Capítulo 4

Geração de Séries Temporais Utilizando Redes Generativas Adversárias: da Teoria à Prática

Iran F. Ribeiro[1], Breno Krohling[1], Giovanni Comarela[1], Vinícius F. S. Mota[1]

[1]Departamento de Informática - Universidade Federal do Espírito Santo

{iran.ribeiro@edu.ufes, breno.krohling}@edu.ufes.br

{gc.inf, vinicius.mota}@inf.ufes.br

Abstract

Time series is a concept used to model a sequence of data gathered over time. These temporal data usually present properties that allow modeling and forecasting of their applications' scenarios, such as eHealth, intelligent transportation, media recommendation, or any business intelligence application. However, due to business roles and privacy issues, most of the data gathered by enterprises are private and protected. An alternative to tackle these problems is providing the data model instead of the raw data, leading to more transparency and participation of the research community. In such a case, a good data model can provide synthetic data based on the original data, preserving its characteristics while maintaining its privacy. Thus, this chapter discusses the main concepts of modeling time series data. We discuss the main concepts, classical statistical techniques, and deep learning approaches to model the data. In particular, we focus on the generative adversarial network, used initially to transform images, as a novel technique to reproduce time series. We follow with a hands-on lab to model and reproduce synthetic data stemmed from the raw data.

Resumo

Série temporal é um conceito usado para modelar uma sequência de dados coletados ao longo do tempo. Esses dados temporais geralmente apresentam propriedades que

permitem a modelagem e previsão dos cenários de suas aplicações, como eHealth, transporte inteligente, recomendação de mídia, entre outras. No entanto, devido às regras de negócio e problemas de privacidade, a maioria dos dados coletados por empresas é protegida. Uma alternativa é fornecer o modelo de dados ao invés dos dados brutos, levando a mais transparência e participação da comunidade de pesquisa. Nesse caso, um bom modelo de dados pode fornecer dados sintéticos com base nos dados originais, preservando suas características e mantendo sua privacidade. Deste modo, este capítulo discute os principais conceitos para a modelagem de dados de séries temporais. Para isto, são apresentados os principais conceitos, técnicas estatísticas clássicas e abordagens de aprendizado profundo para modelar os dados, em particular, as redes adversárias generativas. Esta última, usada inicialmente para transformar imagens, e agora sendo utilizada como uma nova técnica para reproduzir séries temporais. Por fim, o capítulo apresenta um laboratório prático com implementações para modelar e reproduzir dados sintéticos derivados dos dados brutos.

4.1. Introdução

O aumento de dispositivos de sensoriamento coletando dados continuamente, bem como o armazenamento estruturado de uma enorme quantidade de informações permite uma melhor compreensão da sociedade, natureza, saúde e inteligência de negócios. Em especial, eventos cuja coleta de dados deve ocorrer a cada período podem ser modelados como séries temporais. Exemplos de tais eventos vão desde tráfego de veículos por hora nas ruas e avenidas da cidade, observação da concentração de monóxido de carbono (*CO*) em uma região durante um período, medição de eletrocardiograma (ECG) e monitoramento de acessos simultâneos (carga) em servidores.

A modelagem destes dados como séries temporais permite realizar classificações e predições de informação. Por exemplo, detecção de valores atípicos (*outliers*), agrupamento por semelhança, e predição de eventos, o que agrega de fato valor aos dados brutos. Contudo, embora haja um esforço contínuo para tornar os dados em domínio público para pesquisadores, tais dados podem conter informações restritas, tais como posicionamento ao longo do tempo, transições entre estações-base de uma rede sem fio (ZHENG et al., 2009; Malandrino; Chiasserini; Kirkpatrick, 2018), transições entre sessões de conferências técnicas (SCOTT et al., 2009; RIBEIRO et al., 2021a). Além disso, na prática, dados reais são propensos a conterem erros, a base de dados brutos pode ser grande demais para compartilhamento, falta de dados e exigem garantias de privacidade antes de serem disponibilizados ao público.

Uma abordagem para contornar os problemas acima é a geração de modelos de séries temporais capazes de gerar dados com as mesmas propriedades estatísticas dos dados brutos. Além disso, um modelo generativo permite maior repetibilidade para pesquisas e experimentos, uma vez que cada base gerada é diferente. Deste modo, um

grupo de pesquisa que possua dados sensíveis pode compartilhar com a comunidade científica apenas o modelo gerador da série temporal ao invés dos dados brutos, preservando, assim, a privacidade dos dados. É importante ressaltar que um modelo generativo deve considerar as características intrínsecas de cada cenário e aplicação. Tal heterogeneidade de cenários dificulta uma solução única e que capture as propriedades da aplicação que se deseja gerar dados.

A Figura 4.1 ilustra três cenários de aplicações distintas que geram dados continuamente, tais como (a) mobilidade de tráfego de veículos, monitoramento de redes e dados de sensores de saúde. No lugar de compartilhar os dados brutos, os dados podem ser processados visando identificar o melhor modelo generativo (b). Deste modo, um modelo generativo da série temporal pode ser compartilhado para que demais pesquisadores gerem os modelos sintéticos. Portanto, as primeiras perguntas que surgem e guiam este capítulo são: (1) Como gerar modelos generativos a partir de dados brutos? (2) Como mensurar a qualidade dos modelos geradores?

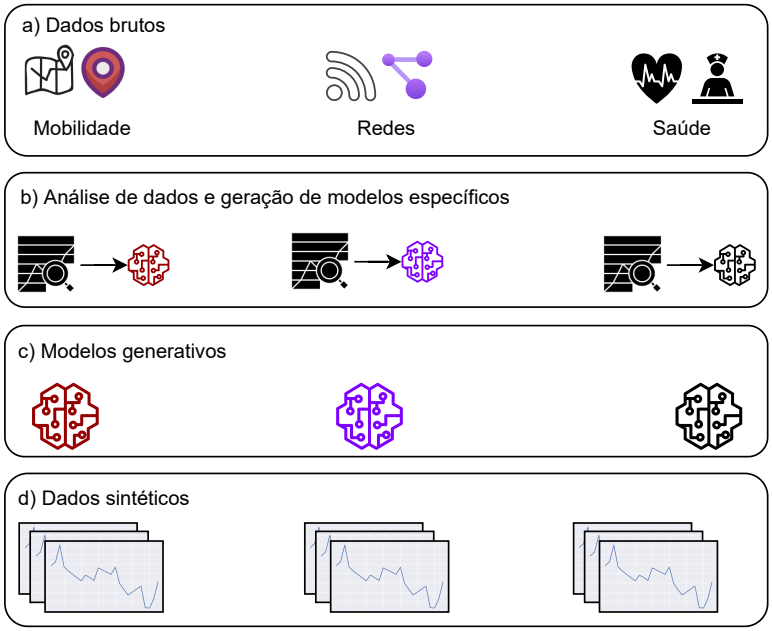


Figura 4.1: Aplicações heterogêneas que geram dados continuamente podem ter modelos capazes de gerar dados sintéticos similares.

Neste sentido, *o foco deste capítulo é apresentar as técnicas disponíveis na literatura para a geração de séries temporais*. Para responder a primeira pergunta, iremos apresentar as técnicas estatísticas clássicas, seguido por uma técnica de aprendizado profundo chamada Redes Generativas Adversárias (*Generative Adversarial Networks* – GANs). As GANs visam a otimização de modelos generativos baseados em aprendizado

profundo, originalmente focadas em projeção e recuperação de imagens (GOODFELLOW et al., 2014). O bom desempenho no campo de visão computacional levou pesquisadores de diversas áreas perceberam o potencial das GANs em gerar dados sintéticos com as características dos dados brutos reais (YI; WALIA; BABYN, 2019).

Adicionalmente, este capítulo discute o desafio de *como mensurar a qualidade do modelo para gerar conjuntos de dados sintéticos*. Normalmente, os modelos em séries temporais são utilizados para fazer previsões. Tais modelos preditivos consideram métricas, como erro absoluto médio (do inglês *Mean Average Error* MAE) ou seus equivalentes, que mensuram os erros de previsão. Tais modelos esperam valores de erros baixos, e um erro igual a zero significa que o modelo é ótimo, isto é, o modelo previu exatamente o valor esperado. No entanto, ao avaliar modelos generativos para reproduzir dados, baixos valores de erros ao comparar os dados sintéticos com os dados reais indicam que os dados sintéticos são praticamente uma cópia do original. Nestes casos, os modelos e os dados sintéticos podem representar um problema, principalmente quando se trata de questões de privacidade. Assim, os modelos generativos requerem métricas que capturem a variabilidade do conjunto de dados sintéticos reproduzidos pelo modelo.

Após a discussão teórica dos conceitos básico, técnicas e métricas, este capítulo apresenta um *hands on* visando a implementação prática para a geração de séries temporais de aluguel de bicicletas em estações de autosserviço nos EUA. Após a geração do modelo, os dados gerados por este modelo serão avaliados de forma qualitativa e quantitativa em relação aos dados originais.

4.2. Motivação e Casos de uso de Geração de Séries Temporais

Os algoritmos de aprendizado profundo têm se mostrado eficientes em diversas áreas de estudo e sendo aplicados para a resolução dos mais diversos tipos de problemas. Nesse sentido, apesar de ser uma tecnologia relativamente recente, ao longo deste capítulo apresentamos brevemente diversos exemplos de aplicação de GANs em séries temporais. Nota-se com os exemplos que, apesar dos desafios e limitações, as GANs podem ser aplicadas com eficiência para problemas envolvendo séries temporais.

4.2.1. Música

A geração dos mais diversos tipos de arte têm se mostrado uma das mais fascinantes aplicações da Inteligência Artificial nas últimas décadas. Sem dúvida, as GANs desempenham um papel fundamental nesse contexto, com a geração de imagens, vídeos e texto, por exemplo. O desempenho das GANs nessas áreas motivou também a investigação da geração de músicas a partir de modelos generativos, entre eles, as GANs. A seguir apresentamos apenas dois exemplos em relação à geração de músicas. Recomendamos a leitura dos trabalhos de (DONAHUE; MCAULEY; PUCKETTE, 2018; ENGEL et al.,

2019) que descrevem abordagens distintas sobre o mesmo assunto.

O primeiro trabalho a propor uma arquitetura de GANs para a geração de músicas é o de (MOGREN, 2016), que já foi mencionado em seções anteriores neste capítulo. Essa arquitetura é muito similar à GAN clássica, composta por um Gerador e um Discriminador, onde cada um dos componentes é uma rede neural recorrente, especificamente, LSTMs. O modelo foi treinado com arquivos de música no formato *midi* contendo características como duração, tom, intensidade e tempo desde o início do último tom. Além disso, cada arquivo representa uma música de um compositor clássico e é possível definir quais compositores serão usados para treinamento do modelo.

Em seu artigo, (DONG HAO-WEN E HSIAO; YANG; YANG, 2018) apresentam os principais desafios envolvendo a geração de notação musical, como dependência temporal, uma estrutura hierárquica interna (um único símbolo e um conjunto do mesmo símbolo possuem sentidos diferentes), necessidade de atender expectativas de ouvintes humanos (coerência, ritmo, tensão) e o fato de uma música geralmente ser composta de diferentes instrumentos. No trabalho, os autores consideram que existem três tipos de música: i) improvisada, onde os músicos executam os instrumentos sem nenhum arranjo pré-definido; ii) composta, em que a música é composta antes da execução; e iii) híbrido, onde os dois tipos são mesclados. Como são tarefas distintas, os autores propõem uma arquitetura específica para cada uma delas.

4.2.2. Mobilidade e Redes

Como mencionamos anteriormente, bases de dados de diversas áreas, como redes e mobilidade, têm seu acesso e divulgação limitados devido a diversos fatores, entre eles, a privacidade. Nesse sentido, ressalta-se que a área de redes é diretamente afetada pela mobilidade das pessoas, pois é tal mobilidade que dita planejamentos ou modificações de estruturas de rede (antenas, roteadores, etc.), bem como o estudo e criação de novas tecnologias. Além disso, a mobilidade também é importante para o estudo de redes sociais de pessoas, por exemplo, entender as características de grupos de indivíduos em uma cidade ou para avaliação de algoritmos de redes oportunísticas. Devido à falta de cenários reais, é comum que novas tecnologias sejam testadas em cenários simulados, que podem falhar em representar o ambiente em que a tecnologia será de fato utilizada. Nesse contexto, as GANs têm se mostrado uma alternativa a permitir que mais bases de dados representando cenários reais de mobilidade e, consequentemente, de redes, sejam disponibilizadas.

Exemplos de como a geração de mobilidade é feita com o uso de GANs podem ser vistos nos trabalhos de (AMIRIAN; HAYET; PETTRÉ, 2019; RAO et al., 2020), em que os autores criam modelos para geração de trajetórias feitas por pessoas e veículos ou em (ZHANG et al., 2019b, 2019a) em que são gerados dados representando o tráfego de carros em ruas de cidades. A partir de dados de redes de celulares (HUANG; KUR-

NIAWAN; SUN, 2022) propõem a utilização de GANs como uma forma de identificar anomalias em métricas importantes para o gerenciamento de redes de celular. Com o objetivo de aumentar a eficiência de algoritmos de previsão em dados de redes de celular, (WANG et al., 2020) propõem o uso de GANs como técnica de *data augmentation* (técnica onde mais amostras são adicionadas à base de dados).

Outros cenários interessantes abordando a mobilidade podem ser vistos nos trabalhos de (CHEN et al., 2019; ZHANG, 2019), que tratam da geração de dados para predição da formação de conexões entre pessoas, por exemplo, por *bluetooth* ou redes *ad hoc*. Esse tipo de dado pode ser usado em experimentos considerando diversos cenários, por exemplo, para comparar novos algoritmos com outros já conhecidos ou para avaliar algoritmos em novas tecnologias, como o 5G.

4.2.3. Saúde

Na área da saúde, uma das grandes aplicações de séries temporais é poder representar a evolução de pacientes ao longo do tempo. Essa representação é importante, por exemplo, para realização de previsões sobre a condição do paciente a partir da evolução do seu quadro clínico. Da mesma forma, esse tipo de dado pode auxiliar a entender como se dá a ocupação de leitos de hospitais ao longo do tempo, considerando que cada enfermidade exige tempos diferentes de tratamento e recuperação.

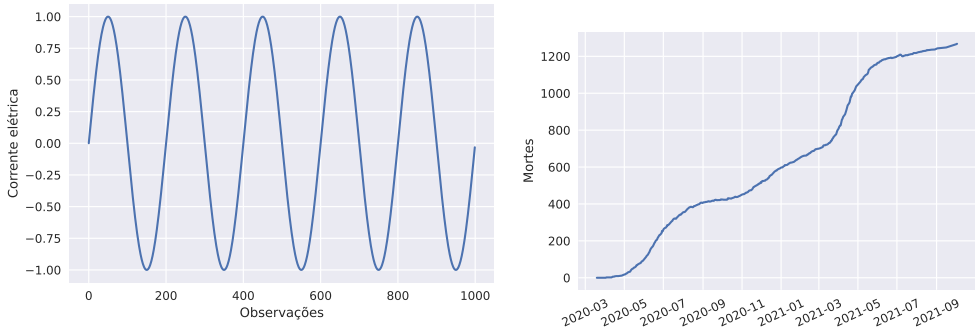
Um dos primeiros trabalhos a utilizar GANs para geração de dados médicos foi o de (ESTEBAN; HYLAND; RÄTSCH, 2017). Especificamente, os autores utilizam séries temporais de uma UTI para treinamento do modelo, considerando que tais séries podem conter dados numéricos ou categóricos. Uma aplicação direta dos dados gerados pelo modelo, é o treinamento de médicos que, segundo o autor, é geralmente feito por dados simulados que podem não representar a realidade de uma UTI. Em outro trabalho, (KADRI et al., 2022) utilizam dados de hospitais para prever o tempo em que determinados pacientes ficam nas salas de emergência. Essa é uma informação importante para diretores de hospitais, pois é possível prever e otimizar, com certa segurança, como os atendimentos no hospital ocorrem.

Nesse sentido, dados faltantes nas bases de dados são um grande problema em diversas áreas. Por exemplo, para realizar tarefas de previsões é esperado uma quantidade predefinida de variáveis de entrada. Nesse contexto, diversas técnicas podem ser utilizadas para tratar desses dados faltantes, entretanto, podem não ser ideais considerando a questão temporal dos dados. Como exemplos de trabalhos que utilizam GANs para tratar desse problema tem-se (LUO et al., 2018) e (OH et al., 2021).

4.3. Introdução às Séries temporais

Uma série temporal é um conjunto de observações registradas em um tempo específico (BROCKWELL; DAVIS, 2009). Cada observação pode representar, simultaneamente, uma ou mais variáveis. Deste modo, as séries temporais podem ser aplicadas em diversas áreas: no audiovisual, pode modelar dados de música (e.g, músicas)(ENGEL et al., 2019); em mobilidade humana, auxilia na análise e modelagem de trajetórias feitas por pessoas (RAO et al., 2020); na área de redes móveis, pode ser aplicada na modelagem de contatos feitos entre pessoas através de *bluetooth* (ZHANG, 2019); e na medicina, pode ser utilizada para análise e modelagem do tempo que um paciente demora em uma sala de emergência (KADRI et al., 2022).

Uma série temporal pode ser categorizada em dois tipos, discreta ou contínua, conforme os intervalos de tempo entre cada observação. A série temporal será discreta quando o intervalo de observações (t) é um conjunto discreto e cada observação ocorre em intervalos de tempo fixo, por exemplo, à cada segundo, minuto, ou hora. Por outro lado, a série será contínua quando as observações são registradas continuamente em um determinado intervalo de tempo, por exemplo, durante 1 minuto, são registradas todas a observações que ocorrem. Como exemplo, a Figura 4.2a mostra uma série temporal de uma corrente que passa por um resistor (contínua) durante 1 segundo e a Figura 4.2b representa as mortes (acumuladas) de Covid-19 entre março de 2020 e setembro de 2021, em Vitória-ES (discreta). Neste sentido, o foco principal deste capítulo será nas séries temporais discretas.



(a) Corrente passando por um resistor (b) Mortes causadas por Covid19

Figura 4.2: Exemplos de séries temporais: contínua, com a corrente que passa por um resistor (a); e discretas, com as mortes causadas por Covid-19 em Vitória-ES (b).

Uma série temporal pode possuir quatro propriedades que auxiliam no seu entendimento. Como estão diretamente relacionadas à situação real que gera a série temporal, seja por efeito natural, humano ou ambos, essas propriedades podem informar também

possíveis causas e correlações entre dados distintos. As quatro propriedades são descritas a seguir (CHATFIELD, 2003):

- **Sazonalidade**: são variações nas observações dos dados que ocorrem em um certo período de tempo (ou temporadas). Uma série temporal contendo a temperatura de um país é um exemplo de série temporal com sazonalidade, com variação anual (o inverno tende a ser mais frio e verão mais quente).
- **Ciclos**: é um tipo de variação similar à sazonalidade, sendo geralmente causada por processos externos à natureza da série temporal. Diferente da sazonalidade, os ciclos podem ocorrer por tempos variados e em intervalos de tempo distintos. As variações de temperatura durante um dia são um exemplo de variação cíclica.
- **Tendência**: a tendência pode ser definida como uma mudança na média da série temporal ocorrendo por um longo período. Essa variação pode fazer com que a média da série temporal sofra um aumento (tendência positiva), uma diminuição (tendência negativa) ou nula (tendência horizontal).
- **Irregularidades**: são alterações aleatórias nos dados, sem nenhum padrão específico. Geralmente são variações de curta duração.

Uma série temporal pode ser entendida como uma sequência de observações, que assim como muitos outros processos reais, são aleatórias, e podem possuir propriedades probabilísticas. Dessa forma, uma das formas mais comuns de se entender e modelar uma série temporal é tentar identificar o processo estocástico que a pode ter gerado.

Segundo (SHUMWAY; STOFFER, 2000), um processo estocástico é um conjunto de variáveis aleatórias $\{X_t\}$ onde t, é um valor discreto que varia no intervalo dos números inteiros ($t = 0, \pm 1, \pm 2, ...$). Nesse sentido, uma forma de se descrever um processo estocástico é entender os momentos do processo, nesse caso, o primeiro e segundo momentos, os quais são a média (primeiro momento) e variância e autocovariância (segundo momento) (CHATFIELD, 2003). A autocovariância (γ_k), nesse caso, basicamente calcula a covariância entre valores atuais (t) e futuros ($t+k$) de um processo:

$$\gamma_k = \frac{\sum_{t=1}^{N-k}(X_t - \mu)(X_{t+k} - \mu)}{N-k} \tag{1}$$

onde k representa um intervalo de defasagem entre as observações, μ é a média e N é o número de observações do processo.

Uma das classes mais importantes de processos estocásticos são os processos estacionários. Há diferentes tipos de níveis de restrições para que um processo seja

considerado estacionário. Uma série temporal será estritamente estacionária se as distribuições conjuntas de $X_{t_1},...,X_{t_n}$ e $X_{t_1+\tau},...,X_{t_n+\tau}$ são iguais, ou seja, as distribuições não dependem do tempo t ou de qualquer deslocamento τ aplicado à série temporal. Essa definição, entretanto, possui um nível de restrição muito alto, sendo comum a utilização de uma definição com menos restrições, conhecida como estacionária de segunda ordem. Logo, uma série temporal será estacionária quando sua média for constante e sua função de autocovariância dependa somente da diferença entre intervalos distintos ($t_2 - t_1$).

A análise estatística das propriedades de uma série temporal permite a identificação da categoria de processo estocástico da qual ela faz parte. Assim, alguns dos processos mais comuns que auxiliam na classificação e modelagem de séries temporais são (CHATFIELD, 2003):

- **Puramente aleatório**: um processo que consiste em uma sequência de variáveis aleatórias $\{Z_t\}$ que são mutuamente independentes e identicamente distribuídas. Esse tipo de processo possui média e variância constantes e, como sua função de autocovariância não depende do tempo, será estacionário de segunda-ordem.

- **Passeio Aleatório**: supondo-se que $\{Z_t\}$ seja um processo puramente aleatório com média μ e variância σ_z^2. Um processo $\{X_t\}$ será um passeio aleatório se $X_t = X_{t-1} + Z_t$. Como a média e a variância são dependentes do tempo, o processo é não-estacionário.

- **Média móveis**: Considerando que $\{Z_t\}$ seja um processo aleatório com média 0 e variância σ_z^2, um processo $\{X_t\}$ será de médias móveis de ordem q, representado por MA(q) (do inglês, *Moving Average*), se:

  $$X_t = \beta_0 Z_t + \beta_1 Z_{t-1} + ... + \beta_q Z_{t-q}.$$

  onde β_i, com $i = 0,...,q$, são constantes. Esse processo é estacionário, pois sua função de autocovariância não depende do tempo t e a média é constante.

- **Auto-regressivo**: Sendo $\{Z_t\}$ um processo puramente aleatório com média 0 e variância σ_z^2, um processo $\{X_t\}$ será auto-regressivo de ordem p se:

  $$X_t = \alpha_1 X_{t-1} + ... + \alpha_p X_{t-p} + Z_t. \quad (2)$$

  Considerando o caso mais simples, onde o processo é de primeira ordem (p=1), tem-se que a Equação 2 pode ser simplificada para:

  $$X_t = \alpha_1 X_{t-1} + Z_t. \quad (3)$$

nesse caso, considerando que $-1 \leq \alpha \leq +1$, podemos rescrever a Equação 3 como um processo MA de ordem infinita:

$$X_t = Z_t + \alpha Z_{t-1} + \alpha^2 Z_{t-2} + \ldots$$

Dessa forma, a função de autocovariância para esse tipo de processo não depende do tempo t e o processo será estacionário (de segunda-ordem) quando $-1 \leq \alpha \leq +1$. Para o caso geral ($p > 1$), referenciamos o livro de Chatfield (2003). Assim como em um processo de médias móveis, um processo auto-regressivo é normalmente abreviado para AR(p) (*Auto-Regressive*, na sigla em Inglês).

Considerando os tipos de processos apresentados, é comum a definição de métodos capazes de modelar mais de um processo simultaneamente. Como foi visto, um processo Puramente Aleatório é usado na definição dos processos de Passeio Aleatório, Média Móvel e Auto-regressivos. Além disso, um dos tipos de modelos mais úteis para séries temporais é resultado da combinação de um processo Auto-regressivo (AR) e um processo de Médias Móveis (MA), resultando em um modelo conhecido como ARMA. Desta forma, um processo ARMA de ordem i(p,q), abreviado para ARMA(p,q), é definido por:

$$X_t = \alpha_1 X_{t-1} + ... + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + ... + \beta_q Z_{t-q}.$$

A vantagem da utilização deste tipo de modelo é que mais comum que uma série temporal estacionária seja descrita por um processo ARMA contendo poucos parâmetros do que por um processo MA ou AR sozinhos (CHATFIELD, 2003). Nesse sentido, na prática, a maioria das séries temporais não são estacionárias, dificultando a utilização de modelos ARMA. A partir dessas limitações, surgiu o modelo ARIMA, que será descrito com mais detalhes a seguir.

4.3.1. ARIMA

O ARIMA (*Auto-Regressive Integrated Moving Average*) é um modelo estatístico linear bastante conhecido na análise de séries temporais (ZHANG, 2003). Ele é uma generalização do modelo ARMA, sendo utilizado em séries temporais não-estacionárias, ou seja, uma série cujos dados oscilam sobre uma média que pode variar temporalmente.

Um modelo ARIMA é definido utilizando-se os parâmetros p, d e q, onde p representa o número de parâmetros auto-regressivos, d o número de diferenciações para tornar a série estacionária e q o número de parâmetros utilizado nas médias móveis. Um modelo ARIMA definido pelos parâmetros p, d e q é comumente representado por ARIMA(p, d, q).

Existem diversas metodologias que podem ser utilizadas para se encontrar os parâmetros p, d e q de um modelo ARIMA, sendo a de Box & Jenkins (BOX et al., 2015) a mais conhecida delas. Nessa metodologia, existem 3 etapas fundamentais (Identificação do modelo, Estimação dos parâmetros e Verificação do modelo), descritas a seguir.

4.3.1.1. Identificação do processo

Nesta fase, identifica-se o processo estocástico que gerou os dados. Para isso, algumas técnicas são utilizadas para se encontrar a estrutura do modelo, ou seja, os parâmetros p, d e q. O parâmetro d pode ser identificada por meio de uma sequência de diferenciações aplicadas à série temporal para torná-la estacionária. O número de vezes que a diferenciação precisa ser realizada define o valor de d.

Os parâmetros p e q podem ser identificados ao se analisar, respectivamente, as funções de autocorrelação parcial e de autocorrelação (respectivamente, PACF e ACF, nas siglas em inglês) utilizando-se a série que já passou pelo processo de diferenciação para se tornar estacionária. Cada uma dessas funções exibe a correlação entre a série temporal e um certo número de defasagens. Assim, elas representam como os coeficientes de correlação se comportam em diferentes números de defasagens.

Na Figura 4.3 tem-se uma série temporal, gerada por um processo ARIMA$(1,0,0)$ que representa a variação do PIB de um país entre 1900 e 2000. Nota-se que a série varia entre -1.5 e 2.5, mas não há tendências de crescimento ou decrescimento ao longo tempo, ou seja, a média não varia em função do tempo, logo, $d = 0$. Na Figura 4.4 têm-se os gráficos PACF e ACF da série temporal. Pode-se observar na Figura 4.4a que, após o número de defasagem 1, os coeficientes são muito próximos de zero 0, indicando que a série pode ser representada por um processo com $p = 1$. Além disso, a Figura 4.4b mostra que os valores dos coeficientes decrescem à medida que se aumenta o número de defasagens, indicando ser improvável que a série seja gerada por um processo de médias móveis, ou seja, $q = 0$.

4.3.1.2. Estimação dos Parâmetros

Nesta etapa deseja-se estimar os valores de β_i e α_i das Equações 4.3 e 2 considerando os valores de p e q encontrados na etapa anterior. Diversos procedimentos podem ser utilizados para isso, como funções de verossimilhança e de somas dos quadrados, estimação não-linear, análise dos processos específicos (caso seja verificado que a série é gerada por um processo auto-regressivo, de médias-móveis ou uma combinação de ambos) e estimação usando o Teorema de Bayes. Uma descrição mais detalhada de tais procedimentos pode ser encontrada em Box et al. (2015).

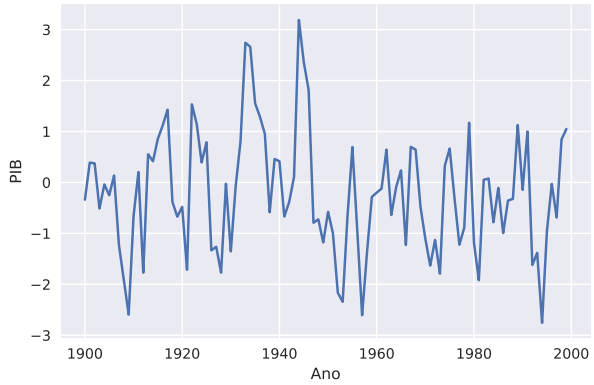
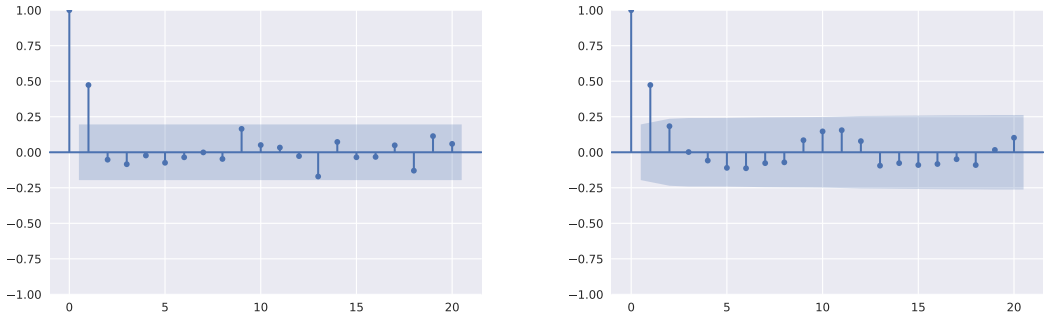


Figura 4.3: Exemplo de série temporal estacionária.



(a) PACF

(b) ACF

Figura 4.4: Gráficos PACF (a) e ACF (b) de um processo ARIMA$(1,0,0)$.

4.3.1.3. Verificação

Esta etapa consiste na realização de testes estatísticos para avaliar o desempenho do modelo ARIMA ajustado à série temporal. A ideia geral é identificar se o modelo definido está adequado para a série em questão e quais modificações no modelo precisam ser realizadas para garantir uma melhor adequação.

Um dos testes conhecidos é baseado na análise dos resíduos. A ideia é verificar se os resíduos do modelo ajustado apresentam algum tipo de dependência temporal, o que pode ser feito através dos gráficos de autocorrelação parcial anteriormente citados. Caso seja identificada qualquer tipo de dependência, o modelo não conseguiu capturar corretamente as dependências temporais da série original.

A metodologia de Box & Jenkins é largamente utilizada para definição e escolha

dos parâmetros de um modelo ARIMA. Entretanto, existem alternativas mais rápidas e livres de má interpretação das características da série temporal, que usam a metodologia de Box & Jenkins internamente. Por exemplo, em alguns casos, não é simples identificar se uma série é estacionária e, caso seja, quantas vezes a diferenciação precisa ser feita para torná-la estacionária. Dessa forma, algumas linguagens de programação, como o Python e R, possuem bibliotecas que facilitam as análises preliminares e possibilitam a identificação automática dos parâmetros do modelo.

É importante ressaltar, contudo, que o ARIMA tende a ser mais eficiente em bases de dados pequenas (MONTGOMERY; HINES, 1980). Ainda assim, pela facilidade em se encontrar os parâmetros p, d e q e por ser um modelo relativamente rápido de se treinar, o ARIMA mostra-se um bom modelo para se realizar as primeiras análises dos dados, indicando se um modelo não linear, por exemplo, uma Rede Neural Recorrente, precisa ser ou não utilizado, evitando o desperdício de tempo e de poder computacional. Embora o ARIMA seja comumente utilizado para a previsão de novas observações de séries temporais, também pode ser aplicado em geração de dados sintéticos. Para isso, ajusta-se um modelo a uma série temporal e simula-se uma nova série com base nos parâmetros do modelo que melhor se ajustam à série.

4.4. Análises de séries temporais

De modo geral, as séries temporais são utilizadas para três tipos de análises: Classificação, predição e modelagem da curva. Sendo esta última, a que permite entender a série para que realizar a geração de dados.

4.4.1. Predição

No problema da predição de séries temporais, tem-se um ou mais conjuntos de dados sequenciais e deseja-se realizar algum tipo de previsão com base nas informações obtidas dos dados. Nesse sentido, a previsão de séries temporais pode ser dividida em três métodos (CHATFIELD, 2003):

- **Subjetivo**: utiliza julgamentos subjetivos, intuição e qualquer informação relevante sobre os dados.
- **Univariado**: as previsões são feitas com base em valores passados de uma única série temporal
- **Multivariado**: nesse método, a previsão de uma variável depende, parcial ou totalmente, de valores de uma ou mais séries temporais adicionais.

Conhecimentos subjetivos sobre um determinado problema podem aumentar a eficiência dos métodos usados. De fato, na prática, é muito comum uma combinação dos

métodos, por exemplo, em que se faz uma previsão univariada ou multivariada com base em informações subjetivas que se tem das séries temporais (CHATFIELD, 2003).

Como um exemplo de previsão utilizando o método univariado, tem-se um gerente que deseja saber a variação de estoque de um certo produto durante os próximos 3 meses. Dado que ele tenha valores passados suficientes do estoque do produto (2 anos de dados, por exemplo), o gerente pode utilizar esses valores e estimar a variação do estoque ao longo dos meses desejados. Nesse sentido, os valores estimados podem ser obtidos com base no conhecimento subjetivo que o gerente possui do histórico de vendas. Existem dezenas de trabalhos na literatura baseados em previsão de séries univariadas. Nesse sentido, referenciamos o *survey* de (LIM; ZOHREN, 2021), o qual analisa trabalhos que utilizam técnicas baseadas em redes neurais para previsão de séries temporais univariadas.

É muito comum, contudo, que diversos fatores possam influenciar a variação de uma determinada série temporal. Isso pode fazer com que seja necessário que os métodos considerem mais de uma série temporal para realizar previsões. Como exemplo dessa situação, pode-se citar os diversos indicadores econômicos de um país, mensurados mensalmente, como: índices de preços no varejo, porcentagem de desemprego e índice da bolsa de valores. A partir desses indicadores, pode ser de grande interesse a construção de um modelo que descreva as relações presentes entre as variáveis e então utilizá-lo para realizar algum tipo de predição (e.g., variação da inflação). O leitor interessado em saber mais sobre estudos que realizam previsões multivariadas pode consultar o trabalho de (BEERAM; KUCHIBHOTLA, 2021).

4.4.2. Classificação

Diversos autores têm estudado formas de se classificar séries temporais com base em suas características. Diferente da classificação tradicional, entretanto, a ordem dos atributos possui um papel fundamental durante a classificação (BAGNALL et al., 2017). Na prática, o problema de classificação pode ser resumido em, dado um conjunto séries temporais não rotuladas, mapear cada uma delas a uma classe de um conjunto predefinido de classes (WEI; KEOGH, 2006).

Nesse caso, diferentes técnicas podem ser usadas para a realização da classificação, como as técnicas baseadas em características, baseadas em modelos e aprendizado de máquina. No primeiro caso, as características podem ser extraídas de vários domínios: tempo, frequência e ondas da série temporal. Entre as abordagens baseadas em modelos tem-se o ARIMA (vide Seção 4.3.1), modelos Gaussianos e Bayesianos (SYKACEK; ROBERTS, 2001; POVINELLI et al., 2004; KOTSIFAKOS; PAPAPETROU, 2014). Por fim, dentre as abordagens baseadas em aprendizado de máquina, podemos citar as árvores de classificação e Máquina de Vetores de Suporte (DOUZAL-CHOUAKRIA; AMBLARD, 2012; KAMPOURAKI; MANIS; NIKOU, 2008).

O trabalho de (MAHARAJ; ALONSO, 2014) é um exemplo de trabalho que demonstra a classificação de séries temporais sintéticas considerando o domínio temporal, mais especificamente, ondas. No trabalho os autores simularam sinais de eletrocardiogramas variando um parâmetro λ de forma que, quando $\lambda > 1$, o sinal passa a representar uma pessoa com sofreu um ataque cardíaco (*Acute Myocardial Infarction* - AMI, em inglês). Assim, é possível analisar a variação e a correlação das ondas para identificar quando ocorreu um ataque cardíaco, como foi proposto no trabalho. Esse tipo de análise possui aplicações práticas em situações reais, auxiliando na prevenção e tratamento de pessoa com problemas cardíacos.

4.4.3. Geração

O acesso a bases de dados é fundamental em diversas áreas de pesquisa, como agrupamento de dados, classificação, previsão e detecção de anomalias. Na prática, contudo, existem diversos desafios que limitam ou impedem a utilização eficiente de certos tipos de dados, por exemplo:

- **Tamanho da base de dados**: dependendo do objetivo de utilização dos dados, o tamanho (número de registros) pode impactar significativamente os resultados. No entanto, em algumas situações, pode ser inviável a obtenção de mais dados da fonte original. Por exemplo, em medicina, alguns tipos de doenças ocorrem em poucas pessoas no mundo todo. De forma similar, alguns processos industriais podem levar horas para serem concluídos, de forma que é inviável esperar até que uma grande quantidade de dados esteja disponível para ser utilizada.
- **Dados faltantes com erro**: em muitas situações, é comum ocorrerem erros no momento em que os dados são armazenados. Consequentemente, isso faz com que porções dos dados não possam ser usadas.
- **Privacidade**: A preocupação com a privacidade de dados pessoais tem se intensificado nos últimos anos. A criação da LGPD (BRASIL, 2018) no Brasil é resultado direto dessa preocupação. Nesse sentido, apesar de estarmos em uma era em que nossos dados são coletados diariamente por diversas empresas, a divulgação desses dados depende de muitos processos e pode não ocorrer, devido às questões de privacidade. Além disso, mesmo que as próprias empresas tratem da privacidade com algum tipo de método (e.g., anonimização e inserção de ruído), a utilidade dos dados pode ficar comprometida.

Nesse sentido, a geração de dados sintéticos torna-se uma alternativa para minimizar os impactos das limitações mencionadas anteriormente. É importante ressaltar que no contexto deste minicurso, o termo geração é sinônimo do termo simulação, já que o objetivo, de fato, é simular um cenário (conjunto de dados) real. Por exemplo, a partir de

dados sintéticos, conhecendo-se a distribuição dos dados, uma base de dados pequena pode ser aumentada, lacunas dos dados podem ser preenchidas e dados que simulem os dados reais podem ser disponibilizados publicamente. O maior interesse na geração de dados, principalmente nesse último caso, é permitir maior repetibilidade para pesquisas e experimentos, uma vez que cada base gerada é diferente. Apesar dos benefícios, o uso de dados sintéticos é desafiador, pois é necessário garantir que, de fato, os dados gerados representem a realidade do problema que se deseja tratar.

Contudo, como foco deste minicurso, este ainda é um campo de estudo pouco explorado na literatura quando se trata da geração de séries temporais a partir de um conjunto de dados reais. Além dos desafios que é gerar dados com características realistas, a dependência temporal dos dados deve ser considerada (LIN et al., 2020). Por exemplo, suponha que pesquisadores da área de medicina desejem estudar a evolução do quadro clínico de pessoas com uma determinada doença durante um ano. Por questões de privacidade, o hospital pode não divulgar os dados dos pacientes. Uma alternativa, então, é o hospital disponibilizar um modelo, obtido a partir dos dados reais, que gere dados sintéticos para a realização da pesquisa.

Em teoria, desde que os dados possam ser organizados como séries temporais, o mesmo princípio pode ser aplicado para dados de outros tipos de problemas. Na prática, há exemplos de geração de séries temporais de sensores (ALZANTOT; CHAKRABORTY; SRIVASTAVA, 2017), pacientes em hospital (ESTEBAN; HYLAND; RÄTSCH, 2017) e rede elétrica inteligente (ZHANG et al., 2018). Nesse sentido, as técnicas de geração vão desde modelos essencialmente estatísticos ou lineares (SINGH; RAY, 2021) a modelos baseados em aprendizado profundo (YOON; JARRETT; SCHAAR, 2019). Neste minicurso, apresentamos o último caso, os modelos conhecidos como Redes Generativas Adversárias, uma das técnicas mais recentes e mais bem sucedidas em geração de dados.

4.5. Introdução ao Aprendizado de Máquina

Inteligência artificial, aprendizado de máquina e aprendizado profundo são paradigmas que estão diretamente conectados. De modo geral, todos eles buscam representar o comportamento humano ao realizar alguma tarefa. Segundo Chollet (2021) a diferença entre tais conceitos é dada pela forma como cada algoritmo adquiri a base de conhecimento utilizada para realizar tais tarefas.

Algoritmos de inteligência artificial são desenvolvidos de maneira que as regras, ou conhecimento, são pré-definidos ao nível de modelo. Dessa maneira, dado um conjunto de dados e as regras pré-definidas, o algoritmo consegue inferir uma resposta. Ao contrário dos algoritmos de inteligência artificial, algoritmos de aprendizado de máquina baseiam-se na premissa do aprendizado através de exemplos. Nesse paradigma, as regras são aprendidas por padrões presentes nos dados e respostas previamente conhecidos

em um processo de treinamento e, com o conjunto de regras aprendidas, o algoritmo deve conseguir inferir conclusões ao processar novas amostras de dados pertencentes ao mesmo domínio. Por fim, aprendizado profundo é um sub-campo de aprendizado de máquina onde modelos são compostos por camadas de redes neurais artificiais, treinadas usando algoritmos de otimização em um processo iterativo.

A seguir, apresentamos com mais detalhes alguns dos conceitos presentes nos paradigmas citados. Primeiro, introduzimos os tipos de problemas de aprendizado de máquina. Após isso, serão apresentados os conceitos de redes neurais artificiais.

4.5.1. Tipos de Problema de Aprendizado de Máquina

Problemas de aprendizado de máquina podem ser classificados de acordo com as características dos dados a serem utilizados na etapa de treinamento. Dentre eles, os de Aprendizado Supervisionado e Aprendizado Não-Supervisionado englobam a maioria dos problemas existentes. Para ser caracterizado com um problema de Aprendizado Supervisionado, o conjunto de dados utilizado no processo de treinamento do algoritmo deve possuir, além dos dados de entrada, um dado de saída referente a cada amostra de entrada. Desse modo, o algoritmo busca aprender uma função de modo que possa mapear todos os dados de entrada $\mathbf{x}$ em um dado de saída y. Comumente nos referimos a esse tipo de dados como rotulados. Porém, muitos problemas não possuem dados rotulados, seja pela quantidade de dados presentes em um conjunto de dados, o que geraria um esforço elevado para rotular todos as amostras manualmente, ou pela falta de pessoas com conhecimento especializado que possa realizar tal trabalho. Problemas que envolvem tal tipo de dados (não rotulados), são conhecidos como problemas de Aprendizado Não-Supervisionado. De acordo com Goodfellow, Bengio e Courville (2016), algoritmos supervisionados buscam encontrar uma saída y dado uma amostra de entrada $\mathbf{x}$ ao estimar $P(y|\mathbf{x})$ e algoritmos Não-supervisionados buscam encontrar a distribuição de probabilidade $P(\mathbf{x})$ que gerou os dados.

Problemas de Aprendizado Supervisionado e Não-supervisionado ainda podem ser internamente agrupados de acordo com seu objetivo final. Dentre os problemas existentes, os mais comuns são: 1) Classificação, 2) Predição/Regressão e 3) Agrupamento. A seguir introduzimos cada um desses problemas.

4.5.1.1. Classificação

Problemas de classificação buscam encontrar uma função capaz de mapear uma entrada $\mathbf{x}$ em sua respectiva saída y, onde y representa um valor categórico, ou rótulo, relacionado a entrada do problema (GOODFELLOW; BENGIO; COURVILLE, 2016).

De modo prático, suponha que queremos classificar se um e-mail recebido é um

spam. Inicialmente, necessitamos de uma base de dados de treinamento $X = \{\mathbf{x}_0, \cdots, \mathbf{x}_N\}$, e seus respectivos rótulos $Y = \{y_0, \cdots, y_N\}$ informado se o e-mail em questão é considerado spam. O algoritmo deve então, de maneira supervisionada, aprender uma função capaz de mapear as amostras $\mathbf{x} \in X$ da base de dados de treinamento ao seu respectivo rótulo y. Posteriormente, a função aprendida deve ser capaz de realizar a mesma tarefa para amostras além do grupo de treinamento.

4.5.1.2. Regressão

Problemas de regressão possuem como objetivo prever um valor numérico. Ao contrário de problemas de classificação que querem prever um valor categórico pertencente a um conjunto de rótulos, problemas de regressão irão nos fornecer uma saída contida no conjunto de números reais. Dessa forma, em problemas de regressão, um modelo busca aprender uma função do tipo $f : \mathbb{R}^n \to \mathbb{R}$ (GOODFELLOW; BENGIO; COURVILLE, 2016).

Dado um conjunto n de características, uma regressão linear pode ser descrita pela Equação 4

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_b x_n + b, \tag{4}$$

onde x_i representa a i-ésima característica, w_i representa seu peso, treinável associado e b representa o *bias*.

A fim de caracterizar tal problema, podemos utilizar como exemplo de regressão a tarefa de estimar o número de pessoas que irão assistir um determinado video na plataforma YouTube, baseando-se na relação das características de vídeos similares com seus respectivos número de visualizações.

4.5.1.3. Agrupamento

Por fim, problemas de agrupamento buscam encontrar protótipos que possam sumarizar os dados de estudo de acordo com seus padrões e estruturas. O objetivo final do algoritmo é de agrupar os dados dentre um número finito de possíveis grupos.

Segundo Wunsch e Xu (2008), um algoritmo de agrupamento busca agrupar o conjunto de dados $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, onde $\mathbf{x}_i \in \mathbb{R}^n$, em k grupos $C = \{C_1, \cdots, \cdots, C_k\}$ com $(k \leq N)$, seguindo as seguintes restrições:

1. $C_i \neq \emptyset$; $i = 1, ..., k$;

2. $\bigcup_{i=1}^{k} C_i = X$;

3. $C_i \cap C_j = \emptyset$; $i, j = 1, ..., k$ e $i \neq j$.

O processo pela busca de partições está relacionado com a escolha de uma métrica de comparação e uma função objetivo a ser otimizada. Por exemplo, o algoritmo *K-means* busca agrupar os dados ao minimizar a variância da partição. Esse processo é feito ao minimizar os erros quadrados entre as amostras o centroide de um grupo C_i.

Algoritmos de agrupamento podem ser separados de acordo com sua abordagem de processamento dos dados. Algoritmos de partição buscam agrupar os dados em um conjunto finito de grupos pré-definidos, sem necessidade de estruturas mais complexas. Algoritmos hierárquicos não possuem um número de partições pré-definidas nas quais os dados devem ser agrupado. Essa abordagem utiliza de uma estrutura de árvore para encontrar o número ideal de partições para cada conjunto de dados. Algoritmos hierárquicos podem apresentar estratégia de agrupamento do tipo *top-down*, onde inicialmente existe um grupo com todos os dados que são recursivamente separados em partições menores, ou do tipo *bottom-up*, onde cada amostra dos dados representa uma partição que são posteriormente reagrupadas de acordo com suas características. Mais detalhes referentes aos tópicos abordados podem ser encontrados em Zaki e Jr (2020).

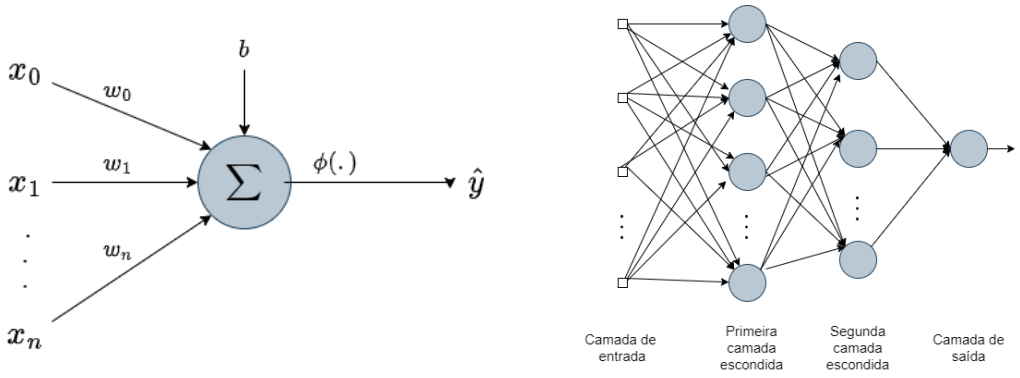
4.5.2. Redes Neurais Artificiais

Redes neurais artificiais (ANN, do inglês: *Artificial Neural Networks*) representam um conjunto de modelos que buscam representar computacionalmente o funcionamento biológico do cérebro humano. Uma ANN é um modelo computacional paralelo e distribuído criado a partir de unidades de computação simples com capacidade de armazenar conhecimento e utilizar o conhecimento adquirido (HAYKIN, 2009). A junção de unidades de computação simples, também conhecidas como neurônios artificiais, formam camadas que são responsáveis por processar e propagar as informações para outras camadas do modelo.

4.5.2.1. Redes Neurais do tipo *Feedforward*

Como apresentado anteriormente, redes neurais artificiais são modelos compostos por camadas de neurônios interconectados entre si. Quando a informação é propagada somente em um sentido do modelo, ou seja, somente para camadas subsequentes, temos uma Rede Neural do tipo *Feedforward* (FNN, do inglês: *Feedforward Neural Network*). O objetivo de uma FNN é aproximar alguma função $f(\mathbf{x}) = y$, de modo que a função possa mapear a entrada $\mathbf{x}$ em uma saída y (GOODFELLOW; BENGIO; COURVILLE, 2016). A unidade de computação simples de uma FNN é o *perceptron*.

Segundo Haykin (2009), o *perceptron* é um modelo computacional formado por uma função afim seguida por um operador não linear. Na Figura 4.5a, podemos observar uma representação gráfica de um *perceptron*. Matematicamente, o *perceptron* é definido conforme a Equação 5, a seguir:



(a) Perceptron

(b) Perceptron de Múltiplas Camadas

Figura 4.5: Representações de um perceptron, unidade básica de processamento de uma FNN (a) e uma MLP, composta de uma camada de entrada, duas camadas escondidas e a camada de saída com um perceptron (b). Adaptado de Haykin (2009).

$$f(\mathbf{x}) = \phi(\sum_{i=0}^{n} w_i x_i + b), \tag{5}$$

onde **x** representa os dados de entrada, **w** é o vetor de pesos, ou conhecimento, associados aos dados de entrada, b representa o viés e ϕ é a função de ativação do *perceptron*. A função de ativação é um operador diferenciável responsável por aplicar, na maioria dos casos, não-linearidade ao modelo (ZHANG et al., 2021). Algumas funções de ativação comumente utilizadas são listadas a seguir:

- Unidade Linear Retificada - ReLU

$$ReLU(x) = max(x, 0); \tag{6}$$

- Sigmóide

$$sigmoide(x) = \frac{1}{1+e^{-x}}; \tag{7}$$

- Tangente Hiperbólica

$$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}. \tag{8}$$

Apesar de ser descrita como uma unidade de computação simples, ao combinarmos células de perceptrons, podemos gerar estruturas complexas, conhecidas como Perceptron de Múltiplas Camadas (MLP, do inglês: *Multilayer Perceptron*). MLPs são redes neurais do tipo *feedforward* compostas por, no mínimo, três camadas. As camadas de entrada e saída e pelo menos uma camada oculta. Cada camada de uma MLP, com exceção da de entrada, é definida por um conjunto de *perceptrons*. Devido à natureza de uma FNN, cada célula de *perceptron* conecta-se com todas as células de perceptron da camada subsequente, realizando assim a propagação de informação. Na Figura 4.5b é ilustrado a arquitetura básica de uma MLP.

4.5.2.2. Treinamento de Redes Neurais

Como estabelecido anteriormente, algoritmos de aprendizado de máquina adquirem seu conhecimento através de um processo de treinamento. O processo baseia-se na aplicação de métodos de otimização a fim de encontrar os melhores parâmetros para um modelo. Em uma MLP, por exemplo, deseja-se obter o melhor conjunto de pesos $\mathbf{w}$ associados a cada *perceptron* presente nas camadas do modelo. Tal processo é realizado de forma iterativa, ao minimizar ou maximizar uma função objetivo $f(x)$. No contexto de redes neurais, a função objetivo é comumente chamada de função de perda.

Função de Perda: A função de perda tem por objetivo estimar o erro entre a saída $\hat{y}$ de uma rede neural com o valor y real associado aos dados de entrada da rede. Por exemplo, considere um problema de aprendizado supervisionado com objetivo de realizar uma tarefa de predição. Dado o conjunto de amostras $X = \{\mathbf{x}_0, \cdots, \mathbf{x}_N\}$, seus respectivos valores de saída $Y = \{y_0, \cdots, y_N\}$ e os valores de saída estimados pelo modelo $\hat{Y} = \{\hat{y_0}, \cdots, \hat{y_n}\}$, podemos calcular o erro L obtido através de uma função de perda como demonstrado na Equação 9.

$$L(Y, \hat{Y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{9}$$

A função de perda demonstrada na Equação 9 é conhecida como Erro Médio Quadrático (MSE, do inglês: *Mean Squared Error*). Outras funções de perda utilizadas em treinamento de redes neurais são:

- Raiz Quadrada do Erro Médio (RMSE)

$$L(Y, \hat{Y}) = \sqrt{\frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2}; \tag{10}$$

- Erro médio absoluto (MAE)

$$L(Y,\hat{Y}) = \frac{1}{N}\sum_{i=0}^{N}|y_i - \hat{y}_i|; \tag{11}$$

- Entropia Cruzada

$$L(Y,\hat{Y}) = -\sum_{i=0}^{N}(y_i \log \hat{y}_i). \tag{12}$$

A escolha da função de perda está intrinsecamente relacionada ao problema a ser resolvido. Funções como RMSE e MAE são utilizadas para problemas problemas de regressão, quando a saída do problema é um valor numérico. Outras funções de perda, como entropia cruzada, são aplicadas a problemas de classificação.

Otimização por Descida de Gradiente: O objetivo do treinamento de uma rede neural é de minimizar uma função de perda $y = f(x)$ (caso o objetivo seja de maximizar a função, podemos atingir o mesmo objetivo ao minimizar a função $y = -f(x)$). Segundo Goodfellow, Bengio e Courville (2016), ao derivarmos a função $f(x)$ temos uma pequena descida do valor de $f(x)$ no ponto x. Em outras palavras, podemos definir uma variação ε que aplicada a x irá resultar em uma variação correspondente em $f(x)$.

No contexto de treinamento de redes neurais, o objetivo é minimizar a função de perda ao otimizar os valores dos pesos do modelo. Seja $\nabla f(\mathbf{w})$ o gradiente de f. Cada iteração do algoritmo de descida de gradiente irá gerar um novo conjunto de pesos $w^{'}$, definido como:

$$w' = w - \varepsilon \nabla f(\mathbf{w}), \tag{13}$$

onde ε é definido como taxa de aprendizado. Desse modo, a otimização por descida de gradiente consiste em realizar pequenas mudanças do valor de $\mathbf{w}$ em direção oposta ao gradiente, de modo a obter pequenas melhoras nos resultados de saída do modelo.

Backpropagation: Uma MLP pode possuir várias camadas conectadas, ou seja, ela é composta por uma cadeia de funções. Nesse cenário, realizar o cálculo de forma analítica do gradiente pelo algoritmo de descida de gradiente torna-se inviável, uma vez que a operação é computacionalmente custosa e crescente com base no tamanho da rede. Para lidar com esse problema, Rumelhart, Hinton e Williams (1985) propuseram o algoritmo de retro-propagação para o cálculo de gradientes em uma ANN. Inicialmente, o algoritmo realiza os cálculos dos gradientes da rede a partir do erro, obtido através da função de perda, ao avaliar a saída real e a saída predita pela rede. Após essa etapa, é feita a atualização dos pesos da rede. Esse processo é realizado iterativamente até que a rede convirja para uma solução (RUMELHART; HINTON; WILLIAMS, 1985).

4.5.2.3. Redes Neurais Recorrentes

Apesar de FNNs apresentarem bons resultados ao processar dados tabulares, ela perde expressividade ao lidar com dados sequenciais, ou seja, que possuem algum tipo de dependência posicional e/ou temporal. Além disso, dados dessa natureza podem apresentar outras características as quais uma FNN não possui maneiras de processar, como dados de entrada de diferentes tamanhos. Imagine, por exemplo, um problema de processamento de linguagem natural. Sentenças de entrada, assim como frases ditas por pessoas e seu dia a dia, podem ser de tamanhos diferentes. Para lidar com esse tipo de problema, Rumelhart, Hinton e Williams (1986) propuseram uma arquitetura de ANNs, conhecida por Redes Neurais Recorrentes (RNN, do inglês: *Recurrent Neural Networks*).

RNNs são modelos de aprendizado profundo que buscam lidar com características sequenciais existentes em diversos problemas através de conexões recorrentes (ZHANG et al., 2021). Esse processo é realizado através de ciclos dentro da célula de computação simples de uma RNN, que permite o compartilhamento do mesmo conjunto de parâmetros através de uma sequência temporal. Na Figura 4.6, podemos observar a ideia por trás da célula de computação de uma RNN, onde $\mathbf{x}_t$ e h_t representam, respectivamente, a entrada e saída do modelo no tempo t.

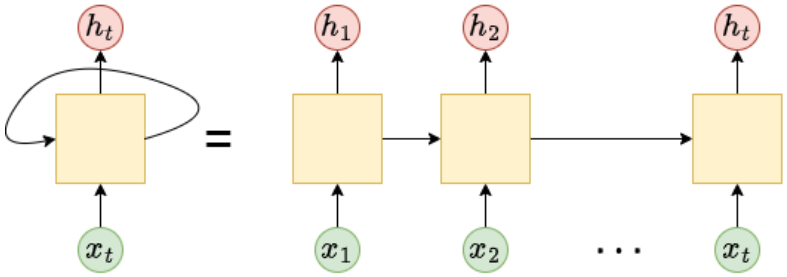


Figura 4.6: Ilustração da célula de computação de uma RNN. Através do ciclo interno presente na célula, é possível manter informações temporais dos dados. Adaptada de Zhang et al. (2021)

Dados utilizados em RNNs são sequências que possuem vetores $\mathbf{x}^{(t)}$ com um intervalo de tempo t variando de 1 a τ. Por exemplo, considere uma série temporal de temperaturas de uma cidade quaisquer aferidas diariamente durante 60 dias. Nesse caso, cada intervalo de tempo t é representado por um dia, ou seja, $\mathbf{x} = \{x^{(1)}, \cdots, x^{(60)}\}$. Agora imagine que queremos prever a temperatura do próximo intervalo de tempo, $x^{(61)}$. É natural supor que a temperatura dos dias anteriores possa influenciar o valor a ser predito. Dessa forma, uma RNN irá utilizar informações do passado, disponível pela natureza de sua arquitetura recorrente, para prever a temperatura do próximo dia. O principal

objetivo de uma RNN é poder utilizar o contexto passado para prever comportamentos futuros.

Apesar de apresentar bons resultados para exemplos em que a dependência temporal dos dados é curta, alguns modelos de RNN não conseguem obter bons resultados quando a dependência temporal dos dados é grande (GOODFELLOW; BENGIO; COURVILLE, 2016). Esse problema é referido como dependência de longo prazo, causado pelo desaparecimento ou explosão do gradiente durante o processo de treinamento de uma RNN (BENGIO; SIMARD; FRASCONI, 1994) e (HOCHREITER et al., 2001). Com o intuito de contornar o problema de dependência de longo prazo de outros modelos de RNN, Hochreiter e Schmidhuber (1997) propuseram um novo modelo conhecido como LSTM (do inglês: *Long Short-Term Memory*). A principal diferença entre a célula LSTM para a célula de uma RNN padrão é a adição de um mecanismo de memória com capacidade de manter informações de estados passados, evitando os problemas de desaparecimento e explosão do gradiente.

Devido aos resultados superiores obtidos por RNNs construídas com LSTM em detrimento de outros modelos, muitos problemas com dados de natureza sequencial passaram a utilizar soluções com aplicação de LSTM. Alguns exemplos de tais aplicações são análises de séries temporais (KARIM et al., 2019) e processamento de linguagem natural (GHOSH et al., 2016). Informações detalhadas da arquitetura de uma LSTM podem ser encotontradas em Hochreiter e Schmidhuber (1997).

4.6. Introdução às Redes Generativas Adversárias

As Redes Generativas Adversárias (GANs) são um *framework* utilizado para a otimização de modelos generativos baseados em aprendizado profundo. A ideia geral do funcionamento de uma GAN pode ser vista na Figura 4.7, que mostra um esquema de falsificação de pinturas. No esquema, o objetivo é treinar um pintor (G) para produzir imagens realistas de um pintor famoso, por exemplo, Pablo Picasso. Para que G tenha conhecimento sobre quão realista estão suas pinturas, o esquema possui também um especialista (D) sobre o estilo de pintura de Picasso. Durante um determinado número de vezes, G produz lotes de pinturas que serão avaliadas por D que, por sua vez, retorna uma nota média que represente o nível de fidelidade das pinturas falsas. Para cada lote, G utiliza um conjunto diferente de cores, garantindo que mesmo que as pinturas sejam idênticas às reais, ainda serão falsas. Assim, considerando que esse processo se repita por um número infinito de vezes, chegará o momento em que D não saberá diferenciar as pinturas falsas das reais, e objetivo do esquema será atingido.

De um ponto de vista técnico, considerando a arquitetura original, as GANs são definidas por um "jogo" de min-max (GOODFELLOW et al., 2014), onde G e D são duas redes neurais treinadas simultaneamente, como pode ser visto na Figura 4.8. A rede G, conhecida como um Gerador ($G(z;\theta_g)$), é um perceptron multicamadas que produz

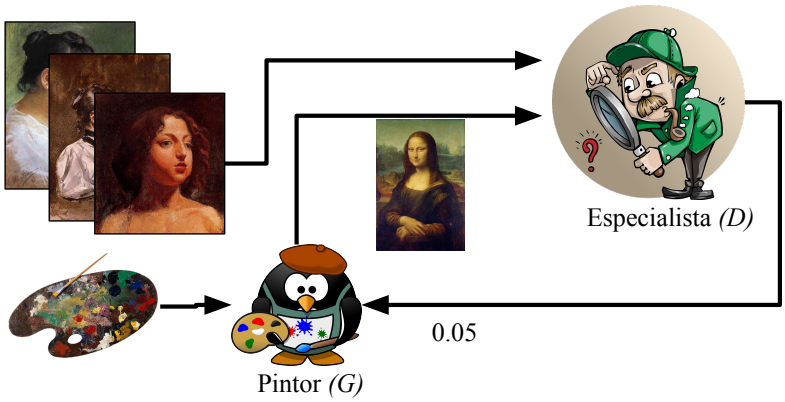


Figura 4.7: Visão geral do funcionamento de uma GAN.

dados falsos com base em entradas aleatórias $p_z(z)$ (as cores do exemplo anterior). D, por sua vez, é o Discriminador ($D(x;\theta_d)$), outro perceptron multicamadas que classifica a qualidade dos dados gerados, considerando uma base de dados reais x. Nas redes, z, θ_g e θ_d significam, respectivamente, o espaço latente dos dados (entradas aleatórias, como distribuições normais), os parâmetros para o perceptron que define G e os parâmetros para o perceptron que define D.

Assim como no exemplo anterior, o objetivo de G é conseguir enganar D. Nesse caso, gerando amostras de dados falsas cuja distribuição $p_g(x)$ se aproxime tanto da distribuição real $p_{data}(x)$, que D não consiga fazer distinção entre elas. A principal diferença entre o exemplo anterior, nesse sentido, é que no início do treinamento, D não pode ser um classificador com uma porcentagem de acurácia muito alta, pois caso isso ocorra, G nunca irá conseguir melhorar a qualidade dos dados gerados. Assim, ao mesmo tempo que D é treinado para distinguir entre os dados oriundos de G e os reais, G terá seus pesos atualizados considerando o *feedback* fornecido por D.

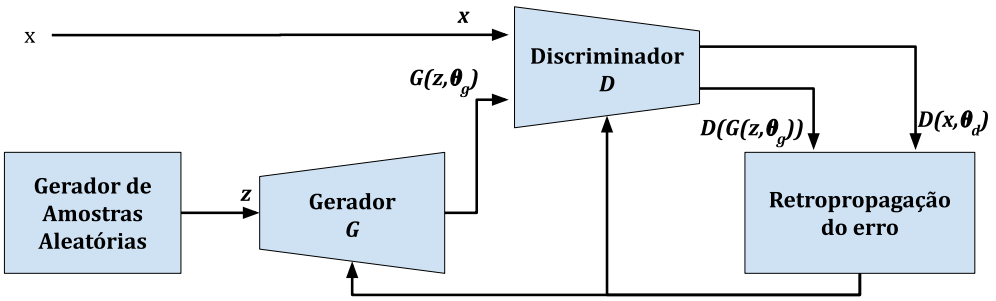


Figura 4.8: Funcionamento de uma GAN. Adaptado de Goodfellow et al. (2014).

Em teoria, como descrito em Goodfellow et al. (2014), ao utilizar a técnica

de min-max e com tempo e recursos computacionais suficientes, espera-se que D e G entrem em equilíbrio. Entretanto, as técnicas usadas para o treinamento de GANs normalmente se baseiam em gradientes descendentes que não são apropriadas para encontrar o equilíbrio do jogo min-max da GAN. Dessa forma, as primeiras GANs poderiam apresentar algumas instabilidades, por exemplo, dada a complexidade dos dados, D e G poderiam nunca chegar a um ponto de convergência. Nesse sentido, o trabalho de Salimans et al. (2016) apresentam uma série de técnicas que podem ser aplicadas para a melhorar o treinamento de GANs.

Figura 4.9: Rostos gerados por uma GAN treinada com a base de dados TFD (SUSSKIND; ANDERSON; HINTON, 2010). Figura de Goodfellow et al. (2014)

Um dos primeiros resultados da aplicação das GANs pode ser visto na Figura 4.9. Os rostos sintéticos foram gerados a partir da base original TFD (SUSSKIND; ANDERSON; HINTON, 2010), uma base de dados com faces de pessoas em baixa resolução e preto e branco. Trabalhos posteriores ao de (GOODFELLOW et al., 2014), utilizando algumas das técnicas propostas por Salimans et al. (2016) em conjunto com novas estruturas de GANs, mostraram a capacidade das GANs de gerarem imagens extremamente realistas, por exemplo, imagens sintéticas com base em diferentes categorias de imagens reais (Figura 4.10) (BROCK; DONAHUE; SIMONYAN, 2018) ou com base em imagens de celebridades (Figura 4.11).

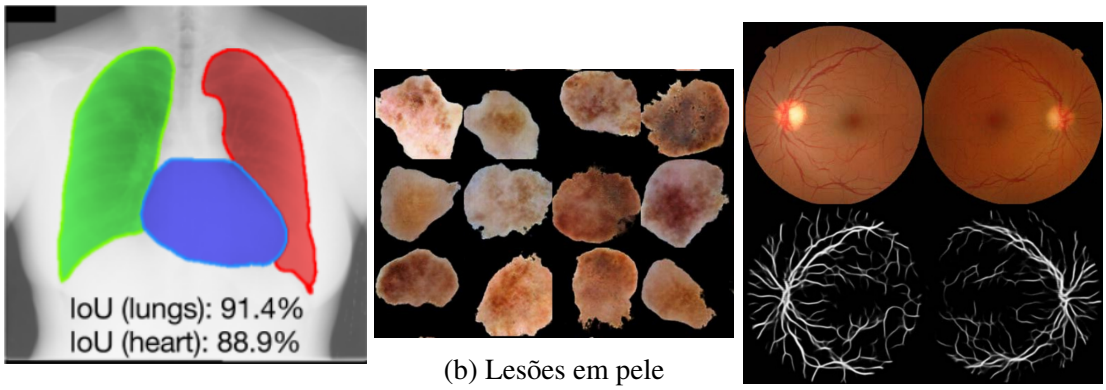
O bom desempenho no campo de visão computacional, principalmente em imagens, chamou a atenção de pesquisadores de diversas áreas. Em medicina, por exemplo, GANs foram utilizadas para na segmentação de órgãos com base em imagens de raios-X (DAI et al., 2018), geração de imagens de lesões em pele (YI; WALIA; BABYN, 2018) e obtenção do fundo de olho a partir de imagens de vasos sanguíneos (COSTA et al., 2017). Esses tipos de dados são fundamentais para o treinamento de modelos classificadores (por exemplo, para identificação de câncer de pele), e a utilização de GANs permite que, a partir de um conjunto pequeno de dados, tenha-se um conjunto maior de dados disponíveis. A Figura 4.12 mostra imagens dos exemplos mencionados. Ressalta-se que

Figura 4.10: Diferentes categorias de imagens geradas pela BigGAN (BROCK; DONAHUE; SIMONYAN, 2018).

esses são apenas alguns exemplos de aplicações de GANs considerando especificamente o campo de visão computacional. Mais estudos sobre a aplicação em medicina podem ser encontrados nos *survey* de Singh e Raza (2021). Outros estudos sobre GANs em visão computacional podem ser encontrados em Aggarwal, Mittal e Battineni (2021).

Figura 4.11: Rostos gerados com base em imagens de celebridades (KARRAS et al., 2017).

Apesar do seu bom desempenho, a utilização de GANs é desafiadora devido ao seu treinamento "adversário". A questão principal é que, geralmente, não é possível inferir a qualidade dos dados gerados da forma como normalmente é feito para avaliação de modelos de redes neurais, por exemplo, analisando a acurácia ou variação dos valores da função de perda. Do ponto de vista da arquitetura das GANs, um gerador G pode produzir dados que são, estatisticamente, indistinguíveis dos reais para um discriminador D, mas que não fazem sentido para o problema em questão. Isso torna necessário uma avaliação periódica da qualidade dos dados gerados pelos modelos treinados, que pode variar em função do domínio do problema. No caso de dados imagens de rostos, por exemplo, um conjunto de imagens pode ser gerado a cada x iterações a fim de se avaliar



(a) Segmentação de órgãos

(b) Lesões em pele

(c) De vasos para fundo de olho

Figura 4.12: Aplicações de GANs em diferentes áreas da medicina. As figuras foram retiradas dos seus respectivos artigos. (a) Exemplo de segmentação de órgãos (coração e pulmões) à partir de Raios-x (DAI et al., 2018); (b) Diferentes tipo de lesões em pele sinteticamente geradas (YI; WALIA; BABYN, 2018); (c) Obtenção do fundo de olho a partir dos vasos sanguíneos (COSTA et al., 2017).

se os rostos gerados são, de fato, humanos.

Outro desafio diz respeito à construção de um modelo que siga a arquitetura conceitual das GANs. De modo geral, D e G precisam ter capacidades similares, para que nenhuma delas se sobressaia durante o treinamento. Como mencionado anteriormente, D pode ser muito mais poderosa que G e G não consiga evoluir. Da mesma forma, G pode se tornar especialista em gerar um certo tipo de conjunto de dados, pois, pelo *feedback* recebido, sabe que D não consegue diferenciá-lo do real. Ou seja, a distribuição $p_g(x)$ que é aprendida por G foca em poucas variações da distribuição $p_{data}(x)$. Nos exemplos de imagens, ao invés de produzir imagens significativamente diferentes, o modelo acaba gerando imagens com as mesmas características. Esse problema é conhecido como *mode collapse* (YI; WALIA; BABYN, 2019).

Os bons resultados obtidos pelas GANs motivaram pesquisadores de outras áreas a investigar a aplicação das GANs em cenários diferentes do que foi proposto no trabalho original. Nesse sentido, Yu et al. (2017) mostram que a arquitetura original da GAN, com poucas alterações, consegue tratar de dados sequenciais, ainda que com suas limitações. Evidenciando o potencial das GANs para tratar de séries temporais, por exemplo. Trabalhos posteriores evidenciaram que a metodologia proposta nas GANs conseguem ser utilizadas com eficiência para a geração de séries temporais (YOON; JARRETT; SCHAAR, 2019). Contudo, ainda há diversos desafios inerentes à arquitetura das GANs que podem dificultar sua utilização em cenários de séries temporais ainda não

estudados. Alguns desses desafios, bem como oportunidades de pesquisa, serão descritas na próxima seção.

4.7. Desafios e Oportunidades de Pesquisa

Nessa seção, discutiremos alguns problemas e desafios de pesquisa acerca da utilização de GANs para geração de séries temporais. Leitores mais interessados podem se aprofundar nos detalhes em (LIN et al., 2020) e (ZHANG; MA; XIA, 2022).

4.7.1. Implementação, Treinamento e Avaliação

Um dos principais desafios da utilização de GANs, de uma maneira geral, é encontrado já durante sua implementação, principalmente em relação às capacidades dos modelos e função de perda utilizada. No primeiro caso, considerando a arquitetura original, o gerador e o discriminador precisam ter capacidades equivalentes. Caso um dos componentes seja muito mais poderoso que o outro, pode ocorrer o problema de *model collapse*, citado anteriormente. A função de perda é outro fator importante, pois é ela que dá o *feedback* que faz com que o gerador melhore a qualidade dos dados gerados.

A avaliação dos modelos é outro fator importante durante a utilização de GANs. Um dos problemas recorrentes na literatura são trabalhos que propõe modelos específicos para um campo de estudo, mas testados em apenas uma base de dados, como modelos para geração de música (ENGEL et al., 2019) e sinais de eletroencefalografia (HARTMANN KAY GREGOR E SCHIRRMEISTER; BALL, 2018). Tais modelos, apesar de eficientes, precisam ser avaliados em outras bases de dados. Além disso, em muitas situações a adequação dos dados sintéticos aos problemas reais necessitam da avaliação "manual" de um humano, o que pode tornar o processo de treinamento ainda mais trabalhoso. Modelos de GANs para geração de músicas (ENGEL et al., 2019) e dados de mobilidade (RIBEIRO et al., 2021b) são exemplos onde a avaliação manual é necessária para a definição de um bom modelo. Um grande desafio, nesse sentido, é encontrar formas automatizadas de se realizar esse tipo de avaliação.

Por fim, é importante ressaltar os problemas que envolvem modelos de aprendizado profundo, os componentes principais das GANs. No geral, o treinamento desses modelos envolve um processo que demanda tempo e complexo, em um longo exercício de tentativa e erro, já que diversos aspectos devem ser considerados, como hiperparâmetros, algoritmos de otimização (e seus hiperparâmetros), entre outros (RAMOS et al., 2021). Assim, a busca por um bom modelo de GAN também pode encontrar esses mesmos problemas e técnicas para buscas automáticas de modelos se fazem necessárias. Nesse sentido, já é possível encontrar essas técnicas considerando aplicações de visão computacional (GAO et al., 2020), mas a área de séries temporais ainda carece de estudos nessa direção.

4.7.2. Generalização

Uma das grandes características das GANs para visão computacional é que um mesmo modelo pode ser utilizado, sem exigir grandes modificações, para problemas similares. Por exemplo, a GAN original, o modelo mais simples já definido, obteve resultados satisfatórios em duas bases de dados semanticamente diferentes (rostos de pessoas e dígitos) mas conceitualmente as mesmas, pois se tratavam de imagens. Assim, como sendo o foco das GANs desde o início, nota-se que há arquiteturas e configurações específicas para problemas de visão computacional.

Para séries temporais, contudo, a obtenção de um modelo generalista encontra desafios adicionais. O principal fator é que uma série temporal não se limita a conter dados de um único tipo. Por exemplo, dados de mobilidade urbana podem ser representados por uma série temporal e conter dados numéricos (latitude, longitude, altitude, velocidade) e categóricos (tipo de local visitado, tipo de trajetórias) (RAO et al., 2020). Outro exemplo são dados de pacientes de um hospital, que podem conter informações numéricas (leituras feitas por aparelhos eletrônicos) ou categóricas (rótulos associados aos pacientes) (ESTEBAN; HYLAND; RÄTSCH, 2017). Nesse contexto, em situações práticas pode ser necessário que se encontre formas eficientes de se tratar esses dados simultaneamente, o que pode adicionar ainda mais complexidade à tarefa.

Além disso, na literatura é comum encontrar problemas similares sendo abordados por metodologias diferentes, onde cada metodologia possui suas vantagens e desvantagens que devem ser considerados durante sua aplicação. Por exemplo, em relação à geração de músicas, (MOGREN, 2016) propôs uma arquitetura similar à GAN clássica, onde o gerador e discriminador são RNNs. Por outro lado, (YU et al., 2017) propõe uma arquitetura baseada em aprendizagem por reforço, a qual é uma metodologia diferente do que é geralmente utilizada em GANs. Neste exemplo, caso deseje-se comparar os dois modelos, será necessário compreender bem as duas formas de aprendizado de máquina que podem ser bem distintas. Essas diferenças tornam-se maiores quando se trata de áreas diferentes, por exemplo, considerando a área de mobilidade urbana. Alguns modelos para geração de mobilidade combinam RNNs e Redes Convolucionais (JAUHRI et al., 2020).

4.7.3. Privacidade

Por definição, as GANs já fornecem privacidade aos dados sintéticos gerados, pois cada geração depende de entradas aleatórias. Entretanto, dado que as GANs conseguem capturar as características dos dados reais, diferentes questões de privacidade surgem a partir da aplicação de GANs em diferentes áreas. Em (LIN et al., 2020) os autores apresentam diversos aspectos sobre privacidade e GANs:

4.7.3.1. Proteção de segredos empresariais

Uma das grandes vantagens das GANs é permitir o compartilhamento de dados sintéticos ou modelos que gerem dados sintéticos. Segundo (LIN et al., 2020), uma das preocupações de quem detém os dados é o vazamento de informações sigilosas como recursos disponíveis e em uso na empresa, que estão geralmente embutidas em metadados. Contudo, para a realização de estudos, por exemplo, simulação de dados de uma aplicação, pode ser interessante que as correlações entre medições reais da aplicação e esses metadados sejam mantidas.

As alternativas para minimizar esse problema são mudar ou ofuscar a distribuição dos metadados. Nesse caso, (LIN et al., 2020) propõe uma arquitetura para realizar esse tipo de procedimento, em que medições reais e metadados são gerados por modelos independentes. Assim, após o treino nos dados reais, retreina-se o modelo dos metadados para geração de distribuições desejadas. A questão principal e maior interesse de pesquisa é como otimizar a distribuição dos atributos, já que o treinamento, nestes casos, funciona a partir de meios condicionais, isto é, a geração dos metadados é condicionada, probabilisticamente, por outras variáveis de entrada do modelo.

4.7.3.2. Proteção da privacidade de usuários

Com a criação da LGPD (BRASIL, 2018), a preocupação com a proteção e manutenção da privacidade dos dados de usuários intensificou-se em diversas áreas que envolvem aplicações que utilizam ou compartilham dados de usuários. Como que o objetivo principal de se utilizar GANs é reproduzir as características dos dados reais, tem-se um desafio enorme em equilibrar a geração de dados sintéticos e manutenção da privacidade. Um exemplo de como a privacidade de um usuário pode ser comprometida ocorre quando um modelo generativo memoriza as características de um indivíduo específico e vaza essas informações durante a geração dos dados (CARLINI et al., 2019).

Existem diversas formas de tratar destas questões e em muitas delas o principal desafio é como manter o equilíbrio entre utilidade e privacidade dos dados, isto é, quanto mais privacidade fornecida aos dados, menos úteis eles tendem a ser. Nesse contexto, uma das métricas mais utilizadas para avaliar a privacidade de usuários é a privacidade diferencial (DWORK, 2008). Essa métrica define que um modelo não deveria depender dos dados de um usuário específico de forma que, no contexto de modelos generativos, os dados sintéticos representem as características gerais dos dados. Na literatura, diversos estudos propuseram modelos de GANs para séries temporais considerando a privacidade diferencial (FRIGERIO et al., 2019), por adição de ruídos no processo de treinamento (principalmente durante a aplicação da descida do gradiente). Mais recentemente (QU et al., 2020) utilizaram esta técnica em dados de mobilidade, e

mostraram o que mencionamos anteriormente: quanto mais privacidade era fornecida aos dados, menos características dos dados reais era aprendida. Além disso, segundo (LIN et al., 2020), a utilização da privacidade diferencial ainda carece de estudos que avaliem os dados, principalmente em relação à qualidade dos dados.

Outra técnica utilizada para quantificar a privacidade é através dos ataques de inferência de membro (*membership inference attack*) (CHEN et al., 2020). O objetivo deste ataque é, considerando um conjunto de amostra de dados e um modelo de aprendizado de máquina, inferir se tal conjunto faz parte dos dados de treino. Nesse sentido, um modelo em que a privacidade diferencial tenha sido corretamente aplicado reduziria as chances de sucesso desse tipo de ataque. Além disso, em seu estudo utilizando a base de dados *Wikipedia Web Traffic* (GOOGLE, 2018), (LIN et al., 2020) verificaram que as chances de sucesso do ataque podem ser reduzidas quando se utiliza uma base de dados maior durante o treino.

4.8. Estudo de Caso - Hands-on

Para pôr em prática os conceitos aprendidos durante o capítulo, nesta seção apresentamos uma aplicação de geração de séries temporais utilizando GANs. A base de dados utilizadas representa o número de bicicletas alugadas em um serviço de locação em 7 cidades dos Estados Unidos e estão disponíveis um repositório no Github[1]. A Figura 4.13 apresenta a metodologia que será seguida nas próximas subseções. Primeiramente, faremos a análise e descrição dos dados (Subseção 4.8.1), seguido das modelagens estatísticas (Subseção 4.8.2) e com GANs (Subseção 4.8.3). Por fim, apresentamos como os modelos podem ser avaliados (Subseção 4.8.4).

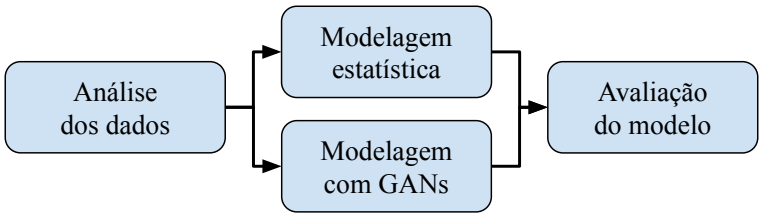


Figura 4.13: Metodologia para geração de séries temporais

4.8.1. Análise e Descrição dos dados

A base de dados possui 78888 registros de bicicletas alugadas, entre janeiro de 2011 e dezembro de 2019. Cada valor representa o número de bicicletas alugadas simultaneamente em intervalos de 1 hora durante cada dia entre 2011 e 2019. Como um dia possui 24 horas, a base de dados pode ser representada com um vetor com 3287 registros, onde

[1] https://github.com/ifribeiro/minicurso_webmedia22

cada registro possui 24 linhas (cada linha representa uma hora) e cada linha armazena uma variável (o número de bicicletas daquela hora). A Tabela 4.1 mostra as 5 primeiras linhas das três colunas da base de dados. Nosso interesse aqui é apenas na coluna *cnt*, mas as colunas *date* e *hour* auxiliam na visualização e entendimento dos dados.

Tabela 4.1: Primeiras 5 linhas da base de dados *Bikesharing*

date	**hour**	**cnt**
2011-01-01	0	16
2011-01-01	1	40
2011-01-01	2	32
2011-01-01	3	13
2011-01-01	4	1

Ao longo do minicurso, utilizaremos uma série de bibliotecas que implementam boa parte das funcionalidades de que necessitamos, principalmente as bibliotecas dos modelos ARIMA e do modelo de GAN utilizado e bibliotecas de visualizações de dados. Para facilitar a execução, algumas funções auxiliares de visualização foram implementadas e estão disponíveis no repositório. As versões das bibliotecas utilizadas são destacadas no git.

Assim, inicialmente carregamos a base de dados utilizando a biblioteca pandas e visualizamos todos os dados com o código a seguir:

```
bikes = pd.read_csv('datasets/bike_sharing_2011to2019.csv')
lista_datas = p.get_list_dates(data_size=bikes.shape[0],
year=2011, month=1, day=1)
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(lista_datas, bikes['cnt'])
ax.set_xlabel("Data")
ax.set_ylabel("Numero de bicicletas")
```

A Figura 4.14 mostra a série temporal representando o número de bicicletas alugadas nas 7 cidades de onde os dados foram coletados entre janeiro de 2011 e dezembro de 2019. Nota-se que há uma tendência de crescimento no número de bicicletas alugadas entre 2011 e início de 2019. Além disso, em cada ano os dados apresentam características similares (sazonais) em que há menos bicicletas alugadas no início do ano, com uma tendência de crescimento até por volta do meio do ano e, por fim, uma tendência de redução até o final do ano.

Apesar de fornecer informações importantes sobre os dados, a Figura 4.14 dificulta a visualização de informações que podem auxiliar no melhor entendimento dos dados, como pode ser visto na Figura 4.15. Na imagem à esquerda, tem-se o primeiro

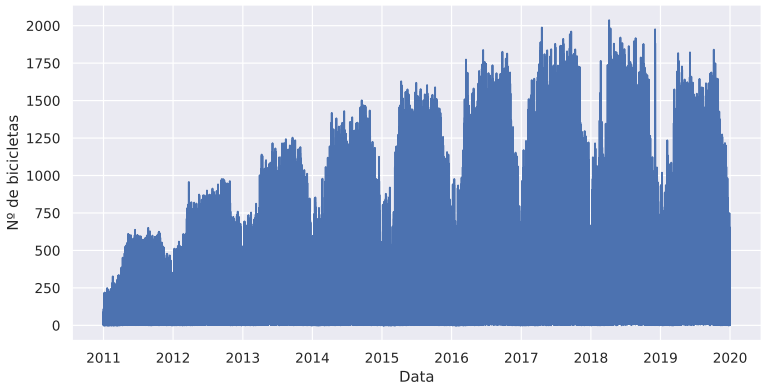


Figura 4.14: Número de bicicletas alugadas entre Janeiro de 2011 e Dezembro de 2019

mês dos dados. Nota-se que existe uma sazonalidade semanal dos dados, com uma sequência de valores altos e baixos. Além disso, podemos olhar especificamente para a primeira semana dos dados (imagem à direita), e notar que cada dia apresenta propriedades específicas. Por exemplo, em 01/01 e 02/01 (dois primeiros dias) os dados apresentam um pico por volta meio-dia. Se olharmos no calendário de 2011, esses dias correspondem ao sábado e domingo, respectivamente. Nos 5 dias subsequentes (dias úteis), como pode ser visto, os dados possuem um pico no início e no fim do dia, que variam pouco de um dia para o outro.

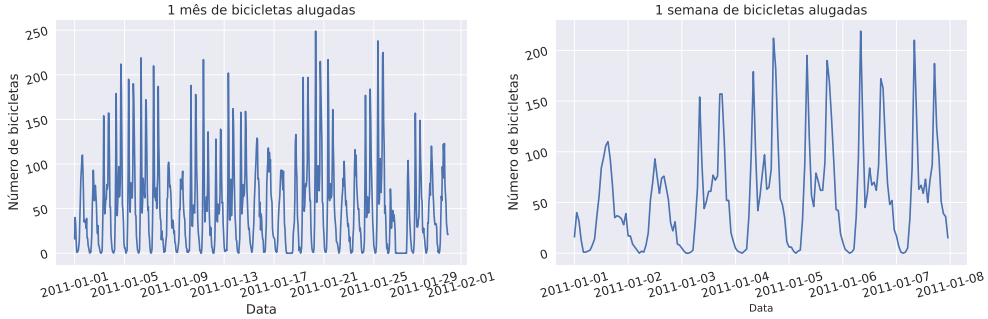


Figura 4.15: Um mês (figura à esquerda) e uma semana (figura à direita) de bicicletas alugadas.

Apesar de já entendermos melhor parte dos dados, apenas olhar para as duas figuras anteriores, não seria o suficiente para entendê-los como um todo. Nesse sentido, a Figura 4.16 apresenta a sumarização dos dados por semana (aqui denominados de soma por intervalo), em que computamos o número de bicicletas alugadas por hora do dia,

considerando os valores acumulados de cada dia da semana. A Figura evidencia o que foi discutido anteriormente, mostrando que há uma diferença clara entre os dias úteis e finais de semana. Além disso, conseguimos identificar que os dois picos dos dias úteis são, respectivamente, 9 da manhã e 6 da tarde.

```
real_data = bikes["cnt"].values.reshape(bikes.shape[0]//24, 24)
df_real = p.get_df(list_dates=lista_datas,
                   data=bikes[["cnt"]].values, timesteps=24)
lista_cnt = [p.get_count(df_real,w,24,
                   column="ts")["cnt_0"] for w in range(7)]
p.plot_sum_real(list_cnt_real=lista_cnt, list_dates=lista_datas)
plt.ticklabel_format(style="sci", axis="y", scilimits=(0,0))
plt.tight_layout()
```

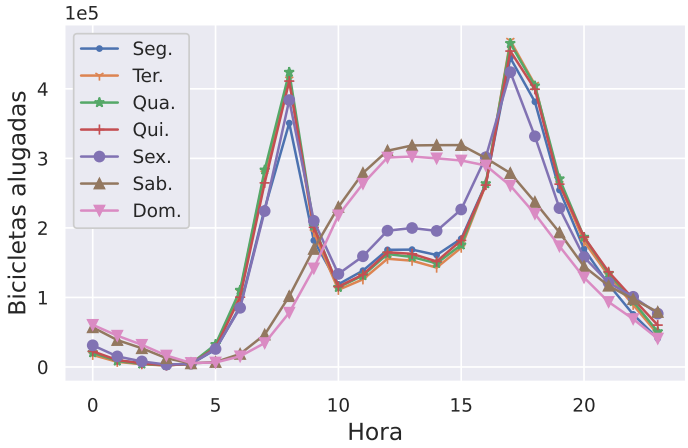


Figura 4.16: Somas por intervalo para a base de dados *Bikesharing*

As análises que fizemos até aqui serão de grande importância para entender como melhor definir os modelos que serão utilizados na geração dos dados sintéticos, bem como as causas de possíveis problemas que podemos encontrar durante as modelagens. Nas próximas subseções, apresentamos as técnicas de modelagem de dados.

4.8.2. Modelagem estatística

Para mostrar a diferença entre as GANs e modelos lineares, apresentamos aqui uma modelagem dos dados utilizando o ARIMA. Esta etapa é importante para verificarmos a necessidade ou não de se utilizar modelos mais complexos. Por exemplo, em outras situações, a utilização de um modelo baseado em aprendizado profundo pode ser desnecessário caso um modelo ARIMA já produza os resultados desejados.

Como mencionamos anteriormente, por ser um modelo baseado em regressão, o ARIMA faz algumas suposições sobre os dados, a principal é que eles sejam estacionários (a média não varia significativamente em função do tempo). Caso a série temporal não seja estacionária, precisamos aplicar diferenciações nos dados, de forma que a série resultante passe a ser estacionária. Há diversas formas de se verificar isso, mas neste minicurso utilizamos uma biblioteca do Python que já possui as ferramentas necessárias.

Devido à grande quantidade de dados, alguns códigos podem demorar alguns minutos para finalizar a execução, dependendo do poder computacional disponível. Assim, neste estudo de caso, utilizaremos apenas o primeiro ano dos dados reais a partir desta subseção. Ressaltamos que isso não altera a forma como os dados são executados, apenas o tempo de execução. Desta forma, primeiramente verificamos se os dados são estacionários com o código a seguir.

```
# realiza teste de estacionariedade
adf_test = pm.arima.stationarity.ADFTest(alpha=0.05)
treino = bikes["cnt"].iloc[:365*24].values
p_value, dif = adf_test.should_diff(treino)
print(p_value, dif)
```

No teste acima se a série temporal deve passar por alguma tipo de diferenciação. Nesse caso, o valor da variável `dif` pode ser *False*, caso não seja necessário a diferenciação e *True*, do contrário. Em nosso estudo de caso, o teste retorna que os dados são estacionários, logo nenhuma diferenciação precisa ser feita. Esse resultado já nos informa que o parâmetro d do ARIMA será igual a 0.

Em seguida, verificaremos quais os valores para os outros dois parâmetros, p e q, do modelo. Anteriormente apresentamos a metodologia de Box & Jenkins, que poderia ser utilizada para estimar tais parâmetros, mas utilizaremos uma biblioteca que automatiza esse processo com uma metodologia similar. Para isso, definiremos quais os valores possíveis de cada parâmetro para que a biblioteca verifique qual combinação deles se ajusta melhor aos dados. Assim, no trecho de código abaixo, definimos que p e q iniciam-se em 0 (`star_p=0`, `start_q=0`) e terminam em 3 (`max_p=3`, `max_q=3`). Além disso, o teste é executado 5 vezes (`maxiter=5`). Nesse caso, valores maiores para o parâmetro `maxiter` garantem maior confiabilidade no resultado obtido.

Desta forma, no código abaixo verificamos quais parâmetros geram um modelo que se ajusta melhor aos dados. É importante ressaltar que os valores (p, d e q) encontrados podem variar um pouco dependendo dos parâmetros passados à biblioteca e devido à natureza probabilística da sua implementação. Neste exemplo, a biblioteca retorna que os melhores parâmetros são: $p = 3$, $d = 0$ e $q = 1$. Com esses valores, ajustamos um modelo ARIMA aos dados, com as duas últimas linhas do código abaixo.

```
# verifica melhores parâmetros para o modelo (d=0, por padrão)
pm_auto = pm.auto_arima(treino, maxiter=10, start_p=0, max_p=3,
                        start_q=0, max_q=3, stationary=True)
# ajusta um modelo usando os parametros identificados
arima = ARIMA(treino, order=pm_auto.order)
res = arima.fit()
```

4.8.3. Definição e treinamento das GANs

Após a definição do modelo ARIMA, podemos definir o modelo GAN. Como mostramos na Subseção 4.8.1, os dados variam entre valores muito pequenos e muito grandes, o que pode prejudicar o processo de treinamento. Assim, antes da definição dos parâmetros é preciso padronizar os dados para que eles estejam em uma escala menor, por exemplo, no intervalo $[0,1]$. Isso pode ser feito com o código a seguir:

```
scaler = MinMaxScaler().fit(treino.reshape(-1,1))
treino_scaled = scaler.transform(treino.reshape(-1,1))
treino_s = treino_scaled.reshape(len(treino)//24, 24, 1)
```

Nesse caso, é importante ressaltar o tipo de entrada esperada pela GAN. Como utilizamos um modelo baseado em séries temporais, a entrada esperada precisa ter 3 dimensões (n,m,v), onde o n é o número de registros, m é o tamanho da sequência em cada registro e v e o número de variáveis. Em nosso caso, a variável `treino_s` do código anterior tem as dimensões $(365,24,1)$.

Nesse sentido, a GAN que utilizamos neste estudo de caso possui diversos parâmetros para serem configurados e alguns deles têm uma influência maior nos resultados e outros no treinamento. Discutiremos isso com mais detalhes na próxima subseção. O importante, nesta etapa, é entender a escolha de alguns parâmetros com base na análise feita na Subseção 4.8.1, especificamente: `seq_len`, `n_seq`, `hidden_dim`, `batch_size` e `learning_rate`. Nesse sentido, os dois primeiros parâmetros são obtidos diretamente das duas últimas dimensões dos dados. Assim, `seq_len=24` e `n_seq=1`.

Definimos também o parâmetro `hidden_dim=24`, referente ao número de unidades em cada camada escondida das redes neurais que compõe a GAN em que conseguimos controlar a capacidade de aprendizado do modelo. Como temos uma quantidade relativamente pequena de dados, esse valor não precisa ser muito grande. Em seguida, definimos `batch_size=28`, que é o tamanho do lote de dados utilizado durante cada iteração do treinamento que, em nosso problema, é a quantidade de dias utilizados para treinar o modelo. Geralmente o `batch_size` é um valor razoavelmente representativo dos dados, que possibilite que o modelo aprenda e garanta que o modelo

irá receber valores diferentes de dados à cada iteração. Como vimos, na etapa de análise, que os dados apresentam um padrão semanal, optamos por definir esse parâmetro com o valor 28, ou seja, o modelo recebe, aproximadamente, um mês de dados a cada etapa do treinamento. Por fim, definimos o parâmetro `learning_rate=5e-4 (0.00005)`, que controla a velocidade com que os modelos aprendem as características dos dados. Após a definição dos parâmetros, chamamos a função `ModelParameters` que salva os parâmetros na forma esperada pela biblioteca utilizada.

```
seq_len=24
n_seq = 1
hidden_dim=24
batch_size = 28
learning_rate = 5e-4
gamma=1
dim = 128
noise_dim = 32
gan_args = ModelParameters(batch_size=batch_size,
                           lr=learning_rate,
                           noise_dim=noise_dim,
                           layers_dim=dim)
```

Em seguida, podemos realizar o treinamento do modelo. No código do exemplo abaixo definimos que o modelo será treinado por 3000 vezes. É importante ressaltar que esse valor pode influenciar bastante o tempo de treinamento, principalmente se a máquina utilizada não possui recursos suficientes. Por exemplo, utilizando a versão grátis do Google Colab[2], o código abaixo demora cerca de 1h para finalizar. Contudo, caso seja de interesse do leitor, um modelo já treinado encontra-se salvo no repositório do minicurso. Após o treinamento, podemos salvar o modelo com o código da linha 3 para uso futuro.

```
df_sample = p.get_df(lista_datas[:365*24], treino.reshape(-1,1 ),
                     timesteps=24)
dict_weeks_real_arima = {wk:p.get_count(df_sample,wk,timestep=24,
                     column="ts")["cnt_0"] for wk in range(7)}

path_lista_fakes = glob.glob("datasets/generated/arima/*.npy")
dict_weeks_fakes = {i:p.get_list_wks(path_lista_fakes,
                     lista_datas[:365*24], 24, wk=i) for i in range(7)}
p.plot_compare_sum(dict_weeks_fakes, dict_weeks_real_arima,
                     bbox=(1.45, -0.1), figtitle="", scaler=None)
```

[2]https://colab.research.google.com/

4.8.4. Avaliação dos modelos

Com os modelos treinados, podemos gerar os dados sintéticos e verificar o desempenho dos modelos em realizar a tarefa desejada. Ressalta-se que um bom modelo terá dados que mantenham as principais características dos dados e em que cada base sintética gerada seja diferente. Assim, para realizar as avaliações, primeiro geramos os dados sintéticos com o modelo ARIMA e com o modelo GAN. Para garantir os resultados obtidos, é interessante que seja gerada uma quantidade razoável de dados sintéticos. No exemplo apresentado aqui, vamos gerar e salvar 10 bases para cada modelo:

```
# geração de 10 bases sintéticas com o modelo arima
for i in range(10):
    synth_arima = res.simulate(nsimulations=len(treino))
    np.save('datasets/generated/arima/arima_{}.npy'.format(i),
            synth_arima)
# Geração de 10 bases sintéticas usando GANs
for i in range(10):
    synth_data = synth.sample(len(treino_s))
    np.save("datasets/generated/timegan/timegan_{}.npy".format(i),
            synth_data)
```

Com as bases geradas, podemos realizar as avaliações. Nesse sentido, realizamos dois tipos de avaliação. Na quantitativa, verificamos os resíduos produzidos pelos modelos, que permite avaliar, ao mesmo tempo, se os dados sintéticos mantém as principais características dos modelos e se possuem variação. Na avaliação qualitativa nosso foco é verificar se os dados gerados são de fato similares aos reais e em quais pontos o modelo pode ser melhorado.

4.8.4.1. Avaliação quantitativa

A avaliação quantitativa é feita usando os erros residuais dos modelos treinados. Os erros residuais e de um modelo consistem da diferença entre o que se espera de saída de um modelo (representado por y) e o que o modelo retorna (representado por $\hat{y}$), ou seja, $e = y - \hat{y}$. A ideia de se utilizar os erros residuais é que, supondo que os dados sintéticos tenham capturado as características dos reais, seria possível utilizar um modelo de previsão para compará-los. Por exemplo, considere que um modelo tenha gerado um conjunto de dados sintéticos para uma série temporal com 200 registros. Se um modelo de previsão tenha sido treinado utilizando os 100 primeiros registros dos dados reais, a previsão resultante para os 100 registros restantes deve possuir características similares aos 100 últimos registros dos dados sintéticos.

Um modelo que tenha capturado com eficiência as propriedades dos dados reais, terá resíduos próximos de 0. Assim, caso o modelo anterior tivesse gerado a saída ($\hat{y}$)

exatamente como esperado (y), tanto a média quanto o desvio padrão seriam 0. Entretanto, o ideal é que os dados sintéticos gerados pelos modelos possuam variabilidade, ou seja, cada base é diferente entre si e, ao mesmo tempo, possuam as propriedades dos dados reais. Em outras palavras, os resíduos devem ser um ruído de média zero e variância constante.

Nesse sentido, a Figura 4.17 apresenta os erros residuais nas situações em que as séries temporais previstas pelo modelo são iguais (os dados sintéticos são iguais aos reais) e quando são levemente diferentes. No primeiro caso, os erros são todos iguais a 0, bem como a média e desvio padrão, contudo, isso significa que não temos dados sintéticos, e sim uma cópia dos dados reais. No segundo caso, ao calcularmos a média e desvio padrão obtemos valores diferentes de 0, mas notamos que os dados são similares.

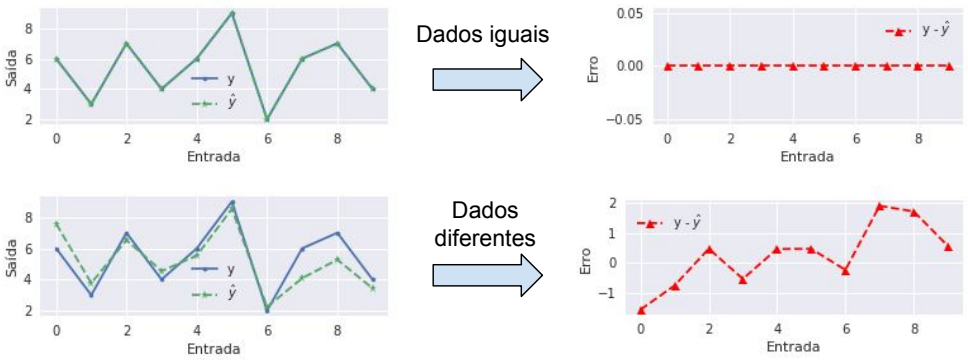


Figura 4.17: Exemplo de erros residuais quando os dados sintéticos e reais são iguais e diferentes

Em nosso exemplo, a obtenção dos resíduos precisa ser feita com os modelos adequados para cada situação. Assim, os erros residuais do ARIMA são obtidos diretamente do modelo ajustado aos dados e os erros residuais das GAN de um modelo de rede neural recorrente. Primeiramente definimos a rede neural recorrente que servirá como modelo de previsão para geração dos resíduos. Para isso, utilizamos a função `make_rnn_model` disponível no código `residuals.py` do repositório. No exemplo de código abaixo, nas linhas de 1 a 3 definimos e compilamos a rede neural. Nas linhas 4 e 5 transformamos os dados de treino no padrão esperado pela RNN. O treinamento, neste caso, consistem em prever, a partir de 24 registros (1 dia), quais os próximos 24 registros. Por fim, na linha 7 treinamos a rede nos dados reais, definindo, com a função `EarlyStopping`, que o treinamento será interrompido caso não haja melhoras significativas no valor retornado pela função de perda.

```
rnn_model = r.make_rnn_model(units=32, n_layers=2, net_type="lstm")
opt = Adam(learning_rate=5e-4)
rnn_model.compile(optimizer=opt, loss="mse")
X, Y = r.split_sequence(treino_s.flatten(),24)
X = X.reshape(X.shape[0],X.shape[1],1)
early_stop = EarlyStopping(monitor="loss")
hist=rnn_model.fit(X,Y, epochs=50, batch_size=28,
            callbacks=[early_stop])
```

Com o modelo treinado, podemos obter os resíduos. Primeiramente, listamos os nomes dos arquivos gerados pela GAN (linha 1). Em seguida, usando o modelo treinado, prevemos qual os dados obteríamos caso a entrada fosse os dados reais (linha 3) e, na linha 4, obtemos os resíduos de cada dado sintético com a função `get_residuals`. Nesse sentido, o mesmo processamento feito nas linhas 4, 5 do código anterior é feito para cada dado sintético. Assim, o que é esperado pelo modelo (`X_pred`) e o que o modelo gera podem ser utilizados para obtenção dos resíduos. A partir dos cálculos dos resíduos podemos calcular e armazenar as média e o desvio padrão, nas linhas de 6 a 9 do código abaixo.

```
synth_m3 = glob.glob("datasets/generated/timegan/3/*.npy")
X_pred = rnn_model.predict(X)
reid_m3 = [r.get_residuals(f, X_pred) for f in synth_m3]
dict_resid = {
    "mu": [ np.mean(resid_arima), np.mean(reid_m3)],
    "std": [np.std(resid_arima), np.std(reid_m3)]}
df_resid = pd.DataFrame(dict_resid, index=["ARIMA", "GAN"])
```

A Tabela 4.2 mostra os resultados dos resíduos de cada modelo. Nota-se que no geral a média de ambos os modelos são muito próximas de 0, entretanto o ARIMA tem um desvio padrão muito maior do que o modelo GAN. Isso é um indicador muito forte de que o ARIMA não conseguiu gerar dados com as características principais dos dados reais. Para confirmar isso, é importante que verifiquemos a análise qualitativa a seguir.

	Média	Desvio padrão
ARIMA	-0.009	64.608
GAN	-0.049	0.276025

Tabela 4.2: Média e desvio padrão residual dos modelos

4.8.4.2. Avaliação qualitativa

Na análise qualitativa, o que fazemos é basicamente gerar um resultado similar ao da Figura 4.16, que é a soma por intervalo, mas considerando as médias e desvios padrão das somas. Assim, primeiramente calculamos as somas, para cada dia da semana, de cada base sintética, e obtemos a média a desvio padrão. Por fim, comparamos esses valores com a soma por intervalo dos dados reais do dia da semana correspondente.

O código abaixo realiza esse cálculo para o modelo ARIMA. Primeiramente, obtemos as somas dos dados reais nas linhas 1 e 2. Em seguida, das linhas 4 a 6, calculamos as somas por intervalo dos dados sintéticos e geramos a Figura 4.18 que compara os resultados. Na Figura, a linha azul corresponde aos dados reais e as linhas de outras cores à média das somas por intervalo dos dados sintéticos. A parte sombreada em cada linha colorida corresponde ao desvio padrão dos dados sintéticos e indica como cada base de dados varia em cada dia da semana.

```
1 df_sample = p.get_df(lista_datas[:365*24], treino.reshape(-1,1),
2                      timesteps=24)
3 dict_weeks_real_arima = {wk:p.get_count(df_sample,wk,timestep=24,
4                 column="ts")["cnt_0"] for wk in range(7)}
5 path_lista_fakes = glob.glob("datasets/generated/arima/*.npy")
6 dict_weeks_fakes = {i:p.get_list_wks(path_lista_fakes,
7                 lista_datas[:365*24], 24, wk=i) for i in range(7)}
8 p.plot_compare_sum(dict_weeks_fakes, dict_weeks_real_arima,
9                 bbox=(1.45, -0.1), figtitle="", scaler=None)
```

A partir da figura anterior, fica evidente que o ARIMA não conseguiu capturar as características dos dados, como foi esperado a partir da análise quantitativa. Assim, nesse caso, o ideal é buscar por outros modelos que apresentem um desempenho melhor.

Para avaliar a GAN, realizamos um processo muito similar ao anterior. A diferença aqui é que utilizamos os dados reais que estão na escala [0, 1] para realização das análises, já que os dados gerados pela GAN também estão neste intervalo. Assim, a Figura 4.19 mostra as somas por intervalos das GANs geradas a partir das 10 bases sintéticas. Fica evidente que, diferente do ARIMA, a GAN conseguiu capturar com eficiência as principais características dos dados reais: os picos de bicicletas alugadas às 8 e às 18 horas, bem como o leve aumento de bicicletas alugadas entre 12 e 13 horas. Além disso, as áreas sombreadas mostram que os dados sintéticos possuem uma boa variabilidade. Nesse sentido, ressaltamos que a Figura está em um escala reduzida. No geral, a área sombreada indica que há uma diferença de cerca de 100 bicicletas entre cada base sintética.

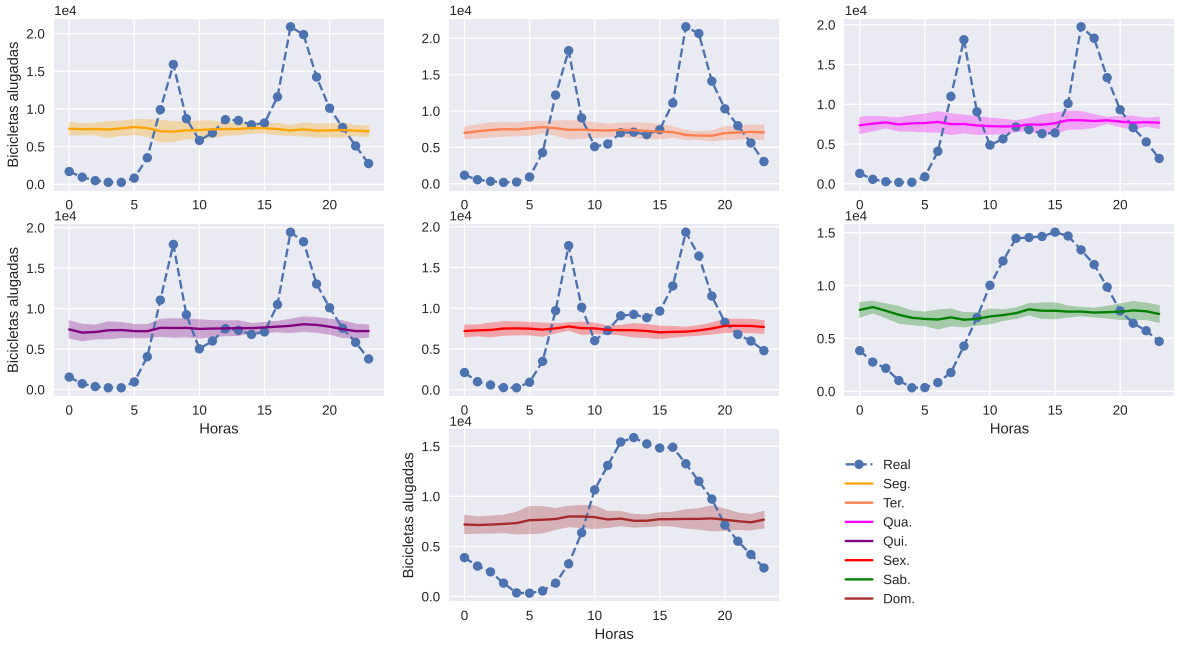


Figura 4.18: Soma por intervalo dos dados sintéticos gerados pelo ARIMA

```
df_sample = p.get_df(lista_datas[:365*24], treino_s,
                     timesteps=24)
dict_weeks_real = {wk:p.get_count(df_sample,wk,timestep=24,
                     column='ts')['cnt_0'] for wk in range(7)}
path_lista_fakes = glob.glob("datasets/generated/timegan/3/*.npy")
dict_weeks_fakes = {i:p.get_list_wks(path_lista_fakes,
                     lista_datas[:365*24], 24, wk=i) for i in range(7)}
p.plot_compare_sum(dict_weeks_fakes,dict_weeks_real,
                     bbox=(1.45, -0.1), figtitle="")
```

O maior desafio da GAN, contudo, é capturar as características dos finais de semana, já que em um ano existem mais dias úteis. Isso faz com que o modelo acabe extrapolando as características dos dias úteis (picos, principalmente) aos finais de semana. Uma alternativa para minimizar esse desafio seria utilizar o parâmetro para o `batch_size` que representa-se apenas alguns dias, ao invés do mês inteiro. Entretanto, isso iria impactar o resultado durante os dias úteis. Outra alternativa, seria fazer modelos específicos para os dias úteis para os finais de semana, com o risco de algumas dependências temporais serem perdidas e com uma complexidade maior para treinamento e

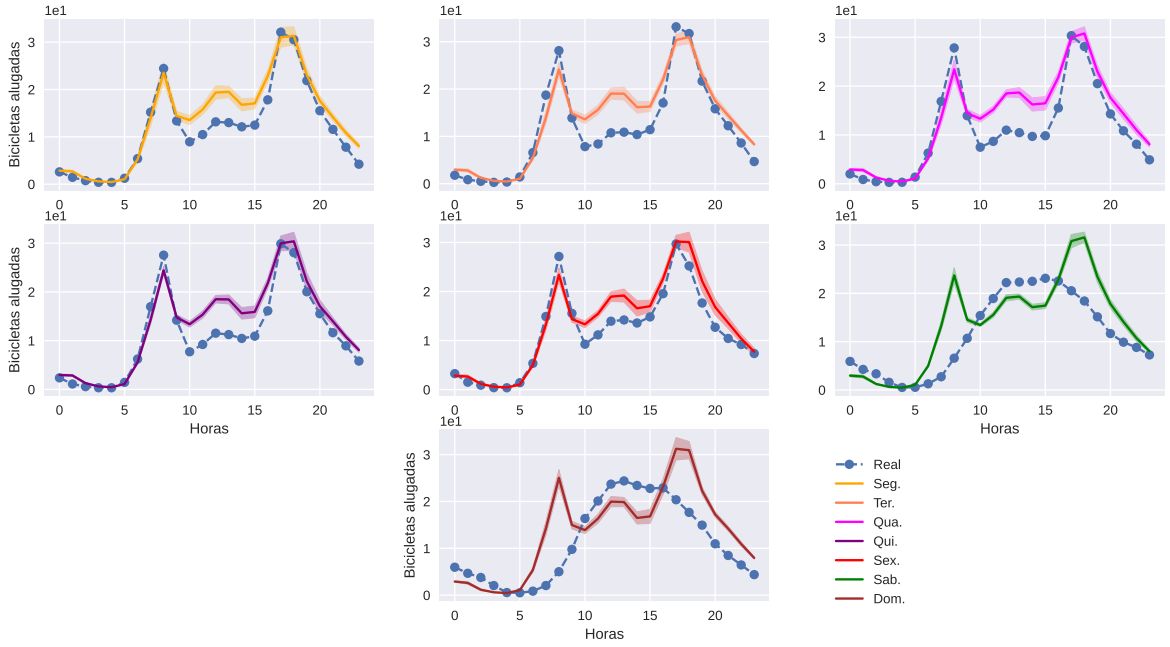


Figura 4.19: Soma por intervalo dos dados sintéticos gerados pela GAN.

avaliação, já que agora seriam necessários 2 modelos.

4.9. Considerações Finais

Este minicurso introduziu o conceito de GANs para a geração de séries temporais. Inicialmente apresentamos alguns exemplos de como as GANs têm sido usado com eficiência em aplicações de séries temporais. Em seguida, fazemos uma breve introdução aos conceitos de séries temporais, suas propriedades e técnicas estatísticas que auxiliam seu entendimento e modelagem. Apresentamos também os conceitos de aprendizado profundo, que são fundamentais ao entendimento da arquitetura das GANs. Introduzimos também o conceito de GANs, seu funcionamento e alguns exemplos de casos de uso da arquitetura original. Finalmente, a partir do que foi exposto ao longo do capítulo, apresentamos os principais desafios de pesquisa de GANs em séries temporais e finalizamos o capítulo um caso de uso prático dos conceitos aprendidos.

A arquitetura GAN é uma tecnologia relativamente nova, mas que usa conceitos bastante conhecidos de aprendizado profundo. Isso faz com que as GANs sejam uma técnica extremamente versátil em aprender e reproduzir as características principais de

diferentes conjuntos de dados, sendo um dos modelos generativos mais promissores, principalmente no campo de visão computacional.

O conceito de séries temporais é utilizado em vários aspectos da vida humana, desde músicas à área da saúde. Nesse contexto, a literatura evidencia que as GANs podem ser utilizadas com eficiência para tratar de aplicações envolvendo séries temporais. Da mesma forma, os desafios e limitações dos exemplos encontrados na literatura mostram que ainda há muito o que ser explorado em relação utilização de GANs para tratar de problemas envolvendo séries temporais.

Agradecimentos

O presente trabalho foi realizado no Laboratório de Pesquisa em Redes e Multimídia (LPRM) do Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Espírito Santo (UFES), com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) - Código de Financiamento 001, do CNPq, da Fundação de Amparo à Pesquisa do Espírito Santo (FAPES) e FAPESP (Grant #2020/05182-3).

Referências

AGGARWAL, A.; MITTAL, M.; BATTINENI, G. Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, Elsevier, v. 1, n. 1, p. 100004, 2021.

ALZANTOT, M.; CHAKRABORTY, S.; SRIVASTAVA, M. Sensegen: A deep learning architecture for synthetic sensor data generation. In: IEEE. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. [S.l.], 2017. p. 188–193.

AMIRIAN, J.; HAYET, J.-B.; PETTRÉ, J. Social ways: Learning multi-modal distributions of pedestrian trajectories with gans. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. [S.l.: s.n.], 2019. p. 0–0.

BAGNALL, A. et al. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, Springer, v. 31, n. 3, p. 606–660, 2017.

BEERAM, S. R.; KUCHIBHOTLA, S. Time series analysis on univariate and multivariate variables: a comprehensive survey. *Communication Software and Networks*, Springer, p. 119–126, 2021.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, IEEE, v. 5, n. 2, p. 157–166, 1994.

BOX, G. E. et al. *Time series analysis: forecasting and control*. [S.l.]: John Wiley & Sons, 2015.

BRASIL. *Lei Geral de Proteção de Dados (LGPD)*. 2018. <https://www2.camara.leg.br/legin/fed/lei/2018/lei-13709-14-agosto-2018-787077-publicacaooriginal-156212-pl.html>. Acessado em: 13 Ago. 2021.

BROCK, A.; DONAHUE, J.; SIMONYAN, K. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

BROCKWELL, P. J.; DAVIS, R. A. *Time series: theory and methods*. [S.l.]: Springer Science & Business Media, 2009.

CARLINI, N. et al. The secret sharer: Evaluating and testing unintended memorization in neural networks. In: *28th USENIX Security Symposium (USENIX Security 19)*. [S.l.: s.n.], 2019. p. 267–284.

CHATFIELD, C. *The analysis of time series: an introduction*. [S.l.]: Chapman and Hall/CRC, 2003.

CHEN, D. et al. Gan-leaks: A taxonomy of membership inference attacks against generative models. In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. [S.l.: s.n.], 2020. p. 343–362.

CHEN, J. et al. Generative dynamic link prediction. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, AIP Publishing LLC, v. 29, n. 12, p. 123111, 2019.

CHOLLET, F. *Deep learning with Python*. [S.l.]: Simon and Schuster, 2021.

COSTA, P. et al. End-to-end adversarial retinal image synthesis. *IEEE transactions on medical imaging*, IEEE, v. 37, n. 3, p. 781–791, 2017.

DAI, W. et al. Scan: Structure correcting adversarial network for organ segmentation in chest x-rays. In: *Deep learning in medical image analysis and multimodal learning for clinical decision support*. [S.l.]: Springer, 2018. p. 263–273.

DONAHUE, C.; MCAULEY, J.; PUCKETTE, M. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

DONG HAO-WEN E HSIAO, W.-Y.; YANG, L.-C. e; YANG, Y.-H. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2018. v. 32, n. 1.

DOUZAL-CHOUAKRIA, A.; AMBLARD, C. Classification trees for time series. *Pattern Recognition*, Elsevier, v. 45, n. 3, p. 1076–1091, 2012.

DWORK, C. Differential privacy: A survey of results. In: SPRINGER. *International conference on theory and applications of models of computation*. [S.l.], 2008. p. 1–19.

ENGEL, J. et al. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

ESTEBAN, C.; HYLAND, S. L.; RÄTSCH, G. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

FRIGERIO, L. et al. Differentially private generative adversarial networks for time series, continuous, and discrete open data. In: SPRINGER. *IFIP International Conference on ICT Systems Security and Privacy Protection*. [S.l.], 2019. p. 151–164.

GAO, C. et al. Adversarialnas: Adversarial neural architecture search for gans. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2020. p. 5680–5689.

GHOSH, S. et al. Contextual lstm (clstm) models for large scale nlp tasks. *arXiv preprint arXiv:1602.06291*, 2016.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

GOODFELLOW, I. et al. Generative adversarial nets. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 2672–2680.

GOOGLE. *Web Traffic Timeseries Forecasting*. 2018. <https://www.kaggle.com/c/web-traffic-time-series-forecasting>.

HARTMANN KAY GREGOR E SCHIRRMEISTER, R. T.; BALL, T. Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals. *arXiv preprint arXiv:1806.01875*, 2018.

HAYKIN, S. *Neural Networks and Learning Machines*. [S.l.]: Prentice Hall, 2009. (Neural networks and learning machines, v. 10). ISBN 9780131471399.

HOCHREITER, S. et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. [S.l.]: A field guide to dynamical recurrent neural networks. IEEE Press In, 2001.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HUANG, J.; KURNIAWAN, E.; SUN, S. Cellular kpi anomaly detection with gan and time series decomposition. In: IEEE. *ICC 2022-IEEE International Conference on Communications*. [S.l.], 2022. p. 4074–4079.

JAUHRI, A. et al. Generating realistic ride-hailing datasets using gans. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, ACM New York, NY, USA, v. 6, n. 3, p. 1–14, 2020.

KADRI, F. et al. Towards accurate prediction of patient length of stay at emergency department: a gan-driven deep learning framework. *Journal of Ambient Intelligence and Humanized Computing*, Springer, p. 1–15, 2022.

KAMPOURAKI, A.; MANIS, G.; NIKOU, C. Heartbeat time series classification with support vector machines. *IEEE transactions on information technology in biomedicine*, IEEE, v. 13, n. 4, p. 512–518, 2008.

KARIM, F. et al. Multivariate lstm-fcns for time series classification. *Neural Networks*, Elsevier, v. 116, p. 237–245, 2019.

KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

KOTSIFAKOS, A.; PAPAPETROU, P. Model-based time series classification. In: SPRINGER. *International Symposium on Intelligent Data Analysis*. [S.l.], 2014. p. 179–191.

LIM, B.; ZOHREN, S. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, The Royal Society Publishing, v. 379, n. 2194, p. 20200209, 2021.

LIN, Z. et al. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In: *Proceedings of the ACM Internet Measurement Conference*. [S.l.: s.n.], 2020. p. 464–483.

LUO, Y. et al. Multivariate time series imputation with generative adversarial networks. *Advances in neural information processing systems*, v. 31, 2018.

MAHARAJ, E. A.; ALONSO, A. M. Discriminant analysis of multivariate time series: Application to diagnosis based on ecg signals. *Computational Statistics & Data Analysis*, Elsevier, v. 70, p. 67–87, 2014.

Malandrino, F.; Chiasserini, C.; Kirkpatrick, S. Cellular network traces towards 5g: Usage, analysis and generation. *IEEE Transactions on Mobile Computing*, v. 17, n. 3, p. 529–542, 2018.

MOGREN, O. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

MONTGOMERY, D. C.; HINES, W. W. *Probability and statistics in engineering and management science*. [S.l.]: John Wiley & Sons, 1980.

OH, E. et al. Sting: Self-attention based time-series imputation networks using gan. In: IEEE. *2021 IEEE International Conference on Data Mining (ICDM)*. [S.l.], 2021. p. 1264–1269.

POVINELLI, R. J. et al. Time series classification using gaussian mixture models of reconstructed phase spaces. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 16, n. 6, p. 779–783, 2004.

QU, Y. et al. Gan-driven personalized spatial-temporal private data sharing in cyber-physical social systems. *IEEE Transactions on Network Science and Engineering*, IEEE, 2020.

RAMOS, H. S. et al. Aprendizado federado aplicado à internet das coisas. *Sociedade Brasileira de Computação*, 2021.

RAO, J. et al. Lstm-trajgan: A deep learning approach to trajectory privacy protection. *arXiv preprint arXiv:2006.10521*, 2020.

RIBEIRO, I. et al. Mobility and community detection based on topics of interest. In: *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*. [S.l.]: IEEE, 2021. p. 1–6.

RIBEIRO, I. et al. Uma abordagem para geração de séries temporais de mobilidade urbana baseada em aprendizado profundo. In: *Anais do V Workshop de Computação Urbana*. Porto Alegre, RS, Brasil: SBC, 2021. p. 251–264. ISSN 2595-2706.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. *Learning internal representations by error propagation*. [S.l.], 1985.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

SALIMANS, T. et al. Improved techniques for training gans. *Advances in neural information processing systems*, v. 29, p. 2234–2242, 2016.

SCOTT, J. et al. *CRAWDAD dataset cambridge/haggle (v. 2009-05-29)*. 2009. Downloaded from https://crawdad.org/cambridge/haggle/20090529.

SHUMWAY, R. H.; STOFFER, D. S. *Time series analysis and its applications*. [S.l.]: Springer, 2000. v. 3.

SINGH, H.; RAY, M. R. Synthetic stream flow generation of river gomti using arima model. In: *Advances in Civil Engineering and Infrastructural Development*. [S.l.]: Springer, 2021. p. 255–263.

SINGH, N. K.; RAZA, K. Medical image generation using generative adversarial networks: A review. *Health informatics: A computational perspective in healthcare*, Springer, p. 77–96, 2021.

SUSSKIND, J.; ANDERSON, A.; HINTON, G. E. *The Toronto face dataset*. [S.l.], 2010.

SYKACEK, P.; ROBERTS, S. J. Bayesian time series classification. *Advances in Neural Information Processing Systems*, v. 14, 2001.

WANG, Z. et al. Data-augmentation-based cellular traffic prediction in edge-computing-enabled smart city. *IEEE Transactions on Industrial Informatics*, IEEE, v. 17, n. 6, p. 4179–4187, 2020.

WEI, L.; KEOGH, E. Semi-supervised time series classification. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2006. p. 748–753.

WUNSCH, D.; XU, R. *Clustering*. [S.l.]: John Wiley & Sons, 2008.

YI, X.; WALIA, E.; BABYN, P. Unsupervised and semi-supervised learning with categorical generative adversarial networks assisted by wasserstein distance for dermoscopy image classification. *arXiv preprint arXiv:1804.03700*, 2018.

YI, X.; WALIA, E.; BABYN, P. Generative adversarial network in medical imaging: A review. *Medical image analysis*, Elsevier, v. 58, p. 101552, 2019.

YOON, J.; JARRETT, D.; SCHAAR, M. van der. Time-series generative adversarial networks. In: WALLACH, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf>.

YU, L. et al. Seqgan: Sequence generative adversarial nets with policy gradient. In: *Proceedings of the AAAI conference on artificial intelligence*. [S.l.: s.n.], 2017. v. 31, n. 1.

ZAKI, M. J.; JR, W. M. *Data mining and machine learning: Fundamental concepts and algorithms*. [S.l.]: Cambridge University Press, 2020.

ZHANG, A. et al. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

ZHANG, C. et al. Generative adversarial network for synthetic time series data generation in smart grids. In: IEEE. *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. [S.l.], 2018. p. 1–6.

ZHANG, D.; MA, M.; XIA, L. A comprehensive review on gans for time-series signals. *Neural Computing and Applications*, Springer, p. 1–21, 2022.

ZHANG, G. P. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, Elsevier, v. 50, p. 159–175, 2003.

ZHANG, L. Stggan: Spatial-temporal graph generation. In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. [S.l.: s.n.], 2019. p. 608–609.

ZHANG, Y. et al. Gcgan: Generative adversarial nets with graph cnn for network-scale traffic prediction. In: IEEE. *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2019. p. 1–8.

ZHANG, Y. et al. Trafficgan: Network-scale deep traffic prediction with generative adversarial nets. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 22, n. 1, p. 219–230, 2019.

ZHENG, Y. et al. Mining interesting locations and travel sequences from gps trajectories. In: ACM. *World wide web*. [S.l.], 2009. p. 791–800.