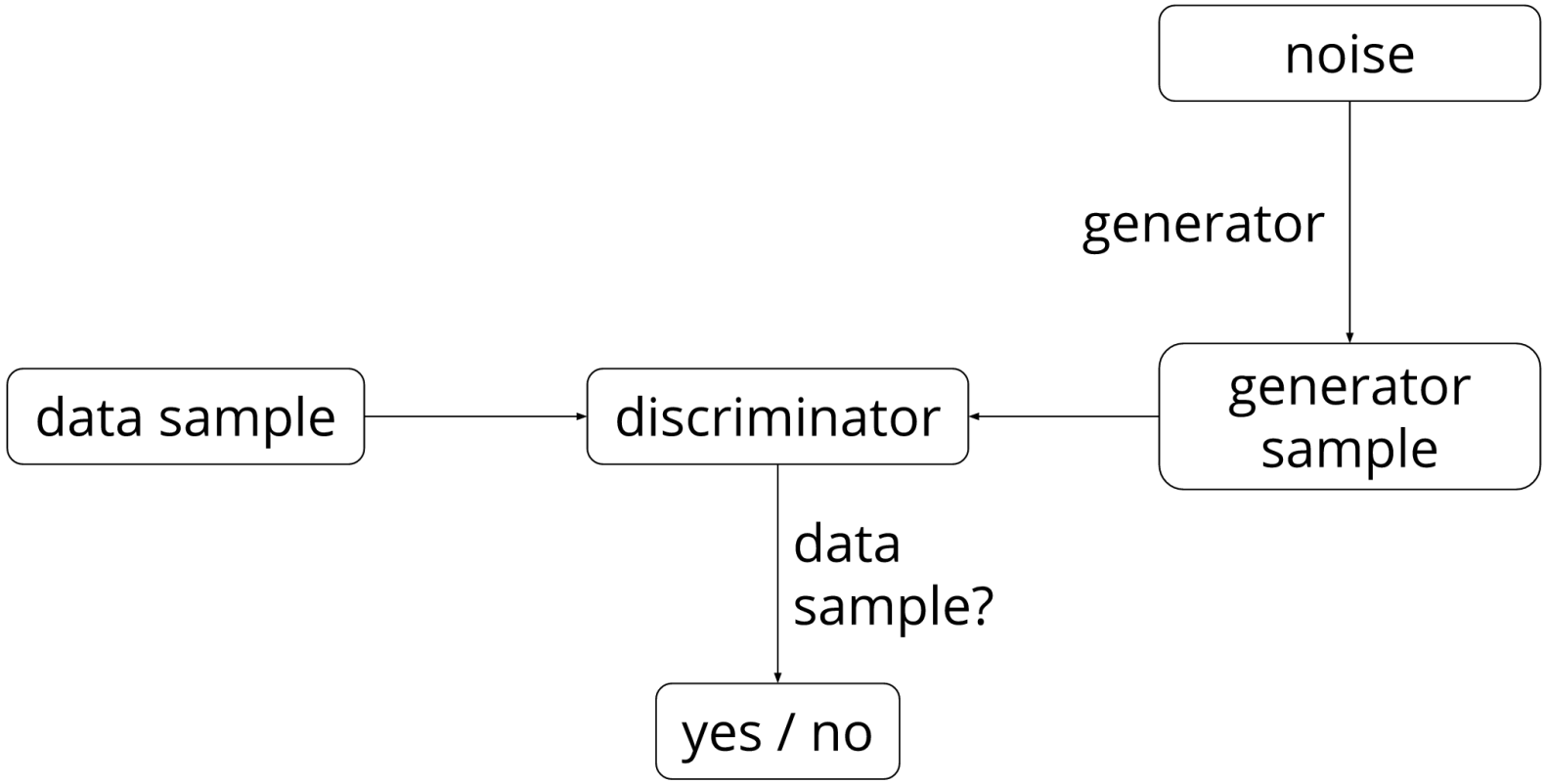
GAN – architecture

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



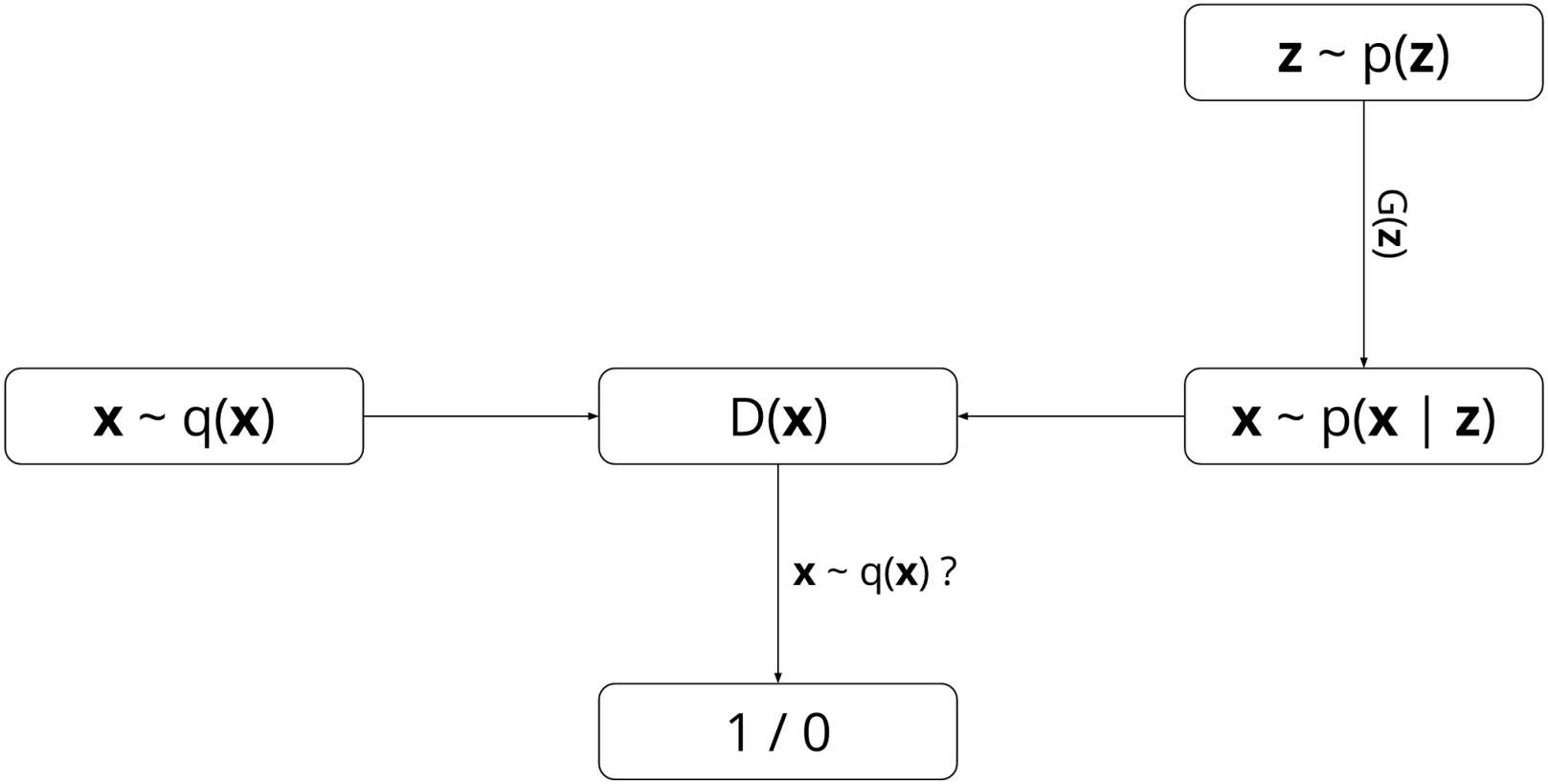
[figure courtesy: Ian Goodfellow]

GAN – architecture



GAN – architecture

- Probabilistic view



GAN – objective

- Minimax objective function

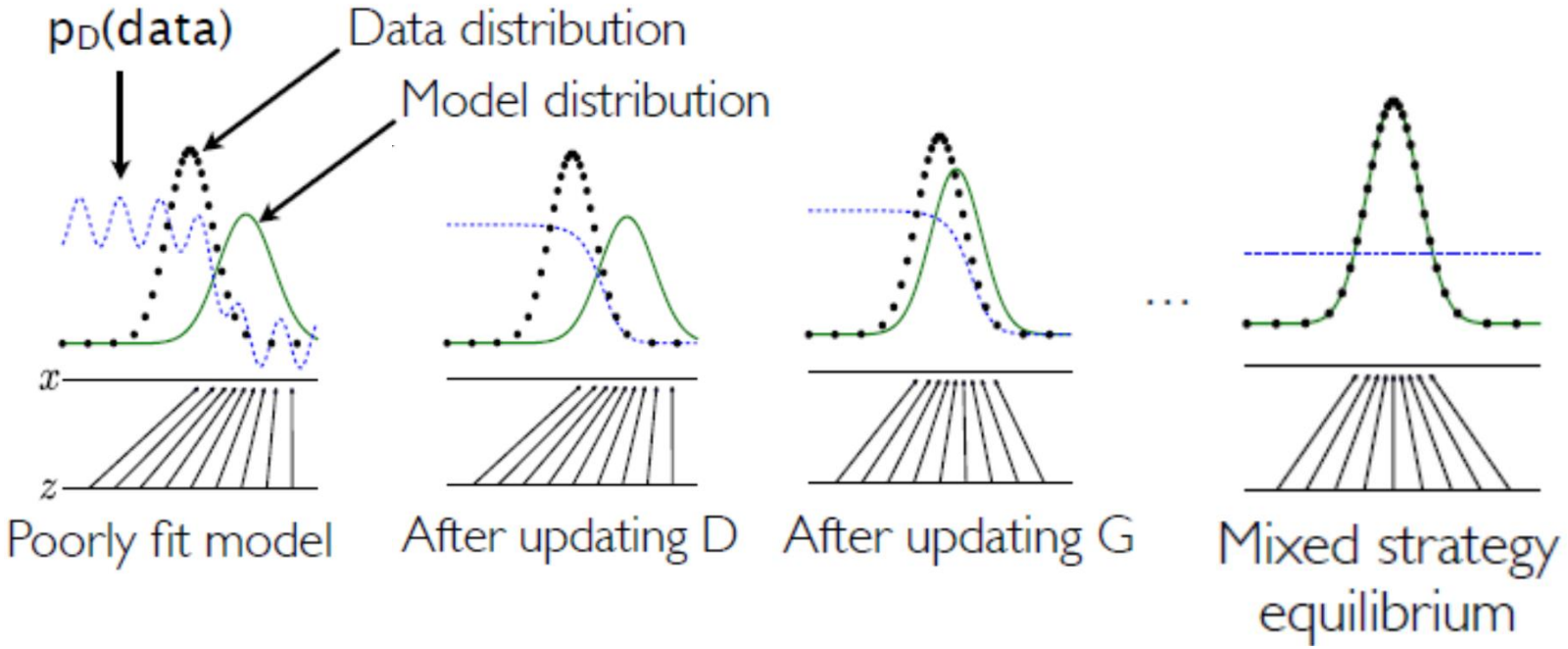
$$\min_G \max_D \; \mathbf{E}_{p_{\text{data}}(\mathrm{x})}[\log D(\mathrm{x})] + \mathbf{E}_{p(\mathrm{z})}\left[\log\left(1 - D\left(G(\mathrm{z})\right)\right)\right]$$

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

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

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

GAN – learning process

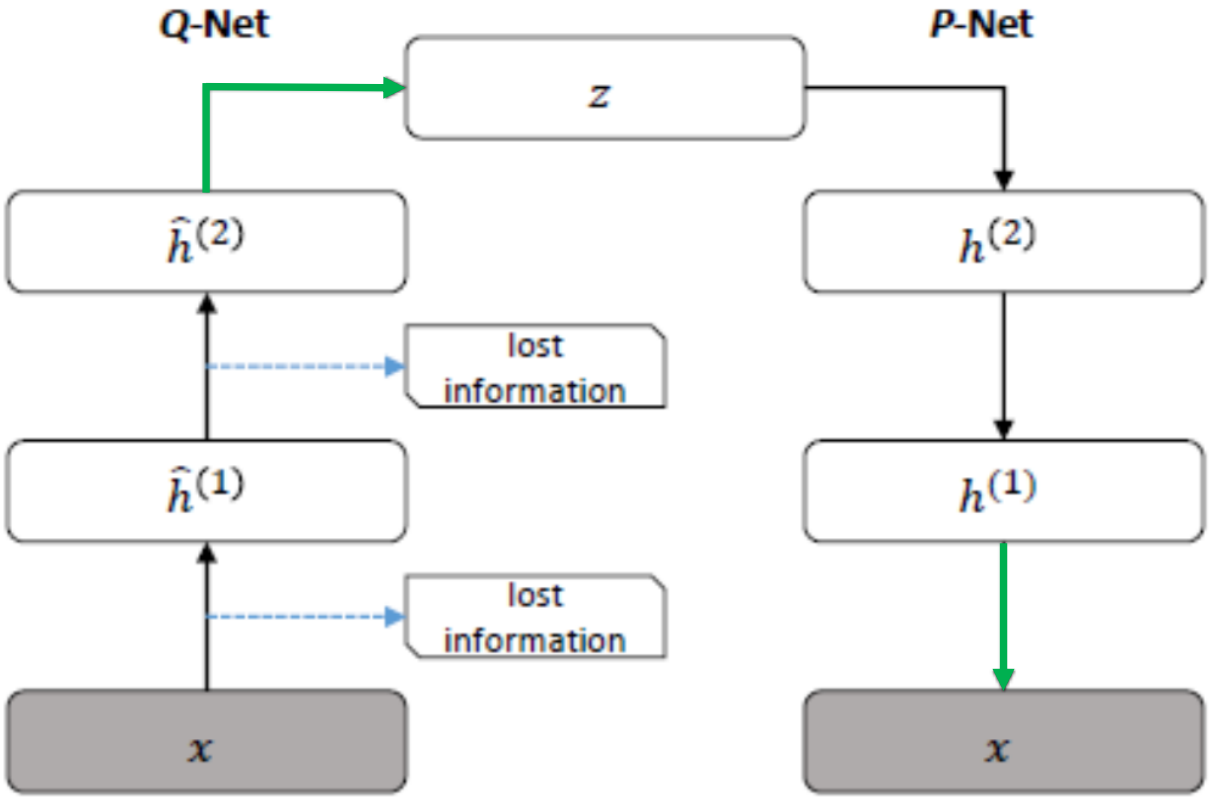


- An adversarial pair near convergence: D is partially accurate
- Update discriminator distribution *D* according to the optimal strategy
- Update generator distribution to be more similar to data distribution
- Iterate until convergence

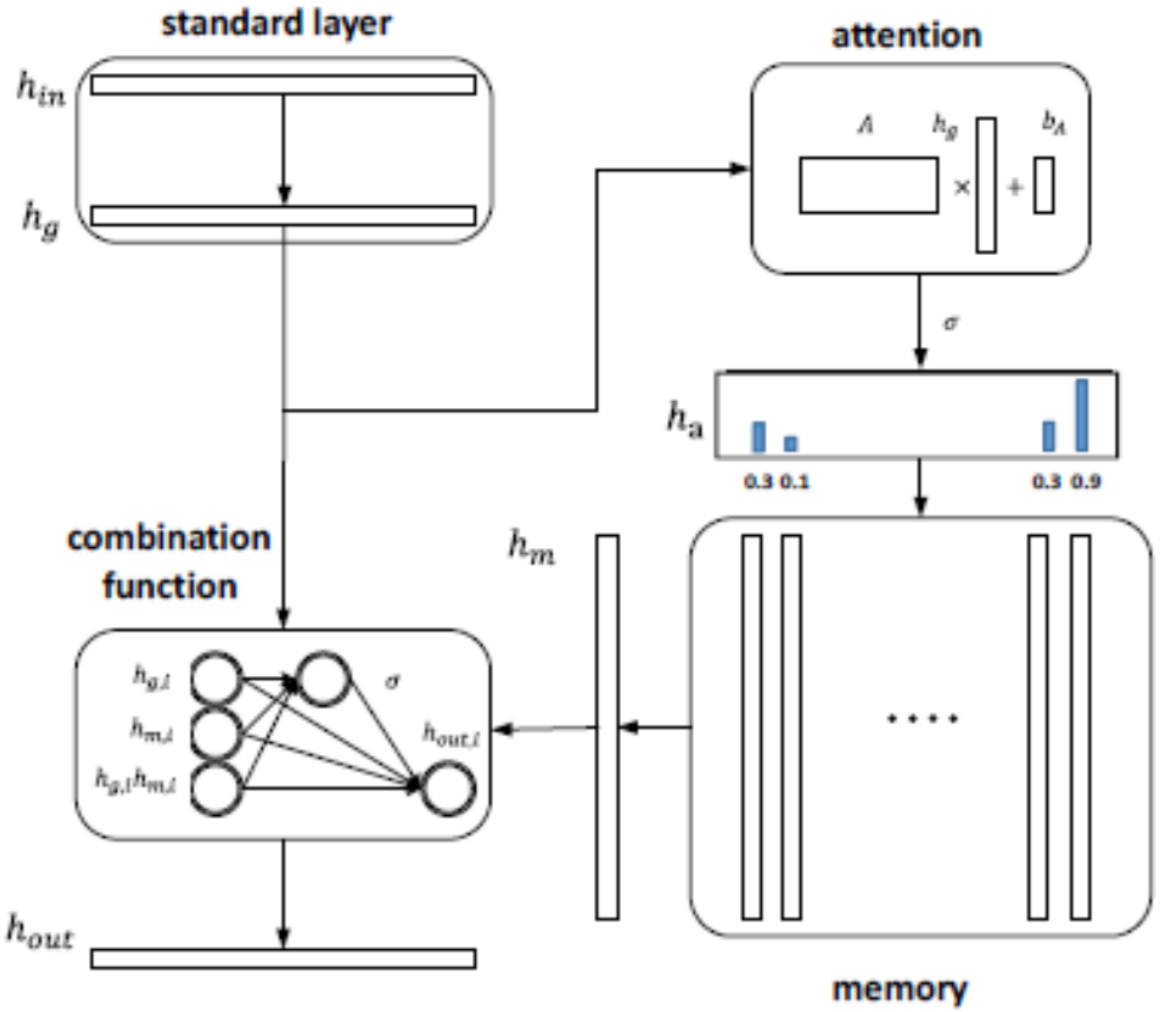
Some extensions

Symmetric Q-P Network

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



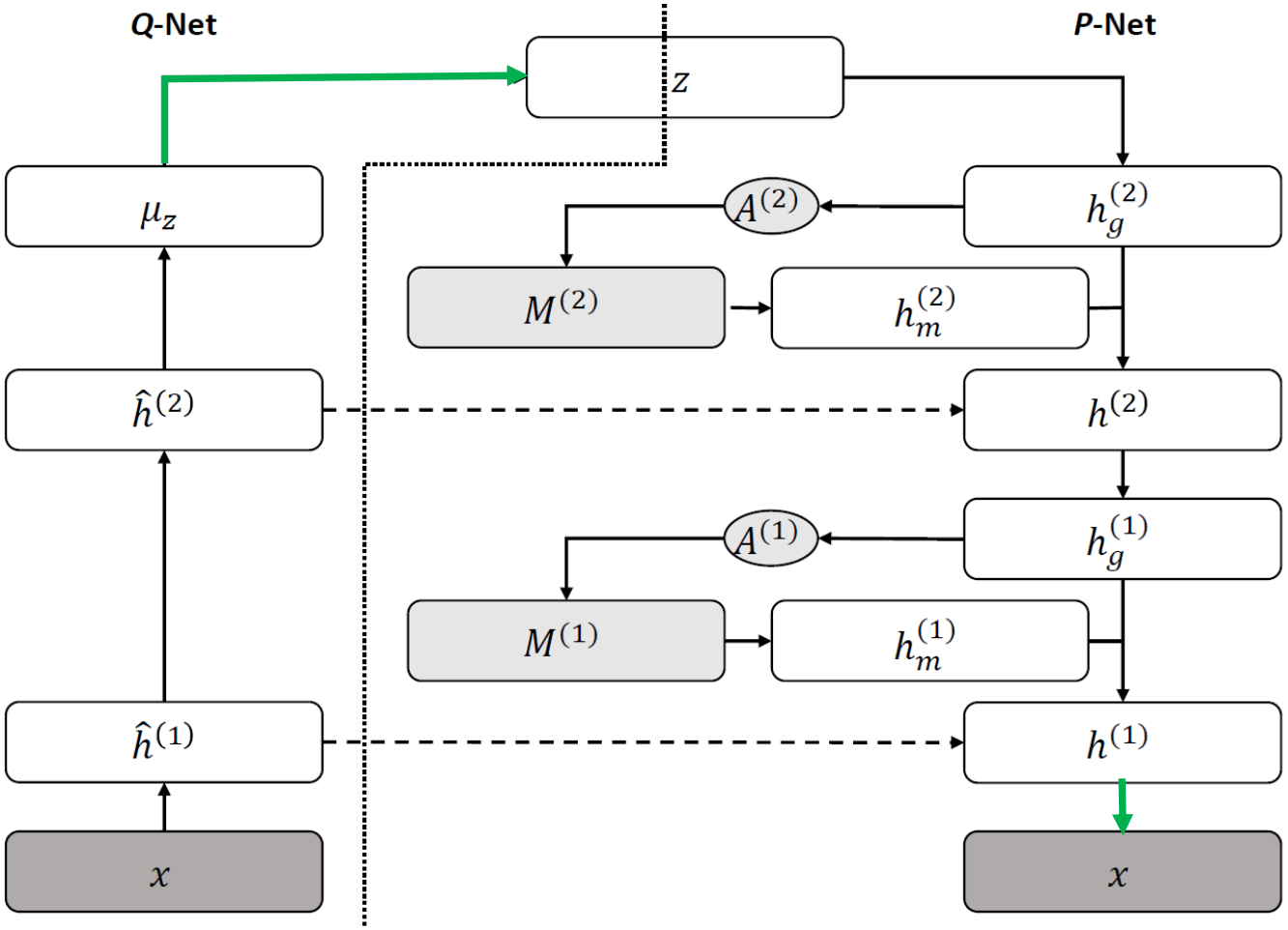
A Layer with Memory and Attention



[Li, et al., ICML 2016]

A Stacked Deep Model with Memory

- Asymmetric architecture



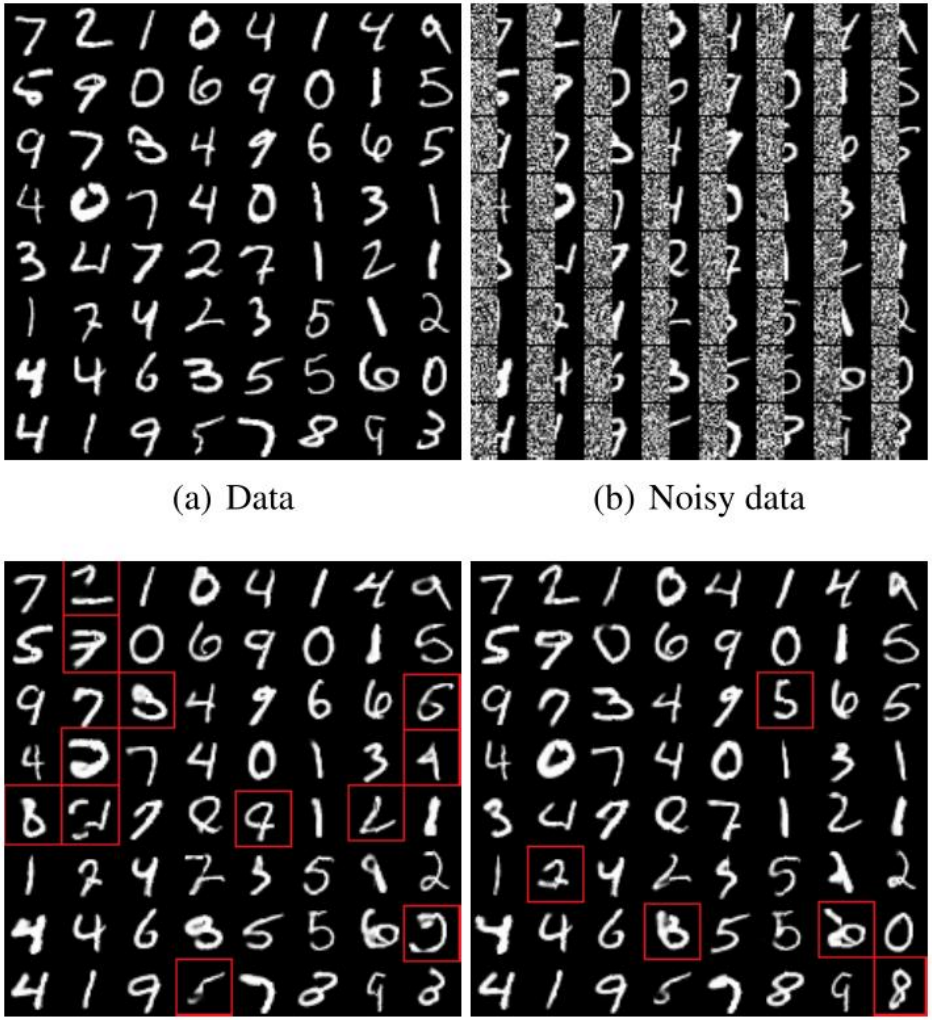
Some Results

- 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

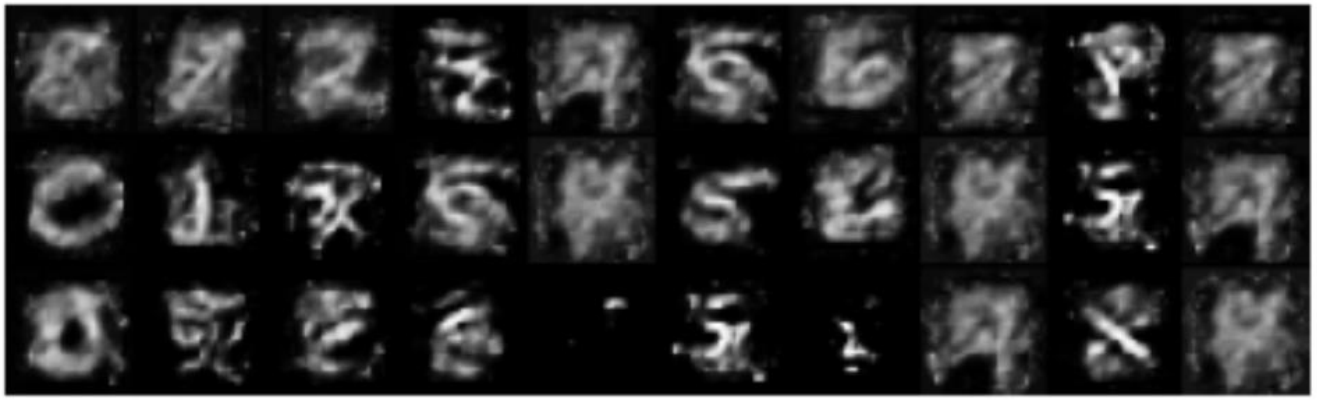
(c) Results of VAE (d) Results of MEM-VAE

Learnt Memory Slots

- Average preference over classes of the first 3 slots:

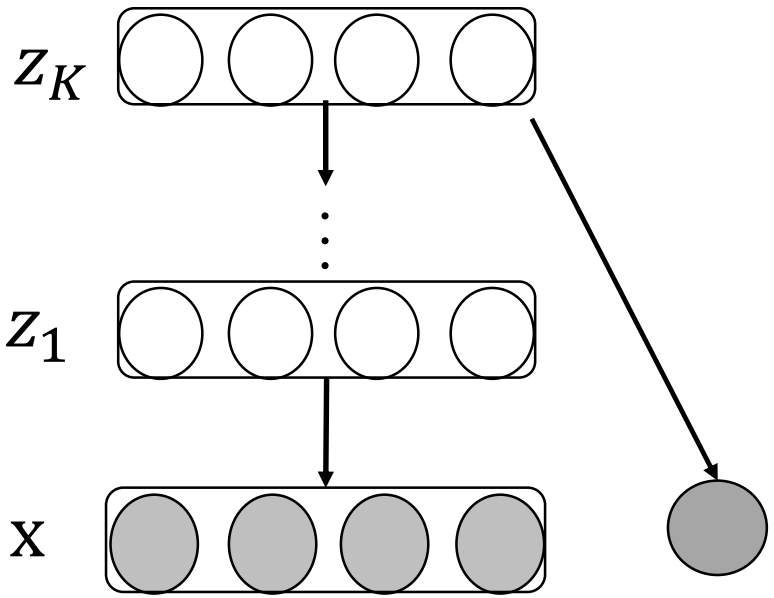
"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
0.27	0.82	0.33	0.11	0.34	0.15	0.49	0.27	0.09	0.28
0.24	0.09	0.06	0.11	0.30	0.13	0.12	0.27	0.09	0.21
0.18	0.05	0.06	0.11	0.07	0.07	0.05	0.11	0.09	0.18

- Corresponding images:



Learning Problem

- A supervised deep generative model



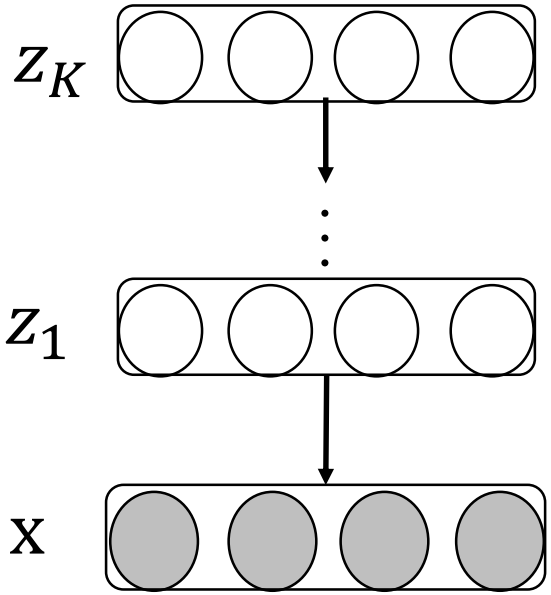
$z \sim p(z|\alpha)$

$x \sim p(x|z,\beta)$

$$\min_{\theta, q(\eta, \mathbf{Z})} \mathcal{L}(\theta, q(\eta, \mathbf{Z}); \mathbf{X}) + C\mathcal{R}(q(\eta, \mathbf{Z}; \mathbf{X}))$$

Variational Objective

- Unsupervised data fitting

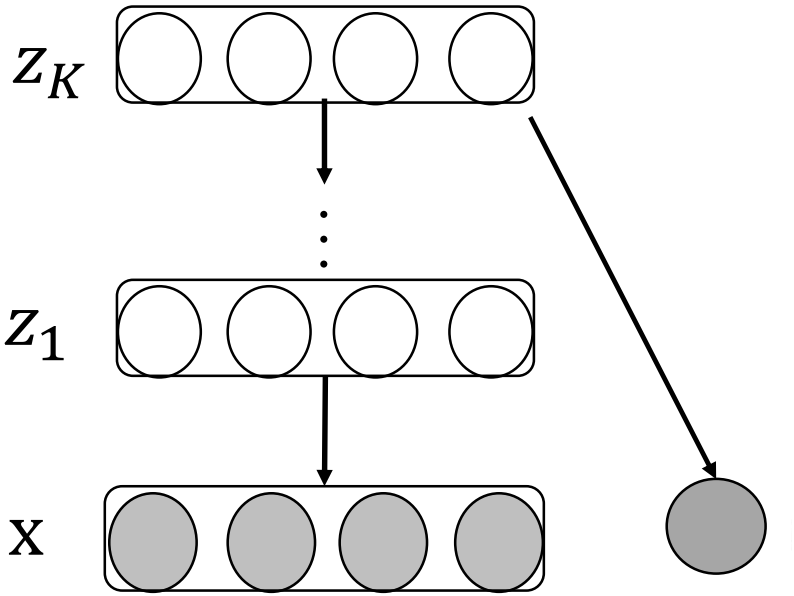


An upper bound of the per sample NLL $-\log p(\mathbf{x}_n|\theta)$:

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

Classification Loss

- Hinge-loss of an averaging classifier



$$f(y; \mathrm{x}) = \mathbf{E}_q[\eta_y^{\mathrm{T}} \mathbf{z}]$$

$$R(q(\eta, \mathbf{Z}); \mathbf{X}) = \sum_{n=1}^{N} \max_y (\Delta \ell_n(y) + f(y; \mathrm{x}_n) - f(y_n; \mathrm{x}_n))$$

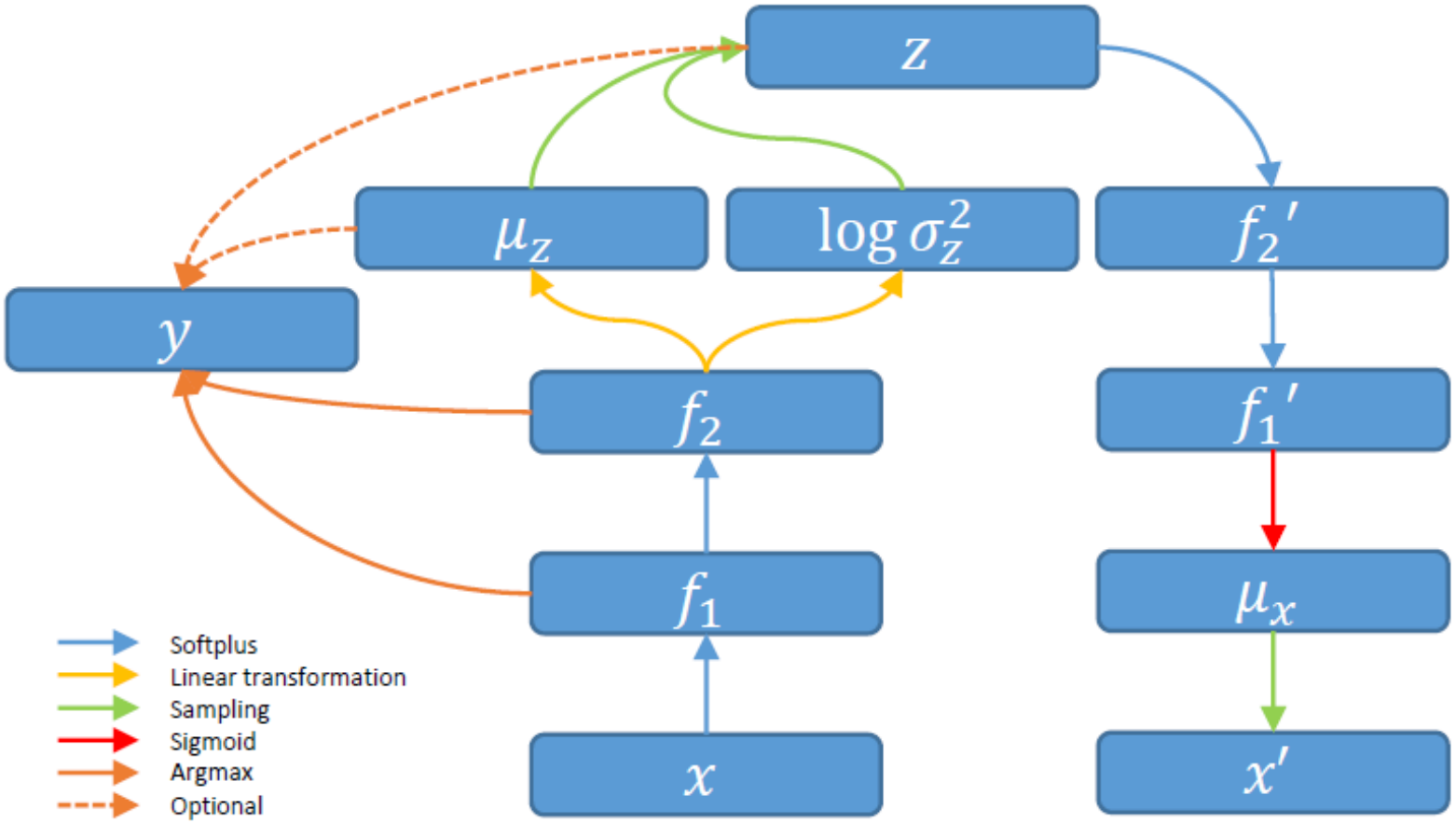
Doubly stochastic subgradient descent

- A doubly stochastic generalization of Pegasos (Shalev-Shwartz et al., 2011)
- Stochastic unbiased estimate of the objective by random mini-batches:

$$\tilde{\mathcal{L}}_m := \frac{N}{m}\sum_{n=1}^{m}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}_n) + \frac{||\boldsymbol{\lambda}||^2}{2\sigma^2} + \frac{NC}{m}\sum_{n=1}^{m}\ell(\boldsymbol{\lambda},\boldsymbol{\phi};\mathbf{x}_n)$$

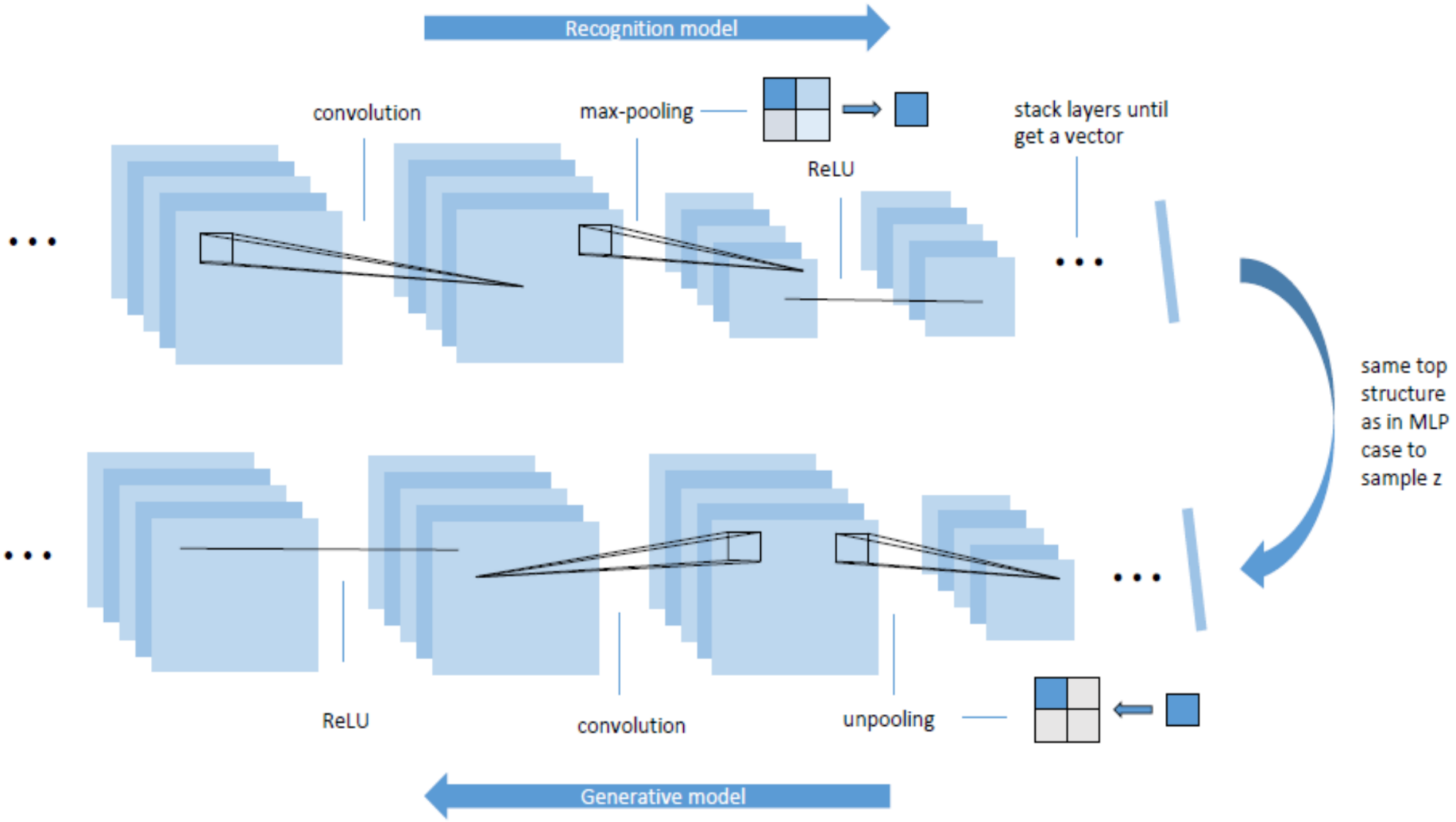
- Stochastic estimate of the per-sample variational bound with a Q-network and its subgradient (unbiased)

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



Classification Performance

- MNIST

Table: Error rates (%) on MNIST dataset.

MODEL	ERROR RATE
VA+Pegasos	1.04
VA+Class-conditionVA	0.96
MMVA	0.90
CVA+Pegasos	1.35
CMMVA	0.45
Stochastic Pooling (ZEILER AND FERGUS [2013])	0.47
Network in Network (LIN ET AL. [2014])	0.47
Maxout Network (GOODFELLOW ET AL. [2013])	0.45
DSN (LEE ET AL. [2015])	0.39

- SVHN

MODEL	ERROR RATE
CVA+Pegasos	25.3
CMMVA	3.09
CNN SERMANET ET AL. [2012]	4.9
Stochastic Pooling ZEILER AND FERGUS [2013]	2.80
Maxout Network GOODFELLOW ET AL. [2013]	2.47
Network in Network LIN ET AL. [2014]	2.35
DSN LEE ET AL. [2015]	1.92

Generative Capability

- MNIST & SVHN:

(a) VA (b) MMVA (c) CVA (d) CMMVA

(a) Training data (b) CVA (c) CMMVA (d) CMMVA

Imputation examples

Considering Rect(12×12) noise, CMMVA makes fewer mistakes and refines the images better, which accords with the MSE results.

(a) Original data (b) Noisy data (c) CVA (d) CMMVA

Figure: (a): original test data; (b) test data with missing value; (c-d): results inferred by CVA and CMMVA respectively for 100 epochs.

Classification with Missing Values

CNN makes prediction on the incomplete data directly. CVA and CMMVA with default settings infer missing data for 100 iterations at first and then make prediction on the refined data.

Table: Error rates(%) with missing values on MNIST.

Noise Level	CNN	CVA	CMMVA
Rect (6 × 6)	7.5	2.5	1.9
Rect (8 × 8)	18.8	4.2	3.7
Rect (10 × 10)	30.3	8.4	7.7
Rect (12 × 12)	47.2	18.3	15.9

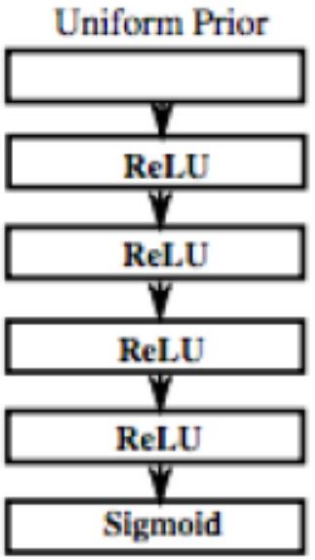
CMMVA outperforms both VA and CNN in this scenario.

Moment-Matching DGMs

- Moment-matching:
  - draw samples from p via DGM:

$$y_i = f(\epsilon_i; \theta), \quad \epsilon_i \sim \text{Uniform}(0,1)$$

$$\widehat{D} = \{y_i\}_{i=1}^{M}$$



  - Kernel MMD:

$$\mathcal{L}_{MMD^2} = \left\| \frac{1}{N}\sum_{i=1}^{N}\phi(x_i) - \frac{1}{M}\sum_{j=1}^{M}\phi(y_i) \right\|_{\mathcal{H}}^2$$

    - Rich enough to distinguish any two distributions in certain RKHS
    - Back-propagation to update the parameters

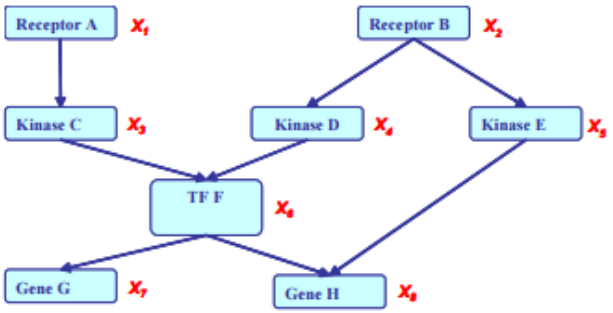
[Li et al., ICML 2015]

Conditional Models

- We're more interested in conditional distributions in many cases
  - Predictive modeling: $p\left(\text{y} = cat \;\middle|\; \text{x} = \right)$ 
    - Better performance with fewer training samples
  - Contextual generation:

$$p\left(\text{y} = \right)$$  $$\;\middle|\; \text{x} = cat\ with\ glass\ and\ tie\right)$$

  - Building large networks:



Conditional Moment-Matching Networks

- Two sets of data samples (one from training set, one from the mdoel)

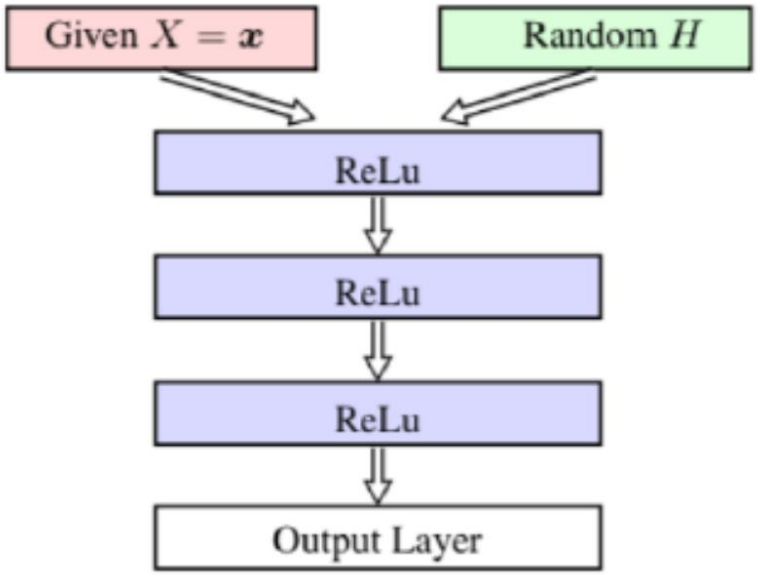
$$\mathcal{D}^s_{XY} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$$

$$\mathcal{D}^d_{XY} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^M$$

- A DGM:

$$p(\mathbf{h}) = \prod_{i=1}^{d} U(h_i)$$

$$\mathbf{y} = f(\mathbf{x}, \mathbf{h}|\mathbf{w})$$



[Ren et al., NIPS 2016]

Conditional Moment-Matching Networks

- We define conditional max-min discrepancy (CMMD):

$$\begin{aligned}\widehat{\mathcal{L}}^2_{\mathrm{CMMD}} &= \left\| \Phi_d (K_d + \lambda I)^{-1} \Upsilon_d^\top - \Phi_s (K_s + \lambda I)^{-1} \Upsilon_s^\top \right\|^2_{\mathcal{F}\otimes\mathcal{G}} \\ &= \mathrm{Tr}\left(K_d \widetilde{K}_d^{-1} L_d \widetilde{K}_d^{-1} \right) + \mathrm{Tr}\left(K_s \widetilde{K}_s^{-1} L_s \widetilde{K}_s^{-1} \right) \\ &\quad - 2 \cdot \mathrm{Tr}\left(K_{sd} \widetilde{K}_d^{-1} L_{ds} \widetilde{K}_s^{-1} \right),\end{aligned}$$

- the superscripts s and d denote the two sets of samples
- $\widetilde{K} = K + \lambda I$
- $\Phi_d := (\phi(\mathbf{y}_1^d), ..., \phi(\mathbf{y}_N^d))$, $\Upsilon_d := (\phi(\mathbf{x}_1^d), ..., \phi(\mathbf{x}_N^d))$; Φ_s, Υ_s: similarly for $\mathcal{D}^s_{XY}$.
- $K_d = \Upsilon_d^\top \Upsilon_d, K_s = \Upsilon_s^\top \Upsilon_s$: gram matrices for input variables; $L_d = \Phi_d^\top \Phi_d, L_s = \Phi_s^\top \Phi_s$:gram matrices for output variables
- $K_{sd} = \Upsilon_s^\top \Upsilon_d, L_{ds} = \Phi_d^\top \Phi_s$:gram matrices between the two datasets on input and output variables.

[Ren et al., NIPS 2016]

Connections with MMD

- MMD:

$$\mathcal{L}^2_{MMD} = \mathrm{Tr}(L_d \cdot \mathbf{1}) + \mathrm{Tr}(L_s \cdot \mathbf{1}) - 2 \cdot \mathrm{Tr}(L_{ds} \cdot \mathbf{1}),$$

where $\mathbf{1}$ is the matrix with all entities equal to 1.

- CMMD:

$$\mathcal{L}^2_{CMMD} = \mathrm{Tr}(L_d \cdot C_1) + \mathrm{Tr}(L_s \cdot C_2) - 2 \cdot \mathrm{Tr}(L_{ds} \cdot C_3),$$

where C_1, C_2, C_3 are some matrix based on the training and generated data.

Instead of putting uniform weights on the gram matrix, CMMD applies non-uniform weight, reflecting the influence of conditional variables.

[Ren et al., NIPS 2016]

Mini-batch Training Algorithm

- BP to compute the gradients:

Input: Dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$
Output: Learned parameters $\mathbf{w}$
Randomly divide training dataset $\mathcal{D}$ into mini batches
While Stopping criterion not met **do**
  Draw a minibatch $\mathcal{B}$ from $\mathcal{D}$;
  For each $\mathbf{x} \in \mathcal{B}$, generate a $\mathbf{y}$; and set $\mathcal{B}'$ to contain all the generated $(\mathbf{x}, \mathbf{y})$;
  Compute the gradient $\frac{\partial \widehat{\mathcal{L}}^2_{\mathrm{CMMD}}}{\partial \mathbf{w}}$ on $\mathcal{B}$ and $\mathcal{B}'$;
  Update $\mathbf{w}$ using the gradient with proper regularizer.
EndWhile

[Ren et al., NIPS 2016]

Some Results – Classification

Classification accuracy on MNIST:

Model	Error Rate
VA+Pegasos	1.04
MMVA	0.90
CGMMN	0.97
CVA + Pegasos	1.35
CGMMN-CNN	0.47
Stochastic Pooling	0.47
Network in Network	0.47
Maxout Network	0.45
CMMVA	0.45
DSN	0.39

Classification accuracy on SVHN:

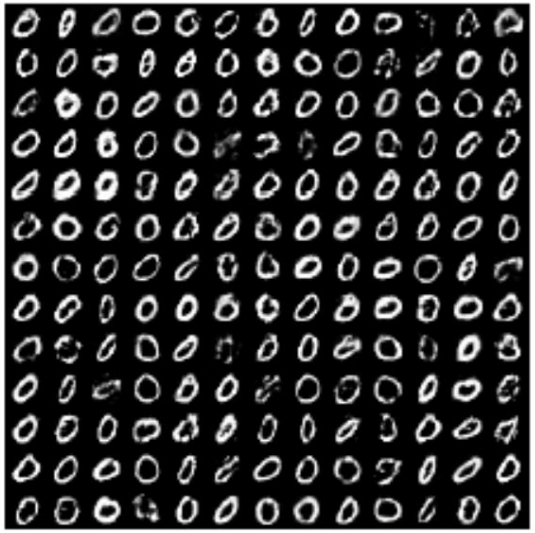
Model	Error Rate
CVA+Pegasos	25.3
CGMMN-CNN	3.13
CNN	4.9
CMMVA	3.09
Stochastic Pooling	2.80
Network in Network	2.47
Maxout Network	2.35
DSN	1.92

Some Results – Generation

Generating performance on MNIST:

(a) MNIST samples

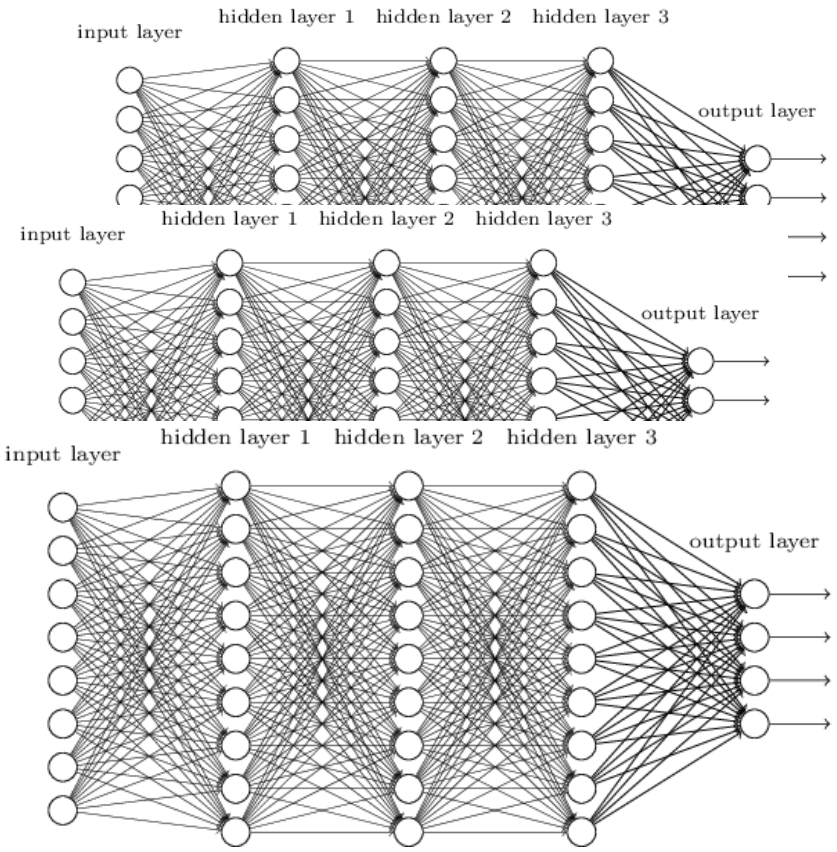
(b) Random CGMMN samples



(c) Samples conditioned on label 0

Some Results – Bayesian Dark Knowledge

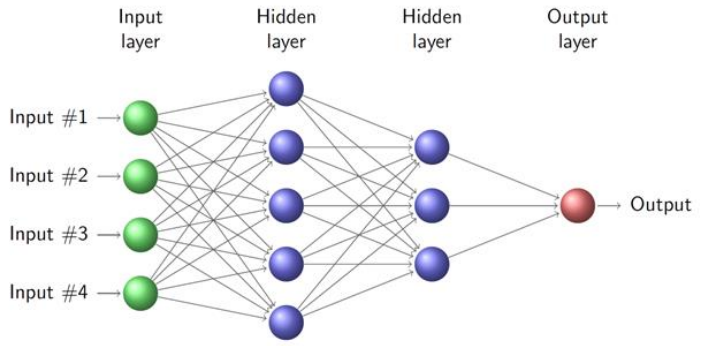
- Distilling the knowledge (Hinto et al., 2015) & Bayesian dark knowledge (Korattikara et al., 2015)



$$p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}$$

Bayes prediction is expensive!

Teach a student network for prediction



Some Results – Bayesian Dark Knowledge

Distilling knowledge in Bayesian networks on Boston housing dataset:

- one-dimensional regression:506 data points where each data is of dimension 13
- distill a PBP model
- test whether the distilled model will degrade the prediction performance.

PBP prediction	Distilled by CGMMN
2.574 ± 0.089	2.580 ± 0.093

- measured by MSE
- MLP network with three hidden layers and $(100, 50, 50)$ hidden units for middle layers.
- $N = 3,000$ sample pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
- $\mathbf{x}_i$ is generated by adding noise into training data to avoid fitting the testing data directly

Summary

- Deep generative models are flexible in density estimation, sampling generation, etc.
  - VAE, GAN, etc.
- Memory and attention are helpful for DGMs
- Supervised/Semi-supervised learning for DGMs with state-of-the-art performance
- Moment-matching networks can learn conditional distributions well

- Code: https://github.com/thu-ml

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
- Li, C., Zhu, J., and Zhang, B. *Max-margin deep generative models*. In NIPS, 2015
- Li, C., Zhu, J., and Zhang, B. *Max-margin deep generative models for (semi-)supervised learning*, arXiv, 2016
- Ren, Y., Li, J., Luo, Y., and Zhu, J. *Conditional generative moment-matching networks*, NIPS, 2016.

Thanks