Machine Learning Course - CS-433

Generative Models

Dec 11, 2024

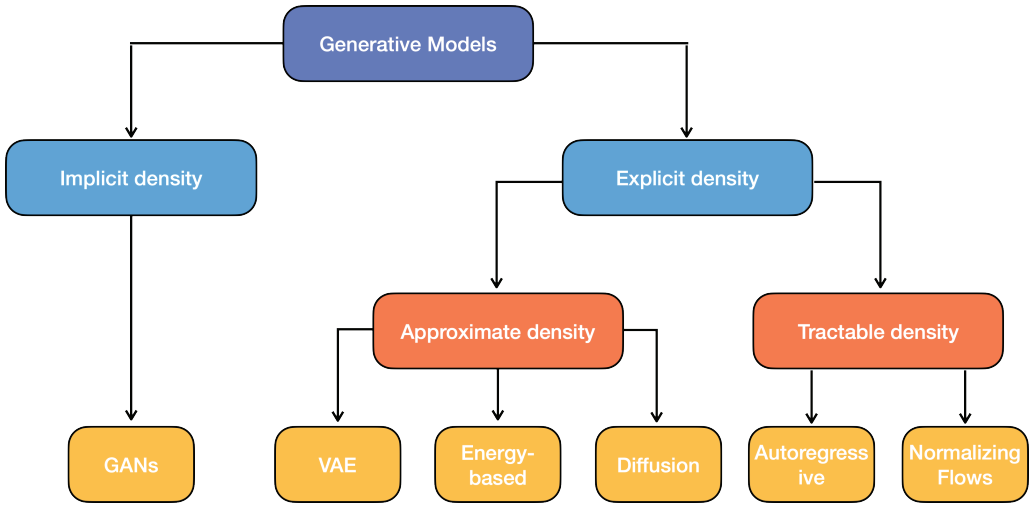
Martin Jaggi
Last updated on: December 10, 2024

credits to Tatjana Chavdarova, Francesco D'Angelo and Lara Orlandic

Generative Models

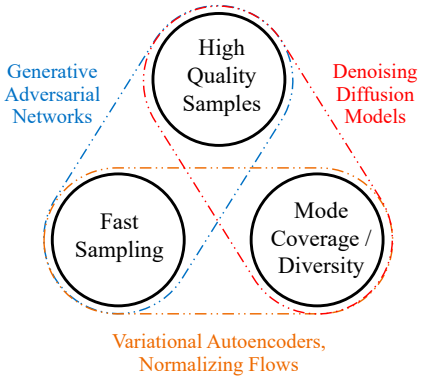
Given a data sample x, a *discriminative* model aims at predicting its label y, hence it models the conditional distribution $p(y|x)$. Generative models instead model the distribution $p(x)$ defined over the datapoints x. Generative models models can be differentiated into two categories: *explicit* or *implicit*. In the former case, we explicitly model the probability distribution of the data, in the latter we aim at generating samples according to it.

Taxonomy. The following figure depicts the taxonomy of the existing generative methods:



- **Implicit density models**, epitomized by GANs, excel in generating high-quality, realistic samples but are often challenging to train due to mode collapse, where the model fails to capture the diversity of the data distribution.
- **Tractable density models**, such as Autoregressive models and Normalizing Flows, allow for exact likelihood computation, which can be beneficial for tasks that require likelihood evaluation and interpretability. Autoregressive models, which generate data sequentially, can be slow in sampling due to their nature. Normalizing Flows, however, enable more efficient sampling and inference by using invertible transformations, although they can be computationally intensive to train.
- **Approximate models**, including VAEs and Energy-based models, use a different strategy. VAEs, for example, optimize a lower bound on the likelihood, allowing for easier and more stable training than GANs but often resulting in less sharp samples.
- **Diffusion models** represent a hybrid approach. They iteratively convert noise into samples via a reverse Markov process and have shown promising results in generating high-quality images and audio. However, they can be slow at sampling time.

In summary, the choice among generative models depends on the specific application and the trade-off between sample quality, sampling speed, and diversity [Xiao et al., 2021].



(The above figure is taken from [Xiao et al., 2021])

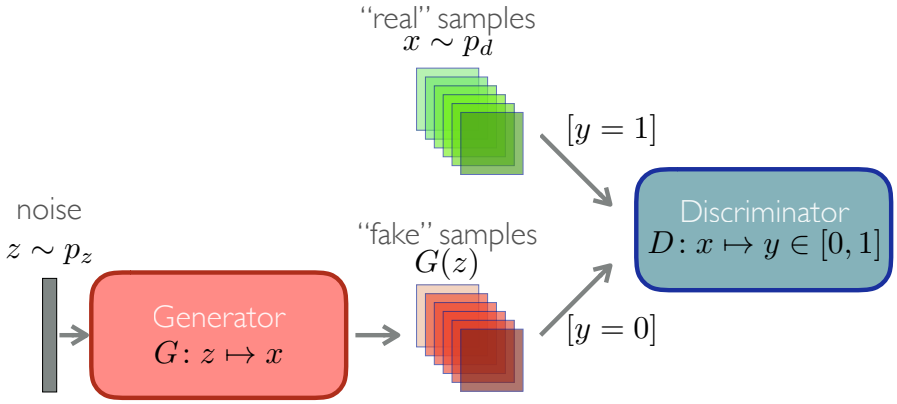
Generative Adversarial Networks

Generative Adversarial Networks (GANs, Goodfellow et al. [2014]) are a family of implicit generative algorithms that are fast to sample from. The GAN framework is composed of two models: a **generator** G with paramters $\boldsymbol{\theta}$ and a **discriminator** D with parameters $\boldsymbol{\varphi}$. The former lears to generate samples from the data distribution p_d, while the latter learns to distinguish between real and fake samples. Therefore, contrary to single-objective minimization $f\colon \mathcal{X} \to \mathbb{R}$, the optimization of a GAN is formulated as a differentiable two-player game where the generator G, and the discriminator D, aim at minimizing their own cost function $\mathcal{L}^{\boldsymbol{\theta}}$ and $\mathcal{L}^{\boldsymbol{\varphi}}$, respectively, as follows:

$$\begin{aligned} \boldsymbol{\theta}^\star &\in \arg\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}^\star) \\ \boldsymbol{\varphi}^\star &\in \arg\min_{\boldsymbol{\varphi}\in\Phi} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi})\,. \end{aligned} \tag{2P-G}$$

When $\mathcal{L}^{\boldsymbol{\theta}} = -\mathcal{L}^{\boldsymbol{\varphi}}$ the game is called a zero-sum game and (2P-G) is a minmax problem.

1. The discriminator "distinguishes"
   real vs. *fake* samples:

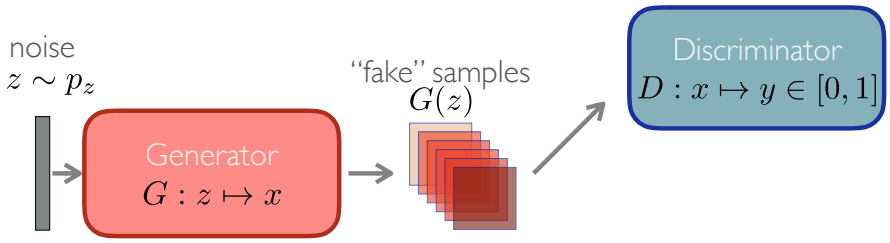


   p_z known "noise" distribution, *e.g.* $\mathcal{N}(0, 1)$

   p_d the real data distribution

   D mapping $D: x \mapsto y \in [0, 1]$,
   where y is an estimated probability that $x \sim p_d$

2. The generator aims at fooling
   the discriminator that its samples are real:



   G mapping $G: z \mapsto x$, such that if $z \sim p_z$,
   then hopefully $x \sim p_d$

   p_g the "fake" data distribution

3. Objective

$$\min_G \max_D \mathop{\mathbb{E}}_{x\sim p_d}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z}[\log(1-D(G(z)))]$$

- Loss for D: distinguish between $x \sim p_g$ and $x \sim p_d$ (binary classification):

$$\mathscr{L}_D(G, D) = \max_D \mathop{\mathbb{E}}_{x\sim p_d}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_z}[\log(1-D(G(z)))]$$

- Loss for G: fool D that $G(z) \sim p_d$:

$$\begin{aligned}\mathscr{L}_G(G, D) &= \min_G \mathop{\mathbb{E}}_{z\sim p_z}[\log(1 - D(G(z)))] \\ &:= \text{(in practice)} \max_G \mathop{\mathbb{E}}_{z\sim p_z}[\log(D(G(z)))]\end{aligned}$$

4. **Theoretical Solution:** The optimum is reached when $p_g = p_d$ and the optimal value is $-\log 4$ (proof in function space, see next slides).

Alternating–GAN algorithm

In practice, G and D are parametrized models (typically neural networks), and are optimized with gradient based methods.

- G: deep neural network $G(\mathbf{z}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$
- D: deep neural network $D(\mathbf{x}; \boldsymbol{\varphi})$ with parameters $\boldsymbol{\varphi}$

In most GAN implementations G and D have different losses $\mathcal{L}^{\boldsymbol{\theta}}$ and $\mathcal{L}^{\boldsymbol{\varphi}}$[a], resp. In the following, we present the most commonly used algorithm for training GANs.

Algorithm 1 *alternating*–GAN

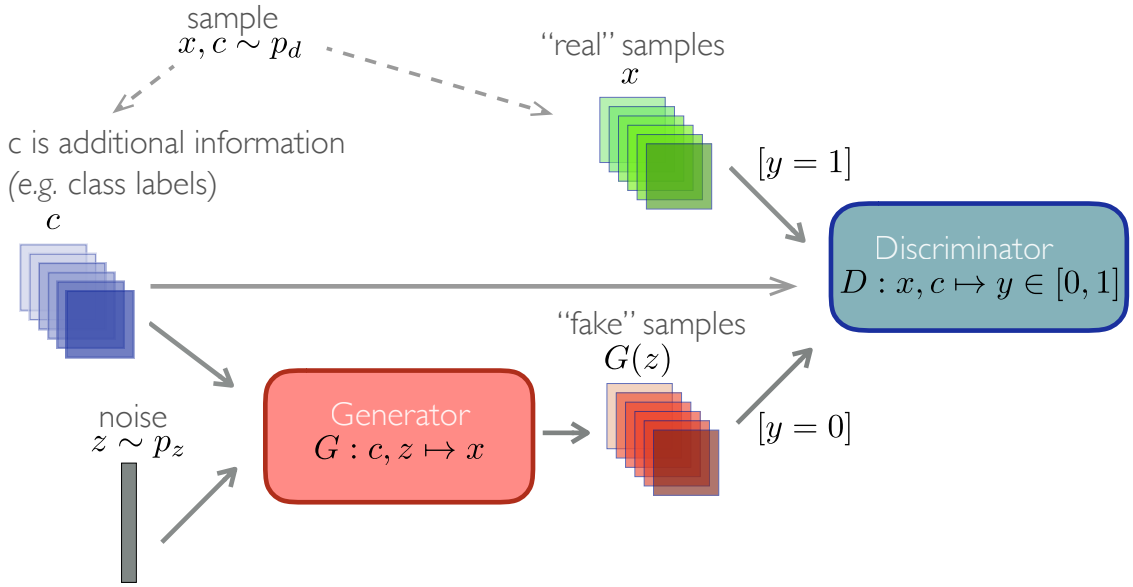
Input: dataset $\mathcal{D}$, known distribution p_z, learning rate η, generator loss $\mathcal{L}^{\boldsymbol{\theta}}$, discriminator loss $\mathcal{L}^{\boldsymbol{\varphi}}$
Initialize: $D(\cdot; \boldsymbol{\varphi})$, $G(\cdot; \boldsymbol{\theta})$
for $t = 1$ **to** T **do**
  $\mathbf{x} \sim p_x$ {sample real data}
  $\mathbf{z} \sim p_z$ {sample noise from p_z}
  $\boldsymbol{\varphi} = \boldsymbol{\varphi} - \eta \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathbf{x}, \mathbf{z})$ {update D}
  $\mathbf{z} \sim p_z$ {sample noise from p_z}
  $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathbf{z})$ {update G}
end for
Output: $\boldsymbol{\theta}, \boldsymbol{\varphi}$

For simplicity, in Alg.1 we used gradient descent, however in practice GANs are often optimized using Adam [Kingma and Ba, 2015], and developing well performing minimax optimization methods is an active research area.

[a] Although we *minimize* both the loses, this notation generalizes the zero-sum game, as the latter holds when $\mathcal{L}^{\boldsymbol{\theta}} = -\mathcal{L}^{\boldsymbol{\varphi}} := \mathcal{L}$.

Conditional GAN – (CGAN)

Many applications require generative model of a conditional probability distribution (*e.g.* "in-painting", segmentation, predicting the next frame of a video *etc.*).
GANs can also generate samples of conditional distribution, called Conditional GAN (CGAN) [Mirza and Osindero, 2014]. In CGANs both the Generator and the Discriminator are conditioned during training by some additional information, typically the class labels (but could also be images *e.g.* auto-generated edges of an image, conditioning on non-occluded portion–so as to generate the occluded part of the image *etc*).



When conditioning on the class labels, typically one-hot vector representation of the class labels is used (empirically shown to perform better).

GAN architectures for images

In the context of images synthesis, [Radford et al., 2015] propose specific architectures of the two models, named Deep Convolutional GANs – *DCGAN*. Notably, the generator uses *transposed convolutional layer* (*a.k.a.* fractionally strided convolutions), also informally called "Deconvolution layers" (wrongfully). Simlest way to explain these is that they "swap" the forward and the backward passes of a convolution layer: the forward transposed convolution operation can be thought of as the gradient of some convolution with respect to its input, which is usually how transposed convolutions are also implemented in practice.

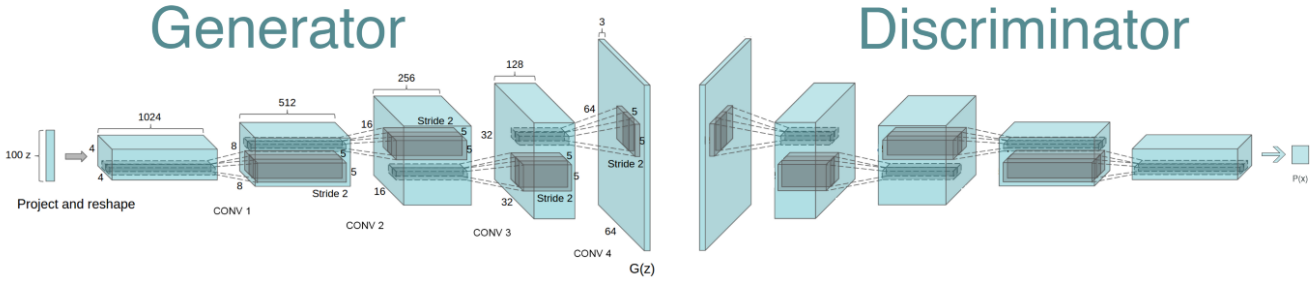
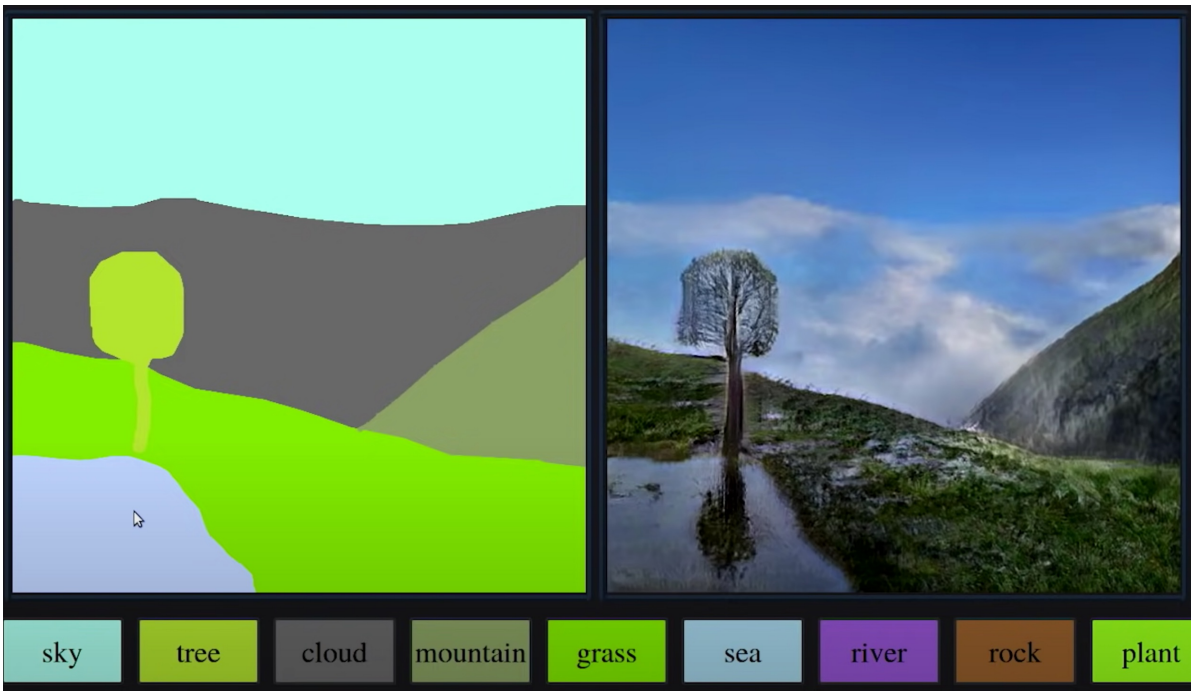


Image to image translation

Isola et al. (2016), (CGAN): automatically detected edges ↦ handbags

Isola et al. (2016), generalization of the CGAN model trained on edges ↦ photo (see previous figure) to human-drawn sketches

Conditional Generation of Images

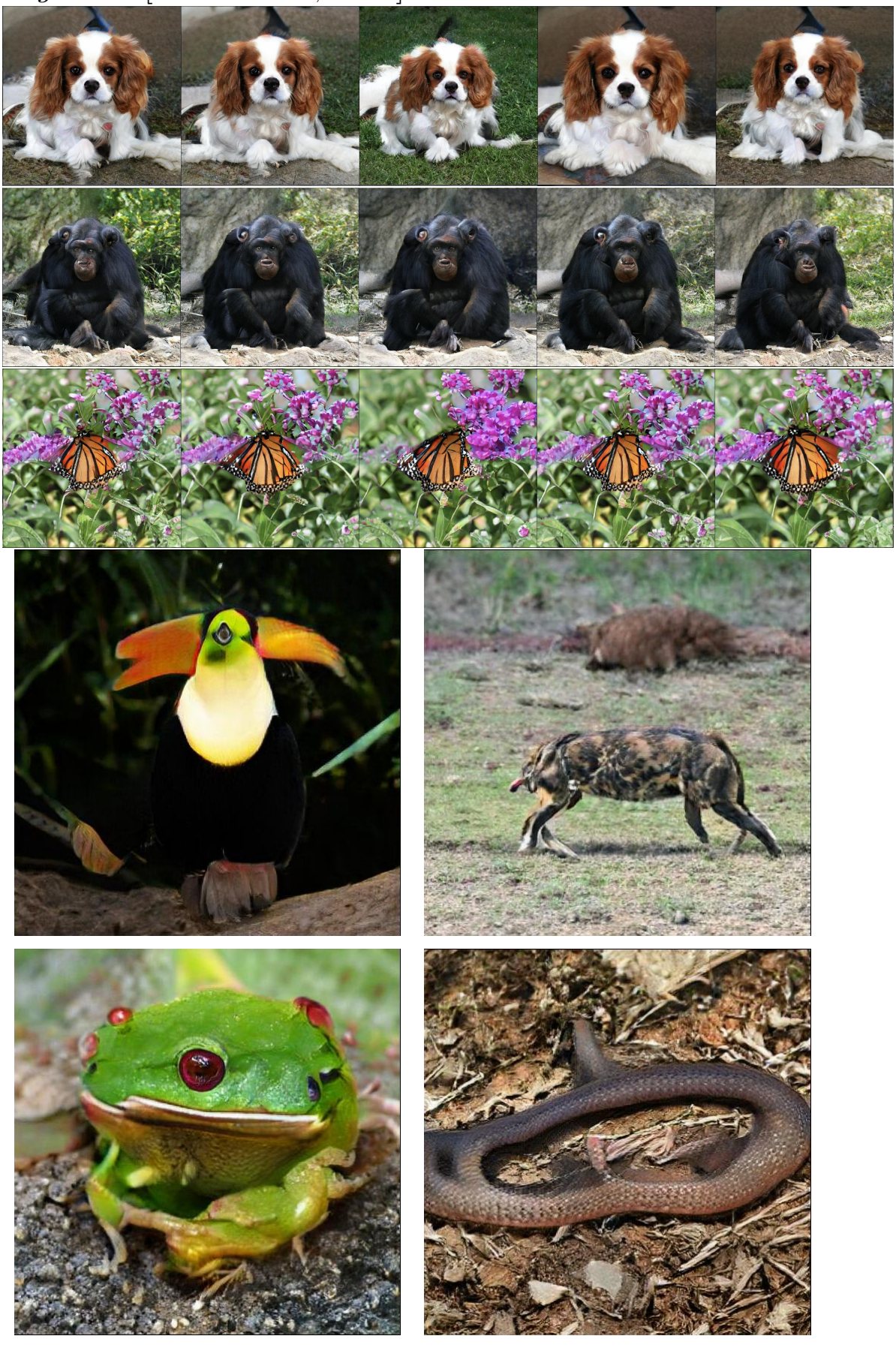


(Picture from blogs.nvidia.com/blog/2021/11/22/gaugan2-ai-art-demo/.)

"A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019, https://arxiv.org/abs/1812.04948

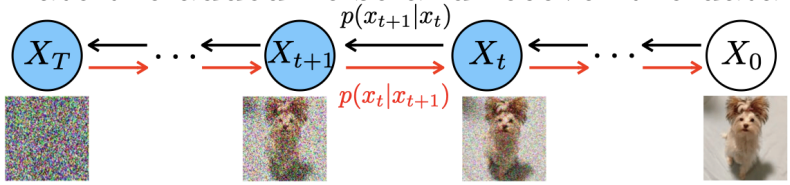
GAN generated images

512×512 samples from the class-conditional *BigGAN* [Brock et al., 2019].



Diffusion models

Diffusion models are another class of generative models that have gained popularity in recent years, both for their enhanced performance and implementation in AI image generators like DALL-E 3, Stable Diffusion, and Midjourney. Unlike GANs, which train two separate models for data generation and discrimination, Diffusion Models work by progressively adding noise to input data and training one single model to estimate the added noise and recover the data.



(The following content is based on this lecture)

To formalize this process, consider a Markov chain with $X_0 \sim p_0$ and a forward transition such that $X_{t+1} \sim p(\cdot|X_t)$ then we call **forward decomposition**:

$$p(x_{0:T}) = p_0(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t)\,.$$

where at each step, the marginal distribution of X_t satisfies:

$$p(x_t) = \int p(x_t|x_{t-1})p(x_{t-1})\,\mathrm{d}x_{t-1}\,.$$

The backward transition can be obtained with Bayes'rule $p(x_t|x_{t+1}) = p(x_{t+1}|x_t)p(x_t)/p(x_{t+1})$, and we call **backward decomposition**:

$$p(x_{0:T}) = p_T(x_T) \prod_{t=0}^{T-1} p(x_t|x_{t+1})\,.$$

A generative model can be constructed sampling from $p(x_{0:T})$ using **ancestral sampling**:

$$\begin{aligned}&\texttt{Sample} \quad X_T \sim p_T(\cdot) \quad \texttt{then:}\\ &\texttt{for } k = T-1, \dots, 0:\\ &\qquad \texttt{Sample} \quad X_t \sim p(\cdot|X_{t+1})\end{aligned}$$

- We consider $p_0 = p_{\text{data}} \in \mathcal{P}(\mathbb{R}^d)$
- We chose the forward transition to be Gaussian $p(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; \alpha x_t, (1-\alpha^2)\mathbb{I}_d)$
- The conditional $p(x_t|x_0)$ is also Gaussian $\mathcal{N}(x_t; \alpha^t x_0, (1-\alpha^{2t})\mathbb{I}_d)$
- The Markov chain converges to $p_{\text{ref}} = \mathcal{N}(x; 0, \mathbb{I}_d)$ for $T \to \infty$.
- Therefore for T large enough $p_T(x) \approx p_{\text{ref}}(x)$.
- To generate samples we use ancestral sampling replacing p_T with p_{ref}.

Key problem: the backward transition $p(x_t|x_{t+1})$ can't be computed, we need an approximation!

Using a Taylor expansion we can approximate the backward transition:

$$\begin{aligned}&p(x_t|x_{t+1}) = p(x_{t+1}|x_t)\exp[\log p(x_t) - \log p(x_{t+1})]\\ &\approx \mathcal{N}(x_t; (2-\alpha)x_{t+1} + (1-\alpha^2)\underbrace{\nabla \log p(x_{t+1})}_{\textbf{Score}}, (1-\alpha^2)\mathbb{I}_d)\,.\end{aligned}$$

Denoising score matching

The Stein **Score** is analytically intractable, but we can approximate it using a **neural network** $s_\theta(x_t)$. Surprisingly, we can learn the score solving a simple regression problem via the Denoising Score Matching (DSM) loss:

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta) \coloneqq \frac{1}{2}\mathbb{E}_{X_t}[\|\nabla \log p(X_t) - s_\theta(X_t)\|^2] \tag{DSM-Loss}$$

This problem is still intractable, we do not have access to the marginal $p(x_t)$ at each step but we can manipulate it:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathrm{C}_1 + \frac{1}{2}\mathbb{E}_{X_t}[\|s_\theta(X_t)\|^2] - \mathbb{E}_{X_t}[\nabla \log p(X_t)^\top s_\theta(X_t)] \\ &= \mathrm{C}_1 + \frac{1}{2}\mathbb{E}_{X_t}[\|s_\theta(X_t)\|^2] - \int \nabla \log p(X_t)^\top s_\theta(X_t) p(X_t)\,\mathrm{d}x_t\end{aligned}$$

Using that $p(x_t) = \int p_0(x_0)p(x_t|x_0)\,\mathrm{d}x_0$ we get:

$$\nabla \log p(X_t) = \mathbb{E}_{X_0|X_t}[\nabla \log p(X_t|X_0)]\,.$$

Substituding in the DSM-loss:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathrm{C}_1 + \frac{1}{2}\mathbb{E}_{X_t}[\|s_\theta(X_t)\|^2] - \mathbb{E}_{X_0}\mathbb{E}_{X_t|X_0}[\nabla \log p(X_t|X_0)^\top s_\theta(X_t)] \\ &= \mathrm{C}_1 + \frac{1}{2}\mathbb{E}_{X_0}\mathbb{E}_{X_t|X_0}[\|s_\theta(X_t) - \nabla \log p(X_t|X_0)\|^2] \\ &\quad - \frac{1}{2}\underbrace{\mathbb{E}_{X_0}\mathbb{E}_{X_t|X_0}[\|\nabla \log p(X_t|X_0)\|^2]}_{\text{const.}} \\ &= \mathrm{C}_2 + \frac{1}{2}\mathbb{E}_{X_0}\mathbb{E}_{X_t|X_0}[\|s_\theta(X_t) - \nabla \log p(X_t|X_0)\|^2]\end{aligned}$$

In practice we estimate all the scores simultaneously i.e. $s_{\theta^\star}(t, X_t)$ such that:

$$\theta^\star = \arg\min_\theta \frac{1}{2} \sum_{t=1}^{T} \mathbb{E}_{X_0} \mathbb{E}_{X_t|X_0} [\| s_\theta(t, X_t) - \nabla \log p(X_t|X_0) \|^2]$$

Recalling now our choice of forward transition for which $p(x_t|x_0) = \mathcal{N}(x_t; \alpha^t x_0, (1 - \alpha^{2t}) \mathbb{I}_d)$:

- $\nabla \log p(x_t|x_0) = -\frac{(x_t - \alpha^t x_0)}{(1-\alpha^{2t})}$
- $X_t = \alpha^t X_0 + \sqrt{1 - \alpha^{2t}} \xi_t \ \xi_t \sim \mathcal{N}(0, \mathbb{I}_d)$
- $\nabla \log p(X_t|X_0) = -\frac{\xi_t}{\sqrt{1-\alpha^{2t}}}$

If we parameterize the score function as $s_\theta(t, X_t) = -\frac{\hat{\xi}_\theta(t, X_t)}{\sqrt{1-\alpha^{2t}}}$ then the problem is equivalent to:

$$\theta^\star = \arg\min_\theta \frac{1}{2} \sum_{k=1}^{T} \mathbb{E}_{X_0} \mathbb{E}_{X_t|X_0} [\| \hat{\xi}_\theta(t, X_t) - \xi_t \|^2]$$

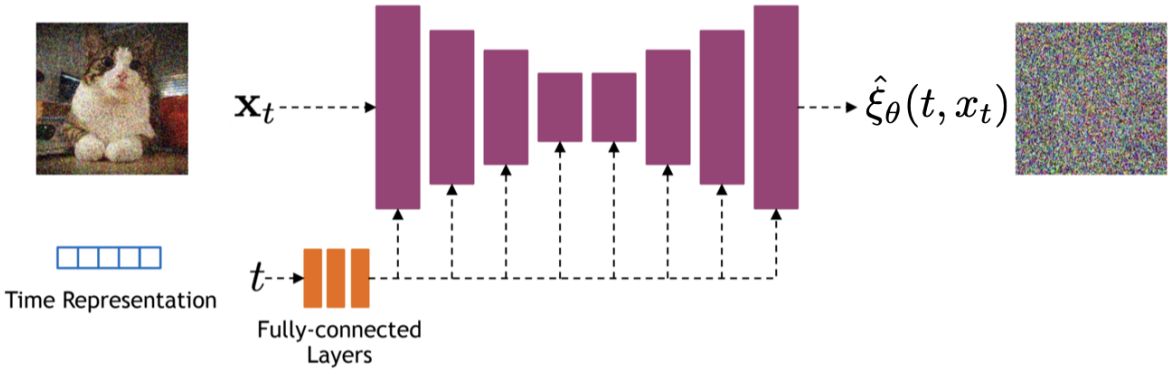
which has the illuminating interpretation of learning to predict the added noise ξ_t from the noised data X_t.

Finally, we can generate new samples from the backward process by sampling $X_T \sim p_{\text{ref}}$ and using the approximation of the backward transition and the learned score to perform ancestral sampling:

$$X_t = (2-\alpha) X_{t+1} + (1-\alpha^2) s_{\theta^\star}(t+1, X_{t+1}) + \sqrt{1 - \alpha^2} \xi_{t+1} \quad \xi_{t+1} \sim \mathcal{N}(0, \mathbb{I}_d).$$

Implementation

Diffusion models often use a U-net architecture with ResNet blocks and self attention layers to represent the score function $\hat{\xi}_\theta(t, X_t)$.



(The above figure is taken from this tutorial)

The time step can be represented with sinusoidal positional encodings as done for transformers or with random Fourirer features. The time features are then fed to the residual blocks using either spatial addition or adaptive group normalization layers Dhariwal and Nichol [2021].

Training and Sampling

Algorithm 2 Training

Input: Data distribution p_0, α
Output: Learned parameters θ^*
Initialize: Parameters θ
repeat
  Sample initial data $X_0 \sim p_0$
  Sample time step $t \sim \text{Uniform}(\{1, \dots, T\})$
  Sample noise $\xi_t \sim \mathcal{N}(0, \mathbb{I}_d)$
  Update θ using gradient descent step on:
    $\nabla_\theta \frac{1}{2} \mathbb{E}_{X_0} \mathbb{E}_{X_t|X_0} \left[\left\| s_\theta(t, \alpha^t X_0 + \sqrt{1-\alpha^{2t}}\xi_t) - \xi_t \right\|^2 \right]$
until convergence

Algorithm 3 Sampling

Input: Learned parameters θ^*, p_{ref}, α
Output: Generated sample X_0
Sample $X_T \sim p_{\text{ref}}$
for $t = T - 1$ **to** 0 **do**
  Sample noise $\xi_{t+1} \sim \mathcal{N}(0, \mathbb{I}_d)$:

$$X_t = (2-\alpha)X_{t+1} + (1-\alpha^2)s_{\theta^*}(t+1, X_{t+1}) + \sqrt{1-\alpha^2}\xi_{t+1}$$

end for

Return x_0

Exponential moving average

The training of the U-net via Denoising Score Matching can be very unstable. To regularize it, usually an exponential moving average of the weights is used along the training:

$$\bar{\theta}_{n+1} = (1-\beta)\bar{\theta}_n + \beta\theta_n\,.$$

Where the parameter β regulates the rate of forgetting of the initial conditions.

Conditional training

Has we have seen already for GANs, it is often useful to condition the generative model on some additional information y (e.g. a class label or a text). In the case of diffusion models, the conditioning is done by adding an additional input to the score function such that a conditional score is learned:

$$\theta^\star = \arg\min_\theta \frac{1}{2}\sum_{k=1}^{T} \mathbb{E}_{X_0,Y}\mathbb{E}_{X_t|X_0}[\|s_\theta(t, X_t^Y; Y) - \nabla \log p(X_t^Y|X_0^Y)\|^2]$$

The conditioning is fed into the network in a similar way than the time features.

Diffusion generated images

An illustration of an avocado sitting in a therapist's chair saying 'I just feel so empty inside' with a pit-sized hole in its center. The therapist, a spoon, scribbles notes

An expressive oil painting of a basketball payer dunking, depicted as an explosion of a nebula.

A portrait of a dog in a library, Sigma 85mm f/ 1.4

An astronaut riding a horse

(The above figures are taken from https://openai.com/dall-e-3, https://openart.ai/ and https://the-decoder.com.)

Applications of GANs and Diffusion Models

- Image Generation and Editing
- Art Creation
- Data Augmentation
- Adversarial Examples
- Drug Discovery
- Protein Structure Prediction
- Anomaly Detection
- Weather Forecasting
- Video Generation
- Voice Synthesis and Modification
- Restoration of Old Photographs and Artworks
- Fashion Design
- Architectural Visualization
- Gaming
- Virtual Reality and Augmented Reality
- Language Translation and Interpretation
- Financial Modeling

GANs and diffusion models have also been used for other data modalities: For raw-waveform audio synthesis, examples include WaveGAN [Donahue et al., 2019] and MelGAN [Kumar et al., 2019], among others. For generating realistic tabular data, see e.g. [Kotelnikov et al., 2022].

Summary

We have studied:

1. Generative models
2. Generative Adversarial Networks
    - Players (generator & discriminator)
    - Objectives
    - Solution & Algorithm, Conditional GANs
3. Diffusion models
    - Ancestral sampling
    - Denoising Score Matching
    - Conditional training

Additional Notes (optional material)

KL and JS divergences

Before proving that at the equilibrium of the above GAN framework $p_g = p_d$ we need to define some measures of similarity between two probability distributions: The KL and JS divergences.

The Kullback–Leibler (KL) divergence is defined as:

$$\mathbb{D}_{KL}(p_d||p_g) := \int_x \log\left(\frac{p_d(x)}{p_g(x)}\right) p_d(x)\,\mathrm{d}x\,.$$

KL is also called *relative entropy*, as it measures how one probability distribution is different from a "reference" probability distribution, and it is asymmetric.

The Jenson-Shannon (JS) divergence is defined as:

$$\mathbb{D}_{JS}(p\|q) := \frac{1}{2}\mathbb{D}_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2}\mathbb{D}_{KL}(q\|\frac{p+q}{2})$$

Note that contrary to the KL divergence defined above, the JS divergence is symmetric.

The GAN framework: Equilibrium at $p_g = p_d$

In the following, we'll assume the neural network models G and D have infinite capacity, so can represent any probability distribution. We will study the convergence of the loss function in the space of probability density functions.

The discriminator maximizes:

$$\begin{aligned}\mathcal{L}(G, D) &= \int_x p_d(x) \log(D(x))\, \mathrm{d}x + \int_z p_z(z) \log(1 - D(G(z)))\, \mathrm{d}z \\ &= \int_x p_d(x) \log(D(x)) + p_g(x) \log(1 - D(x))\, \mathrm{d}x\end{aligned}$$

Where we used $x = G(z)$, and p_g is the distribution of x.
The above integrand can be written as: $f(y) = a \log y + b \log(1 - y)$. To solve for its critical points: $f'(y) = 0 \Rightarrow \frac{a}{y} - \frac{b}{1-y} = 0 \Rightarrow y = \frac{a}{a+b}$. Moreover, if $a+b \neq 0$ we obtain that $\frac{a}{a+b}$ is a maximum as $f''(\frac{a}{a+b}) < 0$.
Hence, optimal discriminator $D^\star$ is:

$$D^\star(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

By replacing the optimal discriminator in the above objective, we obtain that the generator minimizes:

$$\begin{aligned}\mathcal{L}(G, D^\star) &= \mathop{\mathbb{E}}_{x\sim p_d}[\log D^\star(x)] + \mathop{\mathbb{E}}_{x\sim p_g}[\log(1 - D^\star(x))] \\ &= \mathop{\mathbb{E}}_{x\sim p_d}[\log \frac{p_d(x)}{p_d(x) + p_g(x)}] + \mathop{\mathbb{E}}_{x\sim p_g}[\log \frac{p_g(x)}{p_d(x) + p_g(x)}] \\ &= -\log 4 + \mathbb{D}_{KL}\Big(p_d \Big\| \frac{p_d + p_g}{2}\Big) + \mathbb{D}_{KL}\Big(p_g \Big\| \frac{p_d + p_g}{2}\Big) \\ &= -\log 4 + 2 \cdot \mathbb{D}_{JS}(p_d \| p_g)\end{aligned}$$

where to obtain the third expression we used the definition of logarithm:

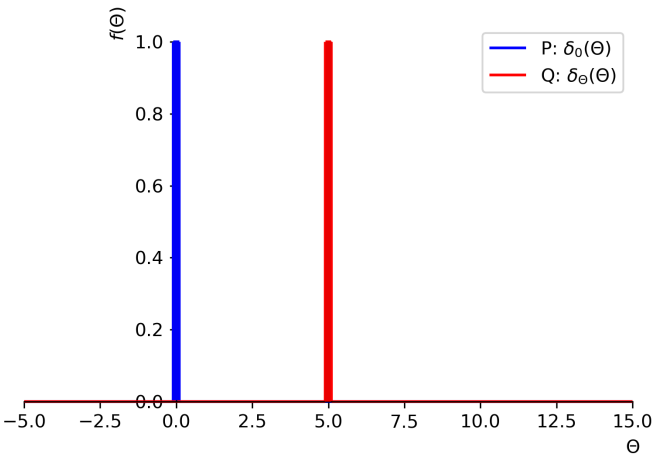
$$\begin{aligned}&\log 2 + \log\Big(\frac{p_d(x)}{p_g(x) + p_d(x)}\Big) \\ &= \log\Big(2\frac{p_d(x)}{p_g(x) + p_d(x)}\Big) \\ &= \log\Big(\frac{p_d(x)}{\frac{p_g(x)+p_d(x)}{2}}\Big).\end{aligned}$$

Above, $\mathbb{D}_{KL}$ and $\mathbb{D}_{JS}$ again denote the Kullback–Leibler and the Jenson-Shannon divergences (see previous slides).

The optimum is reached when $p_g = p_d$ (note $D^\star = \frac{1}{2}$), and the optimal value is $-\log 4$.

Drawback of using JS divergence for GANs

Example: Let us consider two probability distributions defined on $\mathbb{R}$: P: $\delta_0(x)$ and Q: $\delta_\theta(x)$.



Note that when the supports of the two distributions are disjoint ($\theta \neq 0$), we obtain $\mathbb{D}_{KL}(P||Q) = +\infty$, and $\mathbb{D}_{JS}(P||Q) = \log 2$, both yielding non-smooth gradient, making gradient-based methods hard to use [Arjovsky et al., 2017].

Wasserstein Distance

The previous example motivates the use of the Wasserstein distance (*aka Earth mover's* distance) in the context of GANs, described next.

Wasserstein-1 distance is defined as:

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

where $\Pi(p_d, p_g)$ is the set of all possible joint probability distributions between p_d and p_g, whose marginals are p_d, p_g, resp.

Intuitively, the two distributions can be viewed as a mass on each point. The goal is to move these masses so that one distribution can be transformed into the other. As there are infinitely many ways of doing so, $\mathbb{D}_W(p_d, p_g)$ is the *minimum cost* we need to spend. The cost is the amount of mass that has to be transported times the distance it has to be moved.

Note that in the above example $\mathbb{D}_W(\delta_0(x), \delta_\theta(x))=|\theta|$, and it provides "usable" gradient. However, $\mathbb{D}_W(\cdot)$ does not scale with the dimensionality of the input random variables, as the number of states of the joint probability distribution grows exponentially. Fortunately, it can be alternatively formalized (see additional material).

Wasserstein GAN

(optional material)

Def. $f : \mathbb{R} \to \mathbb{R}$ is called k-Lipschitz continuous if $\exists k \in \mathbb{R}$ s.t.

$$|f(x_1) - f(x_2)| \leq k|x_1 - x_2|, \quad \forall x_1, x_2.$$

The so called Wasserstein GAN [WGAN, Arjovsky et al., 2017, Gulrajani et al., 2017] replaces the JS divergence with the Wasserstein distance. As the Wasserstein distance is intractable for Deep Neural Nets, WGANs make use of the so called Kantorovich-Rubinstein duality principle, which tells us that:

$$\mathbb{D}_W(p_d, p_g) = \sup_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{x \sim p_d} [f(x)] - \mathop{\mathbb{E}}_{x \sim p_g} [f(x)] \,,$$

where the supremum is over 1-Lipschitz functions $f : \mathcal{X} \mapsto \mathbb{R}$. Its derivation is out of the scope of this course (if interested see proof sketch in the Appendix).
In the context of GANs, f is the function represented by the discriminator (called *critic* in WGAN), yielding:

$$\min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{x \sim p_d} [D(x)] - \mathop{\mathbb{E}}_{x \sim p_g} [D(x)] \,,$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions (see next slide).

Some GAN variants with Lipschitz Discriminator

(optional material)

The constraint that the discriminator should be 1-Lipschitz can be enforced in several ways. The table below summarizes some of the GAN variants which enforce such constraint.

WGAN [Arjovsky et al., 2017] uses straightforward weight clipping. [Gulrajani et al., 2017] point out that this may lead to optimization difficulties, and proposed adding an extra penalty term to the training loss of the Discriminator, which penalizes gradients whose norm is higher than 1. As enforcing this for any input is intractable, this is done by considering a line between fake and real samples, obtained by interpolating between these. [Kodali et al., 2017] penalize gradients whose norm is higher than 1 while considering samples in a region around real data points. Note that *DRAGAN* & *WGAN−GP* are computationally more expensive as each parameter update requires computation of a second-order derivative (as the loss includes gradient penalty). [Miyato et al., 2018] make use of the power iteration method to estimate the largest singular value per layer, so as to divide the parameters of that layer with it. In the context of training GANs, although the power iteration method requires multiple iterations to compute the largest singular value, it cam be implemented efficiently due to the fact that it is combined with Stochastic gradient based method. In other words, it was shown in practice that one update of it per stochastic parameter update is sufficient to obtain good estimates of the largest singular value. It was empirically shown that enforcing 1-Lipschitz Discriminator helps the convergence of GAN trained with JS-based losses, and JS-based GANs remained widely used in practice. Moreover, besides lacking theoretical backup, one of the well performing large-scale GAN variants further enforces that both the Generator and the discriminator are 1-Lipschitz functions.

	$\mathbb{D}\_(p_d\|\|p_g)$	G/D	Mean of enforcing Lipschitz continuity
WGAN [Arjovsky et al., 2017]	$\mathbb{D}_W$	D	Weight clipping: $w = clip(w, -c, c)$
WGAN−GP [Gulrajani et al., 2017]	$\mathbb{D}_W$	D	Gradient penalty: line between real and fake points
DRAGAN [Kodali et al., 2017]	$\sim \mathbb{D}_{JSD}$	D	Gradient penalty: around real data points
SNGAN [Miyato et al., 2018]	$\sim \mathbb{D}_{JSD}$	D	Spectral norm using the power iteration method
bigGAN [Brock et al., 2019]	$\sim \mathbb{D}_{JSD}$	$G\&D$	Spectral norm using the power iteration method

Appendix A: Wasserstein Distance

Continious probability distributions

The Wasserstein distance between two probability distributions μ and ν is defined as:

$$W(\mu,\nu) = \inf_{\pi(\mu,\nu)} \iint c(x,y)\pi(dx,dy), \tag{1}$$

Equation (1) assumes continuous distributions μ and ν. If we use the Euclidean distance we have:

$$\begin{aligned} W(\mu,\nu) &= \inf_{\pi} \iint \|x-y\|\pi(x,y)\,\mathrm{d}x\,\mathrm{d}y \\ &= \inf_{\pi} \mathbb{E}_{x,y\sim\pi}[\|x-y\|]. \end{aligned} \tag{2}$$

Discrete probability distributions

Lets consider two discrete distributions $\mathbb{P}_x, \mathbb{P}_y$, with s states each: x_i and y_i, $i = 1 \dots s$. Thus,

$$\begin{aligned} W(\mathbb{P}_x,\mathbb{P}_y) &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \sum_{x,y} \|x-y\|\Pi(x,y) \\ &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \mathbb{E}_{(x,y)\sim\Pi}[\|x-y\|] \\ &= \inf_{\Pi(\mathbb{P}_x,\mathbb{P}_y)} \langle \Pi, D\rangle, \end{aligned} \tag{3}$$

where with $\langle\cdot,\cdot\rangle$ we denote sum of element-wise multiplication, and $\Pi \in \mathbb{R}^{s\times s}$ is the joint probability and $D \in \mathbb{R}^{s\times s}$ is the Euclidean distance between each $x_i, y_j, i = 1 \dots s, j = 1 \dots s$.

Kantorovich-Rubinstein dualty principle

From Kantorovich-Rubinstein dualty [Villani, 2008]:

$$\mathbb{D}_W(p_d,p_g) = \frac{1}{K} \sup_{\|f\|_L\le K} \mathop{\mathbb{E}}_{x\sim p_d}[f(x)] - \mathop{\mathbb{E}}_{x\sim p_g}[f(x)]$$

Villani [2008] gives the following intuitive interpretation of the above Kantorovich duality principle. Namely, if our goal is to transfer a huge

amount of mass distributed over certain area, to a different distinct area, we would like to minimize the cost for transport, thus we have an inf over the implied cost. Suppose we have a middle-man who offers to handle the problem for us, by claiming that he will not charge us more than the actual transport cost (thus $\varphi(y) - \psi(x) \leq c(x, y)$). Then, our initial problem is in fact equal to the one of the middle-man trying to maximize his profit. His profit on the other hand is defined as a price for loading goods: $\varphi(x)$ and a price for unloading them at destination y: $\psi(y)$. Naturally, he aims at maximizing his profit (thus the sup). Note however that the middle man is ready to give financial compensations for some places, in the form of negative prices.

Proof sketch. See [Villani, 2008] for full formal proof.

- x cont. r.v.
- μ distribution of x
- ν target distribution
- $c(\cdot)$ cost, *e.g.* ℓ_p norm

$$\begin{aligned}\mathbb{D}_W(\mu, \nu) &= \inf_{\gamma \in \pi} \iint c(x, y)\, \mathrm{d}\gamma(x, y) \\ &= \sup_{\varphi, \psi \to \mathbb{R}} \int \varphi(y)\, \mathrm{d}\nu - \psi(x)\, \mathrm{d}\mu\end{aligned}$$

where $\varphi(y) - \psi(x) \leq c(x, y)$, and π is set of all non-negative Borel measures, whose marginals are $\int_x \pi = \nu$ and $\int_y \pi = \mu$.

$$
\begin{aligned}
\mathbb{D}_W(\mu,\nu) &= \inf_{\gamma\in\pi}\iint c(x,y)\,\mathrm{d}\gamma(x,y)\\
&= \inf_{\gamma\in\pi}\iint c(x,y)\,\mathrm{d}\gamma(x,y) + \begin{cases}0, & \text{if } \gamma\in\pi\\ -\infty, & \text{otherwise}\end{cases}\\
&= \inf_{\gamma\in\pi}\{\iint c(x,y)\,\mathrm{d}\gamma(x,y) + \sup_{\varphi,\psi\to\mathbb{R}}[\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu - \iint(\varphi(y)-\psi(x))\,\mathrm{d}\gamma(x,y)]\}\\
&= \inf_{\gamma\in\pi}\sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \iint(c(x,y)-(\varphi(y)-\psi(x)))\,\mathrm{d}\gamma(x,y)\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \inf_{\gamma\in\pi}\iint(c(x,y)-(\varphi(y)-\psi(x)))\,\mathrm{d}\gamma(x,y)\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu + \begin{cases}0, & \text{if } \varphi(y)-\psi(x)\le c(x,y)\\ -\infty, & \text{otherwise}\end{cases}\}\\
&= \sup_{\varphi,\psi\to\mathbb{R}}\{\int \varphi(y)\,\mathrm{d}\nu - \int \psi(x)\,\mathrm{d}\mu\}.
\end{aligned}
$$

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *ICLR*, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. In *NIPS*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Naveen Kodali, Jacob D. Abernethy, James Hays, and Zsolt Kira. How to train your DRAGAN. *arXiv:1705.07215*, 2017.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. *arXiv preprint arXiv:2209.15421*, 2022.

K. Kumar, Rithesh Kumar, T. D. Boissière, L. Gestin, Wei Zhen Teoh, J. Sotelo, A. D. Brébisson, Yoshua Bengio, and Aaron C. Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In *NeurIPS*, 2019.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.

Cédric Villani. *Optimal Transport: Old and New*. Springer, 2009 edition, September 2008. ISBN 3540710493.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021.