Aprendizado Profundo

Generative Adversarial Networks

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Visão Geral

1. O que são GANs
2. MinMax Loss
3. DCGAN
4. Colapso para a Moda
5. Análise da Loss
6. Vanishing Gradient
7. Wasserstein GAN
8. Tricks de Treinamento
9. Panorama Geral

Visão Geral

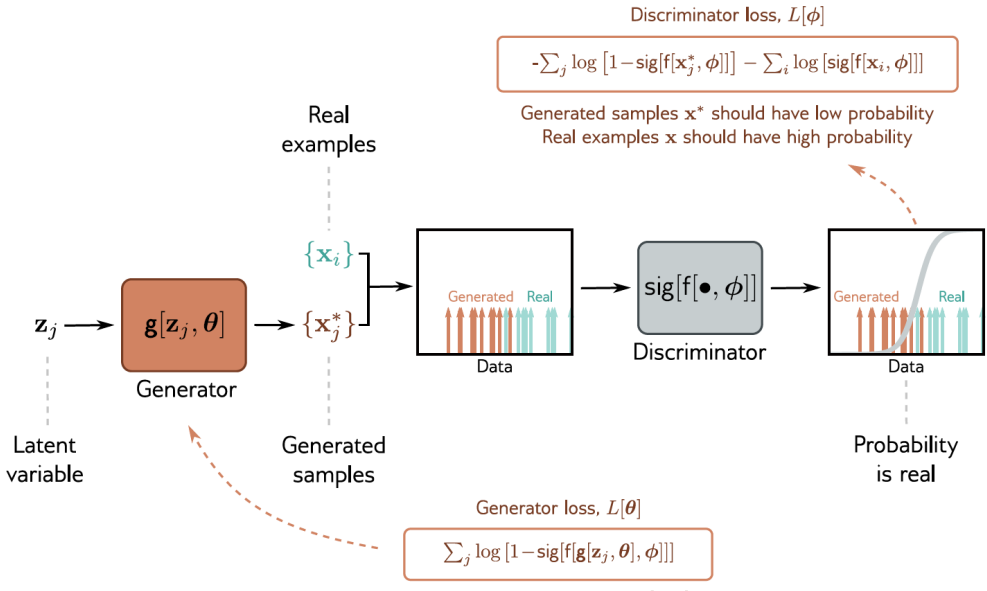
Generative Adversarial Network[1]

Uma **GAN** é um modelo não supervisionado cujo objetivo é **gerar novas instâncias indistinguíveis** do conjunto de dados de treinamento.

- Não constrói explicitamente uma distribuição de probabilidade sobre os dados.
- Em vez disso, aprende a amostrar dessa distribuição.
- Usualmente é formada por **duas redes** treinadas em conjunto:
  - **Gerador**: mapeia uma distribuição aleatória para o espaço dos dados.
  - **Discriminador** (ou crítico): classifica as instâncias como reais ou sintéticas.
- Desenvolvida originalmente por Ian Goodfellow em 2014.

[1] Ian Goodfellow et al. "Generative Adversarial Nets". Em: NeurIPS. vol. 27. 2014.

Gerador e Discriminador em conjunto



Aplicações

Onde GANs já mostraram resultados marcantes:

- **Síntese de faces fotorrealistas**: *StyleGAN*[2] produz retratos indistinguíveis de pessoas reais.
- **Image-to-image translation**: *pix2pix*[3] (mapa → foto, esboço → foto) e *CycleGAN*[4] (cavalo ↔ zebra) sem pares alinhados.
- **Super resolução**: ampliar imagens preservando textura plausível.
- **Aumento de dados**: gerar amostras sintéticas para classes raras em domínios médicos e industriais.
- **Deepfakes**: vídeo e áudio sintéticos de pessoas reais, com todas as questões éticas que isso traz.

[2] Tero Karras, Samuli Laine e Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". Em: CVPR. 2019.

[3] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". Em: CVPR. 2017.

[4] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". Em: ICCV. 2017.

GANs – Princípios

Objetivo: gerar novas instâncias $\{x^{(j)*}\}$ que sigam a mesma distribuição dos dados de treino $\{x^{(i)}\}$.

Processo de geração de uma nova instância

1. Amostrar uma **variável latente** $z^{(j)}$ de uma distribuição simples (normal, uniforme, ...).
2. Usar $z^{(j)}$ como entrada para uma rede neural:

$$x^{(j)*} = g\left[z^{(j)}, \boldsymbol{\theta}\right]$$

onde $\boldsymbol{\theta}$ são os parâmetros do gerador aprendidos durante o treino.

GANs – Princípios

- Teremos atingido o objetivo quando as instâncias geradas forem **indistinguíveis** das instâncias do conjunto de dados.
- Introduzimos então uma segunda rede $f[\cdot, \phi]$ que tenta **classificar** suas entradas como reais ou geradas.
  - Se o discriminador consegue classificar bem, ele pode gerar um sinal de gradiente usado para melhorar o gerador.
  - Se o discriminador não consegue distinguir, o gerador atingiu o objetivo.

Intuição de jogo

O gerador tenta enganar o discriminador; o discriminador tenta não ser enganado. Um equilíbrio de Nash nesse jogo corresponde a $P_{\text{gerada}} = P_{\text{real}}$.

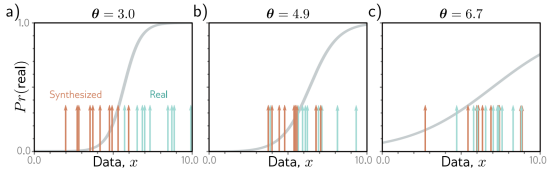
Para enxergar essa dinâmica, vamos olhar um exemplo unidimensional em que o gerador se reduz a um único parâmetro.

GANs – Princípios: um exemplo *toy*

- Os dados reais foram obtidos de uma normal $\mathcal{N}(7, 1)$.
- O latente é amostrado de $z^{(j)} \sim \mathcal{N}(0, 1)$ e o modelo gerador possui **um único parâmetro** que translada essa distribuição:

$$x^{(j)*} = z^{(j)} + \theta.$$

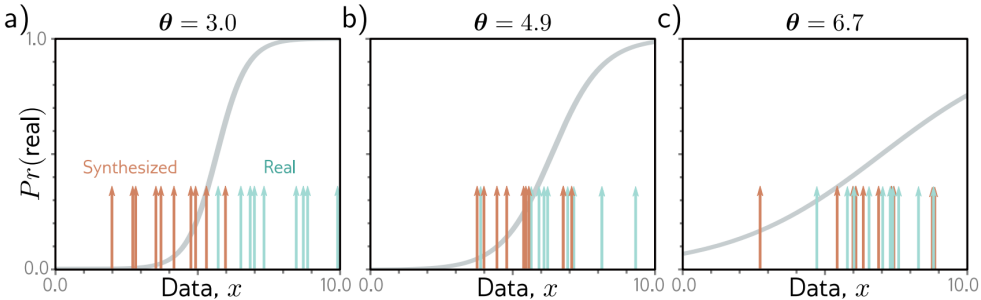
- A sigmóide representa o classificador binário ideal.



Painel (a) com $\theta = 3.0$.

GANs – Princípios: sobreposição de distribuições

- Valores diferentes de θ fazem com que o classificador tenha maior ou menor dificuldade em separar as distribuições.
- Quando $\theta \approx 7$, as distribuições coincidem e o classificador ideal degenera para chance aleatória (sigmóide quase plana).



(a) distribuições separadas; (b) parcialmente sobrepostas; (c) quase coincidentes.

Visão Geral

Loss do Discriminador

Para o discriminador, temos um problema de **classificação binária**. Rotulamos as instâncias reais com $y = 1$ e as geradas com $y = 0$, e minimizamos a entropia cruzada:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[\sum_i -\left(1 - y^{(i)}\right) \ln\left[1 - \sigma\left(f\left[x^{(i)}, \phi\right]\right)\right] - y^{(i)} \ln\left[\sigma\left(f\left[x^{(i)}, \phi\right]\right)\right] \right]$$

- $\sigma(\cdot)$ é a sigmóide que mapeia o logit $f[\cdot, \phi]$ para probabilidade de "real".
- A primeira parcela penaliza rotular geradas como reais; a segunda penaliza rotular reais como geradas.

Loss do Discriminador (separando reais e geradas)

Para tornar mais evidente que algumas instâncias são geradas ($y^{(j)} = 0$), reescrevemos a loss em duas somas distintas:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[\sum_{j} \left(-\ln\left[1 - \sigma\left(f\left[x^{(j)*}, \phi\right]\right)\right]\right) - \sum_{i} \left(\ln\left[\sigma\left(f\left[x^{(i)}, \phi\right]\right)\right]\right) \right]$$

- j indexa instâncias geradas ($x^{(j)*}$), i indexa instâncias reais ($x^{(i)}$).
- O gerador entra apenas no primeiro somatório, por meio das amostras $x^{(j)*} = g[z^{(j)}, \theta]$.

MinMax Loss

Plugando a definição do gerador e **invertendo o objetivo do gerador**, obtemos a formulação adversarial conjunta:

$$\hat{\phi}, \hat{\boldsymbol{\theta}} = \underset{\phi}{\operatorname{argmin}} \ \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_j \left(-\ln\left[1 - \sigma\left(f\left[g[z^{(j)}, \boldsymbol{\theta}], \phi\right]\right)\right]\right) - \sum_i \left(\ln\left[\sigma\left(f\left[x^{(i)}, \phi\right]\right)\right]\right) \right]$$

MinMax Loss

O discriminador **minimiza** a entropia cruzada (classifica bem) enquanto o gerador **maximiza** o mesmo objetivo (tenta enganar o discriminador).

MinMax Loss – Papéis de gerador e discriminador

$$\hat{\phi}, \hat{\boldsymbol{\theta}} = \underset{\phi}{\operatorname{argmin}} \; \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\underbrace{\sum_j \left(-\ln\left[1 - \sigma\left(f\left[g\left[z^{(j)}, \boldsymbol{\theta}\right], \phi\right]\right)\right]\right)}_{\text{Gerador + Discriminador}} \underbrace{- \sum_i \left(\ln\left[\sigma\left(f\left[x^{(i)}, \phi\right]\right)\right]\right)}_{\text{Discriminador}} \right]$$

- O discriminador recebe gradientes dos dois termos: quer classificar bem reais e geradas.
- O gerador só interage com o primeiro termo: só se importa em enganar o discriminador sobre as amostras sintéticas.

MinMax Loss – Loss por rede

Na prática, treinamos as redes alternadamente, cada uma otimizando sua própria loss:

Loss do discriminador (minimizar em ϕ)

$$L[\phi] = \sum_j \left(-\ln\left[1 - \sigma\left(f\left[g[z^{(j)}, \boldsymbol{\theta}], \phi\right]\right)\right]\right) - \sum_i \left(\ln\left[\sigma\left(f\left[x^{(i)}, \phi\right]\right)\right]\right)$$

Loss do gerador (minimizar em θ)

$$L[\boldsymbol{\theta}] = \sum_j \left(\ln\left[1 - \sigma\left(f\left[g[z^{(j)}, \boldsymbol{\theta}], \phi\right]\right)\right]\right)$$

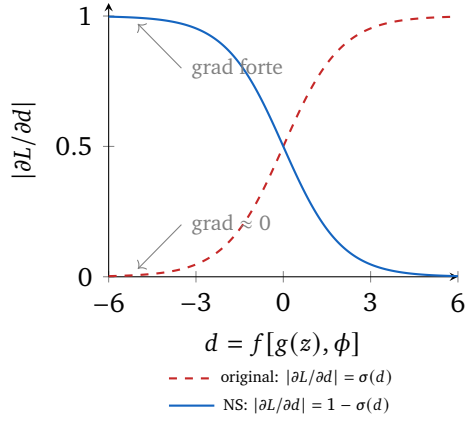
Non-saturating Loss – Truque prático do gerador

No início do treino o discriminador vence facilmente: $\sigma(f[g(z), \phi]) \approx 0$. A loss $\ln[1 - \sigma(\cdot)]$ fica quase plana em 0, e o gradiente que chega ao gerador é minúsculo.

Substituição usada na prática

$$L_{\text{NS}}[\boldsymbol{\theta}] = \sum_j \left(-\ln\left[\sigma\left(f\left[g[z^{(j)}, \boldsymbol{\theta}], \phi \right] \right) \right] \right)$$

- Em vez de minimizar a probabilidade de ser pega, o gerador maximiza a probabilidade de parecer real.
- Mesmo ponto fixo (equilíbrio em $\sigma = \frac{1}{2}$), mas gradiente saudável quando o gerador é fraco.
- Já estava na proposta original de (Goodfellow et al., 2014) e é o default em praticamente todas as implementações.



Quando o discriminador vence ($\sigma \approx 0$, $d \to -\infty$) o gradiente da loss original some, o da NS permanece próximo de 1. O gráfico mostra $|\partial L/\partial d|$, os sinais reais são negativos ($-\sigma(d)$ e $\sigma(d) - 1$).

MinMax Loss – Algoritmo de treino

Treinamos as redes **alternadamente**, em passos de gradiente:

Require: dataset real, número de passos do discriminador k, taxas de aprendizado η_ϕ, η_θ, tamanho de batch m

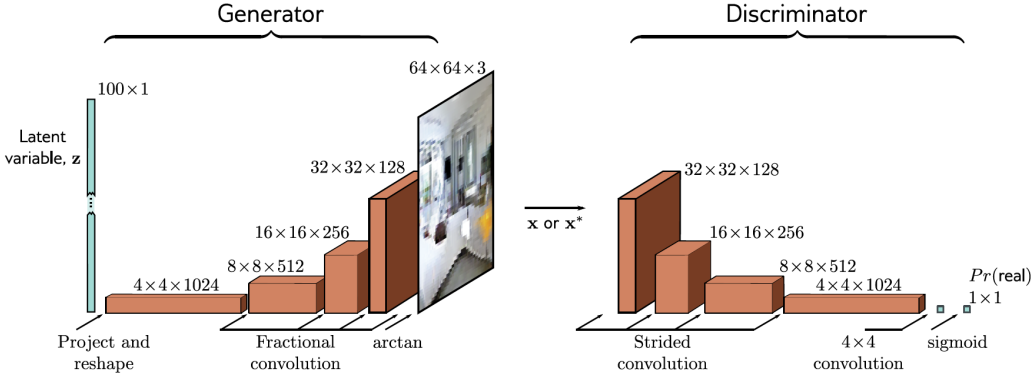
1: **while** θ não convergiu **do**
2: **for** $t = 1, \ldots, k$ **do**
3: Amostre minibatch de reais $\{x^{(i)}\}_{i=1}^{m}$
4: Amostre ruído $\{z^{(j)}\}_{j=1}^{m}$
5: $\phi \leftarrow \phi - \eta_\phi \nabla_\phi L[\phi]$
6: **end for**
7: Amostre novo ruído $\{z^{(j)}\}_{j=1}^{m}$
8: $\theta \leftarrow \theta - \eta_\theta \nabla_\theta L[\theta]$
9: **end while**

- No paper original $k = 1$. Em *WGAN* veremos $k = 5$ (queremos crítico próximo do ótimo).
- Otimizador típico: Adam com $\beta_1 = 0.5$ ($\beta_1 = 0.9$ desestabiliza GANs).

Visão Geral

DCGAN[5]

A primeira arquitetura a popularizar GANs para imagens naturais: **Deep Convolutional GAN**.



- Gerador usa **convoluções fracionárias** (upsampling) partindo de um vetor latente 100×1.
- Discriminador usa **convoluções com stride** para reduzir a resolução.

DCGAN – Receita arquitetural

O paper destila um conjunto de guidelines que dominaram os anos seguintes:

- **Sem *pooling***: troque por convoluções com *stride* (D) e convoluções fracionárias (G).
- **Sem camadas *fully connected*** ocultas em ambas as redes.
- **BatchNorm** em D e G, exceto na saída de G e na entrada de D (estabiliza o treino sem distorcer a saída).
- **Ativações**: ReLU em todas as camadas de G (Tanh na saída); LeakyReLU em todas as camadas de D.
- Vetor latente $z \in \mathbb{R}^{100}$, normal padrão.

Por que importa

Antes da DCGAN, treinar GAN convolucional era considerado quase impossível. Esses cinco pontos viraram baseline para praticamente todas as arquiteturas posteriores.

DCGAN – Amostras geradas

Figure 15.4 Synthesized images from the DCGAN model. a) Random samples drawn from DCGAN trained on a faces dataset. b) Random samples using the ImageNet database (see figure 10.15). c) Random samples drawn from the LSUN scene understanding dataset. Adapted from Radford et al. (2015).

Amostras sintéticas geradas por DCGANs treinadas em três *datasets*: (a) faces, (b) ImageNet, (c) LSUN (cenas). Adaptado de (Radford, Metz e Chintala, 2015).

Visão Geral

Mode Collapse

É comum que o gerador **“colapse para a moda”** da distribuição, gerando instâncias sem diversidade.

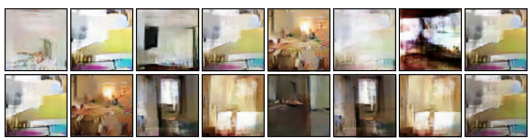
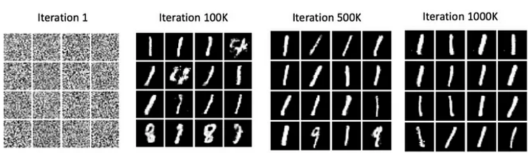


Figure 15.5 Mode collapse. Synthesized images from a GAN trained on the LSUN scene understanding dataset using an MLP generator with a similar number of parameters and layers to the DCGAN. The samples are low quality, and many are similar. Adapted from Arjovsky et al. (2017).

GAN em LSUN: cenas muito similares.



MNIST ao longo das iterações: diversidade some.

O motivo disso fica aparente ao analisarmos a função de custo.

Visão Geral

Análise da Loss – Normalização

Partimos da loss do discriminador, escrita aqui sem o sinal de menos (a forma que o discriminador **maximiza**), e **dividimos cada somatório** pelo número de instâncias (geradas J ou reais I):

$$L[\phi] = \sum_j \left(\ln\left[1 - \sigma\left(f[x^{(j)*}, \phi]\right)\right]\right) + \sum_i \left(\ln\left[\sigma\left(f[x^{(i)}, \phi]\right)\right]\right)$$

$$L[\phi] \approx \tfrac{1}{J} \sum_j \left(\ln\left[1 - \sigma\left(f[x^{(j)*}, \phi]\right)\right]\right) + \tfrac{1}{I} \sum_i \left(\ln\left[\sigma\left(f[x^{(i)}, \phi]\right)\right]\right)$$

Isso não altera o mínimo (multiplicar por constante não muda argmin), mas prepara o próximo passo.

Análise da Loss – Esperança/Expectativa

Ao dividir pelo número de instâncias, cada termo vira uma **esperança** sobre a distribuição correspondente:

$$L[\phi] = \mathbb{E}_{x^*}[\ln[1 - \sigma(f[x^*, \phi])]] + \mathbb{E}_x[\ln[\sigma(f[x, \phi])]]$$

- $\mathbb{E}_{x^*}$: esperança sobre a distribuição gerada $P(x^*)$.
- $\mathbb{E}_x$: esperança sobre a distribuição real $P(x)$.
- Essa reescrita muda o ponto de vista: deixamos de falar de amostras e passamos a falar de distribuições.

Análise da Loss – Integrais

Aplicando a definição de esperança, ficamos com **duas integrais**:

$$L[\phi] = \int P(x^*) \ln[1 - \sigma(f[x^*, \phi])]\, dx^* + \int P(x) \ln[\sigma(f[x, \phi])]\, dx$$

- Os integrandos dependem do discriminador $\sigma(f[\cdot, \phi])$.
- Vamos substituir o discriminador pela expressão ótima (que minimiza essa loss).

Análise da Loss – Discriminador ótimo

A saída $\sigma[f[\tilde{x}, \phi]]$ é a probabilidade que o discriminador atribui ao rótulo real dada uma amostra $\tilde{x}$. Por **regra de Bayes**:

$$\sigma[f[\tilde{x}, \phi]] = P(\text{real} \mid \tilde{x}) = \frac{P(\tilde{x} \mid \text{real})\, P(\text{real})}{P(\tilde{x} \mid \text{real})\, P(\text{real}) + P(\tilde{x} \mid \text{gen})\, P(\text{gen})}$$

Assumindo $I = J$, ou seja $P(\text{real}) = P(\text{gen}) = \frac{1}{2}$, os priors se cancelam:

$$\sigma[f[\tilde{x}, \phi]] = \frac{P(\tilde{x} \mid \text{real})}{P(\tilde{x} \mid \text{real}) + P(\tilde{x} \mid \text{gen})} = \frac{P(x)}{P(x) + P(x^*)}$$

- $P(x)$ é a densidade dos dados reais em $\tilde{x}$, $P(x^*)$ é a densidade gerada no mesmo ponto.
- Em palavras: a probabilidade ideal de ser real é a fração da densidade total naquele ponto que vem do conjunto real.
- É o classificador perfeito, depende apenas das densidades, não de uma rede específica.

Análise da Loss – Substituindo o discriminador ótimo

Substituindo $\sigma(f[x,\phi]) = \frac{P(x)}{P(x)+P(x^*)}$ na loss:

$$L[\phi] = \int P(x^*) \ln\left[\frac{P(x^*)}{P(x)+P(x^*)}\right] dx^* + \int P(x) \ln\left[\frac{P(x)}{P(x)+P(x^*)}\right] dx$$

Os dois termos dentro dos logaritmos viraram **razões de densidades**, que é exatamente a forma de uma divergência KL.

Análise da Loss – Truque do fator 2

Por conveniência matemática, multiplicamos e dividimos por 2 dentro do logaritmo (não altera o argmin):

$$L[\phi] = \int P(x^*) \ln\left[\frac{2P(x^*)}{P(x) + P(x^*)}\right] dx^* + \int P(x) \ln\left[\frac{2P(x)}{P(x) + P(x^*)}\right] dx - 2\ln 2$$

Os dois fatores $\ln 2$ saem das integrais (pois $\int P = \int P^* = 1$). Agora cada termo é exatamente uma divergência KL entre uma distribuição e a mistura $\frac{P(x^*)+P(x)}{2}$.

Análise da Loss – Reconhecendo KL

Cada parcela tem a forma da **KL Divergence**:

$$D_{KL}[p(x) \,\|\, q(x)] = \int p(x) \ln\left[\frac{p(x)}{q(x)}\right] dx$$

Portanto:

$$L[\phi] = D_{KL}\left[P(x^*) \,\middle\|\, \tfrac{P(x^*)+P(x)}{2}\right] + D_{KL}\left[P(x) \,\middle\|\, \tfrac{P(x^*)+P(x)}{2}\right] - 2\ln 2$$

Análise da Loss – *Jensen–Shannon Divergence*

Reconhecemos a definição da **Jensen–Shannon Divergence**:

$$D_{JS}[P(x^*) \,\|\, P(x)] = \tfrac{1}{2} D_{KL}\left[P(x^*) \,\middle\|\, \tfrac{P(x^*)+P(x)}{2}\right] + \tfrac{1}{2} D_{KL}\left[P(x) \,\middle\|\, \tfrac{P(x^*)+P(x)}{2}\right]$$

Logo $L[\phi] = 2\, D_{JS}[P(x^*) \,\|\, P(x)] - 2\ln 2$.

Interpretação

Com discriminador ótimo, a loss de uma GAN é, a menos de uma constante e de um fator, a **divergência de Jensen–Shannon** entre a distribuição real e a gerada.

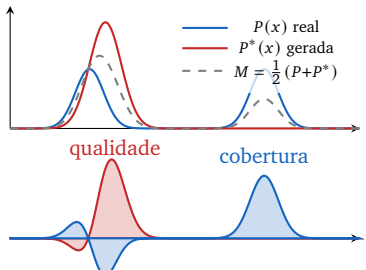
Análise da Loss – Qualidade vs. Cobertura

$$D_{JS}[P(x^*)\|P(x)] = \underbrace{\tfrac{1}{2}D_{KL}\left[P(x^*)\middle\|\tfrac{P(x^*)+P(x)}{2}\right]}_{\text{Qualidade}} + \underbrace{\tfrac{1}{2}D_{KL}\left[P(x)\middle\|\tfrac{P(x^*)+P(x)}{2}\right]}_{\text{Cobertura}}$$

- **Primeiro termo (qualidade)**, penaliza regiões em que $P(x^*)$ tem probabilidade alta mas a mistura não tem, isto é, amostras geradas implausíveis.
- **Segundo termo (cobertura)**, penaliza regiões em que $P(x)$ tem probabilidade alta mas a mistura não tem, isto é, modas ignoradas pelo gerador.

Pergunta

Qual dos dois termos o gerador consegue de fato controlar?



Acima: densidades normalizadas. Abaixo: integrando de $\frac{1}{2}D_{KL}[P^*\|M]$ (vermelho, qualidade) e $\frac{1}{2}D_{KL}[P\|M]$ (azul, cobertura). Áreas acima do eixo somam, abaixo subtraem; a integral líquida é o termo correspondente da D_{JS}. A moda de P em $x \approx 4$, ignorada por P^*, domina a cobertura.

Análise da Loss – Por que há colapso de moda

$$D_{JS}[P(x^*)\|P(x)] = \tfrac{1}{2}D_{KL}\left[P(x^*)\middle\|\tfrac{P(x^*)+P(x)}{2}\right] + \tfrac{1}{2}\underbrace{D_{KL}\left[P(x)\middle\|\tfrac{P(x^*)+P(x)}{2}\right]}_{\text{depende fracamente de } P(x^*)}$$

Consequência

Quando $P(x^*) \ll P(x)$ em uma região, $(P(x^*)+P(x))/2 \approx P(x)/2$, então o argumento do log no segundo termo satura em $\frac{P(x)}{P(x)/2} = 2$, e o integrando reduz-se a $\frac{1}{2}P(x)\log 2$, independente de $P(x^*)$.

O gerador é **incentivado muito mais fortemente a qualidade** que a cobertura → **colapso para a moda**.

Análise da Loss – Recapitulando

A cadeia de manipulações que aplicamos:

1. Loss original = entropia cruzada binária do discriminador.
2. Normalizamos cada soma → esperanças sobre $P(x)$ e $P(x^*)$.
3. Esperanças → integrais.
4. Substituímos $\sigma(f)$ pelo discriminador ótimo $\frac{P(x)}{P(x)+P(x^*)}$.
5. Multiplicamos e dividimos por 2 dentro do logaritmo.
6. Cada termo virou uma KL contra a mistura $\frac{P(x)+P(x^*)}{2}$.
7. Somando, obtemos $L[\phi] = 2\,D_{JS}[P(x^*) \,\|\, P(x)] - 2\ln 2$.

Moral

Treinar uma GAN com discriminador ótimo equivale a minimizar Jensen–Shannon. As fragilidades dessa divergência (assimetria qualidade vs cobertura, comportamento em suportes disjuntos) viram fragilidades do treinamento.

Exemplo Numérico
O Que Cada Valor Representa

Antes de calcular o D_{JS}, vamos ancorar cada símbolo em algo **concreto**. Imagine que estamos treinando uma GAN para gerar imagens em três categorias: gato, cão e pássaro. Identifique cada uma com a, b, c.

- $P(x)$: fração das imagens **reais** (no dataset) em cada categoria. É fixo, a verdade.
- $P^*(x)$: fração das imagens **geradas** pelo modelo atual em cada categoria. Muda conforme o gerador treina.
- $D^*(x)$: probabilidade que o **discriminador ótimo** atribui à categoria x de ser real. Onde P domina, D^* tende a 1; onde P^* domina, D^* tende a 0.
- $M(x) = \frac{1}{2}(P + P^*)$: distribuição de **mistura**, o referencial neutro contra o qual ambas as divergências KL são medidas.
- $D_{JS}(P^* \| P)$: o **único número** que resume a distância entre o que o gerador faz hoje e o que deveria fazer. Zero $\Rightarrow$ gerador perfeito; $\ln 2 \Rightarrow$ suportes disjuntos.

Exemplo Numérico: D^* e D_{JS} em 1D

Suporte $\{a, b, c\}$. Real $P = (0.5,\ 0.3,\ 0.2)$; gerada $P^* = (0.2,\ 0.5,\ 0.3)$.

Discriminador ótimo
$D^*(x) = \frac{P(x)}{P(x)+P^*(x)}$:

$$D^* \approx (\ 0.71,\ 0.38,\ 0.40\ )$$

Mistura $M = \frac{1}{2}(P + P^*)$:

$$M = (\ 0.35,\ 0.40,\ 0.25\ )$$

x	$P\ \ln(P/M)$	$P^*\ \ln(P^*/M)$
a	$+0.178$	-0.112
b	-0.086	$+0.112$
c	-0.045	$+0.055$
Σ	$D_{KL}(P\|M) = 0.047$	$D_{KL}(P^*\|M) = 0.054$

$$D_{JS}(P^*\|P) = \tfrac{1}{2}(0.047 + 0.054) \approx \mathbf{0.05}\ \text{nats}$$

Exemplo Numérico: limite de suporte disjunto

Mesmo suporte $\{a, b, c\}$, mas agora a categoria c **não aparece** nos dados reais: $P = (0.6,\ 0.4,\ 0.0)$, gerada $P^* = (0.2,\ 0.5,\ 0.3)$. Mistura $M = (\,0.40,\ 0.45,\ 0.15\,)$.

Em $x = c$ temos $P(c) = 0$: o termo $P \ln(P/M)$ se anula (convenção $0 \ln 0 = 0$). Já $P^*(c) = 0.3$ e $M(c) = P^*(c)/2$, então

$$P^*(c)\,\ln\frac{P^*(c)}{M(c)} = 0.3 \cdot \ln 2 \approx 0.21.$$

x	$P\,\ln(P/M)$	$P^*\,\ln(P^*/M)$
a	$+0.243$	-0.139
b	-0.047	$+0.053$
c	0	$+0.208$
Σ	$D_{KL}(P\|M) = 0.196$	$D_{KL}(P^*\|M) = 0.122$

$D_{JS}(P^*\|P) \approx \mathbf{0.16}$ nats

Limite de suporte totalmente disjunto

Quando P e P^* não compartilham nenhum ponto de suporte, $D_{JS}(P^*\|P) \to \ln 2 \approx 0.69$ nats (máximo da Jensen–Shannon). D_{JS} trava nesse valor e o gradiente do gerador desaparece, motivando a distância de Wasserstein.

Visão Geral

Por que distribuições disjuntas são a regra

- Dados naturais ocupam apenas uma pequena fatia do espaço de pixels: rostos, por exemplo, formam uma superfície curva de dimensão muito menor que a dimensão da imagem. Chamamos essa superfície de manifold.
- O gerador, por construção, também produz amostras sobre um manifold: a imagem da função $g[\cdot, \boldsymbol{\theta}]$ aplicada ao latente $z \in \mathbb{R}^d$, com $d \ll D$.
- Dois manifolds de dimensão baixa em $\mathbb{R}^D$, escolhidos independentemente, quase nunca se cruzam: a interseção tem **medida zero**.
- Consequência: no início do treino, $P(x)$ e $P(x^*)$ vivem em regiões praticamente disjuntas, e o D_{JS} fica preso em $\ln 2$.

Conclusão

Distribuições disjuntas **não** são caso patológico raro: são a situação típica para dados de alta dimensão. Por isso o problema a seguir aparece em quase todo treino de GAN.

Vanishing Gradient – Motivação

- Vimos que, no caso de discriminador ótimo, a loss minimiza uma **distância** (D_{JS}) entre instâncias reais e geradas.
- Contudo, essa abordagem traz problemas:
  - Se as distribuições são **disjuntas**, a D_{JS} assume valor constante $\ln 2$ e a D_{KL} diverge.
  - Nesse caso, nenhuma "pequena" mudança no gerador melhora a loss, os gradientes desaparecem.

Relembrando KL

$$D_{KL}[p(x)\|q(x)] = \int p(x) \ln\left[\frac{p(x)}{q(x)}\right] dx$$

Diverge sempre que existe x com $p(x) > 0$ e $q(x) = 0$.

Vanishing Gradient – Evidência empírica[6]

- O discriminador é congelado após 1, 10 e 25 épocas e o gerador continua treinando.
- Conforme o discriminador fica "bom demais", o gradiente que chega ao gerador despenca (escala log).
- Sintoma típico: treinamento instável, no qual o discriminador precisa ser deliberadamente enfraquecido para que o gerador aprenda.

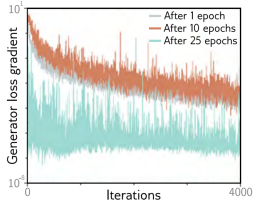


Figure 15.7 Vanishing gradients in the generator of a DCGAN. The generator is frozen after 1, 10, and 25 epochs, and the discriminator is trained further. The gradient of the generator decreases rapidly (note log scale); if the discriminator becomes too accurate, the gradients for the generator vanish. Adapted from Arjovsky & Bottou (2017).

Adaptado de (Arjovsky e Bottou, 2017).

[6] Martin Arjovsky e Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". Em: ICLR. 2017.

Motivação para mudar a distância

- Vimos que a loss original de GANs pode ser interpretada como uma **distância entre distribuições de probabilidade**.
- Porém, é uma distância com propriedades **inadequadas para otimização**:
  - Distribuições disjuntas têm D_{KL} infinita e D_{JS} constante.
  - Gradientes somem rapidamente.
- Vamos escolher uma distância com propriedades mais adequadas para otimização: a distância de **Wasserstein**.

Visão Geral

Wasserstein Distance[7]

- Também conhecida como *Earth Mover's Distance* (para distribuições discretas).
- Definida pela **quantidade de trabalho** necessária para transportar a massa de probabilidade de uma distribuição até transformá-la em outra.
  - Trabalho = massa $\times$ distância movida.
- Oferece **gradientes informativos** mesmo quando as distribuições são disjuntas.

[7] Martin Arjovsky, Soumith Chintala e Léon Bottou. "Wasserstein Generative Adversarial Networks". Em: ICML. vol. 70. 2017, pp. 214–223.

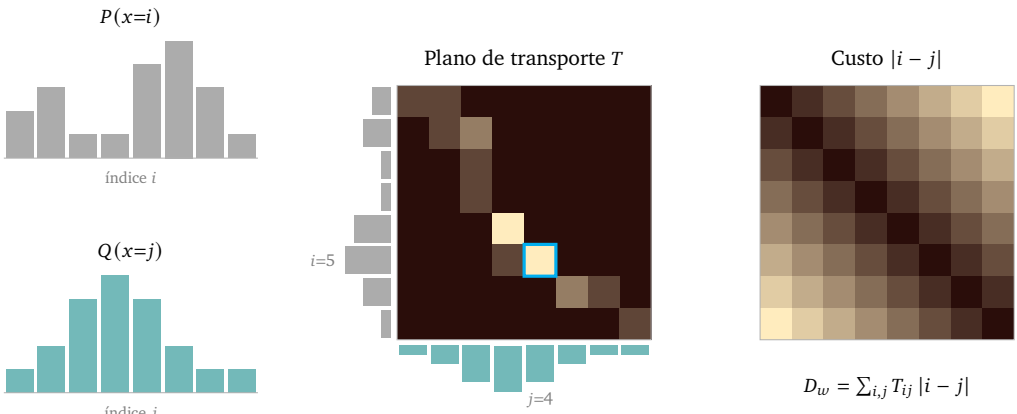
Wasserstein Distance – Caso discreto

Definição

$$D_w[P(x) \,\|\, Q(x)] = \min_T \left[\sum_{i,j} T_{ij} \, |i - j| \right]$$

- T é o **"plano de transporte"**: T_{ij} indica quanta massa vai de i (origem) para j (destino).
- Sujeito às restrições:
  - $\sum_j T_{ij} = P(x = i)$
  - $\sum_i T_{ij} = Q(x = j)$
  - $T_{ij} \geq 0$

Wasserstein Distance – Visualização



Exemplo discreto com suporte $\{0,\ldots,7\}$: as marginais do plano T (barras à esquerda e abaixo da matriz) reproduzem P e Q, e a célula destacada é $T_{54} = 0.2$. Neste exemplo, $D_w = 0.8$.

Wasserstein Distance – Formulação dual

- Para obter o valor de D_w, precisamos resolver um problema de **programação linear** em T.
- Isso não é prático durante o treinamento de GANs.
- Para esse problema específico, a formulação **dual** é mais útil.

Wasserstein Distance – De onde vem o f?

- O primal do slide anterior é um **LP**: minimizamos $\sum_{i,j} T_{ij}\,|i-j|$ sobre o plano T, sujeito às marginais e a $T_{ij} \geq 0$.
- Todo LP tem um **dual**, com uma variável por restrição. As marginais geram as variáveis f_i (nas origens) e g_j (nos destinos), e o dual maximiza:

$$\max_{f,g} \sum_i P(x{=}i)\, f_i + \sum_j Q(x{=}j)\, g_j \quad \text{s.a.} \quad f_i + g_j \leq |i-j|.$$

- Kantorovich–Rubinstein: para o custo $|i-j|$, o ótimo ocorre com $g = -f$. A restrição vira $f_i - f_j \leq |i-j|$, que é exatamente f ser **1-Lipschitz**.
- Sobra uma única função f a maximizar (próximo slide). É daí que vem o f: a variável dual associada às marginais.

Wasserstein Distance – Forma dual

Reescrevendo em sua forma dual, temos uma nova variável de otimização f que **maximizamos** sob a restrição de que valores adjacentes não se alterem em mais de uma unidade:

$$D_w[P(x) \parallel Q(x)] = \max_f \left[\sum_i P(x = i)\, f_i - \sum_j Q(x = j)\, f_j \right]$$

Sob a restrição: $|f_{i+1} - f_i| \leq 1$ (restrição de **1-Lipschitz** no caso contínuo).

Wasserstein GAN – Loss

Considerando que as distribuições são aproximadas pelas instâncias reais e geradas, obtemos a loss do **WGAN**:

$$L[\phi, \boldsymbol{\theta}] = \sum_j f\left[g[z^{(j)}, \boldsymbol{\theta}], \phi\right] - \sum_i f\left[x^{(i)}, \phi\right]$$

Sujeito à restrição: $\left|\frac{\partial f(x,\phi)}{\partial x}\right| \leq 1$

- Não há mais sigmóide: f é um **crítico**, não um classificador.
- A restrição de Lipschitz é imposta via weight clipping (a seguir) ou gradient penalty[5].

Wasserstein GAN – Weight clipping

- Como garantir, na prática, que o crítico f seja **1-Lipschitz** ao longo do treino?
- Weight clipping: depois de cada atualização do crítico, fazemos o clip de todos os pesos para o intervalo $[-c, c]$:

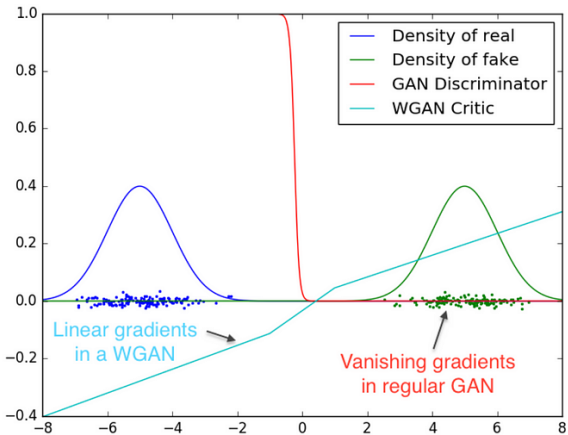
$$w \leftarrow \text{clip}(w, -c, c).$$

- Intuição: limitar a magnitude dos pesos limita o quão rápido f pode variar, ou seja, limita a sua constante de Lipschitz. Não garante exatamente 1, mas prende f numa família de funções com inclinação controlada.

É um instrumento grosseiro

- c pequeno demais: os gradientes somem; c grande demais: o treino fica instável.
- Os pesos tendem a saturar em $\pm c$, desperdiçando a capacidade do crítico (o que motiva o gradient penalty, adiante).
- Não confundir com gradient clipping, que limita os gradientes, não os pesos.

WGAN – Gradientes em distribuições disjuntas



Adaptado de Jonathan Hui (Medium, 2018).

- Discriminador saturado ⇒ gradientes nulos.
- Crítico WGAN produz gradientes **lineares** mesmo em regiões disjuntas ⇒ treinamento muito mais estável.
- A restrição de Lipschitz é a chave: evita que o crítico "exploda" para separar as distribuições.

WGAN – Algoritmo

Require: taxa α, limiar de clipping c, batch m, passos do crítico n_{critic}
Require: parâmetros iniciais w_0 (crítico), $\boldsymbol{\theta}_0$ (gerador)

while $\boldsymbol{\theta}$ não convergiu **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    Amostre $\{x^{(i)}\}_{i=1}^{m} \sim P$ (reais)
    Amostre $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ (prior)
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_i f_w(x^{(i)}) - \frac{1}{m}\sum_i f_w(g_{\boldsymbol{\theta}}(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Amostre $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$
  $g_{\boldsymbol{\theta}} \leftarrow -\nabla_{\boldsymbol{\theta}} \frac{1}{m}\sum_i f_w(g_{\boldsymbol{\theta}}(z^{(i)}))$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \text{RMSProp}(\boldsymbol{\theta}, g_{\boldsymbol{\theta}})$
end while

Defaults do paper: $\alpha = 5 \times 10^{-5}$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$. Adaptado de (Arjovsky, Chintala e Bottou, 2017).

WGAN-GP – Gradient Penalty[8]

Weight clipping impõe Lipschitz, mas reduz a capacidade do crítico (pesos saturam em $\pm c$). **WGAN-GP** substitui o clipping por um termo de penalidade na loss:

$$L_{\text{GP}} = \underbrace{\sum_j f[g(z^{(j)}), \phi] - \sum_i f[x^{(i)}, \phi]}_{\text{loss WGAN original}} + \lambda \, \mathbb{E}_{\hat{x}} \Big[\left(\|\nabla_{\hat{x}} f[\hat{x}, \phi]\|_2 - 1 \right)^2 \Big]$$

- $\hat{x} = \alpha \, x + (1 - \alpha) \, x^*$ com $\alpha \sim \mathcal{U}(0, 1)$: pontos interpolados entre real e gerada.
- Penaliza norma do gradiente diferente de 1, isto é, Lipschitz é atingida na intensidade certa, não no extremo.
- Permite voltar a usar Adam e remover BatchNorm do crítico. $\lambda = 10$ funciona bem na prática.
- Hoje é o default quando se fala em "WGAN".

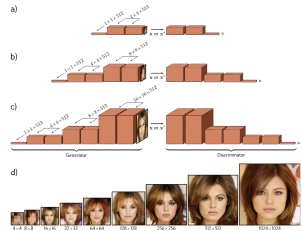
[8] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". Em: NeurIPS. vol. 30. 2017.

Visão Geral

Trick 1: Progressive growing[9]

- Começamos treinando GAN em resolução baixa (4×4).
- À medida que o treino avança, vamos **adicionando camadas** para dobrar a resolução até 1024×1024.
- Transições suaves entre resoluções evitam instabilidade.
- Resultado: imagens de altíssima qualidade (faces CelebA-HQ).

Trick 1: Progressive growing



[9] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. Em: ICLR. 2018.

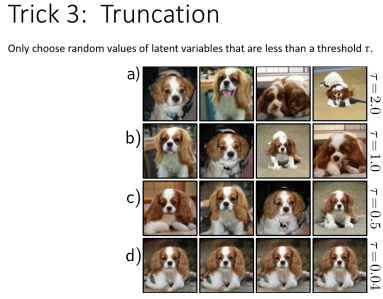
Trick 2: *Minibatch discrimination*[10]

- Adicione **estatísticas do *minibatch*** como entrada extra do discriminador.
- Exemplos: variância intra-batch, distâncias par-a-par entre amostras.
- **Efeito**: força o batch gerado a ter variabilidade semelhante ao batch real.
  - Ataca diretamente o colapso de moda: amostras muito parecidas passam a ser detectáveis.

[10] Tim Salimans et al. "Improved Techniques for Training GANs". Em: NeurIPS. vol. 29. 2016.

Trick 3: Truncation[11]

- Na hora de amostrar, só aceite z cujo módulo seja menor que um limiar τ.
- τ grande → mais diversidade, menos qualidade.
- τ pequeno → mais qualidade, menos diversidade.
- Permite controlar o trade-off qualidade/diversidade sem retreinar a rede.



[11] Andrew Brock, Jeff Donahue e Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". Em: ICLR. 2019.

Visão Geral

Como avaliar uma GAN?

Não há log-verossimilhança para reportar. Métricas usuais:

- **Inception Score**[12]

$$\text{IS} = \exp(\mathbb{E}_{x^*}[\, D_{KL}(p(y \mid x^*) \,\|\, p(y))\,])$$

  Recompensa imagens de classe nítida ($p(y \mid x^*)$ concentrada) e diversidade entre amostras ($p(y)$ espalhada). Limitação: depende do classificador (Inception em ImageNet).

- **Fréchet Inception Distance (FID)**[13]: ajusta gaussianas em features do Inception e calcula

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{tr}\left(\Sigma_r + \Sigma_g - 2\sqrt{\Sigma_r \Sigma_g}\right)$$

  Mais robusto que IS, é o padrão atual; valores menores são melhores.

Regra de bolso

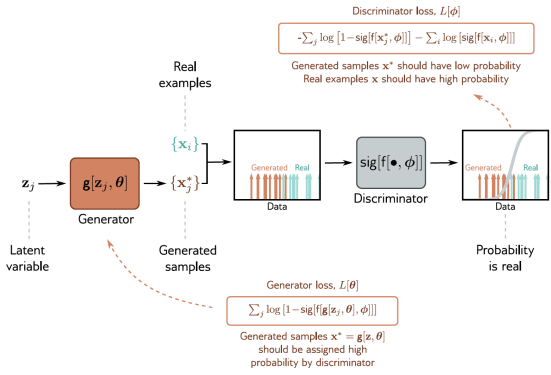
Ambas as métricas são sensíveis ao tamanho da amostra.

[12] Tim Salimans et al. "Improved Techniques for Training GANs". Em: NeurIPS. vol. 29. 2016.

[13] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". Em: NeurIPS. vol. 30. 2017.

TL;DR sobre GANs

- Existem diversas arquiteturas com algum controle sobre as imagens geradas:
  - *Conditional GANs*
  - *StyleGAN*, *BigGAN*, …
- **Pontos fortes**
  - Resultados bastante realistas.
  - Inferência “rápida” (1 passagem pelo gerador).
- **Pontos fracos**
  - Difíceis de treinar.
  - Baixa cobertura de possibilidades (mode collapse).



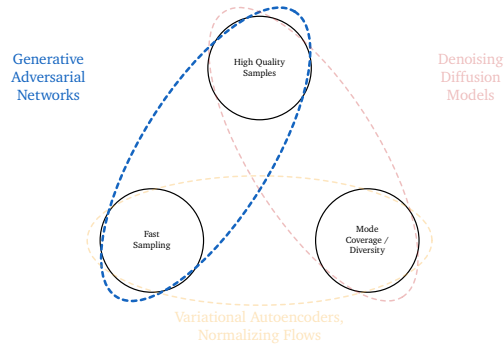
O trilema dos modelos geradores

Todo modelo gerador precisa trocar três objetivos:

- **Qualidade** das amostras.
- **Cobertura** / diversidade de modos.
- **Velocidade** de amostragem.

Posicionamento

- *GANs*: qualidade + velocidade.
- *Diffusion Models*: qualidade + cobertura.
- *VAEs/Normalizing Flows*: velocidade + cobertura.



Leituras Recomendadas

- Capítulo 15: Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023
- Ian Goodfellow et al. “Generative Adversarial Nets”. Em: NeurIPS. vol. 27. 2014
- Alec Radford, Luke Metz e Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. Em: arXiv preprint arXiv:1511.06434 (2015)
- Martin Arjovsky, Soumith Chintala e Léon Bottou. “Wasserstein Generative Adversarial Networks”. Em: ICML. vol. 70. 2017, pp. 214–223
- `https://vincentherrmann.github.io/blog/wasserstein/`

Referências I

[1] Martin Arjovsky e Léon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. Em: ICLR. 2017.

[2] Martin Arjovsky, Soumith Chintala e Léon Bottou. “Wasserstein Generative Adversarial Networks”. Em: ICML. Vol. 70. 2017, pp. 214–223.

[3] Andrew Brock, Jeff Donahue e Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. Em: ICLR. 2019.

[4] Ian Goodfellow et al. “Generative Adversarial Nets”. Em: NeurIPS. Vol. 27. 2014.

[5] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. Em: NeurIPS. Vol. 30. 2017.

[6] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. Em: NeurIPS. Vol. 30. 2017.

[7] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. Em: CVPR. 2017.

[8] Tero Karras, Samuli Laine e Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. Em: CVPR. 2019.

[9] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. Em: ICLR. 2018.

[10] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

[11] Alec Radford, Luke Metz e Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. Em: arXiv preprint arXiv:1511.06434 (2015).

[12] Tim Salimans et al. “Improved Techniques for Training GANs”. Em: NeurIPS. Vol. 29. 2016.

[13] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”. Em: ICCV. 2017.