

[70240413 Statistical Machine Learning, Spring, 2023]

# 深度生成模型 ( VAE, GAN, Flows )

**Jun Zhu**

dcszj@mail.tsinghua.edu.cn

<http://ml.cs.tsinghua.edu.cn/~jun>

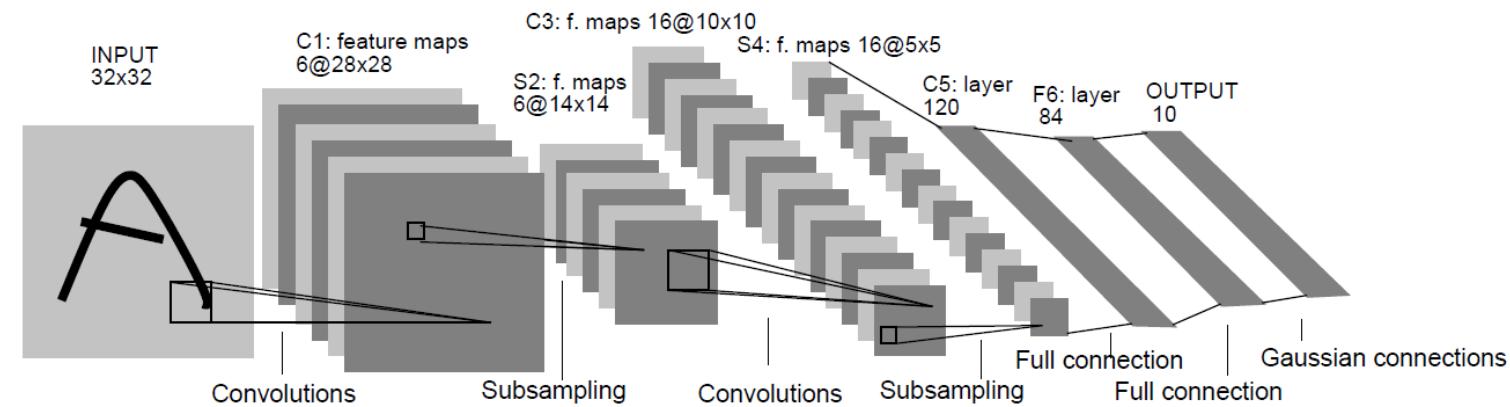
State Key Lab of Intelligent Technology & Systems

Tsinghua University

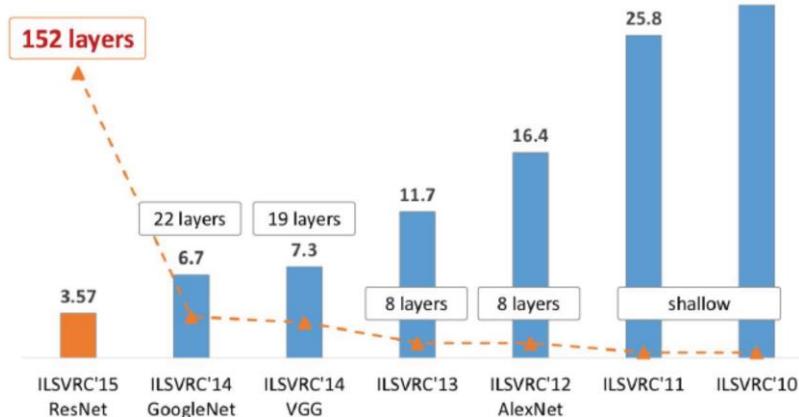
May 16, 2023

# 判别式深度学习

◆ 学习一个从输入到输出的映射网络



- 随机梯度
- 反向传播
- Dropout
- ...



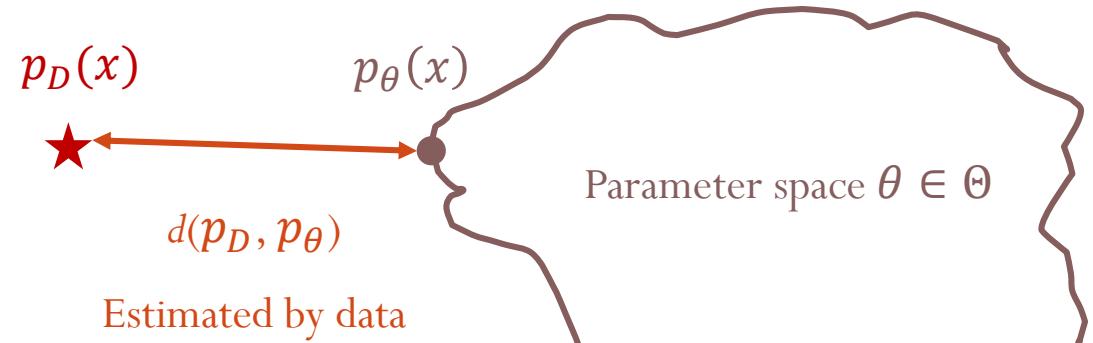
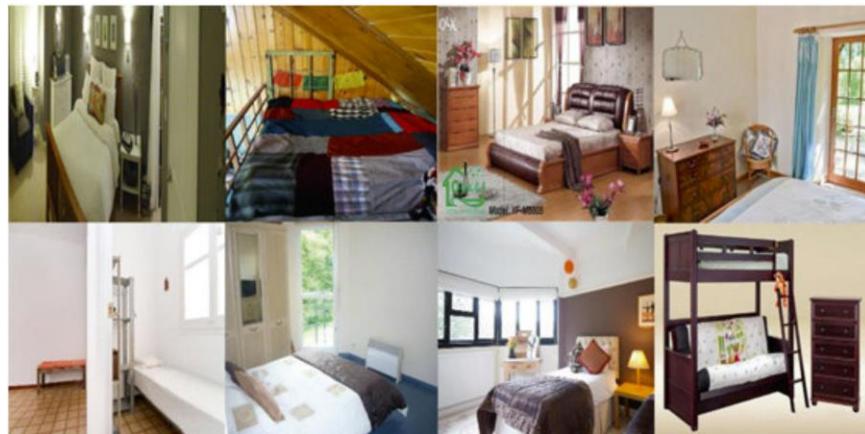
# 生成式人工智能 (Generative AI)

◆Richard P. Feynman: “What I cannot create, I don’t understand”

◆基本设定：

- 从无标注数据中，学习一个（近似）刻画数据分布的模型

$$x_i \sim_{iid} p_D(x), \quad i = 1, 2, 3 \dots$$



# 例1：ChatGPT横空出世，引爆公众对AIGC的关注

从“人工智障”到  
“人工智能”

有效捕捉  
用户意图

上下文理解、  
连续对话能力

对不正确的前  
提出质疑

屏蔽敏感话题、  
拒绝回答

能够体现  
人格化特质

.....

好的，我可以帮你写一个基于 macOS 的自动响闹钟程序。首先，确保你的 Mac 已经安装了 AppleScript，它是 macOS 系统上预装的脚本语言，用于编写自动化任务和应用程序。

下面是一个简单的 AppleScript 程序，可以在指定的时间播放一段音乐作为闹钟：

```
Copy code
set theAlarmTime to "8:00:00" -- 指定闹钟响起的时间 (时:分:秒)
set theMusic to "song.mp3" -- 指定闹钟音乐文件 (假设它位于同一目录下)

repeat
    set theCurrentTime to time string of (current date)
    if theCurrentTime is equal to theAlarmTime then
        play theMusic
    end if
    delay 1 -- 延迟一秒，循环检查闹钟时间
end repeat
```

将上面的代码保存为 `alarm.app`，然后在终端中运行：

写程序

当然，我可以帮你写一份关于金融科技安全现状与未来展望的演讲稿。以下是一份简短的范例，供你参考：

尊敬的各位嘉宾，大家好！

我很荣幸能在这场金融科技峰会上为大家分享一些关于当前金融科技安全现状与未来展望的观点。

随着金融科技的迅速发展，安全问题也越来越多受到关注。在过去的几年中，全球各地发生了许多金融科技安全事件，例如黑客攻击、数据泄露和网络诈骗等。这些事件不仅对个人和企业造成了损失，也对金融体系和经济发展造成了不良影响。

因此，保障金融科技安全已成为各国政府和企业的重要任务。我们需要采取多种措施来应对这一挑战，例如：

1. 加强技术创新。我们需要通过技术手段来应对安全威胁，例如使用人工智能、区块链和密码学等技术来提高数据保护和识别欺诈行为的能力。
2. 建立安全法规和标准。我们需要建立统一的安全标准和法规，以确保金融科技的发展符合安全要求，并且能够得到监管机构的有效监管。
3. 加强合作。金融科技安全是一个全球性的问题，需要各国政府、企业和国际组织之间的合作，共同应对这一挑战。

未来，金融科技将继续发挥着重要作用，同时也将面临更多的安全挑战。为了应对这些挑战，我们需要不断创新、加强合作，并且建立更加健全的安全体系。只有这样，我们才能够确保金融科技能够稳健、安全地发展，并且为人类社会带来更多的价值。

写演讲稿

C 我想让你假扮一个18岁的能言善辩的女生跟我聊天，你接下来只回复一个18岁可爱女生可能回复的内容，不要回复其他内容

你好，我是一个18岁的女生，我喜欢读书、看电影和旅游。我也喜欢结交新朋友，所以很高兴能跟你聊天。

C 你最近在读什么书呀

我最近在读一本叫《平凡的世界》的书。这本书讲述了一个普通人的人生故事，从出生到成长、工作和生活中的各种经历，我觉得很有感触。我也喜欢看一些小说和哲学书籍。

扮演虚拟角色聊天对话

## 例子2：Midjourney V5 – Creates Photorealistic Images

◆ 输入文本：「Portrait of an older man sat in a coffee shop, shot through a window.」



◆ 输入文本：「A pair of young Chinese lovers, wearing jackets and jeans, sitting on the roof, the background is Beijing in the 1990s, and the opposite building can be seen --v 5 --s 250 --q 2」



## 例子3：Gen-2 – Text to Video

◆ 输入文本：

「*The late afternoon sun peeking through the window of a New York City loft.*」



## 例子3：Gen-2 – Text+Image to Video

◆输入文本：

「*A low angle shot of a man walking down a street, illuminated by the neon signs of the bars around him.*」



## 生成式人工智能的历史

- ◆ 用概率模型刻画数据分布，在概率统计中由来已久，学习方法包括最大似然估计、矩匹配等
- ◆ 在人工智能中，早期的朴素贝叶斯、高斯混合模型、隐马尔可夫模型等，均是生成式AI的代表
- ◆ 随着深度学习的进展，深度生成模型取得突破 ...

## 深度生成模型的关键进展

- ◆ 2006年，深度置信网络（DBN）被提出——首个深度学习模型！
- ◆ 2013年，变分自编码器和对抗生成网络取得突破
- ◆ 2015年，流模型（Flows）的基本原理被提出
- ◆ 2015年，扩散概率模型的基本原理被提出
- ◆ 2017年，谷歌提出Transformer自回归神经网络
- ◆ 2021年，OpenAI发布扩散概率模型超过对抗生成网络
- ◆ 2022年3月，OpenAI发布DALLE2大规模文图生成系统
- ◆ 2022年6月，Stability AI发布Stable Diffusion开源项目
- ◆ 2022年11月，OpenAI发布ChatGPT对话系统
- ◆ 2023年3月，OpenAI发布GPT4多模态理解模型
- ◆ 2023年3月，OpenAI发布Copilot X智能代码生成模型
- ◆ ...

## 深度生成模型的基本原理

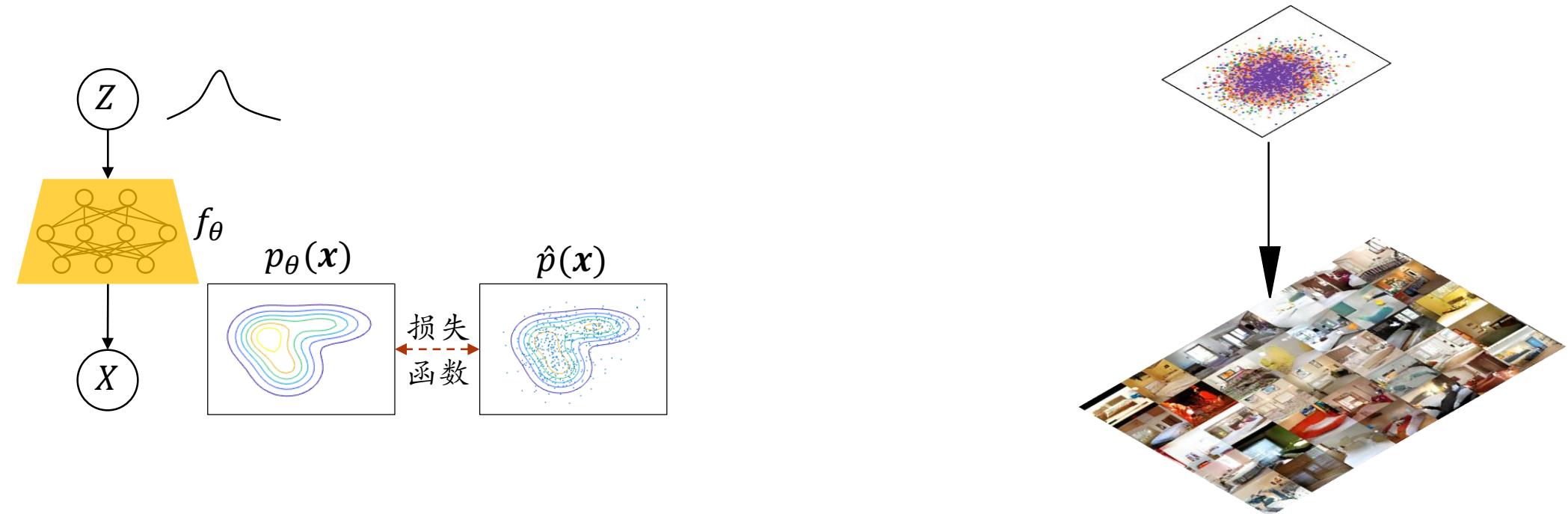
- ◆ More flexible by using differential function mapping between random variables
- ◆ If  $z$  is uniformly distributed over  $(0, 1)$ , then  $y = f(z)$  has the distribution

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

- where  $p(z) = 1$
- ◆ This trick is widely used to draw samples from exponential family distributions (e.g., Gaussian, Exponential)

## 深度生成模型的基本原理

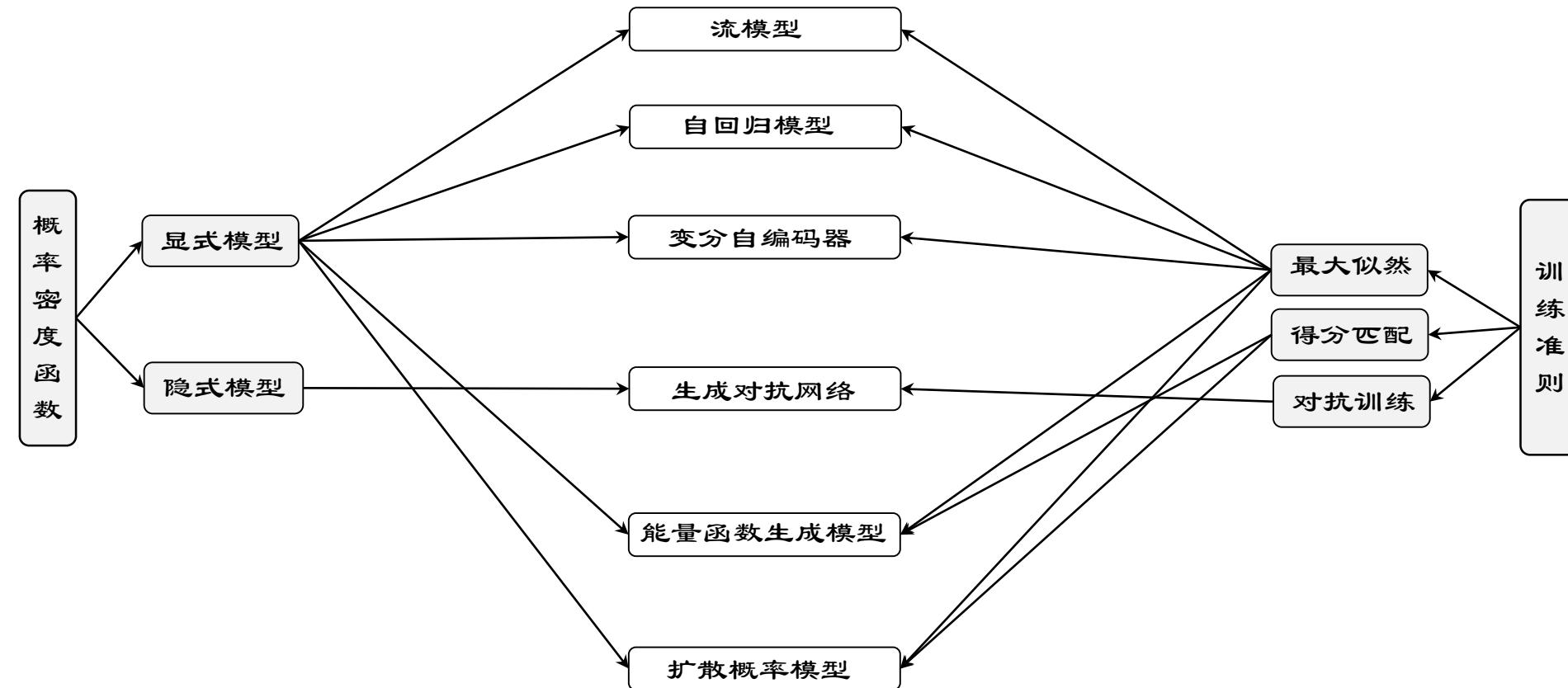
- ◆ 使用可微深度神经网络拟合从简单分布到复杂数据分布的映射



- 完全无监督的训练，或者有部分标注数据的训练
- 具有广泛的应用范围

# 深度生成模型的主要类型

◆ 从建模和学习两个维度进行划分



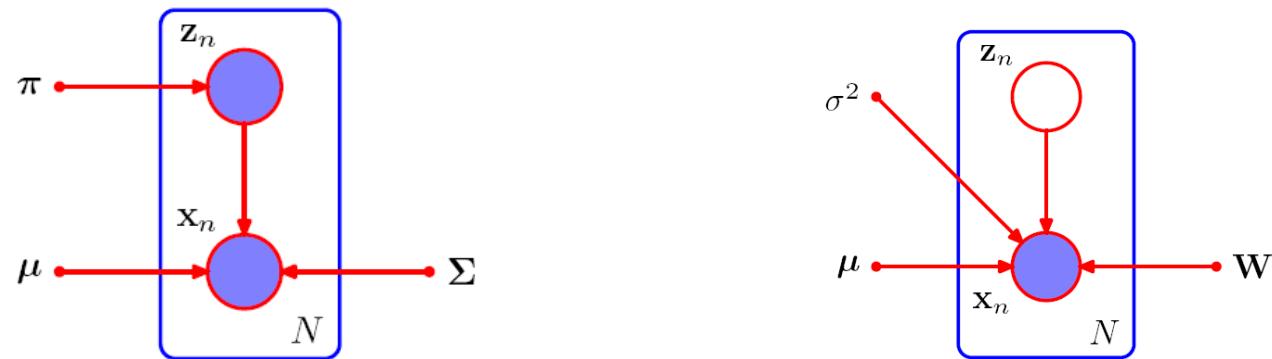
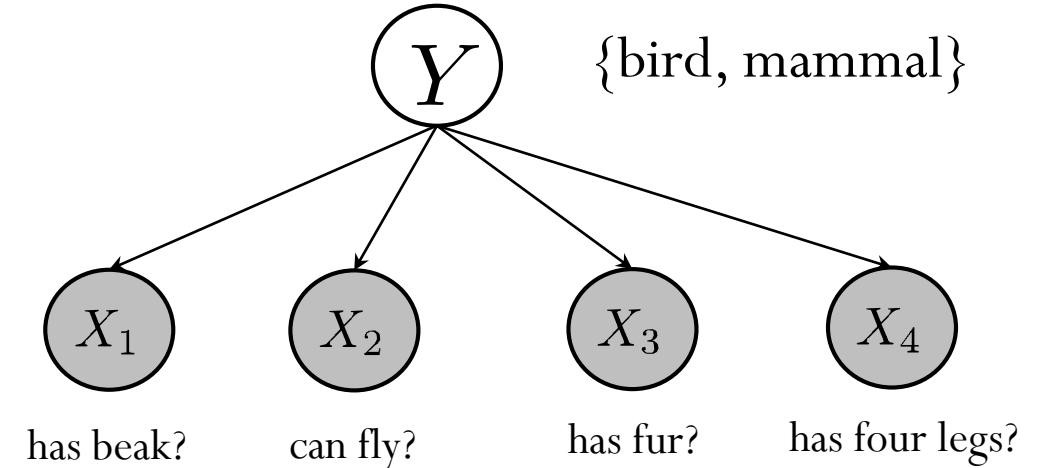
# The Generative AI Application Landscape



APPLICATION LAYER	Marketing (content)						
	Sales (email)	Code generation	Image generation				
	Support (chat / email)	Code documentation	Consumer / Social				
	General writing	Text to SQL	Media / Advertising				
	Note taking	Web app builders	Design	Voice Synthesis	Video editing / generation	3D models / scenes	Gaming
	Other						RPA
							Music
TEXT		CODE	IMAGE	SPEECH	VIDEO	3D	OTHER
MODEL LAYER	OpenAI GPT-3	OpenAI GPT-3	OpenAI Dall-E 2	OpenAI	Microsoft X-CLIP	DreamFusion	TBD
	DeepMind Gopher	Tabnine	Stable Diffusion			NVIDIA GET3D	
	Facebook OPT	Stability.ai	Craiyon			MDM	
	Hugging Face Bloom						
	Cohere						
	Anthropic						
	AI2						
	Alibaba, Yandex, etc.						

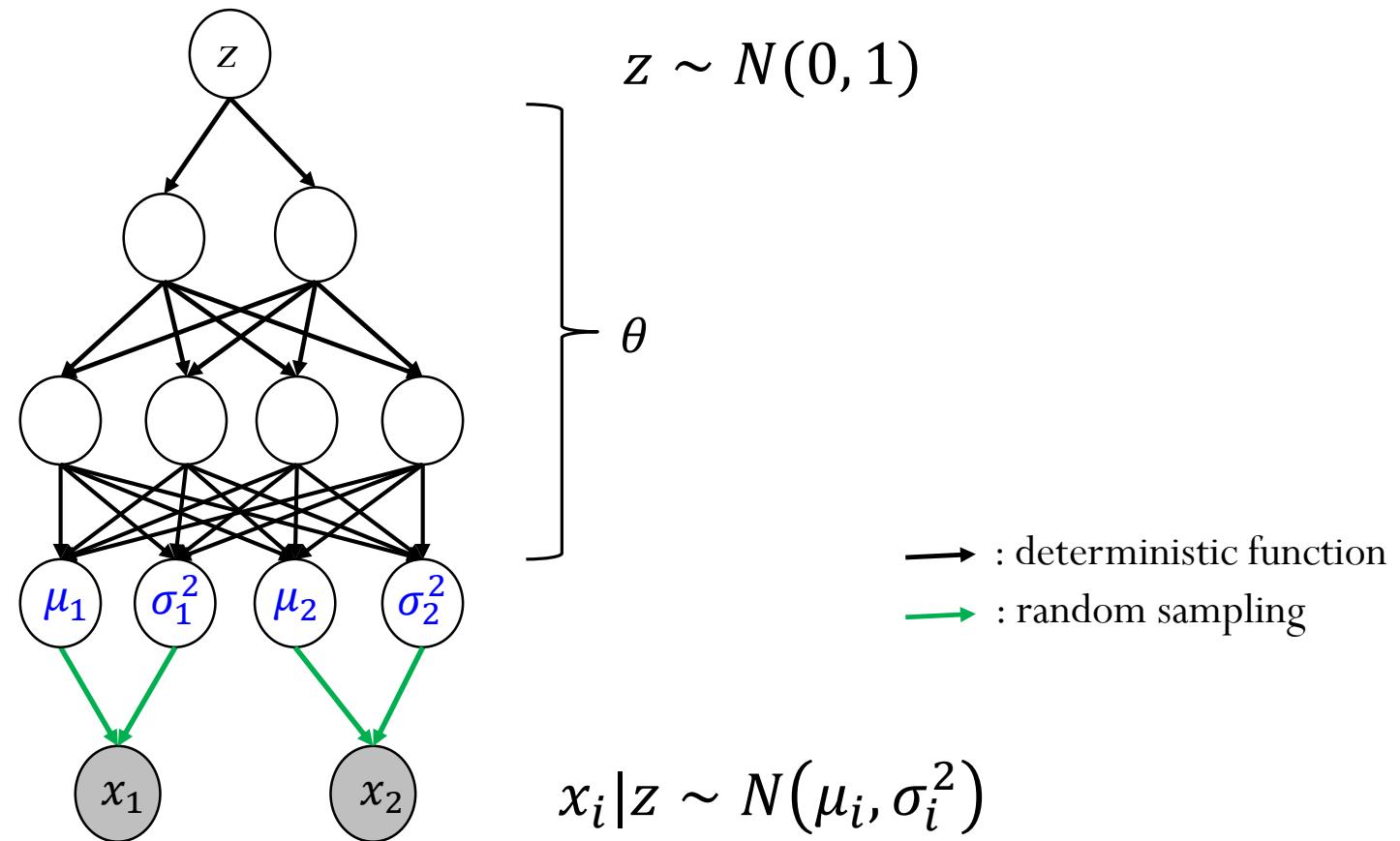
# Generative models are your old friends

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



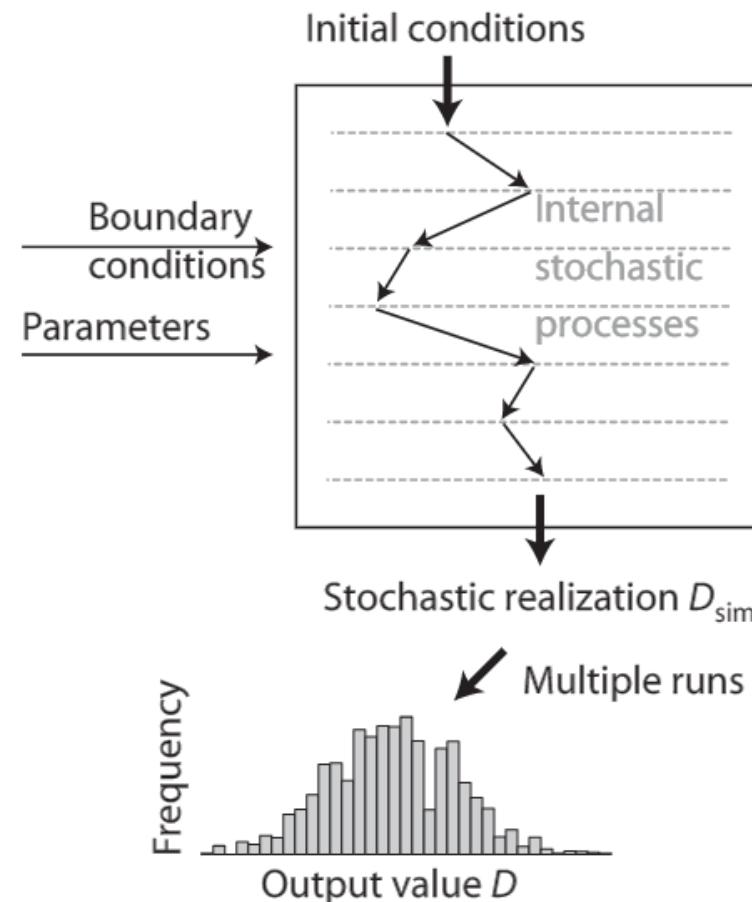
## An example with MLP

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



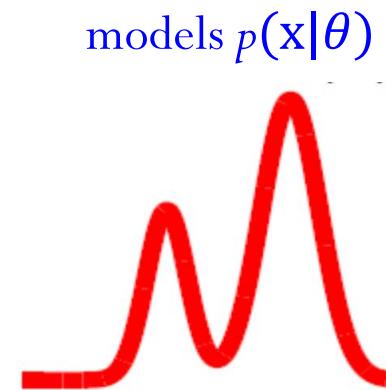
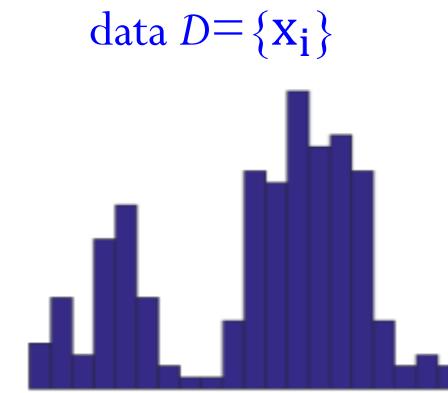
# Implicit Deep Generative Models

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



# Learning Deep Generative Models

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



- ◆ Find a model that minimizes

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

# 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

- ◆ Moment-matching:

- draw samples from  $p$ :

$$\hat{D} = \{y_i\}_{i=1}^M, \text{ where } y_i \sim p(x|\theta)$$

- Kernel MMD:

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

- rich enough to distinguish any two distributions in certain RKHS

# Variational Bayes

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

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

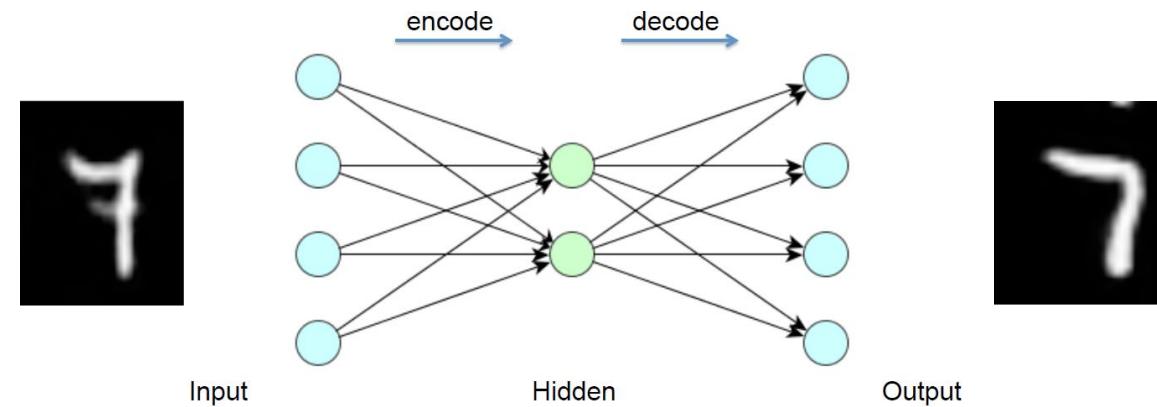
- ◆ Log-integral/sum is annoying to handle directly
- ◆ Derive a variational lower bound  $L(\theta, \phi, \mathbf{x})$

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

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

**reconstruction term**                           **prior regularization**

# Recap: Auto-Encoder

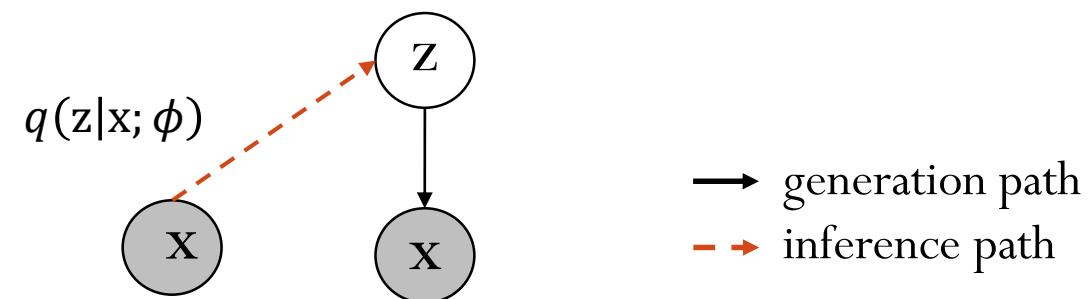


- ◆ Encoder:  $h = s(Wx + b)$
- ◆ Decoder:  $x' = s(W'h + b')$
- ◆ Training: minimize the reconstruction error (e.g., square loss, cross-entropy loss)
- ◆ Denoising AE: randomly corrupted inputs are restored to learn more robust features

# Auto-Encoding Variational Bayes (AEVB)

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

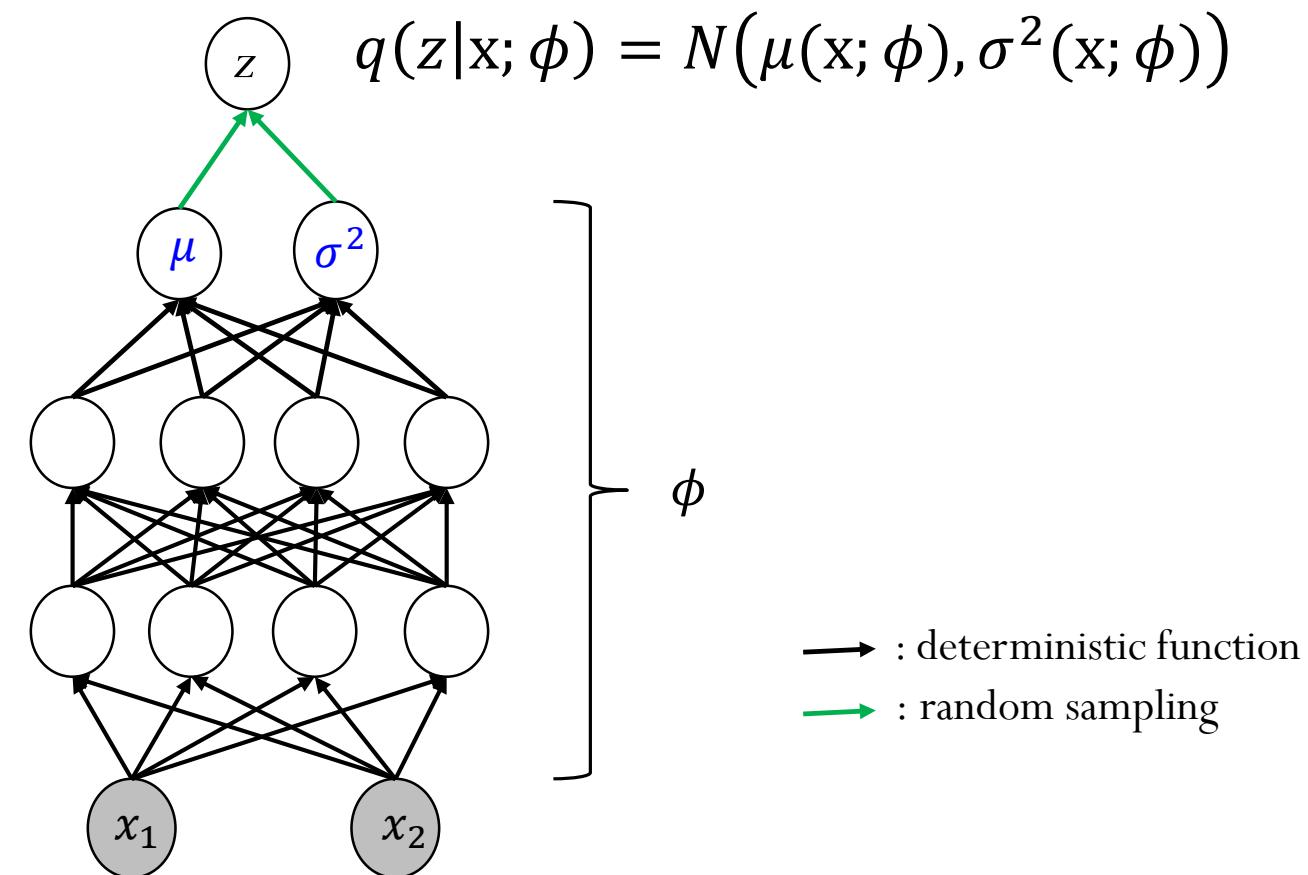
$$q(z|x; \phi) \approx p(z|x; \theta)$$



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

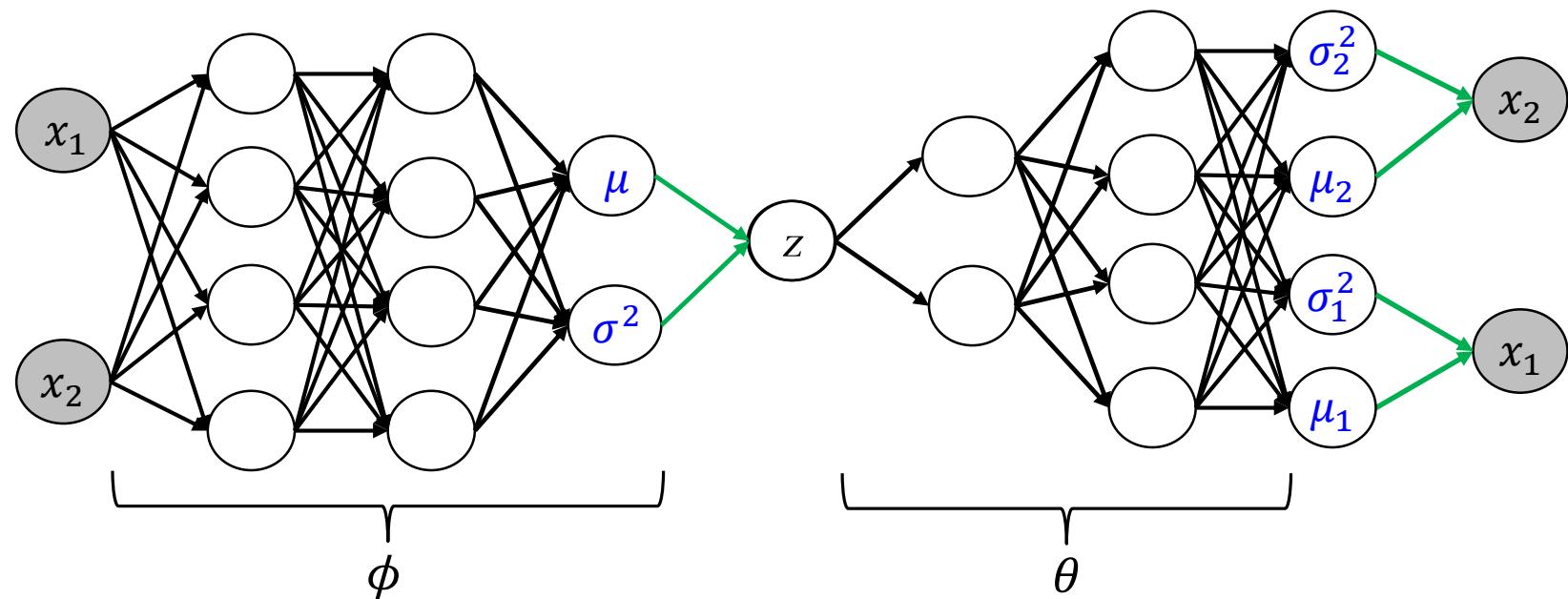
# The Encoder Network

A feedforward NN + Gaussian



# The Complete Auto-encoder

- ◆ The Q-P network architecture:



→ : deterministic function

→ : random sampling

# Stochastic Variational Inference

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

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

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

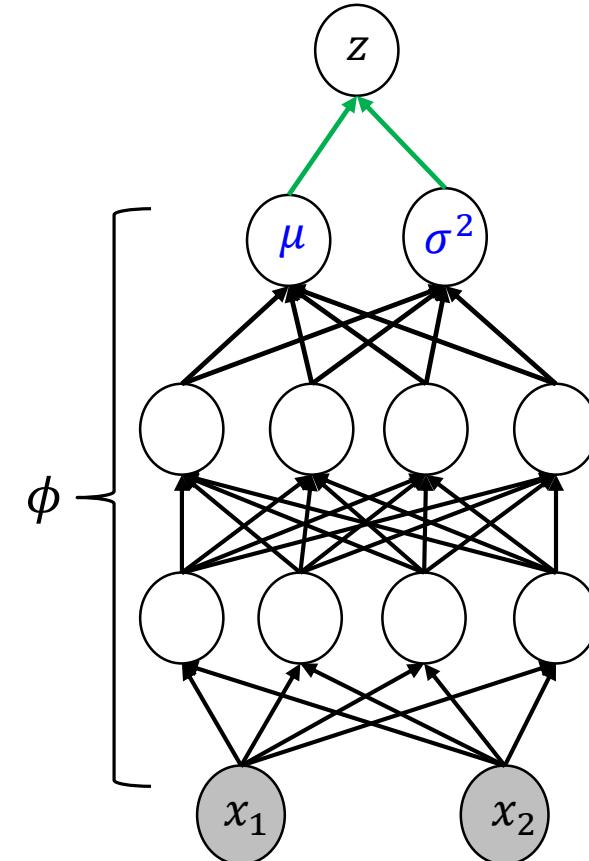
$$L(\theta, \phi, D) = \sum_i \mathbf{E}_{q(z_i|x_i; \phi)}[\log p(x_i|z_i; \theta)] - \text{KL}(q(z_i|x_i; \phi) \| p(z_i; \theta))$$

- Use stochastic gradient methods to handle large datasets
- Random mini-batch
  - for each  $i$ , infer the posterior  $q(z_i|x_i; \phi)$ ; As we parameterize as a neural network, this in fact optimizes  $\phi$
- ◆ *However, calculating the expectation and its gradients is non-trivial, often intractable*

## Example with Gaussian Distributions

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

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



# Example with Gaussian Distributions

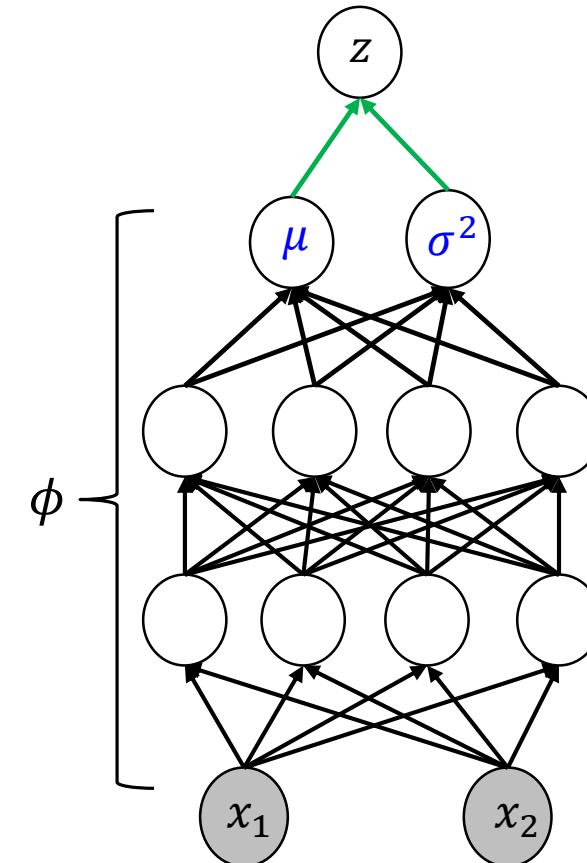
- ◆ Use  $N(0, 1)$  as prior for  $z$ ;  $q(z|x; \phi)$  is Gaussian with parameters  $(\mu(x; \phi), \sigma^2(x; \phi))$  determined by NN
  - The expected log-likelihood

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

- If the likelihood is Gaussian

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

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



## Example with Gaussian Distributions

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

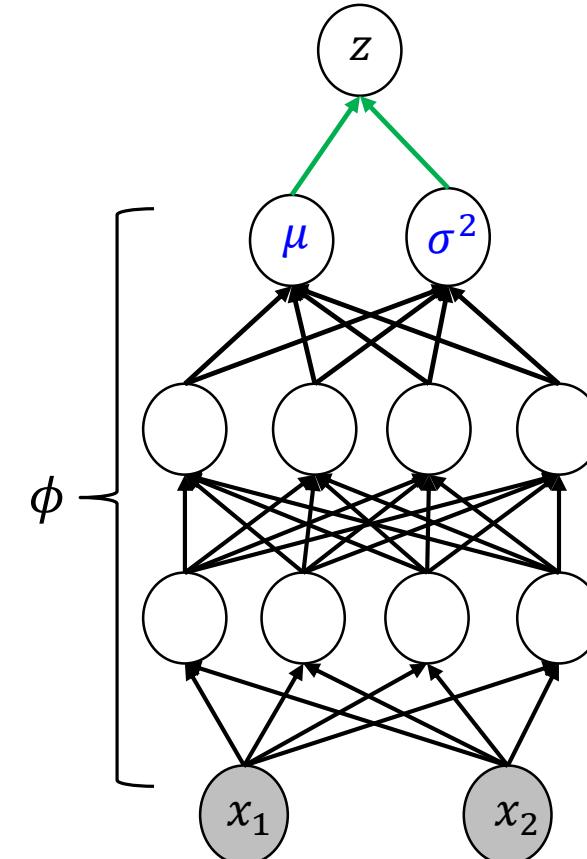
$$\mathbf{E}_{q(z|x;\phi)}[\log p(x|z; \theta)]$$

- Approximate via Monte Carlo methods

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

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

- An unbiased estimator



# Example with Gaussian Distributions

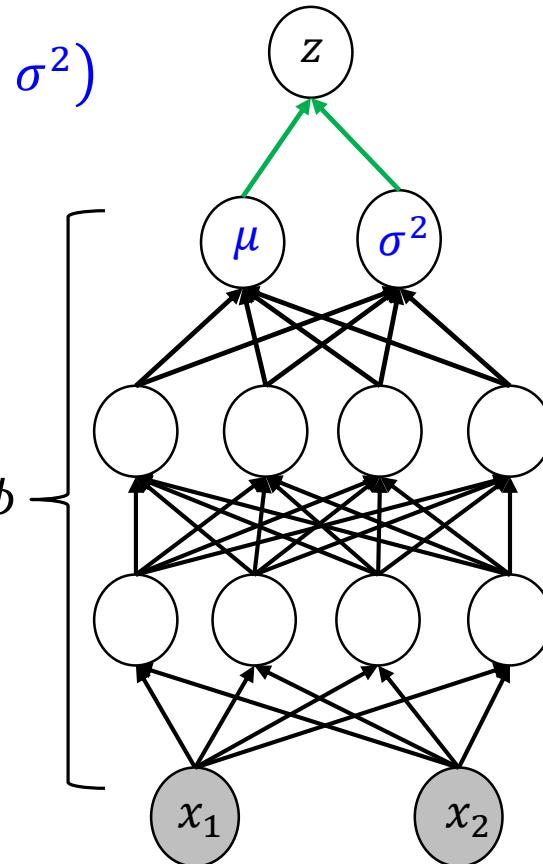
- ◆ The KL-regularization term (closed-form):

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

- Easy to calculate gradient
- ◆ The expected log-likelihood term (MC estimate)

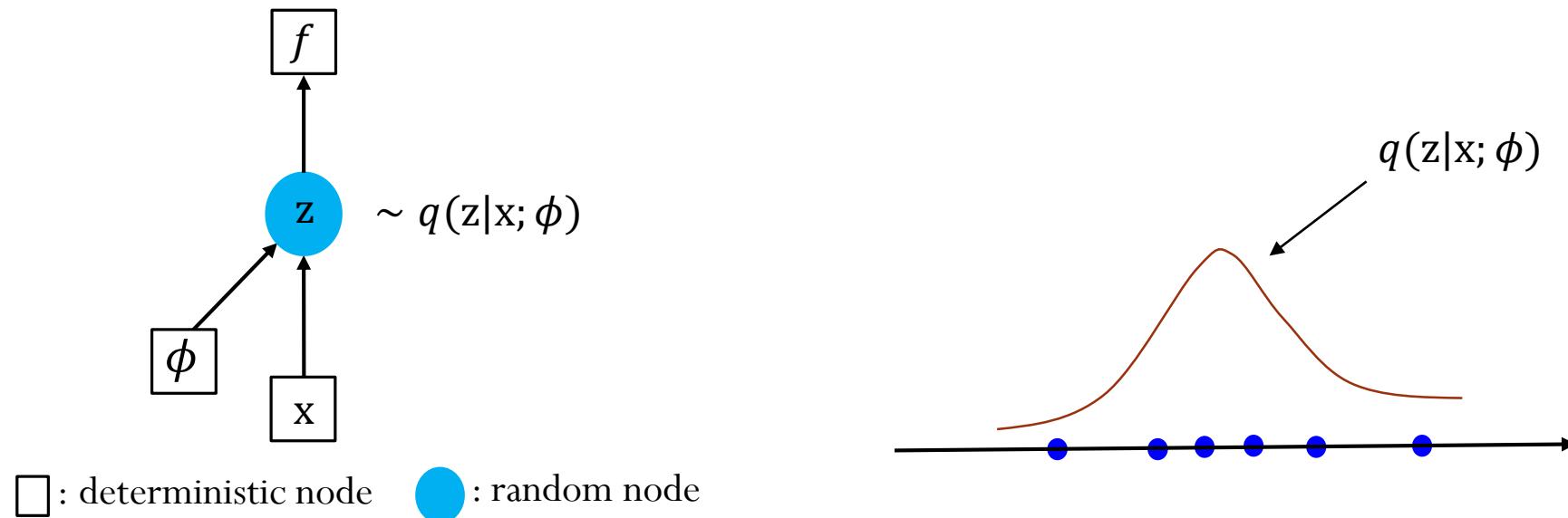
$$\mathbf{E}_{q(z|x;\phi)}[\log p(x|z;\theta)] \approx \frac{1}{L} \sum_k \log p(x|z^{(k)})$$
$$z^{(k)} \sim q(z|x; \phi)$$

- 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



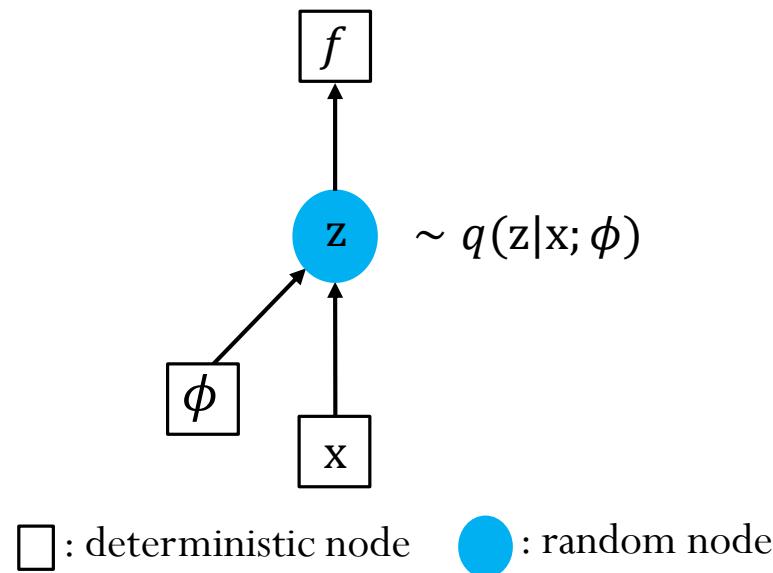
$$z^{(k)} \sim N(\mu(x, \phi), \sigma^2(x, \phi))$$

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

Cannot back-propagate through a  
randomly drawn number

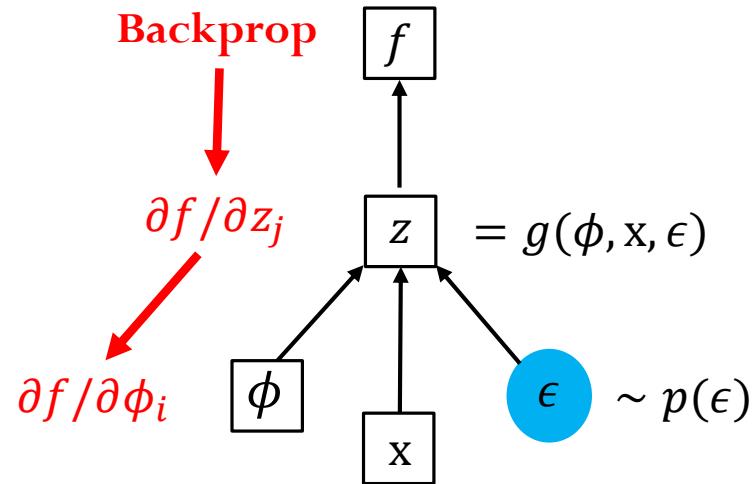
# Reparameterization Trick

- ◆ Backpropagation not possible through random sampling



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

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

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

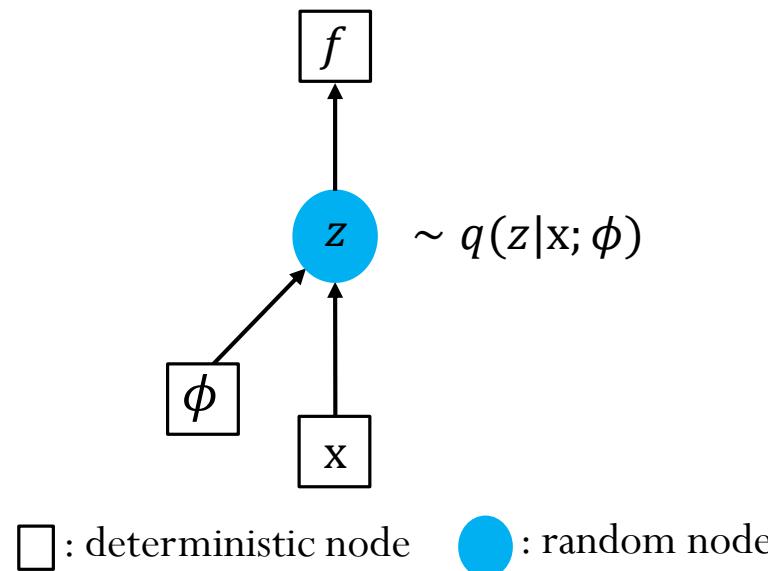
- ◆ Monte Carlo estimate:

$$\begin{aligned} L(\theta, \phi, \mathbf{x}) &\approx \frac{1}{L} \sum_k \log \frac{p(z^{(k)}, \mathbf{x}; \theta)}{q(z^{(k)}|\mathbf{x}; \phi)} \\ z^{(k)} &\sim q(z|\mathbf{x}; \phi) \end{aligned}$$

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

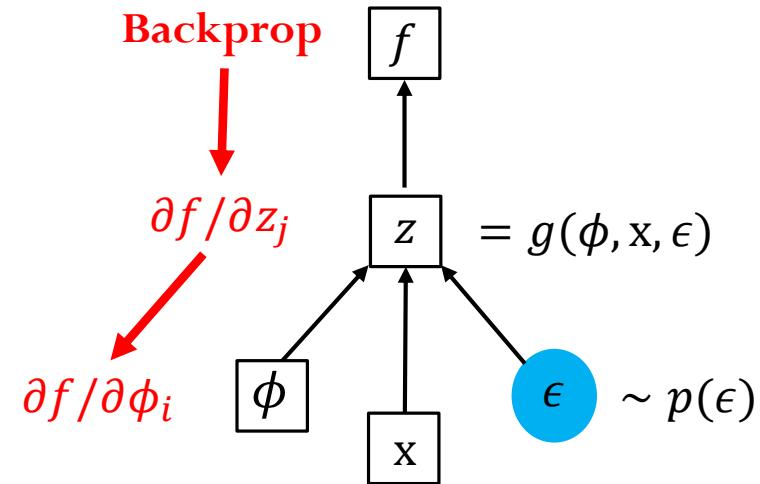
# Reparameterization Trick

- ◆ Backpropagation not possible through random sampling



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

Cannot back-propagate through a randomly drawn number



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

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

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

# Reparam-Trick Summary

- ◆ The VAE bound

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

- Reparameterized as

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

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

- ◆ The gradients are

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

- Back-prop is applied over the deep NN
- Similar for  $\phi$

# Importance Weighted Auto-Encoder (IWAE)

- ◆ The VAE lower bound of log-likelihood

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

- ◆ A better variational lower bound (IWAE)

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

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

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

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

# Reparametrization Trick

- ◆ The IWAE bound:

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

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

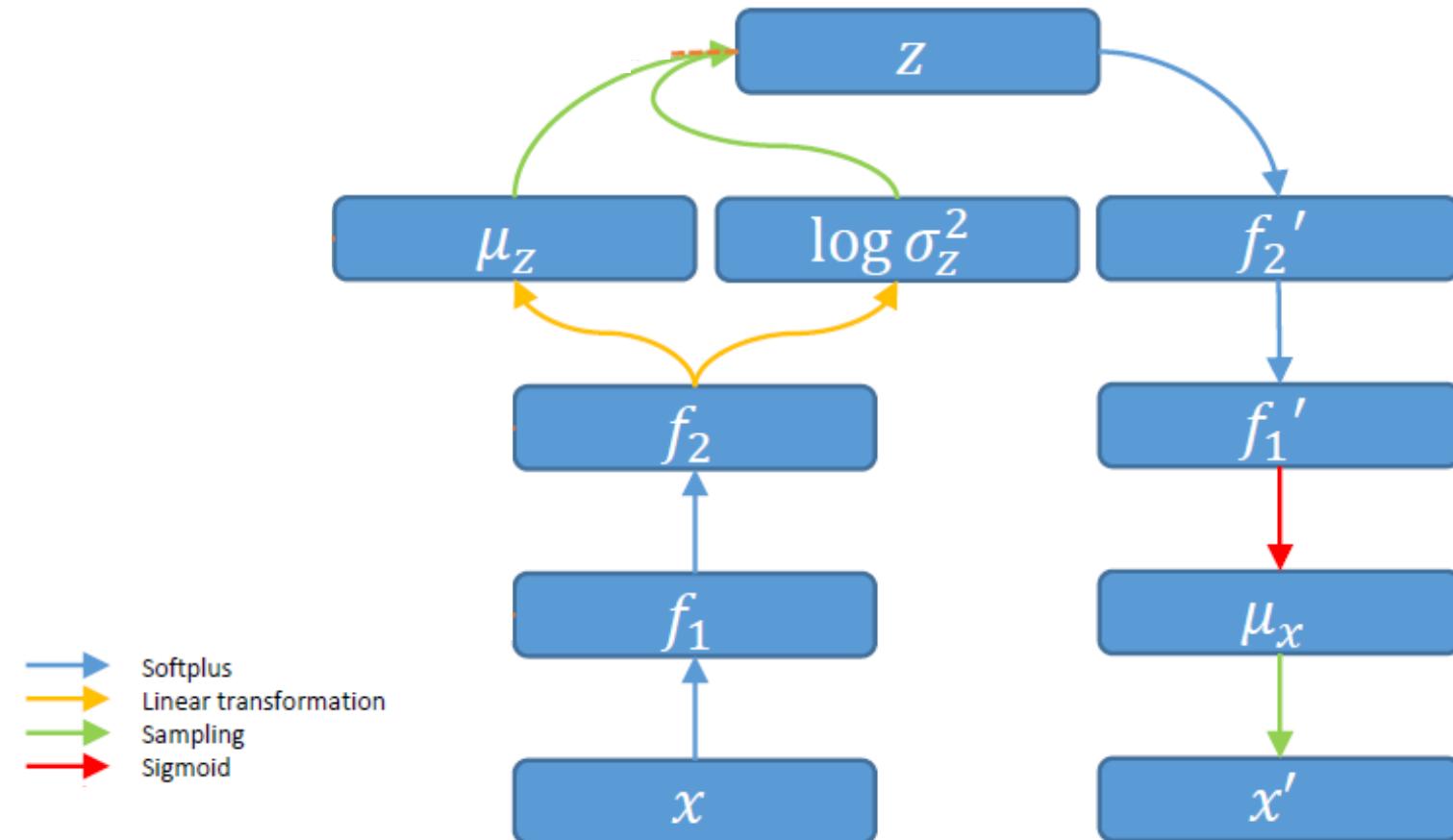
- ◆ Reparameterization form:

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

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

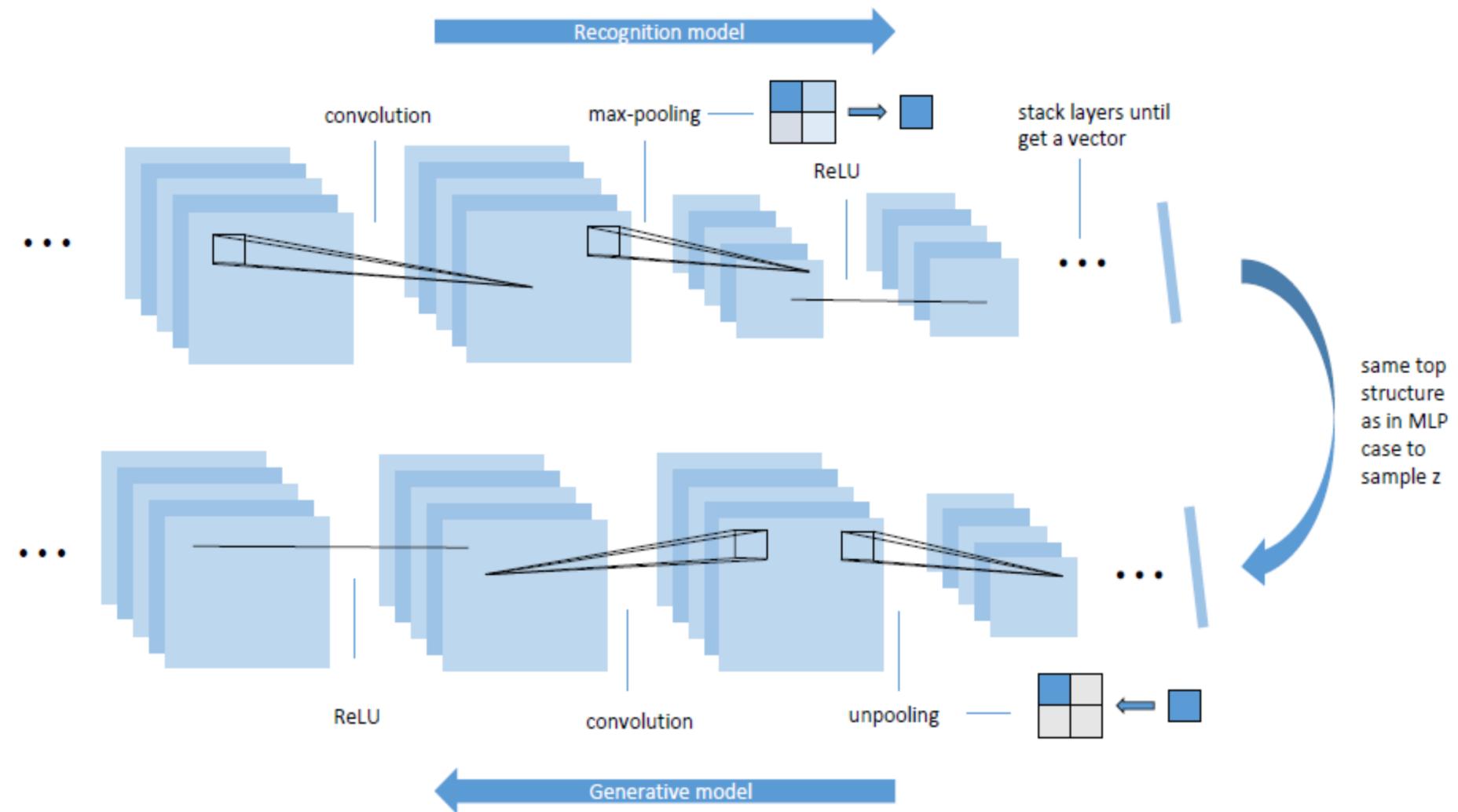
- The gradient can be calculated as in VAE

## 2-Layer MLP: Q-P network architecture



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

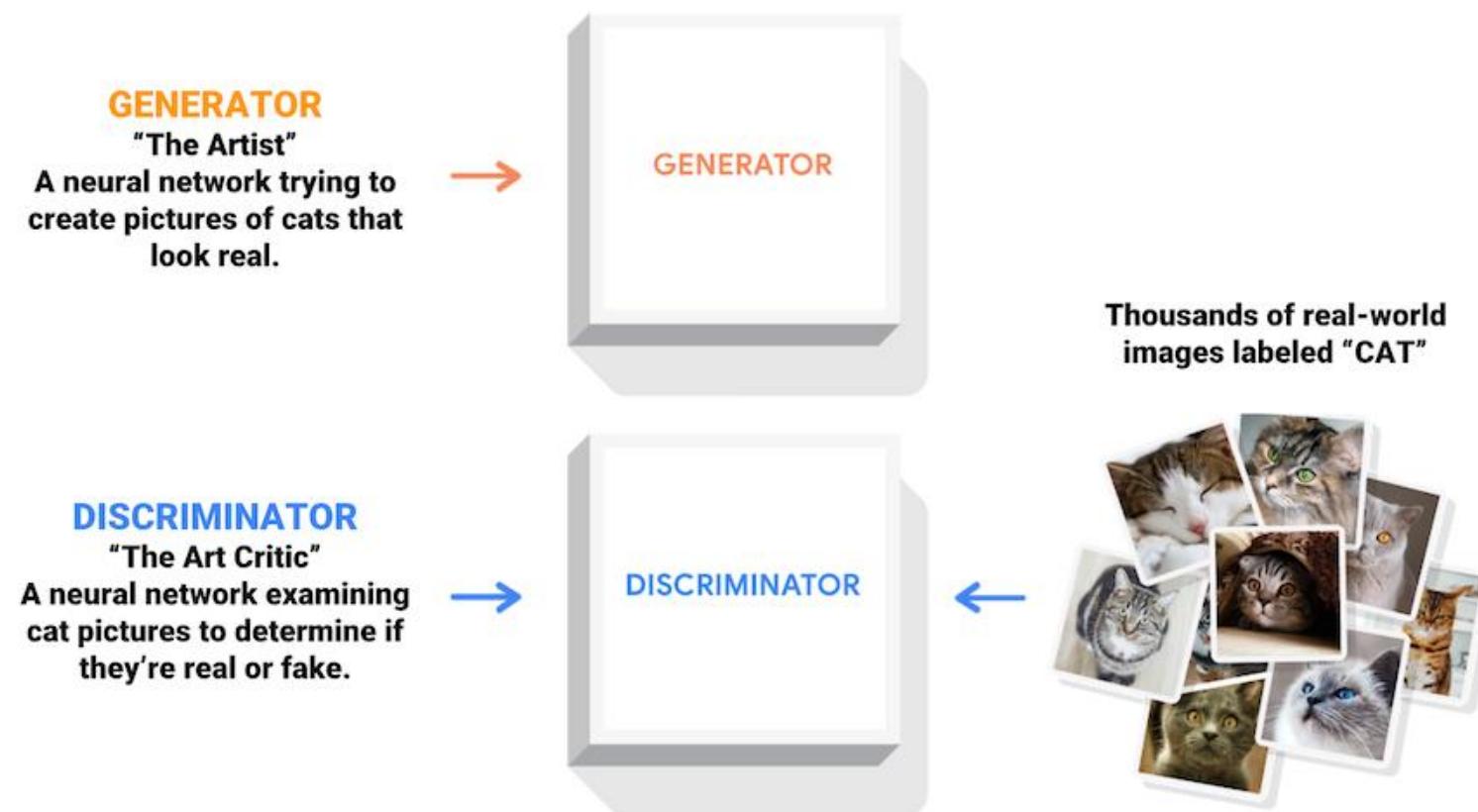
# 5-Layer CNN: Q-P network architecture



# 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.

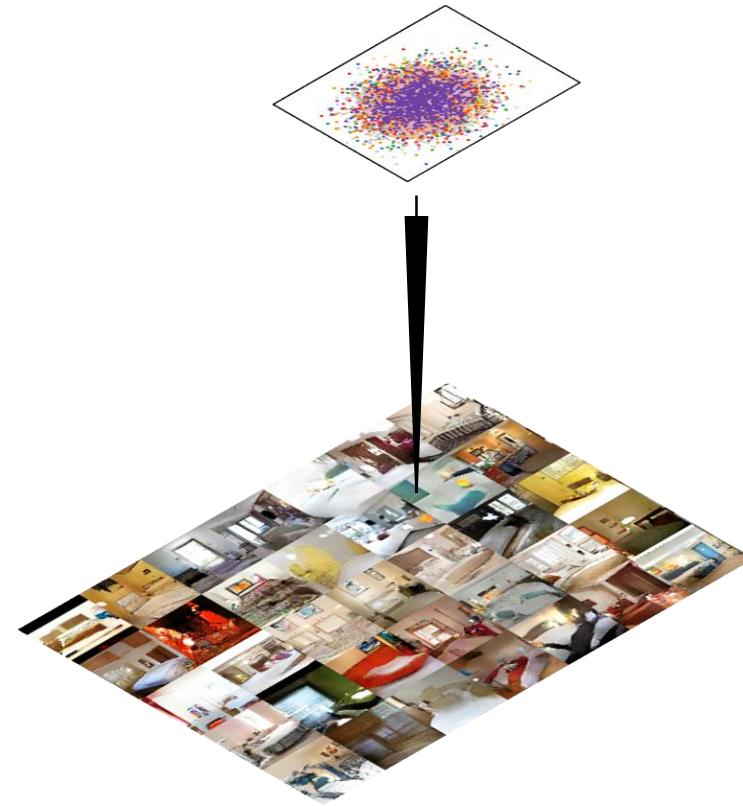
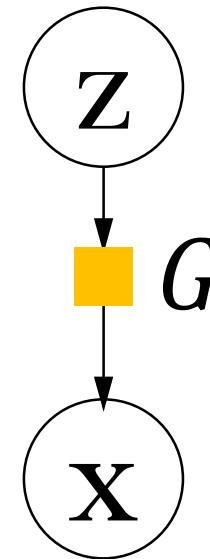
# The “Artist” and “Critic” Argument



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

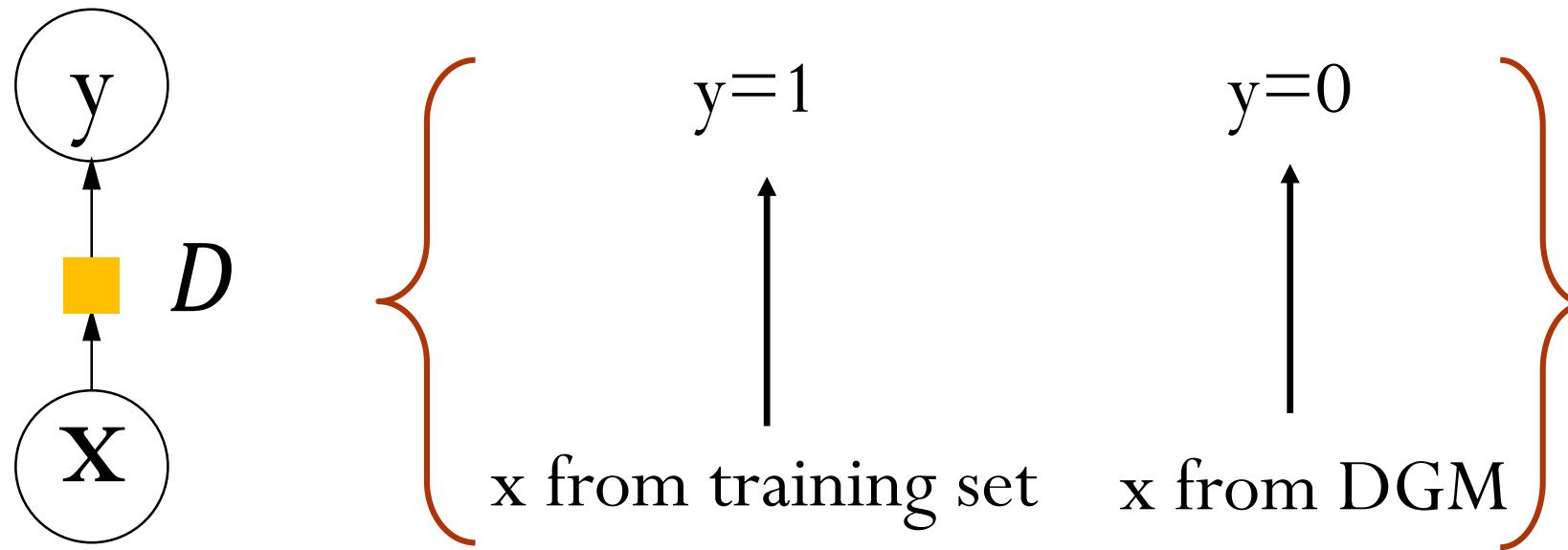
# GAN – architecture

- ◆ 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!

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



x from training set       $G(z)$  from generator

- $D(x) = p(y=1 | x)$
- 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:

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

↑

$G(z)$  is a sample from generator

- Or minimize the likelihood of being fake:

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

# GAN – objective

- ◆ Minimax objective function

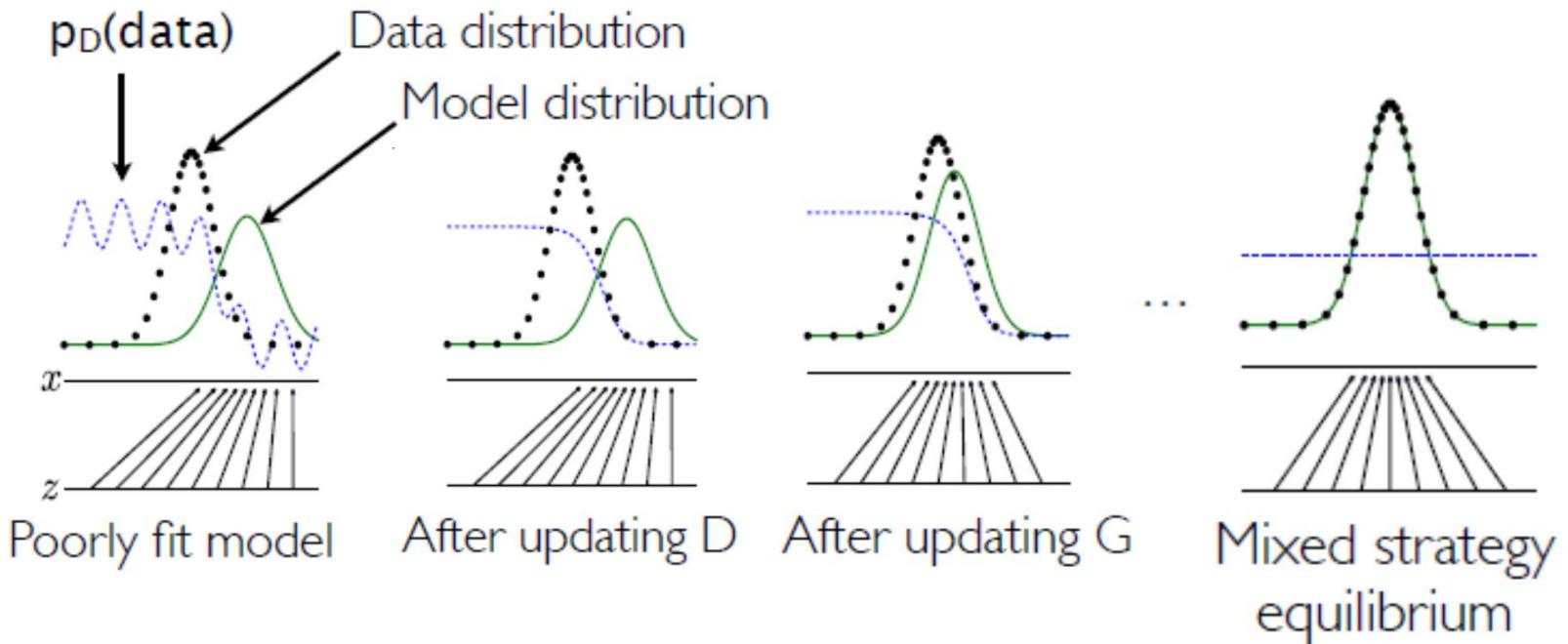
$$\min_G \max_D \mathbf{E}_{p_{\text{data}}(x)}[\log D(x)] + \mathbf{E}_{p(z)} \left[ \log (1 - D(G(z))) \right]$$

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

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

- 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

## More Distances on Distributions

- ◆ Kullback-Leibler (KL) divergence

$$KL(\mathbb{P}_r \parallel \mathbb{P}_g) = \int \log \left( \frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x)$$

- ◆ Jensen-Shannon (JS) distance

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \parallel \mathbb{P}_m) + KL(\mathbb{P}_g \parallel \mathbb{P}_m)$$

- where  $\mathbb{P}_m = (\mathbb{P}_r + \mathbb{P}_g)/2$

- ◆ Earth-Mover (EM) distance or Wasserstein-1 distance

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

- the “cost” of the optimal transport plan

# GAN – objective (revisited!)

- ◆ Minimax objective function

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

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

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

- Then, optimize over G is equivalent to minimize the JS-divergence between  $p_{\text{data}}(x)$  and  $p_{\text{model}}(x)$ !

# An Example: Distance Matters!

- ◆ Setup:
  - A uniform distribution  $Z \sim U[0, 1]$
  - $P_0$  is the distribution of  $(0, Z) \in \mathbb{R}^2$
  - Let  $g_\theta(z) = (\theta, z)$
- ◆ Then, we have
  - $W(P_0, P_\theta) = |\theta|,$
  - $JS(P_0, P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$
  - $KL(P_\theta \| P_0) = KL(P_0 \| P_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$

# An Example: Distance Matters!

## ◆ Setup:

- A uniform distribution
- $P_0$  is the distribution of
- Let

$$Z \sim U[0, 1]$$

$$(0, Z) \in \mathbb{R}^2$$

$$g_\theta(z) = (\theta, z)$$

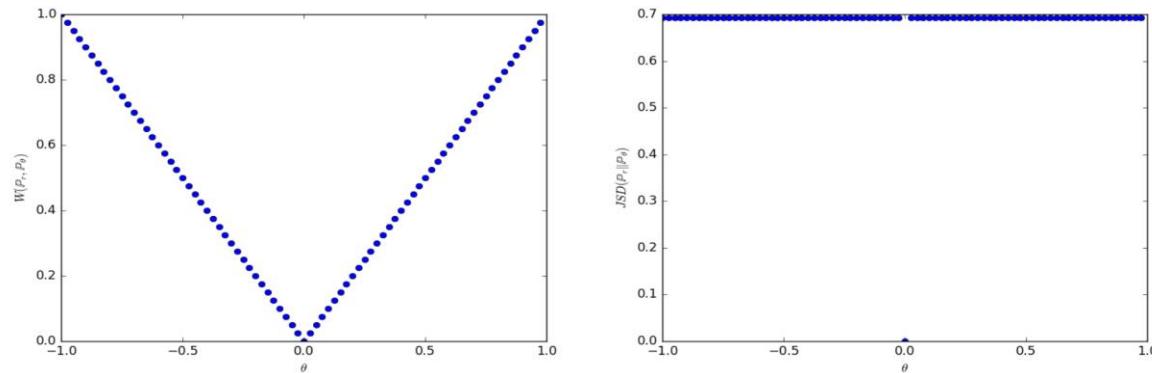


Figure 1: These plots show  $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$  as a function of  $\theta$  when  $\rho$  is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

The example gives us a case where we can learn a probability distribution over a low dimensional manifold by doing gradient descent on the EM distance. This cannot be done with the other distances because the resulting loss function is not even continuous.

# Practice of Training GANs

- ◆ Early stopping of discriminator training to avoid overfitting
  - Multiple updates of  $D$
  - One update of  $G$
- ◆ Gradient saturation
- ◆ Update  $G$  by maximizing

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

# Wasserstein GAN

- ◆ A new objective with Wasserstein distance, which is continuous everywhere and differentiable almost everywhere under mild conditions:

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

- where  $D$  is a 1-Lipschitz function
- under an optimal discriminator (critic), minimizing the value function w.r.t  $G$  minimizes the **W-distance between  $p_{\text{data}}(x)$  and  $p_{\text{model}}(x)$ !**
- Weight clipping is applied to ensure 1-Lipschitzness.

# An Illustration with WGAN

If we train a GAN discriminator and a WGAN critic till optimality, **gradient vanishing** happens in GAN!

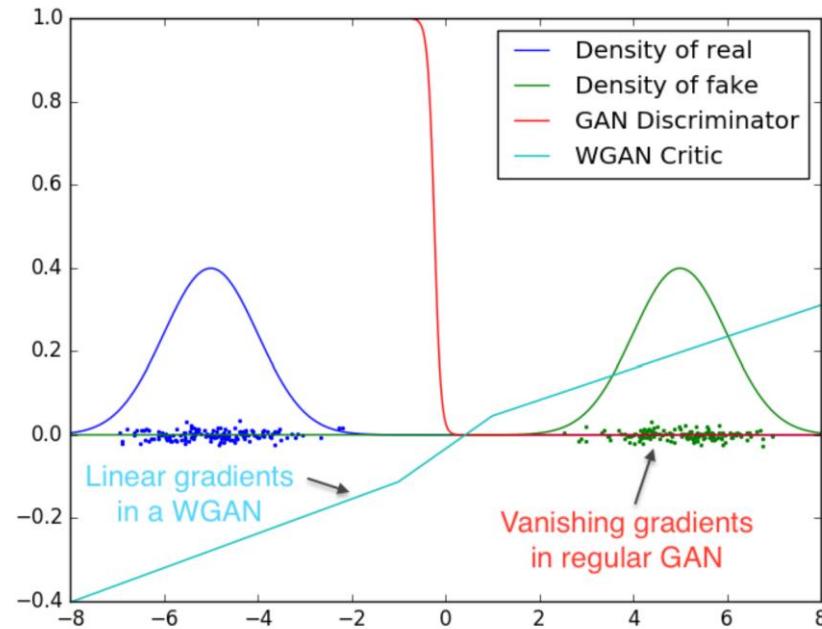


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

# Understand and Stabilize GAN via Control Theory

- ◆ Minimax formulation of GAN

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

- Optimal strategy of the discriminator for any  $p_{\text{model}}(x)$  is  $D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$
- ◆ However, training GANs is typically unstable, with mode collapsing!
- ◆ Some analysis based on  $D^* = \arg \max_D V(G, D); G^* = \arg \min_G V(G, D^*)$ 
$$D^t = D^{t-1} + \frac{\partial V(G^{t-1}, D^{t-1})}{\partial D^{t-1}}; \quad G^t = G^{t-1} + \frac{\partial V(G^{t-1}, D^t)}{\partial G^{t-1}}$$
- ◆ A gap with practice!

# Understand and Stabilize GAN via Control Theory

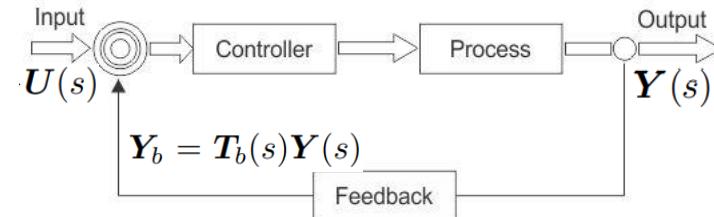
Control theory provides a new perspective to understand and analyze (Kailath, 1980)

- Laplace Transformation:  $\mathcal{F}(h)(s) = \int_0^{\infty} h(t)e^{-st} dt = H(s).$
- An important property:  $\mathcal{F}\left(\frac{dh(t)}{dt}\right) = s\mathcal{F}(h) \Rightarrow$
- Negative real parts of poles (roots of denominator of  $T$ ) => stability!

Closed-loop control (CLC) provides a general recipe to stabilize (Kailath, 1980)

- Adjust the input according to the output:

$$\mathbf{Y}(s) = \frac{\mathbf{T}(s)}{1 + \mathbf{T}_b(s)\mathbf{T}(s)} \mathbf{U}(s)$$



$$\mathbf{T}_b$$

- with properly designed  $\mathbf{T}_b$ , the poles of the dynamic will have negative real parts, thus stabilized!

# Stability Comparison

- ◆ Either theoretically or empirically stabilized

Table 1: The stability characters for the widely-used GANs. Please refer to Appendix A for detailed derivation, which adopts the local linearization technique introduced in Sec. 3.3. With CLC, the training dynamics of Dirac GANs are stable theoretically (see Fig. 1 and Appendix A), and those of normal GANs are stable empirically (see Fig. 2).

	$T_{\mathcal{D}}(s)$	Stability Dirac GAN/normal GAN	$T_{c\mathcal{D}}(s)$	Stability with CLC Dirac GAN/normal GAN
WGAN	$s/(s^2 + 1)$	✗/✗	$1/(s^2 + \lambda s + 1)$	✓/✓
Hinge-GAN	$s/(s^2 + 1)$	✗/✗	$1/(s^2 + \lambda s + 1)$	✓/✓
SGAN	$2s/(4s^2 + 2s + 1)$	✓/✗	$1/(4s^2 + (2\lambda + 2)s + 1)$	✓/✓
LSGAN	$s/(s^2 + 4s + 1)$	✓/✗	$1/(s^2 + (\lambda + 4)s + 1)$	✓/✓

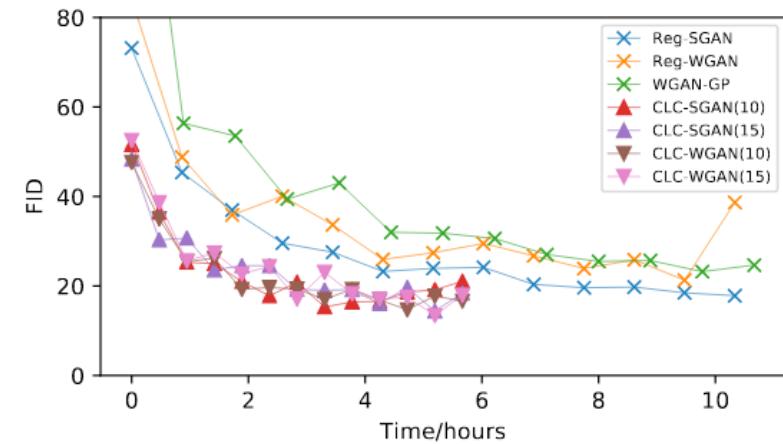
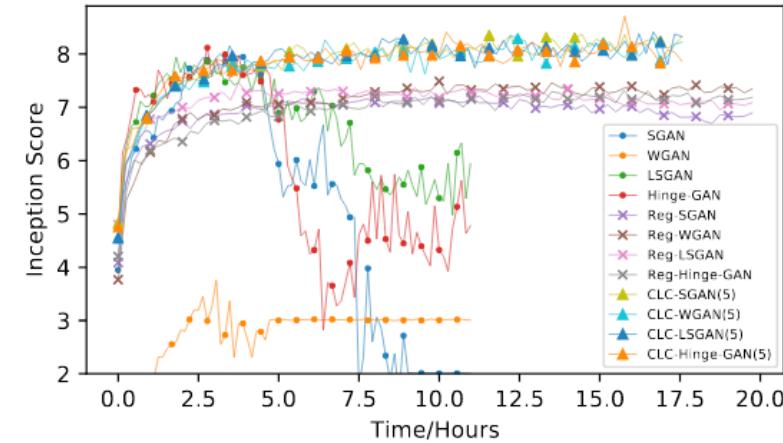
# Results

## ◆ The stability:

- Top: Inception Score on CIFAR10
- Bottom: FID on CelebA

## ◆ The inception score on CIFAR10

Method	WGAN	SGAN	Hinge
LR-GAN <sup>†</sup>	-	7.17	-
SN-GAN <sup>‡</sup>	-	-	8.22
CR-GAN <sup>§</sup>	-	8.40	-
Gradient Penalty	7.82	-	-
Reg-GAN	7.34	7.37	7.37
CLC-GAN(2)	$8.42 \pm .06$	$8.28 \pm .05$	$8.49 \pm .08$
CLC-GAN(5)	<b><math>8.49 \pm .07</math></b>	$8.44 \pm .08$	<b><math>8.54 \pm .03</math></b>
CLC-GAN(10)	$8.38 \pm .10$	<b><math>8.47 \pm .09</math></b>	$8.46 \pm .00$
SN-GAN	3.29	8.17	8.28
CLC-SN-GAN(0.1)	$8.14 \pm .02$	$8.30 \pm .09$	<b><math>8.54 \pm .03</math></b>



# Flow-based Models

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

$$z \sim p(z)$$

$$x = g(z)$$

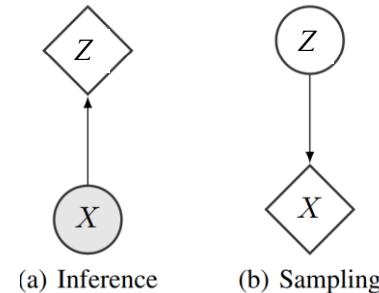
- ◆ The inference network for the latent variable is:

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

- ◆ 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

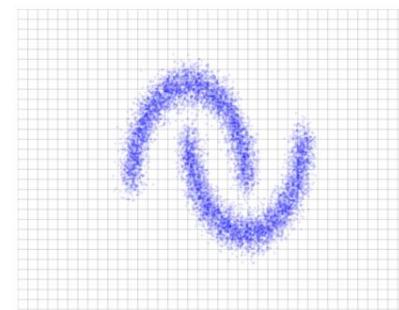


# An illustrative example

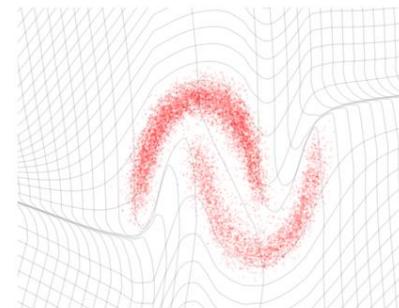
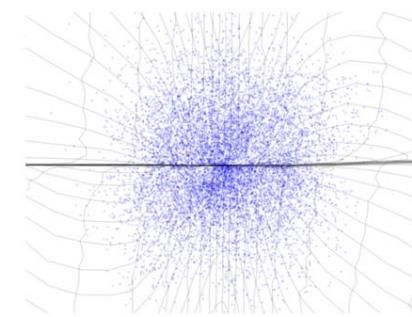
**Inference**  
 $x \sim \hat{p}_X$   
 $z = f(x)$

**Generation**  
 $z \sim p_Z$   
 $x = f^{-1}(z)$

Data space  $\mathcal{X}$



Latent space  $\mathcal{Z}$



# An Example of the Invertible Function

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

$$y_1 = x_1$$

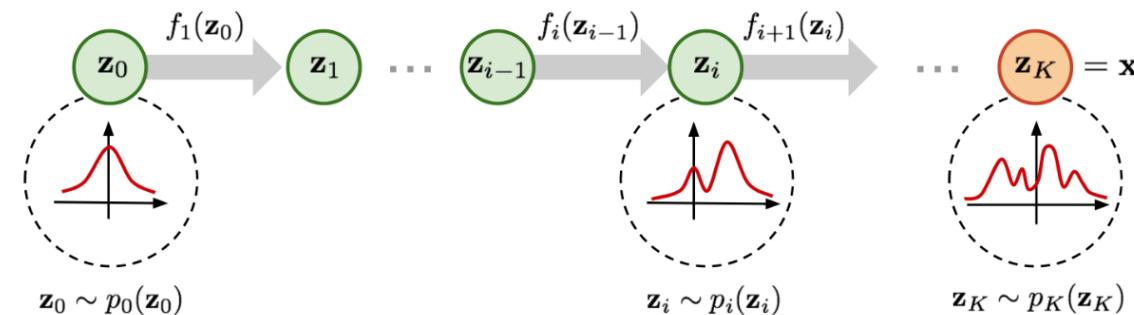
$$y_2 = x_2 + m(x_1)$$

◆ Then the determinant-Jacobian 1!

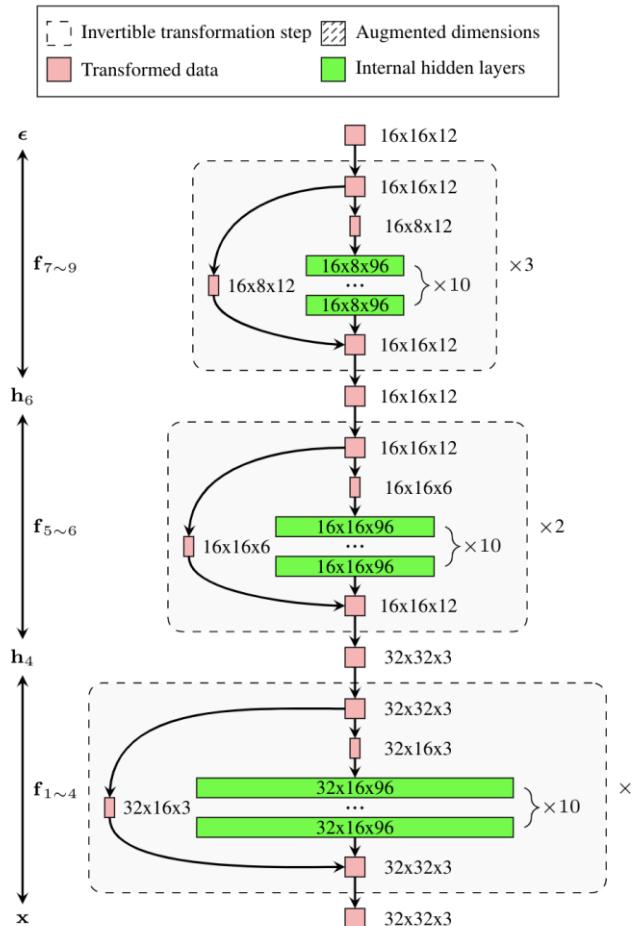
◆ The inverse is  $x_1 = y_1$

$$x_2 = y_2 - m(y_1)$$

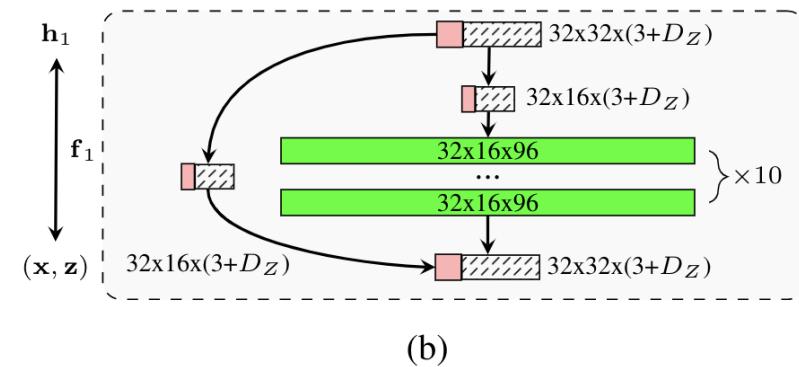
- $m$  is an arbitrarily complex function (e.g., ReLU networks)
- ◆ Define the invertible transform function as a composition of simple functions
  - Thus, the variable change is a sequence, called a **flow** (the sequence on density change is a **normalizing flow**)



# Bottleneck Problem in Flow++



Invertibility imposes constraints on architectures!

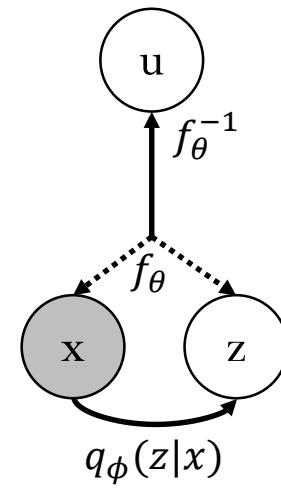


(b)

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  $f_1$  is shown due to space constraint.

# 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



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

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

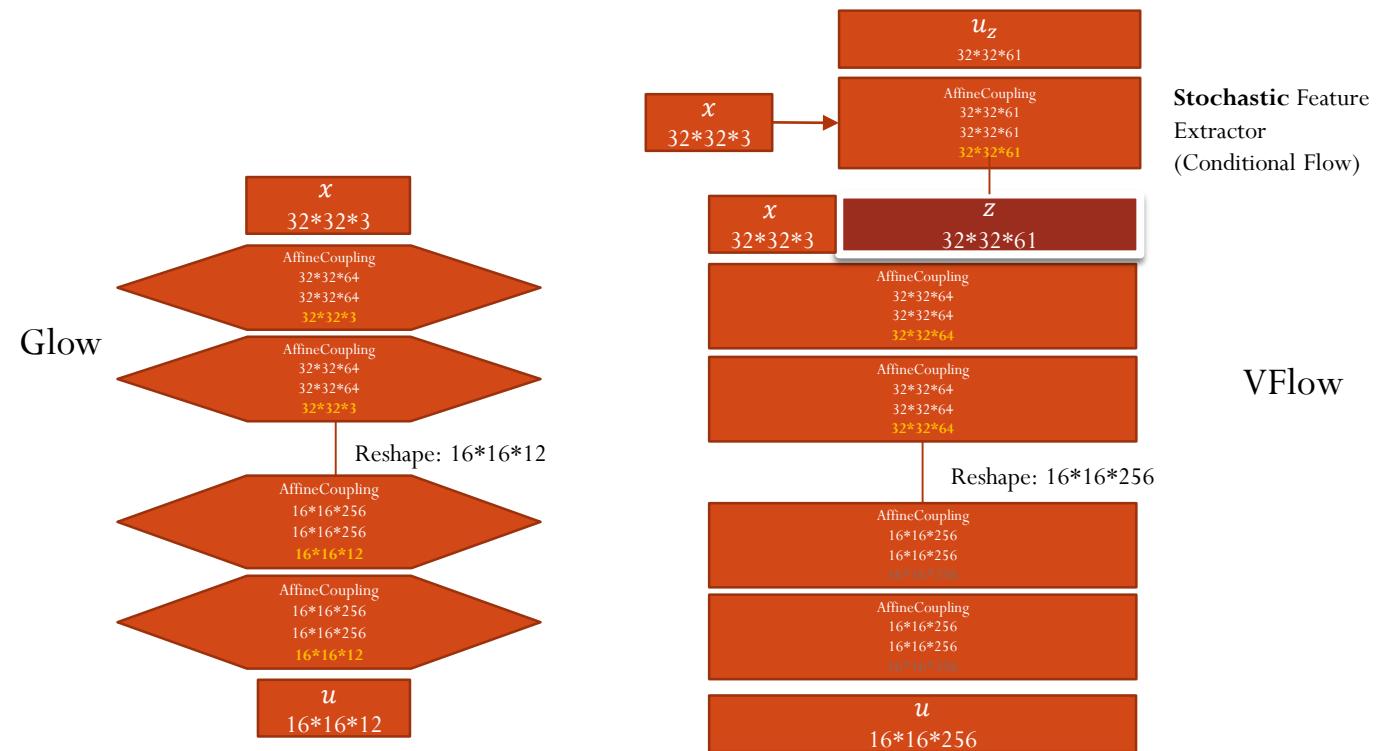
$$\log p(x, z) = \log p(u) + \log \det\left(\frac{du}{dy}\right)$$

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

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

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

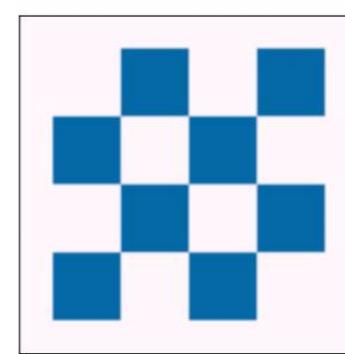
# VFlow



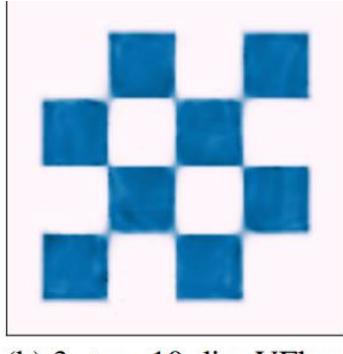
[Chen, Lu, Chenli, Zhu, Tian. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. ICML 2020]

# Results on Toy 2D Data

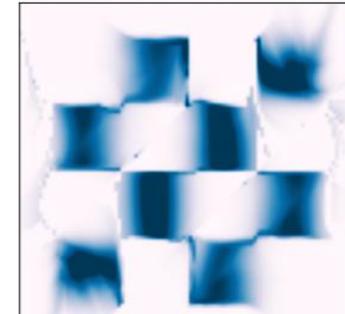
- ◆ VFlow significantly outperforms Glow under similar model size (e.g., 3-step)
- ◆ The 3-layer, 10-dimensional VFlow even outperforms a much larger 20-step Glow



(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

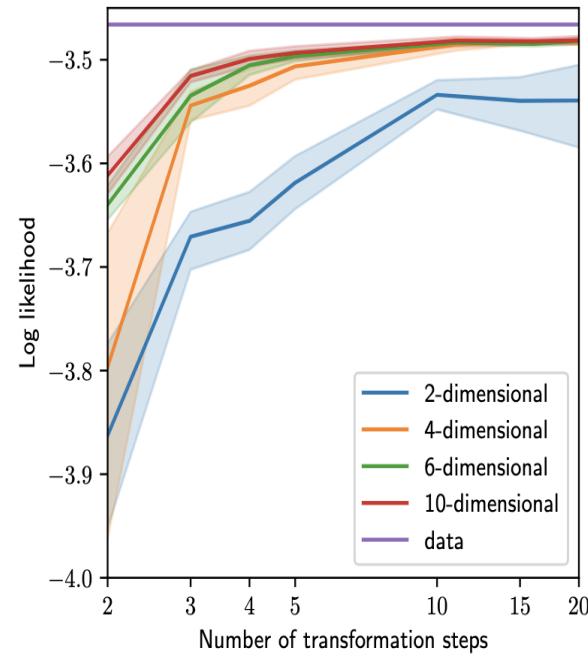


Figure 4. Impact of the dimensionality on the toy dataset.

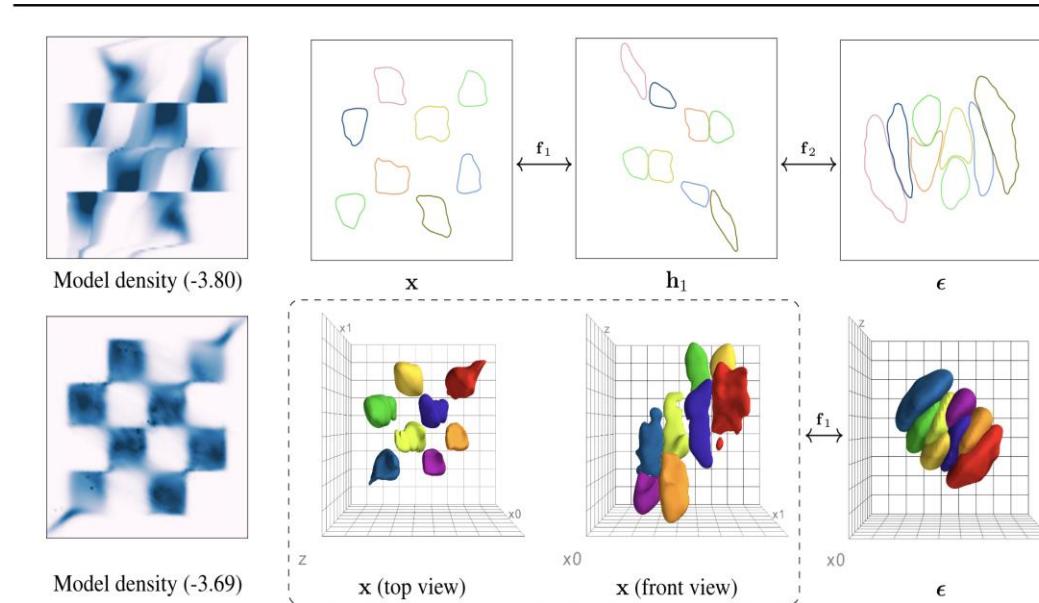


Figure 3. Visualization of learnt transformation on toy data. Top row: 2-step Glow. Bottom row: 2-step, 3-dimensional VFlow. Log-likelihood is shown in parenthesis. We sample  $\epsilon$  and visualize the transformed density in  $x$ ,  $h_1$  and  $\epsilon$  space. The density is estimated from samples by kernel density estimation, and we show the 50% probability contour / isosurface for each mode in different color.

# 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	<b>2.98</b>

Table 3. Parameter efficiency.

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

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	<b>3.12</b>	<b>4.01M</b>	32	10

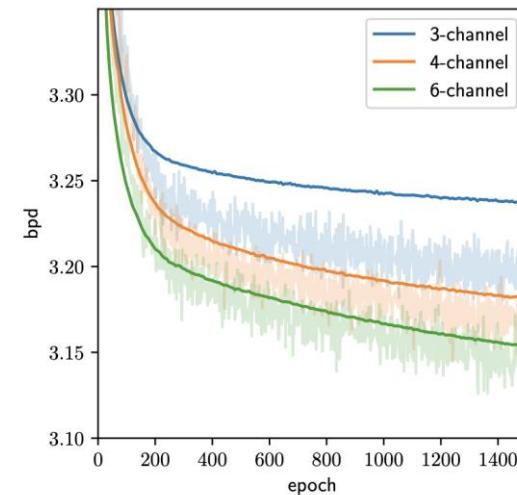


Figure 6. Bpd on training (light) and testing (dark) dataset of Flow++ and VFlow under a 4-million parameter budget. Here bpd is only a upper bound because we evaluate it with ELBO as Eq. (7) instead of the marginal likelihood.

# Requirements for Normalizing Flows

$$\log p_x(x) = \log p_z(f(x)) + \log \left| \det\left(\frac{df(x)}{dx}\right) \right|$$

1. An invertible function

2. Tractable log-determinant of the Jacobian

A crucial problem: Finding **rich** model families that have **tractable** log-determinants.

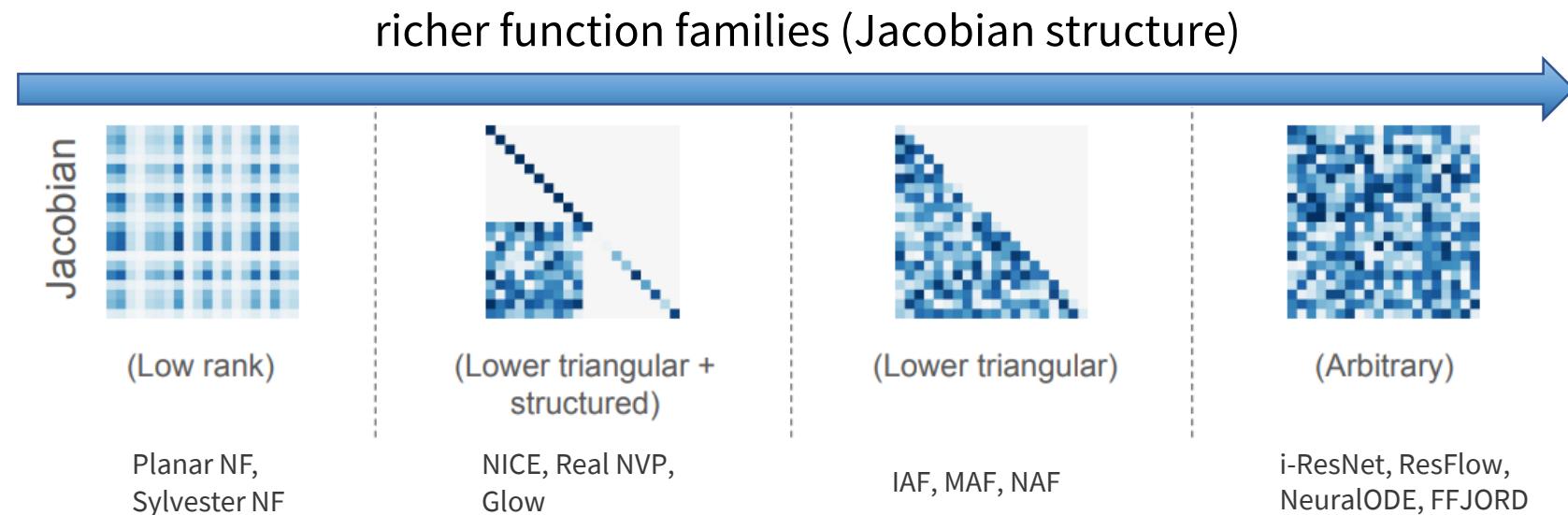


Figure from: [http://www.cs.toronto.edu/~rtqichen/pdfs/residual\\_flows\\_slides.pdf](http://www.cs.toronto.edu/~rtqichen/pdfs/residual_flows_slides.pdf)

# Lipschitz constraints of Jacobians

Free-form is not enough

Additive Coupling

$$\begin{aligned}\mathbf{y} &= f(\mathbf{x}) \\ \mathbf{y}_1 &= \mathbf{x}_1 \\ \mathbf{y}_2 &= \mathbf{x}_2 + t(\mathbf{x}_1)\end{aligned}$$



Structured  
Jacobian

$$\text{Lip}(f) \leq 1 + \text{Lip}(t)$$

Arbitrary  
Lipschitz

Residual Flows

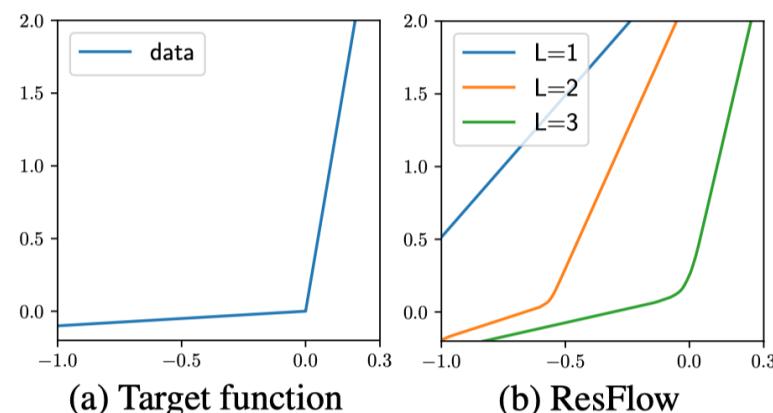
$$\mathbf{y} = f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$$



Free-form  
Jacobian

$$\text{Lip}(f) \leq 1 + \text{Lip}(g) < 2$$

Bounded  
Lipschitz



A 1-D function fitting example.

Due to the Lipschitz constraints of ResFlows, fitting a function with Lipschitz constant  $L$  needs at least  $\log_2 L$  layers.

# Implicit Function Theorem

Another way to define invertible mappings

Let  $F: \mathbb{R}^{2d} \rightarrow \mathbb{R}^d$  be a continuously differentiable function. Let  $\mathbf{z}$  and  $\mathbf{x}$  be two variables in  $\mathbb{R}^d$ . If  $\frac{\partial F(\mathbf{z}, \mathbf{x})}{\partial \mathbf{z}}$  and  $\frac{\partial F(\mathbf{z}, \mathbf{x})}{\partial \mathbf{x}}$  are invertible matrices for any  $\mathbf{z}, \mathbf{x} \in \mathbb{R}^d$ , then

$F(\mathbf{z}, \mathbf{x}) = 0$  defines a unique and invertible mapping

$f: \mathbb{R}^d \rightarrow \mathbb{R}^d$  such that  $\mathbf{z} = f(\mathbf{x})$

A special case: for any previous normalizing flow model  $f(\mathbf{x}) = \mathbf{z}$ , define

$$F(\mathbf{z}, \mathbf{x}) = f(\mathbf{x}) - \mathbf{z}$$

Explicit invertible functions are special cases of implicit functions

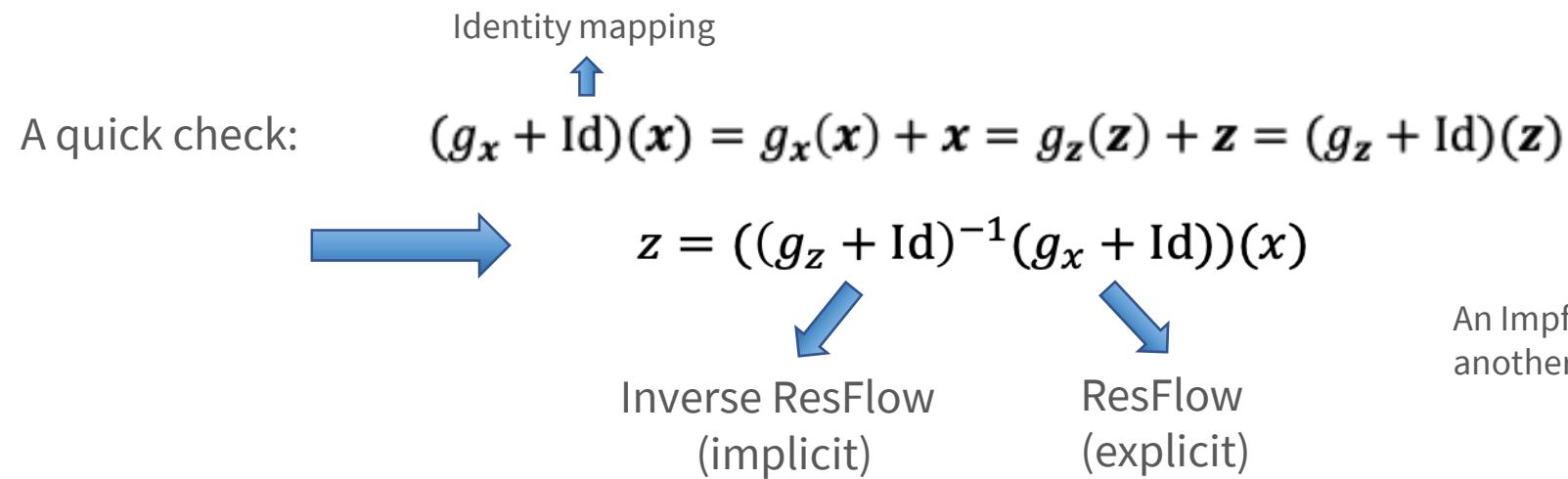
# Implicit Normalizing Flow (ImpFlow)

Defined by a specific form of implicit functions

Let  $g_z: \mathbb{R}^d \rightarrow \mathbb{R}^d$ ,  $\text{Lip}(g_z) < 1$ ,  $g_x: \mathbb{R}^d \rightarrow \mathbb{R}^d$ ,  $\text{Lip}(g_x) < 1$ , define

$$F(\mathbf{z}, \mathbf{x}) = g_x(\mathbf{x}) - g_z(\mathbf{z}) + \mathbf{x} - \mathbf{z}$$

Then  $F(\mathbf{z}, \mathbf{x}) = 0$  defines a **unique** and **invertible** mapping  $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$  such that  $\mathbf{z} = f(\mathbf{x})$ .

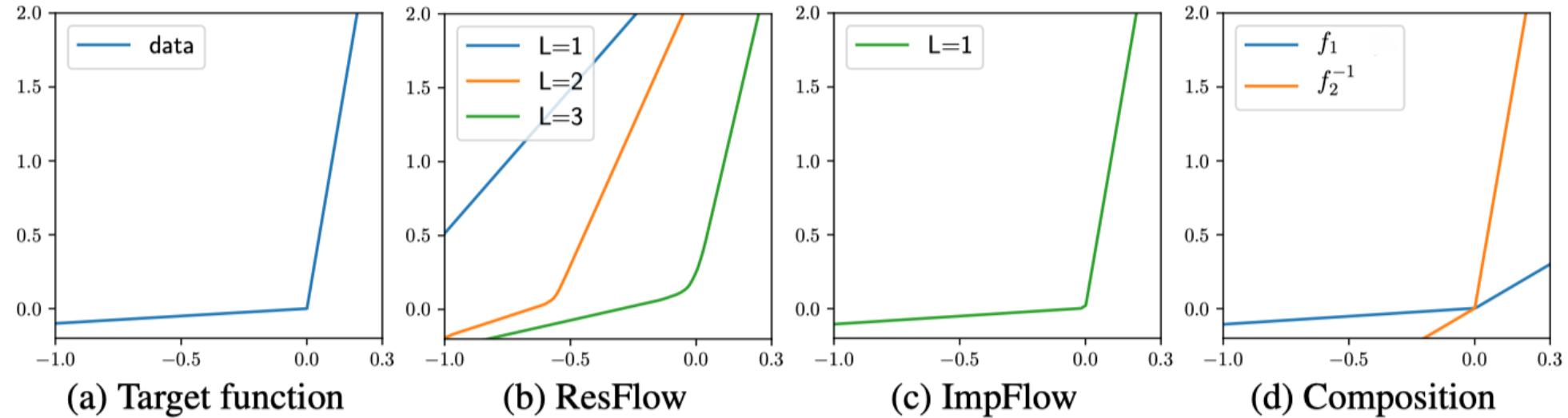


An Impflow is a composition of a forward ResFlow and another inverse ResFlow.

# Relaxing the Lipschitz Constraints

Also have free-form Jacobians

1-D example



Intuitively, the forward ResFlow handles small Lipschitz parts, and the inverse ResFlow handles large Lipschitz parts.

$$z = ((g_z + Id)^{-1}(g_x + Id))(x)$$
$$\downarrow \qquad \downarrow$$
$$f_2^{-1} \qquad f_1$$

# Main Theoretical Results

## Function family of ImpFlows

Let  $\mathcal{D}$  be the set of all bi-Lipschitz  $C^1$  invertible functions from  $\mathbb{R}^d$  to  $\mathbb{R}^d$ . Define

$$\mathcal{F} := \{f \in \mathcal{D} : \inf_{\mathbf{x} \in \mathbb{R}^d, \mathbf{v} \in \mathbb{R}^d, \|\mathbf{v}\|_2=1} \mathbf{v}^T J_f(\mathbf{x}) \mathbf{v} > 0\} \quad \text{"monotonically increasing functions"}$$

- 1-D case:  $\mathcal{F} = \{f \in C^1(\mathbb{R}) : \inf_{x \in \mathbb{R}} f'(x) > 0\}$ .
- Symmetric case: gradients of strongly convex functions. Includes "Convex Potential Flows" (ICLR 2021) [1]

**Theorem**  $\mathcal{I} = \mathcal{F}_2 := \{f : f = f_2 \circ f_1 \text{ for some } f_1, f_2 \in \mathcal{F}\}.$

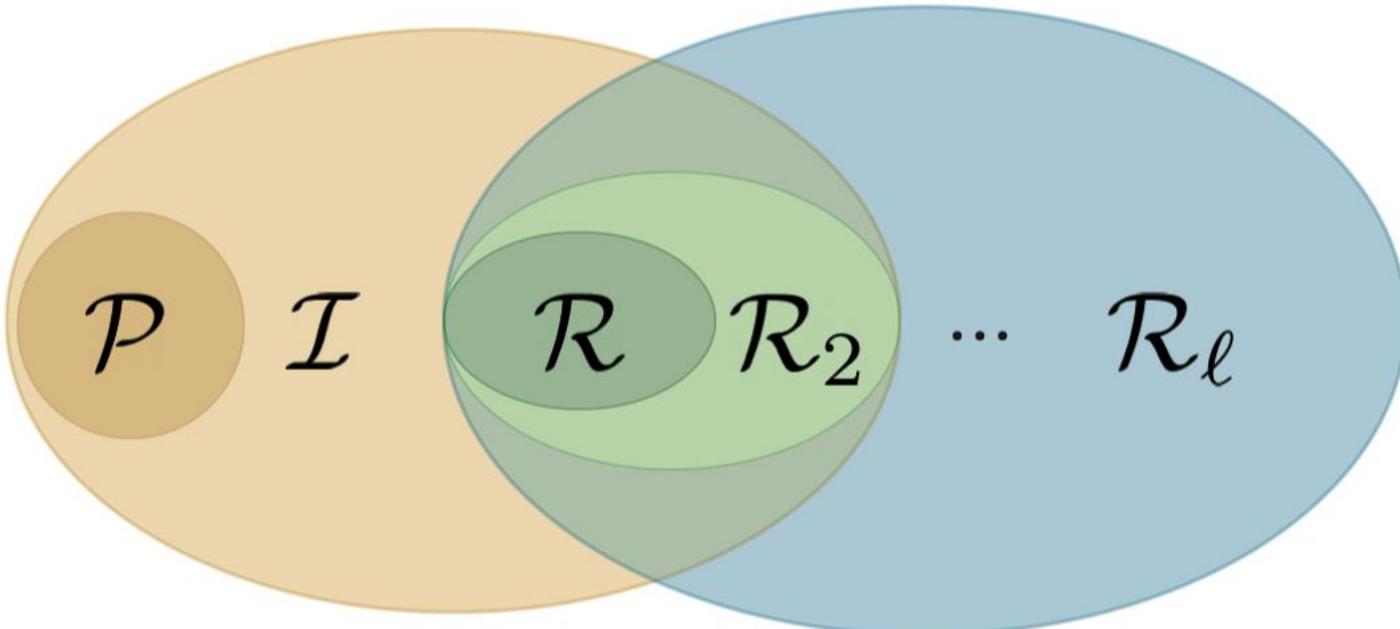


Function family of single-layer ImpFlows

[1] Huang, Chin-Wei, et al. "Convex Potential Flows: Universal Probability Distributions with Optimal Transport and Convex Optimization." *arXiv preprint arXiv:2012.05942* (2020).

# Main Theoretical Results

## Relationship between 1-layer ImpFlow and $\ell$ -layer ResFlow



1.  $\mathcal{R} \subsetneq \mathcal{R}_2 \subsetneq \mathcal{I}$ .

2-layer ResFlows are strictly included in 1-layer ImpFlows.

2.  $\forall \ell > 0, \mathcal{I} \not\subseteq \mathcal{R}_\ell$ .

Any multi-layer ResFlows cannot include 1-layer ImpFlows.

# Generative Modeling: Inference, Sampling, Learning

Remains tractability

- Forward & Inverse: finding fixed point.

$$\mathbf{z} = ((g_{\mathbf{z}} + Id)^{-1}(g_{\mathbf{x}} + Id))(\mathbf{x})$$

1. Simple fixed point iterations.
1. Quasi-Newton methods (i.e. Broyden's method).

- Log-determinant of Jacobians: the same unbiased estimator as ResFlows.

$$\ln p(\mathbf{x}) = \ln p(\mathbf{z}) + \ln \det(I + J_{g_{\mathbf{x}}}(\mathbf{x})) - \ln \det(I + J_{g_{\mathbf{z}}}(\mathbf{z}))$$

- Backpropagation for the fixed point: take total derivative of  $F(\mathbf{z}, \mathbf{x}) = 0$ .

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial (\cdot)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}} J_G^{-1}(\mathbf{z}) \frac{\partial F(\mathbf{z}, \mathbf{x}; \theta)}{\partial (\cdot)}, \text{ where } G(\mathbf{z}; \theta) = g_{\mathbf{z}}(\mathbf{z}; \theta) + \mathbf{z}.$$

# Model Expressiveness

## Classification tasks

Table 1: Classification error rate (%) on test set of vanilla ResNet, ResFlow and ImpFlow of ResNet-18 architecture, with varying Lipschitz coefficients  $c$ .

		Vanilla	$c = 0.99$	$c = 0.9$	$c = 0.8$	$c = 0.7$	$c = 0.6$
CIFAR10	ResFlow		8.24 ( $\pm 0.03$ )	8.39 ( $\pm 0.01$ )	8.69 ( $\pm 0.03$ )	9.25 ( $\pm 0.02$ )	9.94 ( $\pm 0.02$ )
	ResFlow	6.61( $\pm 0.02$ )	<b>7.29</b> ( $\pm 0.03$ )	<b>7.41</b> ( $\pm 0.03$ )	<b>7.94</b> ( $\pm 0.06$ )	<b>8.44</b> ( $\pm 0.04$ )	<b>9.22</b> ( $\pm 0.02$ )
	ImpFlow						
	ImpFlow						
CIFAR100	ResFlow		31.02 ( $\pm 0.05$ )	31.88 ( $\pm 0.02$ )	32.21 ( $\pm 0.03$ )	33.58 ( $\pm 0.02$ )	34.48 ( $\pm 0.03$ )
	ResFlow	27.83( $\pm 0.03$ )	<b>29.06</b> ( $\pm 0.03$ )	<b>30.47</b> ( $\pm 0.03$ )	<b>31.40</b> ( $\pm 0.03$ )	<b>32.64</b> ( $\pm 0.01$ )	<b>34.17</b> ( $\pm 0.02$ )
	ImpFlow						
	ImpFlow						

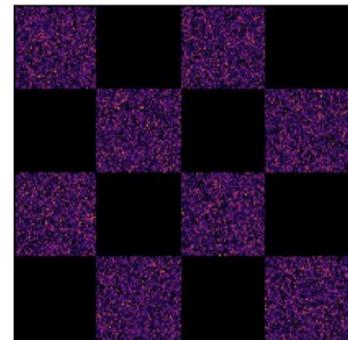
# Density Estimation Experiments

Table 2: Average test log-likelihood (in nats) of tabular datasets. Higher is better.

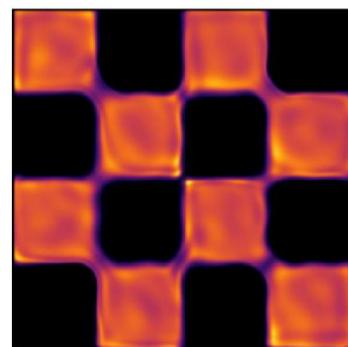
	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
RealNVP (Dinh et al., 2017)	0.17	8.33	-18.71	-13.55	153.28
FFJORD (Grathwohl et al., 2019)	0.46	8.59	-14.92	-10.43	157.40
MAF (Papamakarios et al., 2017)	0.24	10.08	-17.70	-11.75	155.69
NAF (Huang et al., 2018)	<b>0.62</b>	11.96	-15.09	<b>-8.86</b>	<b>157.73</b>
ImpFlow ( $L = 20$ )	0.61	<b>12.11</b>	<b>-13.95</b>	-13.32	155.68
ResFlow ( $L = 10$ )	0.26	6.20	-18.91	-21.81	104.63
ImpFlow ( $L = 5$ )	<b>0.30</b>	<b>6.94</b>	<b>-18.52</b>	<b>-21.50</b>	<b>113.72</b>

Table 3: Average bits per dimension of ResFlow and ImpFlow on CIFAR10, with varying Lipschitz coefficients  $c$ . Lower is better.

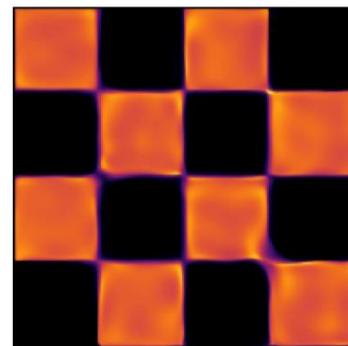
	$c = 0.9$	$c = 0.8$	$c = 0.7$	$c = 0.6$
ResFlow ( $L = 12$ )	3.469( $\pm 0.0004$ )	3.533( $\pm 0.0002$ )	3.627( $\pm 0.0004$ )	3.820( $\pm 0.0003$ )
ImpFlow ( $L = 6$ )	<b>3.452</b> ( $\pm 0.0003$ )	<b>3.511</b> ( $\pm 0.0002$ )	<b>3.607</b> ( $\pm 0.0003$ )	<b>3.814</b> ( $\pm 0.0005$ )



(a) Checkerboard data (5.00 bits)



(b) ResFlow,  
 $L = 8$  (5.08 bits)



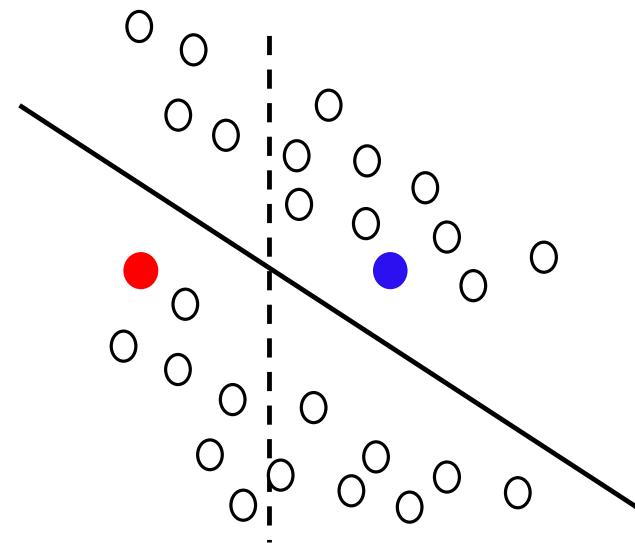
(c) ImpFlow,  
 $L = 4$  (5.05 bits)

# **Semi-supervised Learning with GANs**

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

# Semi-supervised Learning

A toy example



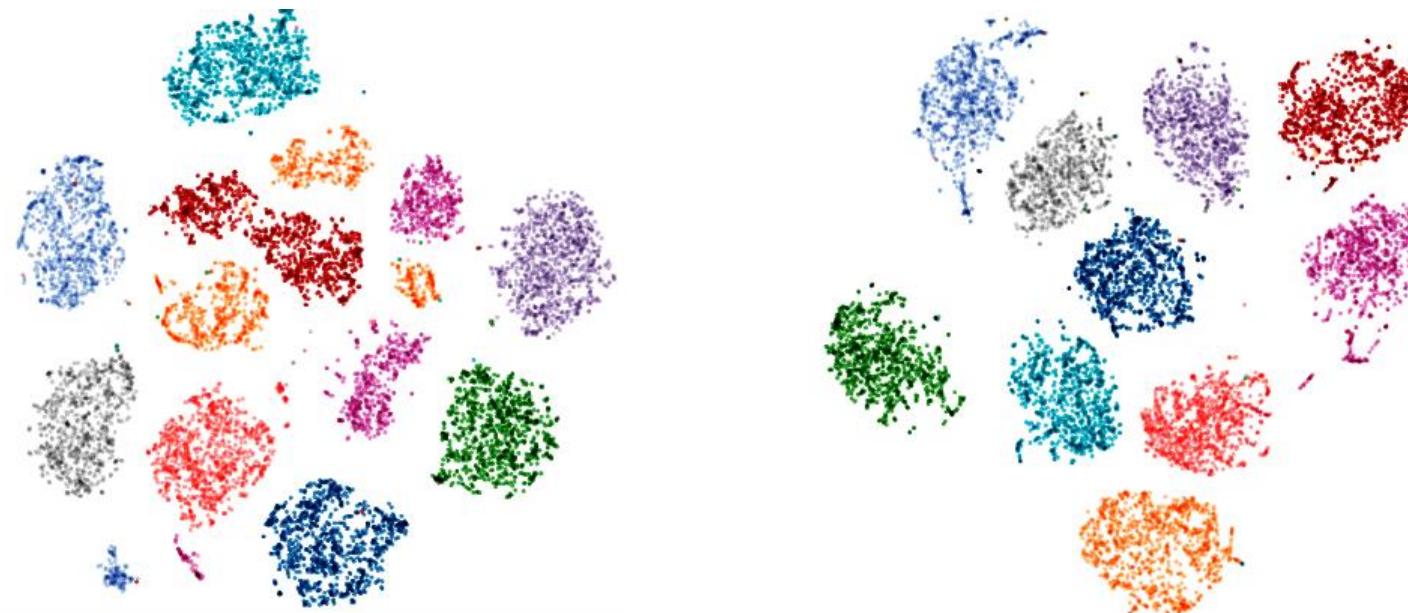
# Representation Matters

- ◆ 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 GANs for SSL

- ◆ A minimax game for semi-supervised learning (SSL)

- GAN is for unsupervised learning
  - We aim to learn the joint distribution

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

$$p_{\text{model}}(x, y) = p_{\text{data}}(x, y)$$

- ◆ We need three players

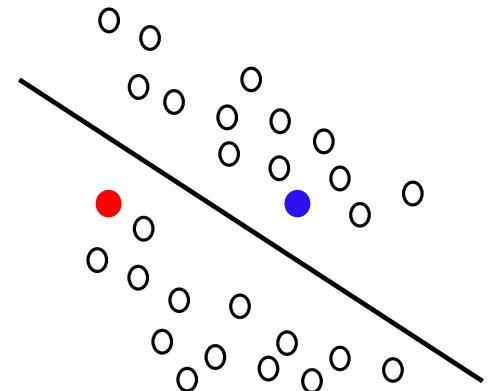
- factorization form with conditionals

$$\begin{aligned} p(x, y) &= p(x)p(y|x) \\ &= p(y)p(x|y) \end{aligned}$$

A classifier

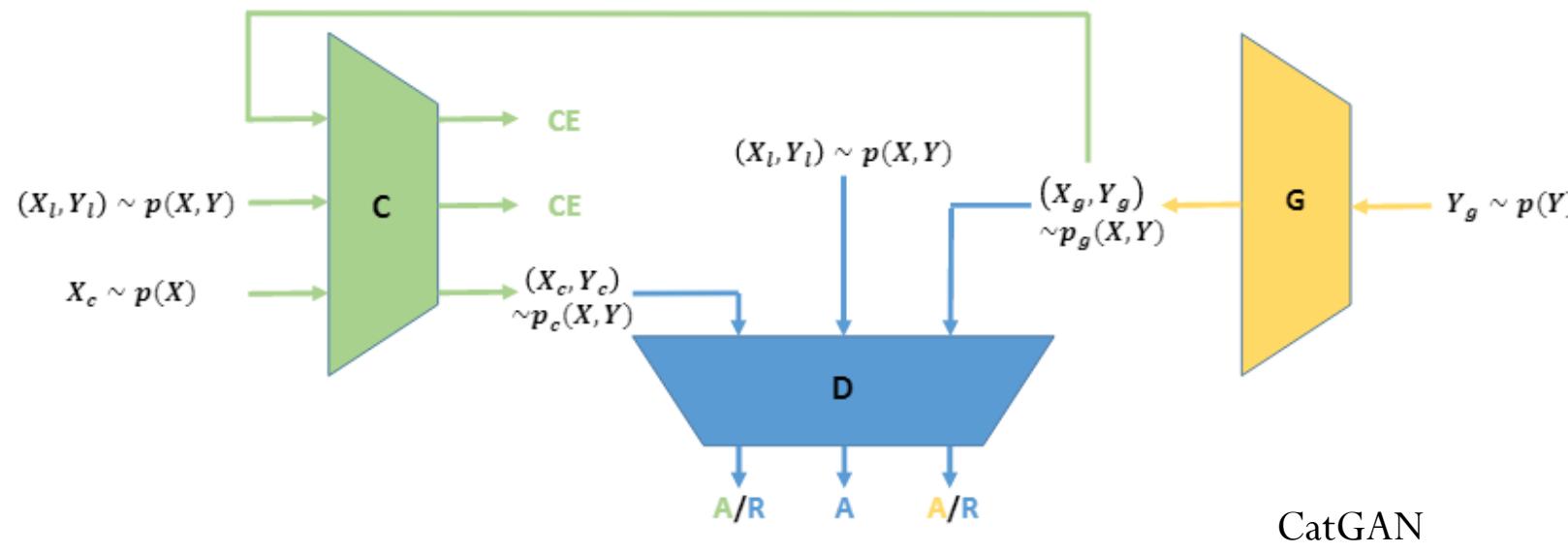
A class-conditional generator

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

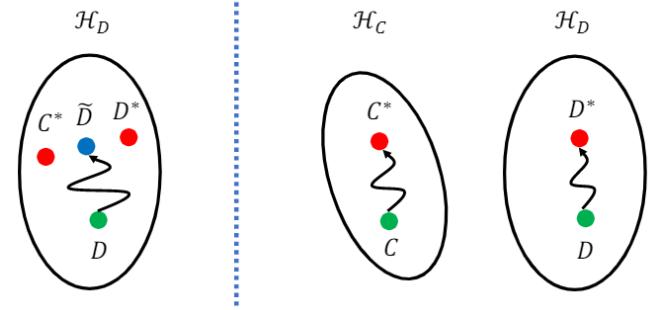


# Triple GANs for SSL

- ◆ The network architecture (de-couple the generators and discriminator)



- ❑ 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  $\alpha$  is often set at 1/2
- The standard supervised loss can be incorporated

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

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

# SSL Results without Data Augmentation

Dataset Number of labels	SVHN 250	SVHN 500	SVHN 1,000	CIFAR10 1,000	CIFAR10 2,000	CIFAR10 4,000	Tiny ImageNet 1,000
<i>M1+M2</i> [29]			36.02 ( $\pm 0.10$ )				
<i>Ladder</i> [49]						20.40 ( $\pm 0.47$ )	
<i>SDGM</i> [60] <sup>§</sup>			16.61 ( $\pm 0.24$ )				
<i>MMCVA</i> [48] <sup>§</sup>			4.95 ( $\pm 0.18$ )				
<i>CatGAN</i> [24]						19.58 ( $\pm 0.58$ )	
<i>ALI</i> [8] <sup>†</sup>			7.3			18.3	
<b>-Using classifier in [26]-</b>							
<i>Improved-GAN</i> [26] <sup>†</sup>			8.11 ( $\pm 1.3$ )			18.63 ( $\pm 2.32$ )	28.47 ( $\pm 0.41$ )
<i>Triple-GAN-V1 (ours)</i> <sup>†</sup>			5.77 ( $\pm 0.17$ )			16.99 ( $\pm 0.36$ )	24.83 ( $\pm 1.23$ )
<i>Triangle-GAN</i> [39] <sup>†</sup>						16.80 ( $\pm 0.42$ )	
<i>BadGAN</i> [27] <sup>†</sup>			4.25 ( $\pm 0.03$ )			14.41 ( $\pm 0.30$ )	
<i>Π model</i> [38] <sup>†</sup>	10.36 ( $\pm 0.94$ )	7.01 ( $\pm 0.29$ )	5.73 ( $\pm 0.16$ )	32.18 ( $\pm 1.33$ )	23.92 ( $\pm 1.07$ )	17.08 ( $\pm 0.32$ )	
<i>Pseudo Label</i> [41] <sup>†</sup>			7.62 ( $\pm 0.29$ )			17.78 ( $\pm 0.57$ )	
<i>SNTG</i> [40] <sup>†</sup>			4.02 ( $\pm 0.20$ )			12.49 ( $\pm 0.36$ )	
<i>VAT+Ent</i> [43] <sup>†</sup>			4.28 ( $\pm 0.10$ )			13.15 ( $\pm 0.21$ )	
<i>MT</i> [32] <sup>†</sup>	5.85 ( $\pm 0.62$ )	5.45 ( $\pm 0.14$ )	5.21 ( $\pm 0.21$ )	30.62 ( $\pm 1.13$ )	23.14 ( $\pm 0.46$ )	17.74 ( $\pm 0.30$ )	
<b>-Using classifier in [32]-</b>							
<i>MT</i> (our code base) <sup>†</sup>	5.82 ( $\pm 0.87$ )	4.48 ( $\pm 0.17$ )	5.39 ( $\pm 0.12$ )	28.34 ( $\pm 0.36$ )	21.24 ( $\pm 0.79$ )	16.37 ( $\pm 0.21$ )	20.67 ( $\pm 1.36$ )
<i>Triple-GAN-V2 (ours)</i> <sup>†</sup>	4.19 ( $\pm 0.62$ )	3.84 ( $\pm 0.17$ )	3.96 ( $\pm 0.15$ )	18.19 ( $\pm 0.35$ )	14.74 ( $\pm 0.89$ )	12.41 ( $\pm 0.41$ )	17.40 ( $\pm 1.06$ )

# SSL Results with Data Augmentation

- ◆ With standard data augmentation using a similar 13-layer CNN classifier
  - Triple-GAN-V2 can benefit from data augmentation as well, ...
  - also outperforms the baselines

Dataset	SVHN 250	SVHN 500	SVHN 1,000	CIFAR10 1,000	CIFAR10 2,000	CIFAR10 4,000	Tiny ImageNet 1,000
<i>Π model</i> [38]	9.69 ( $\pm 0.92$ )	6.83 ( $\pm 0.66$ )	4.95 ( $\pm 0.26$ )	27.36 ( $\pm 1.20$ )	18.02 ( $\pm 0.60$ )	13.20 ( $\pm 0.27$ )	
VAT+Ent [43]			3.86 ( $\pm 0.11$ )				10.55 ( $\pm 0.05$ )
SWA-480 [45]				17.48 ( $\pm 0.13$ )	13.09 ( $\pm 0.80$ )	10.30 ( $\pm 0.21$ )	
Fast-SWA-480 [45]				16.84 ( $\pm 0.62$ )	12.24 ( $\pm 0.31$ )	<b>9.86</b> ( $\pm 0.27$ )	
SNTG [40]	4.29 ( $\pm 0.23$ )	3.99 ( $\pm 0.24$ )	3.86 ( $\pm 0.27$ )	18.41 ( $\pm 0.52$ )	13.64 ( $\pm 0.32$ )	10.93 ( $\pm 0.14$ )	
<i>Label Propagation</i> [44]				16.93 ( $\pm 0.70$ )	13.22 ( $\pm 0.29$ )	10.61 ( $\pm 0.28$ )	
MT [32]	4.35 ( $\pm 0.50$ )	4.18 ( $\pm 0.27$ )	3.95 ( $\pm 0.19$ )	21.55 ( $\pm 1.48$ )	15.73 ( $\pm 0.31$ )	12.31 ( $\pm 0.28$ )	
<b>-Using classifier in [32]-</b>							
MT (our code base)	4.89 ( $\pm 0.28$ )	4.36 ( $\pm 0.22$ )	4.48 ( $\pm 0.17$ )	25.14 ( $\pm 1.32$ )	18.05 ( $\pm 0.45$ )	13.18 ( $\pm 0.21$ )	19.07 ( $\pm 0.42$ )
<b>Triple-GAN-V2 (ours)</b>	<b>3.48</b> ( $\pm 0.06$ )	<b>3.61</b> ( $\pm 0.26$ )	<b>3.45</b> ( $\pm 0.11$ )	<b>15.00</b> ( $\pm 0.57$ )	<b>11.87</b> ( $\pm 0.20$ )	<b>10.01</b> ( $\pm 0.16$ )	<b>17.13</b> ( $\pm 1.40$ )

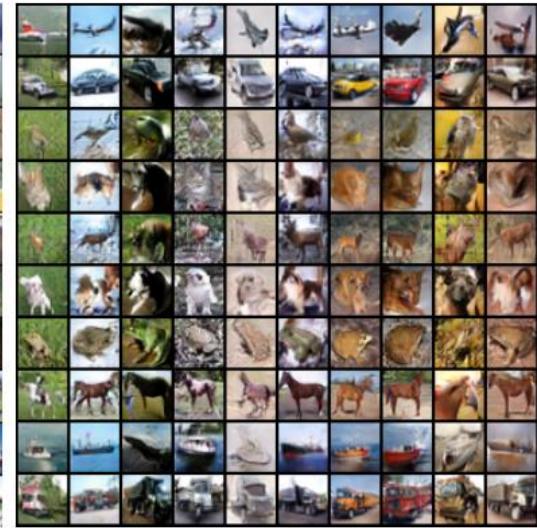
# Results on (Conditional) Generation

## ◆ IS and FID scores

Metrics	IS ↑	FID ↓
<b>-Using GAN in [26]-</b>		
Improved-GAN-FM [26] <sup>†</sup>	3.87	
Triple-GAN-V1 [61] ( <b>ours</b> ) <sup>†</sup>	5.08 ( $\pm 0.09$ )	
<b>-Using ResNet GAN-</b>		
DCGAN [2] <sup>§</sup>	6.16 ( $\pm 0.07$ ) [50]	36.9 [46]
DCGAN [2] <sup>‡</sup>	6.58 [50]	
WGAN-GP [50] <sup>§</sup>	7.86 ( $\pm 0.07$ )	24.8 [46]
WGAN-GP [50] <sup>‡</sup>	8.42 ( $\pm 0.10$ )	
ACGAN [31] <sup>‡</sup>	8.22 [34]	19.7 [34]
<b>-Using GAN in [34]-</b>		
SNGAN [33] <sup>§</sup>	8.22 ( $\pm 0.05$ )	21.7 ( $\pm 0.21$ )
CGAN-PD [34] <sup>‡</sup>	<b>8.62</b>	<b>17.5</b>
CGAN-PD (our code base) <sup>††</sup>	6.03	42.3
Triple-GAN-V2 ( <b>ours</b> ) <sup>†</sup>	8.51	17.9



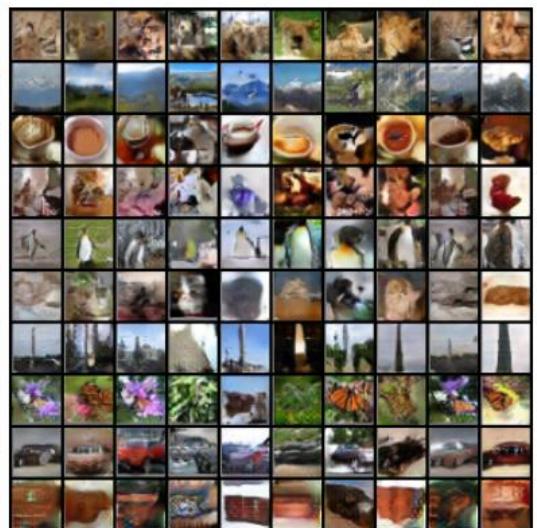
(c) CIFAR10 data



(d) CIFAR10 samples



(e) Tiny Imagenet data



(f) Tiny Imagenet samples

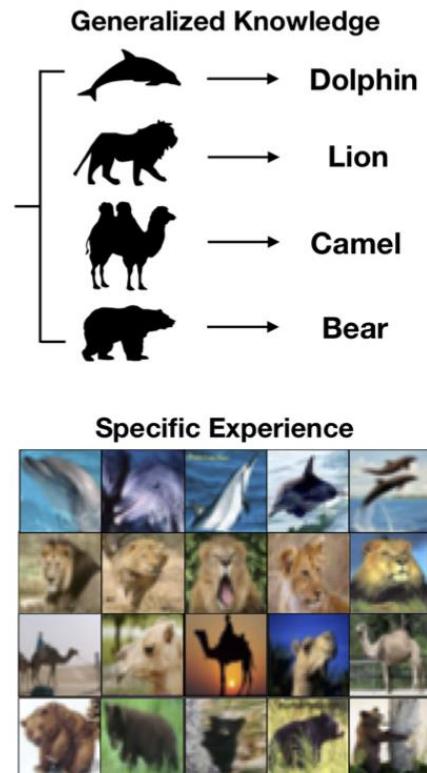
# **Triple Memory Networks (TMNs): a Brain-Inspired Method for Continual Learning**

with Liyuan Wang, Bo Lei, Qian Li, Hang Su and Yi Zhong, TNNLS 2021

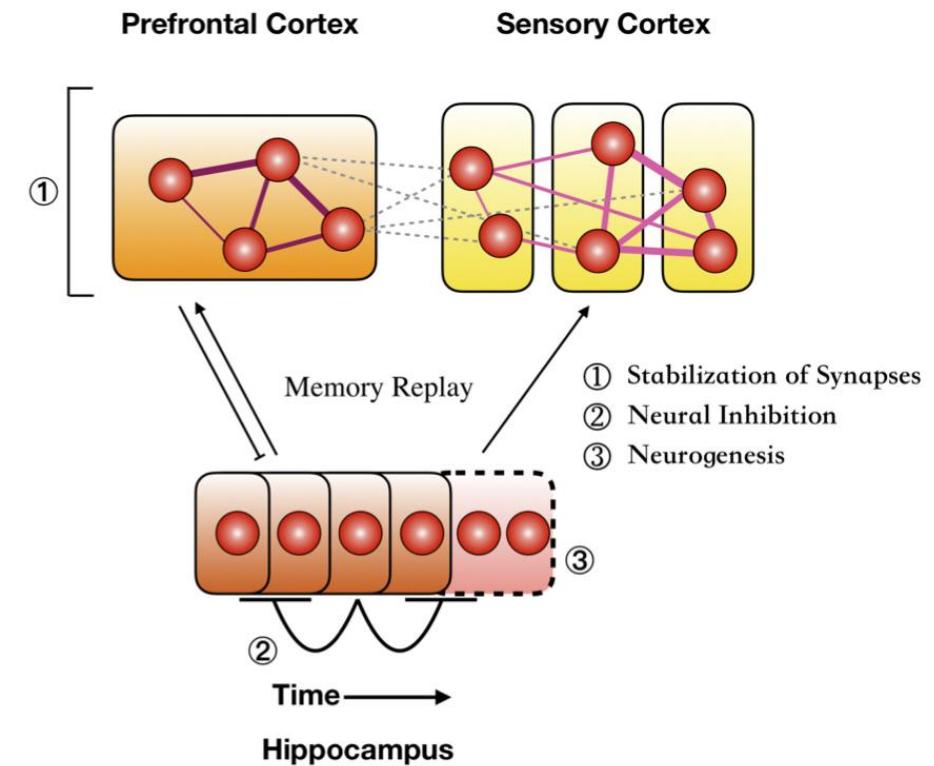
# Motivation

- ◆ The biological brain can successfully avoid catastrophic forgetting by distinctively consolidating memory in different regions and complementing with each other.

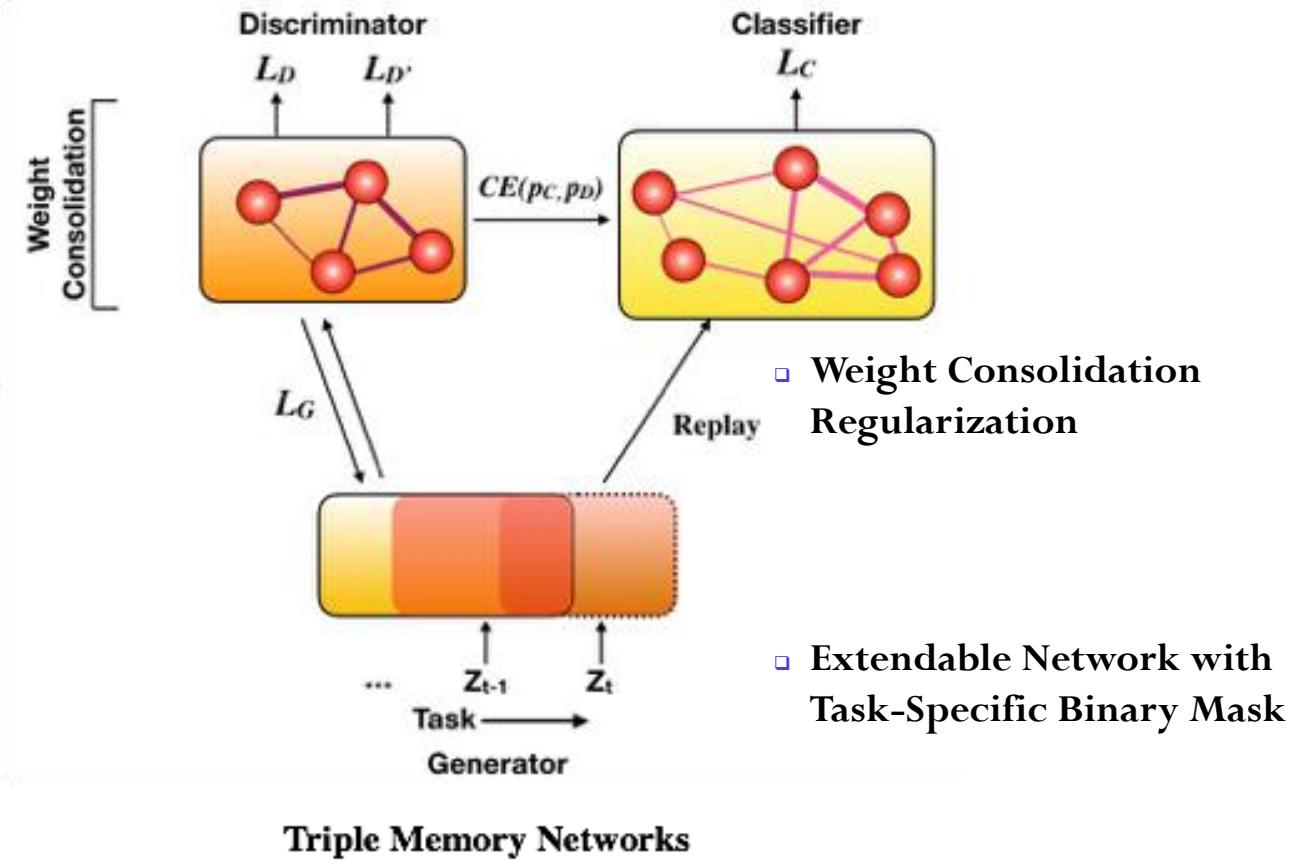
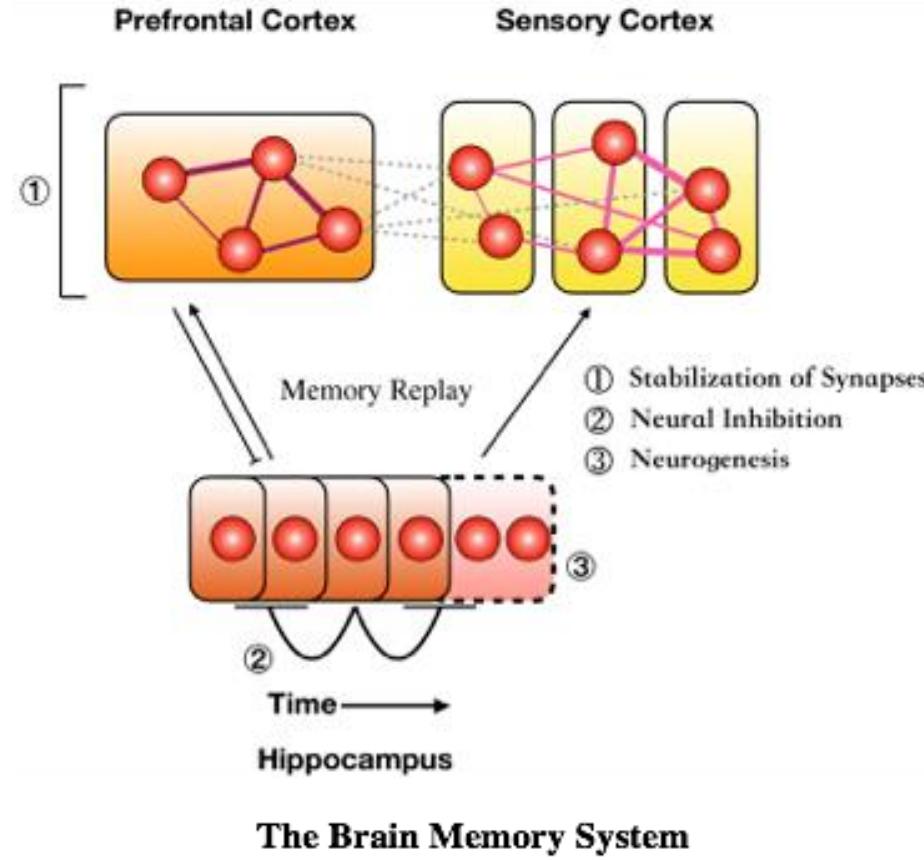
- Neocortex: consolidates generalized knowledge by stabilization of synapses.



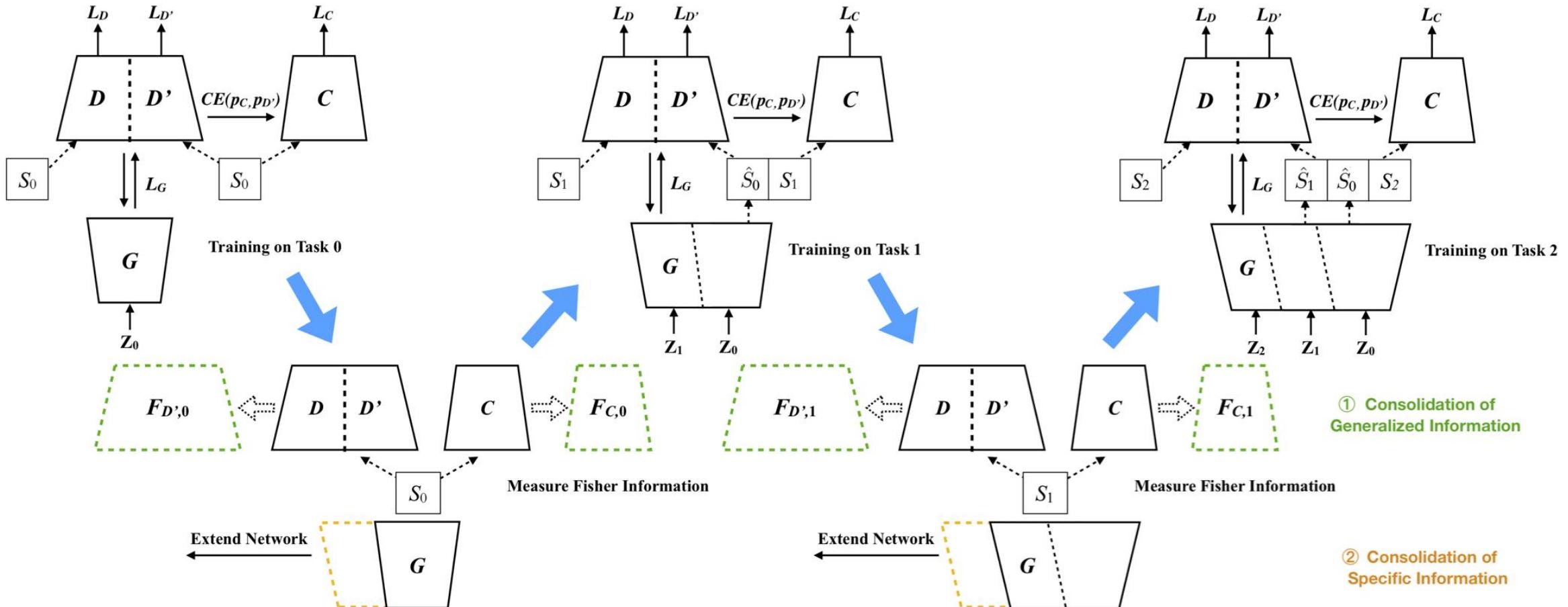
- Hippocampus: consolidates specific experiences by neural genesis and neural inhibition.



# Model the Brain Memory System



# Triple Memory Networks



# Experimental Results

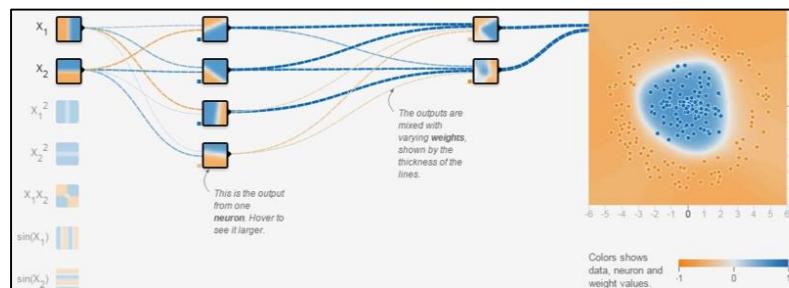
- Without accessing to the old data, Triple Memory Networks (TMNs) achieve the state-of-the-art performance on a variety of continual learning

		MNIST		SVHN		CIFAR-10		ImageNet-50	
Methods		$A_5$	$A_{10}$	$A_5$	$A_{10}$	$A_5$	$A_{10}$	$A_{30}$	$A_{50}$
+ Training Data	Joint Training	99.87	99.24	92.99	88.72	83.40	77.82	57.35	49.88
	EWC-S [13]	79.36	60.83	38.65	25.36	37.39	21.13	-	-
	SI-S [14]	78.40	60.18	37.21	23.86	36.96	20.16	-	-
	RWalk-S [25]	82.08	62.84	39.25	26.63	35.75	22.27	-	-
	MAS-S [15]	80.40	67.66	37.57	25.11	44.38	19.56	-	-
	iCarl [18]	-	-	-	-	57.30	43.69	29.38	<b>28.98</b>
- Training Data	DGMw-S [22]	-	-	-	-	-	-	36.87	18.84
	EWC-M [41]	70.62	77.03	39.84	33.02	-	-	-	-
	DGR [3]	90.39	85.40	61.29	47.28	-	-	-	-
	MeRGAN [21]	98.19	<b>97.00</b>	80.90	66.78	-	-	-	-
	DGMw [22]	98.75	96.46	83.93	74.38	72.45	56.21	32.14	17.82
TMNs (ours)		<b>98.80</b>	96.72	<b>87.12</b>	<b>77.08</b>	<b>72.72</b>	<b>61.24</b>	<b>38.23</b>	28.08

# **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  $y$

Docs » Welcome to ZhuSuan Edit on GitHub

## Welcome to ZhuSuan

ZhuSuan is a python probabilistic programming library for Bayesian deep learning, which conjoints the complimentary advantages of Bayesian methods and deep learning. ZhuSuan is built upon [Tensorflow](#). Unlike existing deep learning libraries, which are mainly designed for deterministic neural networks and supervised tasks, ZhuSuan provides deep learning style primitives and algorithms for building probabilistic models and applying Bayesian inference. The supported inference algorithms include:

- Variational inference with programmable variational posteriors, various objectives and advanced gradient estimators (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.

### Installation

ZhuSuan is still under development. Before the first stable release (1.0), please clone the [GitHub repository](#) and run

```
pip install .
```

[github.com/thu-m1/zhusuan](https://github.com/thu-m1/zhusuan)

# “ZhuSuan” Deep Probabilistic Programming



ZHUSUAN

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



ZHUSUAN

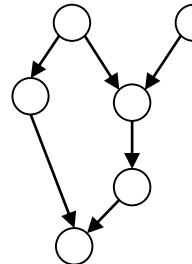
New model reuse strategy

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

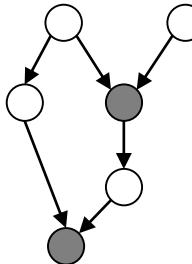
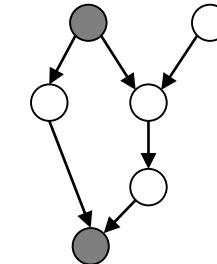
```
@zs.meta_bayesian_net(scope="model", reuse_variables=True)
```

```
def build_model(...):
```

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

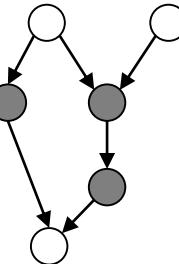


`zs.MetaBayesianNet`



`zs.BayesianNet`

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



[github.com/thu-ml/zhusuan](https://github.com/thu-ml/zhusuan)

# “ZhuSuan” Deep Probabilistic Programming



ZHUSUAN

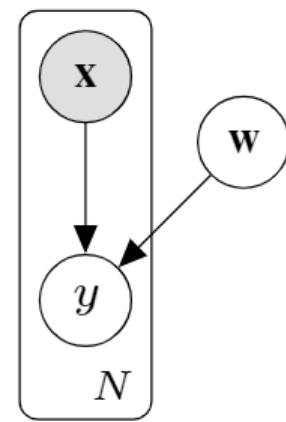
New Algorithms: Stochastic gradient MCMC

变分推理	梯度估计器	实现
<code>elbo()</code>	SGVB Reinforce	<code>.sgvb()</code> <code>.reinforce()</code>
<code>k1pq()</code>	Importance Sampling	<code>.importance()</code>
<code>iw_objective()</code>	SGVB VIMCO	<code>.sgvb()</code> <code>.vimco()</code>
马尔科夫链蒙特卡洛	实现	
	Hamiltonian Monte Carlo	<code>HMC()</code>
Stochastic Gradient MCMC	(Preconditioned) SGLD SGHMC SGNHT	<code>SGLD()</code> , <code>PSGLD()</code> <code>SGHMC()</code> <code>SGNHT()</code>
		<a href="https://github.com/thu-ml/zhusuan">github.com/thu-ml/zhusuan</a>

# Examples: Bayesian Logistic Regression



ZHUSUAN



(a)

```
import zhusuan as zs
import tensorflow as tf

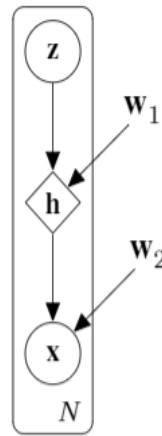
@zs.meta_bayesian_net(scope="blr")
def build_blr(x, alpha, D):
    bn = zs.BayesianNet()
    w = bn.normal('w', mean=tf.zeros([D]),
                  std=alpha, group_ndims=1)
    y_logit = tf.reduce_sum(
        tf.expand_dims(w, 0)*x, axis=1)
    bn.bernoulli('y', y_logit)
    return bn
```

(b)

# Example: Variational Auto-Encoder



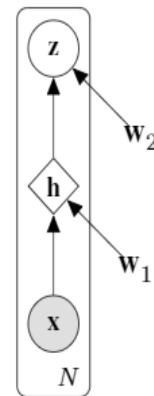
ZHUSUAN



(a)

```
@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)

```
@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
```

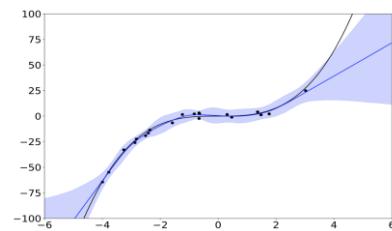
(b)

# “ZhuSuan” Deep Probabilistic Programming

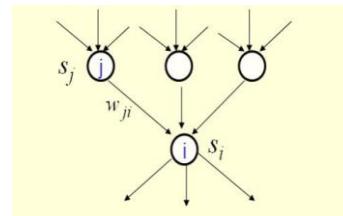
More models

$$D \begin{matrix} \text{document} \rightarrow \text{word} \\ V \end{matrix} = D \begin{matrix} \Theta \\ K \end{matrix} K \begin{matrix} \Phi \\ V \end{matrix}$$

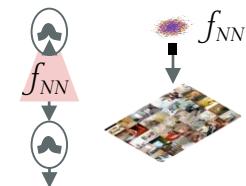
Topic Models; Probabilistic Matrix Factorization  
for text analysis, Recommendor Sys.



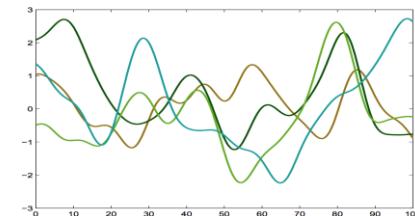
Bayesian Neural Networks  
for uncertainty in deep nets



Deep Belief Nets  
Unsupervised pre-training  
for DNNs



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



Gaussian Processes  
Nonlinear regression problems

[github.com/thu-m1/zhusuan](https://github.com/thu-m1/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/>

The screenshot shows the ZhuSuan documentation website. The top navigation bar includes links for "Docs" (which is active), "Variational Autoencoders" (the current page), and "Edit on GitHub". The main content area is titled "Variational Autoencoders" and discusses the Variational Auto-Encoders (VAE) [VAEKW13]. It explains the generative process of a VAE for modeling binarized MNIST data, involving a latent representation  $(\mathbf{z})$  sampled from a standard Normal distribution, passed through a neural network  $f_{\cdot(\text{NN})}(\mathbf{z})$ , and then through a Bernoulli distribution to produce the observed data  $\mathbf{x}$ . The page also includes a "Build the model" section with sample Python code:

```
import zhusuan as zs
def build_gen(x_dim, z_dim, n, n_particles=1):
    bn = zs.BayesianNet()
```

Below this, it explains the generative process and provides another code snippet for generating latent representations:

```
# z ~ N(z|0, I)
z_mean = tf.zeros([n, z_dim])
z = bn.normal("z", z_mean, std=1., group_ndims=1, n_samples=n_particles)
```

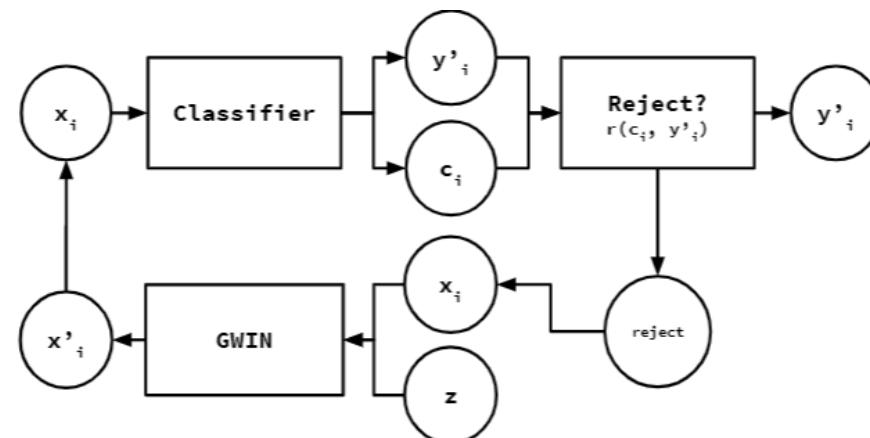
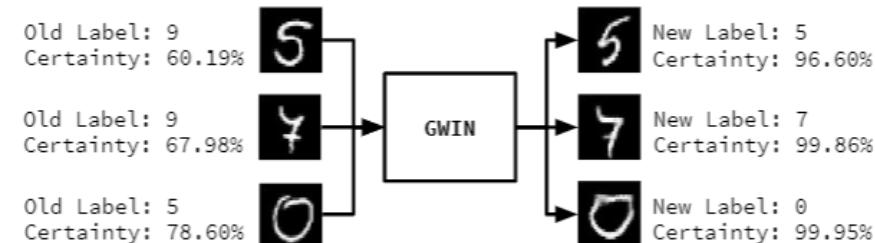
A note at the bottom states: "The method `bn.normal` is a helper function that creates a `Normal` distribution and adds a stochastic node that follows this distribution to the `BayesianNet` instance. The returned `z` is a `StochasticTensor`, which is Tensor-like and can be mixed with Tensors and fed into almost any Tensorflow primitives."

# **Examples**

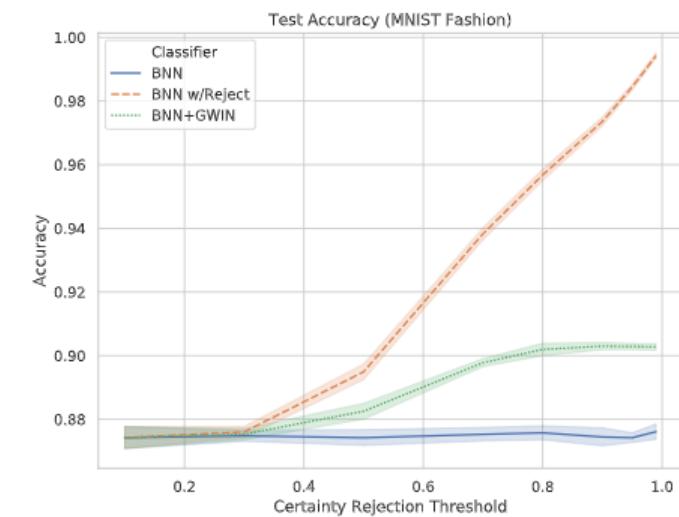
(what problems can be solved by Bayesian deep learning?)

# Example: Learning with Rejection

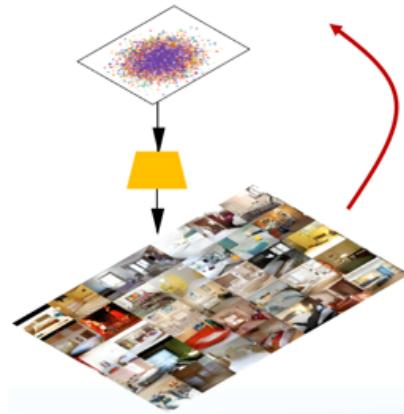
- ◆ Leverage deep generative models to transfer “rejected” samples into high-confidence region



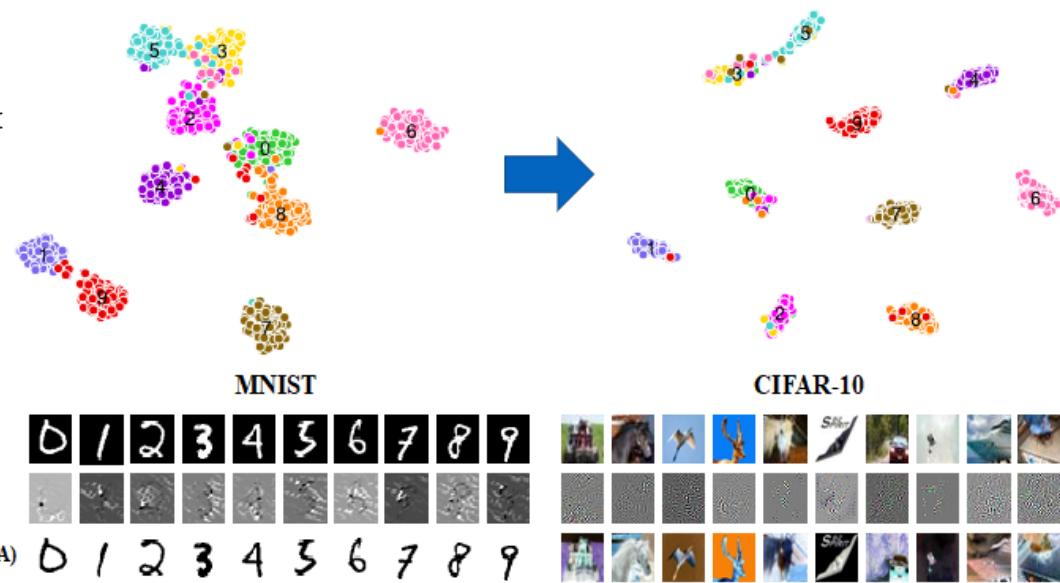
Generative Well-intentioned Networks (GWIN)  
(Cosentino & Zhu, NeurIPS 2019)



# 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

# Summary

- ◆ Deep generative models are powerful in density modeling for high-dimensional data
- ◆ Representative examples:
  - Variational auto-encoders
  - Generative adversarial networks
  - Flow-based generative models
- ◆ Other recent progress (next lecture ...)
  - Diffusion probabilistic models (DPMs)

# Thanks!



**ZhuSuan: A Library for Bayesian Deep Learning**  
[J. Shi](#), et al., arXiv preprint, arXiv:1709.05870 , 2017

**Online Documents:** <http://zhusuan.readthedocs.io/>