GANs

Oualid Missaoui

Agenda

- Introduction
- GAN
  - Architecture
  - Theoretical Guarantees
- Conditional GAN
- InfoGAN
- GAN Problems
- WGAN
  - Weight Clipping
  - Gradient penalty
- GANs for time series
- References

Introduction

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Source: Ian Goodfellow et al, 2014, Generative Adversarial Nets

Introduction: Mathematical Setup of Deep Generative Models (DGM)

Unknown true distribution

We cannot draw new samples from it directly

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

$$\{\mathbf{x}^{(i)}\}_{i=1}^{N} \sim p_{\text{data}}$$

Model distribution

$p_{\phi}(\mathbf{x})$ ϕ The network's trainable parameters

Generative Model

Find ϕ^* that satisfies $p_{\phi^*}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$

Capability of DGM.

Once a proxy of the data distribution, $p_{\phi}(\mathbf{x})$, is available, we can:

- generate an arbitrary number of new data points using sampling methods such as Monte Carlo sampling from $p_{\phi}(\mathbf{x})$.
- Additionally, we can compute the probability (or likelihood) of any given data sample $\mathbf{x}'$ by evaluating $p_{\phi}(\mathbf{x}')$.

Introduction: Illustration of DGM

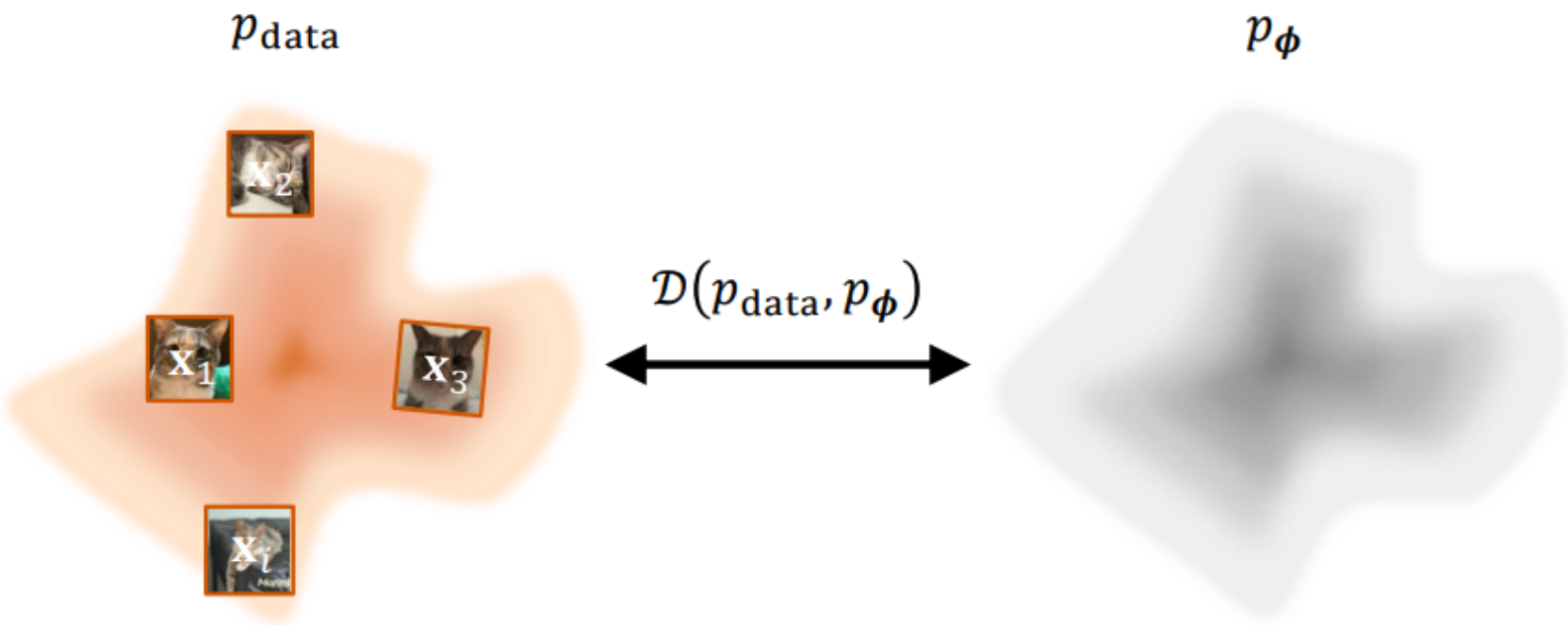
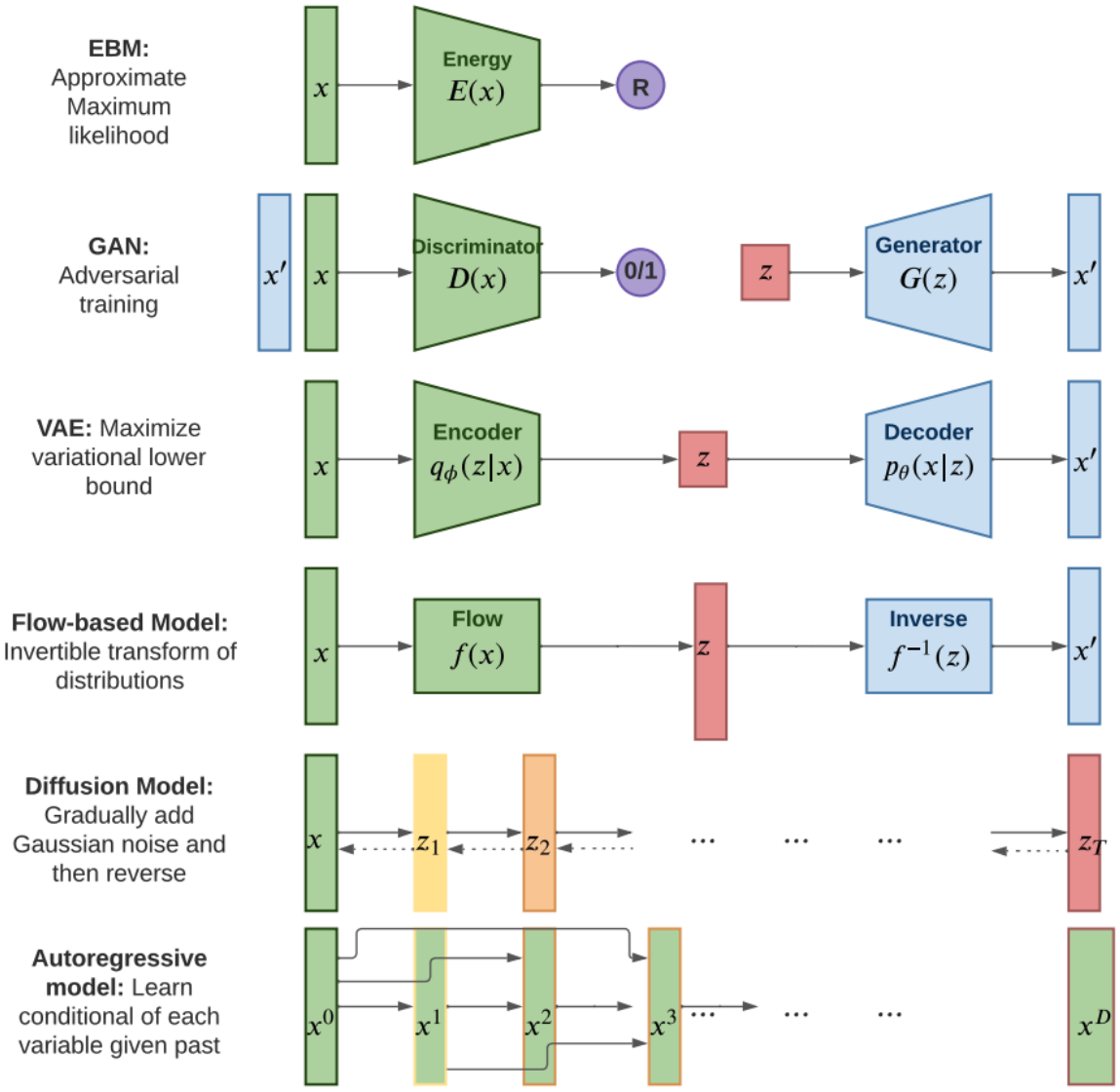


Figure 1.1: Illustration of the target in DGM. Training a DGM is essentially minimizing the discrepancy between the model distribution $p_{\boldsymbol{\phi}}$ and the unknown data distribution p_{data}. Since p_{data} is not directly accessible, this discrepancy must be estimated efficiently using a finite set of independent and identically distributed (i.i.d.) samples, $\mathbf{x}_i$, drawn from it.

Introduction

There are two approaches to fitting $P(x, y)$ to data $\mathcal{D}$:

- Generative: $p(x, y) = p(x|y)p(y)$
  - prior information about the structure of the data
  - finding a suitable generative data model is a difficult task
- Discriminative: $p(x, y) = p(y|x)p(x)$
  - directly addresses finding an accurate classifier based on modeling the decision boundary
  - usually trained as "block-box" classifier with little prior knowledge built.



Source: Kevin Murphy, 2023, Probabilistic Machine learning: Advanced topics (part IV: Generation)

Introduction

Table 1.1: Comparison of Explicit and Implicit Generative Models

	Explicit		Implicit
	Exact Likelihood	**Approx. Likelihood**	
Likelihood	Tractable	Bound/Approx.	Not Directly Modeled/ Intractable
Objective	MLE	ELBO	Adversarial
Examples	NFs, ARs	VAEs, DMs	GANs

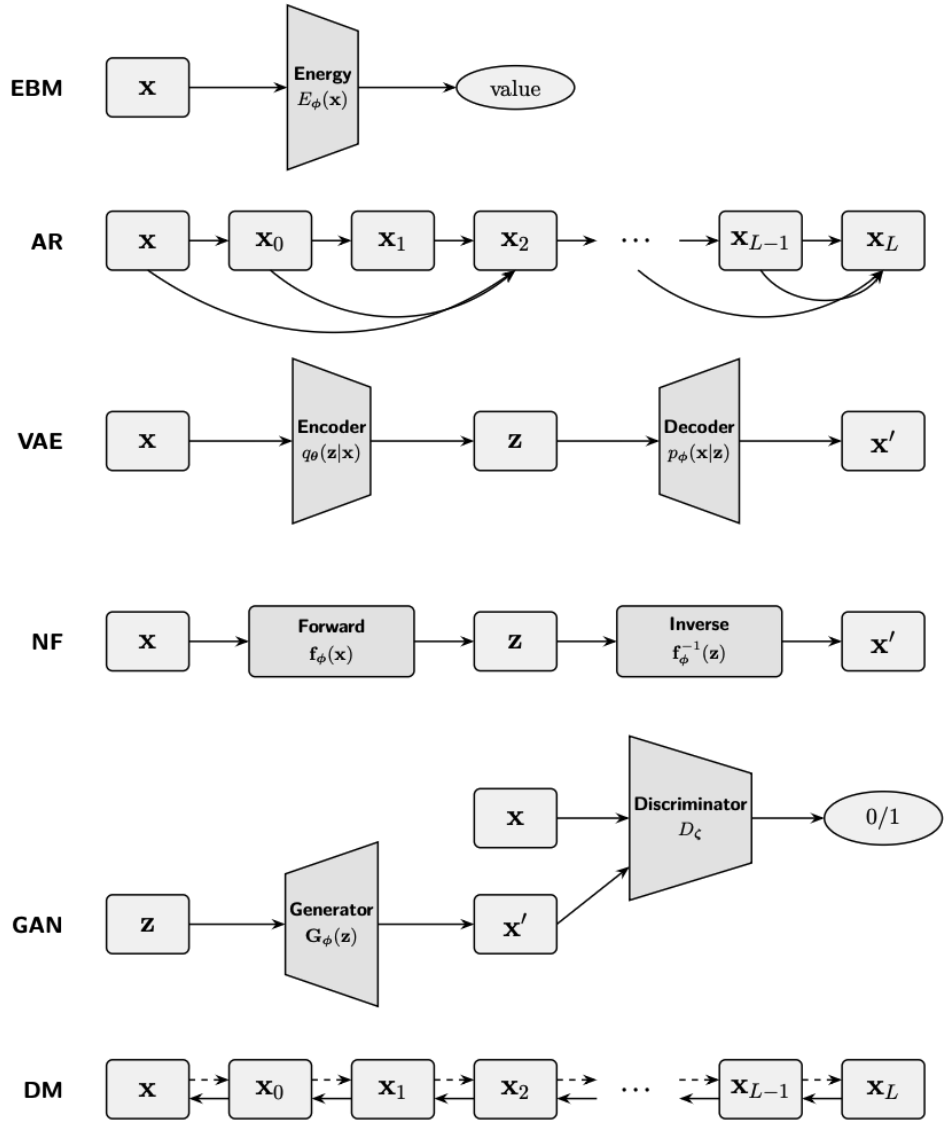


Figure 1.2: Computation graphs of prominent deep generative models. Top to bottom: **EBM** maps an input $\mathbf{x}$ to a scalar energy; **AR** generates a sequence $\{\mathbf{x}_\ell\}$ left to right with causal dependencies; **VAE** encodes $\mathbf{x}$ to a latent $\mathbf{z}$ and decodes to a reconstruction $\mathbf{x}'$; **NF** applies an invertible map $\mathbf{f}_\phi$ between $\mathbf{x}$ and $\mathbf{z}$ and uses $\mathbf{f}_\phi^{-1}$ to produce $\mathbf{x}'$; **GAN** transforms noise $\mathbf{z}$ to a sample $\mathbf{x}'$ that is judged against real $\mathbf{x}$ by a discriminator D_ζ; **DM** iteratively refines a noisy sample through a multi-step denoising chain $\{\mathbf{x}_\ell\}$. Boxes denote variables, trapezoids are learnable networks, ovals are scalars; arrows indicate computation flow.

Chieh-Hsin Lai et al (2025), The Principles of Diffusion Models

Classification

- Density: does the model support pointwise evaluation of the probability density function $p(\boldsymbol{x})$, and if so, is this fast or slow, exact, approximate or a bound, etc? For **implicit models**, such as GANs, there is no well-defined density $p(\boldsymbol{x})$. For other models, we can only compute a lower bound on the density (VAEs), or an approximation to the density (EBMs, UPGMs).
- Sampling: does the model support generating new samples, $\boldsymbol{x} \sim p(\boldsymbol{x})$, and if so, is this fast or slow, exact or approximate? Directed PGMs, VAEs, and GANs all support fast sampling. However, undirected PGMs, EBMs, ARM, diffusion, and flows are slow for sampling.
- Training: what kind of method is used for parameter estimation? For some models (such as AR, flows and directed PGMs), we can perform exact maximum likelihood estimation (MLE), although

  the objective is usually non-convex, so we can only reach a local optimum. For other models, we cannot tractably compute the likelihood. In the case of VAEs, we maximize a lower bound on the likelihood; in the case of EBMs and UGMs, we maximize an approximation to the likelihood. For GANs we have to use min-max training, which can be unstable, and there is no clear objective function to monitor.
- Latents: does the model use a latent vector $\boldsymbol{z}$ to generate $\boldsymbol{x}$ or not, and if so, is it the same size as $\boldsymbol{x}$ or is it a potentially compressed representation? For example, ARMs do not use latents; flows and diffusion use latents, but they are not compressed.[1] Graphical models, including EBMs, may or may not use latents.
- Architecture: what kind of neural network should we use, and are there restrictions? For flows, we are restricted to using invertible neural networks where each layer has a tractable Jacobian. For EBMs, we can use any model we like. The other models have different restrictions.

Model	Chapter	Density	Sampling	Training	Latents	Architecture
PGM-D	Section 4.2	Exact, fast	Fast	MLE	Optional	Sparse DAG
PGM-U	Section 4.3	Approx, slow	Slow	MLE-A	Optional	Sparse graph
VAE	Chapter 21	LB, fast	Fast	MLE-LB	$\mathbb{R}^L$	Encoder-Decoder
ARM	Chapter 22	Exact, fast	Slow	MLE	None	Sequential
Flows	Chapter 23	Exact, slow/fast	Slow	MLE	$\mathbb{R}^D$	Invertible
EBM	Chapter 24	Approx, slow	Slow	MLE-A	Optional	Discriminative
Diffusion	Chapter 25	LB	Slow	MLE-LB	$\mathbb{R}^D$	Encoder-Decoder
GAN	Chapter 26	NA	Fast	Min-max	$\mathbb{R}^L$	Generator-Discriminator

Table 20.1: Characteristics of common kinds of generative model. Here D is the dimensionality of the observed $\boldsymbol{x}$, and L is the dimensionality of the latent $\boldsymbol{z}$, if present. (We usually assume $L \ll D$, although overcomplete representations can have $L \gg D$.) Abbreviations: Approx = approximate, ARM = autoregressive model, EBM = energy based model, GAN = generative adversarial network, MLE = maximum likelihood estimation, MLE-A = MLE (approximate), MLE-LB = MLE (lower bound), NA = not available, PGM = probabilistic graphical model, PGM-D = directed PGM, PGM-U = undirected PGM, VAE = variational autoencoder.

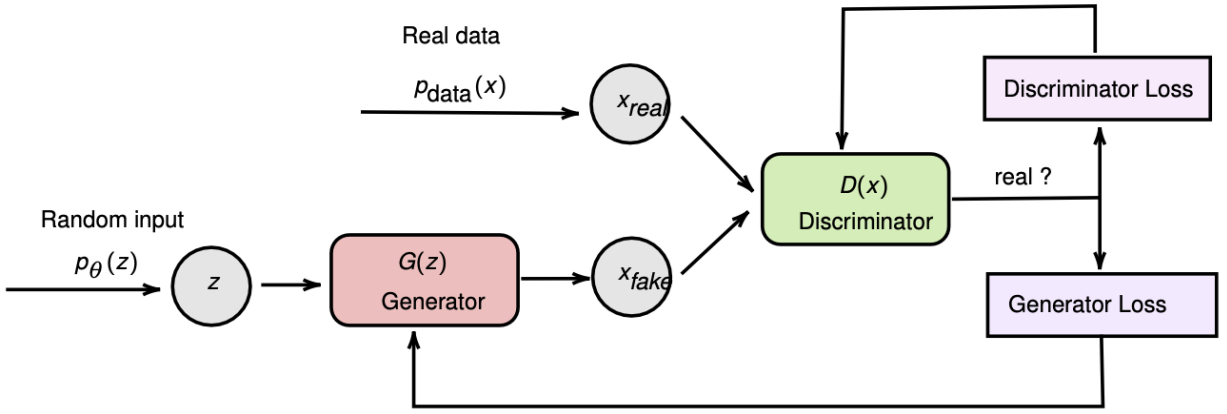
Source: Kevin Murphy, 2023, Probabilistic Machine learning: Advanced topics (part IV: Generation)

GANs

A game theory perspective

GANs operate as a Two Player Game:

- The descriminator is optimized/updated/adjusted to improve its performance at discriminating fake from real data
- The generator is optimized/updated/adjusted to improve generating samples that fool the discriminator (minimize its ability to)
- In this sense, the two models are competing against each other, they are adversarial and play a zero-sum game
  - when the discriminator successfully identifies real and fake samples, it is rewarded or no change is needed to the model parameters, whereas the generator is penalized with large updates to model parameters.
  - Alternately, when the generator fools the discriminator, it is rewarded, or no change is needed to the model parameters, but the discriminator is penalized and its model parameters are updated.
  - At a limit, the generator generates perfect replicas from the input domain every time, and the discriminator cannot tell the difference and predicts "unsure" (e.g. 50% for real and fake) in every case.



From binary classification to the GAN objective

- We train the **discriminator** $D(x)$ as a binary classifier:
  - Input: an image x
  - Output: $D(x) \in (0,1)$, interpreted as $P(y=1 \mid x)$ = "probability that x is **real**".
- Construct a mixed training set:
  - Real samples: $x \sim p_{\text{data}}(x)$, label $y=1$
  - Fake samples: $x = G(z)$ with $z \sim p_z(z)$ (so $x \sim p_g(x)$), label $y=0$
  - For simplicity, assume we see each class with probability $\frac{1}{2}$.
- For a single example (x, y) the binary cross-entropy loss is

$$\ell(D(x), y) = -y \log D(x) - (1-y) \log\big(1 - D(x)\big).$$

- The expected loss of D over the mixed data distribution $q(x, y)$ is

$$J_D = \mathbb{E}_{(x,y)\sim q}\big[\ell(D(x), y)\big].$$

- Because $P(y=1) = P(y=0) = \frac{1}{2}$ and $p(x \mid y{=}1) = p_{\text{data}}(x)$, $p(x \mid y{=}0) = p_g(x)$, we can expand:

$$J_D = -\frac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\big[\log D(x)\big] - \frac{1}{2}\,\mathbb{E}_{x\sim p_g}\big[\log(1 - D(x))\big].$$

From Discriminator loss to Min-Max GAN Objective

- Minimizing the discriminator loss J_D is equivalent to **maximizing** the following value function (just multiply by -2):

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D(x)\big] + \mathbb{E}_{x\sim p_g}\big[\log(1-D(x))\big].$$

- Recall that

$$x \sim p_g(x) \quad \Leftrightarrow \quad x = G(z),\ z \sim p_z(z),$$

  so we usually write

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z}\big[\log(1-D(G(z)))\big].$$

- **Game formulation**
  - The **discriminator** D wants to **maximize** $V(D,G)$
    (classify real as 1 and fake as 0).
  - The **generator** G wants to **minimize** $V(D,G)$
    (make $G(z)$ so realistic that $D(G(z))$ is close to 1).
- This yields the classic GAN min–max objective:

$$\min_G \max_D V(D,G) = \min_G \max_D \Big[\mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\sim p_z} \log\big(1-D(G(z))\big)\Big].$$

- (Optional extra line you can add below the formula:)
  "In practice, we often use the **non-saturating** generator loss
  $\max_G \mathbb{E}_{z\sim p_z}[\log D(G(z))]$ for stronger gradients."

Training

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} \left[\log p_D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \left(1 - p_D(g(z)) \right) \right]$$

where

- p_z the probability distribution of the random data
- g is the generator function/transformation
- p_D the probability distribution of the discriminator

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Illustration

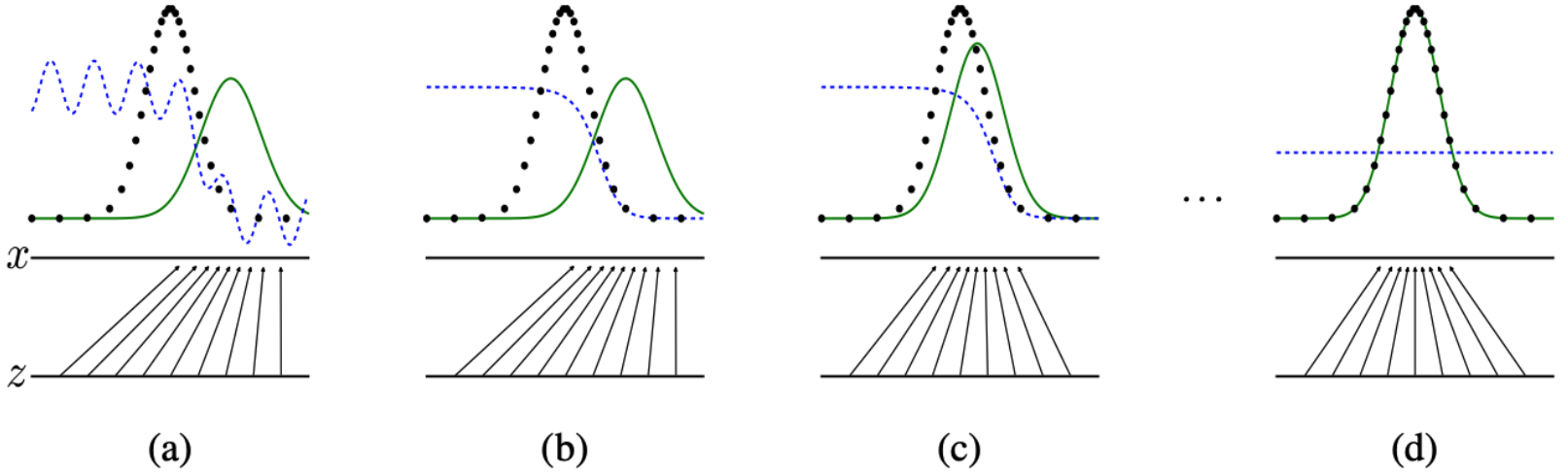


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ 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 $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Source: Ian Goodfellow et al, 2014, Generative Adversarial Nets

Theoretical Analysis

Theorem. For a fixed generator G, the optimal discriminator $D_G^\star$ is given by

$$D_G^\star(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Theorem. Let D be a fixed, differentiable discriminator, and G_θ a generator with parameters θ. Consider the **(saturating) generator loss**

$$J_G(\theta) = \mathbb{E}_{z\sim p_z}\big[\log(1 - D(G_\theta(z)))\big].$$

Then the gradient of J_G w.r.t. θ is

$$\nabla_\theta J_G(\theta) = -\mathbb{E}_{z\sim p_z}\Big[\frac{1}{1 - D(G_\theta(z))}\ \nabla_x D(x)\big|_{x=G_\theta(z)}\ \nabla_\theta G_\theta(z)\Big].$$

Equivalently, the **update direction** for gradient descent,

$$-\nabla_\theta J_G(\theta),$$

is a weighted average of directions $\nabla_\theta G_\theta(z)$, where the **scalar weight** for a sample $x = G_\theta(z)$ is

$$w_D(x) = \frac{1}{1 - D(x)}.$$

- $D_G^\star(x)$ is large (≈ 1) where real data is much more likely than generated data.
- It is small (≈ 0) where the generator puts more probability mass than the real data.
- So the discriminator is really learning a **likelihood ratio** between real and fake:

$$\frac{D_G^*(x)}{1 - D_G^*(x)} = \frac{p_{\text{data}}(x)}{p_g(x)}.$$

- **Intuition.** The optimal D is a "critic" that tells us, for each x, whether the generator is **under-representing** or **over-representing** that region:
  - If $p_g(x) \ll p_{\text{data}}(x)$, then $D_G^*(x) \approx 1$: data lives here but the generator rarely visits it $\rightarrow$ the generator receives a **strong gradient** to move probability mass toward these regions.
  - If $p_g(x) \gg p_{\text{data}}(x)$, then $D_G^*(x) \approx 0$: the generator is putting too much mass where the data almost never occurs $\rightarrow$ the gradient pushes probability mass away.
  - As we alternate updates, D keeps re-estimating this ratio and G keeps shifting its distribution until there is no systematic direction to move mass; i.e. p_g matches p_{data} and $D_G^*(x) = \frac{1}{2}$ everywhere.

Proof

Theorem. For a fixed generator G, the optimal discriminator $D_G^\star$ is given by

$$D_G^\star(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Proof.

$$V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\big[\log p_D(x)\big] + \mathbb{E}_{z\sim p_z(z)}\big[\log\left(1 - p_D(g(z))\right)\big]$$

We have:

$$\mathbb{E}_{x\sim p_{data}(x)}\big[\log p_D(x)\big] = \int_x \log p_D(x) p_{data}(x) dx$$

Let $x = g(z)$ (generator generating data x from noise z). So $x \sim p_g$.

By the law of the unconscious Statistician via push forward measures

$$\int_z f(g(z)) p_z(z) dz = \int_x f(x) p_g(x) dx$$

for $f(x) = \log\left(1 - p_D(x)\right)$, we have

$$\begin{aligned}\mathbb{E}_{z\sim p_z(z)}\big[\log\left(1 - p_D(g(z))\right)\big] &= \mathbb{E}_{z\sim p_z(z)}\big[f(g(z))\big] \\ &= \int_x f(x) p_g(x) dx \\ &= \int_x \log\left(1 - p_D(x)\right) p_g(x) dx\end{aligned}$$

Therefore

$$\begin{aligned}V(G,D) &= \mathbb{E}_{x\sim p_{data}(x)}\big[\log p_D(x)\big] + \mathbb{E}_{z\sim p_z(z)}\big[\log\left(1 - p_D(g(z))\right)\big] \\ &= \int_x \log p_D(x) p_{data}(x) dx + \int_x \log\left(1 - p_D(x)\right) p_g(x) dx \\ &= \int_x \big[\log p_D(x) p_{data}(x) + \log\left(1 - p_D(x)\right) p_g(x)\big] dx \\ &= \int_x \phi(p_D(x)) dx\end{aligned}$$

where

$$\phi(y) = \big[\log y\big] p_{data}(x) + \big[\log\left(1 - y\right)\big] p_g(x)$$

The derivatives of $\phi(y)$ are:

$$\phi'(y) = \frac{1}{y} p_{data}(x) - \frac{1}{1-y} p_g(x)$$

$$\phi''(y) = -\frac{1}{y^2} p_{data}(x) - \frac{1}{(1-y)^2} p_g(x) \leq 0$$

$\phi'(y) = 0 \to (1-y) p_{data}(x) = y p_g(x)$ therefore

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

Setting $\phi'_x(y) = 0$ gives

$$\frac{p_{\text{data}}(x)}{y} = \frac{p_g(x)}{1-y} \implies y^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Convergence Guarantees

Theorem
Assume:

- The generator G and discriminator D have enough capacity to represent the true distributions.
- At each step of Algorithm 1, the discriminator is trained to its optimum for the current generator G, i.e. $D = D^*_G$.
- The generator is then updated so as to **improve** the criterion (the virtual training criterion)

$$C(G) \;=\; \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D^*_G(x)\big] + \mathbb{E}_{x\sim p_g}\big[\log(1 - D^*_G(x))\big]$$

Then the generator distribution p_g **converges** to the data distribution p_{data}.

Recall from Theorem (Global optimum of the generator):

- For the optimal discriminator D^*_G, the generator's objective is

$$C(G) \;=\; \max_D V(D,G) \;=\; -\log 4 + 2\,\mathrm{JSD}\big(p_{\text{data}} \,\|\, p_g\big).$$

- The Jensen–Shannon divergence $\mathrm{JSD}(P\|Q)$ satisfies:
  - $\mathrm{JSD}(P\|Q) \geq 0$ for all P, Q.
  - $\mathrm{JSD}(P\|Q) = 0$ **iff** $P = Q$.

So what is Algorithm 1 doing under the assumptions of Prop. 2?

- Each step:
  1. Train D to optimality $\rightarrow D = D^*_G$.
  2. Update G to **decrease** $C(G)$.
- But

$$C(G) = -\log 4 + 2\,\mathrm{JSD}(p_{\text{data}}\|p_g),$$

  so decreasing $C(G)$ is **exactly the same** as decreasing $\mathrm{JSD}(p_{\text{data}}\|p_g)$.

- The unique global minimum of this objective is when $p_g = p_{\text{data}}$.

Intuition:
In this idealized setting, GAN training is just gradient-based optimization of a divergence between p_g and p_{data} that has a single minimum at the true data distribution. If we keep improving this objective, we must be moving p_g towards p_{data}.

Proof

1. **Identify the objective.**

   By definition of the optimal discriminator D_G^*, the "criterion" in the statement is exactly

$$\begin{aligned} C(G) =& \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D_G^*(x)\big] + \mathbb{E}_{x\sim p_g}\big[\log(1 - D_G^*(x))\big] \\ =& \max_D V(D, G). \end{aligned}$$

2. **Use Theorem 1.**

   From the previous theorem we know

$$C(G) = -\log 4 + 2\,\mathrm{JSD}\big(p_{\text{data}} \,\|\, p_g\big),$$

   and that $C(G)$ has a **unique global minimum** at $p_g = p_{\text{data}}$.

3. **Monotone improvement of $C(G)$.**

   The assumptions of the proposition say:

   - At each step, D is set to D_G^* (so the correct $C(G)$ is used).
   - The generator update **improves** the criterion:

$$C(G_{t+1}) \le C(G_t) \quad \text{for all steps } t,$$

     with strict inequality whenever we are not at an optimum.

   Hence $\{C(G_t)\}$ is a **monotonically decreasing** sequence.

4. **Lower bound and convergence.**

   Since $\mathrm{JSD}(\cdot\|\cdot) \ge 0$, we have

$$C(G_t) \ge -\log 4 \quad \text{for all } t.$$

   Thus the decreasing sequence $\{C(G_t)\}$ is bounded below, so it converges to some limit $C^\star \ge -\log 4$.

5. **Characterize the limit.**

   By the "enough capacity" assumption and the fact that the optimization is always improving the *global* criterion $C(G)$ (idealized gradient descent in function space), the only stable point is the **global** minimum:

$$C^\star = -\log 4.$$

   From Theorem 1, this happens **iff** $p_g = p_{\text{data}}$.

   Therefore, under these idealized assumptions,

$$p_g \longrightarrow p_{\text{data}}$$

   as training proceeds.

Remark: This is an **idealized convergence result**. In practice we don't train D to exact optimality, models have limited capacity, and optimization is non-convex, so real GAN training does not enjoy such a strong theoretical guarantee.

Vanilla GAN as a Zero-Sum Min–Max Game & Nash Equilibrium

- **Players & strategies**
  - Generator: G_θ maps noise $z \sim p_z$ to fake samples $G_\theta(z)$.
  - Discriminator: $D_\phi(x) \in (0,1)$ outputs "probability that x is real".
  - Strategy spaces: parameter sets (θ, ϕ) of the two networks.
- **Value function (payoff of the discriminator)**

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))].$$

- **Zero-sum game**
  - Discriminator payoff: $u_D(D,G) = V(D,G)$.
  - Generator payoff: $u_G(D,G) = -V(D,G)$.
  - Hence $u_D(D,G) + u_G(D,G) = 0$ ⇒ **zero-sum / min–max** game:

$$\min_G \max_D V(D,G).$$

- **Nash equilibrium (theoretical)**
  A pair (D^*, G^*) is a Nash equilibrium if neither player can improve its payoff by changing strategy alone:

$$V(D^*,G^*) \geq V(D,G^*) \quad \forall D, \qquad V(D^*,G^*) \leq V(D^*,G) \quad \forall G.$$

  Under *strong* assumptions (infinite capacity, perfect optimization):
  - For fixed G, the optimal discriminator is

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

  - Plugging back:

$$V(D^*,G) = -\log 4 + 2\,\mathrm{JS}\big(p_{\text{data}} \,\|\, p_g\big),$$

    so minimizing over G is equivalent to minimizing the Jensen–Shannon divergence.
  - Global minimum at $p_g = p_{\text{data}}$, where $D^*(x) = \frac{1}{2}$ for all x ⇒ **Nash equilibrium / saddle point**.
- **How strong are the assumptions?**
  - Networks have **unlimited capacity** (can represent the optimal D^* and any p_g).
  - We can solve the **non-convex** min–max problem to **global** optima.
  - Infinite data, exact gradients, well-behaved distributions (overlapping support).
  - In practice, none of these hold exactly ⇒ training may oscillate, fail to converge, or reach only approximate / local equilibria (e.g. mode collapse).

Evaluation

- Fidelity (quality):
  - The degree to which the generated samples resemble the real ones
  - Are fake samples indistinguishable from reals ?
- Diversity (coverage):
  - Measures whether the generated samples cover the full variability of the real samples
  - Do we see all modes/classes/styles of the data ?
- Assessment:
  - Visual inspection
  - Inception Score (IS)
  - Fréchet Inception Distance (FID)
  - Precision and Recall
  - User studies

Inception Score

1. Setup

- Let $p_g(x)$ be the generator distribution data.
- Pass a generated image x through a pretrained classifier (separate oracle classifier) to get a label distribution:

$$p(y \mid x).$$

- The marginal label distribution over generated samples is

$$p(y) = \int p(y \mid x)\, p_g(x)\, dx \approx \frac{1}{N} \sum_{i=1}^{N} p(y \mid x_i), \quad x_i \sim p_g.$$

2. IS definition

- Inception Score is

$$\mathrm{IS}(p_g) = \exp\left(\mathbb{E}_{x \sim p_g}\left[D_{\mathrm{KL}}\left(p(y \mid x) \,\|\, p(y)\right)\right]\right).$$

- With finite samples:

$$\hat{\mathrm{IS}} = \exp\left(\frac{1}{N} \sum_{i=1}^{N} \sum_{y} p(y \mid x_i) \log \frac{p(y \mid x_i)}{p(y)}\right).$$

3. Interpretation

- High **fidelity** $\Rightarrow$ low entropy $H(p(y \mid x))$: each image confidently belongs to one class.
- High **diversity** $\Rightarrow$ high entropy $H(p(y))$: generated set covers many classes.
- IS mixes these via the KL divergence, but:
  - it doesn't compare to real images, and
  - it can be gamed (e.g., by producing one very realistic image per class).

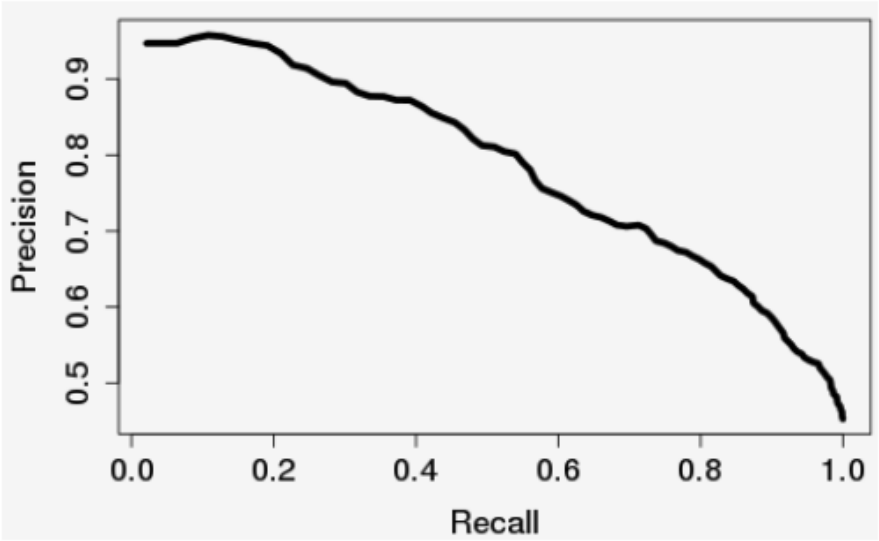
Frechet Inception Distance

Fréchet distance between Gaussians	FID for GAN evaluation in GANs
1. 1-D Gaussians	**2. Feature embeddings for real vs fake**
Let $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$, $Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$. The squared Fréchet (2-Wasserstein) distance is $$d^2(X,Y) = (\mu_X - \mu_Y)^2 + (\sigma_X - \sigma_Y)^2.$$	Use a pretrained network (e.g. Inception-v3) as a feature extractor $\phi(\cdot)$. For real data/images x_i^r, $i = 1, \ldots, N_r$, and generated images x_j^g, $j = 1, \ldots, N_g$: $$r_i = \phi(x_i^r), \quad g_j = \phi(x_j^g).$$
2. Multivariate Gaussians	**3. Sample means**
For $X \sim \mathcal{N}(\mu_X, \Sigma_X)$, $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$, the squared Fréchet distance is $$d^2(X,Y) = \|\mu_X - \mu_Y\|_2^2 + \mathrm{Tr}\Big(\Sigma_X + \Sigma_Y - 2(\Sigma_X^{1/2}\Sigma_Y\Sigma_X^{1/2})^{1/2}\Big).$$ A common equivalent form uses $(\Sigma_X\Sigma_Y)^{1/2}$ instead of the sandwich form.	Approximate the (unknown) real and generator distributions in feature space by Gaussians with empirical means $\mu_r = \frac{1}{N_r}\sum_{i=1}^{N_r} r_i$, $\mu_g = \frac{1}{N_g}\sum_{j=1}^{N_g} g_j$.
3. Covariances of Gaussians	**4. Sample covariances**
For multivariate normals, covariance matrices Σ_X and Σ_Y fully specify the spread and correlations of the distributions. They enter the Fréchet distance via a trace term and a matrix square root term: $$\mathrm{Tr}\Big(\Sigma_X + \Sigma_Y - 2(\Sigma_X^{1/2}\Sigma_Y\Sigma_X^{1/2})^{1/2}\Big).$$	Empirical covariance matrices in feature space are $$\Sigma_r = \frac{1}{N_r - 1}\sum_{i=1}^{N_r}(r_i - \mu_r)(r_i - \mu_r)^\top,$$ $$\Sigma_g = \frac{1}{N_g - 1}\sum_{j=1}^{N_g}(g_j - \mu_g)(g_j - \mu_g)^\top.$$
4. From Fréchet distance to FID	**5. FID formula & interpretation**
We now **identify** the Gaussian for real data with $\mathcal{N}(\mu_r, \Sigma_r)$ and the Gaussian for generated data with $\mathcal{N}(\mu_g, \Sigma_g)$, and plug into the multivariate Fréchet distance.	The Fréchet Inception Distance (FID) between real distribution p_r and generator distribution p_g is $$\mathrm{FID}(p_r, p_g) = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\Big(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\Big).$$ - **Lower FID is better**: the closer (μ_g, Σ_g) is to (μ_r, Σ_r), the closer the generator's feature distribution is to the real one ⇒ good fidelity **and** diversity.

Precision & Recall

Confusion Matrix

		Predicted		
		Positive	Negative	Total
Actual	Positive	TP	FN	P
	Negative	FP	TN	N
	Total	$\hat{P}$	$\hat{N}$	



- Accuracy $= \frac{TP+TN}{P+N}$
- Precision $= \frac{TP}{TP+FP} = \frac{TP}{\hat{P}}$
- Recall $= \frac{TP}{TP+FN} = \frac{TP}{P}$

Probabilistic interpretation of Precision and Recall

- Precision $= \frac{TP}{TP+FP} = \frac{TP}{\hat{P}} = P(y(x) = 1 | \hat{(}y)(x) = 1)$
- Recall $= \frac{TP}{TP+FN} = \frac{TP}{P} = P(\hat{y}(x) = 1 | y(x) = 1)$
-

$$\begin{aligned} \text{Prior} &= p(y(x) = 1) && (2) \\ &= \int p(y(x) = 1, \hat{y} = 1) d\hat{y} && (3) \\ &= \int p(y(x) = 1 | \hat{y} = 1) p(\hat{y} = 1) d\hat{y} && (4) \end{aligned}$$

so the prior is the average precision of all possible models, which is equivalent to the precision of a random model therefore, when selecting a specific classifier, it makes sense to compare the precision of the selected classifier to the prior probability of the positive class.

Precision & Recall

Precision & Recall: Intuition	Precision & Recall: Formal Definitions
1. Feature-space setup Use a pretrained network (e.g. Inception-v3) as feature extractor $\phi(\cdot)$ and work in embedding space $\mathbb{R}^d$.	Real feature set: $R = \{r_i = \phi(x_i^r)\}_{i=1}^{N_r}$, with $x_i^r \sim p_r$. Generated feature set: $G = \{g_j = \phi(x_j^g)\}_{j=1}^{N_g}$, with $x_j^g \sim p_g$.
2. Local neighborhoods (real data) We approximate the "real data manifold" by local balls around each real embedding.	For each real embedding r_i, compute the k-NN distance among reals (excluding itself): $\epsilon_i = \mathrm{kNN}_{dist}\big(r_i, R \setminus \{r_i\}\big)$. Define the real manifold as a union of balls: $\mathcal{M}_R = \bigcup_{i=1}^{N_r} B(r_i, \epsilon_i)$, where $B(r_i, \epsilon_i) = \{v \in \mathbb{R}^d : \|v - r_i\|_2 \le \epsilon_i\}$.
3. Precision (fidelity) Measures: *How many generated samples look like they come from the real manifold?*	Precision is the fraction of generated embeddings that lie inside the real manifold: $$\text{Precision} = \frac{1}{N_g} \sum_{j=1}^{N_g} \mathbf{1}\big[g_j \in \mathcal{M}_R\big].$$ High precision ⇒ most generated samples are "realistic" (few obviously fake / off-manifold samples).
4. Local neighborhoods (generated data) Similarly approximate the "generator manifold" from generated samples.	For each generated embedding g_j, compute its k-NN distance among $G \setminus \{g_j\}$: $\tilde{\epsilon}_j = \mathrm{kNN}_{dist}\big(g_j, G \setminus \{g_j\}\big)$. Define the generator manifold: $\mathcal{M}_G = \bigcup_{j=1}^{N_g} B(g_j, \tilde{\epsilon}_j)$, with $B(g_j, \tilde{\epsilon}_j) = \{v : \|v - g_j\|_2 \le \tilde{\epsilon}_j\}$.
5. Recall (diversity / coverage) Measures: *How much of the real manifold is covered by the generator?*	Recall is the fraction of real embeddings that lie inside the generator manifold: $$\text{Recall} = \frac{1}{N_r} \sum_{i=1}^{N_r} \mathbf{1}\big[r_i \in \mathcal{M}_G\big].$$ High recall ⇒ the generator covers most modes of the real data (little mode dropping).
6. Relationship Precision ↔ fidelity (sample quality). Recall ↔ diversity (mode coverage).	In practice, one often reports a **precision–recall curve** or summarizes models along both axes: - High precision, low recall: sharp but mode-collapsed GAN. - Low precision, high recall: diverse but many bad / noisy samples. - Good model: **high precision & high recall** in feature space.

Conditional GANs

Conditional Generative Adversarial Nets

Mehdi Mirza
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7
mirzamom@iro.umontreal.ca

Simon Osindero
Flickr / Yahoo Inc.
San Francisco, CA 94103
osindero@yahoo-inc.com

Abstract

Generative Adversarial Nets [8] were recently introduced as a novel way to train generative models. In this work we introduce the conditional version of generative adversarial nets, which can be constructed by simply feeding the data, y, we wish to condition on to both the generator and discriminator. We show that this model can generate MNIST digits conditioned on class labels. We also illustrate how this model could be used to learn a multi-modal model, and provide preliminary examples of an application to image tagging in which we demonstrate how this approach can generate descriptive tags which are not part of training labels.

Source: Mehdi Mirza and Simon Osindero, 2014, Conditional Generative Adversarial Nets

Conditional GANs

Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer.

In the generator the prior input noise $p_z(z)$, and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. [1]

In the discriminator x and y are presented as inputs and to a discriminative function (embodied again by a MLP in this case).

The objective function of a two-player minimax game would be as Eq 2

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \quad (2)$$

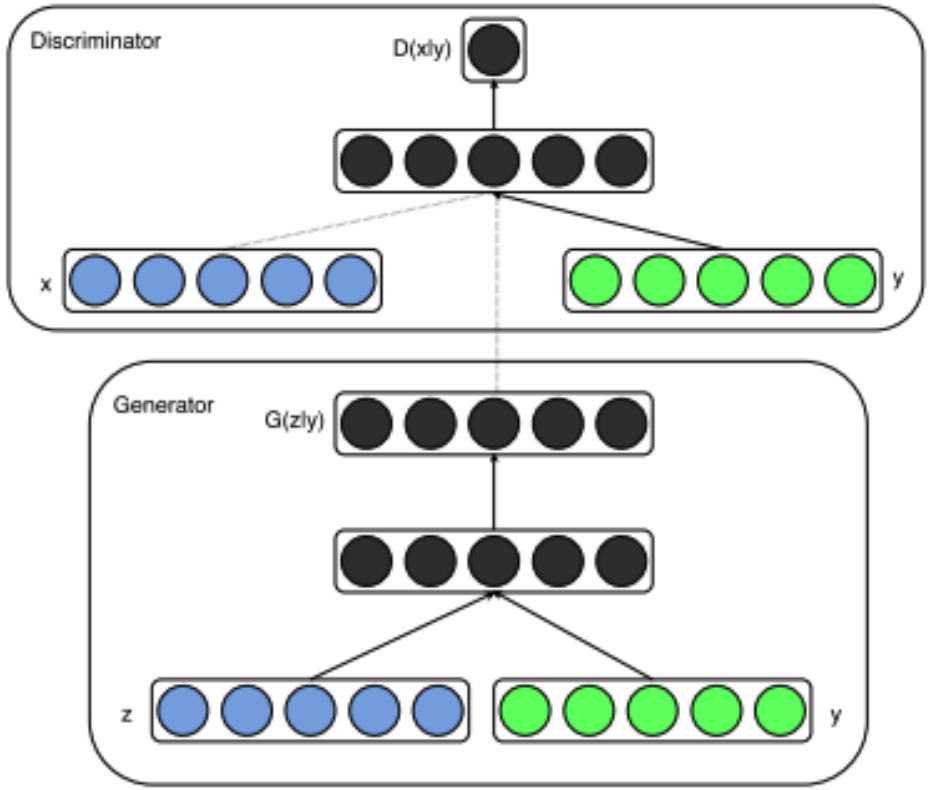


Figure 1: Conditional adversarial net

Source: Mehdi Mirza and Simon Osindero, 2014, Conditional Generative Adversarial Nets

From Unconditional to Conditional GANs

- **Goal of a conditional GAN (cGAN)**
  Learn a generator that models the **conditional distribution**

$$p_{\text{data}}(x \mid y)$$

  instead of the unconditional $p_{\text{data}}(x)$.

- **Data and noise distributions**
  - Labeled data: $(x, y) \sim p_{\text{data}}(x, y)$
    with $p_{\text{data}}(x, y) = p_{\text{data}}(y)\, p_{\text{data}}(x \mid y)$.
  - Noise: $z \sim p_z(z)$ (e.g. standard normal or uniform).
- **Conditional generator**
  - Generator receives both noise and condition:

$$x_g = G(z, y)$$

  - Induces a conditional model

$$p_g(x \mid y)$$

    and a joint model

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

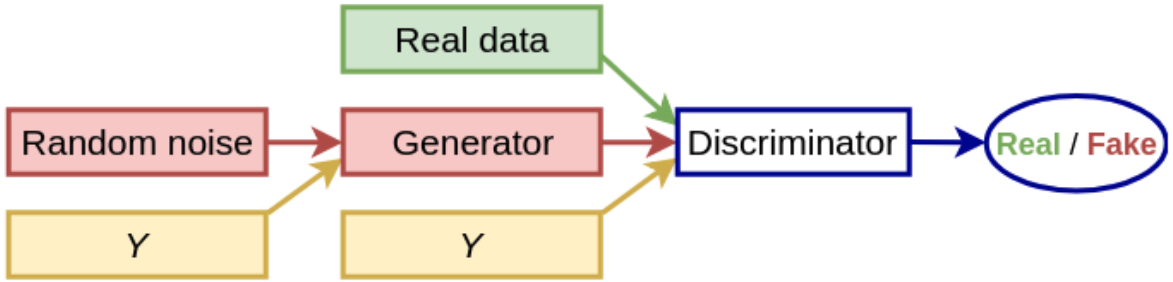
    (we usually reuse the empirical $p_{\text{data}}(y)$ for labels).
- **Conditional discriminator**
  - Discriminator also receives both x and y:

$$D(x, y) \in (0, 1)$$

    interpreted as

$$D(x, y) \approx P(\text{“real”} = 1 \mid x, y).$$

  - In Mirza–Osindero's notation you'll often see $D(x \mid y)$; this is equivalent to feeding (x, y) together into D.



source

Deriving the conditional GAN objective

- **Binary classification view**

  Create a dataset of pairs (x, y) with binary label $t \in \{0, 1\}$:

  - Real pairs: $(x, y) \sim p_{\text{data}}(x, y)$, label $t = 1$.
  - Fake pairs: sample $y \sim p_{\text{data}}(y)$, $z \sim p_z(z)$, form $(x, y) = (G(z, y), y)$, label $t = 0$.

  For a single pair, binary cross-entropy loss is

$$\ell(D(x, y), t) = -t \log D(x, y) - (1 - t) \log\big(1 - D(x, y)\big).$$

- **Expected discriminator loss**

  Let $q(x, y, t)$ be the mixed distribution over real and fake pairs. The expected loss of D is

$$\begin{aligned} J_D &= \mathbb{E}_{(x,y,t)\sim q}\big[\ell(D(x, y), t)\big] \\ &= -\mathbb{E}_{(x,y)\sim p_{\text{data}}}\big[\log D(x, y)\big] - \mathbb{E}_{y\sim p_{\text{data}}(y),\, z\sim p_z}\big[\log(1 - D(G(z, y), y))\big]. \end{aligned}$$

- **Value function and min–max game**

  As in the vanilla GAN case, maximizing the following is equivalent to minimizing J_D (up to a constant factor):

$$V(D, G) = \mathbb{E}_{(x,y)\sim p_{\text{data}}}\big[\log D(x, y)\big] + \mathbb{E}_{y\sim p_{\text{data}}(y),\, z\sim p_z}\big[\log(1 - D(G(z, y), y))\big].$$

  The conditional GAN objective is therefore

$$\min_G \max_D V(D, G) = \min_G \max_D \Big[\mathbb{E}_{(x,y)\sim p_{\text{data}}} \log D(x, y) + \mathbb{E}_{y\sim p_{\text{data}}(y),\, z\sim p_z} \log(1 - D(G(z, y), y))\Big].$$

  - Often written with conditional notation:

$$\min_G \max_D \Big[\mathbb{E}_{x,y\sim p_{\text{data}}} \log D(x \mid y) + \mathbb{E}_{z\sim p_z,\, y\sim p_{\text{data}}(y)} \log(1 - D(G(z \mid y) \mid y))\Big].$$

InfoGAN

InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

Xi Chen[†‡], **Yan Duan**[†‡], **Rein Houthooft**[†‡], **John Schulman**[†‡], **Ilya Sutskever**[‡], **Pieter Abbeel**[†‡]
† UC Berkeley, Department of Electrical Engineering and Computer Sciences
‡ OpenAI

Abstract

This paper describes InfoGAN, an information-theoretic extension to the Generative Adversarial Network that is able to learn disentangled representations in a completely unsupervised manner. InfoGAN is a generative adversarial network that also maximizes the mutual information between a small subset of the latent variables and the observation. We derive a lower bound of the mutual information objective that can be optimized efficiently. Specifically, InfoGAN successfully disentangles writing styles from digit shapes on the MNIST dataset, pose from lighting of 3D rendered images, and background digits from the central digit on the SVHN dataset. It also discovers visual concepts that include hair styles, presence/absence of eyeglasses, and emotions on the CelebA face dataset. Experiments show that InfoGAN learns interpretable representations that are competitive with representations learned by existing supervised methods.

Source: Xi Chen et al (2016) Wasserstein GANs

From GAN to InfoGAN: who controls what?

Vanilla GAN (pure unsupervised)

- Input: random noise $z \rightarrow$ generator $G(z) \rightarrow$ fake sample $\hat{x}$
- Discriminator $D(x)$ learns to tell real vs fake
- No structure is imposed on z:
  - dimensions of z are **entangled**
  - changing one coordinate in z changes "everything a bit"
  - you can't say "this dimension = digit / pose / style"
- Great at realistic sampling, poor at **controllable** generation

Conditional GAN (fully supervised)

- Inputs: noise z **and** label y (class, attribute, text)
  - $G(z, y)$ generates $\hat{x}$ consistent with y
  - $D(x, y)$ checks both realism and label consistency
- Pros: strong control; pick y, you get that class/attribute
- Cons:
  - requires labeled data
  - only learns structure you **explicitly provide** as labels

Motivation for InfoGAN

- We want:
  - unsupervised training like vanilla GAN
  - but with controllable factors like conditional GAN
- Idea: introduce "label-like" latent codes that the model must **discover**, not receive as supervision

InfoGAN: splitting noise and structure

Latent split: z vs c

- InfoGAN uses two parts in the input:
  - z – pure noise (unstructured randomness)
  - c – *structured code* meant to hold semantics
    - **discrete** codes (e.g. 10-way "digit identity")
    - **continuous** codes (e.g. rotation, thickness, style)
- Generator becomes $G(z, c)$, analogous to cGAN's $G(z, y)$

What we want from: c

- Changing c should cause **predictable changes** in the generated sample $\hat{x}$:
  - discrete c could switch between clusters (e.g. digit 0–9)
  - continuous c could smoothly rotate, thicken, or stretch shapes
- In other words, c should behave like:
  - "hidden labels" for classes / groups
  - plus extra knobs for underlying factors of variation

The problem: G might ignore c

- Just feeding c into G is not enough:
  - G could learn $G(z, c) \approx G(z)$ and completely ignore c
- We need a training signal that *forces* G to actually use c in a meaningful way

InfoGAN: mutual information as “make c readable”

Key idea: maximize mutual information

- Define $\hat{x} = G(z, c)$ and maximize $I(c; \hat{x})$ (mutual information)
- Intuition:
  - Given only the generated sample $\hat{x}$, you should be able to **guess** c reliably
  - If $\hat{x}$ doesn't depend on c, then $I(c; \hat{x})$ is small

InfoGAN: Variational Lower Bound

Goal:

We want to *maximize* mutual information
$I(c; x)$ with $x = G(z, c)$, but it depends on the intractable posterior
$P(c \mid x)$.
Trick: introduce an auxiliary distribution $Q(c \mid x)$ and build a **lower bound**.

Start from mutual information

$$I(c; x) = H(c) - H(c \mid x) = H(c) + \mathbb{E}_{x \sim G(z,c)} \mathbb{E}_{c' \sim P(c|x)} [\log P(c' \mid x)].$$

Introduce variational posterior $Q(c \mid x)$

Add and subtract $\log Q(c' \mid x)$:

$$\log P(c' \mid x) = \log Q(c' \mid x) + \log \frac{P(c' \mid x)}{Q(c' \mid x)}.$$

Plugging in:

$$I(c; x) = H(c) + \mathbb{E}_x \Big[KL\big(P(\cdot \mid x) \,\|\, Q(\cdot \mid x)\big) + \mathbb{E}_{c' \sim P(c|x)} [\log Q(c' \mid x)] \Big].$$

Since $KL(\cdot \| \cdot) \geq 0$:

$$I(c; x) \geq H(c) + \mathbb{E}_x \mathbb{E}_{c' \sim P(c|x)} [\log Q(c' \mid x)].$$

Rewrite expectation over the joint (c, x)

Using a change-of-variables / law of total expectation (Lemma in the paper):

$$\mathbb{E}_x \mathbb{E}_{c' \sim P(c|x)} [\log Q(c' \mid x)] = \mathbb{E}_{c \sim P(c),\, x \sim G(z,c)} [\log Q(c \mid x)].$$

Define the **variational lower bound**:

$$L_I(G, Q) = \mathbb{E}_{c \sim P(c),\, x \sim G(z,c)} [\log Q(c \mid x)] + H(c) \leq I(c; x).$$

InfoGAN objective

Add this term to the usual GAN value function (V(D,G)):

$$\min_G \max_{D,Q} \; V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q).$$

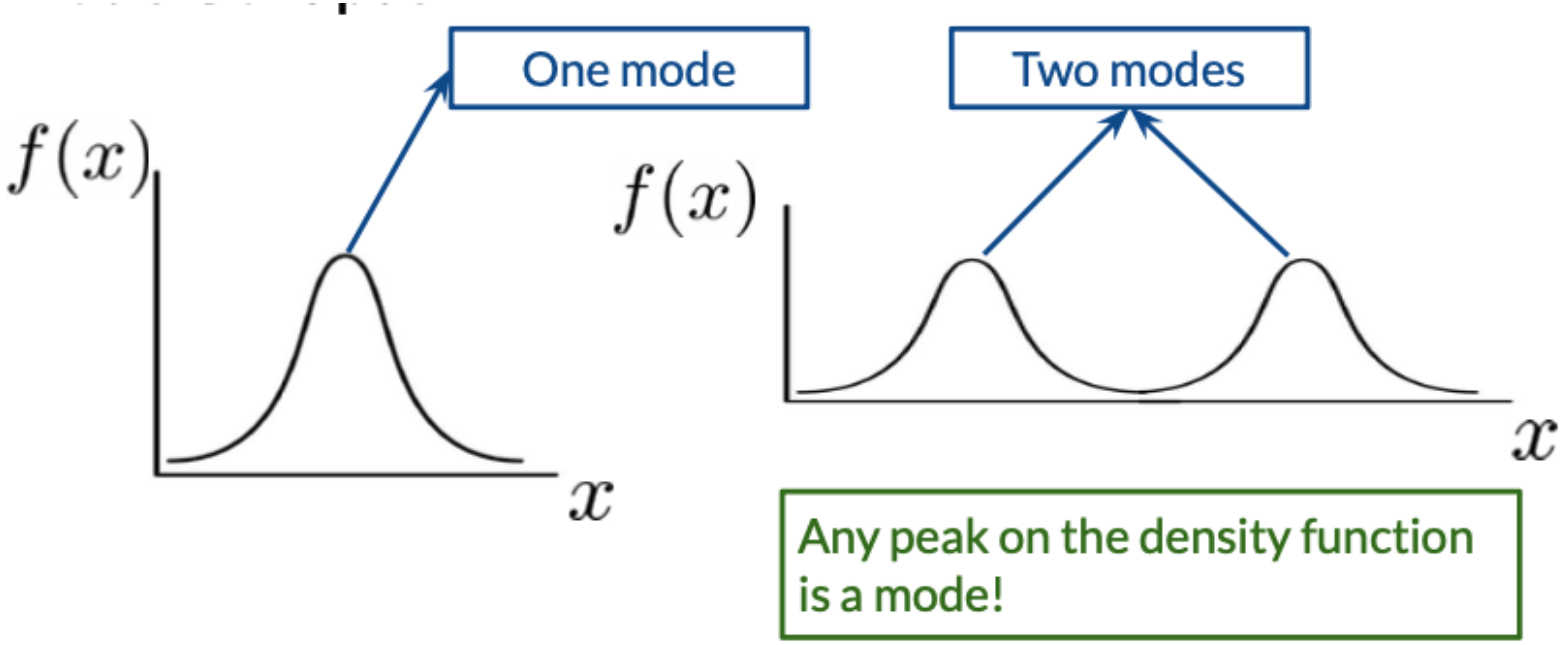
- D: discriminate real vs fake (standard GAN role)
- G: fool D **and** make c highly informative about x
- Q: approximate $P(c \mid x)$ so the bound is tight

Problems in GANs

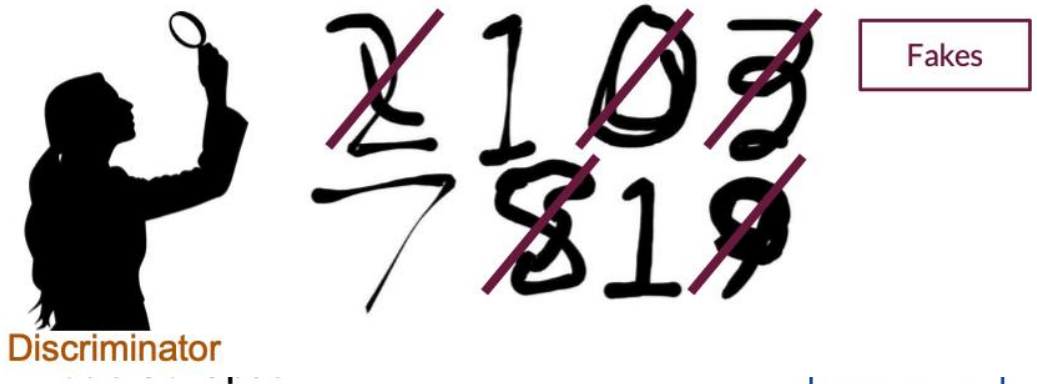
- Hard to achieve Nash equilibrium
- Low dimensional supports
- Vanishing gradient
- Mode collapse
- Lack of proper evaluation metric

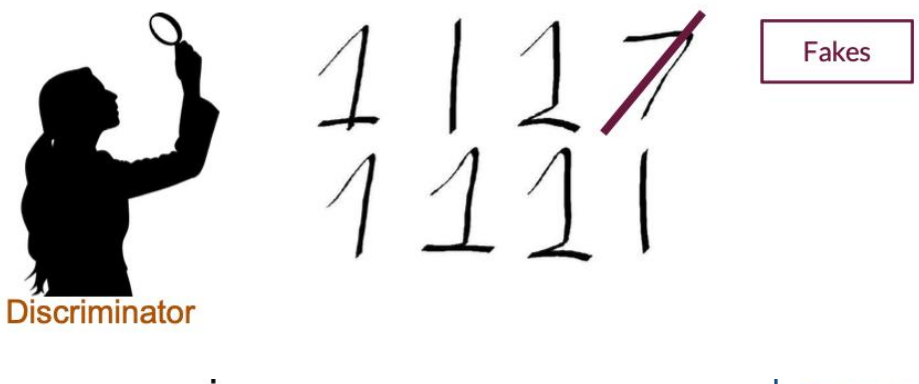
Mode collapse

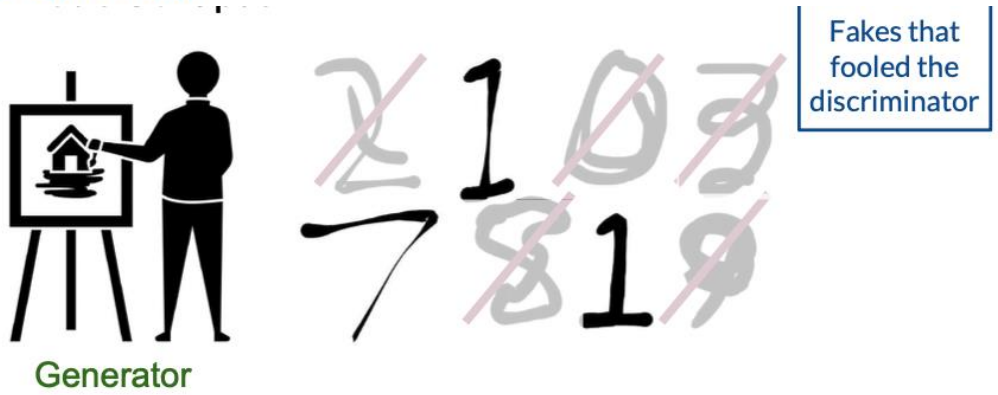
- During training, the generator may collapse to a setting where it always produces the same outputs
- The generator still can be able to trick the discriminator , but it fails to represent the complex real-world data distribution and gets stuck in a small space with extremely low variety
- The generator isn't penalized for lack of diversity

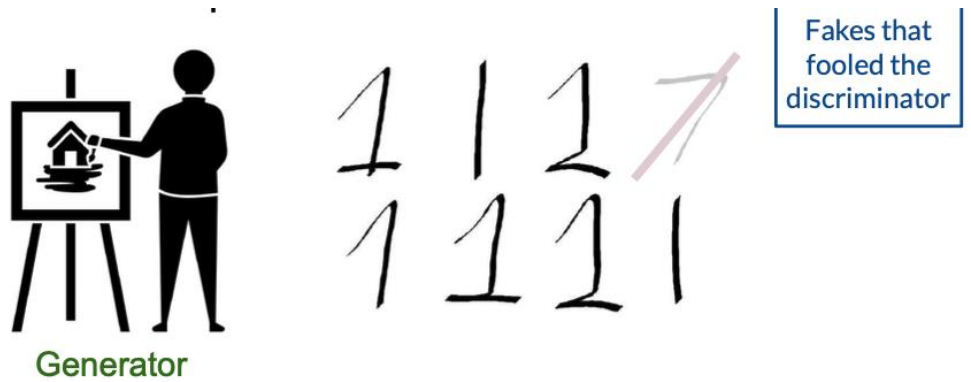


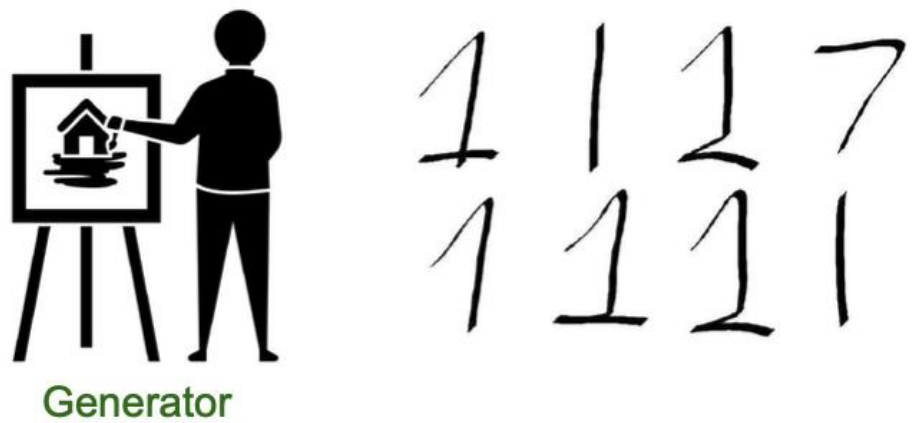
Coursera GANs Specialization

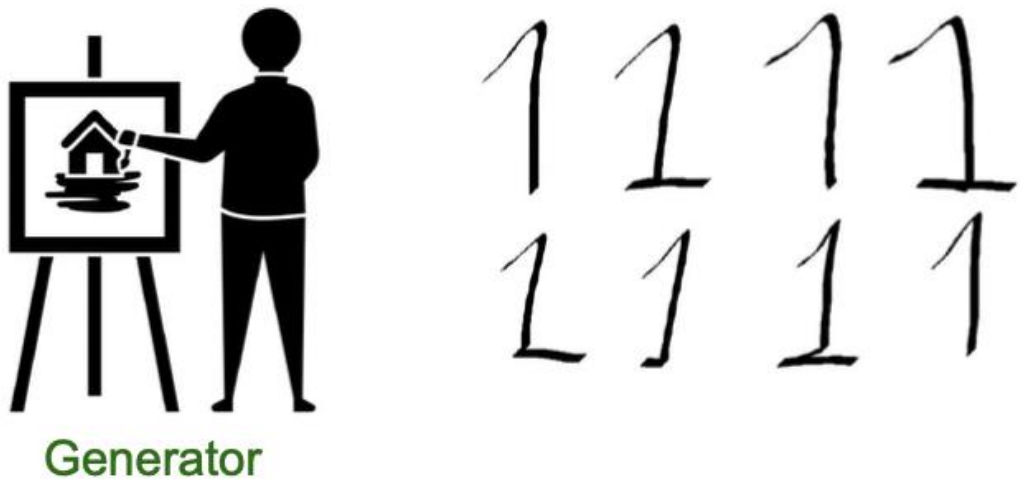












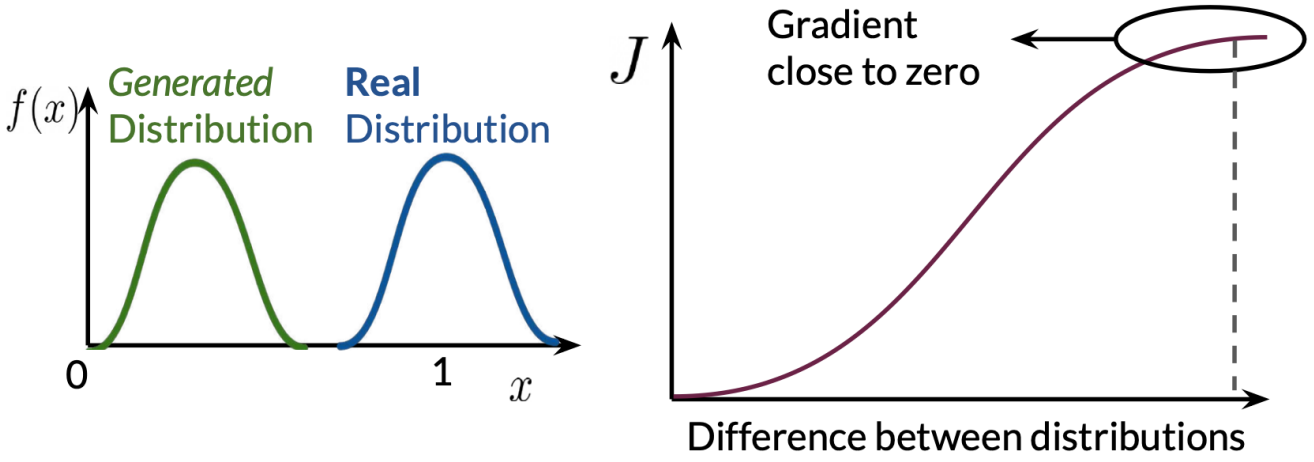
Coursera [GANs Specialization]

Vanishing gradient

GAN faces a **dilemma**:

- If the discriminator behaves badly, the generator does not have accurate feedback and the loss function cannot represent the reality.
- If the discriminator does a great job, the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed.

This dilemma clearly is capable to make the GAN training very tough.



Source: coursera GANs specialization

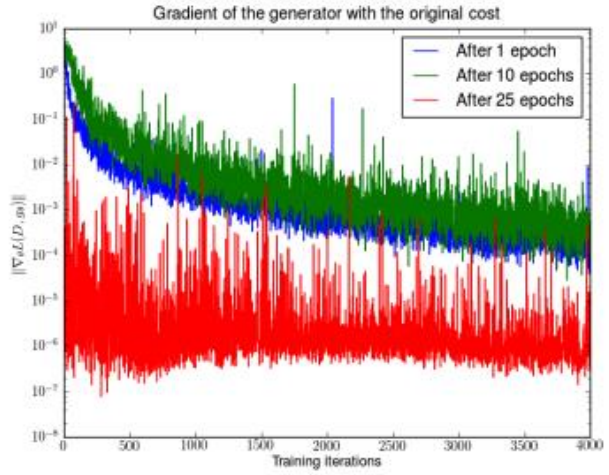
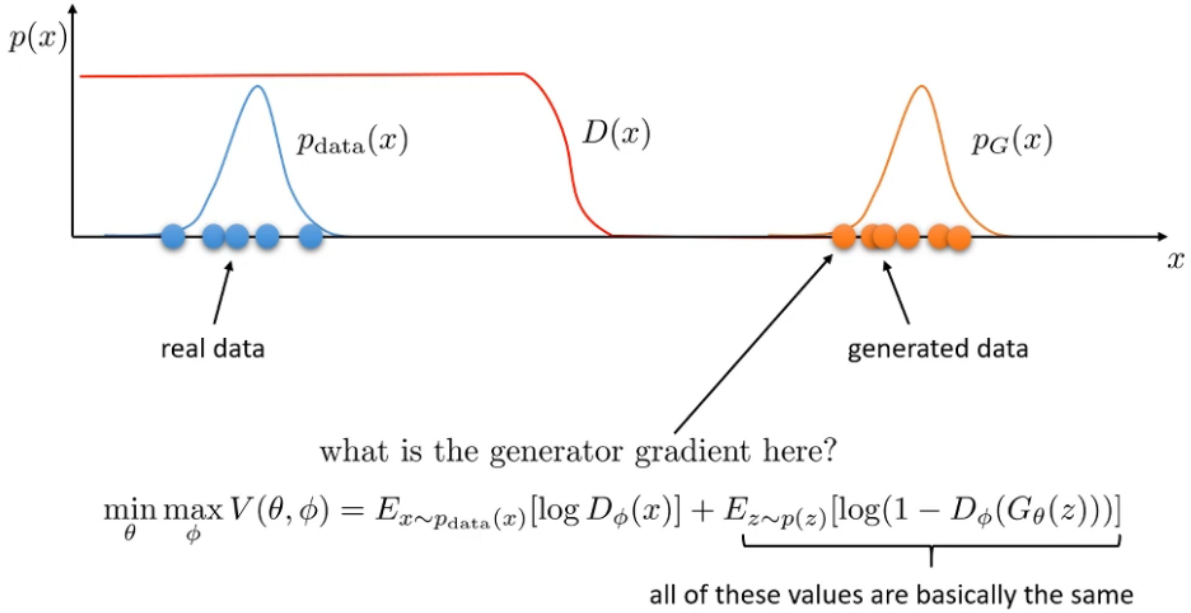


Figure 2: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale.

Source: Martin Arjovsky et al, 2017, Towards principled methods for training GANs



Wasserstein GANs

Wasserstein Generative Adversarial Networks

Martin Arjovsky [1] **Soumith Chintala** [2] **Léon Bottou** [1 2]

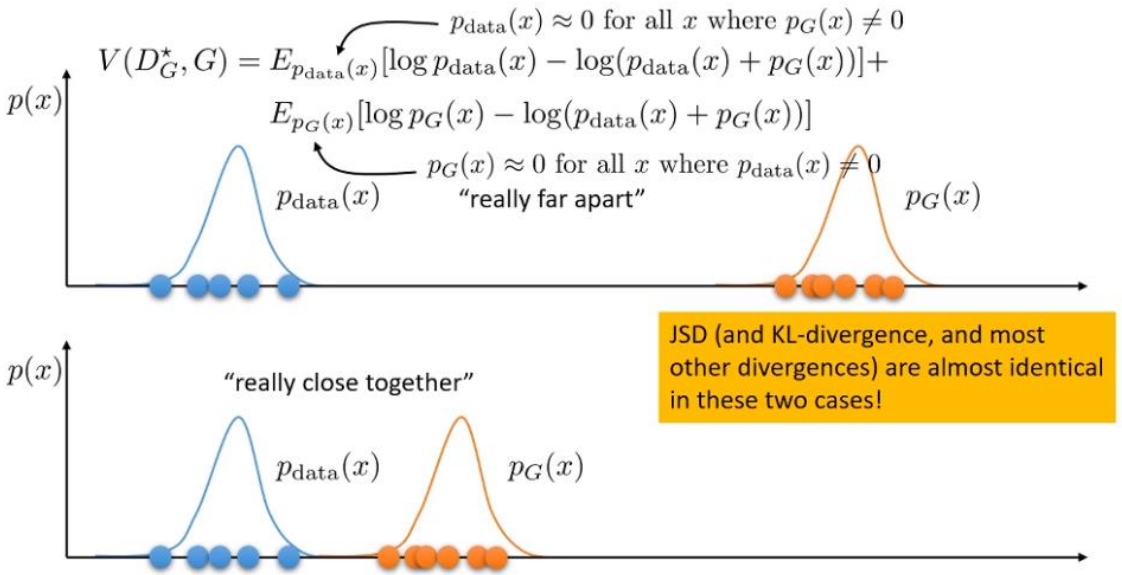
Abstract

We introduce a new algorithm named WGAN, an alternative to traditional GAN training. In this new model, we show that we can improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches. Furthermore, we show that the corresponding optimization problem is sound, and provide extensive theoretical work highlighting the deep connections to different distances between distributions.

Source: Martin Arjovsky et al (2017) Wasserstein GANs

Wasserstein GAN – Intuition

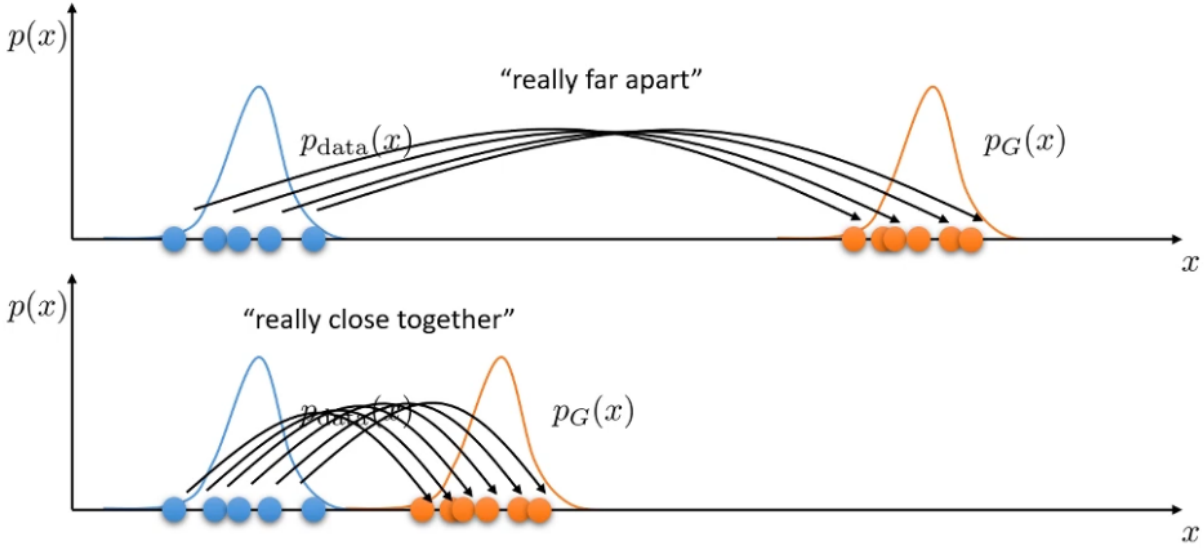
High-level intuition: the JS divergence used by the classic GAN doesn't account for "distance"



JSD (and KL-divergence, and most other divergences) are almost identical in these two cases!

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another



Source: [From GAN to WGAN]

Wasserstein distance Illustration

What is Wasserstein distance?

Wasserstein Distance is a measure of the distance between two probability distributions. It is also called **Earth Mover's distance**, short for EM distance, because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution. The cost is quantified by: the amount of dirt moved x the moving distance.

Source: From GAN to WGAN

Let us first look at a simple case where the probability domain is *discrete*. For example, suppose we have two distributions P and Q, each has four piles of dirt and both have ten shovelfuls of dirt in total. The numbers of shovelfuls in each dirt pile are assigned as follows:

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4 Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$

In order to change P to look like Q, as illustrated in Fig. 7, we:

- First move 2 shovelfuls from P_1 to P_2 => (P_1, Q_1) match up.
- Then move 2 shovelfuls from P_2 to P_3 => (P_2, Q_2) match up.
- Finally move 1 shovelfuls from Q_3 to Q_4 => (P_3, Q_3) and (P_4, Q_4) match up.

If we label the cost to pay to make P_i and Q_i match as δ_i, we would have $\delta_{i+1} = \delta_i + P_i - Q_i$ and in the example:

$$\begin{aligned}
\delta_0 &= 0 \\
\delta_1 &= 0 + 3 - 1 = 2 \\
\delta_2 &= 2 + 2 - 2 = 2 \\
\delta_3 &= 2 + 1 - 4 = -1 \\
\delta_4 &= -1 + 4 - 3 = 0
\end{aligned}$$

Finally the Earth Mover's distance is $W = \sum |\delta_i| = 5$.

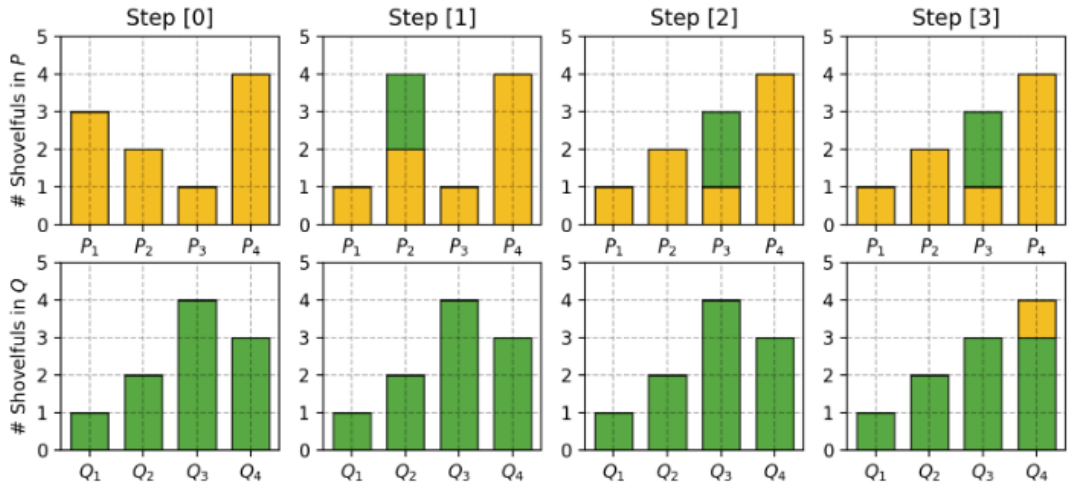


Fig. 7. Step-by-step plan of moving dirt between piles in P and Q to make them match.

WGAN – Wasserstein distance

When dealing with the continuous probability domain, the distance formula becomes:

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

In the formula above, $\Pi(p_r, p_g)$ is the set of all possible joint probability distributions between p_r and p_g. One joint distribution $\gamma \in \Pi(p_r, p_g)$ describes one dirt transport plan, same as the discrete example above, but in the continuous probability space. Precisely $\gamma(x, y)$ states the percentage of dirt should be transported from point x to y so as to make x follows the same probability distribution of y. That's why the marginal distribution over x adds up to p_g, $\sum_x \gamma(x, y) = p_g(y)$ (Once we finish moving the planned amount of dirt from every possible x to the target y, we end up with exactly what y has according to p_g.) and vice versa $\sum_y \gamma(x, y) = p_r(x)$.

When treating x as the starting point and y as the destination, the total amount of dirt moved is $\gamma(x, y)$ and the travelling distance is $|x - y|$ and thus the cost is $\gamma(x, y) \cdot |x - y|$. The expected cost averaged across all the (x, y) pairs can be easily computed as:

$$\sum_{x,y} \gamma(x, y)\|x - y\| = \mathbb{E}_{x,y\sim\gamma}\|x - y\|$$

Finally, we take the minimum one among the costs of all dirt moving solutions as the EM distance. In the definition of Wasserstein distance, the $\inf$ (infimum, also known as *greatest lower bound*) indicates that we are only interested in the smallest cost.

Source: From GAN to WGAN

Wasserstein distance

> **Idea:** How much "work" does it take to move the model distribution p_g onto the real data distribution p_r?

Intuition: piles of dirt

- Think of p_r (real data) as **piles of dirt** at positions x.
- Think of p_g (generator) as **holes** at positions y that must be filled.
- A **transport plan** $\gamma(x, y)$ tells us:
  - what *fraction of dirt* at each x we move to each y.
- Moving that fraction over a distance $\|x - y\|$ costs:

$$\text{cost}(x \to y) = \gamma(x, y)\, \|x - y\|.$$

- The **total work** of a plan is the average cost over all pairs ((x,y)):

$$\text{Work}(\gamma) = \sum_{x,y} \gamma(x, y)\, \|x - y\| = \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|].$$

Formal definition (continuous version)

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|].$$

What each symbol means

- p_r – real data distribution
- p_g – generator distribution
- $\gamma(x, y)$ – a **joint distribution** over pairs (x, y)
- $\Pi(p_r, p_g)$ – the set of all such γ whose:
  - **x-marginal** is p_r: $\sum_y \gamma(x, y) = p_r(x)$
  - **y-marginal** is p_g: $\sum_x \gamma(x, y) = p_g(y)$

  (i.e. γ is a valid way to match all real mass to all generated mass)
- $\|x - y\|$ – distance between x and y (usually Euclidean)
- inf – the **smallest possible value** over all valid transport plans (you can read this as "minimum cost plan").

Key takeaway

> **Wasserstein distance** between p_r and p_g is the **minimum average distance** you must move probability mass to transform the generator's distribution into the real one.

- Large $W(p_r, p_g)$: the generator is **far** from the real data.
- Small $W(p_r, p_g)$: the generator distribution is **close** to the real data.

This is the distance WGAN tries to minimize instead of the JS divergence used in classic GANs, which helps avoid vanishing gradients and mode collapse.

Wasserstein: from distance to loss

It is intractable to exhaust all the possible joint distributions in $\Pi(p_r, p_g)$ to compute $\inf_{\gamma \sim \Pi(p_r, p_g)}$. Thus the authors proposed a smart transformation of the formula based on the Kantorovich-Rubinstein duality to:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

where sup (supremum) is the opposite of inf (infimum); we want to measure the least upper bound or, in even simpler words, the maximum value.

Suppose this function f comes from a family of K-Lipschitz continuous functions, $\{f_w\}_{w \in W}$, parameterized by w. In the modified Wasserstein-GAN, the "discriminator" model is used to learn w to find a good f_w and the loss function is configured as measuring the Wasserstein distance between p_r and p_g.

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

Source: From GAN to WGAN

Wasserstein: from distance to trainable loss (1)

1. Continuous Wasserstein-1 distance

We want a **distance between distributions** p_r (real) and p_g (generator):

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big]$$

- $\gamma(x, y)$: **transport plan** (how much mass we move from x to y).
- $\Pi(p_r, p_g)$: all joint γ whose marginals are p_r and p_g.
- $\inf$: choose the **cheapest possible plan**.

> Problem: searching over all γ is intractable.

---

2. Kantorovich–Rubinstein dual (tractable form)

Using duality, the same distance can be written as a **max over functions**:

$$W(p_r, p_g) = \sup_{\|f\|_L \le 1} \Big(\mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]\Big)$$

- Supremum over all **1-Lipschitz** functions f (slope at most 1 everywhere).

I

3. WGAN loss (what we actually optimize)

Approximate f with a neural network **critic** $c_w(x)$ (parameters w):

$$L_{\text{WGAN}}(p_r, p_g) = \max_w \Big(\mathbb{E}_{x\sim p_r}[c_w(x)] - \mathbb{E}_{z\sim p_z}[c_w(g_\theta(z))]\Big)$$

Training objective:

$$\min_\theta \max_w \Big(\mathbb{E}_{x\sim p_r}[c_w(x)] - \mathbb{E}_{z\sim p_z}[c_w(g_\theta(z))]\Big)$$

- **Critic** (w): maximizes the gap → estimates $W(p_r, p_g)$.
- **Generator** (θ): minimizes the gap → makes p_g approach p_r.
- Constraint: c_w must be **1-Lipschitz** (enforced by weight clipping or gradient penalty).

W-Loss

W-Loss vs BCE Loss

BCE Loss	W-Loss
Discriminator outputs between 0 and 1	Critic outputs any number
$-[\mathbb{E}(\log(d(x))) + \mathbb{E}(1 - \log(d(g(z))))]$	$\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$

W-Loss helps with mode collapse and vanishing gradient problems

Condition on W-Loss

$$\min_{g} \max_{c} \mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$$

Needs to be 1-Lipschitz Continuous

1-L Enforcement: Weight Clipping

Weight clipping forces the weights of the critic to a fixed interval

Gradient descent to update weights

↓

Clip the critic's weights

Limits the learning ability of the critic

Source: coursera GANs specialization

WGAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

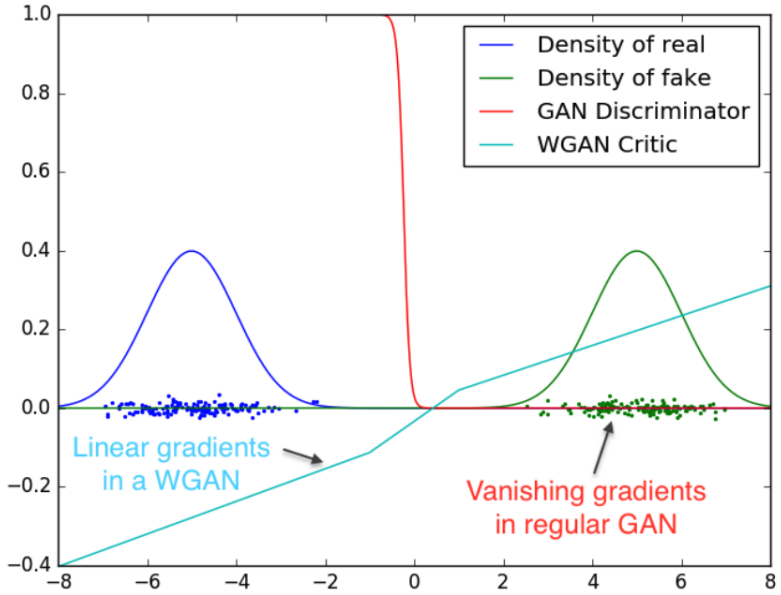


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

Source: Martin Arjovsky et al, 2017, Wasserstein GAN

W-Loss

W-Loss vs BCE Loss

BCE Loss	W-Loss
Discriminator outputs between 0 and 1	Critic outputs any number
$-[\mathbb{E}(\log(d(x))) + \mathbb{E}(1 - \log(d(g(z))))]$	$\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$

W-Loss helps with mode collapse and vanishing gradient problems

1-L Enforcement: Gradient Penalty

$$\min_g \max_c \; \mathbb{E}(c(x)) - \mathbb{E}(c(g(z))) - \lambda \text{reg}$$

Regularization of the critic's gradient

$$\mathbb{E}(||\nabla c(\hat{x})||_2 - 1)^2$$ Regularization term

$$\epsilon x + (1 - \epsilon) g(z)$$ Interpolation

Real Generated

Condition on W-Loss

$$\min_g \max_c \; \mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$$

Needs to be 1-Lipschitz Continuous

Source: coursera GANs specialization

Putting it all together

$$\min_{g} \max_{c} \boxed{\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))} \boxed{-\lambda \mathbb{E}(||\nabla c(\hat{x})||_2 - 1)^2}$$

Makes the GAN less prone to **mode collapse** and **vanishing gradient**

Tries to make the critic be 1-L Continuous, for the loss function to be **continuous and differentiable**

Source: coursera GANs specialization

WGAN with Gradient Penalty

Proposition 1. *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of $\max_{\|f\|_L \leq 1} \mathbb{E}_{y \sim \mathbb{P}_r}[f(y)] - \mathbb{E}_{x \sim \mathbb{P}_g}[f(x)]$. Let π be the optimal coupling between $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as the minimizer of: $W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\pi \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \pi} [\|x - y\|]$ where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\pi(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Then, if f^* is differentiable[‡], $\pi(x = y) = 0$[§], and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$, it holds that $\mathbb{P}_{(x,y) \sim \pi} \left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|} \right] = 1$.*

Corollary 1. *f^* has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.*

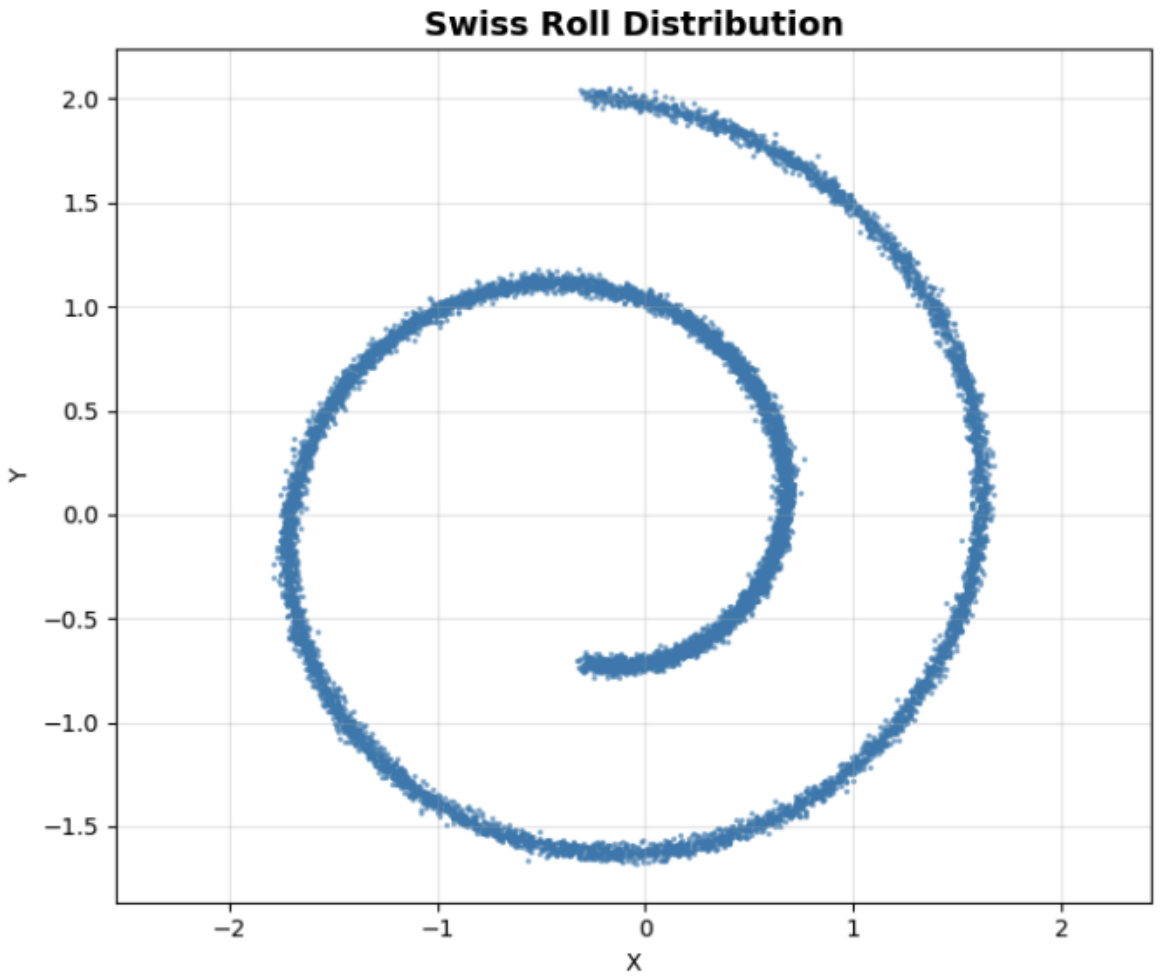
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

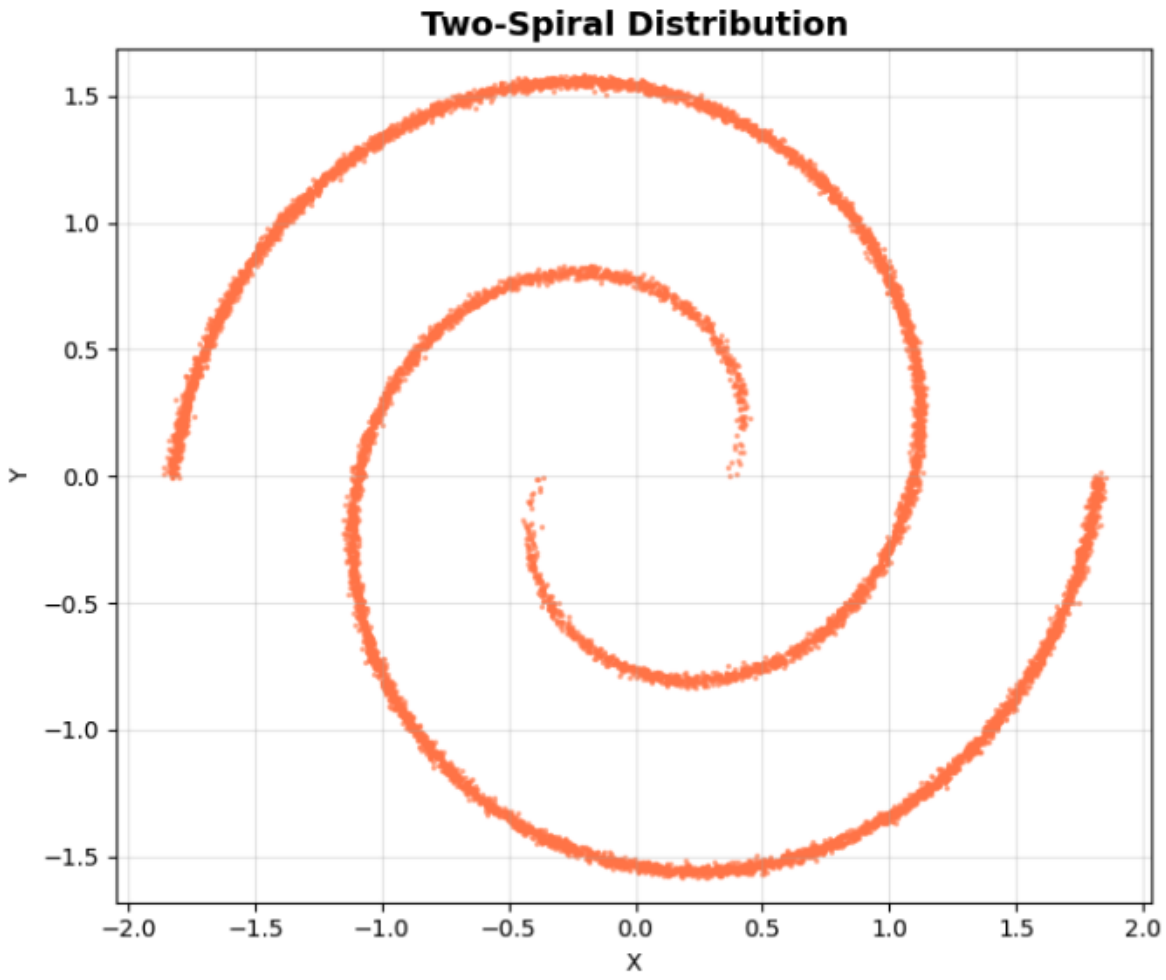
Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

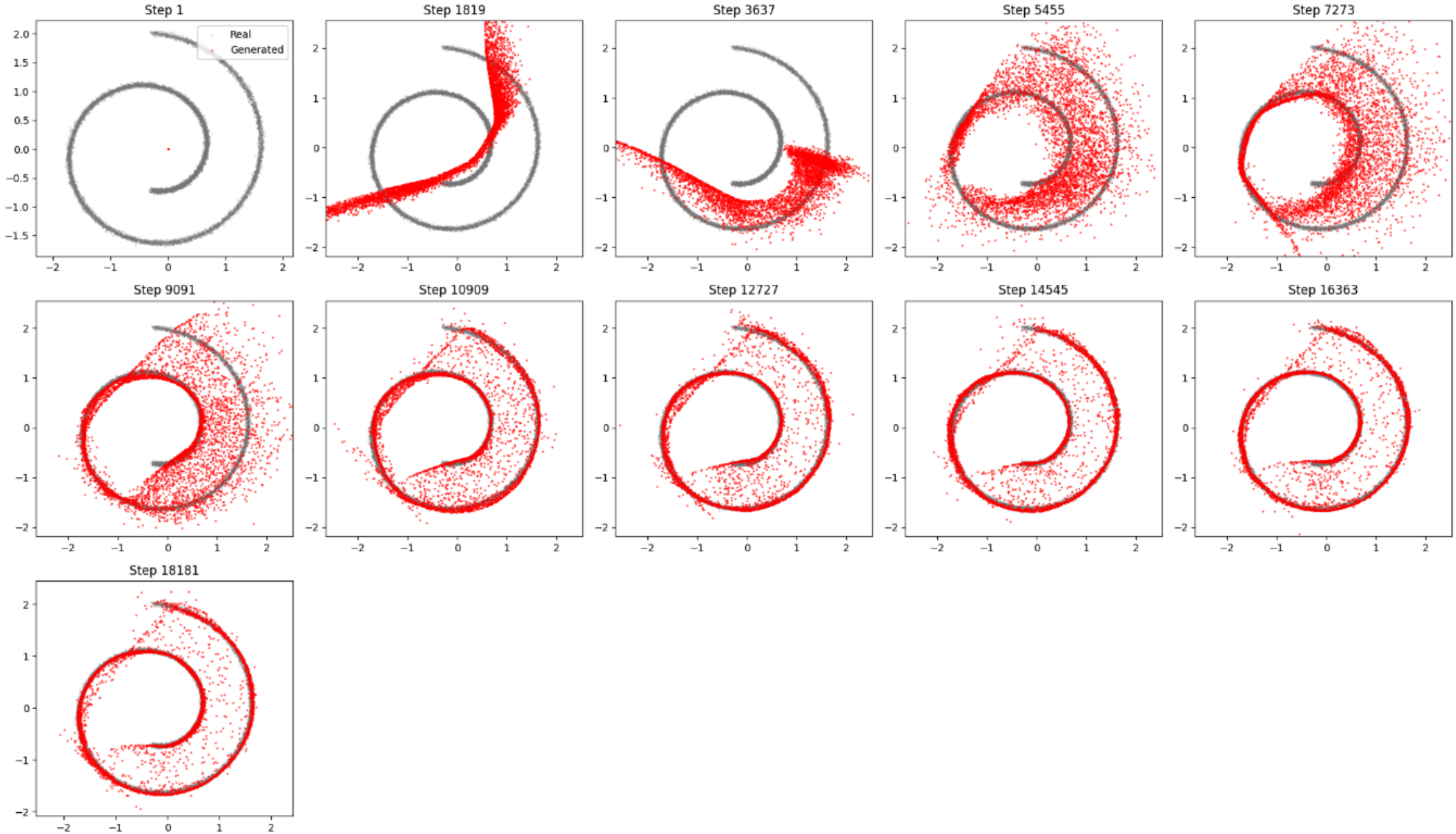
while θ has not converged **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Source: Gulrajani et al, 2017, Improved Training of Wasserstein GANs

Illustration (Vanilla GAN)






Swiss Roll GAN Training Progression
Step 1
Real
Generated
Step 1819
Step 3637
Step 5455
Step 7273
Step 9091
Step 10909
Step 12727
Step 14545
Step 16363
Step 18181

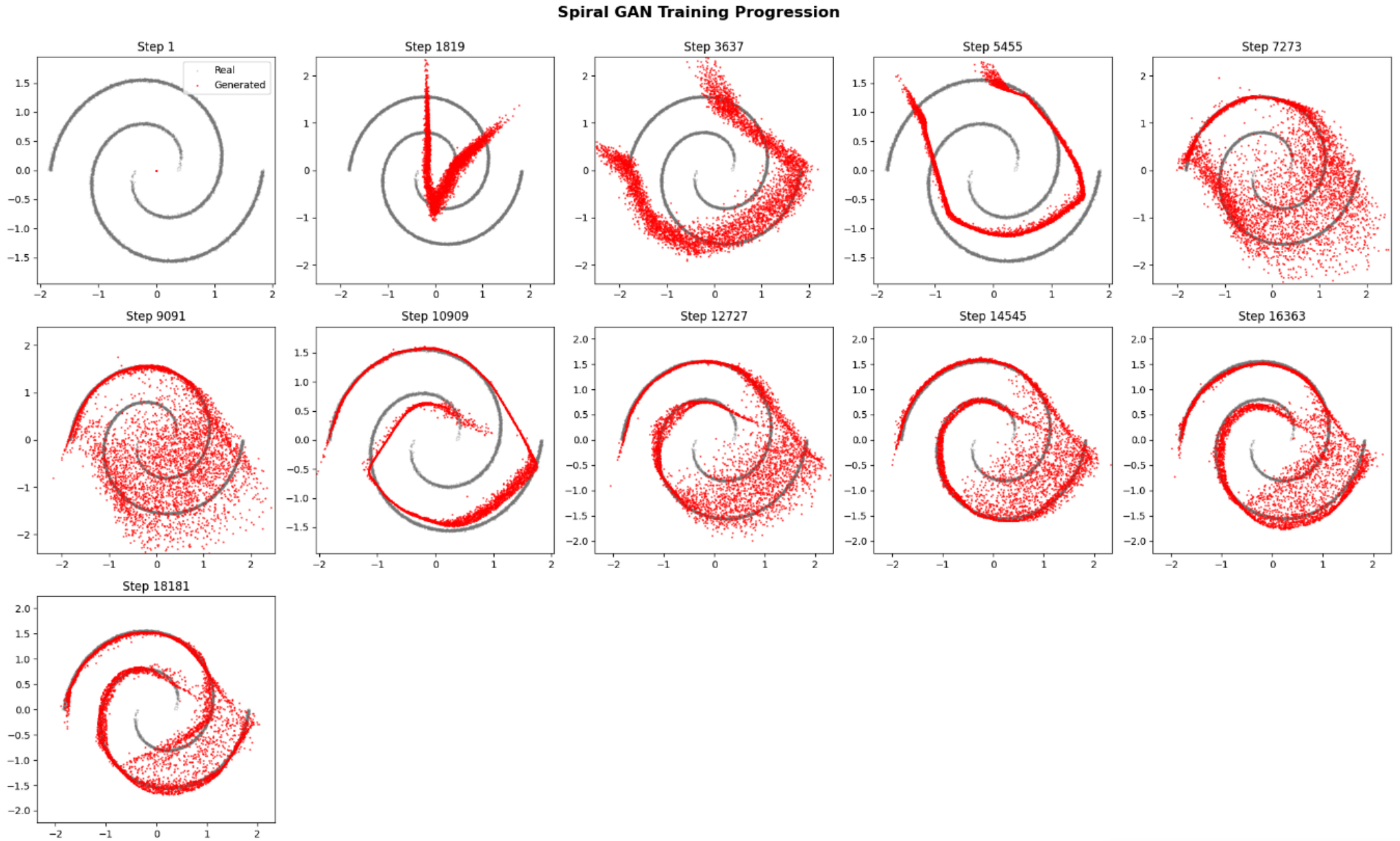
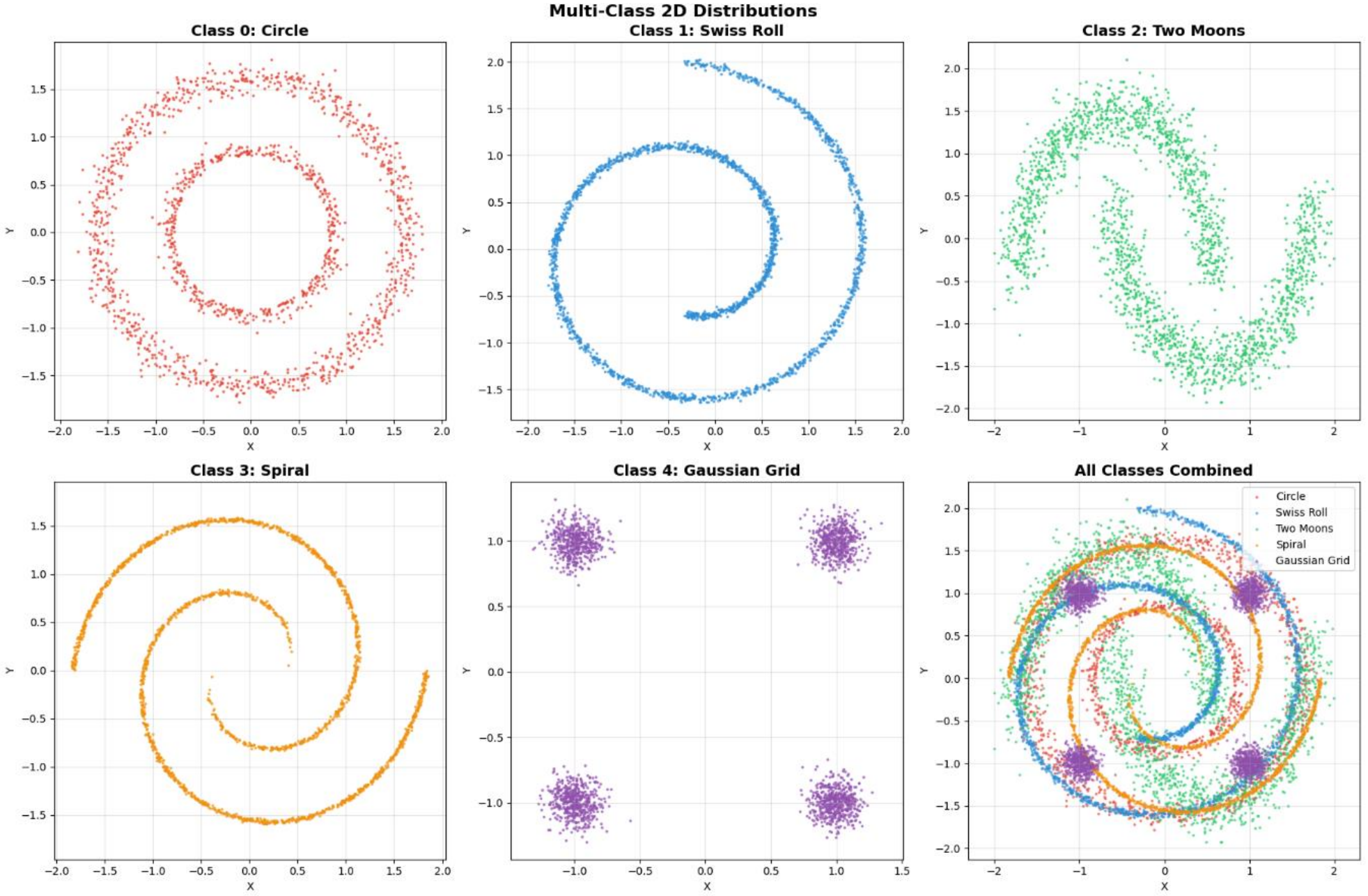
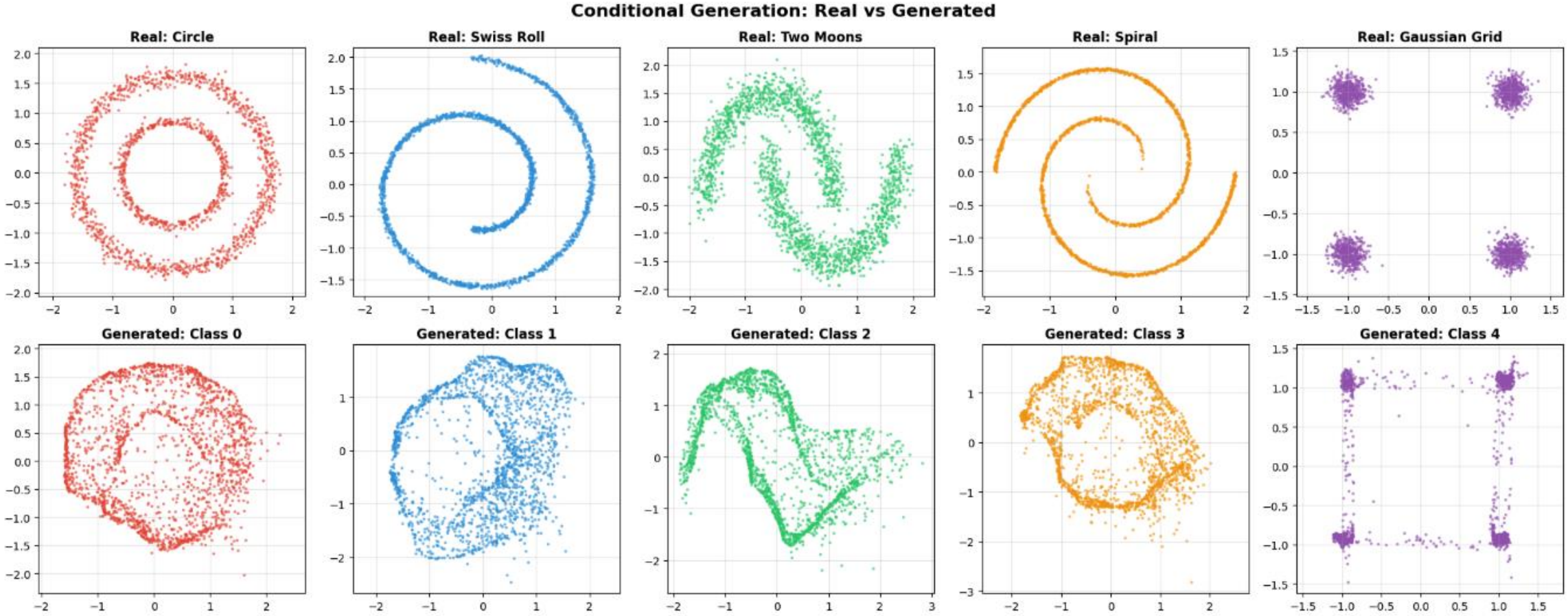
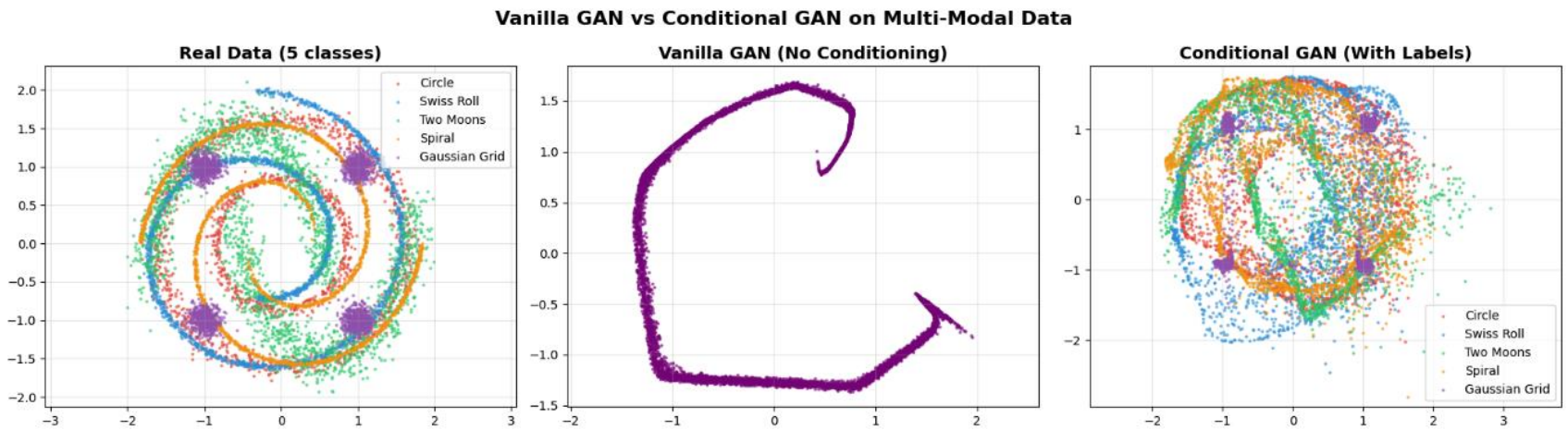

Spiral GAN Training Progression
Step 1
Real
Generated
Step 1819
Step 3637
Step 5455
Step 7273
Step 9091
Step 10909
Step 12727
Step 14545
Step 16363
Step 18181

Illustration (CGAN)




Conditional Generation: Real vs Generated
Real: Circle
Real: Swiss Roll
Real: Two Moons
Real: Spiral
Real: Gaussian Grid
Generated: Class 0
Generated: Class 1
Generated: Class 2
Generated: Class 3
Generated: Class 4


Vanilla GAN vs Conditional GAN on Multi-Modal Data
Real Data (5 classes)
Circle
Swiss Roll
Two Moons
Spiral
Gaussian Grid
Vanilla GAN (No Conditioning)
Conditional GAN (With Labels)
Circle
Swiss Roll
Two Moons
Spiral
Gaussian Grid

GANs for time series

- C-RNN-GAN
- Recurrent Conditional GAN
- TimeGAN
- References

C-RNN-GAN

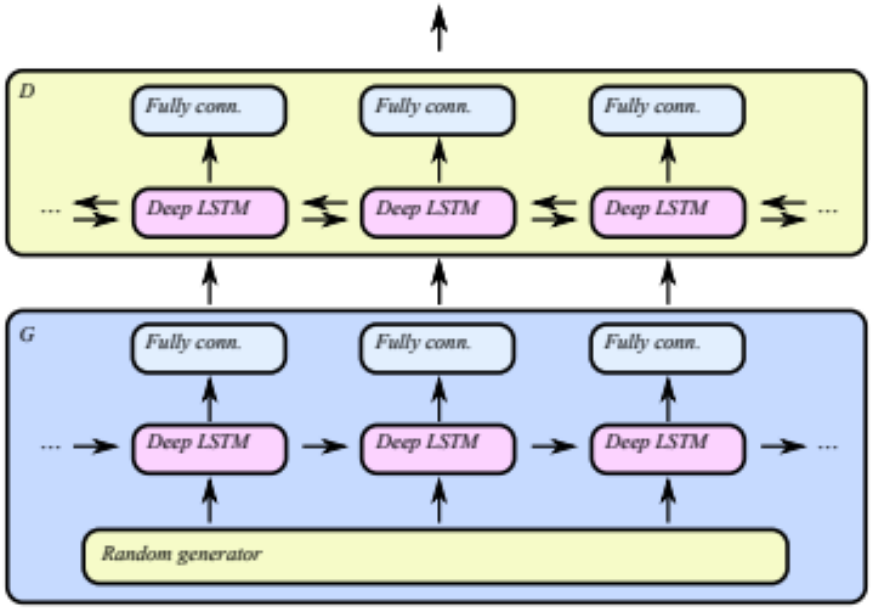
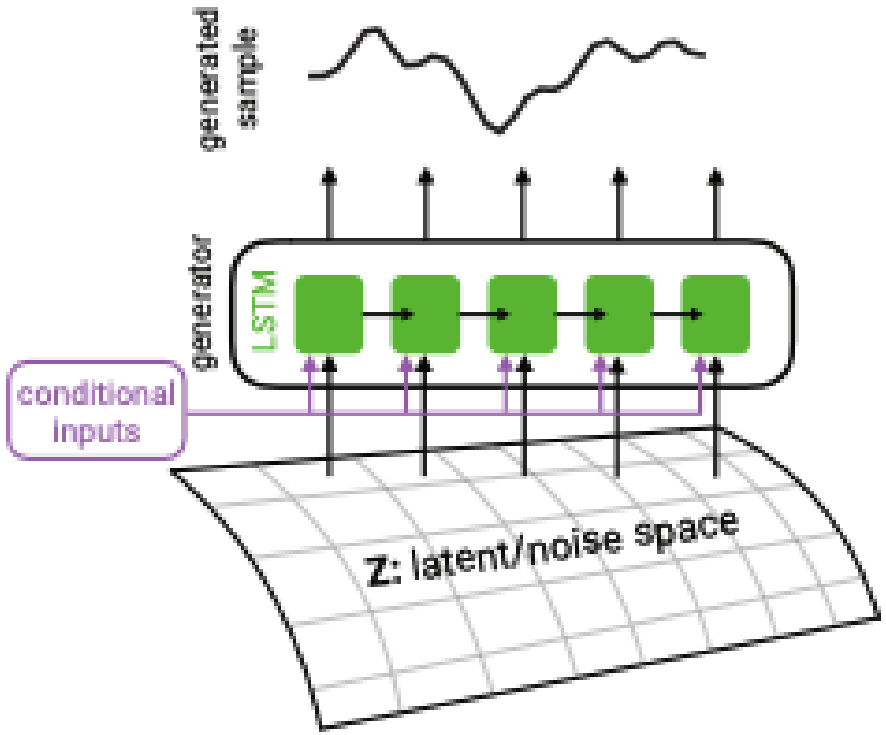


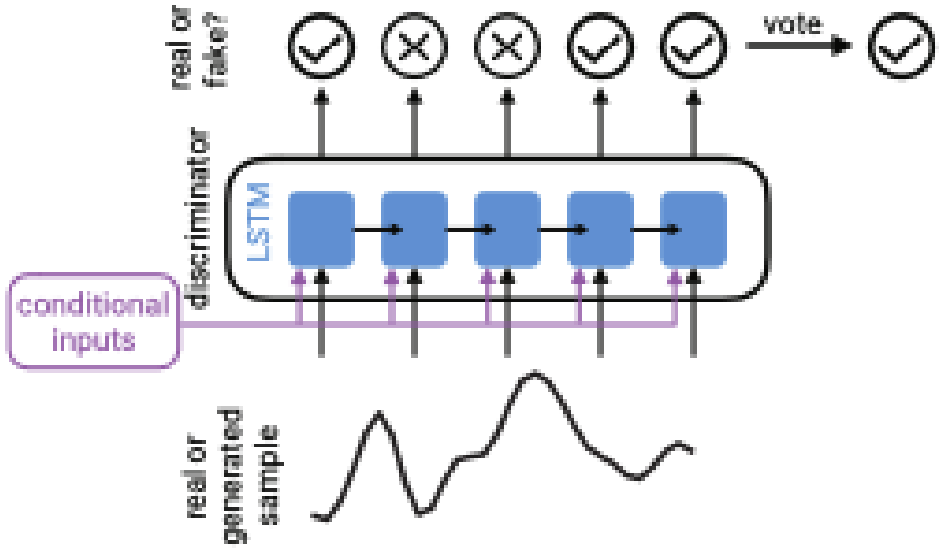
Figure 1: C-RNN-GAN. The generator (G) produces sequences of continuous data events. The discriminator (D) is trained to distinguish between real music data and generated data.

Source: Olof Mogren (2016), [C-RNN-GAN: Continuous recurrent neural networks with adversarial training]

Recurrent Conditional GANs



(a) The generator RNN takes a different random seed at each temporal input, and produces a synthetic signal. In the case of the RCGAN, it also takes an additional input on each time step that conditions the output.



(b) The discriminator RNN takes real/synthetic sequences and produces a classification into real/synthetic for each time step. In the case of the RCGAN, it also takes an additional input on each time step that conditions the output.

Figure 1: Architecture of Recurrent GAN and Conditional Recurrent GAN models.

Stephanie Hyland et al (2017), Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs

TimeGAN

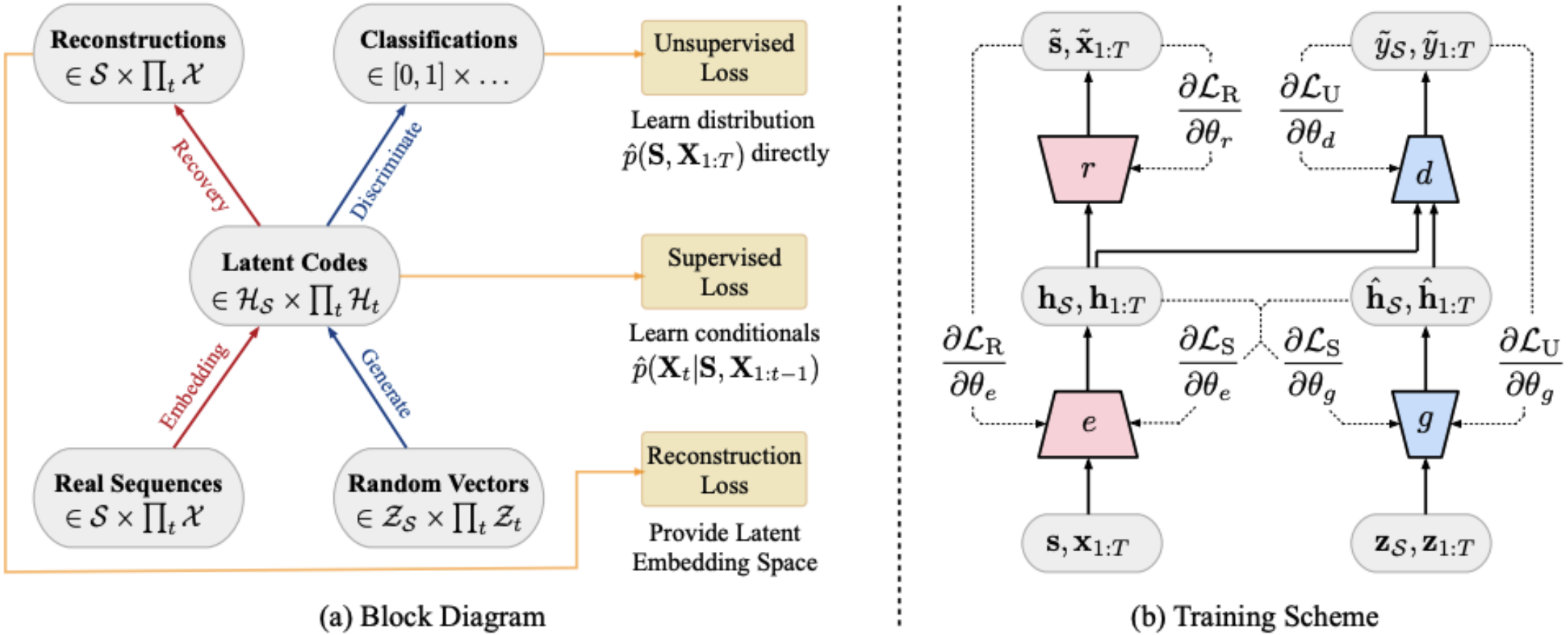


Figure 1: (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients.

Jinsung Yoon et al (2019), Time Series Generative Adversarial Networks

TimeGAN

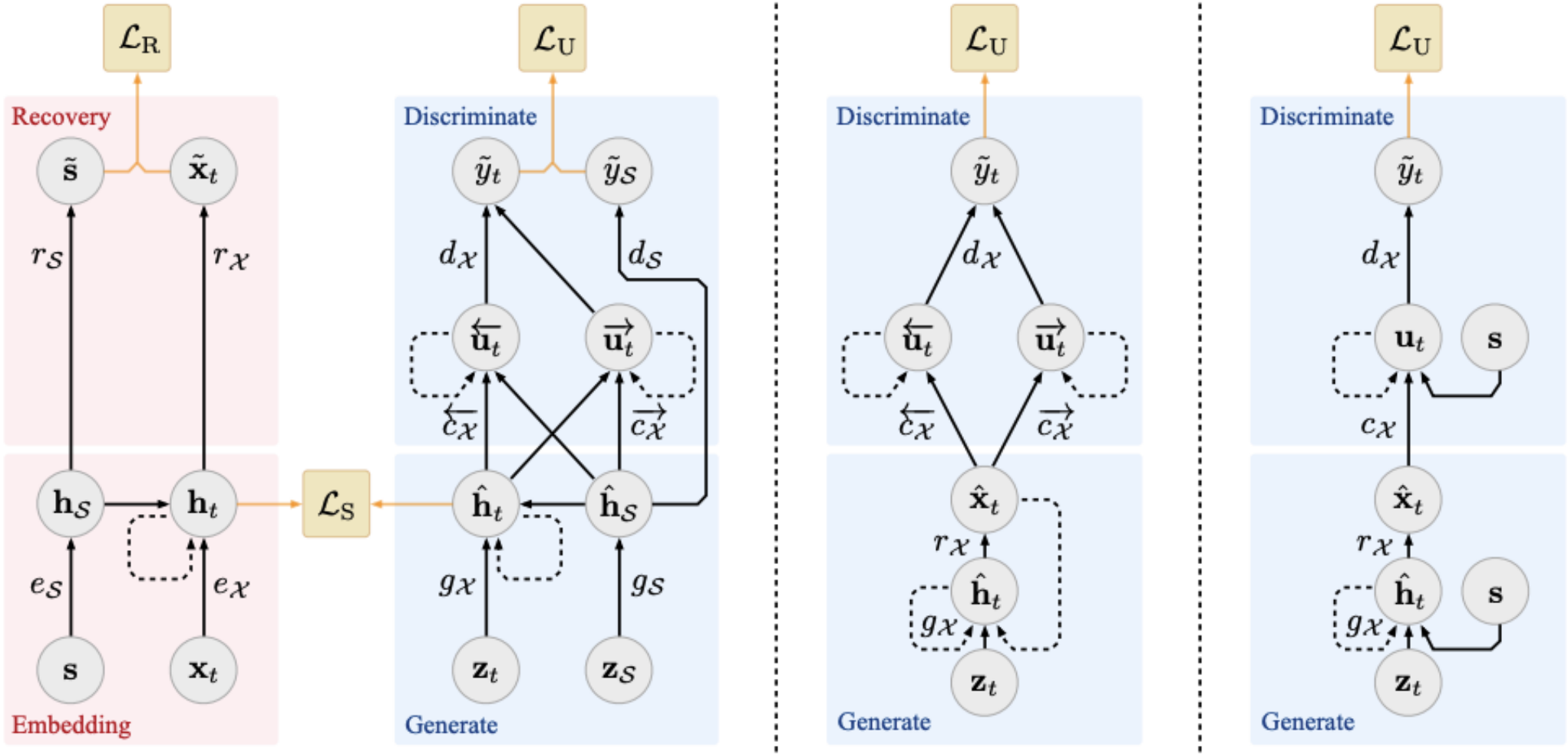


Figure 2: (a) TimeGAN instantiated with RNNs, (b) C-RNN-GAN, and (c) RCGAN. Solid lines denote function application, dashed lines denote recurrence, and orange lines indicate loss computation.

Jinsung Yoon et al (2019), Time Series Generative Adversarial Networks

References

- Blog posts
  - From GAN to WGAN
- Courses:
  - Coursera GANs Specialization
  - Stanford CS236, Deep Generative Models

- Papers
  - Ian Goodfellow et al, 2014, Generative Adversarial Nets
  - Mehdi Mirza, Simon Osindero, 2014, Conditional Generative Adversarial Nets
  - Martin Arjovsky et al, 2017, Wasserstein GAN.
  - Olof Mogren (2016), C-RNN-GAN: Continuous recurrent neural networks with adversarial training
  - Stephanie Hyland et al (2017), Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs
  - Jinsung Yoon et al (2019), Time Series Generative Adversarial Networks

- Books
  - Jakub M. Tomczak, 2022, Deep Generative Modeling , Springer
  - Kevin Murphy, 2023, Probabilistic Machine learning: Advanced topics (part IV: Generation)