[80245013 Machine Learning, Fall, 2020]

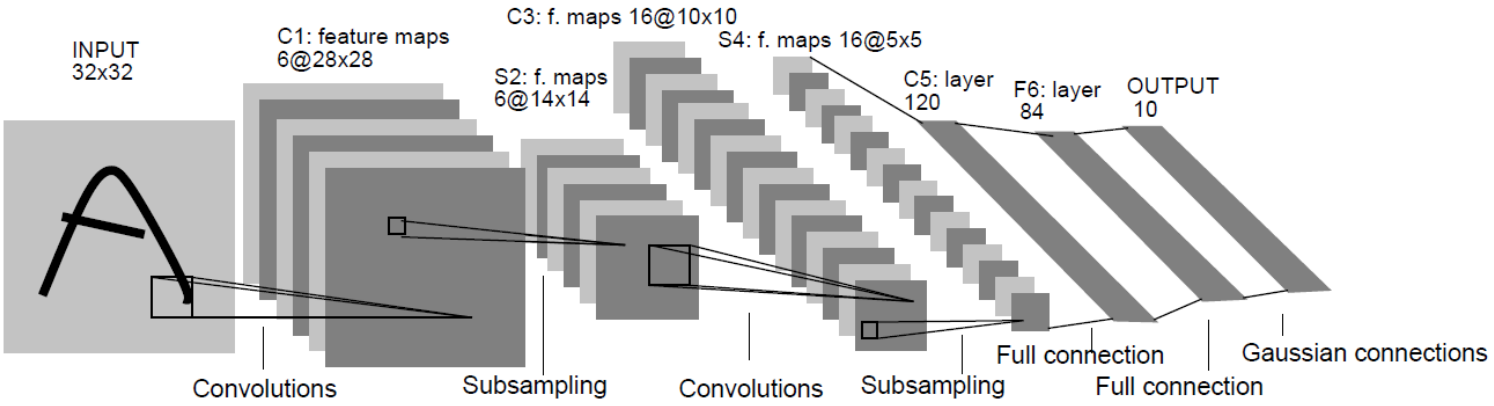
Deep Generative Models

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

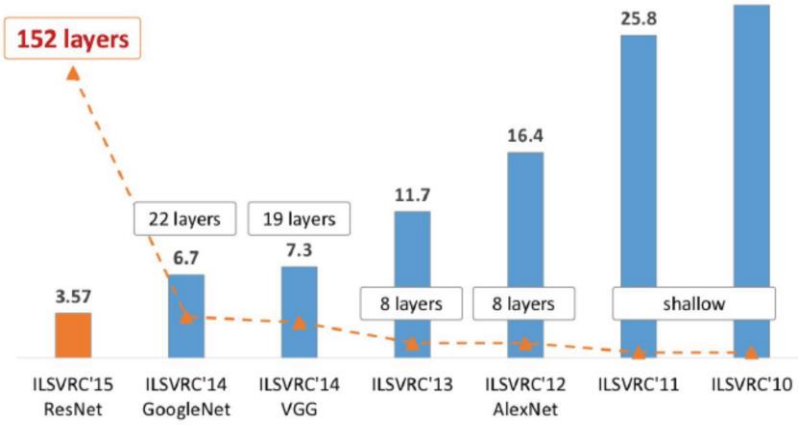
November 3, 2020

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

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

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

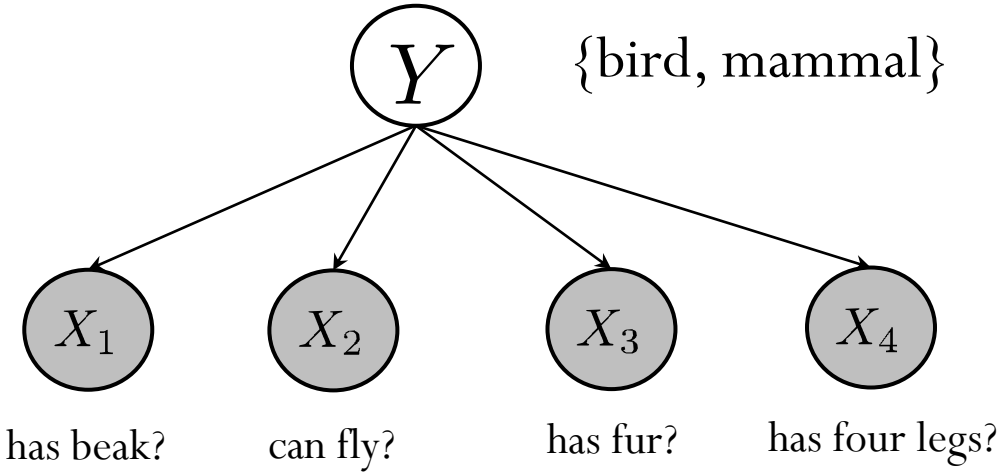
Why generative models?

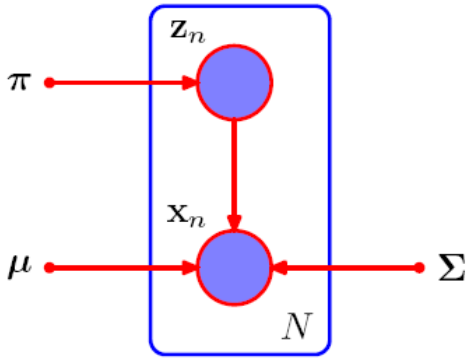
- Leverage unlabeled datasets, which are often much larger than labeled ones
  - Unsupervised learning
    - e.g., clustering, density estimation, feature extraction, dimension reduction, data generation
  - Semi-supervised learning
    - e.g., classification, information extraction, learning-to-rank, network analysis, opinion mining

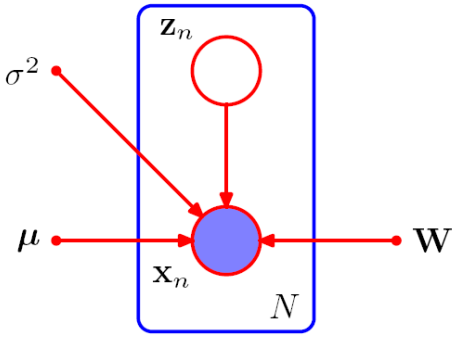
- Conditional generative models
  - Speech synthesis: Text ⇒ Speech
  - Machine Translation: French ⇒ English
  - Image captioning: Image ⇒ Text

Generative models are your old friends

- Naïve Bayes, GMM, Factor Analysis (Probabilistic PCA)







Deep Generative Models

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

- If z is uniformly distributed over (0, 1), then $y = f(z)$ has the distribution

$$p(y) = p(z)\left|\frac{dz}{dy}\right|$$

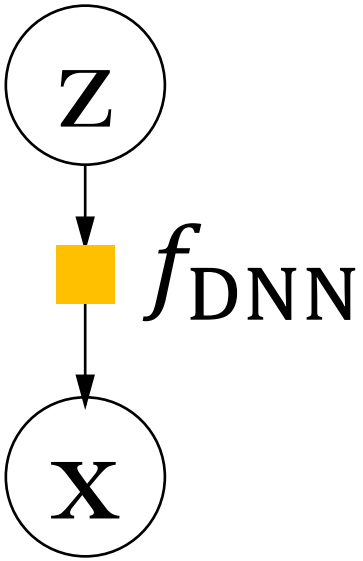
  - where $p(z) = 1$

- This trick is widely used to draw samples from exponential family distributions (e.g., Gaussian, Exponential)

Deep Generative Models

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

- DGMs learn a function transform with deep neural networks



Cause => Disease

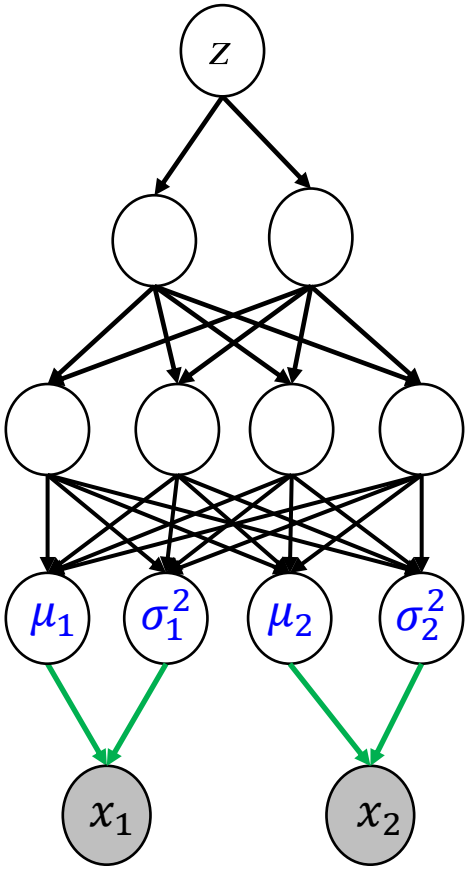
Topics => Docs

Objects => Images

Words => Phonemes

An example with MLP

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



$z \sim N(0,1)$

θ

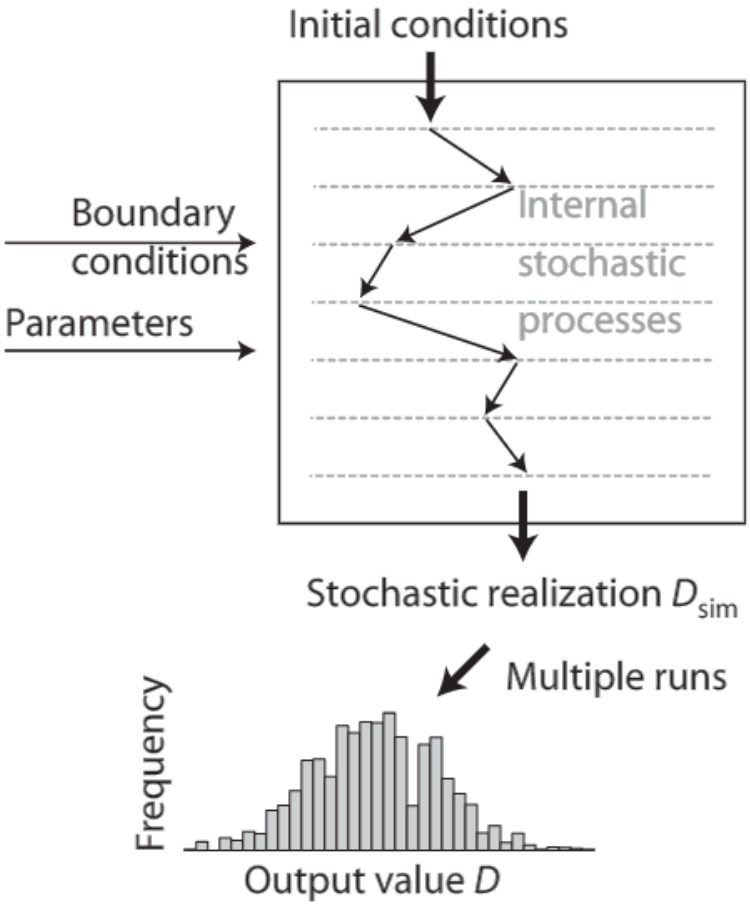
: deterministic function

: random sampling

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

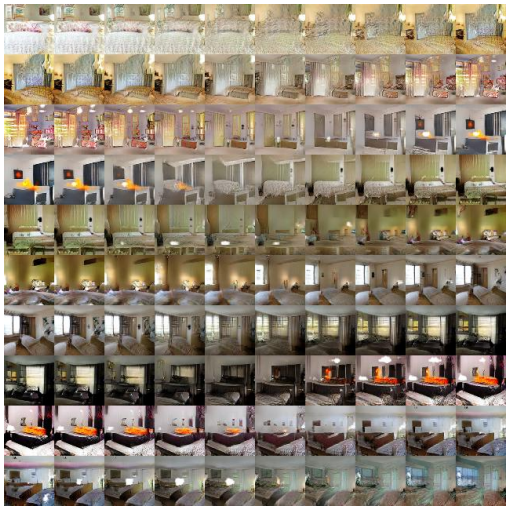
Implicit Deep Generative Models

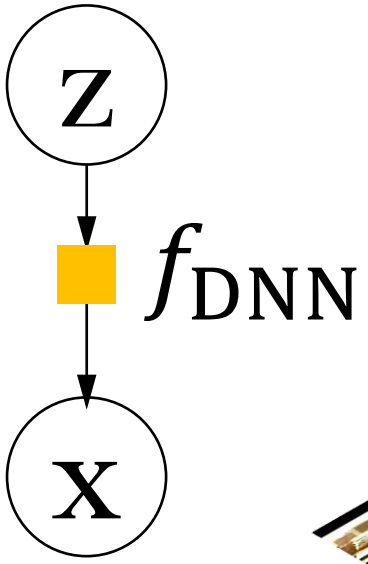
- Generate data with a stochastic process whose likelihood function is not explicitly specified (Hartig et al., 2011)

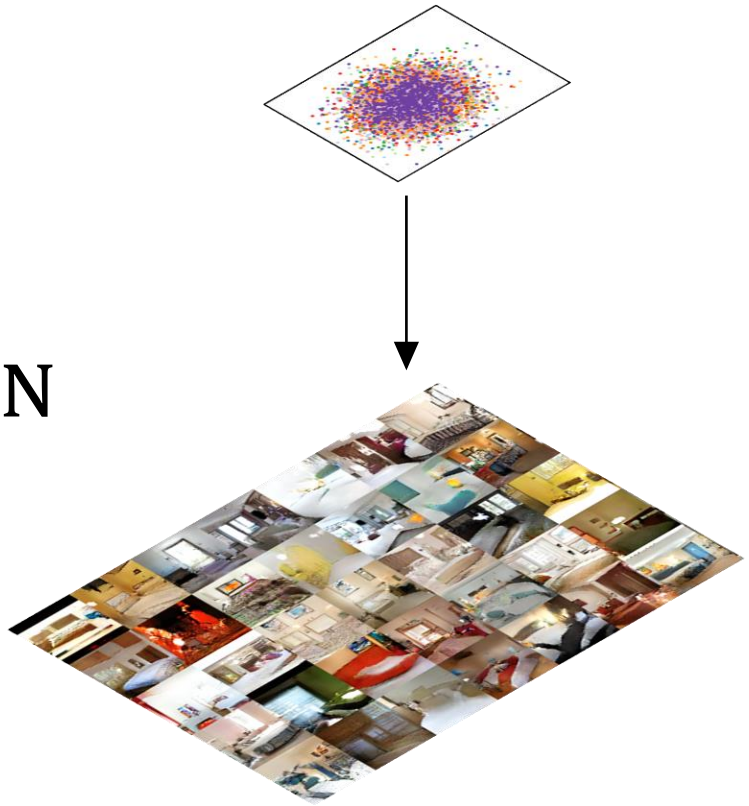


Deep Generative Models

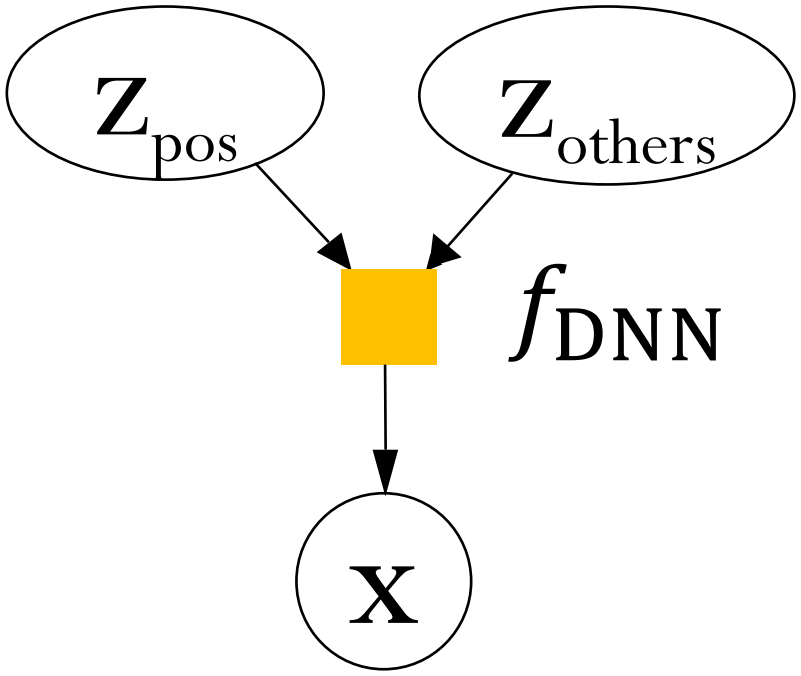
[Image Generation:
Generative Adversarial Nets,
Goodfellow13 & Radford15]

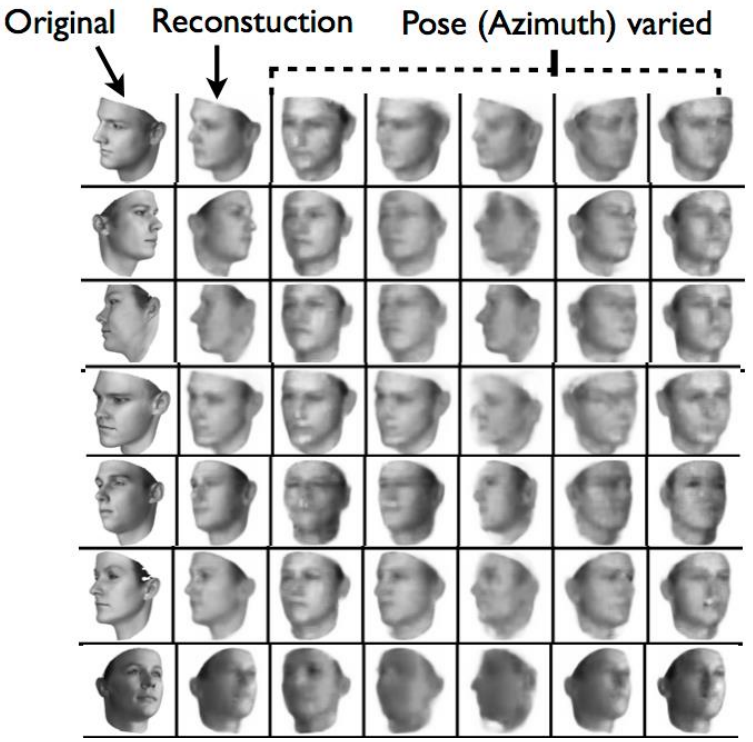






Deep Generative Models



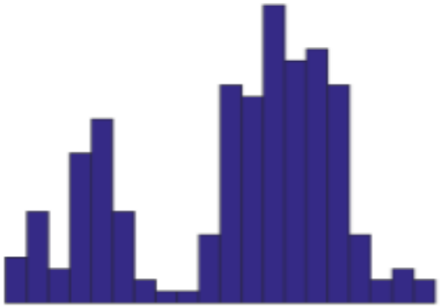


[Image Understanding: Variational Autoencoders, Kingma13 & Tejas15 & Eslami16]

Learning Deep Generative Models

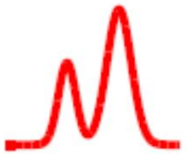
- **Given** a set *D* of unlabeled samples, learn the unknown parameters (or a distribution)

data $D=\{x_i\}$



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):

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

  - has an explicit likelihood model
- Minimax objective (e.g., GAN)
  - a two-player game to reach equilibrium
- 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

Variational Bayes

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

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

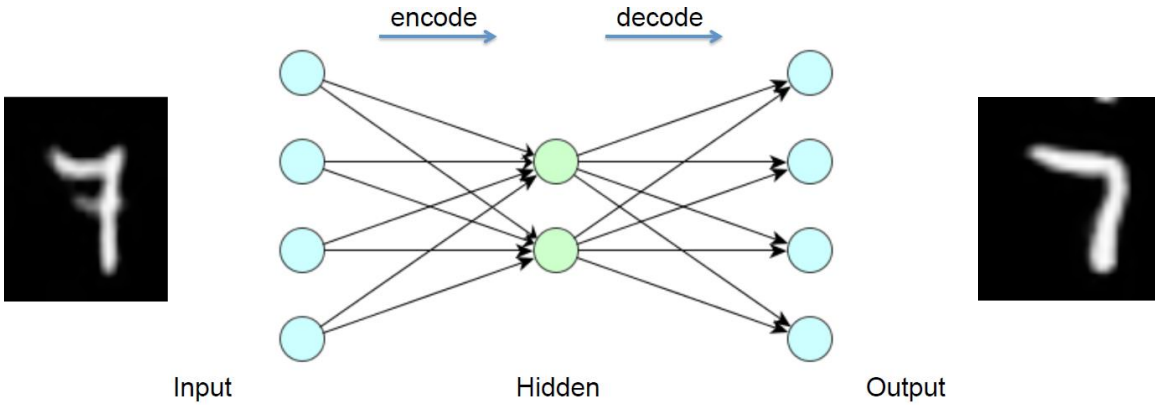
- Log-integral/sum is annoying to handle directly
- Derive a variational lower bound $L(\theta,\phi,\mathrm{x})$

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

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

reconstruction term **prior regularization**

Recap: Auto-Encoder

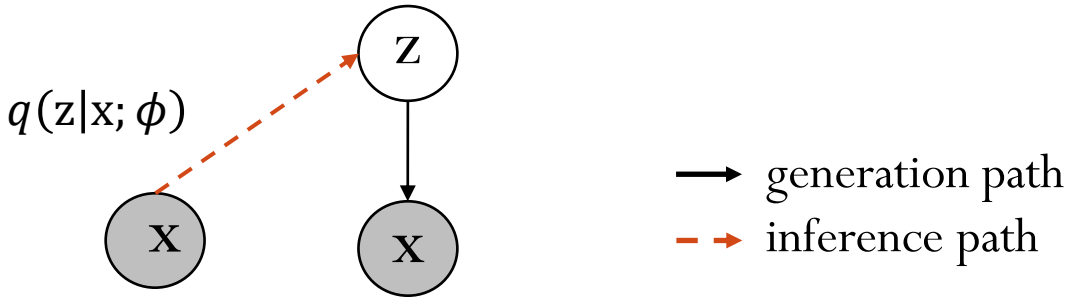


- Encoder: $\mathrm{h} = s(W\mathrm{x} + b)$
- Decoder: $\mathrm{x}' = s(W'\mathrm{h} + b')$
- Training: minimize the reconstruction error (e.g., square loss, cross-entropy loss)
- Denoising AE: randomly corrupted inputs are restored to learn more robust features

Auto-Encoding Variational Bayes (AEVB)

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

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



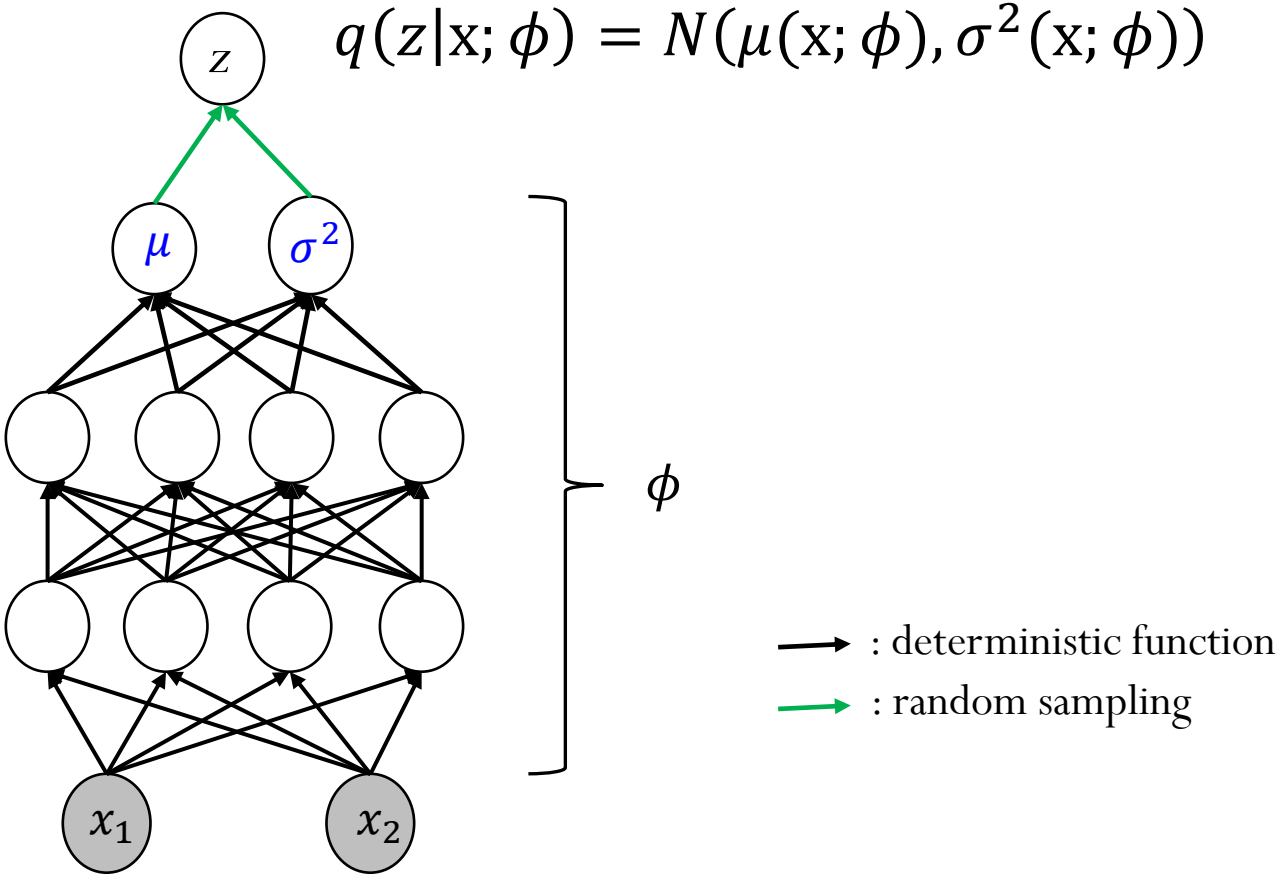
- We call it an inference (recognition, encoder) network or a Q-network
- All the parameters are learned jointly via SGD with variance reduction

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

The Encoder Network

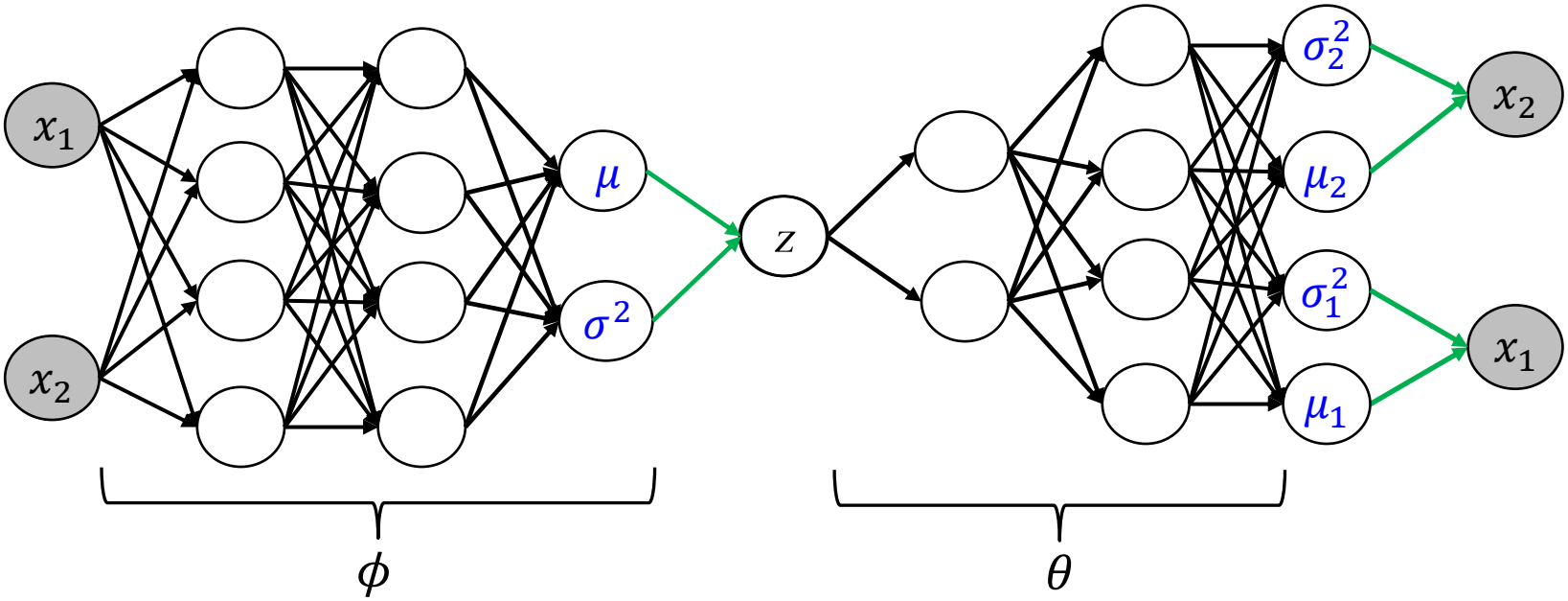
- A feedforward NN + Gaussian

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



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***

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

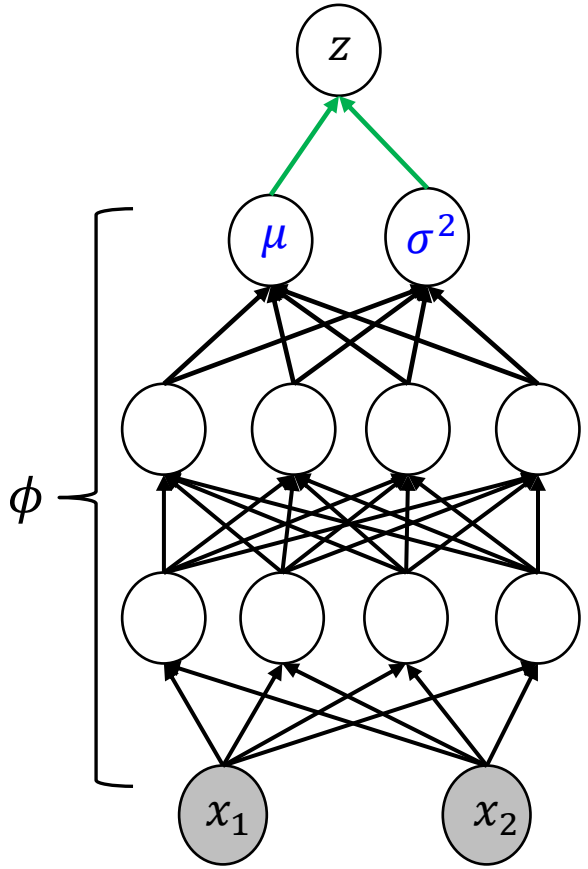
  - Use stochastic gradient methods to handle large datasets
  - Random mini-batch
    - for each i, infer the posterior $q(\mathrm{z}_i|\mathrm{x}_i;\phi)$; As we parameterize as a neural network, this in fact optimizes ϕ

- *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

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



Example with Gaussian Distributions

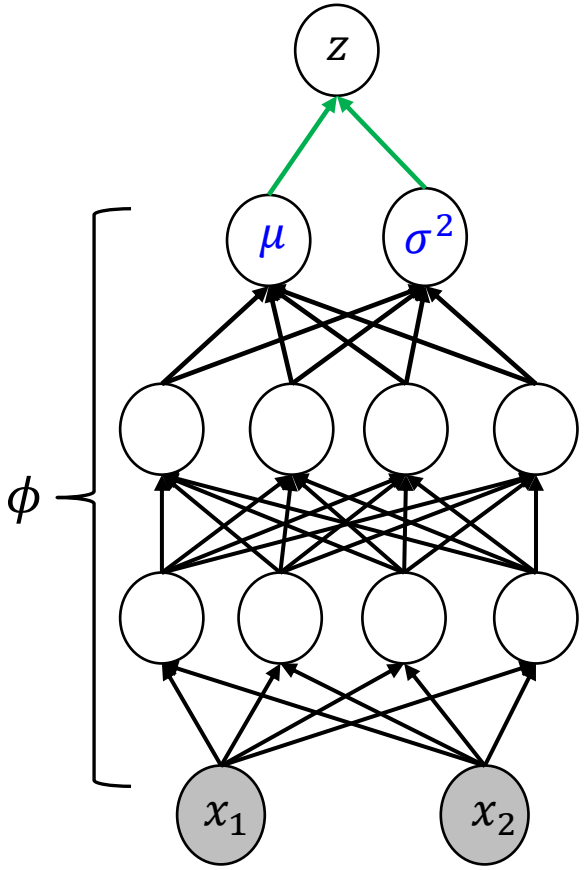
- Use $N(0,1)$ as prior for z; $q(\mathrm{z}|\mathrm{x};\phi)$ is Gaussian with parameters $(\mu(\mathrm{x};\phi), \sigma^2(\mathrm{x};\phi))$ determined by NN
  - The expected log-likelihood

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

  - 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

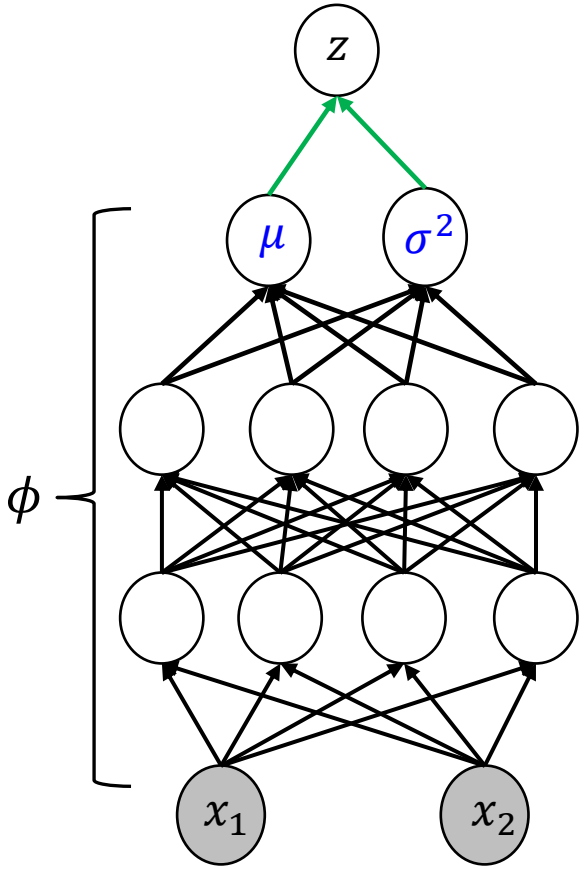
$$\mathbf{E}_{q(\mathrm{z}|\mathrm{x};\phi)}[\log p(\mathrm{x}|z;\theta)]$$

  - 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):

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

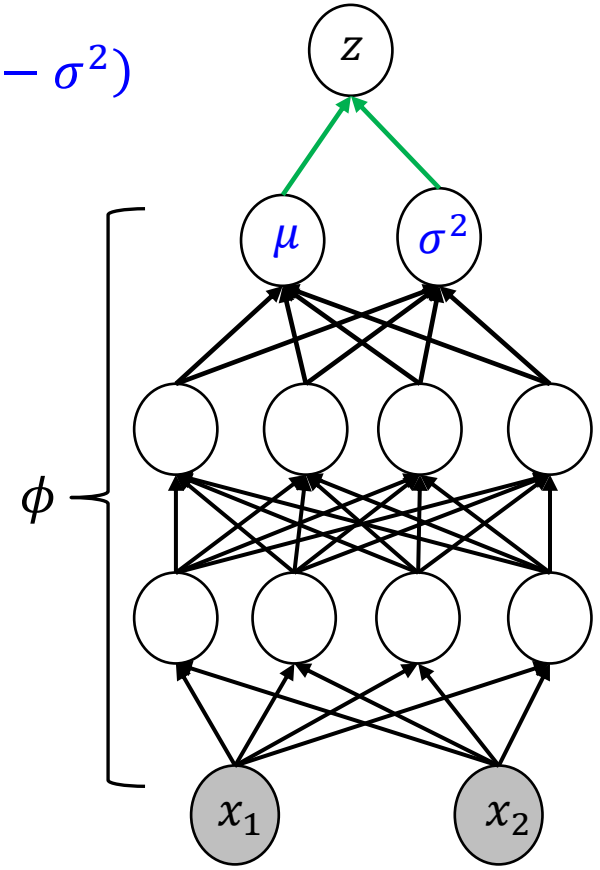
  - Easy to calculate gradient

- The expected log-likelihood term (MC estimate)

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

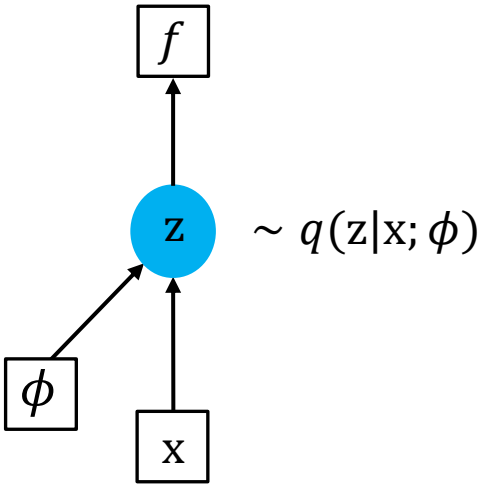
$$z^{(k)} \sim q(z|\mathrm{x};\phi)$$

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

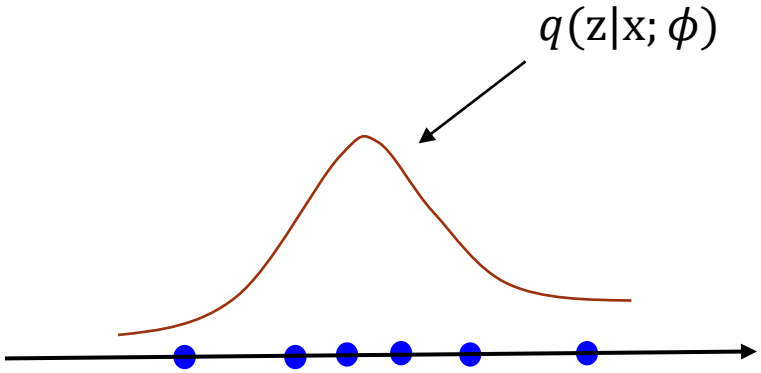


Reparameterization Trick

- Backpropagation not possible through random sampling



□ : deterministic node ● : random node



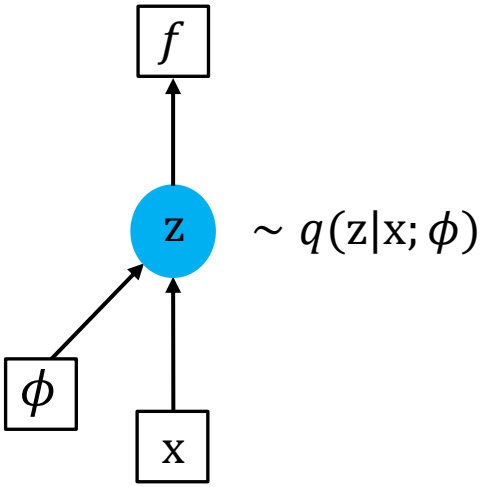
$\{-1.5, -0.5, 0.3, 0.6, 1.5, \ldots\}$

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

Cannot back-propagate through a randomly drawn number

Reparameterization Trick

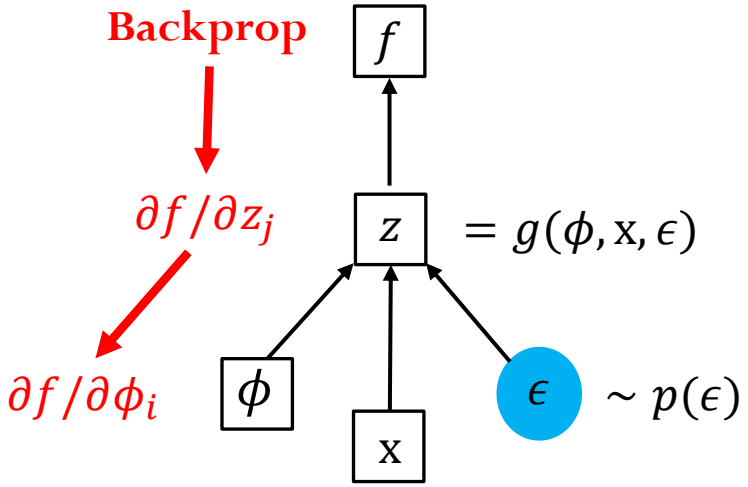
- Backpropagation not possible through random sampling



□ : deterministic node ● : random node

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

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:

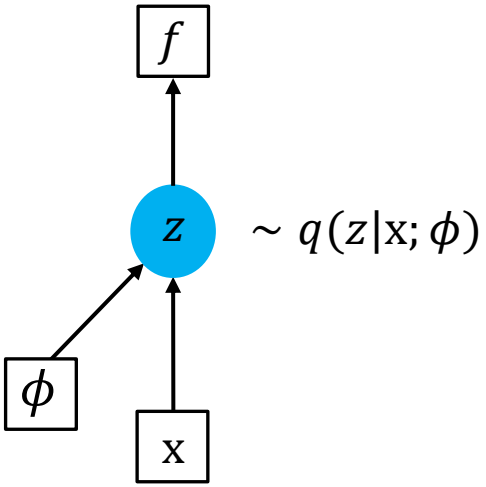
$$L(\theta, \phi, \mathrm{x}) \approx \frac{1}{L}\sum_{k} \log \frac{p\left(\mathrm{z}^{(k)}, \mathrm{x}; \theta\right)}{q\left(\mathrm{z}^{(k)}|\mathrm{x}; \phi\right)}$$

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

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

Reparameterization Trick

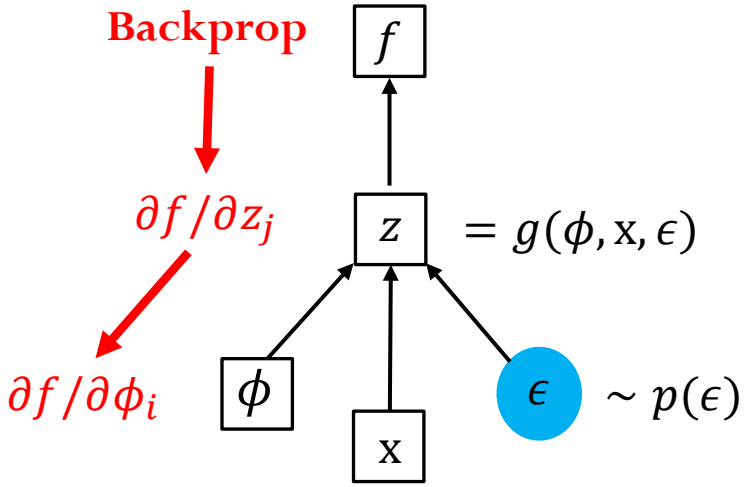
- Backpropagation not possible through random sampling



□ : 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:

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

[Burda et al., arXiv, 2015]

Reparametrization Trick

- The IWAE bound:

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

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

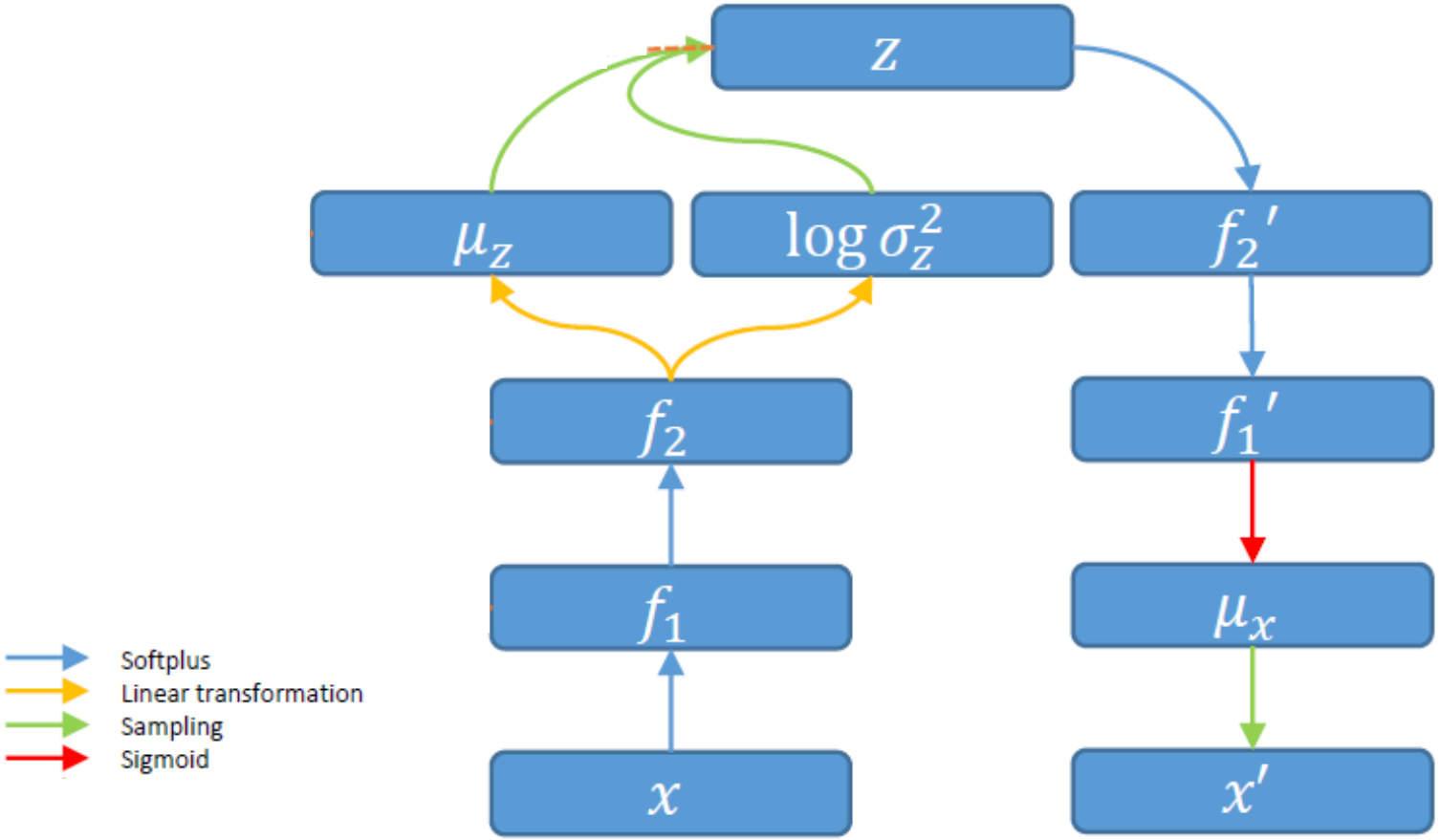
- Reparameterization form:

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

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

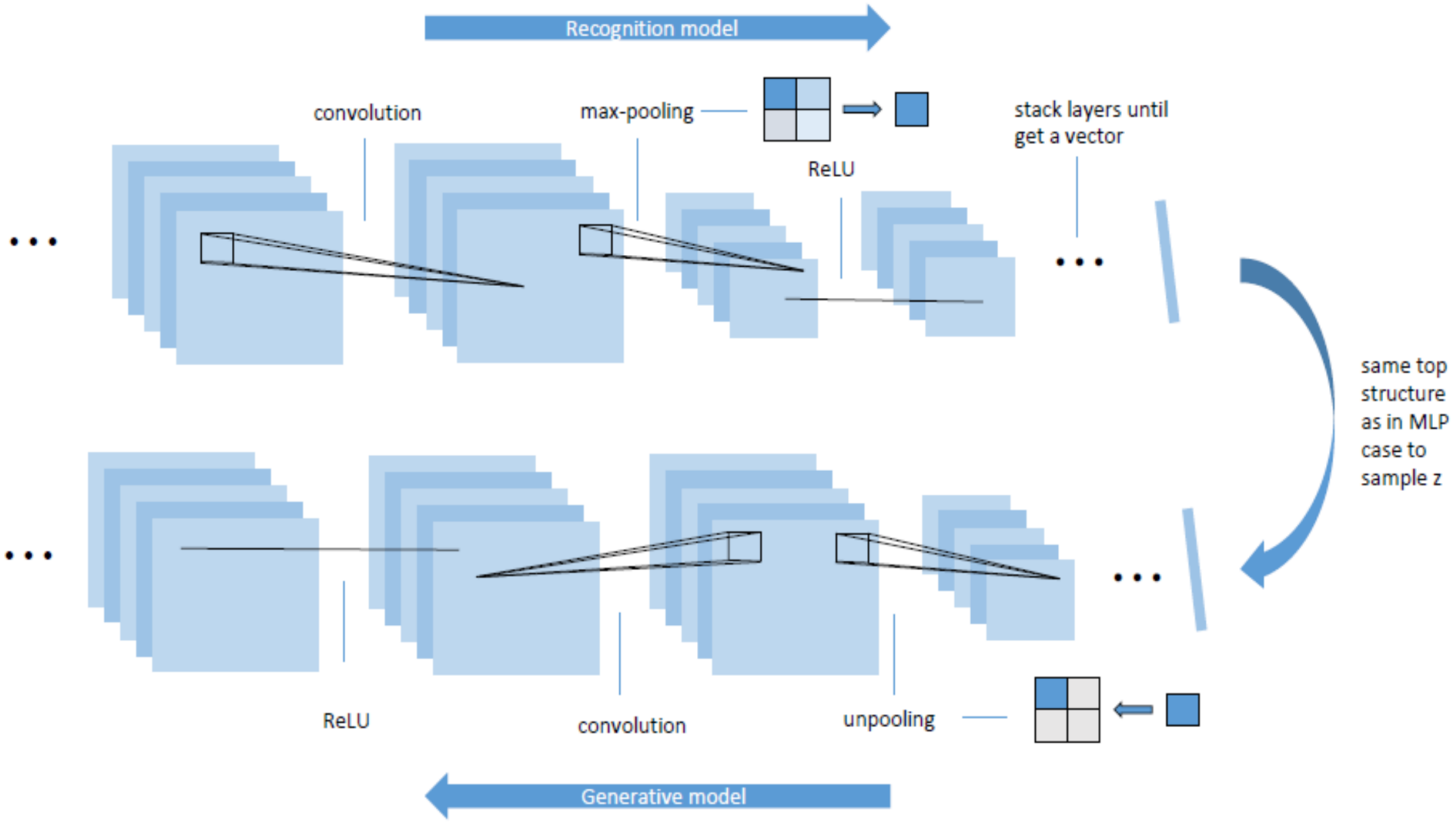
  - The gradient can be calculated as in VAE

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



Generative Adversarial Networks (GAN)

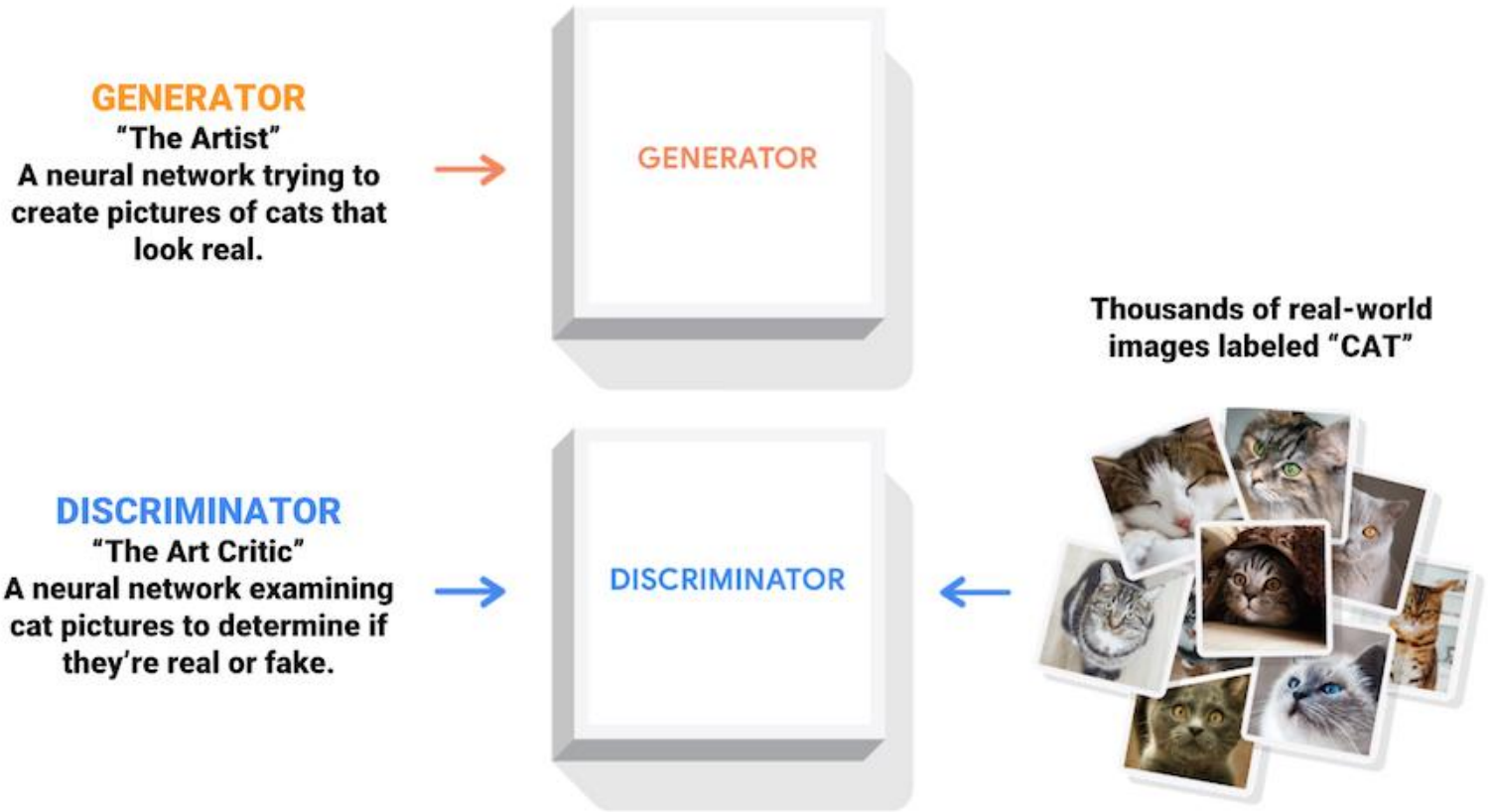
- A game between two players:
  - A discriminator *D*
  - A generator *G*

- *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]

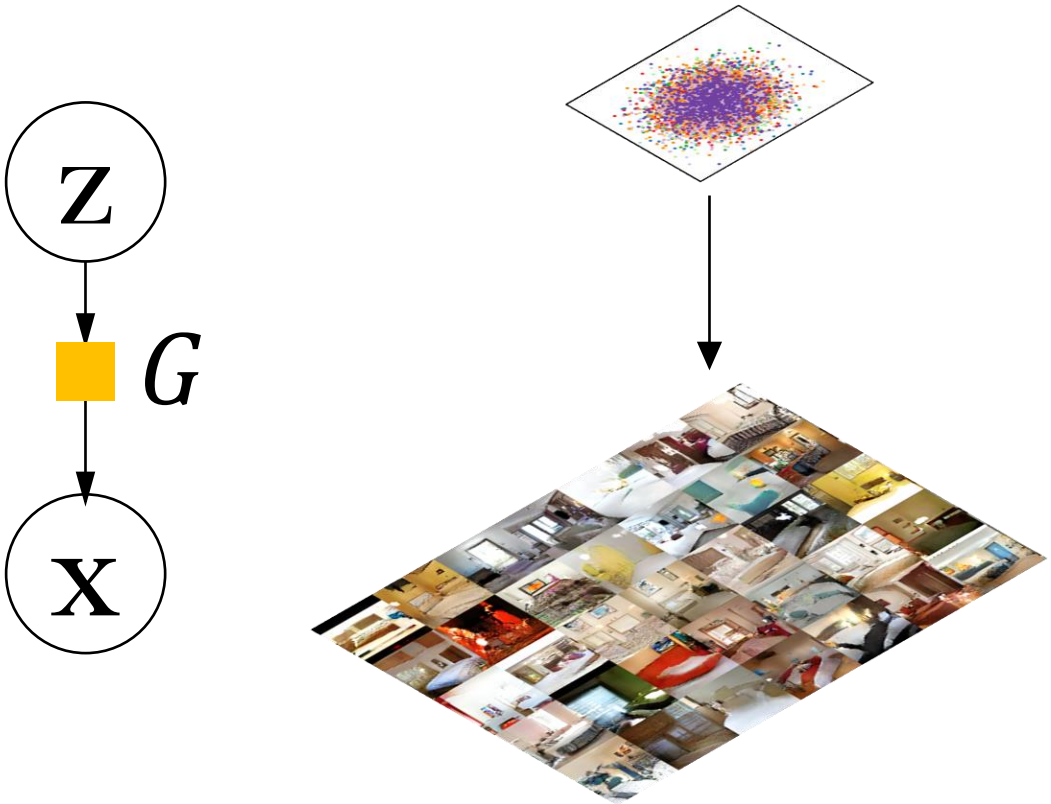
The “Artist” and “Critic” Argument



https://www.tensorflow.org/tutorials/generative/dcgan

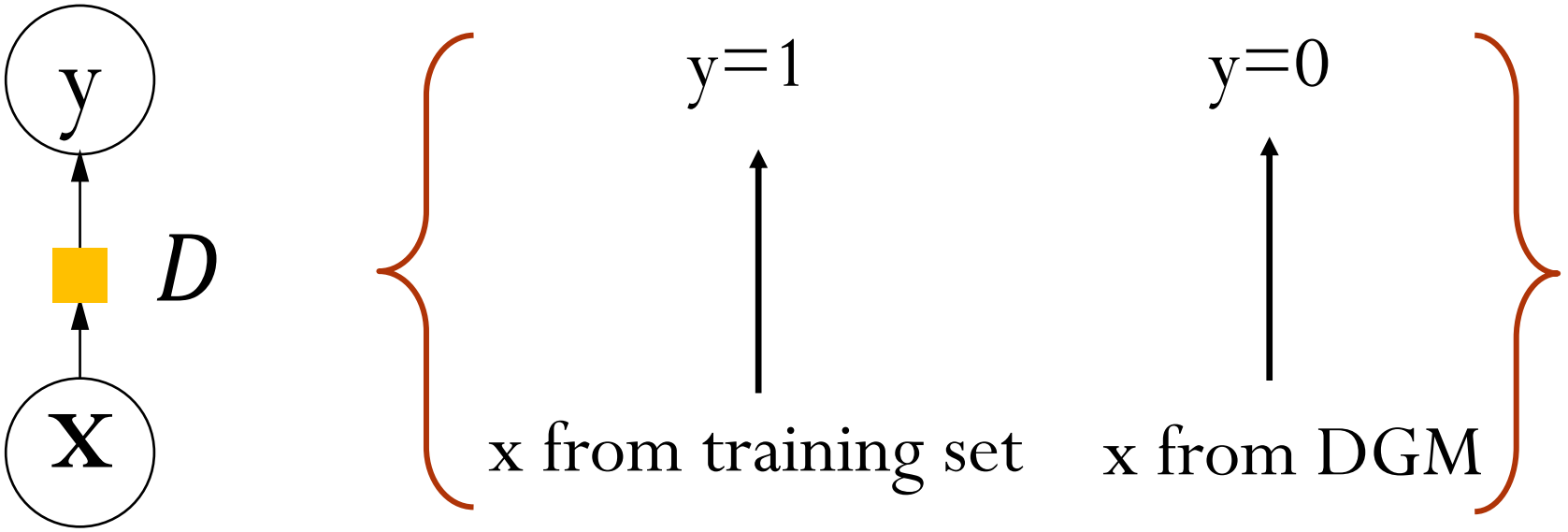
GAN – architecture

- 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

$G(\mathrm{z})$ 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:

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

$G(z)$ is a sample from generator

  - Or minimize the likelihood of being fake:

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

GAN – objective

- Minimax objective function

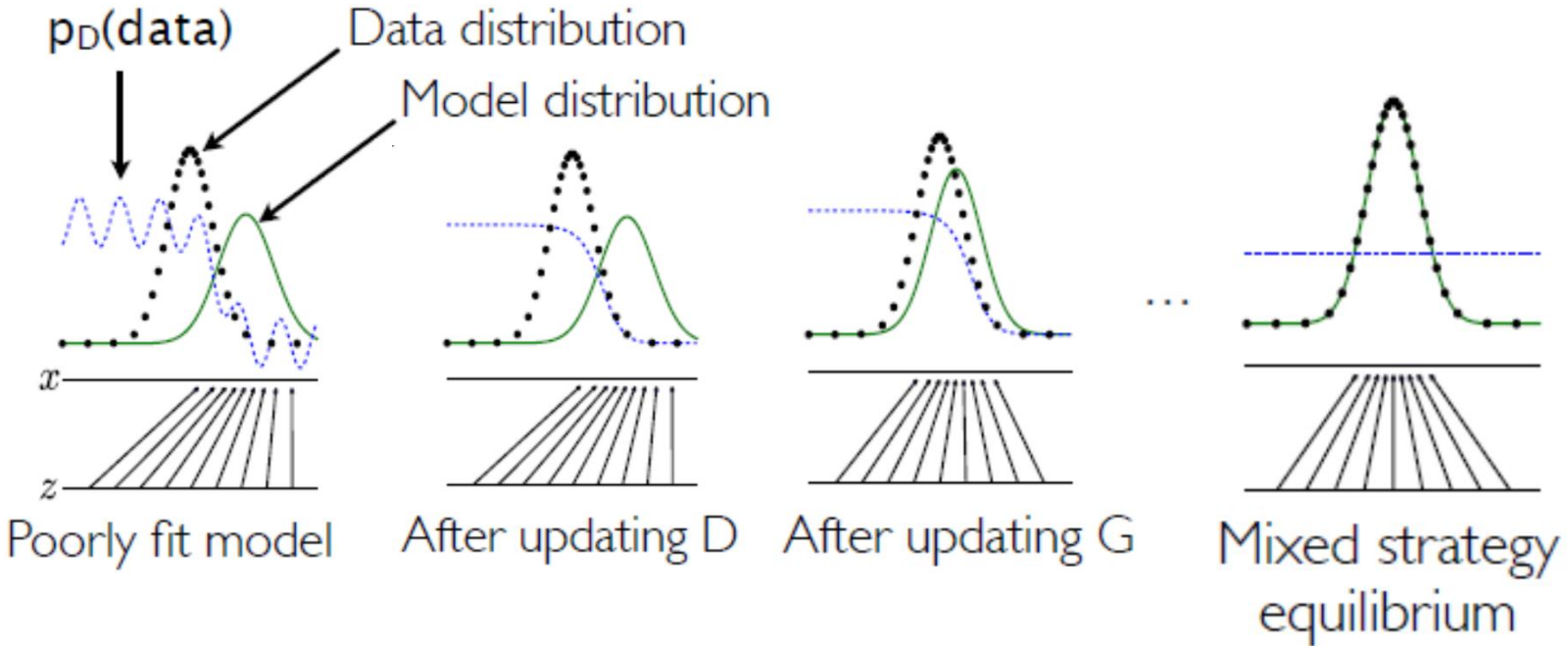
$$\min_G \max_D \; \mathbf{E}_{p_{\text{data}}(\mathrm{x})}[\log D(\mathrm{x})] + \mathbf{E}_{p(\mathrm{z})}\left[\log\left(1 - D\left(G(\mathrm{z})\right)\right)\right]$$

  - 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

Practice

- Early stopping of discriminator training to avoid overfitting
  - Multiple updates of D
  - One update of G
- Gradient saturation

- Implement VAE and GAN on Tensorflow at the Review session.

Flow-based Models

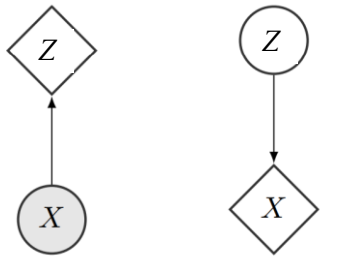
- A generative process with invertible (or bijective) function g:

$$z \sim p(z)$$

$$x = g(z)$$

- The inference network for the latent variable is:

$$z = f(x) = g^{-1}(x)$$



(a) Inference (b) Sampling

- The density:

$$p(x) = p(f(x))\left|\det \frac{\partial f(x)}{\partial x}\right|$$

- Desirae of the function: easy determinant of Jacobian, easy inverse

[Dinh, Krueger, Bengio. Nice: non-linear independent components estimation, workshop at ICLR 2015]

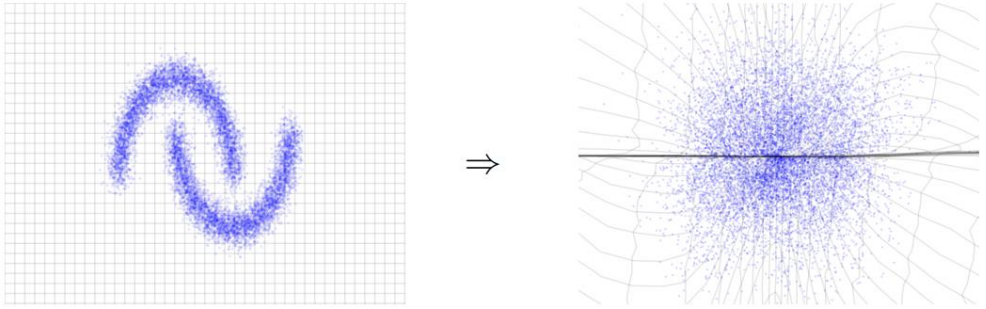
An illustrative example

Data space $\mathcal{X}$ | Latent space $\mathcal{Z}$

Inference

$x \sim \hat{p}_X$

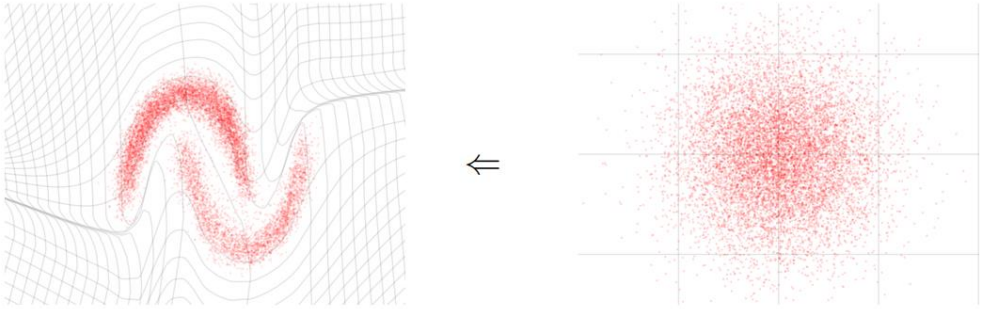
$z = f(x)$



Generation

$z \sim p_Z$

$x = f^{-1}(z)$



An Example of the Invertible Function

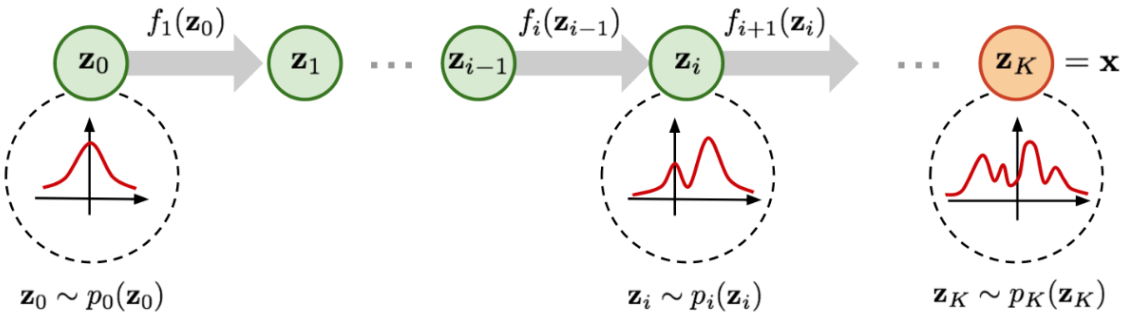
- A simple idea: split *x* into two parts and define

$$y_1 = x_1$$
$$y_2 = x_2 + m(x_1)$$

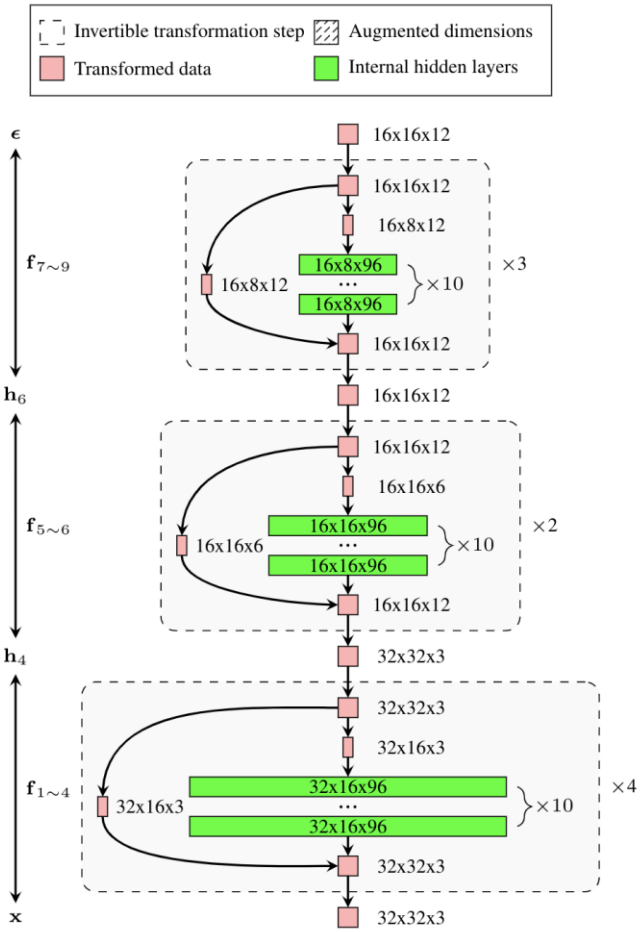
  - Then the determinant-Jacobian 1!
  - The inverse is $x_1 = y_1$, $x_2 = y_2 - m(y_1)$

  - *m* is an arbitrarily complex function (e.g., ReLU networks)

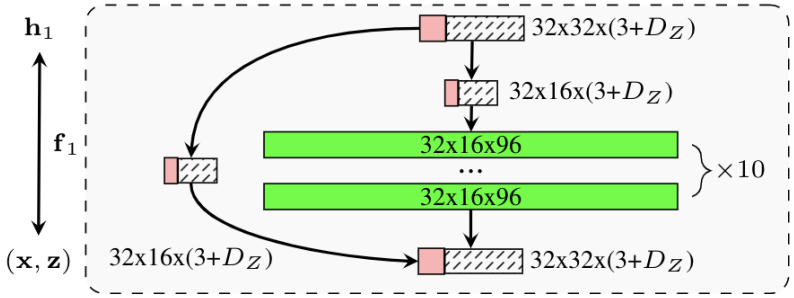
- Define the invertible transform function as a composition of simple functions
  - Thus, the variable change is a sequence, called a ***flow*** (the sequence on density change is a ***normalizing flow***)



Bottleneck Problem in Flow++



Invertibility imposes constraints on architectures!



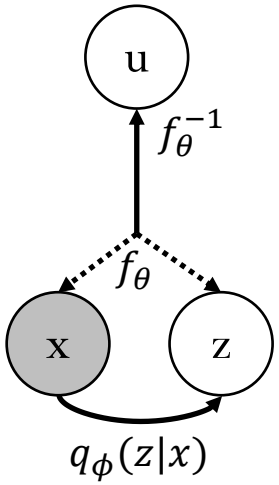
(b)

Figure 1. (a) Bottleneck problem in a Flow++ (Ho et al., 2019) for CIFAR-10. Dimensionality of the transformed data (red) limits the model capacity. (b) Our solution VFlow, where D_Z is the dimensionality of the augmented random variable. Only the transformation step $\mathbf{f}_1$ is shown due to space constraint.

[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

VFlow: more Expressive Generative Flows through Variational Data Augmentation

learn a generative flow $p(x, z)$ in the augmented data space jointly with the augmented data distribution



$$u \sim \mathcal{N}(0, I)$$

$$y = concat([x, z])$$

$$\log p(x, z) = \log p(u) + \log \det(\frac{du}{dy})$$

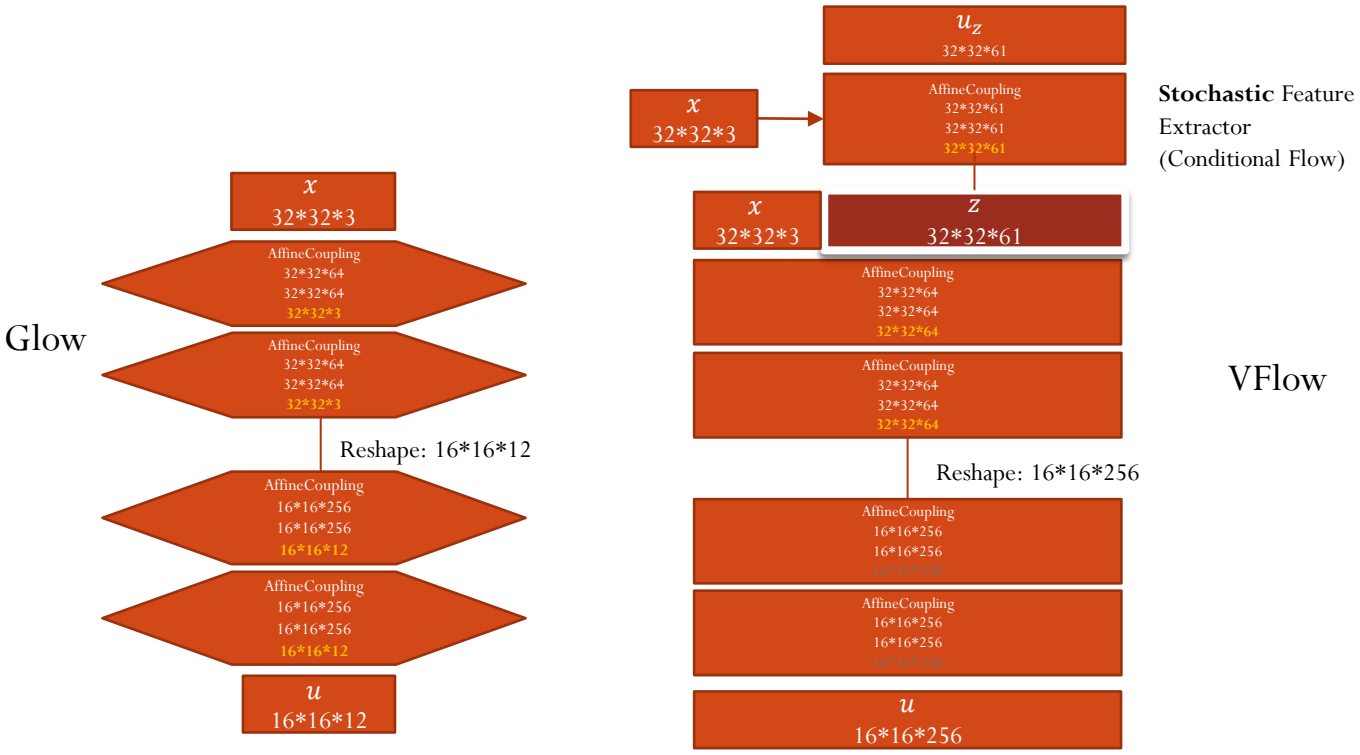
$$\log p(x) \geq \mathrm{E}_{q(z|x)} [\log p(x, z) - \log q(z|x)]$$

$$\max_{\boldsymbol{\theta}, \boldsymbol{\phi}} \mathbb{E}_{\hat{p}(\mathbf{x}) q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})} [\log p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta}) - \log q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})]$$

Provably better than generative flows! Also subsumes VAEs as a special case.

[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

VFlow

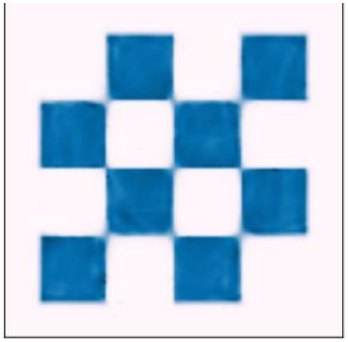


[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

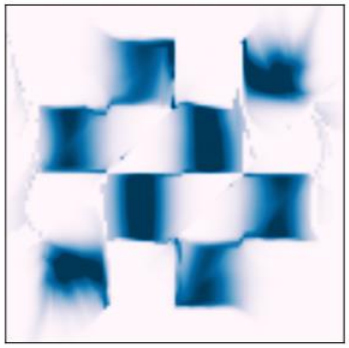
Results on Toy 2D Data

- VFlow significantly outperforms Glow under similar model size (e.g., 3-step)
- The 3-layer, 10-dimensional VFlow even outperforms a much larger 20-step Glow

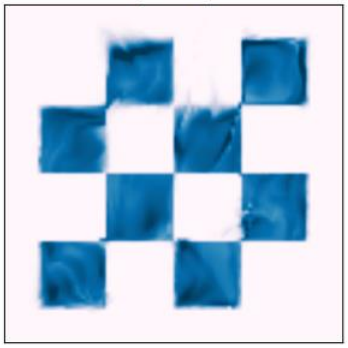
(a) Data (-3.47)



(b) 3-step, 10-dim VFlow (-3.51)



(c) 3-step Glow (-3.66)



(d) 20-step Glow (-3.52)

Results on Toy 2D Data

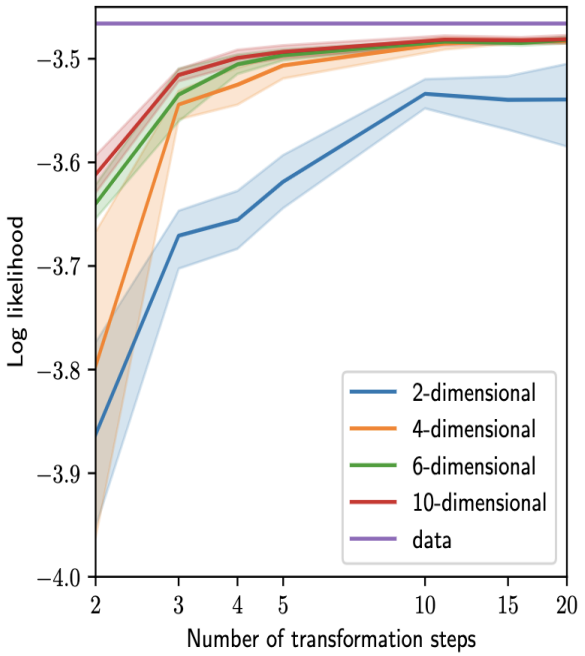


Figure 4. Impact of the dimensionality on the toy dataset.

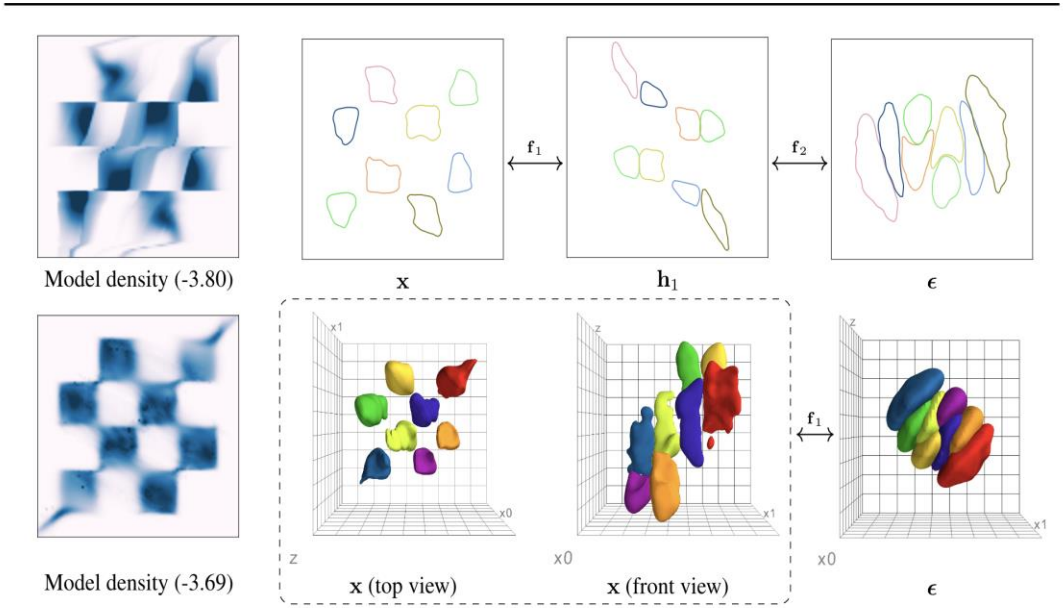


Figure 3. Visualization of learnt transformation on toy data. Top row: 2-step Glow. Bottom row: 2-step, 3-dimensional VFlow. Log-likelihood is shown in parenthesis. We sample ϵ and visualize the transformed density in $\mathbf{x}$, $\mathbf{h}_1$ and ϵ space. The density is estimated from samples by kernel density estimation, and we show the 50% probability contour / isosurface for each mode in different color.

[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

Results on CIFAR-10

Table 1. Density modeling result on the CIFAR-10 dataset.

Model	bpd
Glow (Kingma & Dhariwal, 2018)	3.35
FFJORD (Grathwohl et al., 2019)	3.40
Residual Flow (Chen et al., 2019)	3.28
MintNet (Song et al., 2019)	3.32
Flow++ (Ho et al., 2019)	3.08
VFlow	**2.98**

Table 2. Impact of dimensionality on the CIFAR-10 dataset.

Model	bpd	Parameters	D_H	B
3-channel Flow++	3.21	4.02M	32	13
4-channel VFlow	3.15	4.03M	32	11
6-channel VFlow	**3.12**	**4.01M**	32	10

Table 3. Parameter efficiency.

Model	bpd	Parameters	D_H	B
3-channel Flow++	3.08	31.4M	96	10
6-channel VFlow	**2.98**	37.8M	96	10
6-channel VFlow	3.03	16.5M	64	10
6-channel VFlow	3.08	**11.9M**	56	10

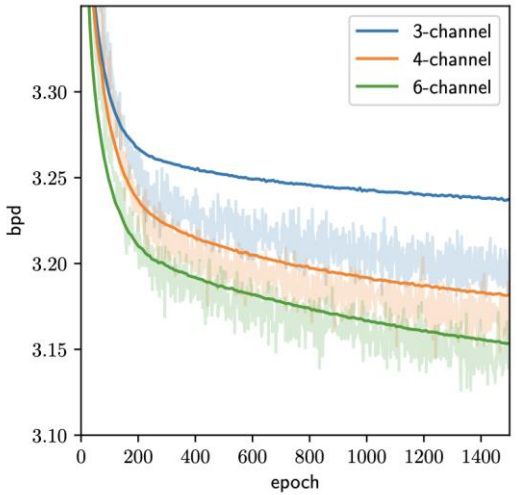


Figure 6. Bpd on training (light) and testing (dark) dataset of Flow++ and VFlow under a 4-million parameter budget. Here bpd is only a upper bound because we evaluate it with ELBO as Eq. (7) instead of the marginal likelihood.

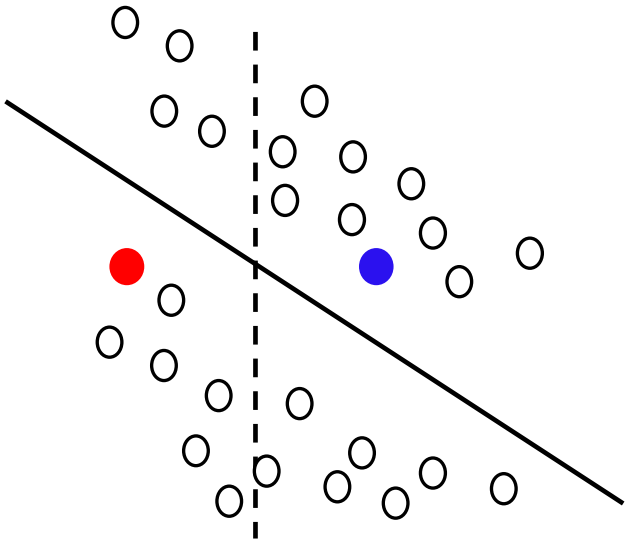
[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

Semi-supervised Learning with GANs

(how to learn efficiently from a limited number of labels?)

Semi-supervised Learning

- A toy example

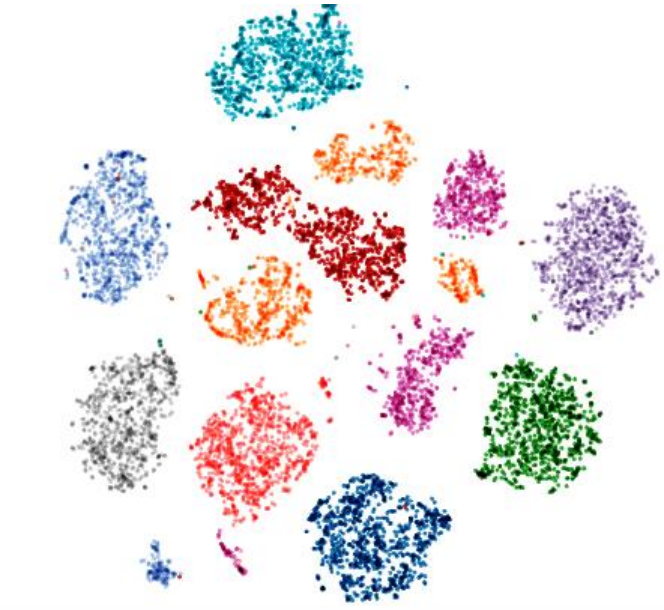


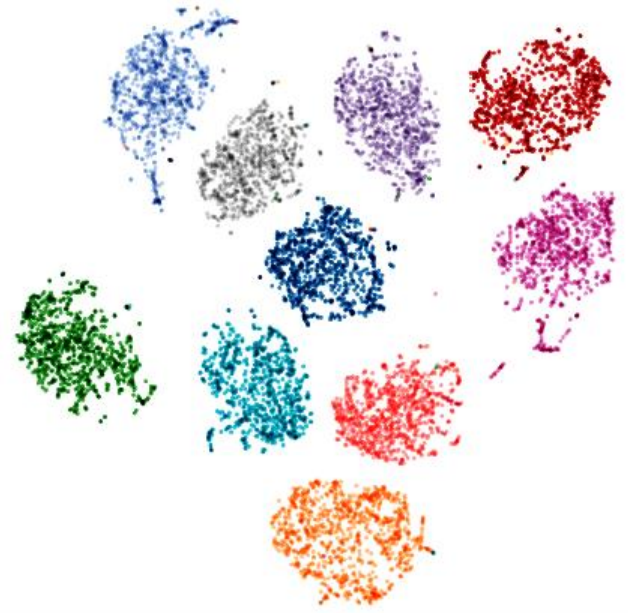
Representation Matters

- PCA embedding of learned representations by different DGM models on CIFAR10

Representation Matters

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

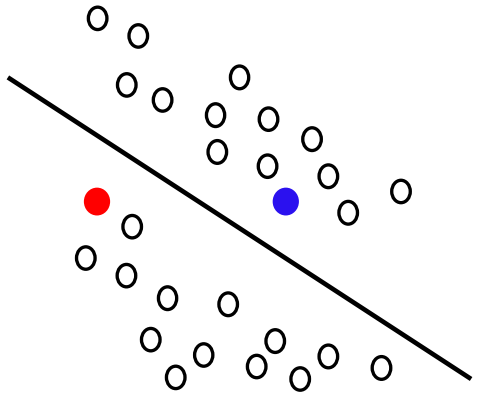




Triple Generative Adversarial Nets

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

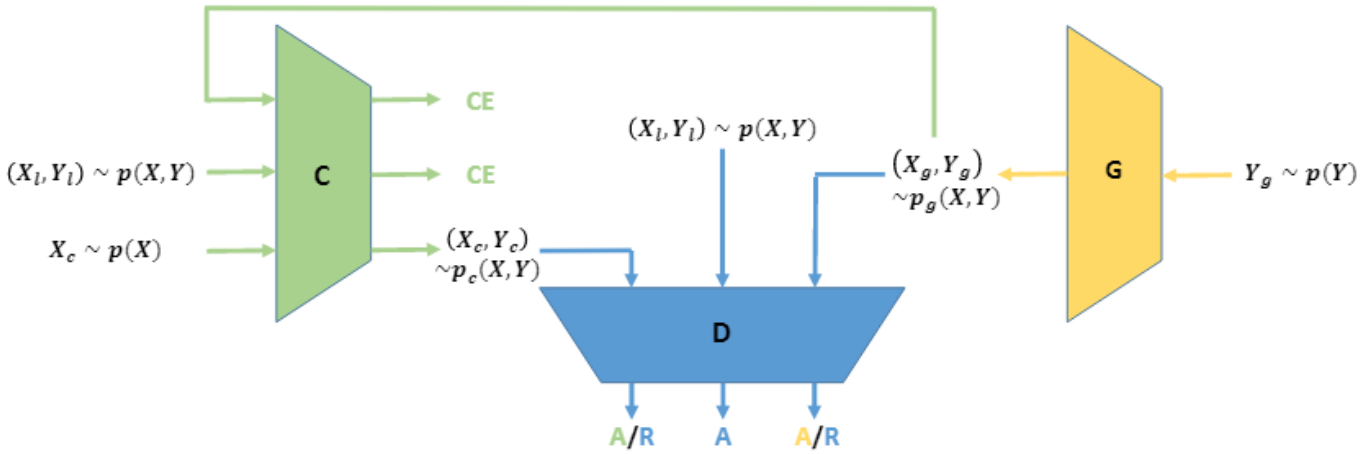
$$p(x, y) = p(x)p(y|x) \quad \leftarrow \text{A classifier}$$
$$= p(y)p(x|y) \quad \leftarrow \text{A class-conditional generator}$$



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

Triple-GAN

- The network architecture



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

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

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

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

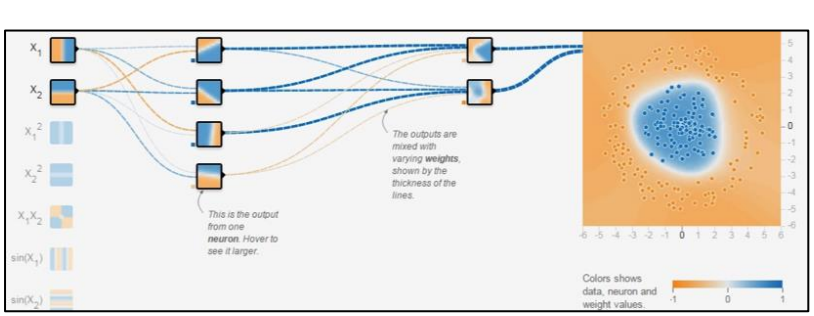
Some Results

- Latent space interpolation on CIFAR10

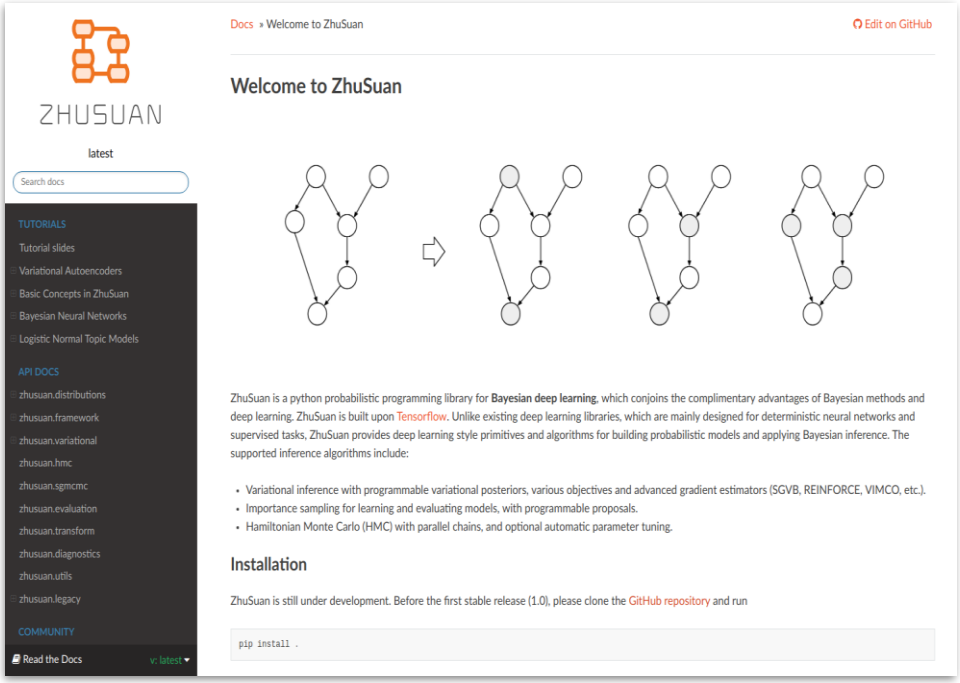
Probabilistic Programming Library

(how to auto/semi-auto implement Bayesian deep learning models?)

DNN Programming ⟶ Deep Probabilistic Programming (Deep PPL)



Learning a mapping from x to y



github.com/thu-ml/zhusuan

"ZhuSuan" Deep Probabilistic Programming

Improved modeling language

```
def build_model(...):
    bn = zs.BayesianNet()
    x = bn.normal("x", x_mean, std=...)
    y = bn.deterministic("y", tf.any_op(x))
Or  y = tf.any_op(x)
    ......
    return bn
```

1. Model build function
2. Create a Bayesian Net
3. Add stochastic node, e.g., a Gaussian variable x
4. Add deterministic nodes, can use any Tensorflow operation
5. Return the built Bayesian Net

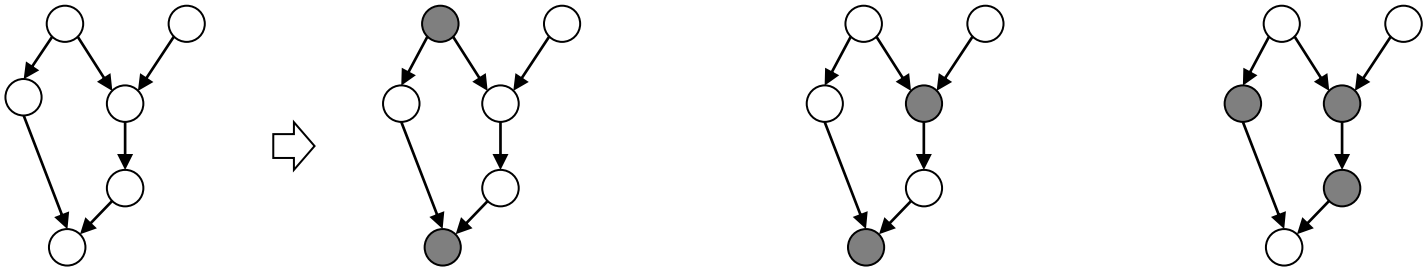
"ZhuSuan" Deep Probabilistic Programming

New model reuse strategy

Just need a decorator `@zs.meta_bayesian_net` to the model-building function

```
@zs.meta_bayesian_net(scope="model", reuse_variables=True)
def build_model(...):
```

The returned is a reusable `zs.MetaBayesianNet` object, just call `observe()` to assign observed values to any subset of random variables




```
bn = meta_bn.observe(x=x_obs, y=y_obs)
```

github.com/thu-ml/zhusuan

“ZhuSuan” Deep Probabilistic Programming

New Algorithms： Stochastic gradient MCMC

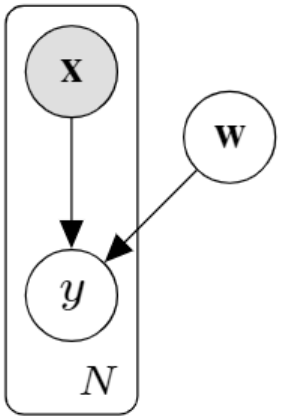
变分推理	梯度估计器	实现
elbo()	SGVB	.sgvb()
	Reinforce	.reinforce()
klpq()	Importance Sampling	.importance()
iw_objective()	SGVB	.sgvb()
	VIMCO	.vimco()

马尔科夫链蒙特卡洛	实现
Hamiltonian Monte Carlo	HMC()
Stochastic Gradient MCMC	(Preconditioned) SGLD	SGLD(), PSGLD()
	SGHMC	SGHMC()
	SGNHT	SGNHT()

github.com/thu-ml/zhusuan

Examples: Bayesian Logistic Regression

ZHUSUAN



(a)

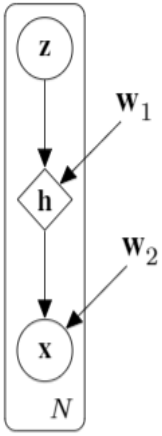
```
import zhusuan as zs
import tensorflow as tf

@zs.meta_bayesian_net(scope="blr")
def build_blr(x, alpha, D):
    bn = zs.BayesianNet()
    w = bn.normal('w', mean=tf.zeros([D]),
                  std=alpha, group_ndims=1)
    y_logit = tf.reduce_sum(
        tf.expand_dims(w, 0)*x, axis=1)
    bn.bernoulli('y', y_logit)
    return bn
```

(b)

Example: Variational Auto-Encoder

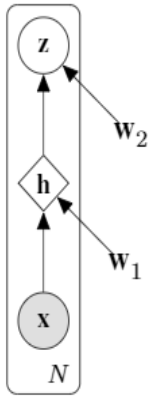
ZHUSUAN



(a)

```python
@zs.meta_bayesian_net("vae", reuse_variables=True)
def build_vae(N, D):
    bn = zs.BayesianNet()
    z_mean = tf.zeros([N, D])
    z = bn.normal('z', z_mean, std=1., group_ndims=1)
    h = tf.layers.dense(z, 500,
                        activation=tf.nn.relu)
    x_logits = tf.layers.dense(h, 784)
    bn.bernoulli('x', x_logits, group_ndims=1)
    return bn
```

(b)



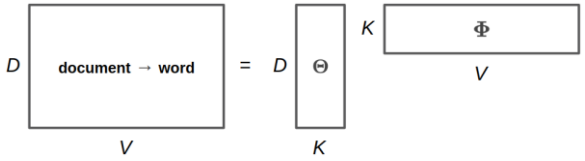
(a)

```python
@zs.reuse_variables("vae_q")
def build_q_net(h_dim, z_dim):
    bn = zs.BayesianNet()
    h = tf.layers.dense(x, h_dim,
                        activation=tf.nn.relu)
    z_mean = tf.layers.dense(h, z_dim)
    z_logstd = tf.layers.dense(h, z_dim)
    bn.normal('z', z_mean, logstd=z_logstd,
              group_ndims=1)
    return bn
```

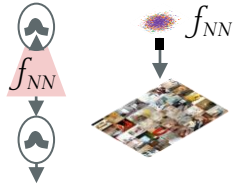
(b)

"ZhuSuan" Deep Probabilistic Programming

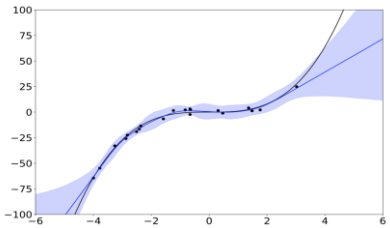
More models



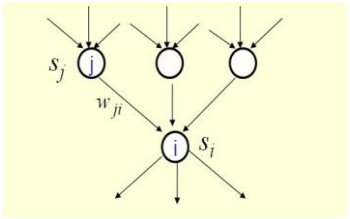
Topic Models; Probabilistic Matrix Factorization
for text analysis, Recommendor Sys.



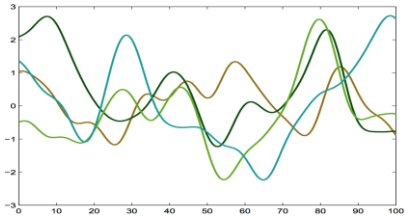
Variational Auto-encoder
for structured data generation, discrete representation learning, semi-supervised learning, etc.



Bayesian Neural Networks
for uncertainty in deep nets



Deep Belief Nets
Unsupervised pre-training
for DNNs



Gaussian Processes
Nonlinear regression problems

github.com/thu-ml/zhusuan

"ZhuSuan" Deep Probabilistic Programming

- Open-sourced in GitHub:

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

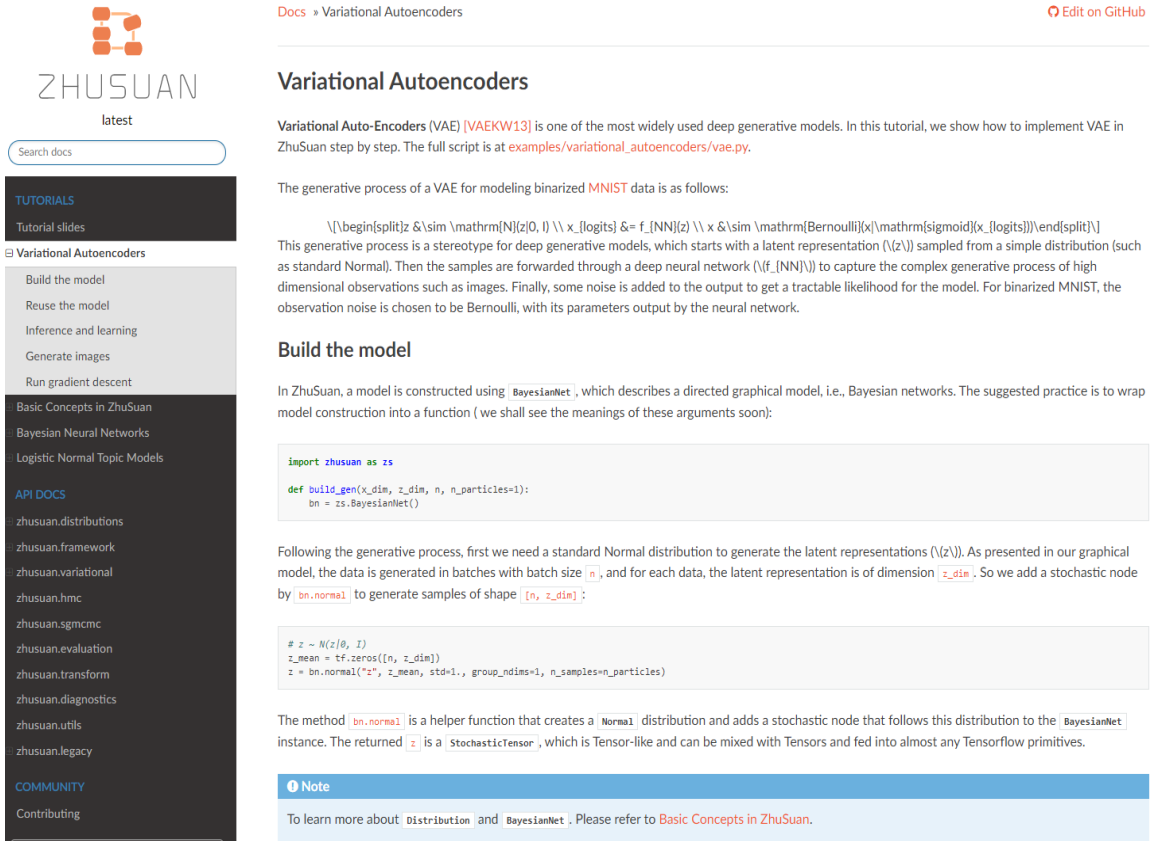
ZhuSuan: A Library for Bayesian Deep Learning

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/



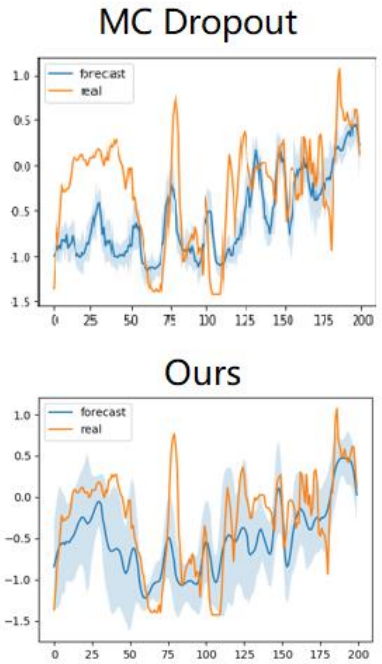
Examples

(what problems can be solved by Bayesian deep learning?)

Examples: Air Quality Prediction

- Bayesian inference for a baseline model with LSTM and attention
  - Calculate the uncertainty of prediction
  - Decrease the prediction error of the baseline model

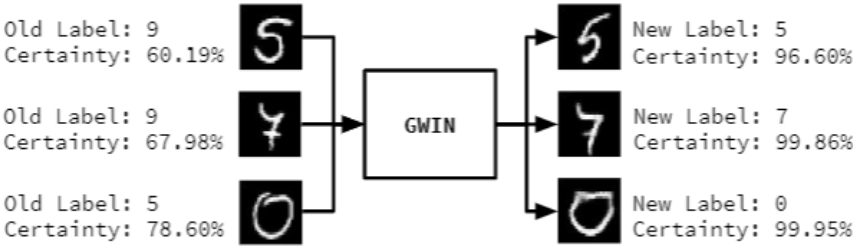
MSE (NO_2)	BNN	Baseline NN
+1 hr	**0.145**	0.160
+7 hr	**0.371**	0.423
+16 hr	**0.389**	0.508

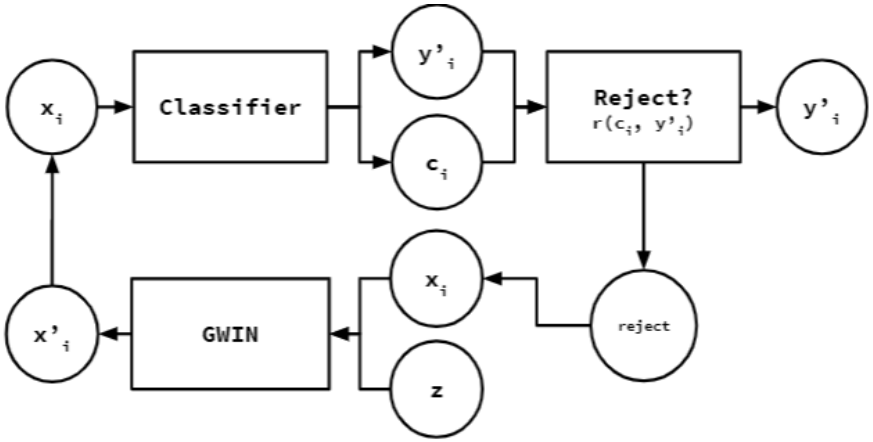


[Function space particle optimization for Bayesian neural networks. Wang et al., ICLR 2019]

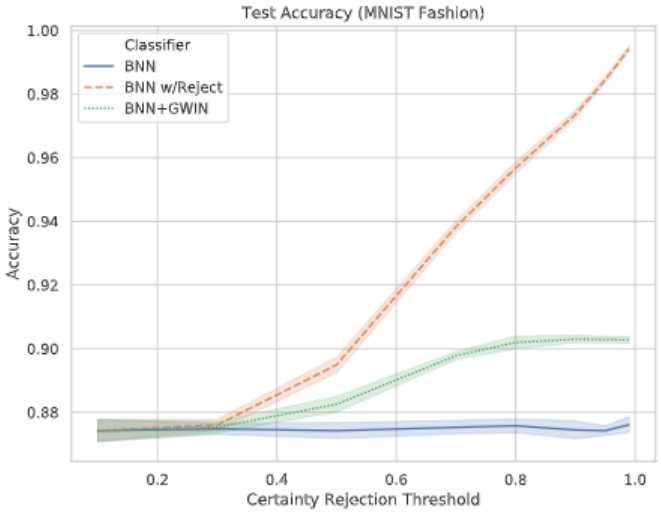
Example: Learning with Rejection

- Leverage deep generative models to transfer "rejected" samples into high-confidence region

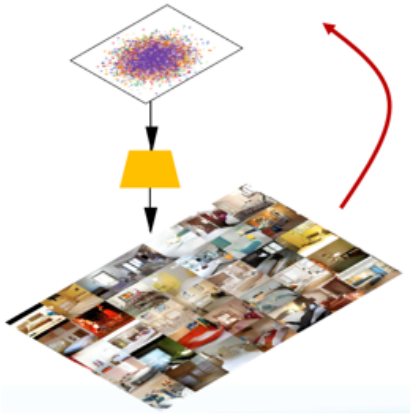




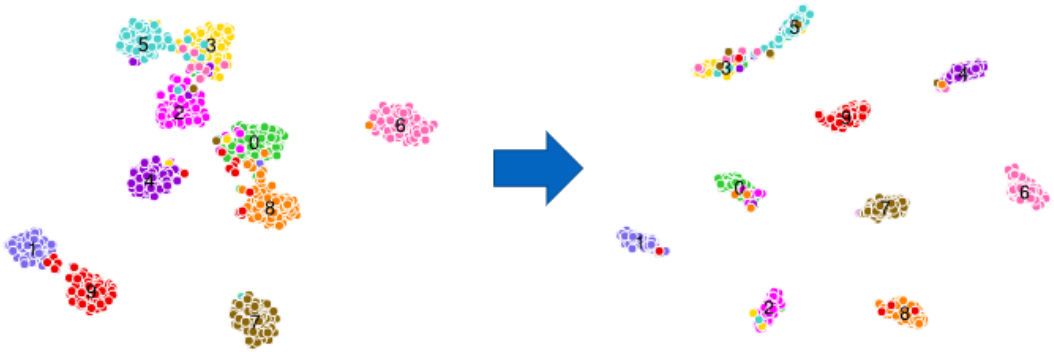
Generative Well-intentioned Networks (GWIN)
(Cosentino & Zhu, NeurIPS 2019)

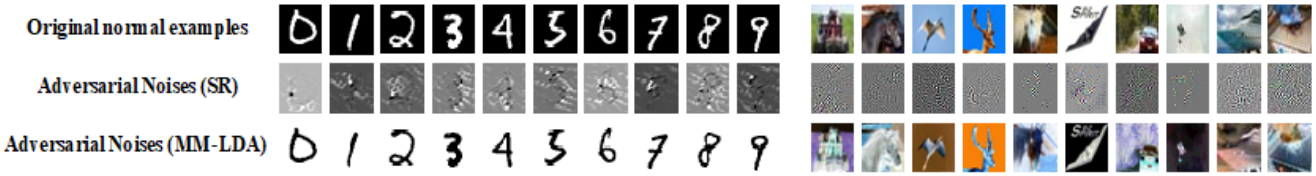


Examples: Adversarial Robustness



Learn a deep network to map the data into a simple MoG distribution





- Theoretically show optimal robustness
- Algorithmically train MM-LDA network using SGD to minimize cross-entropy loss

No extra cost, as compared to vanilla DNN

Max-Mahalanobis Linear Discriminant Analysis Networks (Pang et al., ICML 2018)

Thanks!

ZhuSuan: A Library for Bayesian Deep Learning
J. Shi, et al., arXiv preprint, arXiv:1709.05870 , 2017

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