Deep Generative Models
Part 3: Implicit Models

Jun Zhu

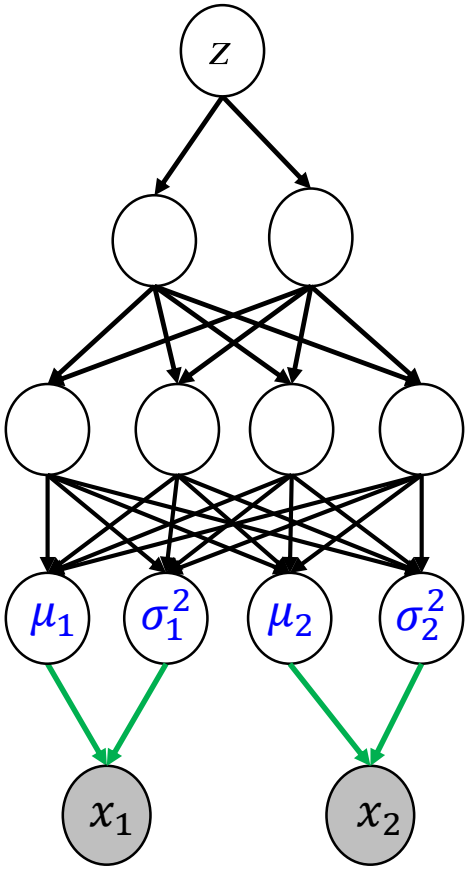
dcszj@mail.tsinghua.edu.cn

Department of Computer Science and Technology

Tsinghua University

An example with MLP

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



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

θ

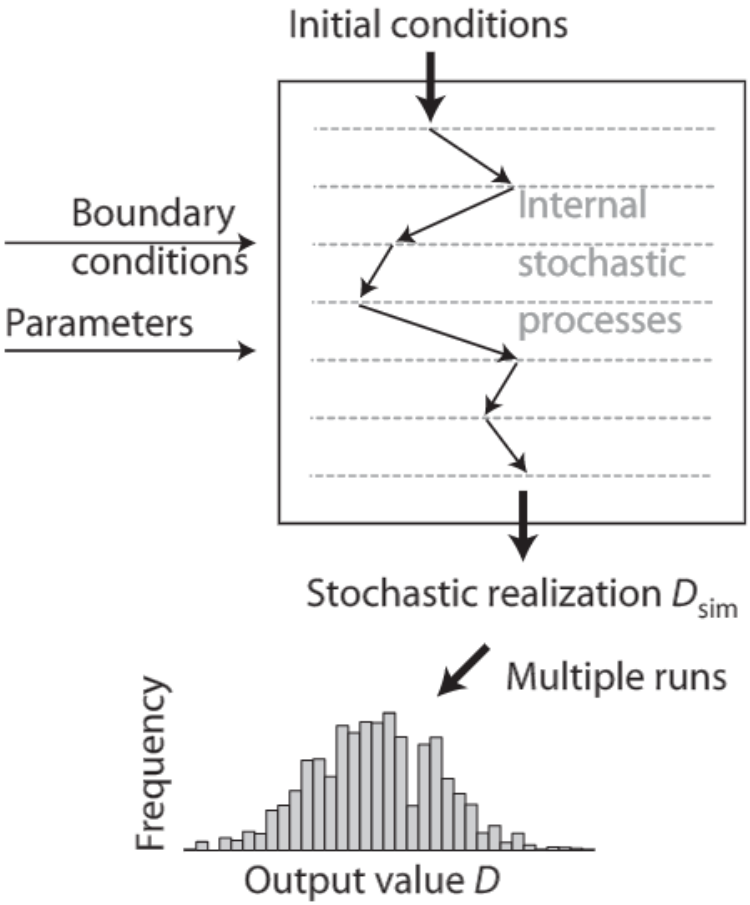
: deterministic function

: random sampling

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

Implicit Deep Generative Models

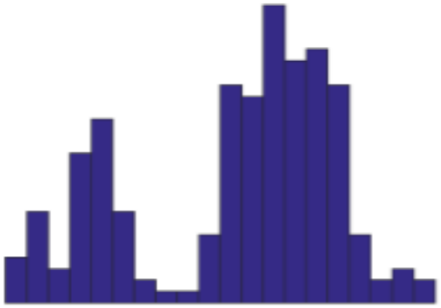
- Generate data with a stochastic process whose likelihood function is not explicitly specified (Hartig et al., 2011)



Learning Deep Generative Models

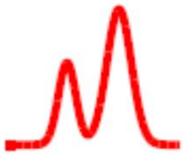
- **Given** a set *D* of unlabeled samples, learn the unknown parameters (or a distribution)

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: $\hat{D} = \{\mathrm{y}_i\}_{i=1}^{M}, \text{where } \mathrm{y}_i \sim p(\mathrm{x}|\theta)$
  - 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

Generative Adversarial Networks (GAN)

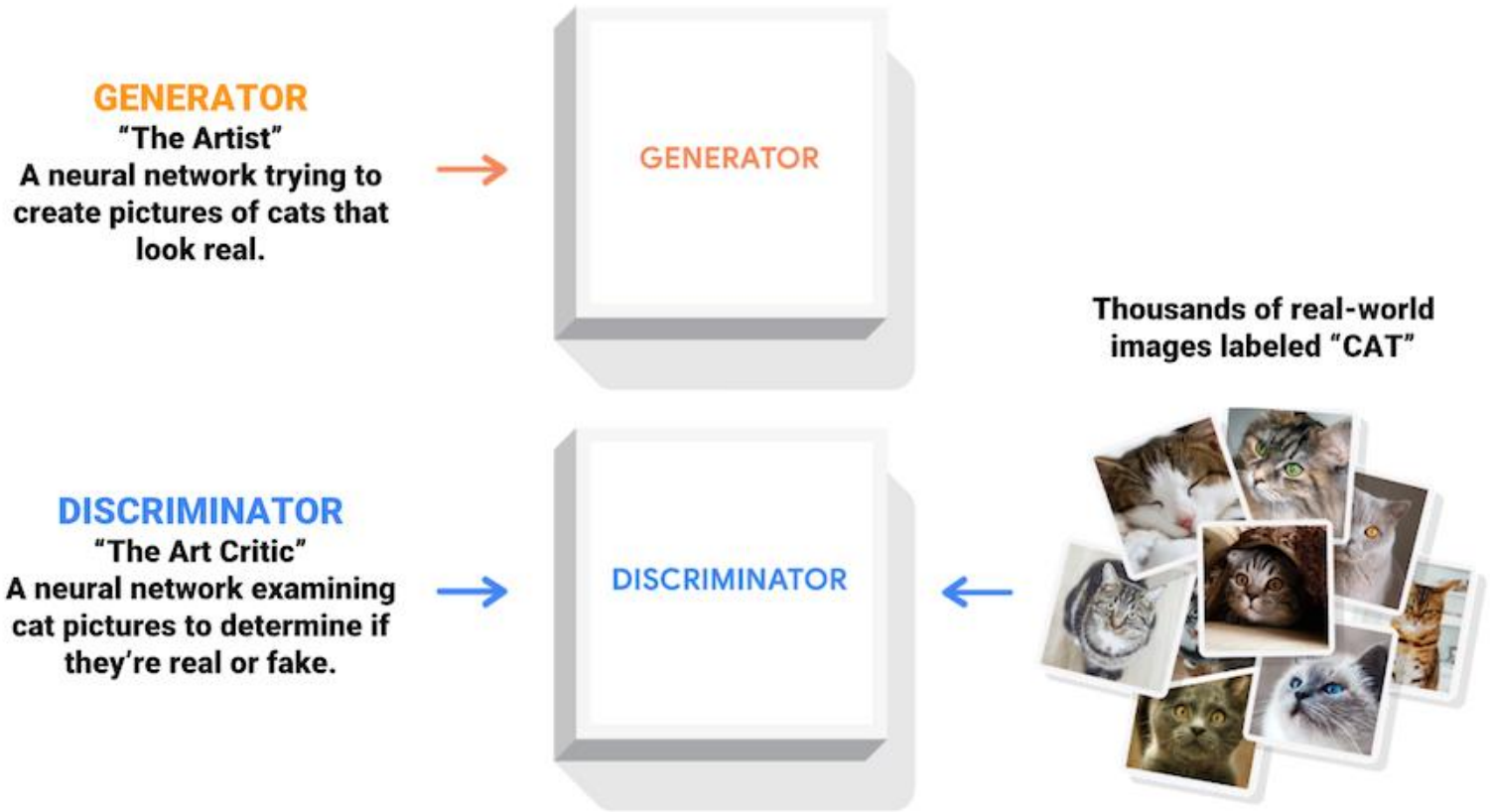
- A game between two players:
  - A discriminator *D*
  - A generator *G*

- *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]

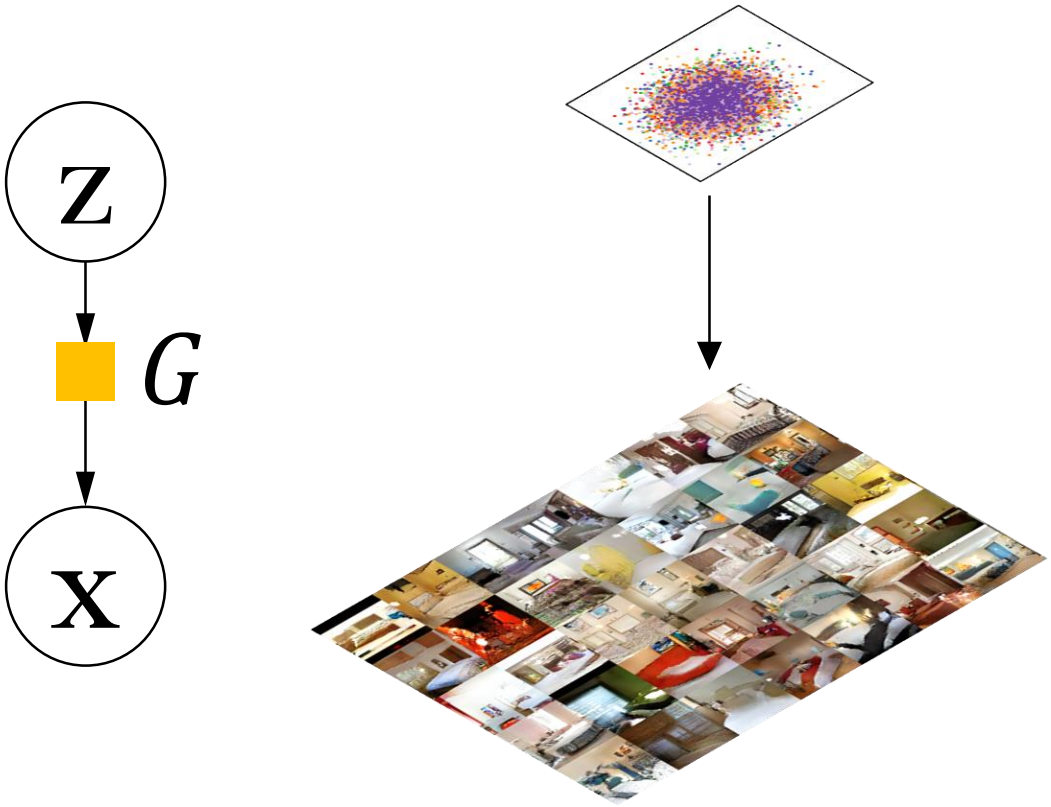
The “Artist” and “Critic” Argument



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

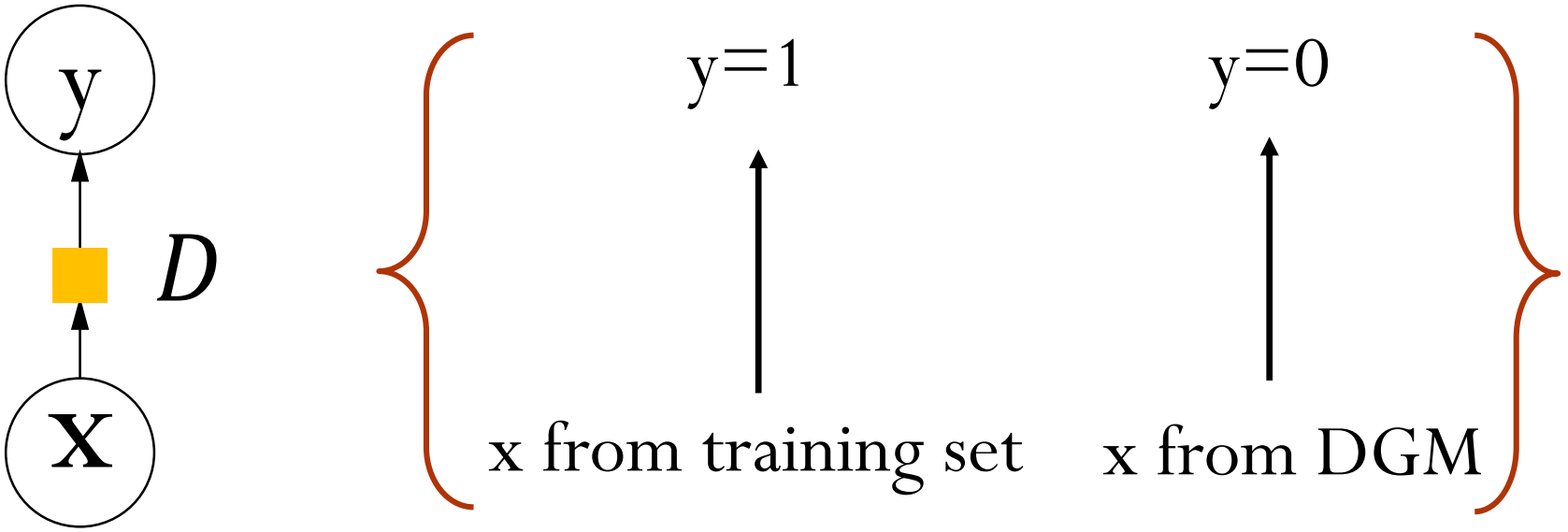
GAN – architecture

- 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

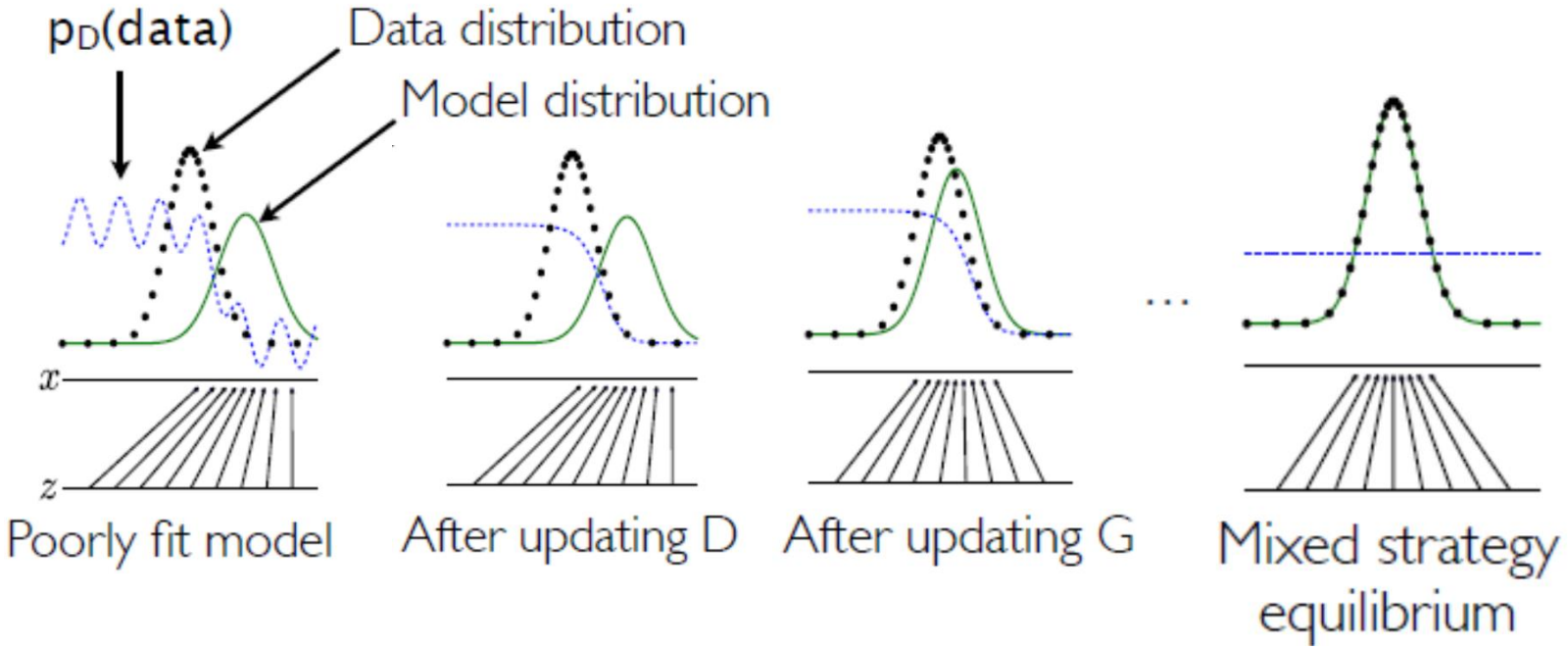
$$\min_G \max_D \; \mathbf{E}_{p_{\text{data}}(\mathrm{x})}[\log D(\mathrm{x})] + \mathbf{E}_{p(\mathrm{z})}\left[\log\left(1 - D\left(G(\mathrm{z})\right)\right)\right]$$

  - 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

More Distances on Distributions

- Kullback-Leibler (KL) divergence

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

- Jensen-Shannon (JS) distance

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

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

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

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

  - the "cost" of the optimal transport plan

[M. Arjovsky, S. Chintala, L. Bottou. Wasserstein Generative Adversarial Networks. ICML, 2017]

GAN – objective (revisited!)

- 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

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

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

An Example: Distance Matters!

- Setup:
  - A uniform distribution $Z \sim U[0,1]$
  - P0 is the distribution of $(0, Z) \in \mathbb{R}^2$
  - Let $g_\theta(z) = (\theta, z)$
- Then, we have
  - $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$
  - $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ , \end{cases}$
  - $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$

[M. Arjovsky, S. Chintala, L. Bottou. Wasserstein Generative Adversarial Networks. ICML, 2017]

An Example: Distance Matters!

- Setup:
  - A uniform distribution $Z \sim U[0, 1]$
  - P0 is the distribution of $(0, Z) \in \mathbb{R}^2$
  - Let $g_\theta(z) = (\theta, z)$

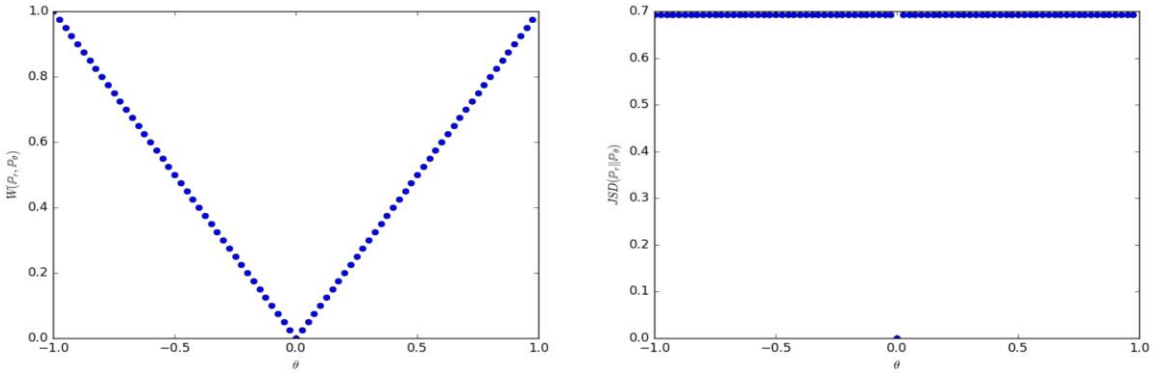


Figure 1: These plots show $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ when ρ is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

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.

[M. Arjovsky, S. Chintala, L. Bottou. Wasserstein Generative Adversarial Networks. ICML, 2017]

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

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

Wasserstein GAN

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

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

  - 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 $p_{\text{data}}(\mathrm{x})$ and $p_{\text{model}}(\mathrm{x})$!
  - Weight clipping is applied to ensure 1-Lipschitzness.

[M. Arjovsky, S. Chintala, L. Bottou. Wasserstein Generative Adversarial Networks. ICML, 2017]

An Illustration with WGAN

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

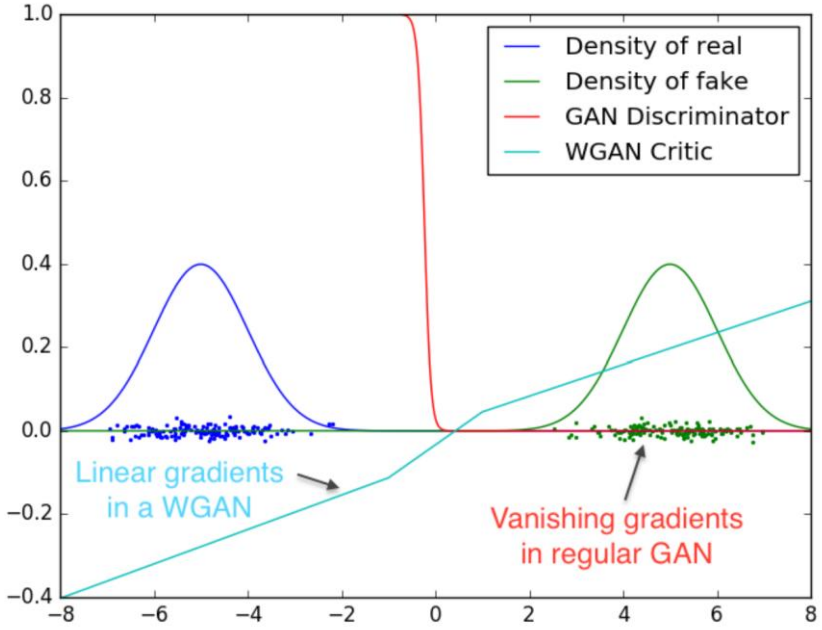


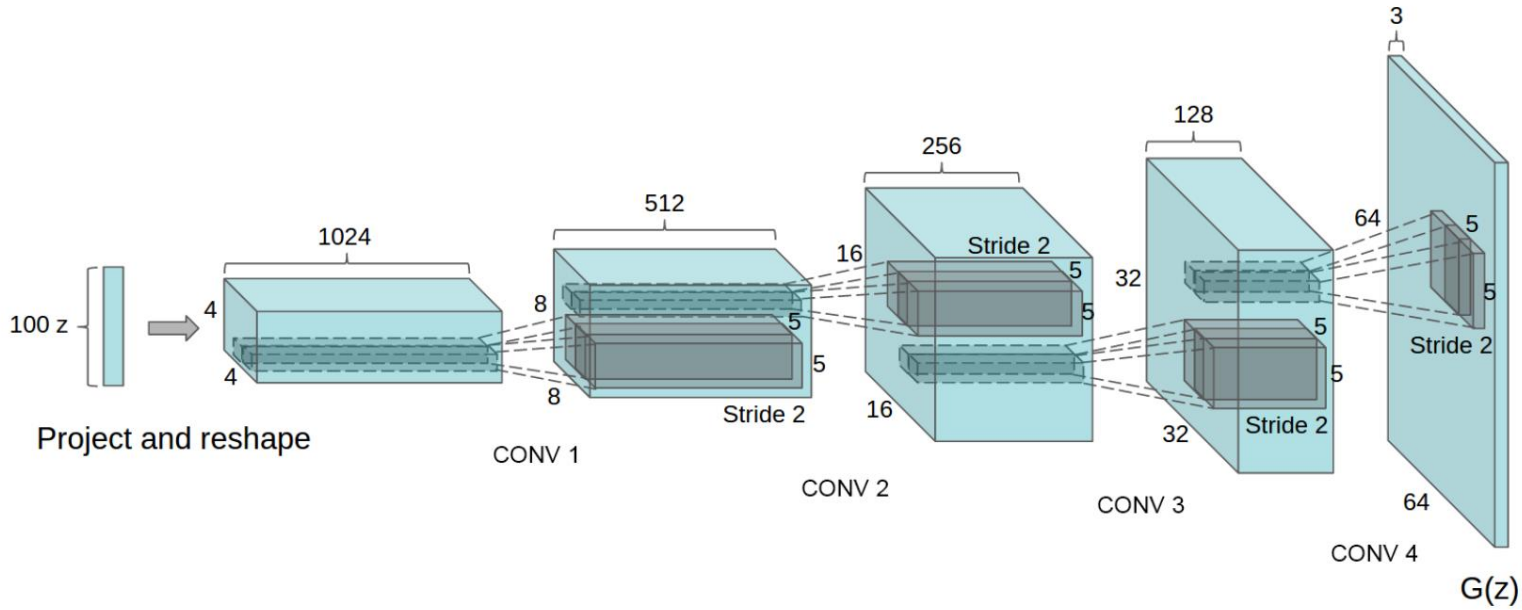
Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

[M. Arjovsky, S. Chintala, L. Bottou. Wasserstein Generative Adversarial Networks. ICML, 2017]

DCGAN: Deep Convolutional GANs

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.



[Radford et al., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, arXiv:1511.06434, 2016]

Understand and Stabilize GAN via Control Theory

- Minimax formulation of GAN

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

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

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

- However, training GANs is typically unstable, with mode collapsing!
- Some analysis based on equilibrium:

$$D^* = \arg\max_D V(G, D); G^* = \arg\min_G V(G, D^*)$$

- A gap with practice!

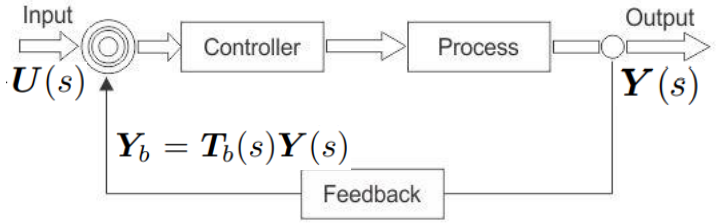
$$D^t = D^{t-1} + \frac{\partial V(G^{t-1}, D^{t-1})}{\partial D^{t-1}}; \quad G^t = G^{t-1} + \frac{\partial V(G^{t-1}, D^t)}{\partial G^{t-1}}$$

[Xu, Li, Zhu, Zhang. *Understanding and Stabilizing GANs' Training Dynamics Using Control Theory.* ICML 2020]

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: $\mathcal{F}(\frac{dh(t)}{dt}) = s\mathcal{F}(h)$ => $\boldsymbol{Y}(s) = \boldsymbol{T}(s)\boldsymbol{U}(s)$
  - 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:

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



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

Understand and Stabilize GAN via Control Theory

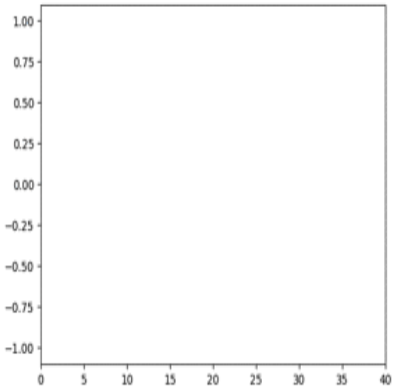
- An example with Dirac GAN (Mescheder et al., ICML 2018)

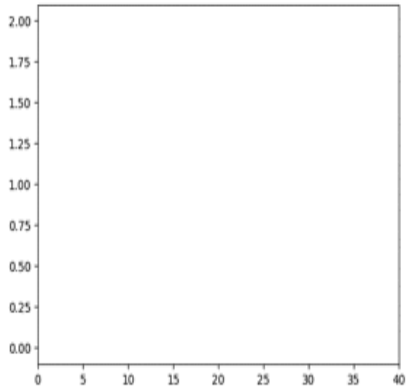
$$p(x) = \delta(x - c)$$
$$p_G(x) = \delta(x - \theta)$$

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = c\phi - \theta\phi$$

$$D(x) = \phi x, |\phi| < 1$$

$$\phi^* = \begin{cases} 1, c - \theta > 0 \\ -1, c - \theta < 0 \end{cases}$$





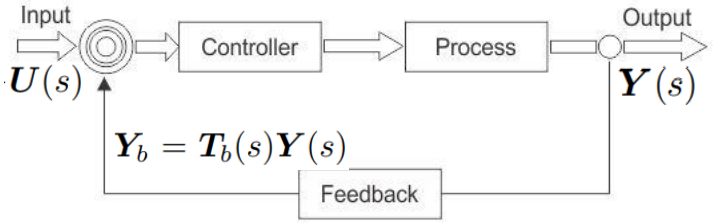
$$\frac{d\phi(t)}{dt} = c - \theta \qquad \frac{d\theta(t)}{dt} = \phi$$

- What about the dynamics?

Understand and Stabilize GAN via Control Theory

- Closed-loop control (CLC) provides a general recipe to stabilize (Kailath, 1980)
  - Adjust the input according to the output:

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



- Apply to Dirac GAN (from unstable to stable):

$$\begin{cases} \boldsymbol{\Phi}(s) = \frac{s}{s^2+1}\boldsymbol{U}(s), \\ \boldsymbol{\Theta}(s) = \frac{1}{s}\boldsymbol{\Phi}(s) = \frac{1}{s^2+1}\boldsymbol{U}(s) \end{cases} \quad \boldsymbol{T}_b(s) = \lambda \quad \boldsymbol{T}_{c\mathcal{D}}(s) = \frac{\frac{s}{s^2+1}}{1 + \frac{\lambda s}{s^2+1}} = \frac{s}{s^2 + \lambda s + 1}$$

The poles have zero real-parts!
Oscillatory!

$$\frac{d\boldsymbol{\phi}}{dt} = c - \boldsymbol{\theta}(t) - \lambda\boldsymbol{\phi}(t)$$

Extend to Normal GANs

- Objectives

$$\max_D \; V_1(D;G) = \mathbf{E}_{p(x)}[h_1(D(x))] + \mathbf{E}_{p_G(x)}[h_2(D(x))]$$

$$\max_G \; V_2(G;D) = \mathbf{E}_{p_z(z)}[h_3(D(G(z)))]$$

- What about the dynamics for alternating GD?

$$\frac{dD(x,t)}{dt} = p(x)\frac{dh_1(D(x))}{dD(x)} + p_G(x)\frac{dh_2(D(x))}{dD(x)}, \forall x$$

$$\frac{dG(z,t)}{dt} = p_z(z)\frac{dh_3(D(G(z)))}{dD(G(z))}\frac{\partial D(G(z))}{\partial G(z)}, \forall z$$

  - Similar to Dirac GAN by letting $D(x) = \phi x$ and $G(z) = \theta$.

CLC-GAN

- Applying CLC to normal GANs:

$$\frac{dD(x,t)}{dt} = \frac{\partial V_1(D;G)}{\partial D} - \lambda D(x), \forall x.$$

  - Regularizing D towards the zero function.
  - Equivalent to optimize:

$$V_1'(D;G) = V_1(D;G) - \frac{\lambda}{2}\int_{x\in X} D^2(x)dx$$

$$R_t(D) = \int_{x\in X} p_u^t(x)D^2(x)dx$$

  - Using real samples and fake samples to approximate the integral.

Algorithm 1 Cloosed-loop Control GAN

Input: Buffer size N_b, feedback coefficient λ, batch size N, initialized G and D, learning rate η.
Initialize B_r and B_f for real samples and fake samples respectively.
repeat
  Sample a batch of $\{x_r\} \sim p$, $\{x_f\} \sim p_G$ of N samples.
  Update B_r with $\{x_r\}$. Update B_f with $\{x_f\}$.
  Sample a batch of $x_r' \sim B_r$, $x_f' \sim B_f$ of N samples respectively.
  Estimate the objective of D: $U(D) = \frac{1}{N}[\sum_{x\in\{x_r\}} D(x) - \sum_{x\in\{x_f\}} D(x)] - \frac{\lambda}{N}[\sum_{x\in\{x_r'\}} D^2(x) + \sum_{x\in\{x_f'\}} D^2(x)]$.
  Update D to maximize $U(D)$ with learning rate η.
  Estimate the objective of G: $U(G) = \frac{1}{N}\sum_{x\in\{x_f\}} D(x)$.
  Update G to maximize $U(G)$ with learning rate η.
until Convergence

Theoretical analysis

- The equilibrium unchanged:

Lemma 1. *Under the non-parametric setting, CLC-GAN has the same equilibrium as the original GAN, i.e,* $p_G = p$ *and* $D(x) = 0$ *for all* x.

- The local convergence about the parameters:

Theorem 1. *CLC-GAN locally converges when G&D are neural nets.*

Stability Comparison

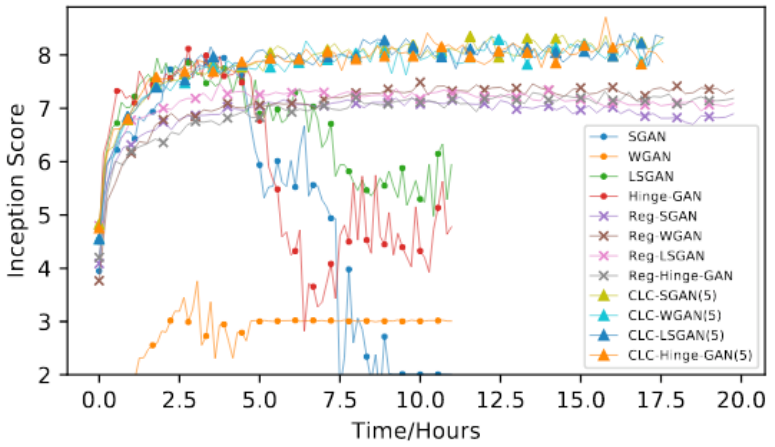
- Either theoretically or empirically stabilized

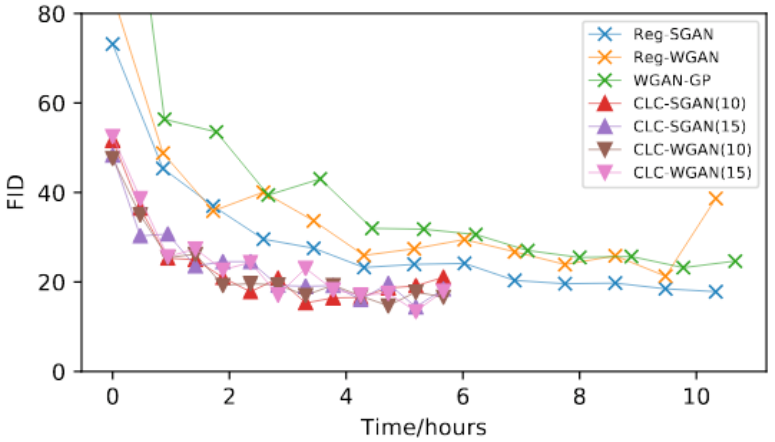
Table 1: The stability characters for the widely-used GANs. Please refer to Appendix A for detailed derivation, which adopts the local linearization technique introduced in Sec. 3.3. With CLC, the training dynamics of Dirac GANs are stable theoretically (see Fig. 1 and Appendix A), and those of normal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

	$\boldsymbol{T}_{\mathcal{D}}(s)$	Stability Dirac GAN/normal GAN	$\boldsymbol{T}_{c\mathcal{D}}(s)$	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



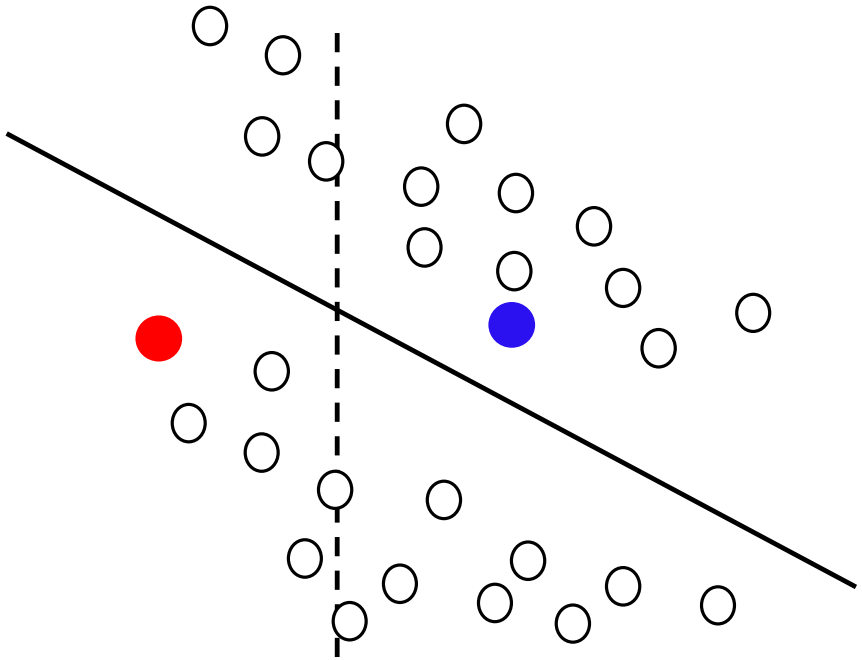


- The inception score on CIFAR10

Method	WGAN	SGAN	Hinge
LR-GAN†	-	7.17	-
SN-GAN‡	-	-	8.22
CR-GAN§	-	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)	$\mathbf{8.49} \pm .07$	$8.44 \pm .08$	$\mathbf{8.54} \pm .03$
CLC-GAN(10)	$8.38 \pm .10$	$\mathbf{8.47} \pm .09$	$8.46 \pm .00$
SN-GAN	3.29	8.17	8.28
CLC-SN-GAN(0.1)	$8.14 \pm .02$	$8.30 \pm .09$	$\mathbf{8.54} \pm .03$

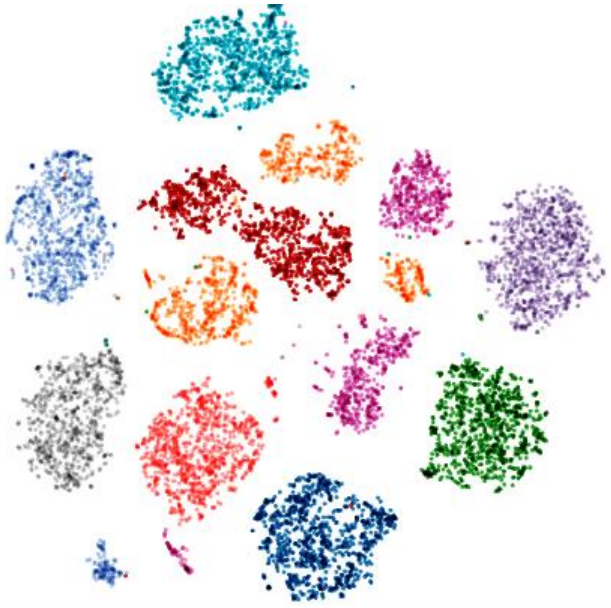
Semi-supervised Learning

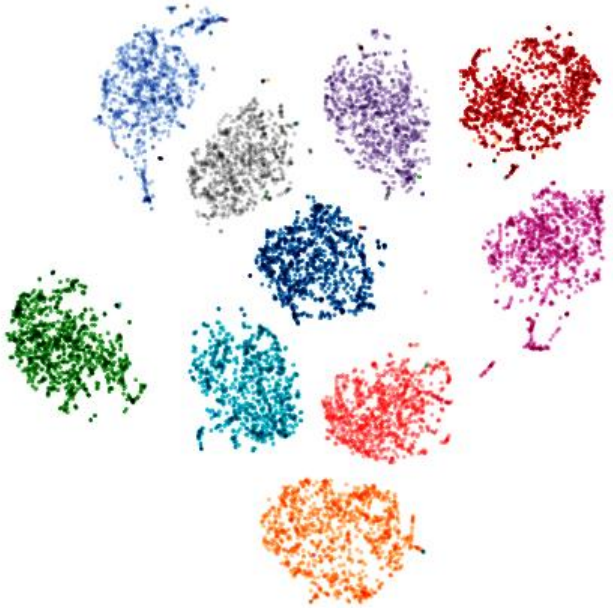
- A toy example



Representation Matters

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

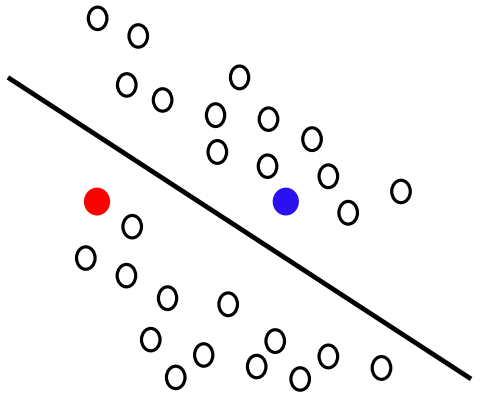




Triple Generative Adversarial Nets

- A minimax game for semi-supervised learning
  - GAN is for unsupervised learning $p_{\text{model}}(\text{x}) = p_{\text{data}}(\text{x})$
  - 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

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

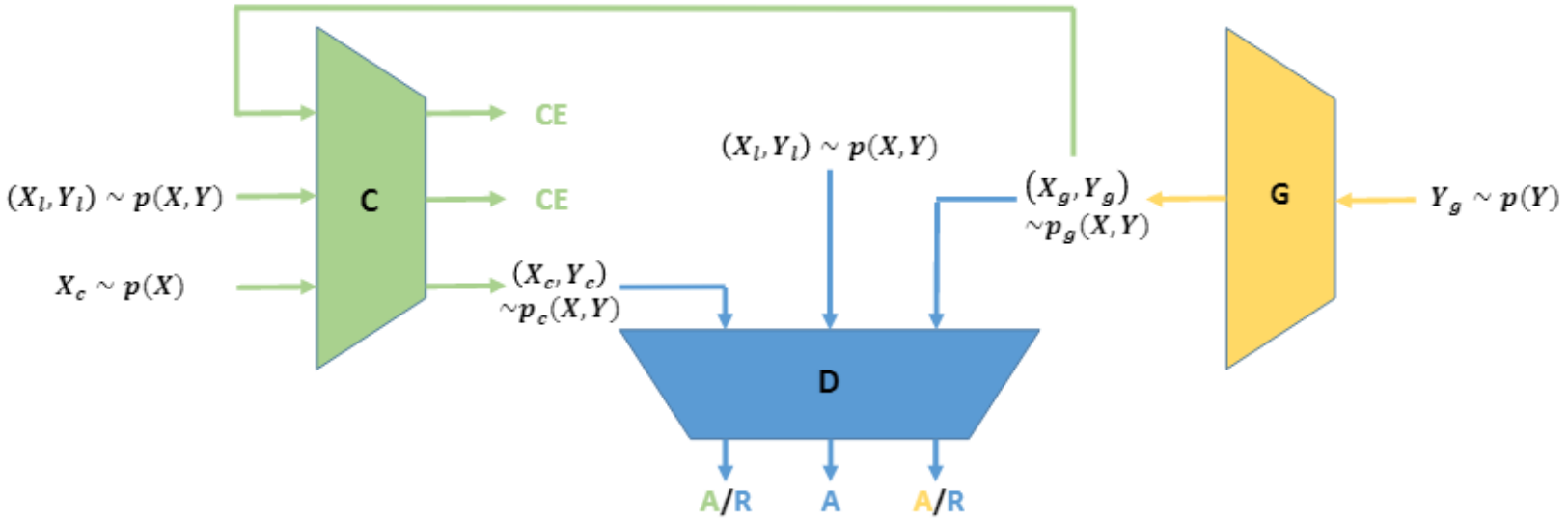


  - Two generators to generate (x, y)
  - A discriminator to distinguish fake (x, y)

[Li et al., NIPS 2017]

Triple-GAN

- The network architecture



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

[Li et al., NIPS 2017]

A minimax game

- The optimization problem

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

  - The hyper-parameter *a* is often set at 1/2

  - The standard supervised loss can be incorporated

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

[Li et al., NIPS 2017]

Major theoretical results

Theorem

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

Lemma

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

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

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

Some Practical Tricks for SSL

- Pseudo discriminative loss: using $(x, y) \sim p_g(x, y)$ as labeled data to train C
  - Explicit loss, equivalent to $KL(p_g(x, y)||p_c(x, y))$
  - Complementary to the implicit regularization by D
- Collapsing to the empirical distribution $p(x, y)$
  - Sample $(x, y) \sim p_c(x, y)$ as true data for D
  - Biased solution: target shifting towards $p_c(x, y)$
- Unlabeled data loss on C
  - Confidence (Springenberg [2015])
  - Consistence (Laine and Aila [2016])

Some Results

- Semi-supervised classification

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

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

Some Results

- Class-conditional generation

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

Some Results

- Disentangle class and style

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

Some Results

- Latent space interpolation on MNIST

Some Results

- Latent space interpolation on SVHN

Some Results

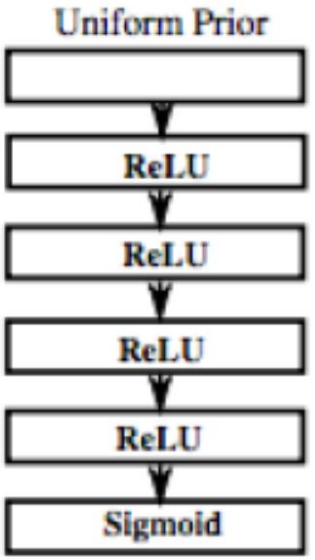
- Latent space interpolation on CIFAR10

Moment-Matching DGMs

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

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

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



  - Kernel MMD:

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

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

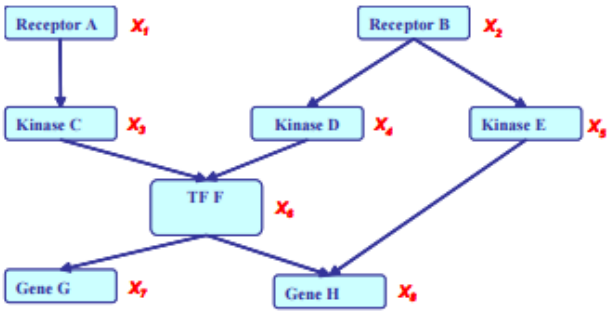
[Li et al., ICML 2015]

Conditional Models

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

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

  - Building large networks:



Conditional Moment-Matching Networks

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

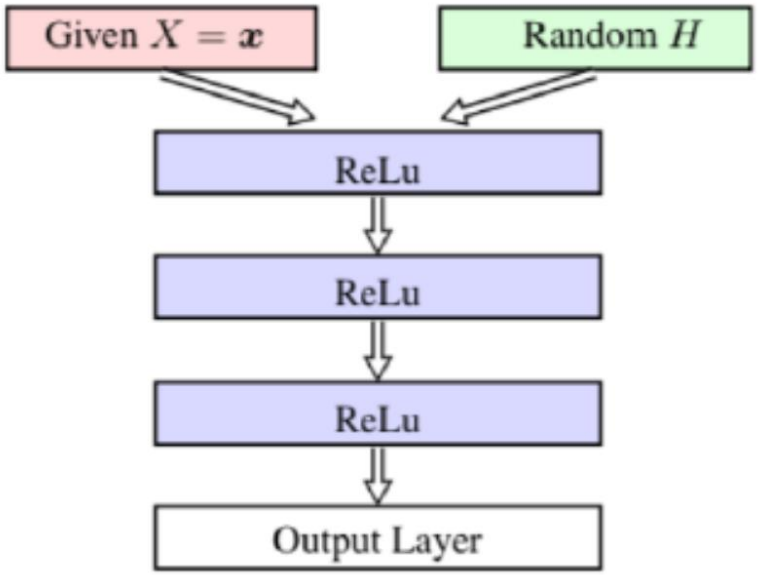
$$\mathcal{D}^s_{XY} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$$

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

- A DGM:

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

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



[Ren et al., NIPS 2016]

Conditional Moment-Matching Networks

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

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

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

[Ren et al., NIPS 2016]

Connections with MMD

- MMD:

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

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

- CMMD:

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

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

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

[Ren et al., NIPS 2016]

Mini-batch Training Algorithm

- BP to compute the gradients:

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

[Ren et al., NIPS 2016]

Some Results – Classification

Classification accuracy on MNIST:

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

Classification accuracy on SVHN:

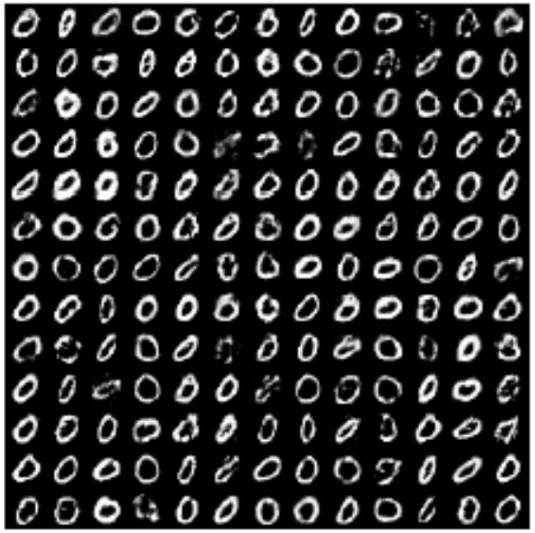
Model	Error Rate
CVA+Pegasos	25.3
CGMMN-CNN	3.13
CNN	4.9
CMMVA	3.09
Stochastic Pooling	2.80
Network in Network	2.47
Maxout Network	2.35
DSN	1.92

Some Results – Generation

Generating performance on MNIST:

(a) MNIST samples

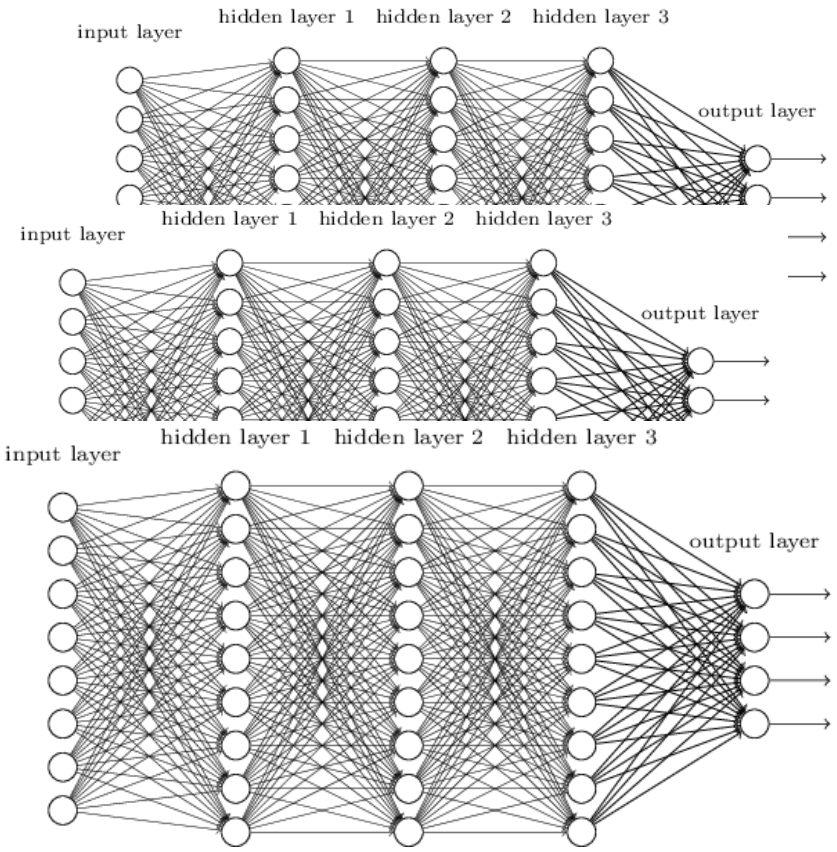
(b) Random CGMMN samples



(c) Samples conditioned on label 0

Some Results – Bayesian Dark Knowledge

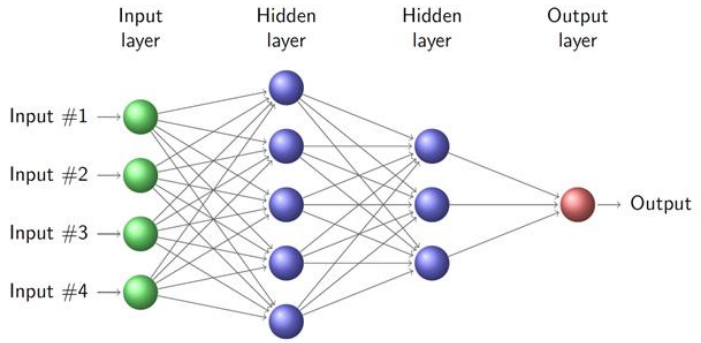
- Distilling the knowledge (Hinto et al., 2015) & Bayesian dark knowledge (Korattikara et al., 2015)



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

Bayes prediction is expensive!

Teach a student network for prediction



Some Results – Bayesian Dark Knowledge

Distilling knowledge in Bayesian networks on Boston housing dataset:

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

PBP prediction	Distilled by CGMMN
2.574 ± 0.089	2.580 ± 0.093

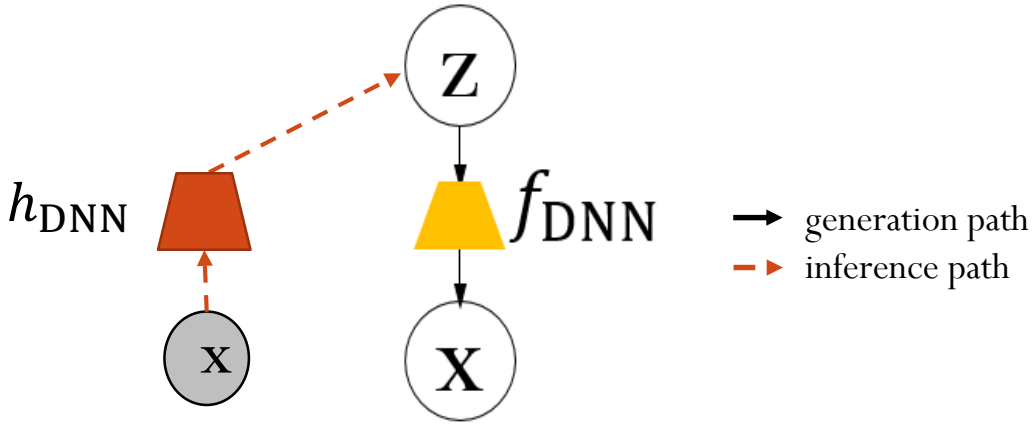
- measured by MSE
- MLP network with three hidden layers and $(100, 50, 50)$ hidden units for middle layers.
- $N = 3,000$ sample pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
- $\mathbf{x}_i$ is generated by adding noise into training data to avoid fitting the testing data directly

"Explicitly" Define Variational Distribution q via an Encoder

- Variational Auto-Encoders (VAE, Kingma & Welling, 2013)

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

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



- Jointly optimize the parameters of decoder and encoder networks θ, ϕ: SGD

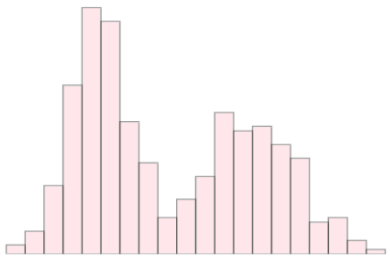
Variational inference with Implicit q

- Can we do variational inference when q is an implicit distribution?

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

  - The objective can be estimated via Monte Carlo, but we can't computer gradients!

- **A fundamental task**: Can we directly estimate the gradient function from samples?

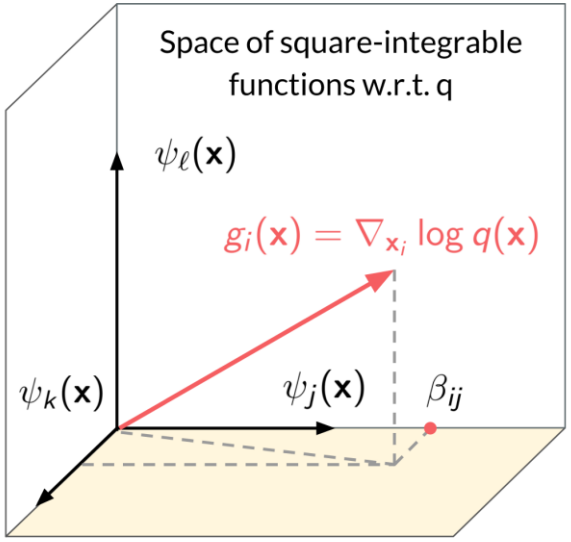


$x^{1:M} \sim q$

$$g(\mathbf{x}) = \nabla_{\mathbf{x}} \log q(\mathbf{x})$$

[A Spectral Approach to Gradient Estimation for Implicit Distributions. Shi et al., ICML 2018]

A Spectral Approach to Gradient Estimation for Implicit Distributions



We proved that (under mild assumptions)

$$\nabla_{x_i} \log q(\mathbf{x}) = -\sum_{j=1}^{\infty} \left[\mathbb{E}_q \nabla_{x_i} \psi_j(\mathbf{x})\right] \psi_j(\mathbf{x})$$

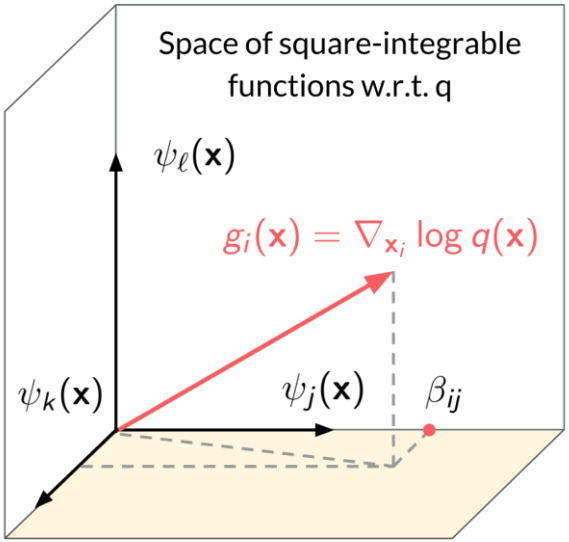
This orthonormal basis can be constructed by spectral decomposition of a p.d. kernel

$$\int k(\mathbf{x}, \mathbf{y}) \psi_j(\mathbf{y}) q(\mathbf{y}) d\mathbf{y} = \mu_j \psi_j(\mathbf{x})$$

[A Spectral Approach to Gradient Estimation for Implicit Distributions. Shi et al., ICML 2018]

A Spectral Approach to Gradient Estimation for Implicit Distributions

- Approximate the eigenfunctions by Nyström method [Nyström, 1930]
- Truncate the series with a finite number of basis functions, according to large eigenvalues



Monte Carlo Nyström approximation

$$\nabla_{x_i} \log q(\mathbf{x}) = -\sum_{j=1}^{J} \left[\mathbb{E}_q \nabla_{x_i} \bar{\psi}_j(\mathbf{x})\right] \bar{\psi}_j(\mathbf{x})$$

Spectral Stein Gradient Estimator

[A Spectral Approach to Gradient Estimation for Implicit Distributions. Shi et al., ICML 2018]

A Spectral Approach to Gradient Estimation for Implicit Distributions

Theorem (Error Bound) Given mild assumptions, the error

$$\int |\hat{g}_i(\mathbf{x}) - g_i(\mathbf{x})|^2 q(\mathbf{x})\, d\mathbf{x}$$

Our estimator True gradient

is bounded by

$$J^2 \left(O_p\left(\frac{1}{M}\right) + O_p\left(\frac{C}{\mu_J \Delta_J^2 M}\right)\right) + J O_p\left(\frac{C}{\mu_J \Delta_J^2 M}\right) + \|g_i\|_{\mathcal{H}}^2 O(\mu_J),$$

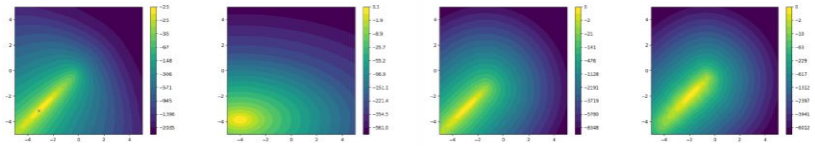
Estimation error

Approximation error due to truncation

where $\Delta_J = \min_{1 \le j \le J} |\mu_j - \mu_{j+1}|$

[A Spectral Approach to Gradient Estimation for Implicit Distributions. Shi et al., ICML 2018]

A Spectral Approach to Gradient Estimation for Implicit Distributions

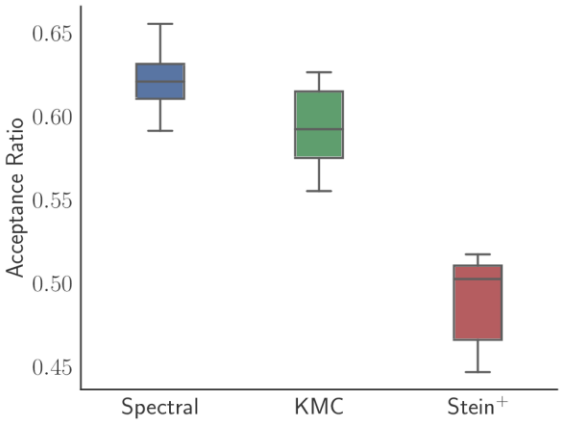


(b) True posterior (c) VI (factorized) (d) HMC (e) KIVI

(b) Implicit VAE, w/o entropy (c) Implicit VAE, Spectral

Improving Deep Generative Models

Variational Inference with Implicit Distributions

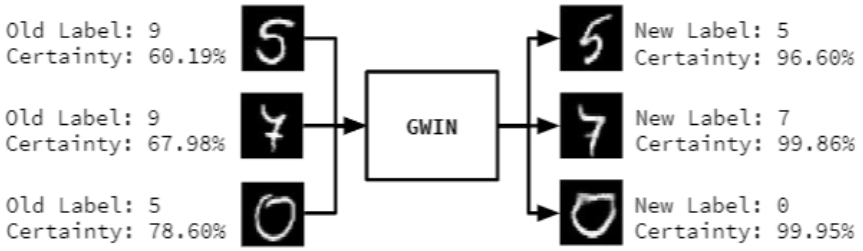


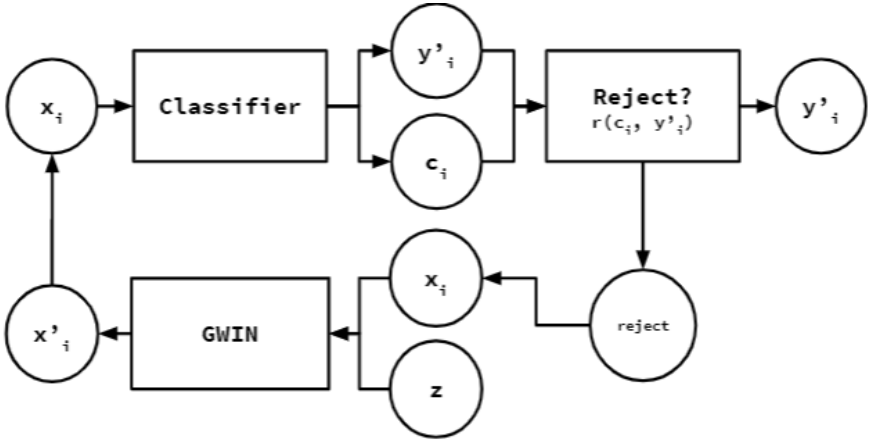
Gradient-free Hamiltonian Monte Carlo

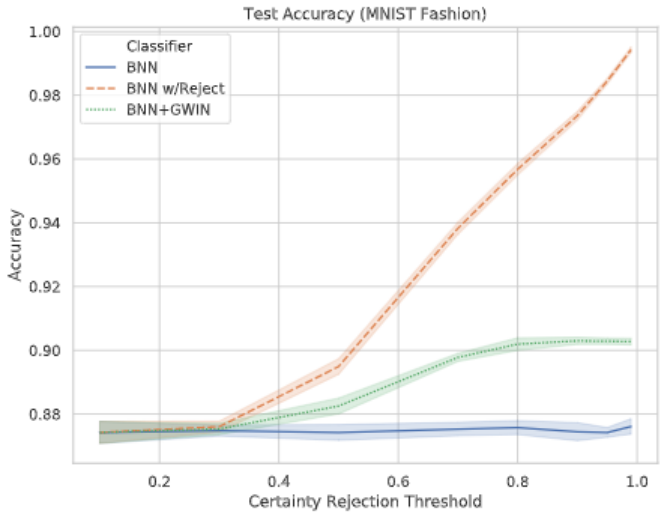
[A Spectral Approach to Gradient Estimation for Implicit Distributions. Shi et al., ICML 2018]

More 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)

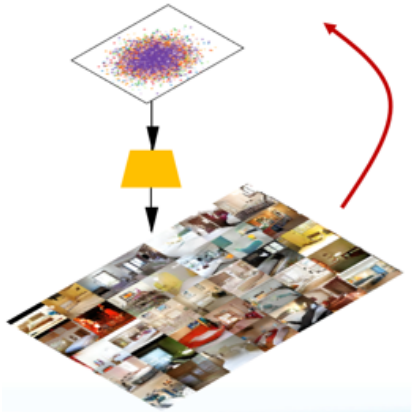
More Examples: Adversarial Robustness

- Deep networks are vulnerable to small adversarial noise

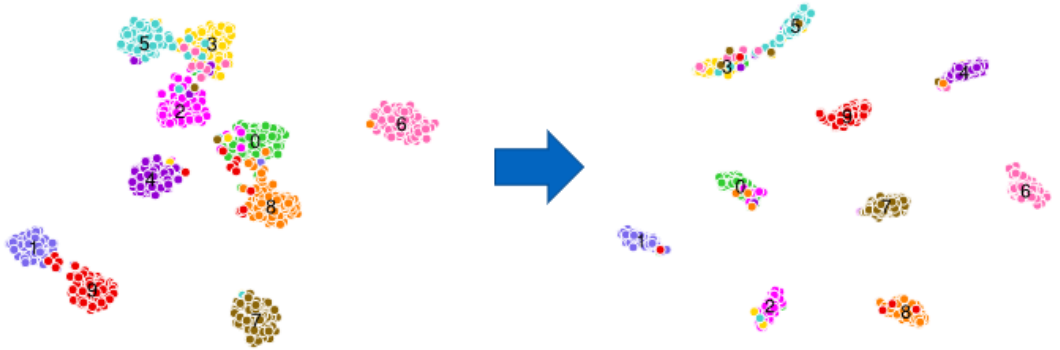
AlexNet: lionfish， confidence 81.3%
VGG-16: lionfish， confidence 93.3%
ResNet-18: lionfish， confidence 95.6%

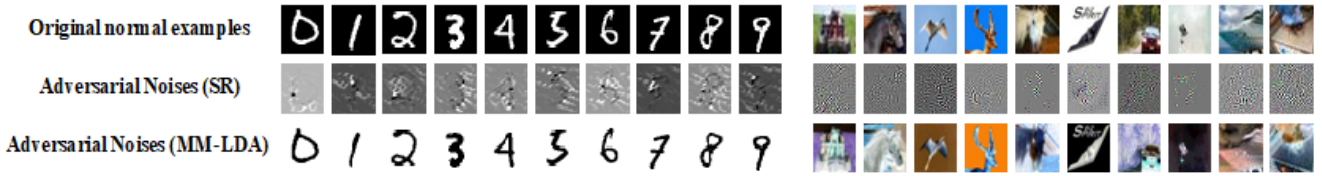
Max-Mahalanobis Linear Discriminant Analysis Networks (Pang et al., ICML 2018)

More 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

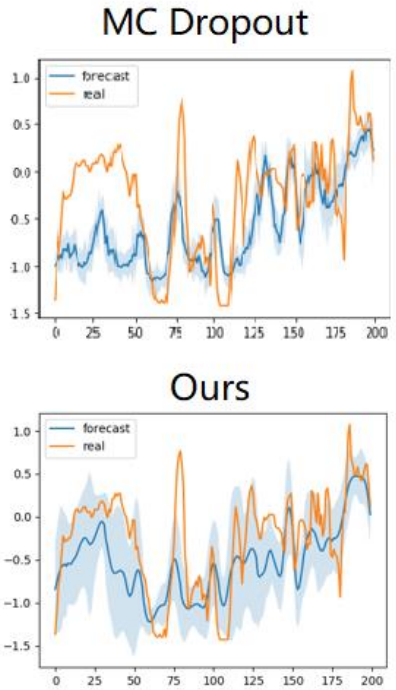
Max-Mahalanobis Linear Discriminant Analysis Networks (Pang et al., ICML 2018)

More Example: 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

- Function-space particle-based variational inference

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]

Summary

- GANs provide an effective mechanism to learn implicit generative models with a lot of developments
  - Wasserstein GAN
  - Triple-GAN
- Moment-matching is a classic method and can be extended to deal with deep generative models
- Implicit generative models can be used in variational inference
- Generative models benefit adversarial robustness and learning with rejection…

Thanks!

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

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