UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Fabio Moreira

GERAÇÃO DE IMAGENS EM SUPER RESOLUÇÃO COM REDES GERADORAS ADVERSÁRIAS

Florianópolis

2019

Fabio Moreira

GERAÇÃO DE IMAGENS EM SUPER RESOLUÇÃO COM REDES GERADORAS ADVERSÁRIAS

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Mauro Roisenberg
Coorientador: Prof. Dr. Alexandre Gonçalves Silva

Florianópolis

2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Moreira, Fabio
Geração de imagens em super resolução com Redes Geradoras Adversárias / Fabio Moreira ; orientador, Mauro Roisenberg, coorientador, Alexandre Silva, 2019.
87 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2019.

Inclui referências.

1. Ciências da Computação. 2. Modelos Geradores. 3. Aprendizagem de Máquina. 4. Geração de Amostras. 5. RAGs. I. Roisenberg, Mauro. II. Silva, Alexandre. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Fabio Moreira

GERAÇÃO DE IMAGENS EM SUPER RESOLUÇÃO COM REDES GERADORAS ADVERSÁRIAS

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 14 de julho 2019.

Prof. Dr. José Francisco Danilo de Guadalupe Correa Fletes
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Mauro Roisenberg
Orientador

Prof. Dr. Alexandre Gonçalves Silva
Coorientador

Prof. Dr. Maicon Rafael Zatelli

À família, amigos e a todos os jogos que despertaram em mim o desejo de trilhar esse caminho...

AGRADECIMENTOS

Agradeço aos meus pais por todo apoio e suporte dado durante todos esses anos, não apenas no curso, mas na vida. Agradeço aos meus tios por permitir que eu residisse em sua casa durante todo o curso, além de toda a ajuda oferecida, muitas vezes mais do que eu merecia.

Agradeço aos grandes amigos e colegas que percorreram esse caminho junto comigo, todos os momentos de descontração e conversas aleatórias que certamente contribuíram para aliviar o espírito e aguentar mais um semestre. Agradeço em especial à Dúnia e o Vinicius, por me aguentar durante esses cinco longos anos, por todas as as nossas discussões, de vez em quando longas e calorosas, seja do curso seja dos mais diversos e aleatórios temas.

Agradeço a todos os colegas do LabSEC, primordialmente o Lucas, por criar um ambiente descontraído, com muito aprendizado e verdadeiro trabalho em equipe. Em especial ao Palma, por compreender e me liberar em todos os momentos de necessidade.

Agradeço aos professores Mauro e Alexandre pela orientação deste trabalho, todas as revisões e auxílios prestados.

We all make choices, but in the end, our choices make us.

Andrew Ryan

RESUMO

A geração de novas amostras é um desafio atual da inteligência artifical. Os modelos geradores tentam há muito tempo resolver esse problema, porém, a complexidade enfrentada era tamanha que poucos modelos foram desenvolvidos. Com o aumento do poder computacional e das técnicas de aprendizagem profunda, essa área floresceu novamente. As Redes Geradoras Adversárias (RAGs) representam o estado da arte dos modelos geradores, fornecendo amostras de alta qualidade. Entretanto, a qualidade dos materiais existentes afeta o pleno entendimento dessa técnica. Neste trabalho, são vistos os principais aspectos teóricos e práticos desses modelos, com a realização de experimentos para suportar e validar a teoria. Por fim, uma RAG estado da arte é utilizada para resolver um problema atual: a super resolução de imagens, área que aprimora a resolução de um sistema de imagens.

Palavras-chave: Modelos Geradores. Aprendizagem de Máquina. Geração de Amostras. RAGs.

ABSTRACT

The generation of new samples is a current challenge of artificial intelligence. Generator models have tried for a long time to solve this problem, but the complexity faced was such that few models were developed. With increasing computational power and deep learning techniques, this area flourished again. The Generator Adversarial Networks (GANs) represent the state of the art generator models, providing high quality samples. However, the quality of existing materials affects the full understanding of this technique. In this work, all the main theoretical and practical aspects of these models are seen, with the realization of experiments to support and validate the theory. Finally, a GAN state of the art is used to solve a current problem: super-resolution imaging, field that enhance the resolution of an imaging system.

Keywords: Generative Models. Machine Learning. Image Generation. GANs.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

O paradigma conexionista da inteligência artificial propõe que os computadores repliquem a forma de obtenção do conhecimento e a habilidade de resolução de problemas empregando modelos gerados a partir de observações da natureza. Desse contexto surgem os modelos geradores, esses modelos representam a habilidade de produzir novas variáveis pela análise de características que podem ser inferidas de variáveis existentes.

Um conjunto de dados pode ser representado por uma distribuição de probabilidade. Desse modo, os modelos geradores representam distribuições de probabilidade sobre múltiplas variáveis (GOODFELLOW; BENGIO; COURVILLE, 2016), definindo como estratégia encontrar a "real" distribuição dos dados. Ao encontrar essa distribuição, será possível gerar novas variáveis que, por pertencerem a mesma distribuição, são similares, mas não iguais, as variáveis existentes.

As *Boltzmann Machines* (FAHLMAN; HINTON; SEJNOWSKI, 1983; ACKLEY; HINTON; SEJNOWSKI, ; HINTON; SEJNOWSKI, 1986; HINTON; SEJNOWSKI; ACKLEY, 1984) foram uma primeira tentativa de implementação dos modelos geradores. Esse grupo de técnicas usa cadeias de Markov para o treinamento e para a geração de amostras do modelo. Com isso, ela assimila distribuições de probabilidade de vetores binários e, dessa forma, consegue modelar relações lineares entre variáveis (GOODFELLOW; BENGIO; COURVILLE, 2016). Embora historicamente importante, seu uso tornou-se raro pois não foi possível estender essa técnica para os problemas atuais, como geração de imagens. Complexos e exigindo muito poder computacional, os modelos geradores foram ignorados em sua época.

O desenvolvimento do *deep learning*, reduzindo a complexidade computacional das implementações, em conjunto com a evolução das unidades de processamento gráfico (GPUs), elevando o poder computacional disponível, proporcionou o renascimento dos modelos geradores. Uma técnica fruto dessas inovações são as *Generative Adversarial Networks* (GANs) (GOODFELLOW et al., 2014).

As GANs são compostas de duas redes neurais, uma discriminadora e uma geradora, competindo entre si. Há inúmeras pesquisas apresentando excelentes resultados em diferentes áreas tais como geração de imagens (KARRAS et al., 2017a) e vídeos (TULYAKOV et al., 2018), processamento de textos (REED et al., 2016) e música (YANG; CHOU; YANG, 2017). Há vários problemas em aberto, visto que é uma

tecnologia nova, sendo os mais relevantes o elevado tempo de treinamento, falta de convergência, de métricas de avaliação e de critério de parada.

GANs são uma tecnologia recente e se encontram na vanguarda de pesquisas envolvendo modelos geradores. O trabalho de Goodfellow (2017), desenvolvido com a intenção de ser um tutorial e utilizado como base desse trabalho, é o único a reunir e discutir os principais elementos das GANs. Porém, não apresenta experimentos e discussões práticas sobre o assunto, comparações com as arquiteturas estado da arte, novas funções de custo e métricas de avaliação desenvolvidas. Desse modo, este trabalho se propõe a realizar um estudo detalhado sobre as GANs, considerando sua arquitetura e os principais conceitos teóricos que as sustentam, vantagens e desvantagens sobre outros modelos geradores, a etapa de treinamento, os principais problemas em aberto, suas limitações e o estado da arte. Este trabalho não se propõe a desenvolver novas ferramentas ou técnicas, tampouco desenvolver um software funcional. Todas as implementações criadas servirão de experimentos com o objetivo de contextualizar com as questões teóricas discutidas ao longo do trabalho.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Realizar um estudo detalhado sobre os aspectos teóricos e práticos das GANs, reunindo em um só trabalho as principais contribuições conhecidas. Desenvolver experimentos que replicam técnicas conhecidas de otimização do treinamento. Implementar técnicas estado da arte desenvolvidas para aperfeiçoar as GANs.

1.1.2 Objetivos Específicos

- Elucidar os aspectos teóricos e práticos das GANs.
- Descrever a arquitetura DCGAN, implementação estado da arte.
- Desenvolvimento de experimentos, com o objetivo de:
  - Relacionar a teoria com a prática.

  - Propor discussões sobre assuntos teóricos que necessitam de contextos práticos.
  - Analisar o impacto de alterações na arquitetura.
  - Analisar as limitações da arquitetura.
  - Validar técnicas propostas para melhorar o treinamento.
  - Discutir os resultados obtidos.

- Analisar arquiteturas de uma GAN, a DCGAN, que possue treinamento estável.
- Organizar, didaticamente, o conteúdo disponível sobre GANs. Esse conteúdo engloba aspectos teóricos já fundamentos na literatura, comparações com outros modelos, arquitetura, treinamento, resultados de experimentos práticos, discussões e dilemas sobre certas questões teóricas e práticas.
- Aplicar uma GAN estado da arte, a SRGAN, como uma técnica de melhoramento de imagem, na área de super resolução de imagens.

1.2 ESTRUTURAÇÃO DO TRABALHO

O Capítulo 2 trata dos principais conceitos teóricos consolidados na literatura, sendo pré-requisito para o pleno entendimento das GANs. O Capítulo 3 apresenta algumas técnicas na área de modelos geradores, expondo diferentes tipos de abordagens. O Capítulo 4 elucida o funcionamento das GANs, apresentando também pesquisas em aberto, tipos de GANs, a DCGAN (GAN estado da arte), dicas de treinamento e finalizando com uma comparação das GANs em relação aos modelos vistos no Capítulo 3. O Capítulo 5 promove discussões teóricas com contexto prático através dos resultados dos experimentos realizados. O Capítulo 6 é dedicado à discutir os resultados obtidos da aplicação da SRGAN. Por fim, o Capítulo 7 conclui o trabalho, indicando possíveis caminhos a serem seguidos em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para compreender a teoria, a arquitetura e a contribuição das GANs é necessário possuir um conjunto de conhecimentos já consolidados na literatura que permitiram a sua criação.

2.1 MODELOS GERADORES

O termo **modelos geradores** possui diversas definições. Neste trabalho, a definição será a mesma empregada pelo idealizador das GANs. Assim, segundo (GOODFELLOW, 2017, p. 2) "um modelo gerador pode ser definido como um modelo que, dado um conjunto de treinamento $\boldsymbol{p_{data}}$, aprende a representar uma estimativa dessa distribuição de alguma forma. O resultado é a distribuição de probabilidade $\boldsymbol{p_{model}}$".

No contexto das GANs, dado a entrada vetorial z, chamada de código latente (definição na seção 2.4), como entrada da função diferenciável G, ou modelo gerador, o objetivo será encontrar parâmetros θ (pesos e vieses da rede) que produzem uma distribução de probabilidade próxima da verdadeira distribuição. A Figura 1 ilustra esse processo. Com isso, a entrada z será manipulada para se encaixar dentro da distribuição gerada e o resultado será uma amostra que, se o modelo gerado for semelhante ao real, também pertencerá a distribuição real.

2.2 MANIFOLDS E SUPORTE

Um manifold pode ser definido como um espaço topológico com a propriedade de se assemelhar localmente a um espaço euclidiano (ROWLAND, n.d.). É possível existir manifolds de várias dimensões, denominando-se nesse caso um **n-manifold**.

Para exemplificar, considere um observador contemplando o horizonte e imaginando a forma da Terra. Do ponto de vista dele, a Terra parece plana, mas se ele andar sempre em linha reta, eventualmente, retornará ao ponto de partida. Isso acontece porque a superfície da terra é um manifold de duas dimensões (2-manifold). Localmente, ela se assemelha a uma forma euclidiana (plano), embora globalmente ela seja uma esfera.

O suporte de uma função f em um espaço topológico X, ou

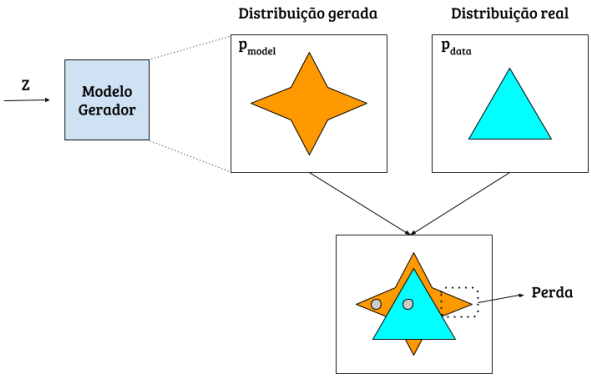


Figura 1 – Estrutura de um modelo gerador e comparações entre distribuições. Dificilmente, na prática, a distribuição gerada será igual à distribuição real, havendo perdas no processo. Considerando os pontos cinzas como amostras de uma distribuição, se um discriminador perfeito analisasse duas amostras da distribuição gerada, como mostrado na parte da figura que faz a sobreposição entre as distribuições, uma delas seria rejeitada, pois não está relacionada à distribuição real. Fonte: Produzida pelo autor.

apenas suporte, pode ser definido como o menor conjunto fechado onde $f \neq 0$ (RUDIN, 1966). Para esse trabalho, X é um espaço topológico e $f : X \to \mathbb{R}$ é uma função. O suporte de f será uma distribuição de probabilidade real ou do modelo, denotada por p_r e p_g, respectivamente.

2.3 EQUILÍBRIO DE NASH

Na teoria dos jogos, o equilíbrio de Nash (NASH, 1950) representa um determinado estado, em um jogo com dois ou mais jogadores, onde cada jogador sabe que o outro está aplicando a melhor estratégia possível e, assim, não teria nada a ganhar mudando a sua estratégia.

Embora os participantes não cooperem, o resultado de cada indivíduo pode levar o jogo a uma estabilidade, de forma que não exista um motivo para qualquer um dos jogadores alterarem seu modo de jogo.

Um exemplo simples para esse conceito é o dilema do prisioneiro, representado na Tabela 1. Neste problema, dois prisioneiros cometaram um crime e cada um possui duas opções: Confessar ou negar o crime. O melhor cenário, individualmente, é que um prisioneiro confesse e o

outro negue o crime, resultando na liberdade do primeiro. Porém, se os dois confessam, o resultado não será o mais favorável para nenhum dos dois. Assim, mesmo em um ambiente de competição, para se obter o resultado mais favorável para ambos, o melhor cenário é a cooperação. Neste caso, se ambos negarem, embora não seja o melhor resultado, não haverá a possibilidade do outro também confessar só para prejudicar aquele que confessou primeiro. A cooperação pode ser obtida através do equilíbrio.

		Prisioneiro 2	
		Negar	*Confessar*
Prisioneiro 1	*Negar*	2 anos, 2 anos	4 anos, livre
	Confessar	livre, 4 anos	3 anos, 3 anos

Tabela 1 – Representação do dilema do prisioneiro. A melhor escolha para ambos é a cooperação.

2.4 ESPAÇO LATENTE E CÓDIGO LATENTE

Em um conjunto de dados, há subcojuntos que compartilham características semelhantes. Em determinados casos, como o dos modelos geradores, é necessário realizar um mapeamento dessas similaridades em um novo espaço, de modo a construir novos dados pela alteração de algumas dessas características. Espaço latente é o local onde as características de uma distribuição residem.

O Exemplo 1 é amplamente utilizado na área de processamento de linguagem natural, válido também para este trabalho, para entender a definição de espaço latente.

Exemplo 1

Considere um vetor Dicionário = [‘I’, ‘love’, ‘to’, ‘hotel’, ‘motel’, ‘Sleep’]. Abaixo, um possível exemplo de mapeamento.

$$\text{I} = [1, 0, 0, 0, 0, 0]$$
$$\text{love} = [0, 1, 0, 0, 0, 0]$$

$$\begin{aligned} \text{to} &= [0, 0, 1, 0, 0, 0] \\ \text{hotel} &= [0, 0, 0, 1, 0, 0] \\ \text{motel} &= [0, 0, 0, 0, 1, 0] \\ \text{Sleep} &= [0, 0, 0, 0, 0, 1] \end{aligned}$$

Esse mapeamento é conhecido como *One-hot enconding*, um vetor de uma dimensão com o valor 1 representando a palavra. Note que, embora as palavras hotel e motel possuem significados semânticos similares, eles não se diferem das demais palavras que não compartilham de definições similares. Agora, considere um mapeamento que necessita associar dados de um mesmo conjunto baseado em suas características. Um espaço análogo a um sistema de coordenadas é interessante, pois dados com características semelhantes são alocados próximos uns aos outros. Assim, podemos considerar esse novo mapeamento como, por exemplo, um sistema de coordenadas de duas dimensões:

$$\begin{aligned} \text{I} &= [2; 6] \\ \text{love} &= [7; 2] \\ \text{to} &= [0; 3] \\ \text{hotel} &= [3; 5] \\ \text{motel} &= [3; 5{,}25] \\ \text{Sleep} &= [9; 0] \end{aligned}$$

Nesse novo esquema, o mapeamento considera a semântica das palavras e as insere nesse espaço, chamado de espaço latente, baseado nessa propriedade.

Código latente pode ser definido como um vetor z que acessa as características de uma base de dados que foram mapeadas para o espaço latente. Utilizando o código latente é possível obter novas amostras que possuem combinações das características armazenadas nesse vetor. Entretanto, o simples mapeamento de z para o código latente apenas garante a recuperação de amostras da base de dados, mas não a geração de novas amostras. Ainda no Exemplo 1, um algoritmo deseja criar uma nova amostra relacionada semanticamente às hospedarias. Para isso, ele seleciona a amostra [3; 5] do espaço latente do exemplo e adiciona um ruído (um valor que irá alterar o valor original), resultando no valor [3,2 ; 5,15]. Esse valor, ao ser mapeado de volta ao conjunto Dicionário, resulta em um hostel, ou seja, é uma nova amostra que é semanticamente semelhante aos elementos originais do conjunto. Como será visto nas próximas seções, o valor de z é aleatório, amostrado de uma distribuição de probabilidade qualquer. Como z é um vetor desestruturado, não é possível gerar amostras com as características desejadas pelo usuário.

2.5 CONSISTÊNCIA ASSINTÓTICA

Um estimador é dito ser assintoticamente consistente se, conforme o número de amostras aumenta, as estimativas resultantes se aproximam do valor real que está sendo estimado. A Figura 2 exemplifica o conceito. Essa é uma propriedade desejada em modelos geradores, pois quanto mais amostras de uma distribuição estiverem disponíveis, maiores as chances do modelo gerado resultar em uma representação fiel à distribuição original.

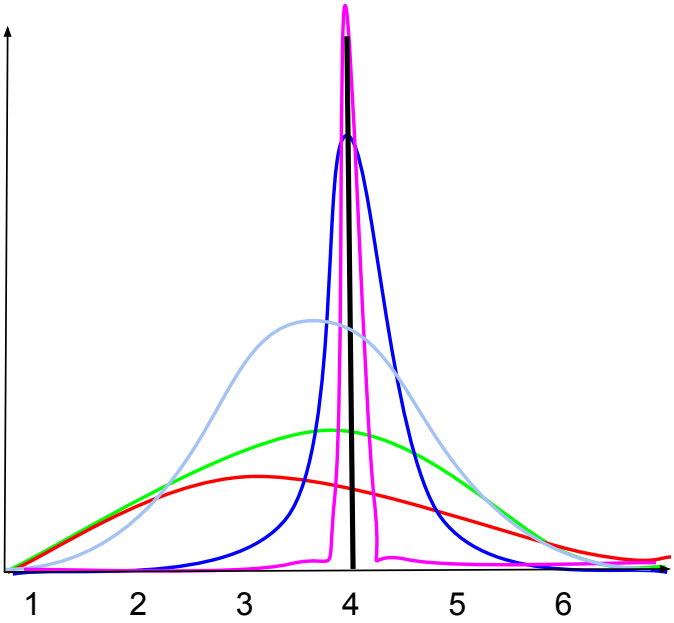


Figura 2 – Exemplo de um estimador sobre um parâmetro $\theta = 4$. Quanto mais dados o estimador obtém, mais próximo o valor estimado se aproxima do real. Fonte: Produzido pelo autor.

3 TAXONOMIA DOS MODELOS GERADORES

GANs não foram os primeiros modelos geradores criados e não são os únicos estudados atualmente. Há várias famílias de modelos que, embora não sejam tão boas quanto as GANs, apresentam resultados teóricos importantes para o avanço dessa área. Nesta seção, os principais modelos são descritos de acordo com a taxonomia da Figura 3.

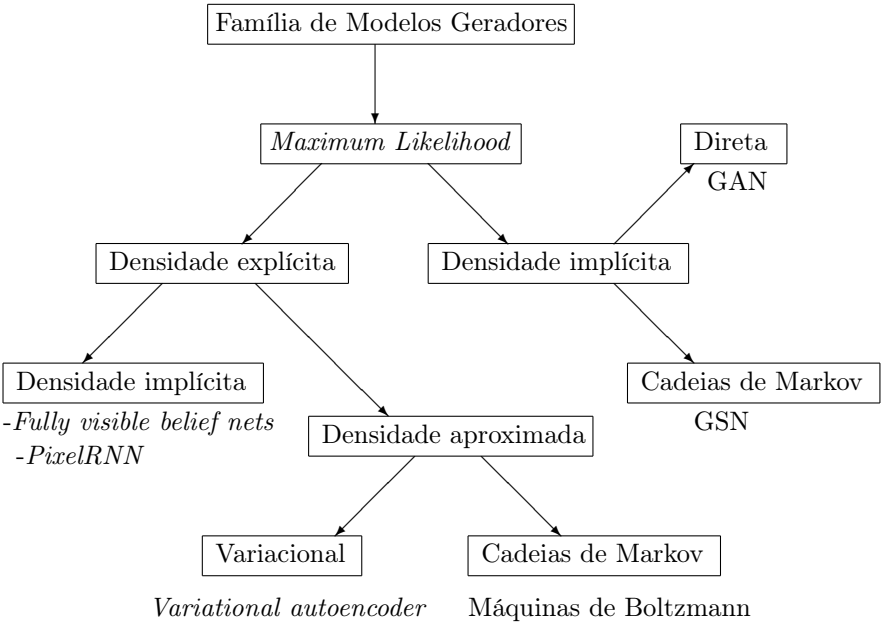


Figura 3 – Taxonomia dos modelos geradores. Apesar dos modelos empregarem outras técnicas em vez de *maximum likelihood*, é possível utilizá-la em todos eles. Assim, para evitar que as diferenças de cada modelo atrapalhem a análise, a técnica comum entre eles foi a escolhida. Fonte: Adaptado de (GOODFELLOW, 2017).

3.1 DENSIDADE EXPLÍCITA

Modelos de densidade explícita são modelos que, dado um conjunto de treinamento, devem explicitamente definir uma distribuição de probabilidade p_{model} que mais se aproxima da verdadeira distribuição daquele conjunto.

O principal problema é capturar, em um único modelo, todas as características do conjunto sem torná-lo intratável (GOODFELLOW, 2017). Para resolver isso, duas estratégias foram adotadas.

A primeira, chamada de densidade tratável, visa capturar todas as características e complexidades inerentes ao conjunto até o ponto onde o modelo seja tratável. A segunda, chamada de densidade aproximada, cria um modelo intratável, porém admite uma aproximação.

3.1.1 *Fully visible belief nets*

As *Fully visible belief nets* (FREY; HINTON; DAYAN, 1996; FREY, 1998), ou FVBNs, são um caso especial das máquinas de Helmhotz. Essa classe de modelos pode ser melhor definida com a discussão do seu modelo mais popular.

PixelRNN (OORD; KALCHBRENNER; KAVUKCUOGLU, 2016) é um modelo da família das FVBNs, a proposta desse modelo é produzir uma distribuição que estime corretamente a distribuição real de um conjunto, onde seja possível gerar novas imagens, mantendo a complexidade em um nível tratável.

O objetivo é predizer o próximo pixel baseado nos píxeis anteriores. Isso pode ser resumido na Equação 3.1.

$$p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}) \tag{3.1}$$

Onde $p(x)$ é a probabilidade conjunta, ou p_{model}, atribuída a uma imagem x composta de $n \times n$ píxeis e $p(x_i|x_1, ..., x_{n-1})$ é a probabilidade do pixel x_i dado os píxeis anteriores a ele.

O principal problema com a família FVBNs em geral é a geração de novas imagens. Para produzir uma nova amostra, técnicas dessa família precisam gerar um pixel por vez.

3.1.2 *Variational autoencoders*

Variational autoencoders (KINGMA, 2013; KINGMA; SALIMANS; WELLING, 2016), ou VAEs, são modelos que utilizam aproximações variacionais e um dos modelos mais populares, junto com as FVBNs e GANs (GOODFELLOW, 2017).

Sua arquitetura pode ser resumida na Figura 4. Um conjunto de imagens é passada a um codificador que irá mapear cada imagem a um par de valores, média e desvio padrão, e os armazenará em dois vetores. A média e o desvio padrão servem como rótulos de uma distribuição. Por exemplo, o par de valores [5,24; 3,24] representa um gato e o par [1,24; 0,22] uma mesa. Assim, para gerar uma nova imagem, a média e o desvio padrão são combinados, devendo seguir uma certa restrição que segue uma distribuição gaussiana, passados pelo decodificador e, após esse processo, uma nova imagem é gerada.

Uma característica importante dessa técnica é a rotulação das imagens. Nas VAEs há restrições sobre as características das imagens. Dessa forma, um gato não pode possuir três orelhas ou um sapo ter um chifre, pois seus valores de média e desvio padrão estarão afastados no espaço latente. Ao gerar uma nova imagem, um par de valores (média, desvio padrão) será selecionado desse espaço, mas não haverá combinações de pares de valores pertencentes a esse espaço, impedindo resultados tais como os citados acima. Assim, ao selecionar um vetor qualquer e adicionar ruído a ele, é possível obter uma imagem semelhante a original, mas sem extrapolar características básicas da imagem.

3.1.3 *Boltzmann machines*

Boltzmann machines são redes estocásticas de aprendizado não-supervisionado sobre vetores binários, ou seja, dado um vetor binário como entrada da rede, o método é capaz de, sem auxílio externo, aprender[1] as representações internas dessa rede. Essas redes foram as primeiras capazes de resolver problemas combinatórios difíceis. Uma propriedade interessante é a possibilidade da alteração da rede para se resolver problemas de busca ou de aprendizado.

A questão com essas redes é o fato de não escalar quando o conjunto é composto de imagens, ou melhor, distribuições de probabilidade de alta dimensionalidade. Em vários momentos, ou o modelo não con-

[1]Para a área da inteligência artificial, aprender significa adquirir informações e regras para utilizá-las.

verge em tempo razoável, levando o usuário a finalizar o treinamento antes da hora, ou o modelo não aprende corretamente. O principal representante dessa família atualmente são as *Restricted Boltzmann Machines*.

As *Restricted Boltzmann Machines* (SMOLENSKY, 1986) facilitam o aprendizado ao restringir a conectividade entre as camadas da rede neural. Nessas redes, há apenas duas camadas: a primeira, chamada de visível, que está relacionada aos componentes de uma entrada, um nodo da camada visível por pixel de uma imagem, por exemplo; a segunda, chamada escondida, modela a dependência entre os componentes da entrada, sendo que a comunicação se dá apenas entre camadas e não intracamadas. Essa diferença permitiu a aplicabilidade dessa técnica a problemas de classificação, regressão, aprendizado entre outros. A geração de um modelo de distribuição de probabilidade para tentar representar a distribuição original, chamado aqui de reconstrução, é realizado pela fase de retropropagação, que atualiza os pesos dos nodos na camada visível da rede.

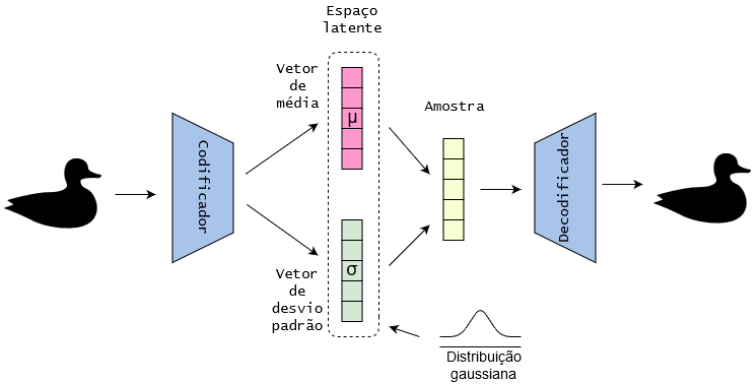


Figura 4 – Arquitetura base de uma VAE. O codificador processa e armazena um par, média e desvio padrão, da imagem. Em seguida, esse par é amostrado e deve-se encaixar em uma distribuição gaussiana, o resultado desse processo é levado ao decodificador, gerando uma nova imagem. Fonte: Produzida pelo autor.

3.2 DENSIDADE IMPLÍCITA

Oposto aos modelos de densidade explícita, os modelos de densidade implícita são modelos que, dado um conjunto de dados, não necessitam que a distribuição do conjunto seja dada explicitamente ou seja descoberta pela algoritmo de aprendizado, apenas amostrando imagens do conjunto é suficiente para o treinamento. Em muitos casos, o modelo é construído iterativamente, como é o caso das GANs. Ao gerar novas amostras, há uma avaliação dessas e o resultado retorna ao modelo gerador, que fará as devidas correções no modelo.

3.2.1 *Generative Stochastic Networks*

As *Generative Stochastic Networks* (GSNs) constroem um modelo de distribuição de probabilidade baseado na observação de um operador de transição de uma cadeia de Markov, cuja distribuição estacionária estima a distribuição dos dados (BENGIO; THIBODEAU-LAUFER; YOSINSKI, 2013).

Dado um conjunto X de imagens amostrado de uma distribuição de probabilidade desconhecida P(X), encontrar P(X) pela corrupção das variáveis X (denominadas $\tilde{X}$), gerando uma distribuição C($\tilde{X}$| X). Após gerada essa distribuição, são utilizadas técnicas de aprendizado supervisionado para treinar uma função que reconstroi X dado $\tilde{X}$. A distribuição gerada dessa reconstrução é menos computacionalmente complexa de modelar do que P(X). Encontrado essa distribuição de reconstrução, a aproximação para P(X) segue uma regra bayesiana (BENGIO; THIBODEAU-LAUFER; YOSINSKI, 2013).

O algoritmo final utiliza cadeias de Markov que vão amostrar pares de valores (X, $\tilde{X}$), além de, iterativamente, realizar alteração nos parâmetros que afetam toda a cadeia de treinamento.

Uma contribuição dessa técnica é a generalização da interpretação dos *denoising auto-enconders*, uma técnica que visa reconstruir os dados de uma entrada corrompida, adicionando o conceito de variáveis latentes para, além de reconstruir, gerar novas entradas. Por esse fato, a utilização das GSNs é focada na área de recuperação de valores perdidos ou corrompidos.

Na questão da densidade, as GSNs estimam a distribuição de dados gerada indiretamente, em vez de tentar gerar p_{model} diretamente. Com isso, é possível evitar a computação de funções intratáveis sem a necessidade de gerar previamente o modelo.

4 REDES GERADORAS ADVERSÁRIAS

Antes da discussão sobre o funcionamento das GANs, é necessário analisar as diferenças entre os modelos discriminadores e geradores.

Os modelos discriminadores resolvem problemas de classificação. Dado um conjunto de treinamento e um conjunto de rótulos, o modelo tentará relacionar corretamente cada instância do conjunto de treinamento a uma instância do conjunto de rótulos. Matematicamente, esse problema pode ser modelado como: considerando y um rótulo e x um conjunto de características extraídas do conjunto de treinamento, pode-se definir a relação de x e y como $p(y|x)$, ou seja, qual a probabilidade de y dado x. Para exemplificar, considere um conjunto de treinamento composto dos números zero a nove. É tarefa do discriminador determinar se, dado um número desse conjunto, ele é ou não o número nove.

Os modelos geradores atuam de forma oposta. Dados os mesmos dois conjuntos definidos acima, o modelo tentará aprender quais características são necessárias para definir um certo rótulo. Aqui, a relação entre y e x é inversa e dada por $p(x|y)$, que expressa a probabilidade de x dado y. Voltando ao exemplo, é tarefa do gerador determinar quais características extraídas do conjunto são necessárias e suficientes para considerar um certo número como sendo o número nove.

4.1 ARQUITETURA

As GANs são compostas por dois modelos:

- Um discriminador **D**, treinado para diferenciar amostras que não pertencem à distribuição real das que pertencem. Esse modelo é uma função diferenciável $D(x, \theta_d)$, onde x é a amostra a ser analisada considerando um conjunto de parâmetros configuráveis θ_d, que resulta em um único valor de saída, geralmente esse valor está no intervalo [0,1], sendo 0 para amostras que o discriminador tem certeza que não pertencem à distribuição e 1 para amostras que ele tem certeza que pertencem.

- Um gerador **G**, treinado para construir uma distribuição p_{model} que é capaz de capturar, em um cenário ideal, todas as características da distribuição original p_{data}. Esse modelo também é uma função diferenciável $G(z, \theta_g)$, onde z é um vetor latente e θ_g são os parâmetros configuráveis de **G**.

A Figura 5 representa um esquemático da arquitetura das GANs. As distribuições do mundo real são complexas e demandam a análise de várias características das amostras para uma cópia fiel dessas distribuições. Em razão disso, o modelo é construído de forma iterativa e incremental. Inicialmente, algumas amostras x da distribuição original são extraídas para realizar o treinamento do modelo discriminador. Esse passo inicial permite que o discriminador classifique corretamente futuras amostras. Em seguida, um código latente z é extraído e passado para o gerador G, que tenta reproduzir a distribuição original, resultando em um p_{model}. Algumas amostras $\widetilde{x}$ do p_{model} são utilizadas para treinar o discriminador. O objetivo agora é treinar o discriminador com as amostras retiradas da distribuição p_{model} e esperar que essas sejam consideradas como amostras da distribuição original, maximizando o erro do discriminador. O resultado desse processo é utilizado para corrigir eventuais falhas de ambos os modelos, que refazem o procedimento considerando essas novas informações.

O processo pode ser visto como um jogo minimax composto de dois jogadores, onde o discriminador tenta maximizar a probabilidade de diferenciar corretamente as amostras e o gerador tenta minimizá-lo. A solução para esse problema é o equilíbrio de Nash. Em certo ponto do treinamento, o discriminador não é capaz de diferenciar amostras reais de falsas, ou seja, a saída sempre será $0,5$, em contrapartida, o gerador não alterará a distribuição p_{model}, pois seu objetivo de enganar o discriminador foi alcançado.

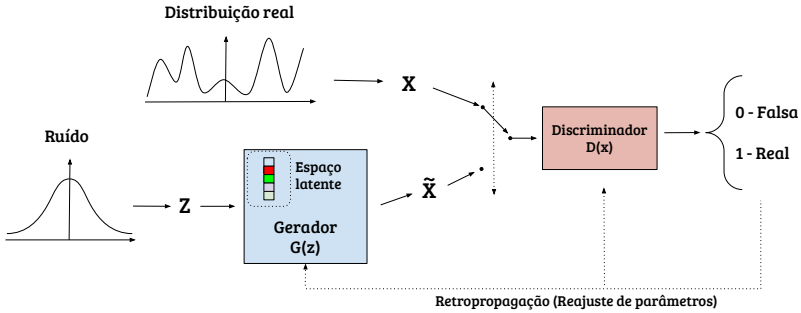


Figura 5 – Arquitetura de uma GAN. Fonte: Produzida pelo autor.

4.2 FUNÇÕES DE CUSTO

Concluída a primeira etapa do treinamento, se faz necessário avaliar os resultados obtidos pelo gerador, a distribuição p_{model} gerada, e os resultados do discriminador, a acurácia. Essa comparação pode ser quantificada através de funções de custo. As funções de custo exercem um papel crítico ao avaliar a diferença entre as duas distribuições de probabilidade, possibilitando o aperfeiçoamento do algoritmo ao indicar o impacto que novas mudanças causarão na qualidade dos resultados obtidos. Se os resultados gerados se diferem da realidade, ou seja, o modelo gerado não corresponde ao modelo real, as funções de custo penalizam a técnica ao retornar valores altos. Se os resultados se aproximam da realidade, os valores retornados são baixos. Para as GANs, os valores retornados pelas funções de custo são repassados aos otimizadores que minimizam o erro dessas funções, refinando o modelo.

4.2.1 Entropia cruzada

A entropia cruzada avalia modelos com saídas em forma de probabilidades com valores entre 0 e 1, comumente utilizada em problemas de classificação. Com exceção da função *Wasserstein*, as funções de custo apresentadas neste trabalho usam a entropia cruzada como função de custo do discriminador, diferindo apenas na função do gerador.

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}\, logD(x) - \frac{1}{2}\mathbb{E}_z\, log(1 - D(G(z))) \quad (4.1)$$

O objetivo do discriminador é maximizar a função de custo representada na Equação 4.1, onde $x \sim p_{data}$ é uma amostra x de uma distribuição p_{data}, z é uma amostra de uma distribuição conhecida e $\theta^{(D)}$ são os parâmetros do discriminador (pesos e vieses). Maximizá-la, nesse contexto, significa maximizar as chances do discriminador estar correto, consequentemente reduzindo o erros cometidos por ele e aumentando sua acurácia, resultando em um modelo mais preciso que consegue distinguir amostras reais das geradas.

Essa função possui algumas características importantes. Ela foi projetada para acelerar o aprendizado, penalizando fortemente tanto discriminadores muito confiantes, retornam com frequência 0 ou 1, quanto muito imprecisos, retornam 0 quando o correto é 1, além de evitar a saturação do gradiente. (NIELSEN, 2015)

4.2.2 Minimax

Como explicado na Seção 4.1, o gerador e o discriminador participam de um jogo, chamado minimax. Esse tipo de jogo também é conhecido como um jogo de soma zero, isto é, para um adversário ganhar o outro deve necessariamente perder, ou melhor, a soma dos custos de todos os jogadores é zero.

$$J^{(G)} = -J^{(D)} \tag{4.2}$$

A Equação 4.2 define a função de custo do gerador e vai ao encontro da definição de minimax, pois cabe ao gerador minimizar o erro de sua função, minimizando as chances do discriminador estar correto. Essa função de custo é interessante do ponto de vista da análise teórica, pois permite demonstrar que o jogo converge para o equilíbrio em dado momento.

Na prática, ela é pouco utilizada, pois o gradiente do gerador facilmente satura. Como o gerador minimiza e o discriminador maximiza a mesma função, se o discriminador conseguir classificar corretamente todas as amostras (o que acontece com bastante facilidade no início do treinamento, pois inicialmente o gerador executa com parâmetros aleatórios), o custo resultante fica próximo de zero, deixando o gerador sem gradiente para aperfeiçoar seu modelo.

4.2.3 Heurística de não-saturação

Considerando o problema da função da seção anterior, a solução encontrada foi mudar o alvo do gerador. Em vez de definir $J^{(G)}$ em termos do valor de sinal trocado do $J^{(D)}$, indicando uma relação de soma zero, utiliza-se a Equação 4.3.

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{z\sim p_{modelo}}\ log(D(G(z))) \tag{4.3}$$

Embora o gerador continue tentando enganar o discriminador, seu objetivo agora é maximizar as chances do discriminador estar errado. Com isso, é possível que, mesmo em situações onde o discriminador seja perfeito, o gerador obtém bons valores de gradientes, valores que permitem uma minimização do erro a uma velocidade considerável, permitindo um aperfeiçoamento mais rápido. Com essa mudança, o jogo deixa de ser soma zero.

4.2.4 *Maximum likelihood*

Ao comparar diferentes técnicas é interessante utilizar parâmetros iguais para minimizar os ruídos presentes nos resultados, causados pelas diferenças de cada implementação. Para os modelos geradores é comum, ao comparar diferentes técnicas, utilizar a mesma função de custo, sendo o *Maximum likelihood* a função mais popular para esse fim. A Equação 4.4 foi utilizada por Goodfellow et al. (2014) para permitir a comparação de outros modelos com as GANs.

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{z\sim p_{modelo}}\ exp(\sigma^{-1}(D(G(z)))) \tag{4.4}$$

Onde z é uma amostra da distribuição p_{model}, exp é a função exponencial, σ^{-1} é a função logit, inversa da função sigmoide.

4.2.5 *Wasserstein*

A função *Wasserstein* é uma modificação da distância de Wasserstein, uma medida de distância entre duas funções de probabilidade. A principal vantagem dessa função é prover resultados suficientes quando duas distribuições se encontram em *manifolds* de baixa dimensão, como mostra a Figura 6. Nesse cenário, duas distribuições dificilmente estão sobrepostas, o que impede que funções de custo tradicionais obtenham resultados satisfatórios. Isso ocorre pois essas funções apresentam um comportamente instável quando duas distribuições não estão sobrepostas, gerando resultados imprevisíveis que desestabilizam o modelo.

A Equação 4.5 define essa função.

$$W(p_r, p_g) = \max_{w\in W} \mathbb{E}_{x\sim p_r}[f_w(x)] - \mathbb{E}_{x\sim p_g}[f_w(g_\theta(z))] \tag{4.5}$$

Em relação às outras funções apresentadas, os logaritmos são trocados por funções f_w, pertencentes à família de funções contínuas K-Lipschitz.

Funções K-Lipschitz são funções do tipo $f : \mathbb{R} \to \mathbb{R}$ que seguem a restrição:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|, \text{ onde } K \geq 0 \tag{4.6}$$

No caso da função utilizada para o cálculo do custo, a restrição é que ela seja 1-Lipschitz, ou seja, $K = 1$. Agora, para calcular a função de custo é necessário encontrar essa função f. A forma descoberta para calculá-la foi alterar o papel do discriminador no processo, que agora passa a ser chamado de crítico. Antes, ele era treinado para diferenciar as amostras produzidas pelo gerador das imagens reais, agora, ele deve procurar pesos w que satisfaçam a restrição 1-Lipschitz. O problema é manter a função com essa restrição durante o treinamento, a solução encontrada foi limitar os pesos w em um pequeno intervalo W. Dependendo do intervalo W definido, a técnica que utiliza essa função pode sofrer convergência lenta, quando o intervalo for muito alto, ou saturação do gradiente, quando o intervalo for muito pequeno (WENG, 2017).

Embora resolva alguns problemas enfrentados por outras funções, como o *Mode Collapse*, a função *Wasserstein* apresenta problemas de instabilidade pelo forma como força uma restrição 1-Lipschitz a sua função. É possível que em alguns casos o modelo produza imagens ruins ou não convirja. A WGAN-GP (GULRAJANI et al., 2017), que utiliza a função de custo *Wasserstein*, substitui o W por uma penalização no gradiente, evitando esses problemas.

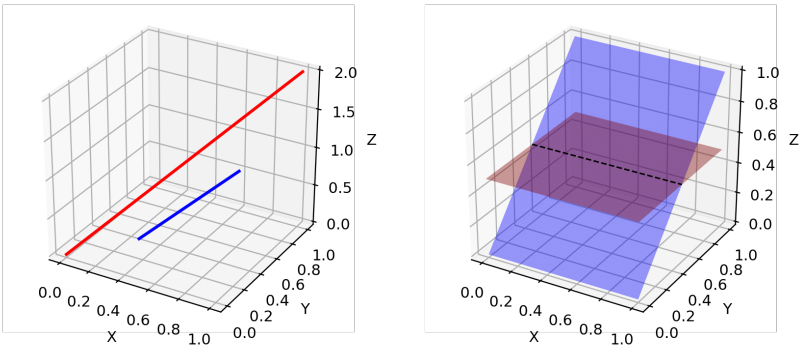


Figura 6 – (Esquerda) Distribuições sem sobreposição. (Direita) Distribuições sobrepostas. Funções de custo tradicionais requerem que as distribuições tenham sobreposições, caso contrário, essas funções exibem comportamentos aleatórios que acabam por desestabilizar o treinamento. Fonte: (WENG, 2017)

4.3 PESQUISAS ATUAIS

GANs são modelos recentes, existindo inúmeras pequisas em aberto. Nesta seção, são apresentados os principais problemas em estudo.

4.3.1 Falta de convergência

O treinamento de uma GAN envolve encontrar o equilíbrio de Nash em um jogo não cooperativo entre dois jogadores. O problema está na convergência do modelo ao realizar o treinamento utilizando a técnica do gradiente descendente. Em certos jogos, ou o equilíbrio é obtido ou cada movimento de um jogador pode desfazer todo o progresso que o outro fez para alcançar o equilíbrio. Isso acontece porque os custos de cada jogador são atualizados desconsiderando o outro e, como consequência, pontos de mínimo local de um são considerados de máximo pelo outro, fazendo com que o gradiente fique rotacionando entre os dois, deixando o treinamento instável. Esse é um problema com jogos em geral, não apenas das GANs. O principal problema nesse tema é o *Mode collapse*.

O *Mode collapse*, ou *Helvetica scenario*, ocorre quando o gerador descobre uma falha no discriminador e passa a explorá-la, gerando amostras com baixa ou nenhuma diversidade. A gravidade desse problema pode ser total, todas as amotras são iguais, ou parcial, algumas amostras são iguais, como o exemplo exibido na Figura 7.

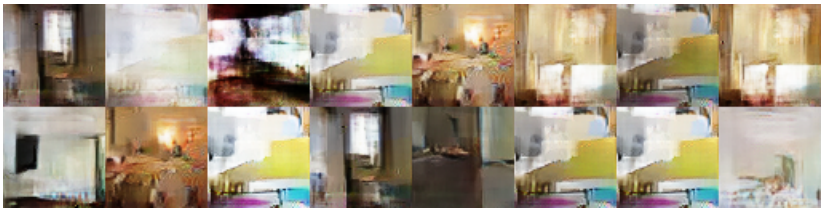


Figura 7 – *Mode collapse* parcial de um conjunto de amostras sobre uma base de dados de camas. Fonte: (Arjovsky; Chintala; Bottou, 2017)

4.3.2 Métricas de avaliação

As funções de custo apresentadas neste trabalho são utilizadas para medir a pontuação de dois jogadores, o gerador e o discriminador, em um jogo. Entretanto, essas funções não avaliam a qualidade ou a diversidade das amostras geradas, que atualmente são avaliadas de forma manual. Algumas técnicas visam tratar esses problemas:

- ***Inception Score*** (IS) (SALIMANS et al., 2016): Avalia a qualidade e a diversidade das amostras utilizando entropia.

  - Qualidade das amostras: Entropia alta, qualidade ruim. Entropia baixa, qualidade boa.
  - Diversidade: Entropia alta, diversidade alta. Entropia baixa, diversidade baixa (*Mode collapse*).

  Esses dois critérios são combinados na Equacao 4.7. Onde D_{KL} representa a divergência de Kullback-Leibler e p(y|x) é a probabilidade de y dado x.

$$IS(G) = exp(\mathbb{E}_{x \sim p_g} D_{KL}(p(y|x)||p(y))) \tag{4.7}$$

  Entropia, nesse contexto, está relacionada à aleatoriedade. Por exemplo, considere uma GAN com uma base de treinamento de imagens de gatos. Ao avaliar, manualmente, um conjunto de imagens geradas a partir dessa base, um avaliador irá escolher uma imagem qualquer e verificar se essa imagem possui as características necessárias para ser considerada uma imagem real de um gato. Em um segundo teste, ele avaliará se há uma quantidade suficiente de imagens de diferentes gatos, e não apenas um grande conjunto com várias imagens de um mesmo gato. Traduzindo esse processo para a técnica do IS, ela irá avaliar se, dada uma imagem qualquer, essa imagem é altamente predizível, ou seja, é possível afirmar que se trata de uma imagem de um gato. Se sim, a entropia dessa imagem será baixa. Em seguida, ela avaliará o conjunto analisando se ele possui diversidade suficiente. Se sim, esse conjunto será impredizível e sua entropia será alta, ou seja, não há um padrão que o gerador segue para gerar novas classes de imagens, ou melhor, ele gera novas classes de imagens seguindo uma distribuição uniforme.

- ***Fréchet Inception Distance*** (FID) (HEUSEL et al., 2017): O FID extrai características das amostras durante o processo de treinamento e modela uma distribuição de dados a partir dessas. Em seguida, o FID entre as amostras geradas g e as amostras reais x é calculado através da Equação 4.8.

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\sum_x + \sum_g - 2(\sum_x + \sum_g)^{\frac{1}{2}}) \quad (4.8)$$

  Sendo $(\mu_x, \sum_x)$ e $(\mu_g, \sum_g)$ a média e covariância das amostras das distribuições reais e geradas, respectivamente, $Tr(A)$ é o traço de uma matriz A e $||x||_2^2$ o produto de um vetor de dimensão 2 com ele mesmo.

  Como a técnica avalia a distância entre duas distribuições, quanto menor o FID, mais semelhantes as distribuições são e, por consequência, a qualidade e a diversidade das amostras são altas.

  Ambas as técnicas são úteis para verificar se ocorreu *Mode collapse*. Porém, o FID é mais robusto ao ruído do que o IS (HEUSEL et al., 2017), conseguindo produzir resultados mais consistentes em relação à diversidade das amostras.

- ***Precision*, *Recall* e *F1 Score***

  São técnicas amplamente conhecidas e utilizadas para avaliar a qualidade de modelos discriminadores. Lučić et al. (2018) utiliza essas técnicas para comparar amostras geradas de diferentes modelos de GANs. *Precision* e *Recall* são equivalentes à qualidade e diversidade de amostras, respectivamente. Um *precision* alto significa que as amostras geradas são semelhantes às imagens reais, enquanto um *recall* alto diz que o modelo gerado consegue gerar qualquer amostra presente no conjunto de dados real. *F1 Score* é a média harmônica do *precision* e *recall*. De acordo com Lučić et al. (2018), essas técnicas são complementares às técnicas IS e FID, pois as GANs possuem algum grau de *overfitting* que pode ser detectado por essas.

No geral, essas métricas permitem avaliar diferentes tipos de configurações de um mesmo modelo de GAN. Por exemplo, é possível avaliar diferentes funções de custo utilizando essas métricas, ou avaliar o impacto que certas configurações (*e.g.*, algoritmos de otimização, tamanho dos *minibatchs*, taxa de aprendizado) têm sobre a qualidade e a diversidade das amostras geradas.

4.3.3 Problemas referentes à imagens

- Contagem

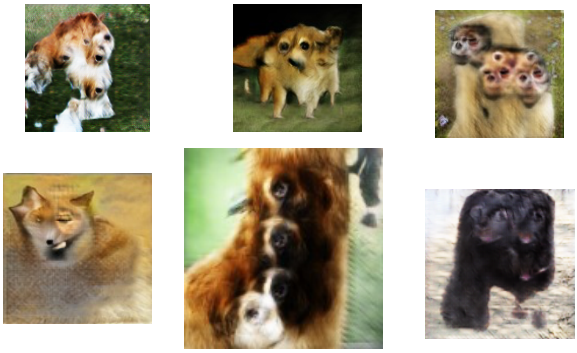


Figura 8 – Imagens com vários elementos semelhantes apresentam problemas, normalmente gerando imagens que exageram na quantidade de partes desse elemento. Fonte: (GOODFELLOW, 2017)

- Perspectiva

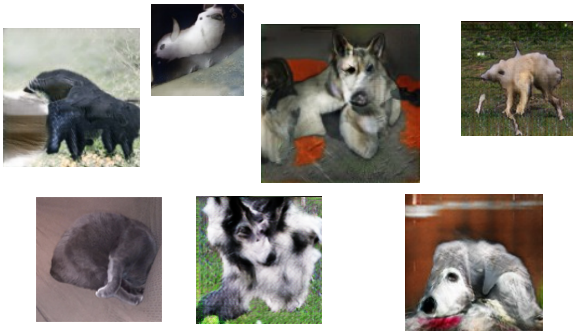


Figura 9 – GANs não conseguem gerar imagens que passam a sensação de possuir espessura ou profundidade, resultando em imagens planas. Goodfellow (2017) deixa como desafio ao leitor descobrir qual dessas imagens é real. Fonte: (GOODFELLOW, 2017)

- Estrutura Global

Figura 10 – GANs não conseguem aprender o conceito de estrutura global, gerando, por exemplo, imagens de cachorros com uma orelha ou três patas. Fonte: (GOODFELLOW, 2017)

4.3.4 Outros problemas

Além dos principais problemas que afetam diretamente o avanço de novos e melhores modelos de GANs, há também outras questões e problemas de menor grau que merecem destaque, são eles:

- **Saídas discretas**: Como o único requisito imposto ao gerador é que ele seja diferenciável, GANs não conseguem produzir dados discretos (GOODFELLOW, 2017). A construção de novos modelos que superam essa limitação podem ser úteis para a área de processamento de linguagens naturais.

- **Tempo de treinamento**: O tempo de treinamento das GANs é elevado, facilmente levando algumas horas em uma GPU tradicional. Karras et al. (2017b) produziu imagens de 1024×1024 píxeis, até agora as maiores imagens conhecidas geradas por GANs, levando em torno de dois dias com oito GPUs, as quais possuem desempenho muito superior em relação às tradicionais por possuirem otimizações específicas para *Deep Learning*, ou duas semanas com uma.

- **Uso do código latente**: O código latente z guarda propriedades úteis de uma imagem x, o problema está na dificuldade de utilizar essas informações. Chen et al. (2016) propõe a criação de uma relação mútua, baseado na teoria da informação, entre z e x. Assim, as informações semânticas da imagem são preservadas no processo de geração. Por exemplo, considere uma base de dados de imagens de dígitos, onde um conjunto de variáveis latentes está relacionado aos aspectos semânticos das imagens. Agora, em vez de repassar ao gerador uma variável latente desestruturada z, é passado uma variável latente estruturada, contendo informações semânticas úteis (o tipo, rotação e/ou largura dos dígitos). Além disso, o custo computacional adicional é insignificante.

- **Relação com outras áreas**: Algumas pesquisas visam adaptar às GANs para outras áreas tradicionais da IA. Salimans et al. (2016) desenvolveu um modelo estado da arte para a área de aprendizado supervisionado, o *feature matching* GANs, enquanto Ho e Ermon (2016) relaciona GANs com aprendizado reforçado.

4.4 VARIAÇÕES

Conforme aumentou em importância, descobriu-se inúmeras aplicações que poderiam se beneficiar com a aplicação desse modelo. A Tabela 2 exibe os principais modelos de GANs e seu campo de aplicação. Hindupur (2018)

Nome	Aplicação
PSGAN	Síntese de texturas
SRGAN	Super resolução de imagens
Context encoder	Reconstrução de imagens
GAWWN	Mapeamento texto para imagem
iGAN	Edição de imagens
VGAN	Geração de vídeo
3DGAN	Geração de objetos 3D
MidiNet	Geração de música
MaliGAN.	Distribuições discretas
GAN Q-learning	Aprendizado reforçado

Tabela 2 – Diversas adaptações das GANs e suas aplicações.

4.5 DCGAN

Deep Convolutional Generative Adversarial Networks (DCGANs) (RADFORD; METZ; CHINTALA, 2015) foi a primeira e, até o momento, a melhor proposta para melhorar aspectos críticos apresentados na versão original. Ela combina as tradicionais redes convolucionais (CNNs) com as GANs, visando construir uma rede de aprendizado não supervisionado funcional, estável e com ótimos resultados. Além disso, essa técnica possui propriedades de aritmética de vetores, definido na Figura 12, permitindo que as propriedades das imagens sejam alteradas de forma simples.

4.5.1 Arquitetura

O principal fator que proporcionou estabilidade para o modelo consistiu na adição e modificação de três mudanças sobre a arquitetura tradicional de uma CNN. As modificações listadas abaixo são do artigo original, nas próximas seções são apresentadas outras modificações que alteraram a implementação original da DCGAN.

- Substituição de *deterministic spatial pooling functions* (*e.g., max-pooling*) por *strided convolutions* no discriminador e *fractionally-strided convolution* no gerador. A Figura 11 exibe a arquitetura do gerador utilizando strides.

- Eliminação de camadas totalmente conectadas, substituindo-os por *global average pooling*. Com isso, a estabilidade do modelo aumenta, mas a velocidade de convergência é penalizada.

- Uso de *batch normalization* (exceto na camada de saída do gerador e entrada do discriminador), apenas o uso dessa técnica já contribui para o aumento da estabilidade do modelo. Isso ocorre pois, se não normalizadas, as entradas podem apresentam certos problemas. Em relação às modificações realizadas, essa é a mais crítica, apresentando o maior impacto sobre a qualidade das amostras resultantes.

- Uso das funções de ativação ReLU para o gerador e *Leaky* ReLU para o discriminador. Utilizando essas funções, o modelo consegue aprender mais rápido.

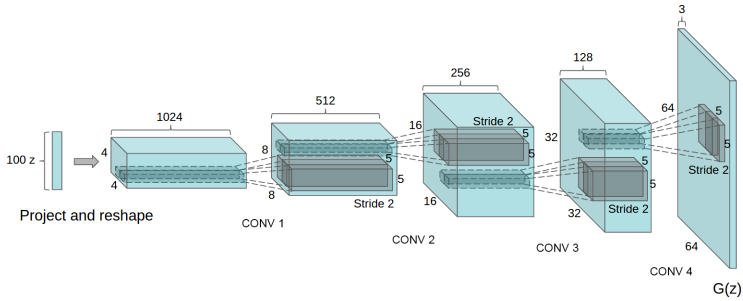


Figura 11 – Arquitetura do gerador utilizado na DCGAN. O código latente Z, representado como um distribuição de dimensão 100 na figura, passa por diversas transformações até o resultado final. Fonte: (RADFORD; METZ; CHINTALA, 2015).

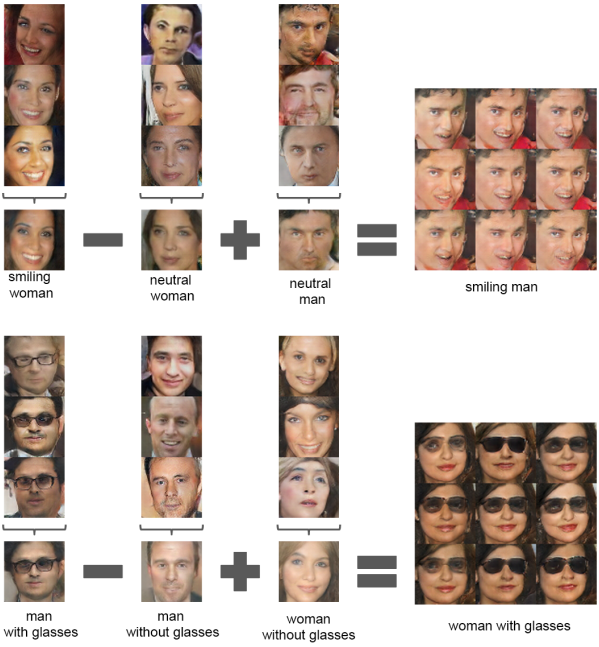


Figura 12 – Aritmética de vetores aplicado à amostras geradas pela DCGAN. GANs são capazes de diferenciar os elementos existentes nas imagens. Fonte: (RADFORD; METZ; CHINTALA, 2015)

4.6 DICAS DE TREINAMENTO

O treinamento das GANs é complexo e desafiador, mudanças mínimas nos parâmetros facilmente afetam todo o modelo. Instabilidade e falta de convergência são os problemas mais comuns. Nesta seção, são apresentadas dicas em relação a alguns parâmetros que ajudam a evitar problemas comuns de treinamento. As cinco primeiras técnicas foram propostas por Salimans et al. (2016) com o objetivo de acelerar a convergência, enquanto as técnicas seis e sete foram propostas por Arjovsky e Bottou (2017) com a intenção de resolver o problema de distribuições disjuntas.

1. ***Feature Matching***: Propõe um novo objetivo para o gerador. Seu propósito agora é gerar dados que correspondem com estatísticas extraídas, pelo discriminador, dos dados reais. A nova equação pode ser definida como: $||\mathbb{E}_{x \sim p_{data}} f(x) - \mathbb{E}_{z \sim p_z(z)} f(G(z))||_2^2$. Sendo $f(x)$ a computação de estatísticas de características importantes para o treinamento, como média ou desvio padrão.

2. ***Minibatch Discrimination***: Um dos problemas que causam o *Mode collapse* é o fato do discriminador não possuir um mecanismo que avise ou obrigue o gerador a gerar amostras com maior diversidade, isso ocorre pois o discriminador analisa cada amostra do conjunto de treinamento separadamente. Essa técnica produz um novo valor, que será utilizado pelo modelo, que relaciona a semelhança de cada amostra em relação às outras amostras do seu *minibatch*. Dessa forma, o discriminador possui um valor que pode ser usado para forçar uma diversidade nas amostras resultantes do gerador.

3. ***Historical Averaging***: Inclui um termo novo ao custo dos jogadores: $||\theta - \frac{1}{t}\sum_{i=1}^{t}\theta[i]||^2$, onde $\theta[i]$ é o histórico de valores do parâmetro θ no tempo i. A ideia é penalizar a velocidade de treinamento se o θ é alterado radicalmente no tempo.

4. ***One-sided Label Smoothing***: Altera o intervalo de saída do discriminador. Em vez de retornar valores no intervalo [0, 1], essa técnica sugere utilizar intervalos suaves, como [0 ; 0,9]. Warde-Farley e Goodfellow (2016) demonstra que essa técnica reduz a vulnerabilidade de redes adversárias.

5. ***Virtual Batch Normalization*** (VBN): Normaliza as amostras utilizando um *batch* de referência, que é fixo e definido no início do treinamento. VBNs são computacionalmente caros, por isso são apenas utilizados no gerador.

6. **Adicionar ruído**: As distribuições p_g e p_{data} podem ser disjuntas em determinados espaços dimensionais, gerando problemas como a saturação do gradiente. Para forçar uma sobreposição dessas distribuições, é proposto a adição de ruído às entradas do discriminador.

7. **Utilizar outras métricas de similaridade de distribuições**: Certas métricas, como a divergência JS, não fornecem resultados significativos quando duas distribuições são disjuntas. Uma alternativa é alterar a métrica utilizada. A métrica *Wasserstein*, por exemplo, gera bons resultados com distribuições disjuntas.

8. **Usar um *Z* esférico**: Como visto na Figura 5, o valor de Z é amostrado de uma distribuição de probabilidade conhecida. No artigo original, essa distribuição era uma uniforme, mas White (2016) recomenda utilizar uma distribuição gaussiana.

4.7 COMPARAÇÃO DOS MODELOS

Apresentadas as GANs, é interessante compará-las com os modelos vistos no capítulo anterior para entender melhor o comportamento e as características de cada um. As conclusões aqui apresentadas se basearam na análise comparativa de (GOODFELLOW, 2017).

- **Densidade**: Ao contrário das FVBNs, VAEs e *Boltzmann Machines*, as GANs não necessitam que a distribuição de um conjunto seja dado explicitamente. O modelo é tratável e construído iterativamente.

- **Consistência assintótica**: GANs são o único modelo que é conhecido por ser assintoticamente consistente.

- **Amostras**: Até o momento, nenhum outro modelo produziu amostras com a qualidade das produzidas pelas GANs.

- **Cadeias de Markov**: Modelos que utilizam cadeias de Markov costumam ser engessados, possuindo várias restrições quanto a função geradora ou a dimensão do vetor latente z. GANs, por outro lado, não possuem essas restrições.

- **Geração de amostras**: GANs podem gerar uma amostra instantaneamente, ao contrário das FVBNs que precisam gerar um pixel por vez, ou em relação a alguns modelos que precisam repetidamente aplicar o operador da cadeia de Markov para gerar novas amostras.
- **Equilíbrio**: GANs costumam ser instáveis, pois é necessário encontrar o equilíbrio de Nash ao realizar o treinamento, o que é uma tarefa difícil.
- **Dados discretos**: GANs têm dificuldade em gerar dados discretos, como textos.

5 EXPERIMENTOS

Na revisão teórica, os principais conceitos foram analisados e, com isso, é possível entender as propostas e modificações existentes, além de propor novos conceitos. Porém, a falta de uma visão prática pode comprometer a assimilação completa da teoria, pois as discussões propostas podem complementar o entendimento sobre o assunto.

As próximas seções visam detalhar e discutir certos aspectos da arquitetura de GANs existentes e estáveis, ou seja, que em algum momento convergem e resultam em amostras de qualidade suficiente[1].

5.1 ALGORITMO

Antes de iniciar os experimentos, é necessário compreender o núcleo de uma GAN e as consequências resultantes de sua arquitetura. O Algoritmo 1 apresenta o processo de treinamento de uma GAN como definido no artigo original, havendo diferentes versões de GANs e melhorias propostas que alteraram esse processo.

Com o algoritmo definido, é interessante notar que:

- Em uma iteração do treinamento, o número de iterações do discriminador pode ser maior que a do gerador, indicando que não há problema em tornar o discriminador mais poderoso que o gerador.

- Implementação de D e G: Não são definidas restrições de implementação para as redes neurais usadas. De fato, várias implementações utilizam diferentes estratégias, como a DCGAN que implementa as redes utilizando convoluções profundas.

- Flexibilidade na utilização de métodos baseados em gradientes. Como comentado no algoritmo, qualquer método pode ser utilizado, inclusive, pode-se definir diferentes métodos para o gerador e o discriminador, como será visto adiante.

[1] Seguindo os critérios definidos na Seção 4.3.2, ou seja, as amostras possuem certas características para serem classificadas como pertencentes a base de dados real e cada lote de amostras deve possuir diversidade.

Algoritmo 1 Treinamento de redes geradoras adversárias com gradientes descendentes estocásticos em lotes. O número de etapas a serem aplicadas ao discriminador, k, é um hiperparâmetro. Nós usamos $k = 1$, a opção mais barata, em nossos experimentos.

para número de iterações de treinamento **faça**

para k passos **faça**

- Amostre um lote de m amostras de ruído $\{z^{(1)}, \ldots, z^{(m)}\}$ da distribuição $p_g(z)$.
- Amostre lotes de m exemplos $\{x^{(1)}, \ldots, x^{(m)}\}$ da distribuição geradora de dados $p_{data}(x)$.
- Atualize o gerador aumentando seu gradiente estocástico:

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

fim para

- Amostre lotes de m amostras de ruído $\{z^{(1)}, \ldots, z^{(m)}\}$ da distribuição $p_g(z)$.
- Atualize o gerador diminuindo seu gradiente estocástico:

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

fim para

As atualizações baseadas em gradiente podem usar qualquer regra de aprendizado padrão baseada em gradiente. Nós usamos *momentum* em nossos experimentos.

Fonte: Traduzido de (GOODFELLOW et al., 2014)

5.2 ANÁLISE DE ALGORITMO - GAN

Embora a criação de uma GAN simples[2] não demande muitas linhas de código, a construção de modelos estáveis são complexos, pois ainda não há ferramentas que auxiliem na busca do erro. Nessa seção, são analisados as principais partes da arquitetura de uma GAN existente simples, mas estável, os resultados obtidos desse modelo, a implementação e análise do impacto de dicas vistas em seções anteriores.

[2]Uma GAN produz novos dígitos em preto e branco a partir da base MNIST.

5.2.1 Gerador

A implementação do gerador, exibido no Fragmento 5.1, envolve:

- 3 camadas ocultas, com 256, 512 e 1024 neurônios, respectivamente. Utilizando funções de ativação ReLU.
- 1 camada de saída, com 784 saídas (28×28 píxeis), utilizando a função de ativação tanh.

A estrutura geral do gerador, como já visto, não se difere de uma rede neural tradicional, onde a saída da rede será uma amostra produzida pelo gerador. A principal discussão dessa arquitetura se deve a função de ativação da camada de saída.

A função tanh retorna valores no intervalo $[-1, 1]$. Ela é preferível à função sigmoide, tipicamente utilizada na camada de saída, pois está centrada em zero, resultando em uma convergência mais rápida (LECUN et al., 1998). Nesse caso, é obrigatório a normalização das entradas para o intervalo $[-1, 1]$, pois os valores das entradas também devem ser centrados em zero.

```
def generator(x):
    # Inicializadores
    w_init = tf.truncated_normal_initializer(mean=0, stddev=0.02)
    b_init = tf.constant_initializer(0.)

    # Primeira camada oculta
    w0 = tf.get_variable('G_w0', [x.get_shape()[1],256], initializer=w_init)
    b0 = tf.get_variable('G_b0', [256], initializer=b_init)
    h0 = tf.nn.relu(tf.matmul(x, w0) + b0)

    # Segunda camada oculta
    w1 = tf.get_variable('G_w1', [h0.get_shape()[1],512], initializer=w_init)
    b1 = tf.get_variable('G_b1', [512], initializer=b_init)
    h1 = tf.nn.relu(tf.matmul(h0, w1) + b1)

    # Terceira camada oculta
    w2 = tf.get_variable('G_w2', [h1.get_shape()[1],1024], initializer=w_init)
    b2 = tf.get_variable('G_b2', [1024], initializer=b_init)
    h2 = tf.nn.relu(tf.matmul(h1, w2) + b2)

    # Camada de saida
    w3 = tf.get_variable('G_w3', [h2.get_shape()[1],784], initializer=w_init)
    b3 = tf.get_variable('G_b3', [784], initializer=b_init)
    o = tf.nn.tanh(tf.matmul(h2, w3) + b3)

    return o
```

Fragmento 5.1 – Estrutura do gerador.

5.2.2 Discriminador

O discriminador, exibido no Fragmento 5.2, é semelhante ao gerador, diferindo em:

- O número de neurônios das camadas ocultas se invertem, sendo utilizados 1024, 512 e 256 neurônios, respectivamente.
- Uso de *dropout*.
- Uso da função de ativação sigmoide na camada de saída.

A rede possui uma saída pois o seu valor de retorno será um escalar entre 0 e 1, indicando a probabilidade da amostra de entrada pertencer ou não a base de dados real. Como a saída pertence ao intervalo [0, 1], a função sigmoide tem preferência, pois mesmo não convergindo tão rápido, ela se encaixa perfeitamente nesse intervalo, sem a necessidade de realizar um pré-processamento dos dados.

A principal característica é o uso da técnica de *dropout*. Como o discriminador atua diretamente com as amostras, diferente do gerador, há a possibilidade de *overfitting*. Por isso, o *dropout* é uma técnica de regularização que reduz esse problema. A ideia é, aleatoriamente, desligar algumas conexões entre neurônios, criando uma rede esparsa e forçando a propagação de várias características pela rede, em vez de propagar apenas algumas, diminuindo a chance de *overfitting*.

```
def discriminator(x, drop_out):

    # Inicializadores
    w_init = tf.truncated_normal_initializer(mean=0, stddev=0.02)
    b_init = tf.constant_initializer(0.)

    # Primeira camada oculta
    w0 = tf.get_variable('D_w0', [x.get_shape()[1],1024], initializer=w_init)
    b0 = tf.get_variable('D_b0', [1024], initializer=b_init)
    h0 = tf.nn.relu(tf.matmul(x, w0) + b0)
    h0 = tf.nn.dropout(h0, drop_out)

    # Segunda camada oculta
    w1 = tf.get_variable('D_w1', [h0.get_shape()[1],512], initializer=w_init)
    b1 = tf.get_variable('D_b1', [512], initializer=b_init)
    h1 = tf.nn.relu(tf.matmul(h0, w1) + b1)
    h1 = tf.nn.dropout(h1, drop_out)

    # Terceira camada oculta
    w2 = tf.get_variable('D_w2', [h1.get_shape()[1],256], initializer=w_init)
    b2 = tf.get_variable('D_b2', [256], initializer=b_init)
    h2 = tf.nn.relu(tf.matmul(h1, w2) + b2)
    h2 = tf.nn.dropout(h2, drop_out)

    # Camada de saida
    w3 = tf.get_variable('D_w3', [h2.get_shape()[1],1], initializer=w_init)
    b3 = tf.get_variable('D_b3', [1], initializer=b_init)
    o = tf.sigmoid(tf.matmul(h2, w3) + b3)

    return o
```

Fragmento 5.2 – Estrutura do discriminador.

5.2.3 Função de custo

A função de custo utilizada no discriminador é a entropia cruzada e a do gerador é a heurística de não-saturação, como pode ser visto no Fragmento 5.3. Conforme visto na Seção 4.2, essa função obtém os melhores resultados mesmo se o discriminador for perfeito. As Equações 5.1 e 5.2 são reproduzidas abaixo para comparação com a implementação.

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}\ logD(x) - \frac{1}{2}\mathbb{E}_z\ log(1 - D(G(z))) \quad (5.1)$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{z\sim p_{modelo}}\ log(D(G(z))) \quad (5.2)$$

```
eps = 1e-2
D_loss = tf.reduce_mean(-tf.log(D_real + eps) - tf.log(1 - D_fake + eps))
G_loss = tf.reduce_mean(-tf.log(D_fake + eps))
```

Fragmento 5.3 – Funções de custo da GAN em estudo.

5.2.4 Minimização do erro

Como explicado na Seção 4.2, o objetivo de uma função de custo é quantificar a qualidade de um modelo em relação a uma tarefa que se propõe a resolver. Para o discriminador, sua tarefa é diferenciar amostras de distribuições diferentes, enquanto a tarefa do gerador é produzir uma distribuição de probabilidade que se assemelha à distribuição alvo. Quanto menor o valor retornado pela função, maior a qualidade do modelo.

Como as funções de custo são, por definição, funções, é possível aplicar uma técnica de otimização que procura o valor mínimo de uma função, essa técnica é conhecida como gradiente descendente. Para o código estudado, conforme Fragmento 5.4, o algoritmo que aplica essa técnica é o *Adam Optimizer*, possuindo os parâmetros:

- D_vars/G_vars - São os pesos e vieses da rede cuja função de custo será minimizada.
- lr - Taxa de aprendizado. Determina a velocidade de mudança das variáveis acima. Ao minimizar a função de custo, essas variáveis são atualizadas para se ajustarem ao aprendizado da rede.

- D_loss/G_loss - Valores atuais da função de custo a ser minimizada.

```
D_optim = tf.train.AdamOptimizer(lr).minimize(D_loss, var_list=D_vars)
G_optim = tf.train.AdamOptimizer(lr).minimize(G_loss, var_list=G_vars)
```

Fragmento 5.4 – Técnicas de grandiente descedente utilizadas pela GAN em estudo. Tanto o discriminador quanto o gerador utilizam *Adam Optimizer*.

5.2.5 Análise dos resultados

A seção anterior detalhou questões importantes de implementação. Com a interpretação e discussão dos resultados obtidos, essa seção pretende completar o processo de análise necessário para compreender e implementar todos os aspectos teóricos e práticos de uma GAN.

5.2.5.1 Especificações e tempo de treinamento

O modelo foi executado em um computador equipado com um processador i3-3250 3.5 GHz, 8 GB de RAM. O código foi escrito em Python, utilizando a biblioteca Tensorflow para a construção do modelo e processamentos complexos, além da biblioteca Numpy. Os códigos da GAN e DCGAN foram adaptados de Kang (2017). Utilizando a base de dados MNIST contendo 55.000 imagens de 28×28, o tempo de processamento total do treinamento para 100 épocas foi de 6474,56 segundos, em torno de 1 hora e 47 minutos, com uma média de 62,29 segundos por época.

5.2.5.2 Histograma

O histograma da Figura 13 foi obtido após a execução do treinamento. Inicialmente, ambas as redes trabalham com valores aleatórios, explicando o alto custo do gerador. Como ele apresentou um valor muito ruim e, por consequência, amostras ruins, mesmo trabalhando com valores aleatórios o discriminador ainda consegue facilmente diferenciar as entradas. Depois de algumas épocas, o gerador obtém valores melhores, ou seja, a otimização minimizou o erro, o que melhorou o modelo, resultando em amostras superiores em relação as anteriores.

Enquanto isso, o discriminador também melhorava, mas como a distribuição p_{model} se tornou mais semelhante a p_{data}, se tornou mais difícil diferenciar as amostras. Depois da época 80, o treinamento estabilizou, nenhuma rede melhorou, indicando que não há uma alteração de estratégia ou, em outras palavras, alcançaram o equilíbrio de Nash.

O papel do histograma vai além da análise pós-treinamento. Como o tempo de processamento é alto, é importante notar as falhas do modelo na fase inicial do processo. De acordo com Chintala (2016), se o custo do discriminador vai a zero, a norma do gradiente está acima de 100 ou o custo do gerador cai de forma constante, o processo falhou.

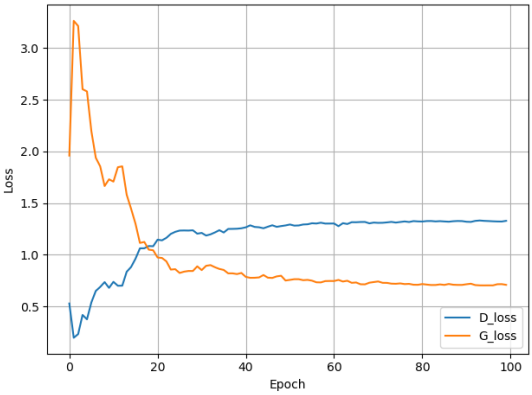
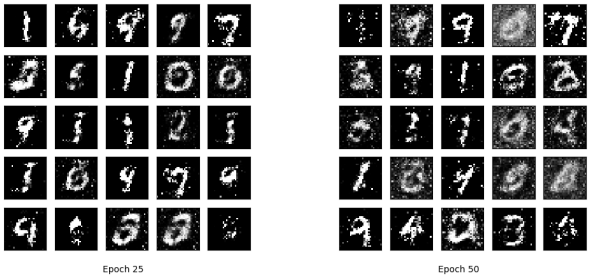


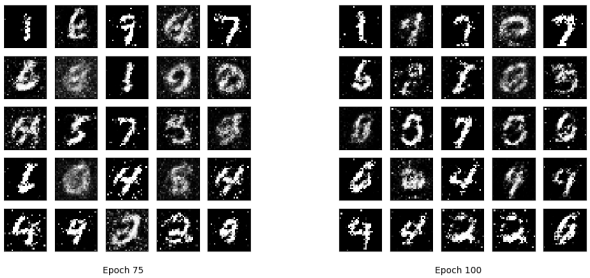
Figura 13 – Histograma das funções de custo do discriminador e gerador ao longo de 100 épocas. Fonte: Produzida pelo autor.

5.2.5.3 Amostras obtidas

O processo de avaliação das amostras, como visto na Seção 4.3.2, consiste na avaliação de dois critérios: qualidade e diversidade. A qualidade das imagens da última época, embora seja possível reconhecer os dígitos, são bem inferiores em relação a base original, exibido na Figura 15. Isso ocorre pois o modelo criado é muito limitado em relação à modelos mais robustos, a próxima seção irá implementar algumas dicas teóricas e empíricas com o objetivo de melhorar o resultado final. Em se tratando de um modelo limitado, a diversidade apresentada é alta, sendo possível visualizar os dígitos 0, 1, 4, 6, 7 e 9, o que perfaz 60% do total de dígitos existentes.



(a) Amostra após 25 épocas (b) Amostra após 50 épocas



(c) Amostra após 75 épocas (d) Amostra após 100 épocas

Figura 14 – Amostras obtidas do treinamento após algumas épocas. Fonte: Produzida pela autor.

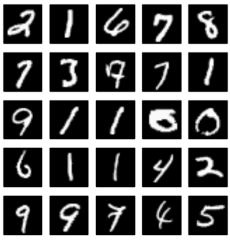


Figura 15 – Amostra da base de dados MNIST. Fonte: (KANG, 2017)

5.2.6 Dicas de implementação

Conforme visto na seção anterior, por causa de sua arquitetura limitada, a implementação gera poucas amostras semelhantes às amostras da base de dados real. Nessa seção, algumas dicas simples, com suporte teórico e empírico, serão implementadas, validando o uso dessas técnicas e mostrando o impacto que pequenas alterações podem causar. Por fim, amostras retiradas do treinamento de uma DCGAN são exibidos para comparação.

5.2.6.1 Técnicas utilizadas

Das dicas de implementação e acompanhamento do treinamento presentes em (CHINTALA, 2016), 5 são modificações que resultaram em amostras com melhor qualidade:

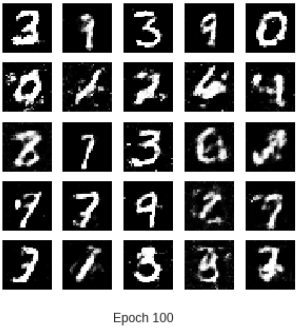
- Uso da função de ativação tanh e normalização das entradas: Como já explicado, funções tanh possuem gradientes mais fortes, convergindo mais rápido. Além disso, para utilizar essa função é necessário normalizar as entradas.
- Uso do z esférico: Amostrar z a partir de uma distribuição gaussiana.
- Evitar o uso de gradientes esparsos: A implementação atual utiliza ReLU, uma função que produz gradientes esparsos, atrapalhando a estabilidade do treinamento. Essa função será trocada, no discriminador e gerador, por LeakyRelU.
- Uso do otimizador ADAM: A dica recomendava o uso do otimizador SGD no discriminador e ADAM no gerador, mas os resultados não foram satisfatórios. Optou-se por usar ADAM em ambas as redes.
- Uso de *dropout* no gerador: A implementação utiliza dropout apenas no discriminador, o gerador também utilizará essa técnica.

5.2.6.2 Análise dos resultados

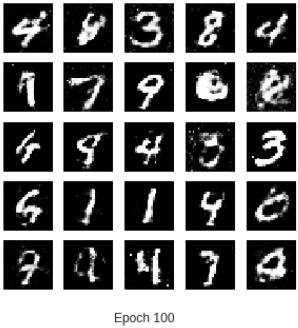
As amostras resultantes do treinamento utilizando a combinação de técnicas é exibida na Figura 16. O salto na qualidade das amostras é visível, a maioria das imagens é mais limpa (possui menos ruído) do que as amostras da implementação antiga. A diversidade também se manteve. É interessante notar a dificuldade do modelo em gerar certos dígitos, como o número 2. Em ambas as implementações, ele não é visivelmente gerado, em alguns casos é possível ver apenas um objeto disforme.

O histograma do treinamento é exibido na Figura 18. Seus valores são um pouco mais altos do que os valores apresentados no histograma da implementação sem as dicas, o que, a princípio, parece contra-intuitivo, pois é ela que apresenta as melhores amostras. No entanto, é pertinente relembrar que o treinamento dessa rede é equivalente a um jogo. Se o discriminador se comportar mal, o gerador não precisará se esforçar para enganá-lo. Assim, a função de custo do gerador retornará um valor baixo, mas as amostras serão de baixa qualidade. Por outro lado, se o discriminador for poderoso, o gerador precisará se esforçar para enganá-lo, a função de custo retornará um valor um pouco mais alto, mas a qualidade das amostras será maior.

Finalmente, vale ressaltar a limitação dessa implementação. A arquitetura do gerador e discriminador são projetadas como redes neurais tradicionais, ou seja, a rede recebe como entrada amostras pré-processadas. Com isso, perde-se muita informação, pois existem vários tipos de pré-processamento focados em diferentes características da imagem. A aplicação de redes convolucionais resolve esses problemas. Além de exigir um mínimo de pré-processamento, elas são fortemente inspiradas nos processos biológicos (MATSUGU et al., 2003). Ou seja, ela "aprende" os filtros necessários, minimizando a perda de informações. A DCGAN é implementada utilizando essas redes e o resultado do treinamento pode ser visto na Figura 17. Visualmente, não há dúvidas quanto à qualidade. Entretanto, o custo da qualidade é alto. O custo computacional e o tempo de processamento é muito elevado em relação à implementação estudada, levando em torno de 3 horas para executar 20 épocas em uma GPU Tesla K80 (com processamento limitado), enquanto o código estudado levou em torno de 10 minutos para executar 100 épocas, nessa mesma configuração.

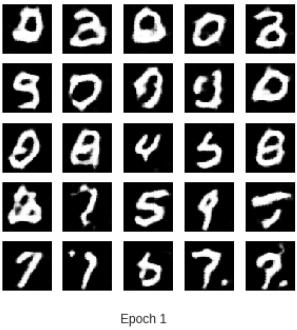


(a) Amostra após 100 épocas

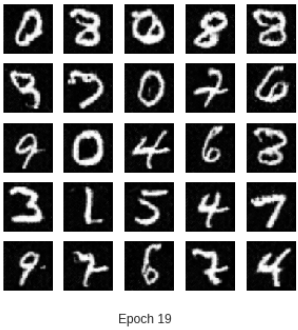


(b) Outras amostras

Figura 16 – Amostras obtidas do treinamento com as dicas após algumas épocas. Fonte: Produzida pelo autor.



(a) Amostra após 1 época



(b) Amostra após 19 épocas

Figura 17 – Amostras obtidas do treinamento de uma DCGAN após algumas épocas. Fonte: Produzida pelo autor.

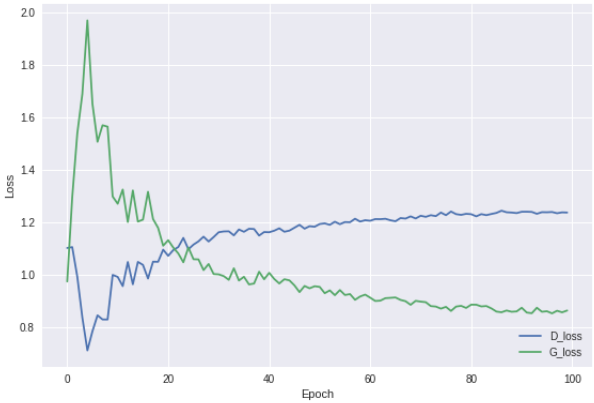


Figura 18 – Histograma do treinamento com dicas. Fonte: Produzida pelo autor.

6 PROVA DE CONCEITO - SRGAN

Elucidado os aspectos teóricos e práticos sobre as GANs, é necessário mostrar uma aplicação real que resolva um problema para o qual hoje, ou não exista solução ou a aplicação das GANs produza resultados melhores em relação às soluções existentes. Esta seção apresenta a *Super Resolution* GAN (SRGAN).

6.1 SRGAN

A SRGAN (LEDIG et al., 2016) visa produzir imagens de alta resolução. Após treinado, o modelo aceita como entrada uma imagem de baixa resolução e produz como saída uma imagem de super resolução que, no mínimo, preserva a qualidade original da imagem.

A arquitetura lógica segue o mesmo princípio das GANs tradicionais. O gerador recebe como entrada uma imagem de baixa resolução (originalmente de alta resolução, sofrendo um processo de subamostragem) e, como saída, produz uma imagem de super resolução. O discriminador diferencia imagens de alta resolução das de super resolução (produzidas pelo gerador).

6.2 MATERIAIS E MÉTODOS

O algoritmo que implementa a SRGAN foi obtido em (DONG et al., 2017)[1], foi implementada pelos desenvolvedores da biblioteca TensorLayer, como uma implementação de exemplo que a utiliza. Ele sofreu apenas uma adaptação para que consiga processar imagens em lote. A aplicação utilizou o modelo pré-treinado existente.

Foram utilizadas 106 imagens de teste coloridas e em preto e branco, retiradas de Agustsson e Timofte (2017), Timofte et al. (2017) e Gonzalez Woods (2017), com tamanho médio de 500×500 e formato PNG. Além disso, algumas imagens foram, aleatoriamente, retiradas da internet para avaliar a aplicação sobre diferentes formatos de imagem.

As configurações do computador utilizado para rodar a aplicação são os mesmo definidos na Seção 5.2.5.1.

[1] Código disponível em: https://github.com/tensorlayer/srgan

6.3 RESULTADOS OBTIDOS

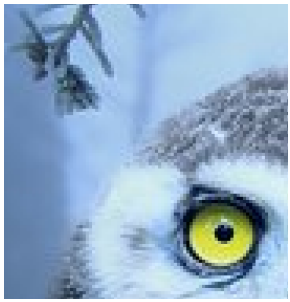
Dos resultados obtidos [2]:

- Para obter bons resultados, é necessário que o formato da imagem possua compressão sem perda de dados. A Figura 19 foi retirada da internet e está no formato JPG. Mesmo apresentando bons resultados, ao aproximar a imagem é possível verificar que a qualidade do resultado é inferior as imagens sem perda de dados.
- Todas as 106 imagens apresentaram excelentes resultados, com um tempo de execução em torno de 60 segundos por imagem. Algumas amostras são exibidas nas Figuras 20 e 21.
- O modelo pré-treinado apresenta ótimos resultados para imagens em preto e branco, não havendo necessidade de misturá-los ou realizar treinamentos separados para cada tipo de imagem.
- O tamanho de imagem mínimo necessário para manter a qualidade da imagem original é de 500 píxeis. Imagens maiores que esse mínimo não foram testadas porque exigiam recursos computacionais maiores do que o existente.

[2] Os resultados podem ser acessados em:
https://drive.google.com/open?id=1CeIouWf6zzbVnmm4f1CaXf$_oYV0kO0aJ$

(a)

(b)

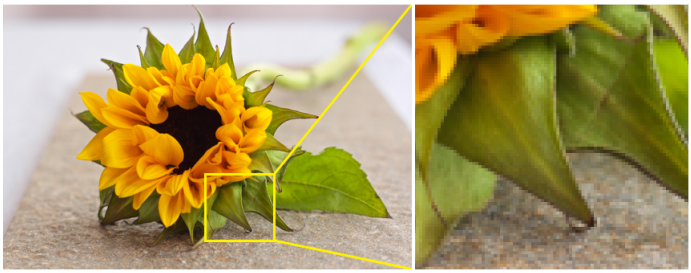


(c)

Figura 19 – (a) Imagem original. (b) Resultado 4×. (c) Imagem original ampliada. Note o ruído produzido em torno das bordas de cada objeto (folha e coruja). Fonte: (a) (MOONZIGG, 2019).

Figura 20 – Resultados obtidos. À esquerda, o resultado da SRGAN (Ampliada 4×). À direita, a imagem original. Fonte: Produzida pelo autor.

(a)



(b)

(c)

(d)

Figura 21 – Resultados obtidos. (a) e (c) são as saídas da SRGAN. (b) e (d) são as imagens originais. (c) e (d) foram recortadas e ampliadas. Fonte: Produzida pelo autor.

7 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foram discutidos os principais aspectos teóricos e práticos das GANs. Assim, a principal contribuição deste trabalho, expor de forma didática esses aspectos, foi alcançada. Com isso, é possível entender e contribuir com pesquisas que buscam resolver os problemas em aberto dessa técnica, como o problema da falta de estabilidade, ou com aplicações que trabalham com GANs e as adaptam para o contexto do problema, por exemplo, a SRGAN. Embora seus conceitos teóricos sejam simples, a complexidade computacional dessa técnica torna o seu treinamento inviável fora de ambientes com alto poder computacional, dificultando a difusão dessa técnica, tanto como solução para empresas privadas, como para a realização de pesquisas.

Os experimentos desenvolvidos permitiram complementar o aprendizado, suprindo uma deficiência de trabalhos relacionados ao ensino de GANs, como, por exemplo, discussões envolvendo o algoritmo ou o papel do histograma no treinamento. Por fim, aplicando a SRGAN na área de super resolução de imagens provou a eficácia das GANs na resolução de problemas atuais, provendo excelentes resultados, desde que as imagens não fossem comprimidas.

Pode-se elencar como trabalhos futuros:

- Estudo sobre novas métricas de avaliação: Novas métricas estão sendo desenvolvidas, como o *Kernel Inception Distance* (BIńKOWSKI et al., 2018), e uma comparação entre elas, considerando vantagens e desvantagens, é interessante. Além disso, é possível avaliar métricas utilizadas em outras técnicas e tentar adaptá-las as GANs.

- Estudo sobre diferentes componentes das GANs: Cada componente influencia o comportamento final do treinamento de uma certa forma, é importante analisar como essa influência acontece e se há componentes com influência sobre outros.

- Adaptar e/ou combinar diferentes GANs: A construção de um modelo estável de GAN é difícil e demanda muito tempo. Assim, a adaptação de modelos existentes pode trazer resultados significativos. Por exemplo, como visto, a WGAN-GP é uma WGAN com penalidade no gradiente, essa ideia pode ser adaptada à DCGAN.

REFERÊNCIAS

ACKLEY, D. H.; HINTON, G. E.; SEJNOWSKI, T. J. A learning algorithm for boltzmann machines. *Cognitive Science*, v. 9, n. 1, p. 147–169. <https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog0901$_7$>.

AGUSTSSON, E.; TIMOFTE, R. Ntire 2017 challenge on single image super-resolution: Dataset and study. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2017.

Arjovsky, M.; Bottou, L. Towards Principled Methods for Training Generative Adversarial Networks. *arXiv e-prints*, p. arXiv:1701.04862, Jan 2017.

Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein GAN. *arXiv e-prints*, p. arXiv:1701.07875, Jan 2017.

BENGIO, Y.; THIBODEAU-LAUFER, E.; YOSINSKI, J. Deep generative stochastic networks trainable by backprop. *CoRR*, abs/1306.1091, 2013. <http://arxiv.org/abs/1306.1091>.

BIńKOWSKI, M. et al. Demystifying MMD GANs. In: *International Conference on Learning Representations*. [s.n.], 2018. <https://openreview.net/forum?id=r1lUOzWCW>.

CHEN, X. et al. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016. <http://arxiv.org/abs/1606.03657>.

CHINTALA, S. *How to Train a GAN? Tips and tricks to make GANs work*. 2016. Acessado: 2019-03-13. <https://github.com/soumith/ganhacks>.

DONG, H. et al. TensorLayer: A Versatile Library for Efficient Deep Learning Development. *ACM Multimedia*, 2017. <http://tensorlayer.org>.

FAHLMAN, S. E.; HINTON, G. E.; SEJNOWSKI, T. Massively parallel architectures for ai: Netl, thistle, and boltzmann machines. p. 109–113, 01 1983.

FREY, B. J. *Graphical Models for Machine Learning and Digital Communication*. Cambridge, MA, USA: MIT Press, 1998. ISBN 0-262-06202-X.

FREY, B. J.; HINTON, G. E.; DAYAN, P. Does the wake-sleep algorithm produce good density estimators? In: TOURETZKY, D. S.; MOZER, M. C.; HASSELMO, M. E. (Ed.). *Advances in Neural Information Processing Systems 8*. MIT Press, 1996. p. 661–667. <http://papers.nips.cc/paper/1153-does-the-wake-sleep-algorithm-produce-good-density-estimators.pdf>.

GONZALEZ WOODS, E. *Image Databases*. 2017. Acessado: 2019-04-03. <http://www.imageprocessingplace.com/root$_{f}iles_{V}3/image_{d}atabases.htm$>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [s.n.], 2016. Book in preparation for MIT Press. <http://www.deeplearningbook.org>.

GOODFELLOW, I. et al. Generative adversarial networks. v. 3, 06 2014.

GOODFELLOW, I. J. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. <http://arxiv.org/abs/1701.00160>.

GULRAJANI, I. et al. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. <http://arxiv.org/abs/1704.00028>.

HEUSEL, M. et al. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. <http://arxiv.org/abs/1706.08500>.

HINDUPUR, A. *The GAN Zoo*. 2018. Acessado: 2019-03-05. <https://github.com/hindupuravinash/the-gan-zoo>.

HINTON, G. E.; SEJNOWSKI, T. Learning and relearning in boltzmann machines. v. 1, 01 1986.

HINTON, G. E.; SEJNOWSKI, T. J.; ACKLEY, D. H. *Boltzmann Machines: Constraint Satisfaction Networks That Learn*. Pittsburgh, PA, 1984.

HO, J.; ERMON, S. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. <http://arxiv.org/abs/1606.03476>.

KANG, H. *tensorflow-MNIST-GAN-DCGAN*. 2017. Acessado: 2019-03-13. <https://github.com/znxlwm/tensorflow-MNIST-GAN-DCGAN>.

KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. 10 2017.

KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. <http://arxiv.org/abs/1710.10196>.

KINGMA, D. P. Fast gradient-based inference with continuous latent variable models in auxiliary form. *CoRR*, abs/1306.0733, 2013. <http://arxiv.org/abs/1306.0733>.

KINGMA, D. P.; SALIMANS, T.; WELLING, M. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016. <http://arxiv.org/abs/1606.04934>.

LECUN, Y. et al. Efficient backprop. In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998. p. 9–50. ISBN 3-540-65311-2. <http://dl.acm.org/citation.cfm?id=645754.668382>.

LEDIG, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016. <http://arxiv.org/abs/1609.04802>.

LUčIć, M. et al. Are gans created equal? a large-scale study. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [s.n.], 2018. <https://arxiv.org/pdf/1711.10337.pdf>.

MATSUGU, M. et al. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural networks : the official journal of the International Neural Network Society*, v. 16, p. 555–9, 06 2003.

MOONZIGG. *Pixabay*. 2019. Acessado: 2019-04-14. <https://pixabay.com/photos/owl-snow-snow-owl-bird-eyes-3184032/>.

NASH, J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, National Academy of Sciences, v. 36, n. 1, p. 48–49, 1950. ISSN 0027-8424. <http://www.pnas.org/content/36/1/48>.

NIELSEN, M. A. *Neural Networks and Deep Learning*. [s.n.], 2015. <http://neuralnetworksanddeeplearning.com/chap3.html>.

OORD, A. van den; KALCHBRENNER, N.; KAVUKCUOGLU, K. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016. <http://arxiv.org/abs/1601.06759>.

RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. <http://arxiv.org/abs/1511.06434>.

REED, S. et al. Generative adversarial text to image synthesis. In: *Proceedings of The 33rd International Conference on Machine Learning*. [S.l.: s.n.], 2016.

ROWLAND, T. *Manifold*. n.d. http://mathworld.wolfram.com/Manifold.html. Acessado: 2018-08-13.

RUDIN, W. *Real and complex analysis*. [S.l.: s.n.], 1966. 38 p.

SALIMANS, T. et al. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. <http://arxiv.org/abs/1606.03498>.

SMOLENSKY, P. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, C. P. R. (Ed.). Cambridge, MA, USA: MIT Press, 1986. cap. Information Processing in Dynamical Systems: Foundations of Harmony Theory, p. 194–281. ISBN 0-262-68053-X. <http://dl.acm.org/citation.cfm?id=104279.104290>.

TIMOFTE, R. et al. Ntire 2017 challenge on single image super-resolution: Methods and results. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2017.

TULYAKOV, S. et al. Mocogan: Decomposing motion and content for video generation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

WARDE-FARLEY, D.; GOODFELLOW, I. 1 adversarial perturbations of deep neural networks. In: . [S.l.: s.n.], 2016.

WENG, L. *From GAN to WGAN*. 2017. Acessado: 2019-03-06. <https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.htmlkullbackleibler-and-jensenshannon-divergence>.

WHITE, T. Sampling generative networks: Notes on a few effective techniques. *CoRR*, abs/1609.04468, 2016. <http://arxiv.org/abs/1609.04468>.

YANG, L.-C.; CHOU, S.-Y.; YANG, Y.-H. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In: *ISMIR*. [S.l.: s.n.], 2017.

APÊNDICE A – ARTIGO DO TCC

ELSEVIER

Pattern Recognition Letters
journal homepage: www.elsevier.com

Geração de imagens com Redes Geradoras Adversárias

Fabio Moreira[1,**], Mauro Roisenberg[1], Alexandre Gonçalves Silva[1]

RESUMO

A geração de novas amostras é um desafio atual da inteligência artifical. Os modelos geradores tentam há muito tempo resolver esse problema, porém, a complexidade enfrentada é tamanha que essa área foi abandonada. Com o aumento do poder computacional e das técnicas de aprendizagem profunda, essa área floresceu novamente. As Redes Geradoras Adversárias (RAGs) representam o estado da arte dos modelos geradores, fornecendo amostras de alta qualidade. Entretanto, a qualidade dos materiais existentes afeta o pleno entendimento dessa técnica. Neste trabalho, são vistos todos os principais aspectos teóricos e práticos desses modelos, com a realização de experimentos para suportar e validar a teoria, através da execução de uma RAG estado da arte utilizada para resolver um problema atual.

1. Introdução

O paradigma conexionista da inteligência artificial propõe que os computadores repliquem a forma de obtenção do conhecimento e a habilidade de resolução de problemas empregando modelos gerados a partir de observações da natureza. Desse contexto surgem os modelos geradores, esses modelos representam a habilidade de produzir novas variáveis pela análise de características que podem ser inferidas de variáveis existentes.

Um conjunto de dados pode ser representado por uma distribuição de probabilidade. Desse modo, os modelos geradores representam distribuições de probabilidade sobre múltiplas variáveis Goodfellow et al. (2016), definindo como estratégia encontrar a "real" distribuição dos dados. Ao encontrar essa distribuição, será possível gerar novas variáveis que, por pertencerem a mesma distribuição, são similares, mas não iguais, as variáveis existentes.

As *Boltzmann Machines* E. Fahlman et al. (1983); Ackley et al.; E. Hinton and Sejnowski (1986); Hinton et al. (1984) foram uma primeira tentativa de implementação dos modelos geradores. Esse grupo de técnicas usa cadeias de Markov para o treinamento e para a geração de amostras do modelo. Com isso, ela assimila distribuições de probabilidade de vetores binários e, dessa forma, consegue modelar relações lineares entre variáveis Goodfellow et al. (2016). Embora historicamente importante, seu uso tornou-se raro pois não foi possível estender essa técnica para os problemas atuais, como geração de imagens. Complexos e exigindo muito poder computacional, os modelos geradores foram ignorados em sua época.

O desenvolvimento do *deep learning*, reduzindo a complexidade computacional das implementações, em conjunto com a evolução das unidades de processamento gráfico (GPUs), elevando o poder computacional disponível, proporcionou o renascimento dos modelos geradores. Uma técnica fruto dessas inovações são as *Generative Adversarial Networks* (GANs) Goodfellow et al. (2014).

As GANs são compostas de duas redes neurais, uma discriminadora e uma geradora, competindo entre si. Há inúmeras pesquisas apresentando excelentes resultados em diferentes áreas tais como geração de imagens Karras et al. (2017) e vídeos Tulyakov et al. (2018), processamento de textos Reed et al. (2016) e música Yang et al. (2017). Há vários problemas em aberto, visto que é uma tecnologia nova, sendo os mais relevantes o elevado tempo de treinamento, falta de convergência, de métricas de avaliação e de critério de parada.

GANs são uma tecnologia recente e se encontram na vanguarda de pesquisas envolvendo modelos geradores. O trabalho de Goodfellow (2017), desenvolvido com a intenção de ser um tutorial e utilizado como base desse trabalho, é o único a reunir e discutir os principais elementos das GANs. Porém, não apresenta experimentos e discussões práticas sobre o assunto, comparações práticas com as arquiteturas estado da arte, novas funções de custo e métricas de avaliação desenvolvidas. Desse modo, este trabalho se propõe a realizar um estudo detalhado

**Corresponding author: Tel.: +55 (49) 99902-0983;
e-mail: fabiomoreira.mf@gmail.com (Fabio Moreira)

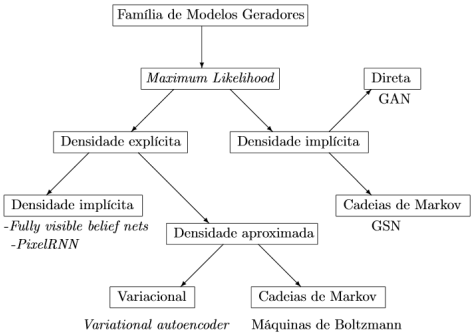


Fig. 1: Taxonomia dos modelos geradores. Apesar dos modelos empregarem outras técnicas em vez de *maximum likelihood*, é possível utilizá-la em todos eles. Assim, para evitar que as diferenças de cada modelo atrapalhem a análise, a técnica comum entre eles foi a escolhida. Fonte: Adaptado de Goodfellow (2017).

sobre as GANs, considerando sua arquitetura e os principais conceitos teóricos que as sustentam, vantagens e desvantagens sobre outros modelos geradores, a etapa de treinamento, os principais problemas em aberto, suas limitações e o estado da arte. Este trabalho não se propõe a desenvolver novas ferramentas ou técnicas, tampouco desenvolver um software funcional. Todas as implementações criadas servirão de experimentos com o objetivo de contextualizar com as questões teóricas discutidas ao longo do trabalho.

2. Taxonomia dos modelos geradores

GANs não foram os primeiros modelos geradores criados e não são os únicos estudados atualmente. Há várias famílias de modelos que, embora não sejam tão boas quanto as GANs, apresentam resultados teóricos importantes para o avanço dessa área. Nesta seção, os principais modelos são descritos de acordo com a taxonomia da Figura 1.

2.1. Densidade explícita

Modelos de densidade explícita são modelos que, dado um conjunto de treinamento, devem explicitamente definir uma distribuição de probabilidade p_{model} que mais se aproxima da verdadeira distribuição daquele conjunto.

O principal problema é capturar, em um único modelo, todas as características do conjunto sem torná-lo intratável Goodfellow (2017). Para resolver isso, duas estratégias foram adotadas.

A primeira, chamada de densidade tratável, visa capturar todas as características e complexidades inerentes ao conjunto até o ponto onde o modelo seja tratável. A segunda, chamada de densidade aproximada, cria um modelo intratável, porém admite uma aproximação.

2.1.1. Fully visible belief nets

As *Fully visible belief nets* Frey et al. (1996); Frey (1998), ou FVBNs, são um caso especial das máquinas de Helmhotz. Essa classe de modelos pode ser melhor definida com a discussão do seu modelo mais popular.

PixelRNN van den Oord et al. (2016) é um modelo da família das FVBNs, a proposta desse modelo é produzir uma distribuição que estime corretamente a distribuição real de um conjunto, onde seja possível gerar novas imagens, mantendo a complexidade em um nível tratável.

O objetivo é predizer o próximo pixel baseado nos pixels anteriores. Isso pode ser resumido na Equação 1.

$$p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}) \tag{1}$$

Onde $p(x)$ é a probabilidade conjunta, ou p_{model}, atribuída a uma imagem x composta de $n \times n$ pixels e $p(x_i|x_1, ..., x_{n-1})$ é a probabilidade do pixel x_i dado os pixels anteriores a ele.

O principal problema com a família FVBNs em geral é a geração de novas imagens. Enquanto as GANs geram novas amostras diretamente, as FVBNs produzem uma amostra nova um pixel por vez.

2.1.2. Variational autoencoders

Variational autoencoders Kingma (2013); Kingma et al. (2016), ou VAEs, são modelos que utilizam aproximações variacionais e um dos modelos mais populares, junto com as FVBNs e GANs Goodfellow (2017).

Um conjunto de imagens é passada a um codificador que irá mapear cada imagem a um par de valores, média e desvio padrão, e os armazenará em dois vetores. A média e o desvio padrão servem como rótulos de uma distribuição. Por exemplo, o par de valores [5.24, 3.24] representa um gato e o par [1.24, 0.22] uma mesa. Assim, para gerar uma nova imagem, a média e o desvio padrão são combinados, devendo seguir uma certa restrição que segue uma distribuição gaussiana, passados pelo decodificador e, após esse processo, uma nova imagem é gerada.

Uma característica importante que a difere das GANs é a rotulação das imagens. Enquanto nas GANs é feito apenas a diferenciação entre imagens "reais" de "falsas", nas VAEs há restrições sobre as características das imagens. Dessa forma, um gato não pode possuir três orelhas ou um sapo ter um chifre, pois seus valores de média e desvio padrão estarão afastados no espaço latente. Ao gerar uma nova imagem, um par de valores (média, desvio padrão) será selecionado desse espaço, mas não haverá combinações de pares de valores pertencentes a esse espaço, impedindo resultados tais como os citados acima. Assim, ao selecionar um vetor qualquer e adicionar ruído a ele, é possível obter uma imagem semelhante a original, mas sem extrapolar características básicas da imagem.

No entanto, a qualidade das imagens das VAEs é inferior as geradas pelas GANs, muitas vezes apresentando borramento.

2.1.3. Boltzmann machines

Boltzmann machines são redes estocásticas de aprendizado não-supervisionado sobre vetores binários, ou seja, dado um vetor binário como entrada da rede, o método é capaz de,

sem auxílio externo, aprender[1] as representações internas dessa rede. Essas redes foram as primeiras capazes de resolver problemas combinatórios difíceis. Uma propriedade interessante é a possibilidade da alteração da rede para se resolver problemas de busca ou de aprendizado.

A questão com essas redes é o fato de não escalar quando o conjunto é composto de imagens, ou melhor, distribuições de probabilidade de alta dimensionalidade. Em vários momentos, ou o modelo não converge em tempo razoável, levando o usuário a finalizar o treinamento antes da hora, ou o modelo não aprende corretamente. O principal representante dessa família atualmente são as *Restricted Boltzmann Machines*.

As *Restricted Boltzmann Machines* Smolensky (1986) facilitam o aprendizado ao restringir a conectividade entre as camadas da rede neural. Nessas redes, há apenas duas camadas: a primeira, chamada de visível, que está relacionada aos componentes de uma entrada, um nodo da camada visível por pixel de uma imagem, por exemplo; a segunda, chamada escondida, modela a dependência entre os componentes da entrada, sendo que a comunicação se dá apenas entre camadas e não intracamadas. Essa diferença permitiu a aplicabilidade dessa técnica a problemas de classificação, regressão, aprendizado entre outros. A geração de um modelo de distribuição de probabilidade para tentar representar a distribuição original, chamado aqui de reconstrução, é realizado pela fase de retropropagação, que atualiza os pesos dos nodos na camada visível da rede.

2.2. Densidade implícita

Oposto aos modelos de densidade explícita, os modelos de densidade implícita são modelos que, dado um conjunto de dados, não necessitam que a distribuição do conjunto seja dada explicitamente ou seja descoberta pela algoritmo de aprendizado, apenas amostrando imagens do conjunto é suficiente para o treinamento. Em muitos casos, o modelo é construído iterativamente, como é o caso das GANs. Ao gerar o modelo, há uma avaliação das amostras geradas por p_{model} e essa avaliação volta ao modelo gerador, que promoverá uma correção das falhas do modelo gerado.

2.2.1. Generative Stochastic Networks

As *Generative Stochastic Networks* (GSNs) constroem um modelo de distribuição de probabilidade baseado na observação de um operador de transição de uma cadeia de Markov, cuja distribuição estacionária estima a distribuição dos dados Bengio et al. (2013).

A base dessa técnica é: dado um conjunto X de imagens amostrado de uma distribuição de probabilidade desconhecida P(X), encontrar P(X) pela corrupção das variáveis X (denominadas $\tilde{X}$), gerando uma distribuição C($\tilde{X}|X$). Após gerada essa distribuição, são utilizadas técnicas de aprendizado supervisionado para treinar uma função que reconstroi X dado $\tilde{X}$. A distribuição gerada dessa reconstrução é menos computacionalmente complexa de modelar do que P(X). Encontrado essa distribuição de reconstrução, a aproximação para P(X) segue uma regra bayesiana Bengio et al. (2013).

O algoritmo final utiliza cadeias de Markov que vão amostrar pares de valores (X, $\tilde{X}$), além de, iterativamente, realizar alteração nos parâmetros que afetam toda a cadeia de treinamento.

Uma contribuição dessa técnica é a generalização da interpretação dos *denoising auto-enconders*, uma técnica que visa reconstruir os dados de uma entrada corrompida, adicionando o conceito de variáveis latentes para, além de reconstruir, gerar novas entradas. Por esse fato, a utilização das GSNs é focada na área de recuperação de valores perdidos ou corrompidos.

Na questão da densidade, as GSNs estimam a distribuição de dados gerada indiretamente, em vez de tentar gerar p_{model} diretamente. Com isso, é possível evitar a computação de funções intratáveis sem a necessidade de gerar previamente o modelo.

2.3. Comparação dos modelos

Apresentados os modelos, é interessante compará-los as GANs para entender melhor o comportamento e as características de cada um. As conclusões aqui apresentadas se basearam na análise comparativa de Goodfellow (2017).

- **Densidade**: Ao contrário das FVBNs, VAEs e *Boltzmann Machines*, as GANs não necessitam que a distribuição de um conjunto seja dado explicitamente. O modelo é tratável e construído iterativamente.
- **Consistência assintótica**: GANs são o único modelo que é conhecido por ser assintoticamente consistente.
- **Cadeias de Markov**: Modelos que utilizam cadeias de Markov costumam ser engessados, possuindo várias restrições quanto a função geradora ou a dimensão do vetor latente z. GANs, por outro lado, não possuem essas restrições.
- **Geração de amostras**: GANs podem gerar uma amostra instantaneamente, ao contrário das FVBNs que precisam gerar um pixel por vez, ou em relação a alguns modelos que precisam repetidamente aplicar o operador da cadeia de Markov para gerar novas amostras.
- **Equilíbrio**: GANs costumam ser instáveis, pois é necessário encontrar o equilíbrio de Nash ao realizar o treinamento, o que é uma tarefa difícil.
- **Dados discretos**: GANs têm dificuldade em gerar dados discretos, como textos.

3. Redes Geradoras Adversárias

Antes da discussão sobre o funcionamento das GANs, é necessário analisar as diferenças entre os modelos discriminadores e geradores.

Os modelos discriminadores resolvem problemas de classificação. Dado um conjunto de treinamento e um conjunto de rótulos, o modelo tentará relacionar corretamente cada

[1] Para a área da inteligência artificial, aprender significa adquirir informações e regras para utilizá-las.

instância do conjunto de treinamento a uma instância do conjunto de rótulos. Matematicamente, esse problema pode ser modelado como: considerando y um rótulo e x um conjunto de características extraídas do conjunto de treinamento, pode-se definir a relação de x e y como $p(y|x)$, ou seja, qual a probabilidade de y dado x. Para exemplificar, considere um conjunto de treinamento composto dos números zero a nove. É tarefa do discriminador determinar se, dado um número desse conjunto, ele é ou não o número nove.

Os modelos geradores atuam de forma oposta. Dados os mesmos dois conjuntos definidos acima, o modelo tentará aprender quais características são necessárias para definir um certo rótulo. Aqui, a relação entre y e x é inversa e dada por $p(x|y)$, que expressa a probabilidade de x dado y. Voltando ao exemplo, é tarefa do gerador determinar quais características extraídas do conjunto são necessárias e suficientes para considerar um certo número como sendo o número nove.

3.1. Arquitetura

As GANs são compostas por dois modelos:

- Um discriminador **D**, treinado para diferenciar amostras que não pertencem à distribuição real das que pertencem. Esse modelo é uma função diferenciável $D(x, \theta_d)$, onde x é a amostra a ser analisada considerando um conjunto de parâmetros configuráveis θ_d, que resulta em um único valor de saída, geralmente esse valor está no intervalo [0,1], sendo 0 para amostras que o discriminador tem certeza que não pertencem à distribuição e 1 para amostras que ele tem certeza que pertencem.

- Um gerador **G**, treinado para construir uma distribuição p_{model} que é capaz de capturar, em um cenário ideal, todas as características da distribuição original p_{data}. Esse modelo também é uma função diferenciável $G(z, \theta_g)$, onde z é um vetor latente e θ_g são os parâmetros configuráveis de **G**.

A Figura 2 representa um esquemático da arquitetura das GANs. As distribuições do mundo real são complexas e demandam a análise de várias características das amostras para uma cópia fiel dessas distribuições, em razão disso, o modelo é construído de forma iterativa e incremental. Inicialmente, algumas amostras x da distribuição original são extraídas para realizar o treinamento do modelo discriminador. Esse passo inicial permite que o discriminador classifique corretamente futuras amostras. Em seguida, um código latente z é extraído e passado para o gerador G, que tenta reproduzir a distribuição original, resultando em um p_{model}. Algumas amostras $\overline{x}$ do p_{model} são utilizadas para treinar o discriminador. O objetivo agora é treinar o discriminador com as amostras retiradas da distribuição p_{model} e esperar que essas sejam consideradas como amostras da distribuição original, maximizando o erro do discriminador. O resultado desse processo é utilizado para corrigir eventuais falhas de ambos os modelos, que refazem o procedimento considerando essas novas informações.

O processo pode ser visto como um jogo minimax composto de dois jogadores, onde o discriminador tenta maximizar a probabilidade de diferenciar corretamente as amostras e o gerador tenta minimizá-lo. A solução para esse problema é o equilíbrio de Nash. Em certo ponto do treinamento, o discriminador não é capaz de diferenciar amostras reais de falsas, ou seja, a saída sempre será $0,5$, em contrapartida, o gerador não alterará a distribuição p_{model}, pois seu objetivo de enganar o discriminador foi alcançado.

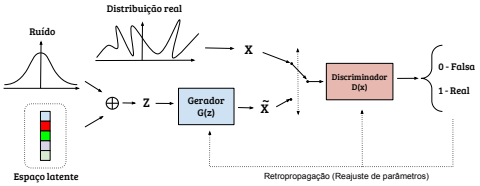


Fig. 2: Arquitetura de uma GAN. Fonte: Produzida pelo autor.

4. Pesquisas atuais

GANs são modelos recentes, existindo inúmeras pequisas em aberto. Nesta seção, são apresentados os principais problemas em estudo.

4.1. Falta de convergência

O treinamento de uma GAN envolve encontrar o equilíbrio de Nash em um jogo não cooperativo entre dois jogadores. O problema está na convergência do modelo ao realizar o treinamento utilizando a técnica do gradiente descendente. Em certos jogos, ou o equilíbrio é obtido ou cada movimento de um jogador pode desfazer todo o progresso que o outro fez para alcançar o equilíbrio. Isso acontece porque os custos de cada jogador são atualizados desconsiderando o outro e, como consequência, pontos de mínimo local de um são considerados de máximo pelo outro, fazendo com que o gradiente fique rotacionando entre os dois, deixando o treinamento instável. Esse é um problema com jogos em geral, não apenas das GANs. O principal problema nesse tema é o *Mode collapse*.

O *Mode collapse*, ou *Helvetica scenario*, ocorre quando o gerador descobre uma falha no discriminador e passa a explorá-la, gerando amostras com baixa ou nenhuma diversidade. A gravidade desse problema pode ser total, todas as amotras são iguais, ou parcial, algumas amostras são iguais, como o exemplo exibido na Figura 3.

Fig. 3: *Mode collapse* parcial de um conjunto de amostras sobre uma base de dados de camas. Fonte: Arjovsky et al. (2017)

4.2. Métricas de avaliação

As funções de custo apresentadas são utilizadas para medir a pontuação de dois jogadores em um jogo. Entretanto, essas funções não avaliam a qualidade ou a diversidade das amostras geradas. Algumas técnicas visam tratar esses problemas:

Inception Score (IS) Salimans et al. (2016): Avalia a qualidade e a diversidade das amostras utilizando entropia.

- Qualidade das amostras: Entropia alta, qualidade ruim. Entropia baixa, qualidade boa.
- Diversidade: Entropia alta, diversidade alta. Entropia baixa, diversidade baixa (*Mode collapse*).

Esses dois critérios são combinados na Equação 2. Onde D_{KL} representa a divergência de Kullback-Leibler e $p(y|x)$ é a probabilidade de y dado x.

$$IS(G) = exp(\mathbb{E}_{x\sim p_g} D_{KL}(p(y|x)\|p(y))) \quad (2)$$

Entropia, nesse contexto, está relacionada à aleatoriedade. Por exemplo, considere uma GAN com uma base de treinamento de imagens de gatos. Ao avaliar, manualmente, um conjunto de imagens geradas a partir dessa base, um avaliador irá escolher uma imagem qualquer e verificar se essa imagem possui as características necessárias para ser considerada uma imagem real de um gato. Em um segundo teste, ele avaliará se há uma quantidade suficiente de imagens de diferentes gatos, e não apenas um grande conjunto com várias imagens de um mesmo gato. Traduzindo esse processo para a técnica do IS, ela irá avaliar se, dada uma imagem qualquer, essa imagem é altamente predizível, ou seja, é possível afirmar que se trata de uma imagem de um gato. Se sim, a entropia dessa imagem será baixa. Em seguida, ela avaliará o conjunto analisando se ele possui diversidade suficiente. Se sim, esse conjunto será impredizível e sua entropia será alta, ou seja, não há um padrão que o gerador segue para gerar novas classes de imagens, ou melhor, ele gera novas classes de imagens seguindo uma distribuição uniforme.

Fréchet Inception Distance (FID) Heusel et al. (2017): O FID extrai características das amostras durante o processo de treinamento e modela uma distribuição de dados a partir dessas. Em seguida, o FID entre as amostras geradas g e as amostras reais x é calculado através da Equação 3.

$$FID(x,g) = \|\mu_x - \mu_g\|_2^2 + Tr(\sum_x + \sum_g - 2(\sum_x + \sum_g)^{\frac{1}{2}}) \quad (3)$$

Sendo $(\mu_x, \sum_x)$ e $(\mu_g, \sum_g)$ a média e covariância das amostras das distribuições reais e geradas, respectivamente, $Tr(A)$ o traço de uma matriz A e $\|x\|_2^2$ o produto de um vetor de dimensão 2 com ele mesmo.

Como a técnica avalia a distância entre duas distribuições, quanto menor o FID, mais semelhantes as distribuições são e, por consequência, a qualidade e a diversidade das amostras são altas.

Ambas as técnicas são úteis para verificar se ocorreu *Mode collapse*. Porém, o FID é mais robusto ao ruído do que o IS Heusel et al. (2017), conseguindo produzir resultados mais consistentes em relação à diversidade das amostras.

4.3. Problemas referentes à imagens

- Contagem

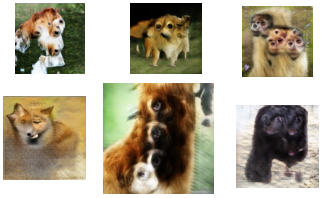


Fig. 4: GANs têm dificuldade em inserir a quantidade semanticamente correta de elementos. Fonte: Goodfellow (2017)

- Perspectiva

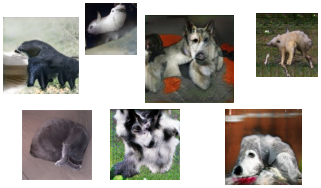


Fig. 5: GANs não produzem imagens que passam a sensação de perspectiva, resultando em imagens planas. Fonte: Goodfellow (2017)

- Estrutura Global

Fig. 6: GANs não conseguem aprender o conceito de estrutura global, gerando imagens de cachorros com duas orelhas. Fonte: Goodfellow (2017)

5. Análise do algoritmo

Antes de demonstrar o funcionamento prático de uma GAN, é necessário compreender o núcleo de uma GAN e as consequências resultantes de sua arquitetura. O Algoritmo 1 apresenta o processo de treinamento de uma GAN como definido no artigo original, há diferentes versões de GANs e melhorias propostas que alteraram esse processo.

Com o algoritmo definido, é interessante notar que:

- Em uma iteração do treinamento, o número de iterações do discriminador pode ser maior que a do gerador, indicando que não há problema em tornar o discriminador mais poderoso que o gerador.

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

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

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

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

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
Fonte: Goodfellow et al. (2014)

- Implementação de D e G: Não são definidas restrições de implementação para as redes neurais usadas. De fato, várias implementações utilizam diferentes estratégias, como a DCGAN que implementa as redes utilizando convoluções profundas.

- Flexibilidade na utilização de métodos baseados em gradientes. Como comentado no algoritmo, qualquer método pode ser utilizado, inclusive, pode-se definir diferentes métodos para o gerador e o discriminador, como será visto adiante.

6. Prova de Conceito - SRGAN

Elucidado os aspectos teóricos e práticos sobre as GANs, é necessário mostrar uma aplicação real que resolva um problema para o qual hoje, ou não exista solução ou a aplicação das GANs produz resultados melhores em relação às soluções existentes. Esta seção apresenta a *Super Resolution* GAN (SRGAN).

6.1. SRGAN

A SRGAN Ledig et al. (2016) visa produzir imagens de alta resolução. Após treinado, o modelo aceita como entrada uma imagem de baixa resolução e produz como saída uma imagem de super resolução que, no mínimo, preserva a qualidade original da imagem.

A arquitetura lógica segue o mesmo princípio das GANs tradicionais. O gerador recebe como entrada uma imagem de baixa resolução (originalmente de alta resolução, sofrendo um processo de subamostragem) e, como saída, produz uma imagem de super resolução. O discriminador diferencia imagens de alta resolução das de super resolução (produzidas pelo gerador).

6.2. Materiais e Métodos

O algoritmo que implementa a SRGAN foi obtido em Dong et al. (2017), foi implementada pelos desenvolvedores da biblioteca TensorLayer, como uma implementação de exemplo que a utiliza. Ele sofreu apenas uma adaptação para que consiga processar imagens em lote. A aplicação utilizou o modelo pré-treinado existente.

Foram utilizadas 106 imagens de teste coloridas e em preto e branco, retiradas de Agustsson and Timofte (2017), Timofte et al. (2017) e Gonzalez (2017), com tamanho médio de 500 × 500.

O modelo foi executado em um computador equipado com um processador i3-3250 3.5 GHz, 8 GB de RAM.

6.3. Resultados obtidos

Fig. 7: Resultados obtidos. À esquerda, o resultado da SRGAN (Ampliada 4x). À direita, a imagem original. Fonte: Produzida pelo autor.

7. Conclusão e trabalhos futuros

Neste trabalho, de forma didática, os principais aspectos teóricos e práticos das GANs foram discutidos. A principal arquitetura estado da arte funcional e estável, a DCGAN, foi

descrita e detalhada. Foram desenvolvidos experimentos para suportar a teoria, além de realizar a validação das técnicas propostas seguidos da discussão dos resultados obtidos e como eles podem ser analisados para inferir possíveis problemas de treinamento. Por fim, uma GAN estado da arte foi utilizada para ser aplicada em problemas enfrentados atualmente, provendo excelentes resultados. Todos esses aspectos qualificam esse trabalho para ser utilizado como ponto de partida para futuros trabalhos sobre GANs. Podemos elencar como trabalhos futuros:

- Estudo sobre novas métricas de avaliação: Novas métricas estão sendo desenvolvidas, como o *Kernel Inception Distance* Bińkowski et al. (2018), e uma comparação entre elas, considerando vantagens e desvantagens, é interessante. Além disso, é possível avaliar métricas utilizadas em outras técnicas e tentar adaptá-las as GANs.

- Estudo sobre diferentes componentes das GANs: Cada componente influencia o comportamento final do treinamento de uma certa forma, é importante analisar como essa influência acontece e se há componentes com influência sobre outros.

- Adaptar e/ou combinar diferentes GANs: A construção de um modelo estável de GAN é difícil e demanda muito tempo. Assim, a adaptação de modelos existentes pode trazer resultados significativos. Por exemplo, como visto, a WGAN-GP é uma WGAN com penalidade no gradiente, essa ideia pode ser adaptada à DCGAN.

References

Ackley, D.H., Hinton, G.E., Sejnowski, T.J., . A learning algorithm for boltzmann machines. Cognitive Science 9, 147–169. doi:10.1207/s15516709cog0901_7.

Agustsson, E., Timofte, R., 2017. Ntire 2017 challenge on single image super-resolution: Dataset and study, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.

Arjovsky, M., Chintala, S., Bottou, L., 2017. Wasserstein GAN. arXiv e-prints , arXiv:1701.07875arXiv:1701.07875.

Bengio, Y., Thibodeau-Laufer, E., Yosinski, J., 2013. Deep generative stochastic networks trainable by backprop. CoRR abs/1306.1091. URL: http://arxiv.org/abs/1306.1091, arXiv:1306.1091.

Bińkowski, M., Sutherland, D.J., Arbel, M., Gretton, A., 2018. Demystifying MMD GANs, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=r1lUOzWCW.

Dong, H., Supratak, A., Mai, L., Liu, F., Oehmichen, A., Yu, S., Guo, Y., 2017. TensorLayer: A Versatile Library for Efficient Deep Learning Development. ACM Multimedia URL: http://tensorlayer.org.

E. Fahlman, S., E. Hinton, G., Sejnowski, T., 1983. Massively parallel architectures for ai: Netl, thistle, and boltzmann machines. , 109--113.

E. Hinton, G., Sejnowski, T., 1986. Learning and relearning in boltzmann machines 1.

Frey, B.J., 1998. Graphical Models for Machine Learning and Digital Communication. MIT Press, Cambridge, MA, USA.

Frey, B.J., Hinton, G.E., Dayan, P., 1996. Does the wake-sleep algorithm produce good density estimators?, in: Advances in Neural Information Processing Systems 8. MIT Press, pp. 661--667. URL: papers.nips.cc/paper/1153-does-the-wake-sleep-algorithm-produce-good-density-estimators.pdf.

Gonzalez, Woods, E., 2017. Image databases. URL: http://www.imageprocessingplace.com/root_files_V3/image_databases.htm. acessado: 2019-04-03.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. URL: http://www.deeplearningbook.org. book in preparation for MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial networks 3.

Goodfellow, I.J., 2017. NIPS 2016 tutorial: Generative adversarial networks. CoRR abs/1701.00160. URL: http://arxiv.org/abs/1701.00160, arXiv:1701.00160.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., Hochreiter, S., 2017. Gans trained by a two time-scale update rule converge to a nash equilibrium. CoRR abs/1706.08500. URL: http://arxiv.org/abs/1706.08500, arXiv:1706.08500.

Hinton, G.E., Sejnowski, T.J., Ackley, D.H., 1984. Boltzmann Machines: Constraint Satisfaction Networks That Learn. Technical Report CMU-CS-84-119. Computer Science Department, Carnegie Mellon University. Pittsburgh, PA.

Karras, T., Aila, T., Laine, S., Lehtinen, J., 2017. Progressive growing of gans for improved quality, stability, and variation .

Kingma, D.P., 2013. Fast gradient-based inference with continuous latent variable models in auxiliary form. CoRR abs/1306.0733. URL: http://arxiv.org/abs/1306.0733, arXiv:1306.0733.

Kingma, D.P., Salimans, T., Welling, M., 2016. Improving variational inference with inverse autoregressive flow. CoRR abs/1606.04934. URL: http://arxiv.org/abs/1606.04934, arXiv:1606.04934.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., Shi, W., 2016. Photo-realistic single image super-resolution using a generative adversarial network. CoRR abs/1609.04802. URL: http://arxiv.org/abs/1609.04802, arXiv:1609.04802.

van den Oord, A., Kalchbrenner, N., Kavukcuoglu, K., 2016. Pixel recurrent neural networks. CoRR abs/1601.06759. URL: http://arxiv.org/abs/1601.06759, arXiv:1601.06759.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H., 2016. Generative adversarial text to image synthesis, in: Proceedings of The 33rd International Conference on Machine Learning.

Salimans, T., Goodfellow, I.J., Zaremba, W., Cheung, V., Radford, A., Chen, X., 2016. Improved techniques for training gans. CoRR abs/1606.03498. URL: http://arxiv.org/abs/1606.03498, arXiv:1606.03498.

Smolensky, P., 1986. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1, MIT Press, Cambridge, MA, USA. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pp. 194--281. URL: http://dl.acm.org/citation.cfm?id=104279.104290.

Timofte, R., Agustsson, E., Van Gool, L., Yang, M.H., Zhang, L., Lim, B., et al., 2017. Ntire 2017 challenge on single image super-resolution: Methods and results, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.

Tulyakov, S., Liu, M.Y., Yang, X., Kautz, J., 2018. Mocogan: Decomposing motion and content for video generation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) .

Yang, L.C., Chou, S.Y., Yang, Y.H., 2017. Midinet: A convolutional generative adversarial network for symbolic-domain music generation, in: ISMIR.