Generative Adversarial Networks

Jie Tang

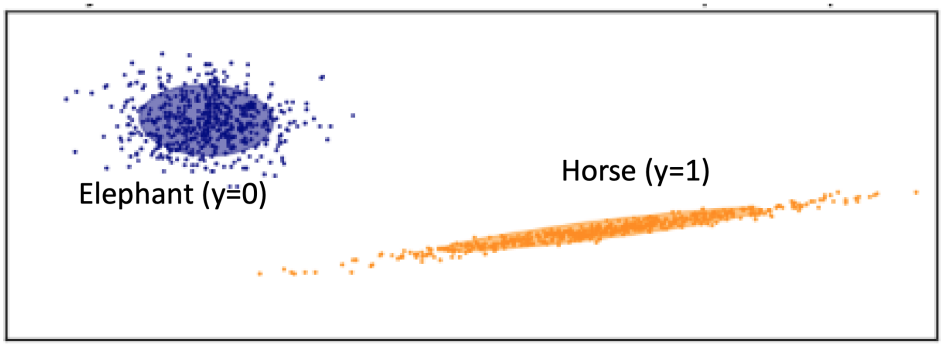
Tsinghua University

May 6, 2020

Generative Models

- Density Estimation
  - Discriminative model: $p(y|x)$
    - $y = 0$ for elephant, $y = 1$ for horse
  - Generative model: $p(x|y)$

$$p(x \mid y=0) \qquad p(x \mid y=1)$$



- Sample Generation

Generative Models

Training samples

Model samples

Training samples

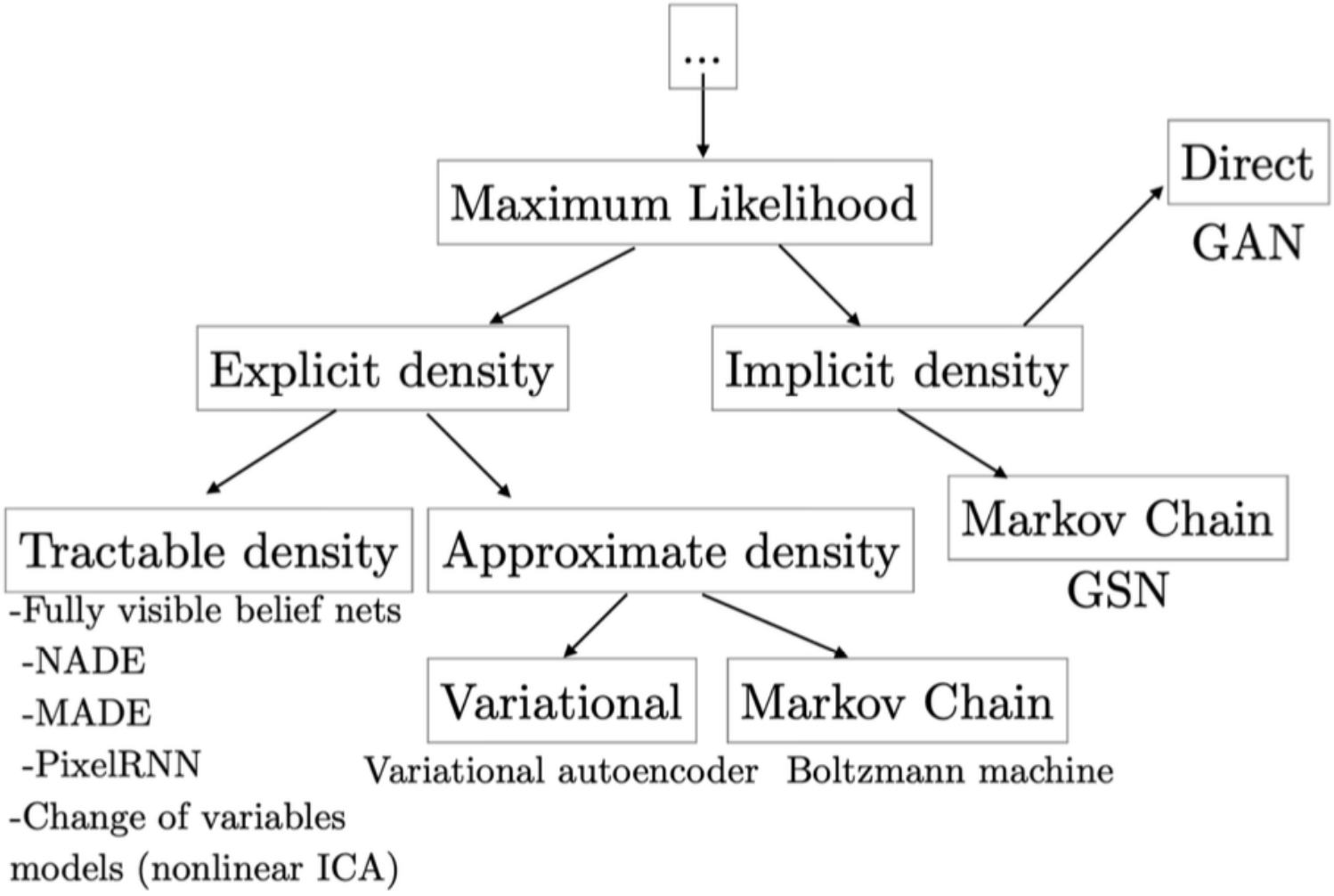
Why Study Generative Modeling

- Generative models focus on modeling a distribution over target data. The learned distribution P_g is an estimate of the real distribution P_{data}.
  - As soon as we obtained P_g, we can generate similar samples to natural data, e.g. images that look real.
- Generative models can be trained with missing data and can provide predictions on inputs that are missing data.
- Training and sampling from generative models is an excellent test of our ability to represent and manipulate high-dimensional probability distributions.
- Generative models can be incorporated into reinforcement learning in several ways.

Generative Model

- Two categories of generative models:
  - **Explicit generative models** Explicitly perform density estimation on samples $\{\boldsymbol{x}_i\}_{i=1}^N$ so that we can derive a log-likelihood of density, and optimize under max-likelihood criterion. For example, mixture of Gaussian model.
  - **Implicit generative models** Not explicitly perform density estimation through max-likelihood criterion. They focus directly on the generation. For example, GAN.
- When real data's distribution is too complex, explicit generative models like VAE have to resort to approximation to perform density estimation, which incurs bias intrinsically.
- Thus, we may prefer implicit generative models like GAN.

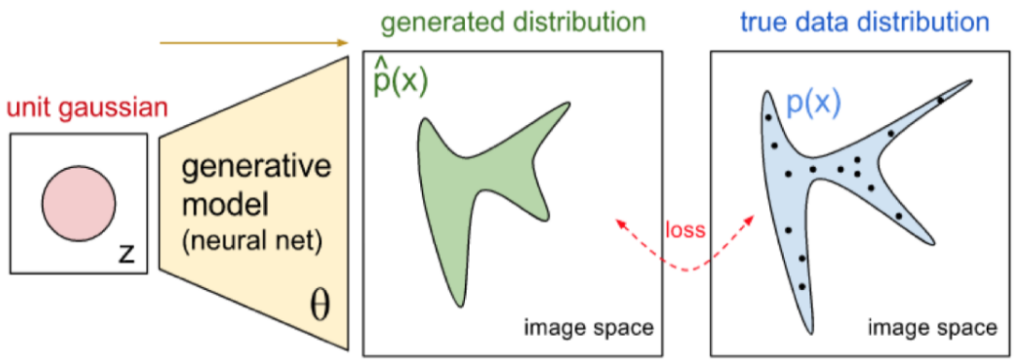
Generative Model



- Deep generative models learning via maximum likelihood, differ with respect to how they represent or approximate the likelihood

Implicit Generative Model

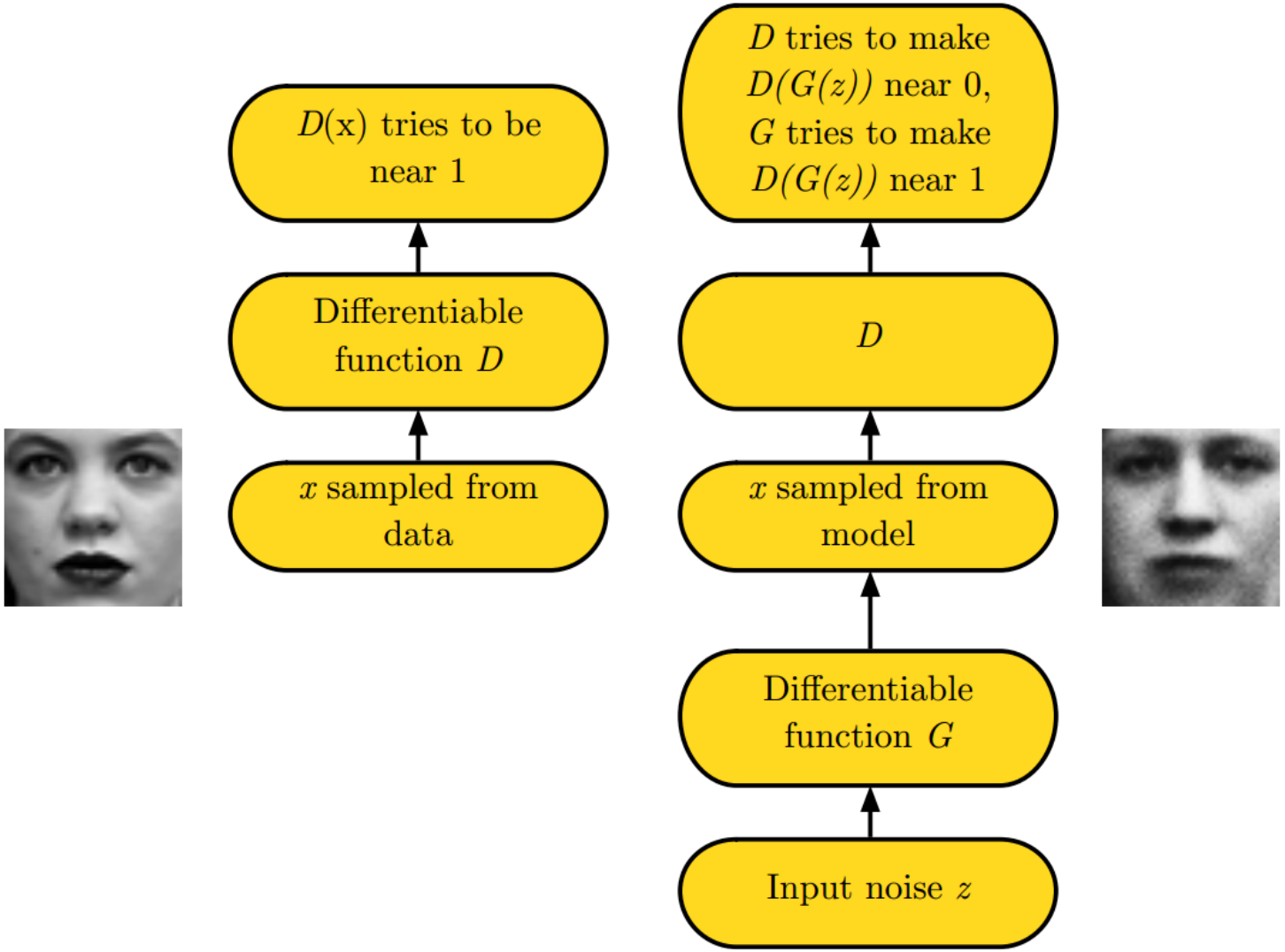
- Implicit Generative Models implicitly define a probability distribution through a mapping
- The input is a vector $\boldsymbol{z}$ sampled from some fixed, simple distribution, for example, a standard Gaussian.
- The model denoted by G could be any mapping, for example, a mapping given by a neural network.
- The output $G(\boldsymbol{z})$ then obeys some other distribution. We are interested when the mapping G maps $\boldsymbol{z}$ into the real data space $\boldsymbol{x}$.



GAN: Generative Adversarial Network

- GAN depends on a generator G and a discriminator D which are adversaries to each other.
- The generator G learns to generate a distribution by mapping $\boldsymbol{z}$.
- The discriminator D learns to distinguish P_g from P_{data}.
- G tries to fool D by generating samples that are hard for D to distinguish from the real data. When a good D is unable to tell P_g and P_{data} apart, the generator will be satisfying.
- The generator and discriminator are trained simultaneously.
- Unlike VAE, GAN is designed to be unbiased. Infinite data leads to $P_g = P_{data}$.

GAN Model Architecture[1]



[1] Goodfellow I J, Pougetabadie J, Mirza M, et al. Generative Adversarial Nets[J]. Advances in Neural Information Processing Systems, 2014, 3:2672-2680.

Discriminator's Cost

$$J^{(D)} = -\frac{1}{2}\mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] - \frac{1}{2}\mathrm{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- Where $p_{data}(\mathbf{x})$ is the distribution of data samples and $p_z(\mathbf{z})$ represents the prior on input noise variables, $\mathbf{z} \sim p_z(\mathbf{z})$.
- Minimizing the cost $J^{(D)}$ is equal to maximizing the following $\mathrm{E}_{\mathbf{x}\sim P_{data}}[logD(\mathbf{x})] + \mathrm{E}_{\tilde{\mathbf{x}}\sim P_g}[log(1 - D(\tilde{\mathbf{x}}))]$, where $\tilde{\mathbf{x}} \sim G(\mathbf{z})$.
- Train a standard binary classifier on two minibatches of data: one from the dataset with label 1, and one from the generator with label 0.

Objective Function

- Formally, we can express the game between G and D with the minimax objective

$$\min_G \max_D \mathrm{E}_{\mathbf{x}\sim P_{data}}[logD(\mathbf{x})] + \mathrm{E}_{\tilde{\mathbf{x}}\sim P_g}[log(1 - D(\tilde{\mathbf{x}}))]$$

- where $\tilde{\mathbf{x}}$ represents the sample from P_g, which is generated by G from $\mathbf{z} \sim p_z(\mathbf{z})$, that is, $\tilde{\mathbf{x}} \sim G(\mathbf{z})$.
- The generator G's input $\mathbf{z}$ is sampled from some simple noise distribution $p(\mathbf{z})$ (e.g. uniform or Gaussian).

Minimax Game[2]

- Minimax is a decision rule used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario.
- In this case, G and D are two players, which have opposite loss function, so this game is also zero-sum games.
- In two-player zero-sum games, the different game theoretic solution concepts of Nash equilibrium, minimax, and maximin all give the same solution, where all players use mixed strategies.
- Therefore according to game theory, we can expect G and D arrive this point of equilibrium with similar mixed strategies, where both G and D have great performance.

[2] Fudenberg D, Tirole J. Game Theory[J]. Mit Press Books, 1991, 1(7):841-846.

Minimax Game

- The simplest version of the game is a zero-sum game.

$$J^{(D)} = -\frac{1}{2}\mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] - \frac{1}{2}\mathrm{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

$$J^{(G)} = -J^{(D)}$$

- Equilibrium is a saddle point of the $J^{(D)}$.
- G minimizes the log-probability of D being correct. $J^{(G)}$ may not provide sufficient gradient for G to learn well.

Minimax Game

$$\min_G \max_D V(D,G) = \mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]$$

- The generative model G will map $\mathbf{z}$ to data space as $G(\mathbf{z};\theta_g)$, where G is a differentiable function represented by a multilayer perception with parameters $G(\mathbf{z};\theta_g)$.
- The multilayer perception $D(\mathbf{x};\theta_d)$ outputs a single scalar and $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the $p_{data}(\mathbf{x})$ rather than $G(\mathbf{z};\theta_g)$.

Goal:Train D to maximize the probability of assigning the correct label to both training examples and samples from G. simultaneously train G to minimize that probability.

Global Optimality of $p_g = p_{data}$

The optimal D for any given G

$$D_G^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

Proof: For given G, D is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G, D) &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log(D(\mathbf{x})) dx + \int_{\mathbf{z}} p_z(\mathbf{z}) \log(1 - D(G(\mathbf{z}))) dz \\ &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) dx \end{aligned}$$

For any $(a, b) \in R^2 \backslash \{0, 0\}$, the function $y \rightarrow a\log y + b\log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{\mathbf{a}}{\mathbf{a}+\mathbf{b}}$.

Global Optimality of $p_g = p_{data}$

$$
\begin{aligned}
C(G) &= V(G, D^*) \\
&= \mathrm{E}_{\mathbf{x} \sim p_{data}}[\log(D^*(\mathbf{x}))] + \mathrm{E}_{\mathbf{z} \sim p_z}[\log(1 - D^*(G(\mathbf{z})))] \\
&= \mathrm{E}_{\mathbf{x} \sim p_{data}}[\log(D^*(\mathbf{x}))] + \mathrm{E}_{\mathbf{x} \sim p_g}[\log(1 - D^*(\mathbf{x}))] \\
&= \mathrm{E}_{\mathbf{x} \sim p_{data}}\left[\log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right] + \mathrm{E}_{\mathbf{x} \sim p_g}\left[\log \frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right] \\
&= -\log 4 + KL(p_{data} || \frac{p_{data} + p_g}{2}) + KL(p_g || \frac{p_{data} + p_g}{2})
\end{aligned}
$$

- If G and D have enough capacity, and at each step of training, D is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion $C(G)$.
- Therefore, p_g will **converges to** p_{data}. (Jensen-Shannon Divergence (JSD))
- Finally, the global minimum of the virtual training criterion $C(G)$ is achieved if and only if $\mathbf{p_g} = \mathbf{p_{data}}$.

Training Procedure

- Both G and D are neural networks.
- Use SGD-like algorithm of choice (Adam) on two minibatches simultaneously:
  - A minibatch of training examples
  - A minibatch of generated samples
  - By iterative optimizing D and G by SGD-like algorithms, we can approximately solve the minimax objective.
    - The objective for training D is

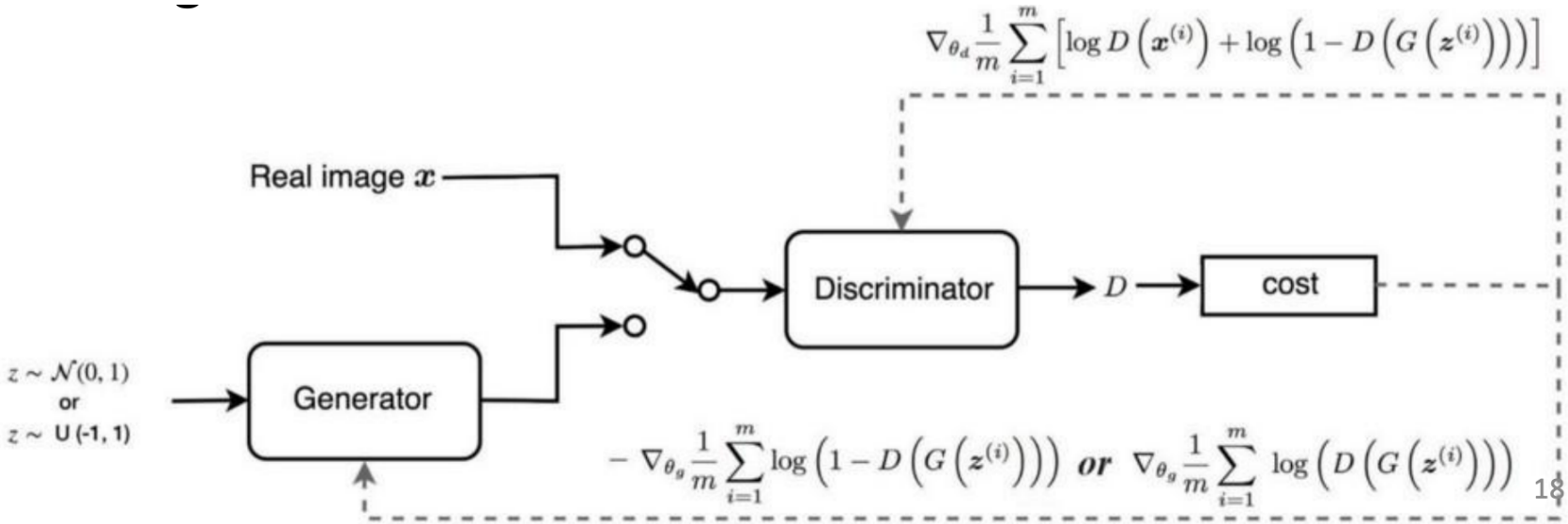
$$\max_D \mathrm{E}_{\mathbf{x}\sim P_{data}}[logD(\mathbf{x})] + \mathrm{E}_{\tilde{\mathbf{x}}\sim P_g}[log(1 - D(\tilde{\mathbf{x}}))]$$

    - The objective for training G is

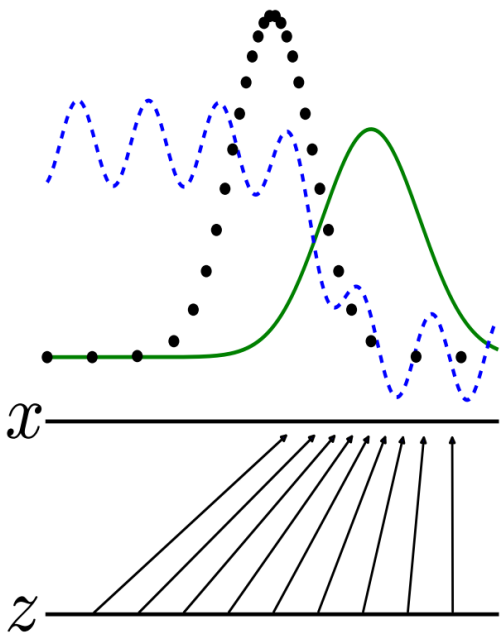
$$\min_G \mathrm{E}_{\tilde{\mathbf{x}}\sim P_g}[log(1 - D(\tilde{\mathbf{x}}))]$$

Training Procedure

- The generator and the discriminator are learned jointly by the alternating gradient descent.
  - Fix the generator's parameters and perform a single iteration of gradient descent on the discriminator using the real and the generated images.
  - Fix the discriminator and train the generator for another single iteration.



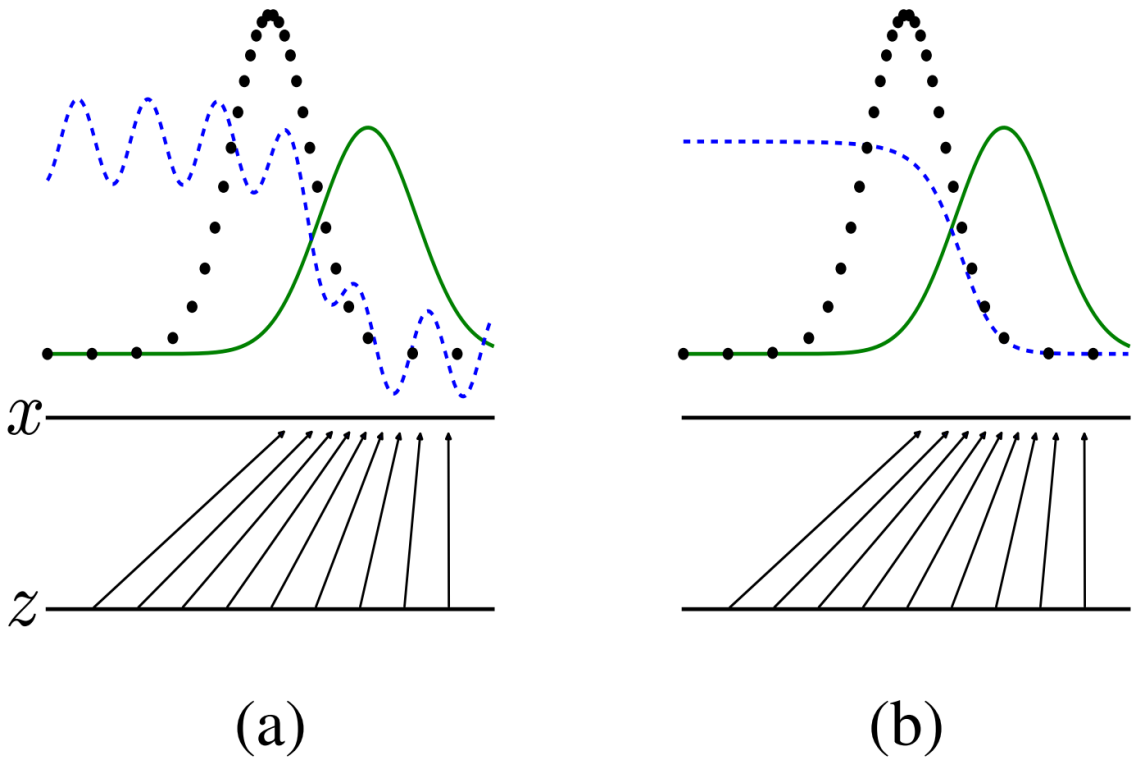
Training Model



(a)

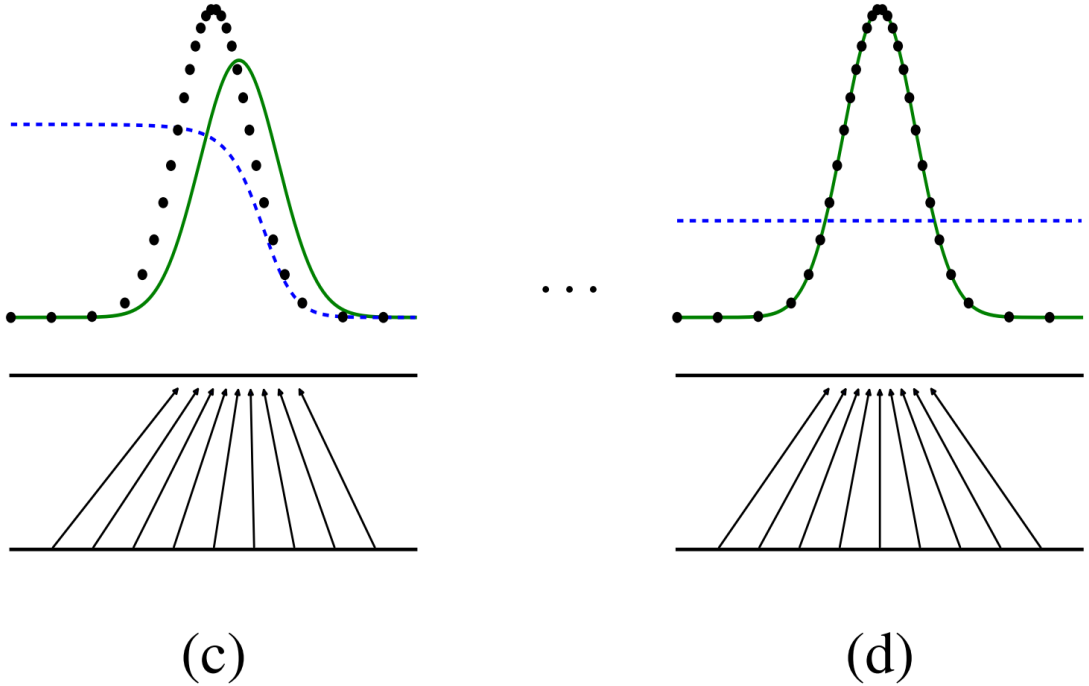
- GANs are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\mathbf{x}}$ from those of the generative distribution $p_g(G)$ (green, solid line).
- The lower horizontal line is the domain from which $\mathbf{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\mathbf{x}$.
- The upward arrows show how the mapping $\mathbf{x} = G(\mathbf{z})$ imposes the non-uniform distribution p_g on transformed samples.

Training Model



- (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier.
- (b) In the step of optimizing D, it is trained to discriminate samples from data, converging to $\frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}$.

Training model



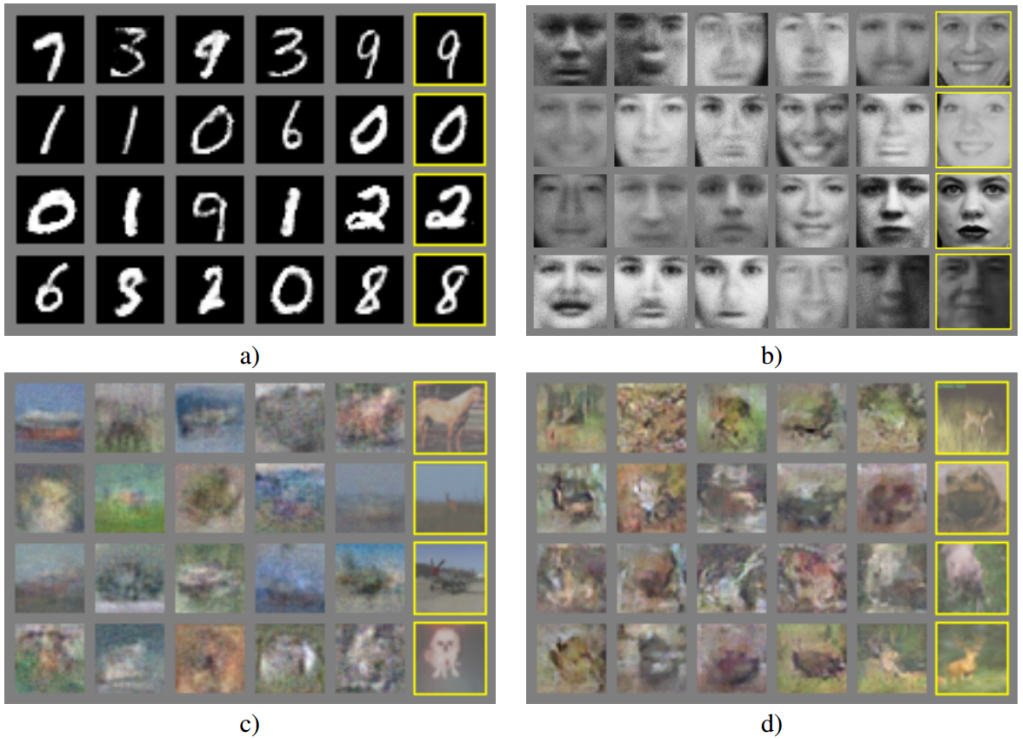
- (c) After an update to G, gradient of D has guided $G(\mathbf{z})$ to flow to regions that are more likely to be classified as data.
- (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $\mathbf{p_g} = \mathbf{p_{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\mathbf{x}) = \frac{1}{2}$

Experiments

Estimate probability of the test set data under p_g by fitting a Gaussian Parzen window to the samples generated with G and reporting the log-likelihood under this distribution.

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [5]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Experiments



Visualization of samples from the model. These images show actual samples from the model distributions and these samples are uncorrelated because the sampling process does not depend on Markov chain.
a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and deconvolutional generator)

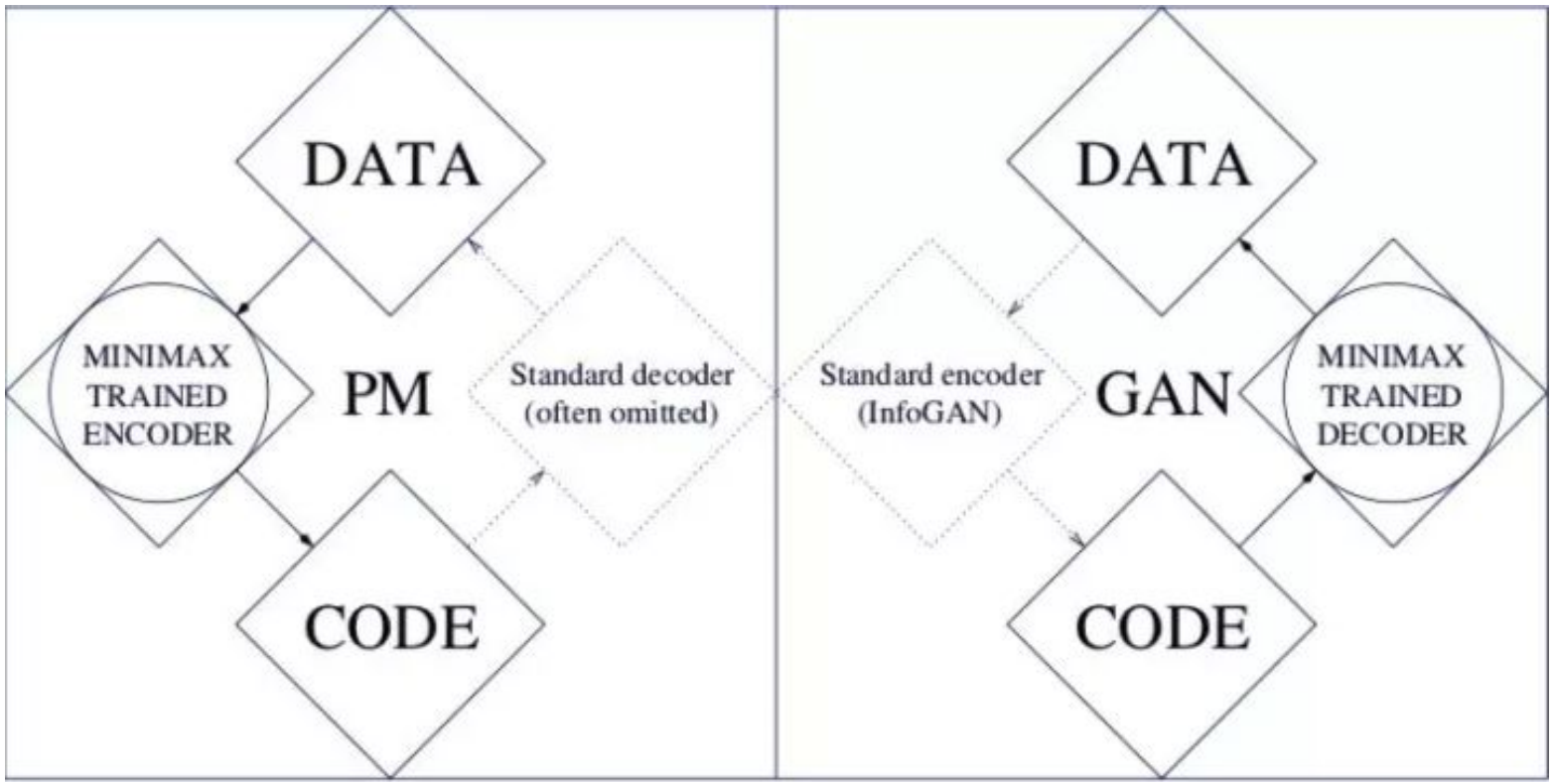
GAN vs. PM vs. AC1900

VS.

- Jurgen's recent single author paper: Unsupervised Minimax: Adversarial Curiosity, Generative Adversarial Networks, and Predictability Minimization
- David Ha (Google Brain) and Jurgen recently published a paper "World Models"

GAN vs. PM vs. AC1900

- GAN vs. PM: Similar but also different.



Non-Saturating Game

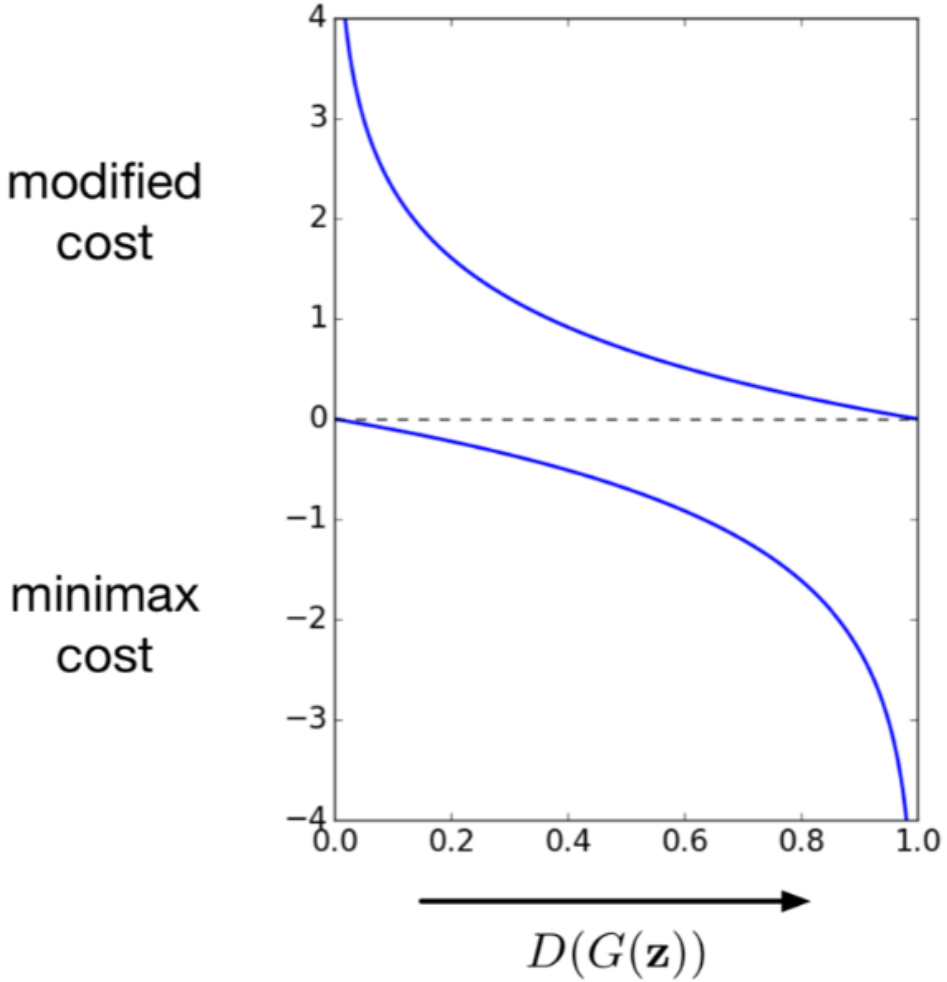
The loss function of D and G are updated as follow £°

$$J^{(D)} = -\frac{1}{2}\mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] - \frac{1}{2}\mathrm{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

$$J^{(G)} = -\frac{1}{2}\mathrm{E}_{\mathbf{z}}[\log(D(G(\mathbf{z})))]$$

- In original minimax game, the generator's gradient vanishes when the discriminator successfully rejects generator samples with high confidence (meaning $D(\tilde{\mathbf{x}})$ is 0).
- G maximizes the log-probability of the D being mistaken. G can still learn even when D successfully rejects all generator samples.

Vanishing Gradient



- The curve of loss with respect to $D(G(\boldsymbol{z}))$. The minimax objective's gradients for G converge to 0 when the discriminator is nearly perfect($D(G(\boldsymbol{z}))$ is 0), which makes it hard to learn the generator.

Maximum Likelihood Game

The loss function of D and G are updated as follow £°

$$J^{(D)} = -\frac{1}{2}\mathrm{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] - \frac{1}{2}\mathrm{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

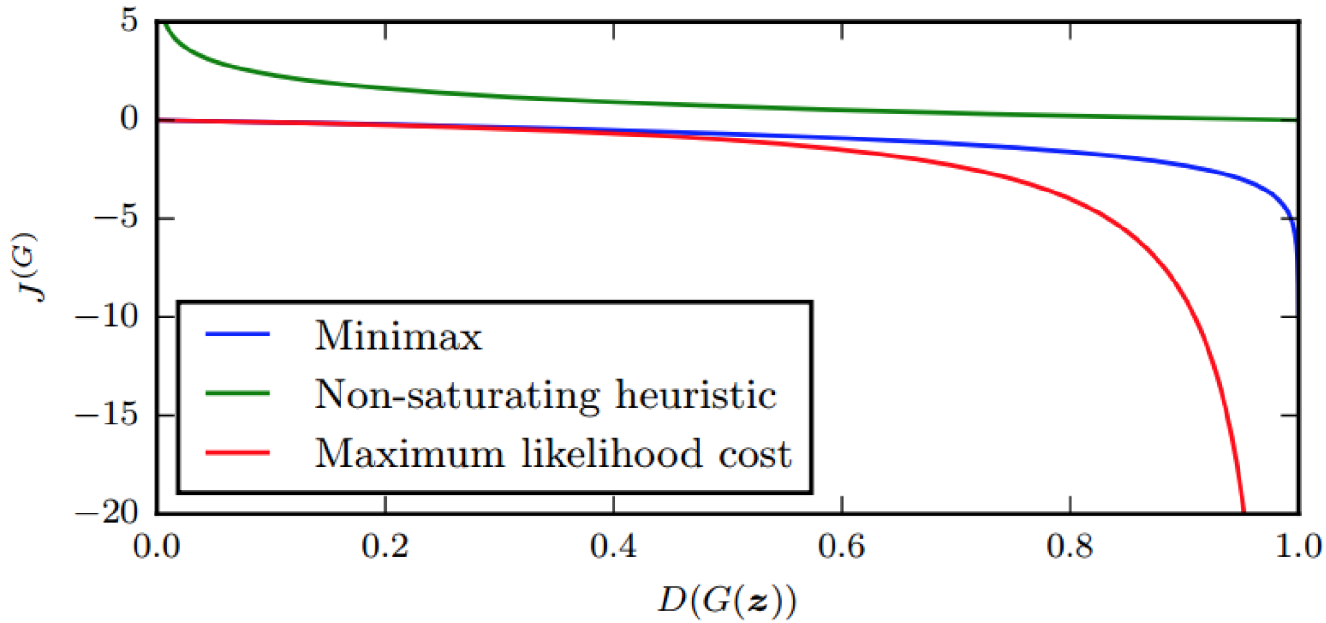
$$J^{(G)} = -\frac{1}{2}\mathrm{E}_{\mathbf{z}} \exp(\sigma^{-1}(D(G(\mathbf{z}))))$$

- Equivalent to minimizing the KL divergence between the data generating distribution and the model[3].

$$\theta^* = \arg\min_{\theta} D_{KL}(p_{data}(x)||p_{model}(x;\theta))$$

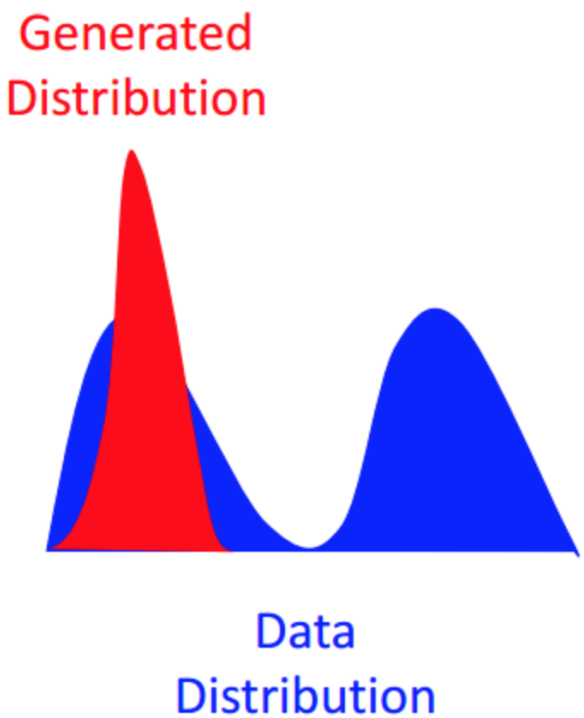
[3] Goodfellow, I. J. (2014). On distinguishability criteria for estimating generative models.

Comparison

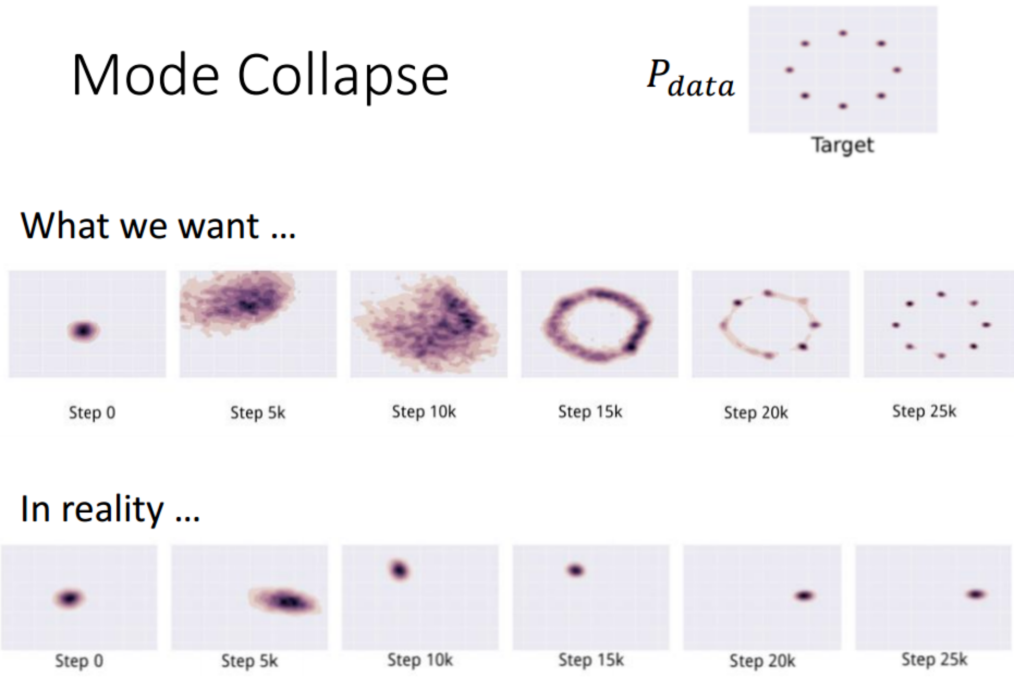

5
0
−5
−10
−15
−20
$J(G)$
0.0
0.2
0.4
0.6
0.8
1.0
$D(G(\boldsymbol{z}))$
Minimax
Non-saturating heuristic
Maximum likelihood cost

Mode Collapse

- Mode collapse is an empirically observed phenomenon that distribution of $G(\boldsymbol{z})$ is likely to fail to capture all modes of P_{data}.
  - e.g. GAN can only generates one subtype of images



Mode Collapse



- The figures at the bottom line are results of GAN. During all the six training steps, GAN fails to capture all 9 modes. In fact, GAN only recover one mode. The mode captured drifts all the time.

GAN Variants

- UnrolledGAN: stabilize GANs by defining the generator objective w.r.t. an unrolled optimization of the discriminator;
- InfoGAN: learn disentangled representations in a completely unsupervised manner;
- CGAN: feeding the data, y, to control both the generator and discriminator;
- DCGAN: Deep Convolutional Generative Adversarial Network;
- wGAN: a new algorithm to train GAN using Wasserstein distance;
- LSGAN: uses least squares loss function for the discriminator;
- BEGAN: a new equilibrium enforcing method paired with a loss derived from the Wasserstein distance;
- Progressive-Growing GAN: a stable approach to training GAN models to generate large high-quality images.

Unrolled GAN

- Unrolled GAN prevents mode collapse by back-propagating through a set of k updates of discriminator D to update generator G parameters once.
- Thus, G's chance of overfitting to a specific D reduces.
- Though k updates of D are used for the update of G, only the first update of D is kept for next round's training. This is to reduce D's chance of overfitting.

Unrolled GAN

Original GAN

- Updates of D and G:

$$\theta_D = \theta_D + \eta \frac{\partial L(\theta_D, \theta_G)}{\theta_D}$$

$$\theta_G = \theta_G - \eta \frac{\partial L(\theta_D, \theta_G)}{\theta_G}$$

Unrolled GAN

- Update of D:

$$\theta_D = \theta_D + \eta \frac{\partial L(\theta_D, \theta_G)}{\theta_D}$$

- Update of G:

$$\theta_D^0 = \theta_D,$$

$$\theta_D^{k+1} = \theta_D^k + \eta^k \frac{\partial L(\theta_D^k, \theta_G)}{\theta_D^k}$$

$$\theta_G = \theta_G - \eta \frac{\partial L_K(\theta_D, \theta_G)}{\theta_G}$$

$$(L_K(\theta_D, \theta_G) := L(\theta_D^K, \theta_G))$$

Wasserstein GAN[4]

In the analysis of GAN, Optimization of G is equivalent to minimize JS divergence of $p_{data}(x)$ and $p_g(x)$.

$$\begin{aligned}
&\mathrm{E}_{\mathbf{x}\sim p_{data}}\left[\log\frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}\right]+\mathrm{E}_{\mathbf{x}\sim p_g}\left[\log\frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}\right]\\
&=-\log 4+KL(p_{data}||\frac{p_{data}+p_g}{2})+KL(p_g||\frac{p_{data}+p_g}{2})\\
&=-\log 4+2\cdot JS(p_{data}||p_g)
\end{aligned}$$

But the optimization is sometimes impossible **by gradient descent**, especially when p_{data} is a manifold in high dimension space. In these cases, gradients of G is 0 most of the time.

[4] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan.

Wasserstein GAN

1-Wasserstein Distance(Earth-Mover Distance):

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

Suppose Z is the uniform distribution over $[0, 1]$.
$p_{data} = (0, Z) \in \mathbb{R}^2$. Our model is $p_g = (\theta, Z)$. Now what will happen?

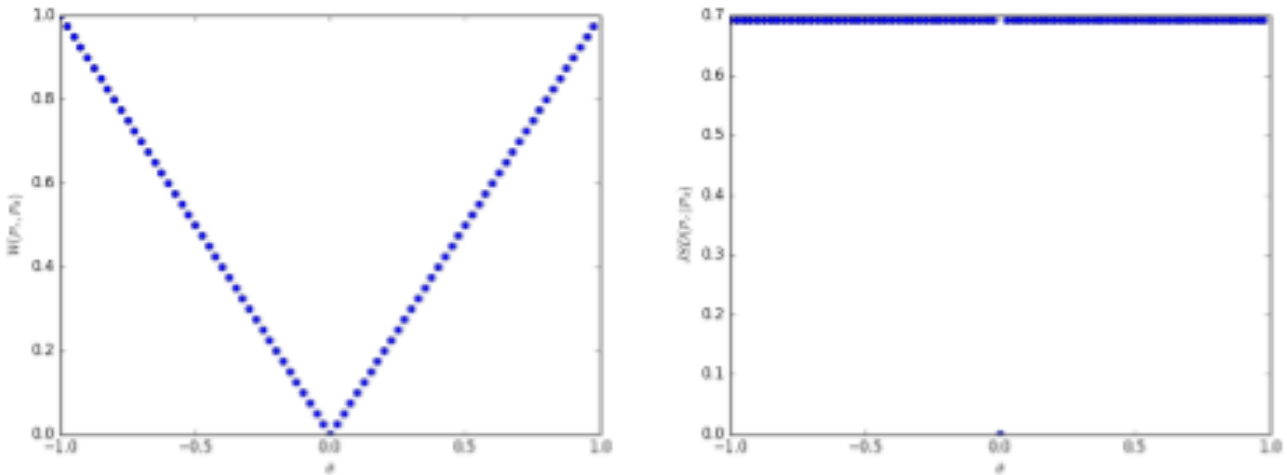


Figure 1: These plots show $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ when ρ is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

Wasserstein GAN - Solution

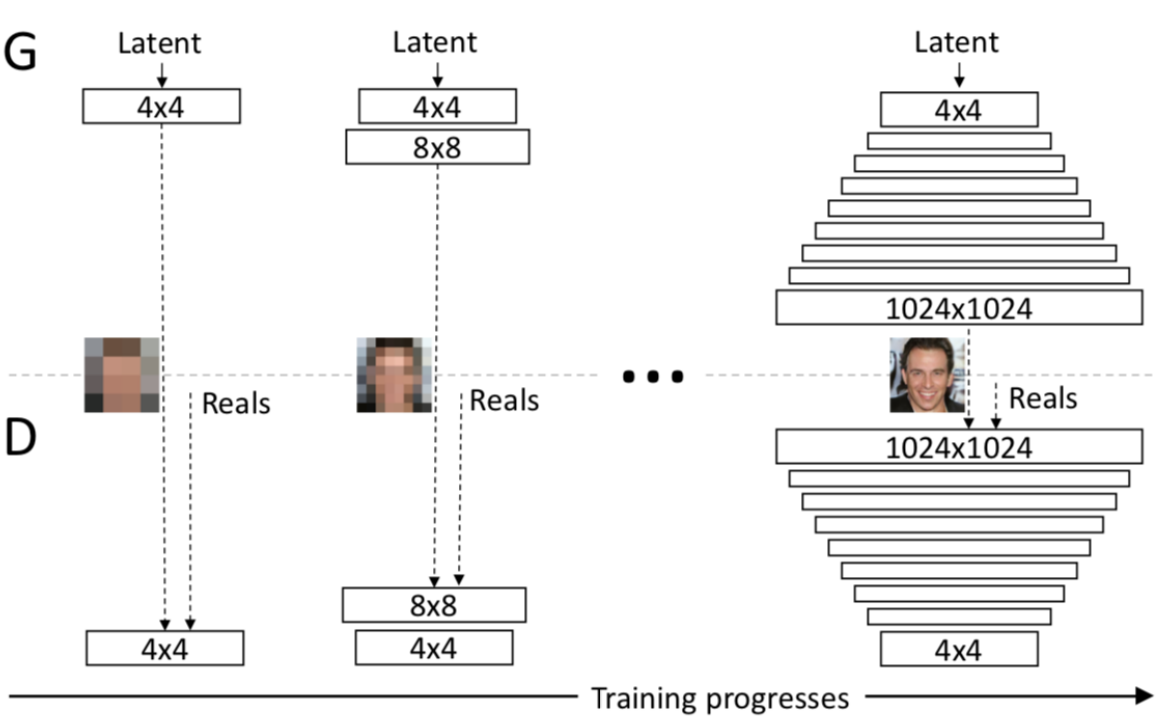
Kantorovich-Rubinstein duality :

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_g}[f(x)]$$

$||f||_L \leq K$ means K-Lipschitz condition. Note that if we replace 1 for K, then we end up with $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$.
Parameterize f by a neural network and boxing the the updates to guarantee compactness.

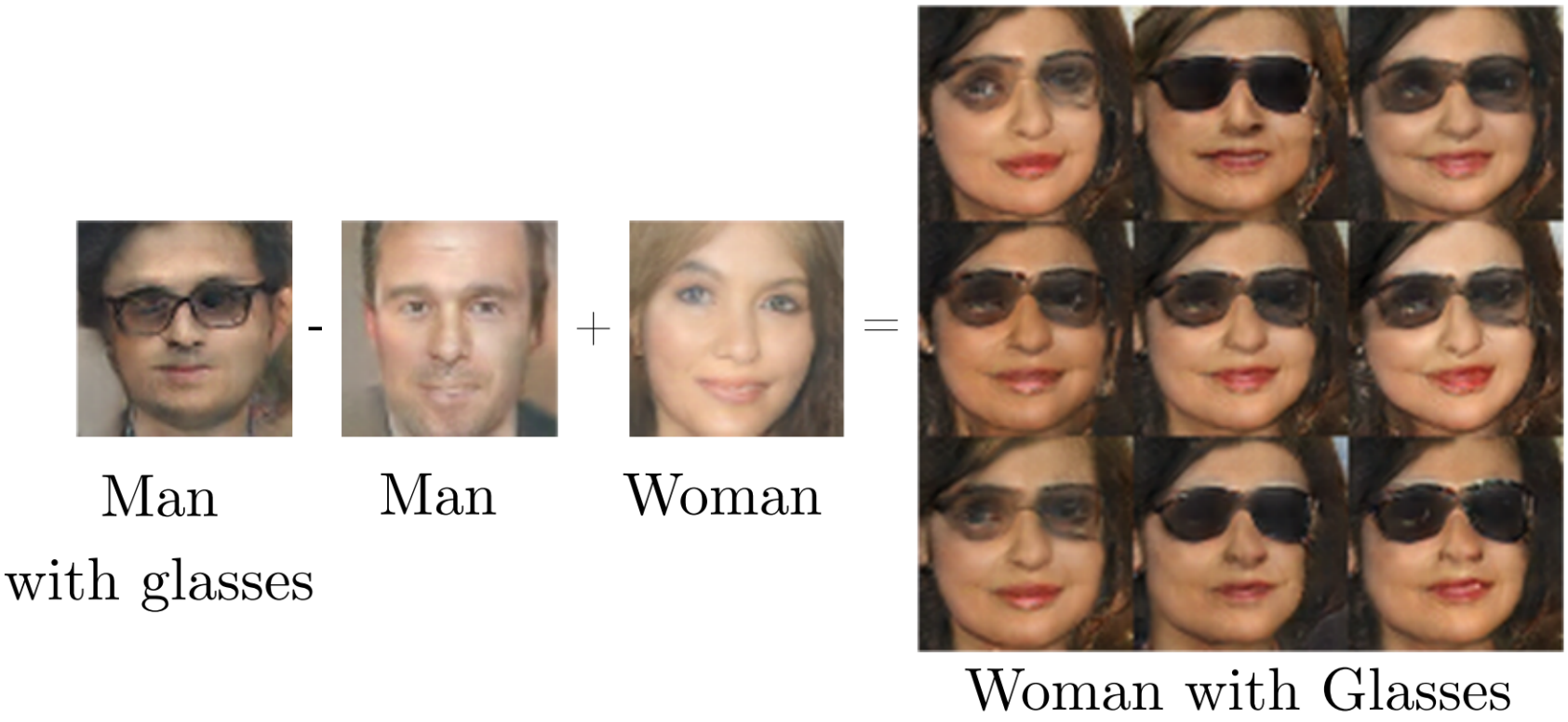
- Gradient Clipping
- Gradient Penalty

Progressive Growing GAN[5]



[5] Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation.

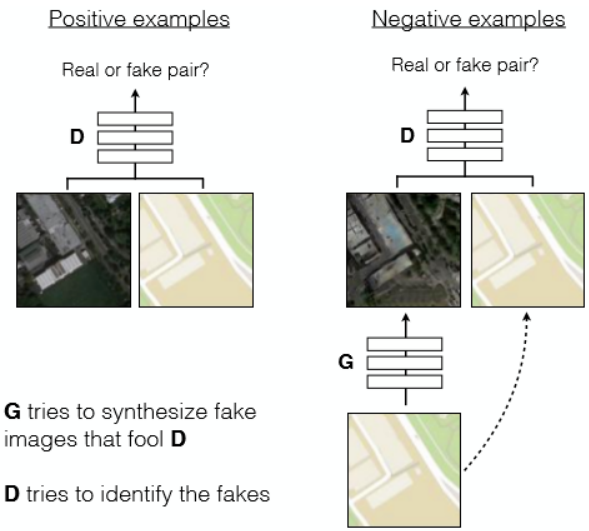
Applications: Vector Space Arithmetic



Arithmetic on noise vector Z. (DCGAN, 2016).
For stability, left images were generated from average value of three noise vectors which could generate specific kinds of images.

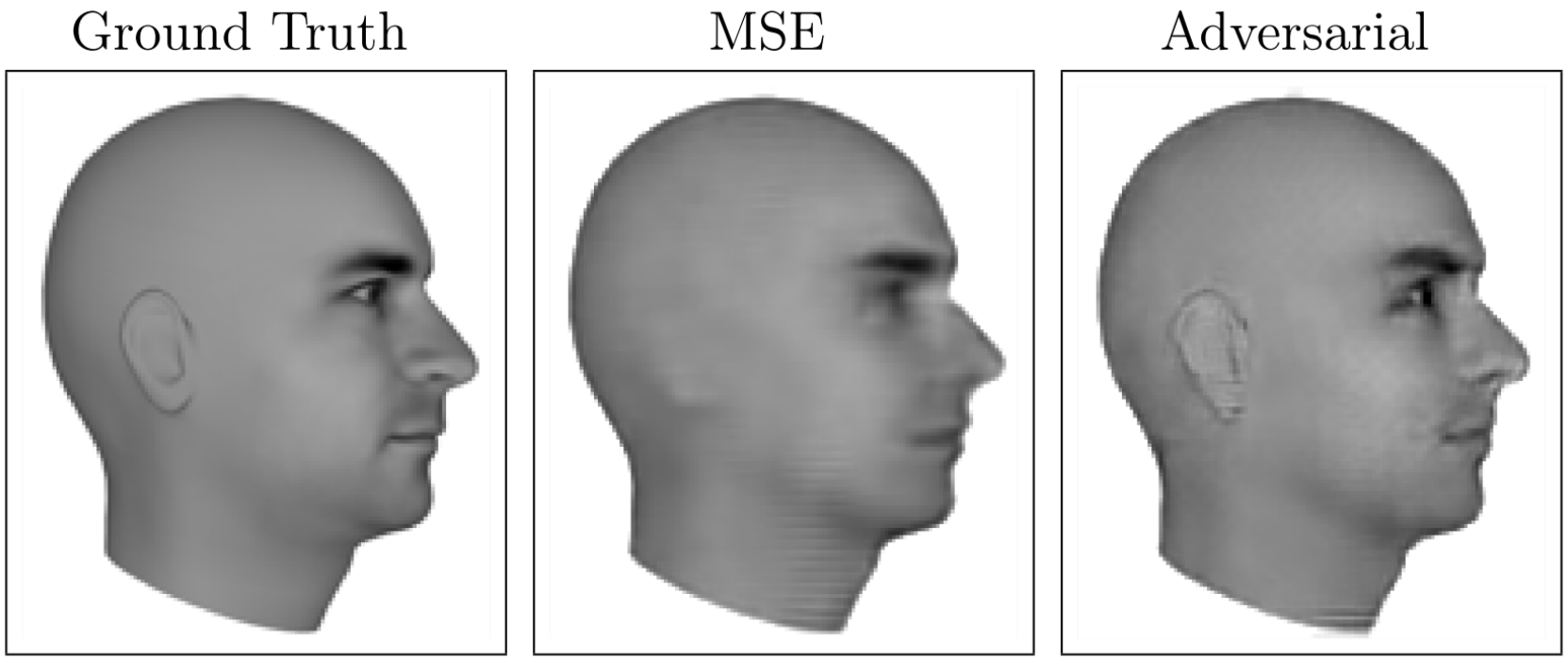
Applications: Image to Image Translation

Applications: Image to Image Translation



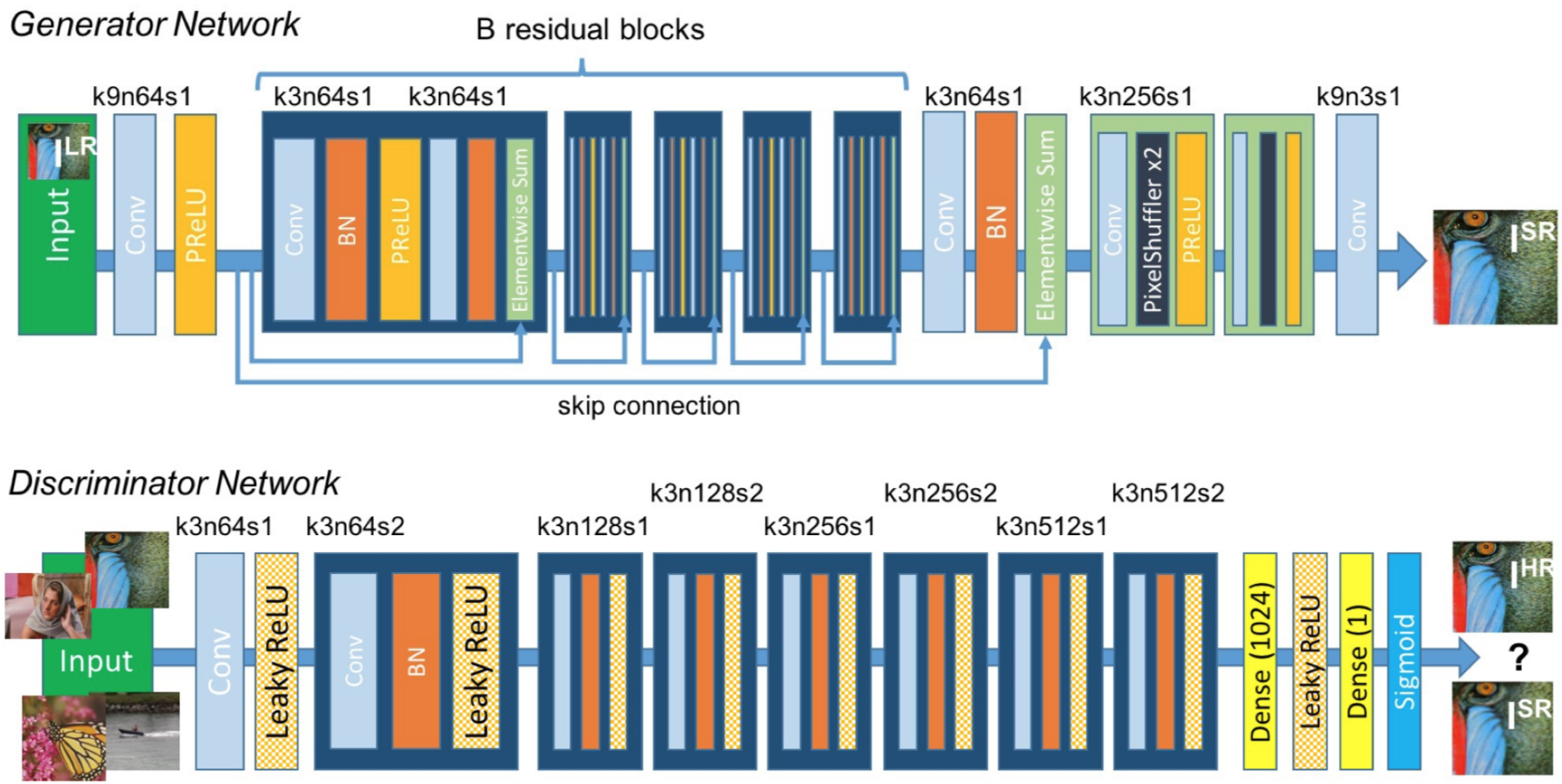
- **Conditional GAN**
- D takes the pair of original image x and translated image y as input. G takes x and noise z as input and tries to fake a translated image.
- $L_{L1}(G) = E_{x,y\sim p_{data}(x,y),z\sim p_z(z)}\left[||y - G(x,z)||_1\right]$
- $L = L_{GAN}(G,D) + \lambda L_{L1}(G)$
- **Defect:** z is useless for G, resulting in no stochastic outputs.

Applications: Next Video Frame Prediction

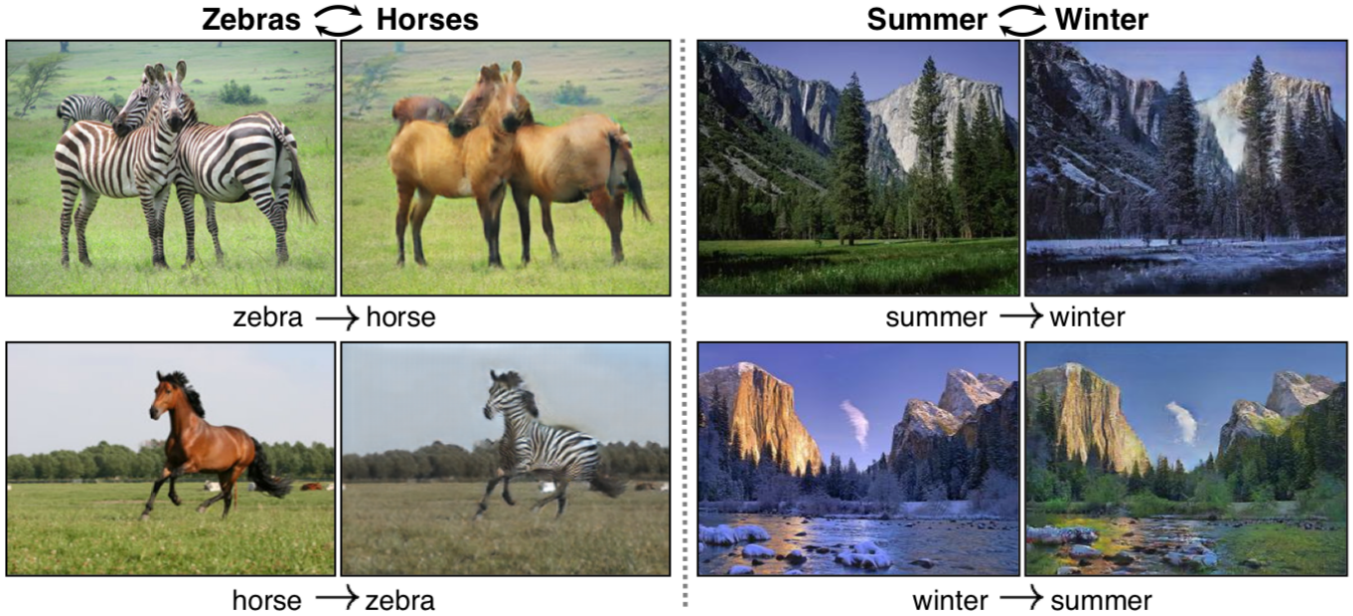


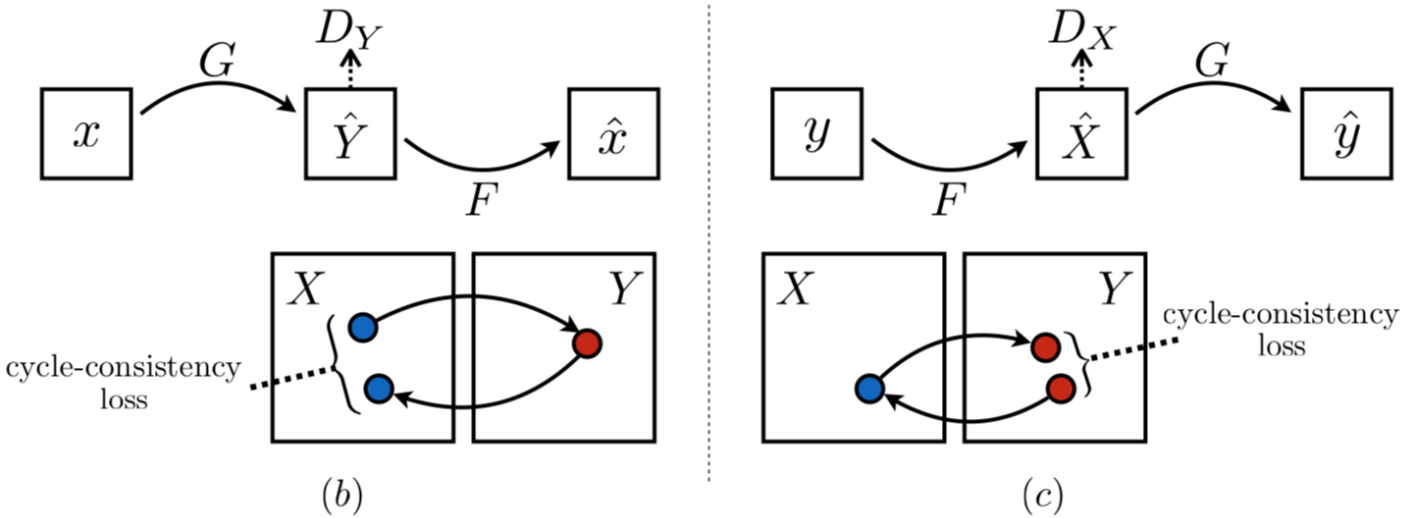
Applications: Super Resolution

A special conditional GAN.



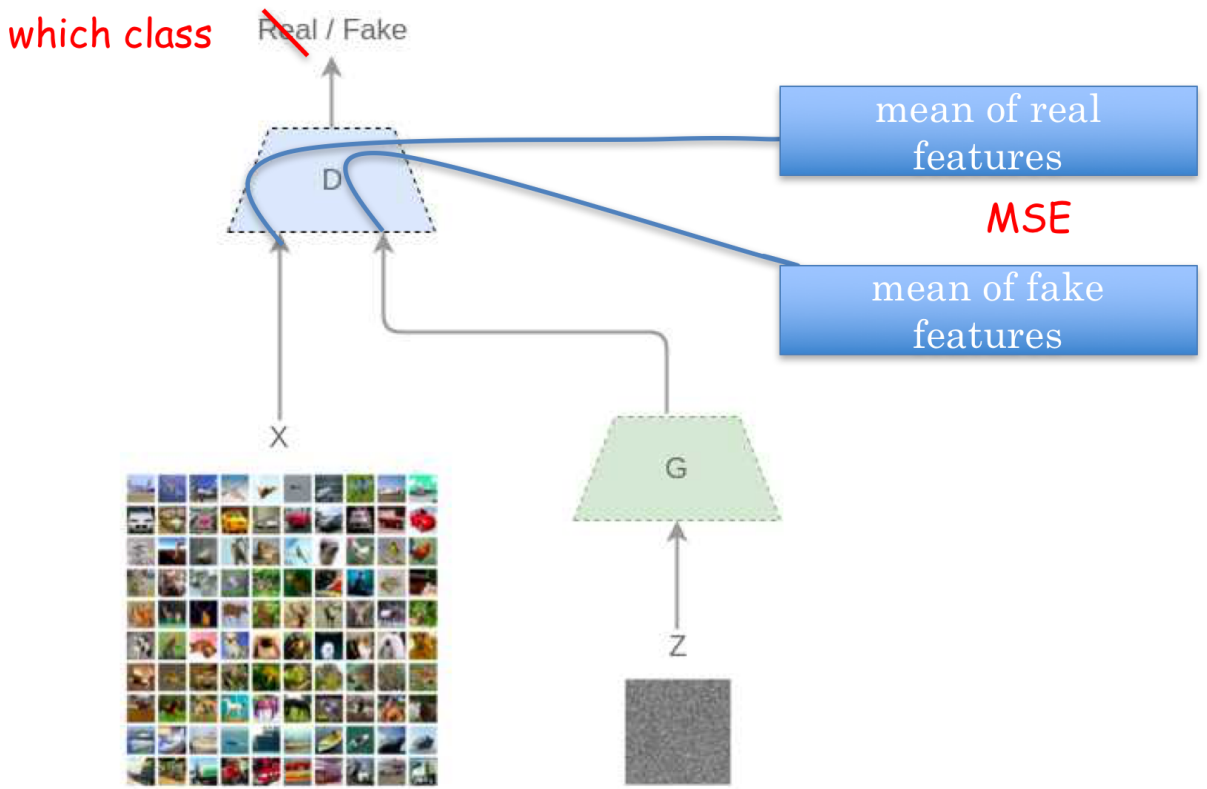
Applications: Unpaired Image-to-Image Translation



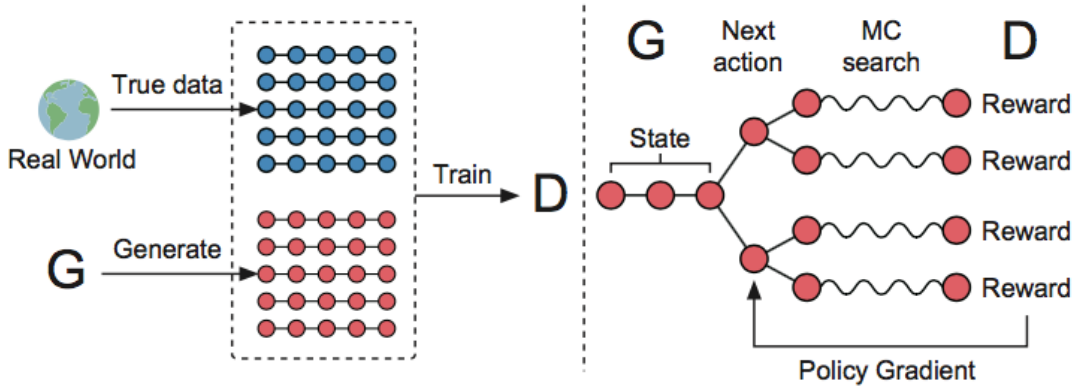


Application: Semi-supervised Learning

Just replace Discriminator with a classifier and use *feature matching* to train G.



Applications: Generation of Discrete Sequential Data



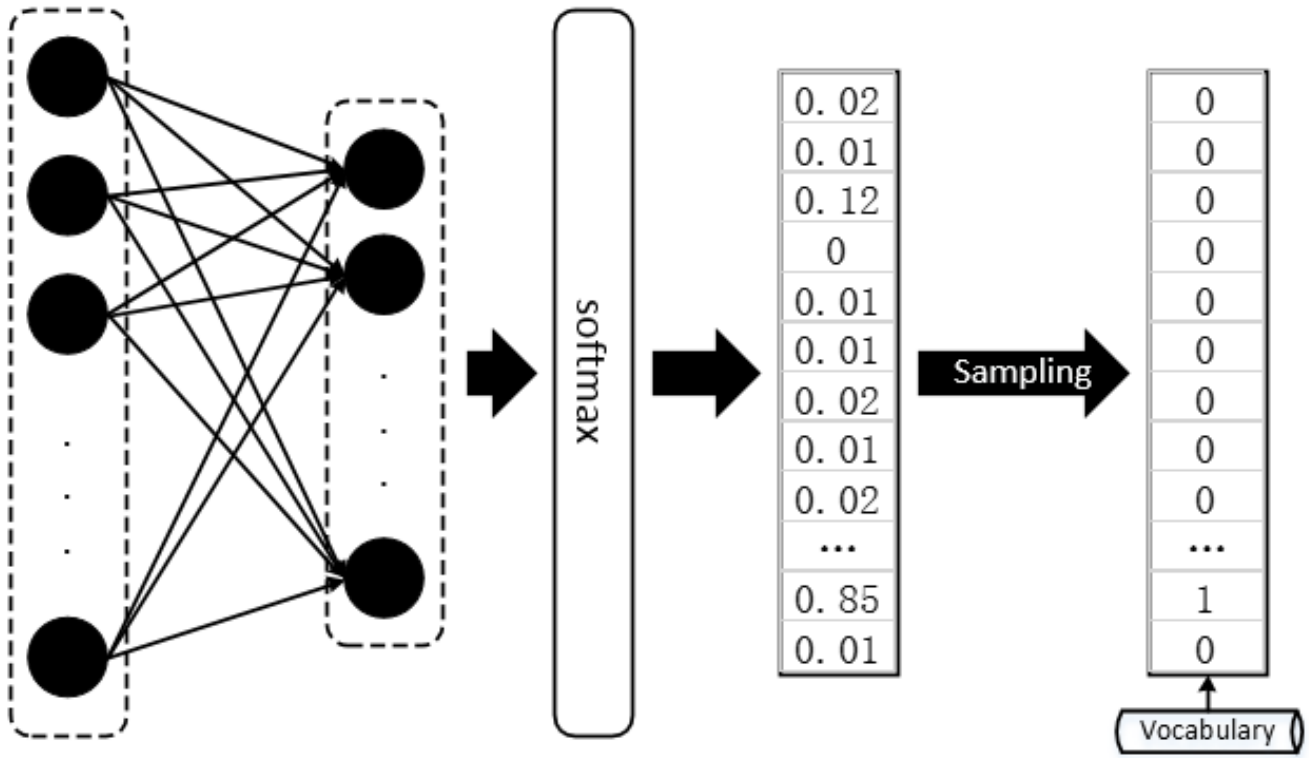
Let $\nabla J(\theta)$ be the policy gradient of G. $Q_{D_\phi}^{G_\theta}$ is the reward function:

$$\nabla J(\theta) = \sum_{y \in \mathcal{Y}} (\nabla G_\theta(y_t | Y_{1:t-1})) Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)$$

$$Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) = \begin{cases} D_\phi(Y_{1:t}), if \quad t = T \\ \frac{1}{N} \sum_{n=1}^{N} D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC^{G_\theta}(Y_{1:t}; N), if \quad t < N \end{cases}$$

More Discussions: Why not NLP?

- GAN is working on the continuous, but not discrete data
- Solution: wGAN is one way; and RL is another way



Conclusion

- GANs are generative models that use supervised learning to approximate an intractable cost function or a density ratio.
- GANs can simulate many cost functions, including the one used for maximum likelihood.
- Adversarial training can be useful for people as well as machine learning models
- Applications include learning from very few labeled examples, interactive artwork generation, and differential privacy.

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn