Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

# Estudo de classes de usuários através de Sistemas de Recomendação usando Redes Neurais Geradoras Adversárias

Victor André Batistella

Monografia Final

mac 0499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Roberto Hirata Junior

São Paulo
30 de Dezembro de 2021

# Agradecimentos

À minha família e amigos que me apoiaram durante toda a graduação e tornaram isso possível.

À minha namorada Larissa que me acompanhou durante todo o processo e me ofereceu apoio.

Ao supervisor Roberto Hirata Junior pela paciência e suporte durante este projeto.

# Resumo

Victor André Batistella. **Estudo de classes de usuários através de Sistemas de Recomendação usando Redes Neurais Geradoras Adversárias**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Redes Geradoras Adversariais (*Generative Adversarial Networks* ou GAN) têm ganhado cada vez mais espaço na geração de dados que se assemelham a uma distribuição alvo. O exemplo mais conhecido de seu uso é a geração de faces de pessoas que não existem, mas com aparência verossímil.

Este trabalho visa estudar o comportamento de classes de usuários em sistemas de recomendação e, a partir de suas escolhas nessas recomendações, simular comportamentos semelhantes utilizando a tecnologia de RecGAN - uma arquitetura de redes neurais que leva em consideração dados temporais no problema de recomendações.

**Palavras-chave:** GAN. Rede Geradora Adversária. Recomendação. Rede Neural.

# Abstract

Victor André Batistella. **Study of user classes through Recommender Systems using Generative Adversarial Neural Networks**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

Generative Adversarial Networks (GAN) have been getting a lot of attention by generating data that is similar to a target distribution. The most famous example of its usage is generating pictures of non-existing humans with believable appearance.

This work studies the behaviour of user classes in recommendation systems and, from the results of their actions based on the given recommendations, simulate similar behaviour with RecGAN technology - a neural network architecture that considers temporal data in the recommendation problem.

**Keywords:** GAN. Generative Adversarial Network. Recommendation. Neural Network.

# Lista de Abreviaturas

| | |
|---|---|
| GAN | Rede Neural Geradora Adversária |
| RS | Sistema de Recomendação |
| ML | *Machine Learning* (Aprendizado de Máquina) |
| DNN | *Deep Neural Networks* (Redes Neurais Profundas) |
| NN | *Neural Network* (Rede Neural) |
| RNN | *Recurrent Neural Network* (Redes Neurais Recorrentes) |
| CF | *Collaborative Filtering* (Filtragem Colaborativa ) |

# Lista de Figuras

2.1 Uma rede neural simples. . . . . 6
2.2 Gradiente Descendente. A função é a *loss* e as setas indicam a direção do gradiente. , acesso em: 10/11/2022 . . . . . 6
2.3 Exemplo de *overfitting* e *underfitting* - que é quando o modelo não aprende. 7
2.4 Exemplos do uso de GAN para gerar faces e animais. . . . . 8
2.5 Diagrama que ilustra a estrutura GAN. . . . . 8
2.6 Ilustração de filtragem colaborativa. . . . . 9
2.7 Uma arquitetura de RNN simples. . . . . 10
2.8 LSTM e GRU. . . . . 11
2.9 Rede neural e rede neural profunda. . . . . 11

3.1 Marcos em sistemas de recomendação em CF . . . . . 15
3.2 Na primeira coluna é acusada a classe real, porém adicionando a imagem da coluna do meio, todas acusam *"ostrich, Struthio or camelus"* . . . . . 16

# Sumário

**1 Introdução** **1**
1.1 Sistemas de Recomendação . . . . . 1
1.2 Comportamento do Usuário . . . . . 1
1.3 Predição do Comportamento do Usuário . . . . . 2
1.4 Proposta . . . . . 2

**2 Fundamentos** **5**
2.1 Redes Neurais . . . . . 5
2.2 Introdução à GAN . . . . . 7
2.3 Sistemas de Recomendação . . . . . 9
2.4 Redes Neurais Recorrentes . . . . . 10
2.5 Redes Neurais Profundas . . . . . 11
2.6 Normalized Discounted Cumulative Gain . . . . . 12
2.7 Problema de Aprendizado . . . . . 12
2.8 Ranqueamento . . . . . 13
2.9 RecGAN . . . . . 13

**3 Revisão Bibliográfica** **15**

**4 Metodologia** **19**

**5 Experimentos e Resultados** **21**
5.1 Conjunto de dados . . . . . 21
5.1.1 Descrição . . . . . 21
5.1.2 Construção do Dataset . . . . . 22
5.1.3 Construção e Processamento dos Dados . . . . . 23
5.1.4 Problemas . . . . . 24
5.2 Redes Neurais . . . . . 25
5.2.1 Geradora . . . . . 25

5.2.2 Discriminadora . . . . . 26
5.3 Framework de Redes Neurais Geradoras Adversárias . . . . . 26
5.3.1 Treinamento . . . . . 27
5.3.2 Ambiente . . . . . 28
5.4 Testes . . . . . 29
5.5 Resultados . . . . . 29

**6 Conclusão** **31**

## Anexos

**A Código dos Experimentos** **33**

**Referências** **35**

# Capítulo 1

# Introdução

## 1.1 Sistemas de Recomendação

Sistemas de recomendação são uma parte intrínseca à sociedade moderna e estão atrelados à maioria das plataformas online conhecidas. Seu propósito é trazer ao usuário uma experiência personalizada para o uso do serviço, sendo utilizados em empresas de grande porte como Youtube (recomendação de vídeos), Amazon (recomendação de produtos), Instagram (recomendação de conteúdo) e Netflix (recomendações de filmes e séries).

Diversas soluções são empregadas para resolver o problema de recomendação, cada uma com tecnologias e paradigmas diferentes. Esse trabalho passará por algumas delas e fará uma análise mais profunda sobre as tecnologias do estado da arte mais empregadas hoje, expondo a evolução dessa problemática.

Além disso, alguns algoritmos utilizados pelo mercado apresentam dificuldades em lidar com obstáculos de sistemas de recomendação, por exemplo, a esparsidade dos conjuntos de dados. Isso se dá, principalmente, pela velocidade com que esses algoritmos evoluem, sendo ainda muito recentes os recursos para mitigar tais problemas. Este trabalho explora algumas técnicas para reduzir seus impactos.

## 1.2 Comportamento do Usuário

O comportamento do usuário pode ser estudado de diversas formas, desde que haja dados para tal. Eles podem ser coletados de maneira ativa (avaliações e comentários) ou passiva (cliques e interações com itens).

Essas informações são intrínsecas ao estudo de comportamento, permitindo realizar análises de padrões, comparação de semelhanças e até identificação de características latentes. O uso de tais dados é essencial para predição de comportamentos futuros, pois a ideia é que as ações de um usuário seguem um padrão com base em seu passado (Luo, 2018).

## 1.3 Predição do Comportamento do Usuário

A predição de comportamento é de grande importância, pois permite prever o processo de decisão dentro de um determinado serviço. Por meio dela, empresas dos mais diversos setores do mercado corporativo obtêm informações sobre os possíveis impactos de seus produtos, propiciando um melhor planejamento de marketing, por exemplo. Além disso, para realizar uma boa predição de comportamento também é preciso que se tenha bons algoritmos de análise. O problema de sistemas de recomendação possui diversas opções para realizar tal trabalho com alta precisão.

Redes neurais profundas conseguem predizer avaliações e padrões de usuários com bastante acurácia. Essa tecnologia passou a ganhar relevância a partir de 2015 (Deldjoo, 2021) com a sua popularização por volta dessa mesma época. Porém, existem alguns problemas com seu uso, como explorado por Szegedy, 2013, que serão tratados neste trabalho com a utilização de Redes Geradoras Aniversárias - ou GAN. Esta tecnologia vem ganhando grande atenção, principalmente na área de visão computacional. A quantidade de artigos publicados nos últimos anos é extraordinária, principalmente depois que Yann LeCun - uma das figuras mais importantes em *deep learning* - ressalta que GAN é a ideia mais interessante de aprendizado de máquina dos últimos 10 anos (LeCun, 2016).

A arquitetura tratada no presente estudo é uma variação de GAN chamada RecGAN (explicada em detalhes em **??**). A vantagem de utilizá-la é levar em conta as características temporais de comportamento dos usuários na hora de fazer uma predição - ou recomendação.

## 1.4 Proposta

Este projeto se propõe a realizar um estudo amplo das tecnologias para predição de comportamentos de classes de usuários utilizando seu histórico de interações, com foco no problema de sistemas de recomendação. Dessa forma, pretende-se trazer a percepção do funcionamento desses algoritmos juntamente com seu contexto do estado da arte.

O segundo capítulo "Fundamentos" apresenta de forma sucinta as tecnologias e terminologias usadas ao longo deste trabalho, com o intuito de trazer familiaridade a esses conceitos chaves.

A "Revisão Bibliográfica" (capítulo 3) traz a história e a trajetória de sistemas de recomendação, focada principalmente em filtragem colaborativa. São apresentados os algoritmos desde a década de 90 até a era adversária atual.

Em "Metodologia" (capítulo 4) são evidenciadas as ferramentas e metodologias aplicadas para a execução da finalidade deste trabalho.

No capítulo "Experimentos e Resultados" é desenvolvida a prática da estrutura RecGAN. Os experimentos partem da problemática de organização, processamento e limpeza de dados, para em seguida passar pela criação das redes neurais e uma arquitetura para o treinamento de GAN e realizar por fim o treinamento, testes e metrificações importantes para o problema de sistemas de recomendação para enfim avaliar os resultados.

Finalmente, no capítulo 6: Conclusão será exposto os principais aprendizados e desafios encontrados, assim como o desfecho do objetivo proposto.

# Capítulo 2

# Fundamentos

Este capítulo detalha alguns conceitos que serão úteis para o entendimento do trabalho. Não se pretende exaurir os assuntos abordados, contudo as referências bibliográficas podem auxiliar o leitor que deseje se aprofundar mais na temática.

## 2.1 Redes Neurais

De acordo com Abu-Mostafa "redes neurais são um modelo inspirado pela própria biologia"[1], de forma que seu funcionamento é baseado no modelo neural do cérebro. Essa tecnologia tem diversas aplicações como no problema de classificação e na área de visão computacional.

As redes neurais são generalizações de um algoritmo usado em aprendizagem de máquina chamado *perceptron*, que usa uma transformação de características dele mesmo para aprender com dados. NN é uma tecnologia bem poderosa e flexível (Abu-Mostafa Y.S., 2015). Pois ela providencia uma forma robusta para aproximar funções alvo. É apropriada para inferir características de um problema (representadas em instâncias de vetores fixos), gerando uma saída unitária ou multivalorada. Porém, a troco de robustez e adaptabilidade seu treinamento é demorado (Mitchell, 1997).

Essa aproximação da função alvo - ou aprendizado - é feita sobre um conjunto de dados. Dele são extraídas um subconjunto de características que o usuário julga importante - juntamente com o alvo que a NN quer aproximar. Esses dados são passados pela rede proporcionando seu treinamento. Mitchell, 1997 cita exemplos de conjuntos de dados como sinais de sensores, fotos, vídeos e sons.

Uma rede neural tem estrutura de camadas, como exibido na Figura 2.1. Cada uma possui uma quantidade de neurônios que se "comunicam"entre si, como será explicado a seguir, proporcionando o resultado do algoritmo intuitivamente denominado fenômeno de aprendizado. O fluxo de dados ocorre a partir da primeira camada, alimentada pelo *input* (entrada), e segue passando camada por camada até atingir a última, onde é gerado o *output* (saída).

[1] "*Neural networks are a biologically inspired model*", em tradução livre (Abu-Mostafa Y.S., 2015, p. 52)

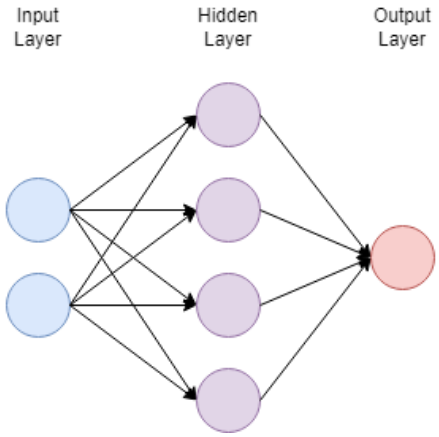


**Figura 2.1:** *Uma rede neural simples.* [a]

[a] Fonte: Criação do autor

Nesse modelo cada neurônio recebe uma informação e tem como função processá-la usando seus pesos - valores que realizam operações sobre sua entrada. Durante o aprendizado os pesos dos neurônios são ajustados, tornando o sinal para os próximos neurônios mais fortes ou mais fracos. Esse processo, chamado gradiente descendente (Figura 2.2). Ele se baseia no cálculo do gradiente da função custo associado a um estado da entrada e saída da rede e, usando um peso chamado taxa de aprendizado (*learning rate*), vai em direção à minimizar o erro. A função de custo (ou *loss*) é relacionada ao erro. Essa é a intuição base para o algoritmo de *back propagation* que ajusta o peso dos neurônios de acordo com o gradiente descendente. Portanto, o objetivo do algoritmo é se aproximar da função alvo minimizando o erro (Mitchell, 1997).

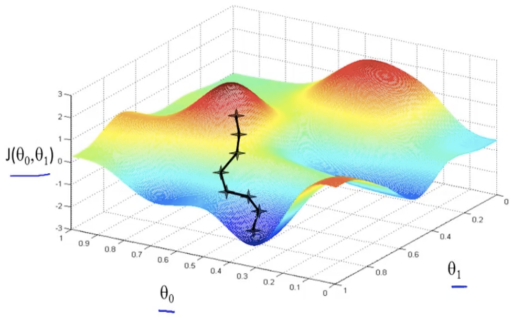


**Figura 2.2:** *Gradiente Descendente. A função é a loss e as setas indicam a direção do gradiente.* [a]*, acesso em: 10/11/2022*

[a] Disponível em: https://edgeaiguru.com/Feedforward-and-Backpropagation

A escolha de uma boa função *loss* é essencial para um bom treinamento de rede e afeta diretamente a sua performance (Reed, 1999). Exemplos comuns de funções são média do erro absoluto, similaridade de cosseno e média do erro quadrático. Neste trabalho é utilizada a *binary cross-entropy* que é bem explanada por Godoy, 2021.

O treinamento da rede se dá por meio de aprendizagem supervisionada - uma aborda-

gem em que é fornecida uma entrada de dados exemplares das características as quais se deseja inferir. Isso normalmente é feito organizando o aprendizado em lotes e em épocas. Enquanto lotes são definidos pela quantidade de amostras de dados selecionados para fazer a atualização de pesos, épocas são a quantidade de iterações (passadas) sobre o conjunto de treinamento.

No entanto é preciso tomar cuidado, pois como há um alto poder de aproximação de função, é possível que ocorra um problema de *overfitting* (Figura 2.3). Neste caso, surge uma situação em que a rede se adapta a um determinado conjunto de dados e, ao invés de aproximar à função objetivo, ela se aproxima ao conjunto de dados atual para minimizar o máximo possível o erro. Isso implica uma falta de generalização do modelo e limita sua aplicabilidade a novas instâncias do mesmo problema.

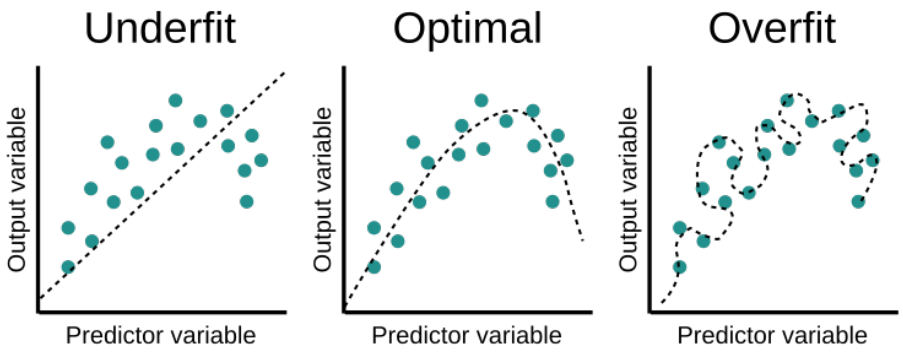


**Figura 2.3:** *Exemplo de overfitting e underfitting - que é quando o modelo não aprende.* [a]

[a] Disponível em: https://www.quora.com/Are-there-any-criteria-to-distinguish-overfitting, acesso em: 10/02/2022

Para medir a performance de uma rede é necessário realizar testes sobre dados reais não utilizados durante o treinamento. Isso é essencial para que o desempenho não tenha viés para um *dataset* específico, já que um valor alto para um teste dessa forma pode indicar *overfitting* (Abu-Mostafa Y.S., 2015). A performance pode ser mensurada a partir de diversas métricas, selecionadas de acordo com o problema. Exemplos comuns de métricas são *mean squared error*, F1 e NDCG.

## 2.2 Introdução à GAN

O principal propósito de GAN é a criação de dados, sejam eles imagens (Figura 2.4), sons ou até mesmo textos. Isso é feito computacionalmente de maneira relativamente simples, em particular para modelos geradores, o que representa um grande avanço para a área de *machine learning*.

Esse modelo de rede foi proposto em 2014 por Ian J. Goodfellow em conjunto com outros autores no artigo *"Generative Adversarial Nets"* (Goodfellow, 2014). Ele propõe um *framework* para estimar modelos geradores via um processo de adversidade entre dois modelos, um gerador e um discriminador.

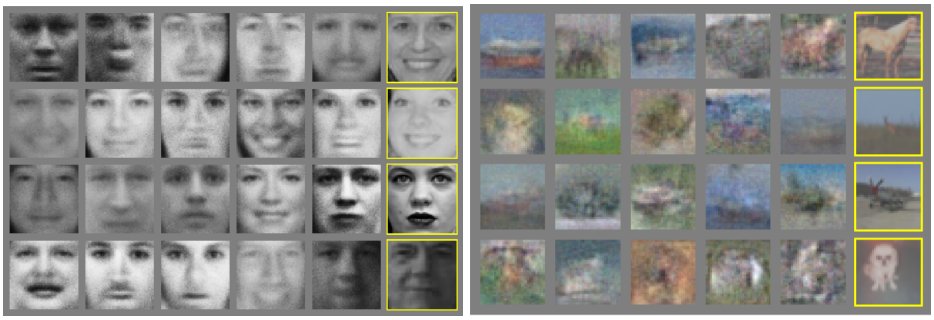

**Figura 2.4:** *Exemplos do uso de GAN para gerar faces e animais.* [a]

[a] Fonte: Goodfellow, 2014

De acordo com Goodfellow, 2014, a adversidade funciona como um jogo de minimax entre as duas redes neurais: uma geradora e outra discriminadora. A primeira tenta enganar a discriminadora, que por sua vez tenta discernir a probabilidade do dado ser originado da rede geradora ou pertencer ao conjunto de dados de treinamento. Dessa forma, treina-se ambos os modelos até que eles convirjam ao equilíbrio *Nash* [2] (K. Wang, 2017).

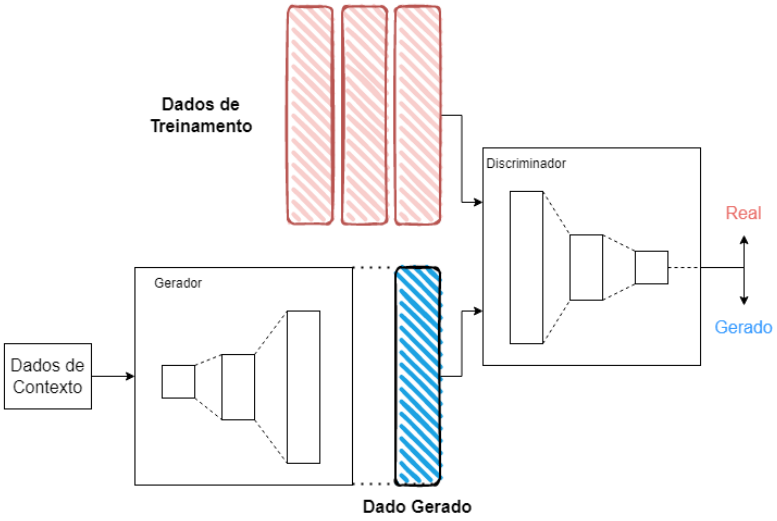


**Figura 2.5:** *Diagrama que ilustra a estrutura GAN.* [a]

[a] Fonte: Criação do autor

A rede neural geradora - em um modelo simples de ML - tem como entrada um vetor aleatório genérico, de tamanho a ser definido de acordo com o problema, e como saída um dado com formato coerente com o resultado desejado (Goodfellow, 2014). Por exemplo, no caso de geração de imagens 32px x 32px em preto e branco, a entrada pode ser um vetor de tamanho 10 e a saída um de tamanho 1024 (32 vezes 32) que será redimensionado para o formato correto posteriormente. Um outro exemplo é no caso de geração de texto, onde a saída é a quantidade de palavras que se deseja ser geradas.

[2] O equilíbrio Nash é quando não há mudanças de estratégia de ambas os jogadores para que eles tenham novas vantagens sobre o outro. (K. Wang, 2017)

É importante saber que ao invés de passar um vetor de valores aleatórios para o gerador também é possível passar um "contexto"do problema, como é feito pela *conditional GAN*, um estudo elaborado por Mirza, 2014. Isso propõe um melhor direcionamento para o dado gerado, fornecendo informações as quais a rede pode se basear durante a inferência ao invés de partir de dados arbitrários.

Já a rede neural discriminadora recebe duas classes de dados - a gerada e a real - e é treinada para diferenciá-las. A saída dessa rede é apenas um inteiro entre 0 e 1, usado para fazer a distinção.

## 2.3 Sistemas de Recomendação

Para o propósito de recomendação, é usado um algoritmo que procura predizer a avaliação ou a preferência de um usuário para um item - tal como um filme, produto ou conteúdo, utilizando as informações desse usuário ou do item. Existem diversas abordagens para resolver esse problema, dentre os mais comuns pode-se citar *collaborative filtering* (filtragem colaborativa) e *content-based filtering* (filtragem baseada em conteúdo).

De acordo com Goldberg, 1992, que criou o termo, a filtragem colaborativa (CF) se baseia na colaboração entre usuários. Nela, confrome cada usuário reage a conteúdos, são geradas informações para que se possa criar um filtro que selecione apenas conteúdo relevante para cada um. À partir desse filtro são elaboradas às recomendações.

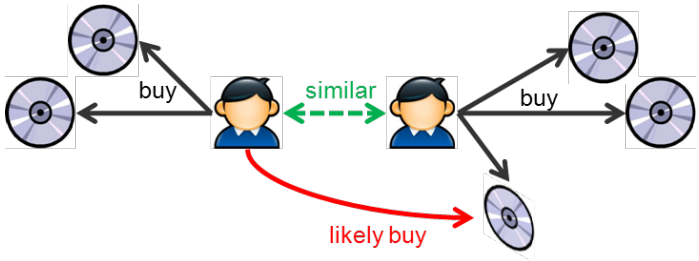


**Figura 2.6:** *Ilustração de filtragem colaborativa.* [a]

[a] Disponível em https://dzone.com/articles/recommendation-engine-models acessado em 14/02/2022

De acordo com Luo, 2018, nos dias atuais a filtragem colaborativa usa o histórico de avaliações de uma base de usuários junto com o histórico do usuário em questão para fazer a predição, baseando-se na intuição de que usuários que concordam entre si hoje vão concordar também no futuro (Figura 2.5). Assim, com essa comparação, assume-se que será possível predizer com certo grau de precisão a preferência de um usuário hoje e no futuro. Como o processo de FC lida tanto com uma grande base de dados quanto com as relações entre as métricas - que não permitem inferir facilmente relações entre os itens recomendados ou seus usuários, há uma grande abertura para resolver o problema com aprendizado de máquina.

Para mais informações sobre filtragem colaborativa, recomenda-se a leitura de Chae, 2018 que faz uma análise mais profunda sobre os algoritmos e desafios do assunto.

Vale também ressaltar que dois dos problemas mais comuns de sistemas de recomendação são a esparsidade e o ruído encontrados em conjuntos de dados (Gaoa, 2020). A esparsidade é uma casualidade da vasta quantia de itens disponíveis em relação a quantia que o usuário realisticamente interage. Como essa diferença é muito grande, é inevitável que as informações relevantes sobre o usuário sejam apenas uma pequena fração da quantidade de itens disponíveis (Gaoa, 2020). Isso causa perda de desempenho nos sistemas de recomendação.

Já os ruídos de dados são problemas que podem ser casuais ou maliciosos. Os casuais são um subproduto de erros de usuários, como uma compra de um produto fora de seus interesses ou até um engano em um momento de avaliação. Já os maliciosos são as situações em que há a intenção de prejudicar o sistema de recomendação(Gaoa, 2020). Ambos acontecem em cenários reais e, como demonstra Szegedy, 2013, modelos de aprendizagem de máquina são sensíveis a esse tipo de alteração.

A utilização de GAN promete mitigar ambos os problemas oferecendo novas ferramentas para isso.

## 2.4 Redes Neurais Recorrentes

De acordo com Wu, 2017, redes neurais recorrentes - ou RNN - são uma classe de redes neurais que permitem o aprendizado de padrões sequenciais ou de tempo. Isso é feito por meio do uso de conexões retroalimentadas (Figura 2.6). Alguns exemplos de usos de redes recorrentes são a predição de movimentações financeiras, síntese musical e aprendizagem em linguagem natural.

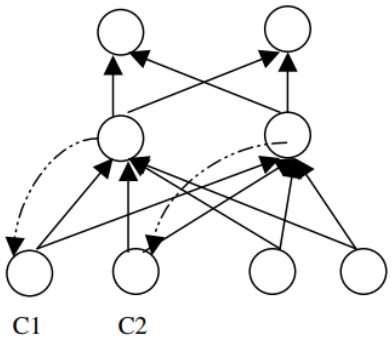


**Figura 2.7:** *Uma arquitetura de RNN simples.* [a]

[a] Disponível em Wu, 2017

O principal problema de RNNs é a falta de memória a longo prazo. Então, caso se esteja analisando uma entrada sequencial muito grande, é possível que informações mais antigas sejam perdidas (Phi, 2021).

A arquitetura usada nos experimentos desse trabalho é a GRU. De acordo com Phi, 2021, RNNs conseguem aprender a importância de uma amostra de um dado sequencial. Quase todas as RNNs no estado utilizam GRU ou LSTM (Figura 2.7) para criar soluções.

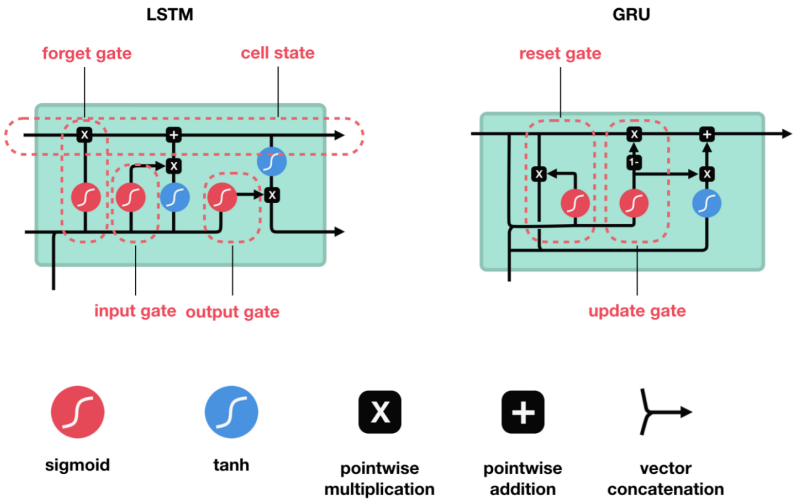


**Figura 2.8:** *LSTM e GRU.* [a]

[a] Fonte: Phi, 2021

Mais explicações matemáticas de como esse modelo funciona em Wu, 2017 e para mais informações sobre GRU e LSTM em Phi, 2021.

## 2.5 Redes Neurais Profundas

Redes Neurais Profundas (DNN) podem ser consideradas redes compostas por várias camadas (Figura 2.8). É usual para uma rede neural padrão ter uma ou duas camadas escondidas, enquanto uma DNN pode conter milhões de neurônios (*Deep Neural Network* 2021). Ao contrário de redes comuns ela também é responsável pela extração de características de suas entradas, não sendo necessário fazer isso manualmente.

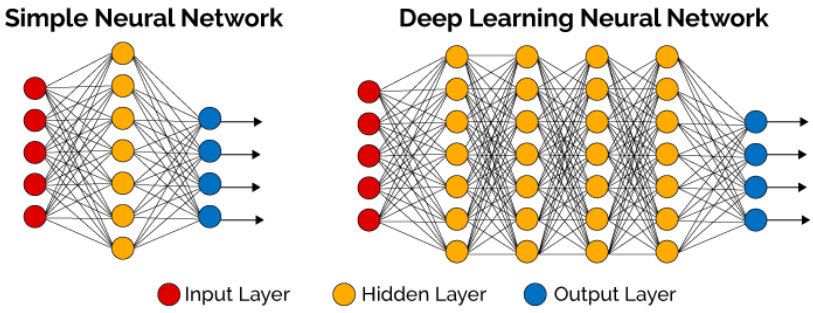


**Figura 2.9:** *Rede neural e rede neural profunda.* [a]

[a] Fonte: http://www2.decom.ufop.br/imobilis/inteligencia-artificial-e-deep-learning/

De acordo com WeiboLiu, 2017 elas podem ter vários usos práticos, dentre eles pode-se destacar as ferramentas de busca, carros autônomos, reconhecimento de caracteres, síntese de voz e processamento de linguagem natural.

Existem vários modelos (*Deep Neural Network* 2021) para esse tipo de rede, como redes neurais convolucionais (CNN) e redes neurais recorrentes (RNN).

Apesar de muito útil, as desvantagens de seu uso são a quantidade de tempo de treinamento e o tamanho do modelo. Por isso uma boa parte dos modelos são otimizados para o uso da GPU, que com sua capacidade de paralelismo, minimiza esses problemas.

## 2.6 Normalized Discounted Cumulative Gain

*Compute Normalized Discounted Cumulative Gain* [3] (NDCG) é uma métrica muito comum para avaliar sistemas de ranqueamento e recomendação. Essa métrica é a soma dos valores verdadeiros na ordem induzida pelos valores preditos depois de aplicar um desconto logarítmico e, então, divide-se pelo *score* que seria ideal. Assim é obtido um valor entre 0 e 1.

O artigo de Deldjoo, 2021 mostra a relevância dessa métrica de avaliação, dado que de 21 modelos mostrados mais da metade (12) usaram NDCG como uma das métricas.

NDCG é dada pela fórmula:

$$NDCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)},$$

onde $rel$ é a relevância (lista de saída do modelo) na posição $i$ e $p$ é uma posição ranqueada escolhida.

Essa é a métrica escolhida para a avaliação do modelo na seção de experimentos.

## 2.7 Problema de Aprendizado

O problema que será resolvido no trabalho é: Como ensinar um modelo à fazer uma boa recomendação para determinada pessoa. Essa necessidade permite que o usuário possa ao invés de ter que escolher entre uma enorme quantidade de itens, escolher apenas entre uma pequena quantidade personalizada às suas preferencias.

Para isso, é preciso que o modelo aprenda a fazer essa filtragem. Isso é feito treinando-o para "predizer"qual nota um usuário irá dar à certo item considerando seu histórico de avaliações anteriores.

A inferência das notas futuras de um usuário se dá pela capacidade do modelo, através de dados históricos, de identificar padrões e tendências de características latentes. Assim, é possível fazer inferência com certa acurácia sobre o histórico específico de dado usuário usando um modelo treinado no histórico geral.

Uma boa recomendação é uma que se aproxime das opiniões do usuário, ou seja, um item recomendado ser bem avaliado pelo usuário é um indicativo da qualidade na recomendação. Quanto mais semelhante for a saída do modelo com a realidade futura do usuário, melhor.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ndcg_score.html

## 2.8 Ranqueamento

Nesse trabalho a recomendação será feita através de uma escolha seleta de todos os itens disponíveis para um usuário. O modelo é responsável por induzir o quão propício o usuário é a gostar de cada item disponível no conjunto de dados. Porém, mostrar todos os itens e suas notas preditas não eliminam o problema de quantidade de opções, por isso, é preciso que haja uma seleção de relevância em cima dessa informação.

Para se obter apenas itens relevantes para o usuário é preciso que seja feito um ranqueamento. Sobre todos os itens avaliados pelo modelo são escolhidos os $N$ itens com maior avaliação predita, sendo $N$ a quantidade de itens que serão apresentados como recomendação para o usuário. Dessa forma, a filtragem de fato acontece.

A avaliação de um ranqueamento é feito por o quão perto a recomendação está das preferências reais do usuário. Isso pode ser feito usando uma métrica de qualidade de ranqueamento como *mean average precision* (MAP) ou *discounted cumulative gain* (NDCG). Nessas métricas é possível escolher um nível de ranqueamento (N) para realizar esse *score*, quanto maior a métrica, melhor é a recomendação. Portanto um bom valor de métrica de qualidade de ranqueamento implica em uma boa recomendação. Essa técnica será usada para a avaliação dos modelos durante os experimentos.

## 2.9 RecGAN

A arquitetura GAN que será usada nos experimentos é a RecGAN. Essa seção explicará brevemente quais são suas vantagens, parâmetros e saídas.

A entrada do modelo é o histórico inteiro de algum usuário, dividido em *timestamps*. Ou seja, para cada usuário se tem uma linha para cada item e uma coluna para cada instante de tempo. Uma tupla (tempo, item) contém a avaliação para aquele item naquele espaço de tempo, sendo a avaliação um número real no intervalo [0, 1], no qual 0 representa um item não avaliado. Um exemplo de uma tupla válida seria (35, 135), o que indica que no momento 35, o usuário avaliou um item de id 135, que pode ser o décimo dia para o filme "The Imitation Game"[4].

A entrada do modelo deve estar nesse formato para a entrada do modelo e deverá ser criada uma matriz que interprete essa especificação para proceder com o treinamento. Um ponto importante a se levantar é uma diferença entre esse modelo e uma GAN clássica: ao invés de ser passado um vetor de números aleatórios como entrada, é passado esse contexto, semelhante ao que é feito por Mirza, 2014. Isso implica que ao invés de gerar um dado vindo de uma entrada arbitrária se tem um contexto por onde a rede se guiará.

A saída do modelo é uma predição das avaliações de um usuário para período de tempo pré-determinado, por exemplo, dois anos depois da última avaliação, e seu formato é um vetor com todos os possíveis itens, cada um contendo a avaliação predita.

Essa saída pode ser interpretada como o valor de recomendação para cada um dos itens do conjunto de dados, um valor de 0 a 1 onde 0 é não avaliado e 1 sendo o mais compatível

[4] The Imitation Game. Direção de Morten Tyldum. 2014 (114 min.)

com um usuário possível. Através desses valores é possível fazer o ranqueamento, isto é, escolher os N maiores valores para serem usados como recomendação, e, fazer a avaliação do modelo permitindo avaliar a qualidade do modelo.

A vantagem de usar RecGAN para resolver o problema de recomendação está na necessidade do modelo de aprender comportamentos de longo e curto prazo levando em consideração a análise temporal, o que apesar de aumentar a performance do modelo não foi muito discutido e implementado pela comunidade científica (Homanga Bharadhwaj e Lim., 2018). Experimentos do artigo da tecnologia (Homanga Bharadhwaj e Lim., 2018) mostram que há um aumento considerável na qualidade das recomendações, mostrando sua relevância.

# Capítulo 3

# Revisão Bibliográfica

Neste capítulo são descritas as etapas de desenvolvimento de sistemas de recomendação até o estado da arte.

A história de sistemas de recomendações começa na década de 90, quando o primeiro artigo sobre o assunto foi publicado. A ideia de filtragem colaborativa foi inventada por Goldberg, 1992, em que ele propõe um sistema de e-mails chamado *Tapestry*. Nele, os usuários colaboram registrando reações sobre um conteúdo para ajudar a criar um filtro e torná-lo mais personalizado.

Desde algoritmos clássicos até arquiteturas complexas de redes neurais, a eficiência desses sistemas vêm aumentando.

É possível fazer uma separação entre duas eras bem distintas na cronologia dos sistemas de recomendação (Figura 3.1). Cada uma delas marca claramente as intenções da área acadêmica para o futuro da tecnologia. São elas a era da maximização da acurácia e a era adversária (Deldjoo, 2021).

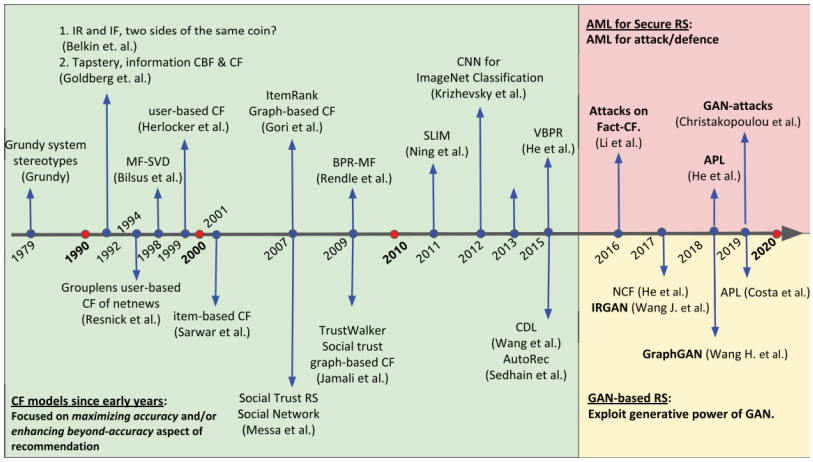


**Figura 3.1:** *Marcos em sistemas de recomendação em CF* [a]

[a] Fonte: Deldjoo, 2021

Nos primeiros anos (de 1979 até 2010) o maior interesse foi no desenvolvimento e melhoria de algoritmos com o propósito de criar recomendações mais precisas para os usuários. Assim, as experiências puderam se tornar cada vez mais personalizadas.

Nesse período eram utilizados algoritmos clássicos não neurais. Os principais baseavam-se no uso da semelhança entre usuários e itens, como por exemplo em Sarwar, 2001; e outros que utilizam predição de avaliação de usuários usando fatoração de matrizes, como explicado em Koren, 2009.

Logo em seguida (a partir de 2010), as redes neurais profundas (DNN) dominaram como solução. Seu sucesso se deu pela capacidade de aprendizagem de características latentes de conjunto de dados das DNNs, provando-se precisas em tarefas de predição. O trabalho de Ouyang, 2014 é um *framework open source* automatizado de recomendações chamado *AUTOREC* que usa redes neurais profundas.

Recentemente (2015 até a atualidade), preocupações sobre DNN como solução surgiram. Como demostrado por Szegedy, 2013, a tecnologia é vulnerável a *amostras adversárias*, isto é, variações sutis na entrada projetadas de forma a induzir a rede a cometer erros de saída. Essas discrepâncias não são perceptíveis por humanos e Szegedy, 2013 mostrou que podem fazer um modelo confundir um cachorro com um avestruz (Figura 3.2).

**Figura 3.2:** *Na primeira coluna é acusada a classe real, porém adicionando a imagem da coluna do meio, todas acusam "ostrich, Struthio or camelus"* [a]

[a] Fonte: Szegedy, 2013

Como esses sistemas são usados por serviços essenciais, como os recentes carros autônomos, os ataques adversários levantam preocupações sobre a segurança e a integridade desses modelos (Deldjoo, 2021). Isso também se aplica ao problema de recomendações.

Para mitigar os efeitos de ataques adversários, a partir de 2016 a comunidade se concentra nos modelos geradores adversários (GAN) por propor formas mais robustas de

se defender contra tais ataques.

Outra vantagem do uso de GAN é que este modelo proporciona formas de lidar com os problemas mais comuns de sistemas de recomendação, como esparsidade e ruído de dados. Por isso, a popularidade desse *framework* vem aumentando na atualidade (Gaoa, 2020) como forma de solução para essas questões.

GAN oferece recursos interessantes para mitigar o problema de ruído casual e malicioso, como as ideias utilizadas em APR e IRGAN:

- **APR**: - ou *Adversarial personalized ranking* - desenvolvido por He, 2018, utiliza maximização da função *loss* BPR (Rendle, s.d.) para aprender o ruído adversário. Uma medida tomada para a defesa contra ruído malicioso.
- **IRGAN**: Desenvolvido por J. Wang, 2017 é um sistema com o propósito de distinguir amostras informativas - sem ruído. Ele seleciona um subconjunto relevante de amostras do conjunto total.

Da mesma forma, para mitigar o problema de esparsidade existem:

- **CFGAN**: Desenvolvido por Chae, 2018, é um sistema de recomendação focado em *e-commerce*. A partir de dados relacionados a compras realizadas por determinado usuário, o algoritmo gera novas informações para simular - e prever - sua preferência.
- **RSGAN**: Desenvolvido por Yu, 2019, tem como objetivo ampliar os recomendadores sociais a partir de relacionamentos entre usuários reais e usuários gerados pelo modelo. Isso faz com que haja uma gama maior de dados para cada um deles.

Outra RS importante para mitigação da esparsidade é a RecGAN proposto por Homanga Bharadhwaj e Lim., 2018. Seu objetivo é gerar recomendações para usuários agregando algumas informações interativas dele, por exemplo sua localização geográfica - como feito em Geo-ALM (Liu, 2019).

No caso de RecGAN, seu modelo utiliza o momento no tempo da avaliação de usuário (*timestamp*). Isso é feito usando uma rede recorrente GRU modificada com *relu* para a análise temporal, que considera a relevância do intervalo de tempo para a recomendação. Algo que faltou no artigo é um estudo mais aprofundado sobre o impacto da modificação *relu* da *GRU*. Esse será o modelo proposto no capítulo 5 "Experimentos e Resultados".

# Capítulo 4

# Metodologia

Esta seção conceitua os métodos e ferramentas utilizados para atingir a finalidade deste trabalho.

O objetivo é explorar o uso da tecnologia GAN, proposto por Ian J. Goodfellow em conjunto com outros autores para a predição de comportamento de classes de usuários, mais especificamente em uma análise de uso e performance de GAN em sistemas de recomendação (Goodfellow, 2014).

Para isso, será usado um modelo de redes neurais chamado Rede Recorrente Geradora Adversária (RecGAN) - que utiliza uma camada recorrente - com o propósito de considerar características latentes e temporais para fazer predições. Isto é, o modelo abrange comportamentos de curto e longo prazo de itens e usuários.

Dessa forma, este estudo visa reproduzir o trabalho do artigo *"RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems"* (Homanga Bharadhwaj e Lim., 2018), onde os autores experimentam o uso de redes recorrentes geradoras adversárias para tentar resolver o problema de recomendação.

O dataset usado será o *Netflix Prize*, um dos mais renomados para a resolução do problema de recomendação. Além disso será feita uma breve análise exploratória para estabelecer o balanceamento da quantidade de avaliações pelos anos e com quais dados serão usados.

Para o treinamento do modelo, os dados devem estar conforme um padrão específico de entrada. O pré-processamento desses dados deverá ser realizado antes de continuar com o desenvolvimento da rede. E será procedido por uma forma de armazenamento dos dados processados em disco para maior facilidade de carregá-los em memória durante o treinamento.

A partir disso serão realizados experimentos para testar o modelo proposto por Homanga Bharadhwaj e Lim (Homanga Bharadhwaj e Lim., 2018). O experimento consiste na implementação detalhada do modelo - composto por duas redes neurais na estrutura GAN, seu treinamento completo juntamente com o tratamento e a modelagem de um grande volume de dados.

O desenvolvimento dos modelos neurais serão realizado utilizando *keras* em *tensorflow*, uma biblioteca renomada para implementação de redes neurais disponível em Python [1]. É de boa prática que o desenvolvimento de ambas as redes estejam prontas antes da implementação dos modelos, assim facilita a criação de um fluxo que leva em consideração esse problema em específico.

O treinamento será realizado de acordo com o fluxo proposto pela arquitetura de GAN onde ambas as redes Geradora e Discriminadora serão treinadas juntas até chegar ao equilíbrio. Isto será implementado de acordo com as especificações de Homanga Bharadhwaj e Lim., 2018, tomando-se algumas liberdades pois alguns detalhes não estão descritos no artigo.

Por fim, a performance da qualidade de recomendação será testada a partir da medição da métrica NDCG [2], o que permite uma análise e conclusão da efetividade do modelo.

Também será feita uma comparação de métricas entre os algoritmos vigentes: Autorec (Ouyang, 2014) e RNN (Wu, 2017). Assim é possível trazer uma comparação com o que é realmente usado, mostrando a performance real do modelo e como ele se encaixa no estado da arte.

Com isso, pretende-se ter ao final do projeto uma avaliação de um algoritmo que reflita o estado da arte de sistemas de recomendação.

[1] https://keras.io/api/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ndcg_score.html

# Capítulo 5

# Experimentos e Resultados

Nesta seção será apresentado o desenvolvimento do modelo na prática, experimentos e resultados obtidos.

Como citado no capítulo 4: Metodologia, este trabalho reproduz o que foi feito no artigo *"RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems"* (Homanga Bharadhwaj e Lim., 2018).

Esse processo ocorre em 5 etapas: construção do conjunto de dados, criação das redes neurais, criação do *framework* de GAN, treinamento e realização de testes.

## 5.1 Conjunto de dados

### 5.1.1 Descrição

O conjunto de dados utilizado nesse projeto é o *"Netflix Prize"* (Netflix, 2006), um *dataset* disponibilizado pela *Netflix* em uma competição pelo melhor algoritmo para predizer avaliações de filmes por usuários. O prêmio final era de um milhão de dólares para quem submetesse o algoritmo com melhor performance.

Esse é um dos *datasets* mais usados para resolver o problema de recomendação devido à grande quantidade de dados que ele disponibiliza e por ele ser aberto. Como esse é o conjunto de dados usado pelo artigo, ele também será usado para esse trabalho para que seja possível fazer a comparação entre os resultados do artigo e os resultados obtidos.

O *dataset* conta com mais de 100 milhões de avaliações de 480 mil usuários para cerca de 17 mil títulos. Esses dados são a totalidade dos dados coletados entre outubro de 1998 e dezembro de 2005 pela Netflix. Todas as avaliações são datadas, e, para manter a anonimidade dos usuários da plataforma, foram atribuídos IDs aleatórios para cada usuário.

Os dados têm aproximadamente 2 Gbs de tamanho e vêm divididos em 7 arquivos, cada um com um propósito diferente:

- *combined_data_{1 até 4}.txt*: Contém todo o *dataset* para treinamento.

- *probe.txt*: *Dataset* usado para testar predições antes de submeter para a competição.
- *movie_titles.csv*: Armazenadas as informações dos filmes, ID, Nome e ano de lançamento.
- *qualifying.txt*: *Dataset* de qualificação para testar predições submetidas.

Os principais arquivos que serão usados para treinar a rede são os *combined_data_n.txt*. Cada arquivo segue a mesma estrutura: para cada ID de filme, especifica-se o ID do usuário, sua avaliação numérica de 1 a 5, e a data em que a avaliação foi feita, como no seguinte exemplo:

```
1    MovieID1:
2    CustomerID11,Review11,Date11
3    CustomerID12,Review12,Date12
4    . . .
5    MovieID2:
6    CustomerID21,Review21,Date21
7    CustomerID22,Review22,Date22
```

Porém, como será preciso mudar essa estrutura para algo que a rede neural aceite como entrada, haverá um pré-processamento mais adiante.

Além disso para facilitar o processamento de dados *probe.txt* não será usado, isso se dá pois a quantidade de dados fornecido pelos arquivos *combined_data_{1 até 4}.txt* é o suficiente para o treinamento e teste do modelo, e, processar além desses arquivos diminuiria muito a performance do fluxo de treinamento.

### 5.1.2 Construção do Dataset

#### Modelagem dos Dados

A estrutura padrão fornecida não é compatível para ser usada no treinamento de uma rede neural comum, portanto, é preciso fazer um pré-processamento para transformar os dados numa estrutura adequada para treinamento.

Como será feita uma predição para um determinado usuário, é necessário que a estrutura seja em nível de usuário, ou seja, cada usuário terá informação de cada avaliação, quando ela foi feita e para qual filme. Assim, é preciso construir uma matriz tridimensional - denominada $M$ - na qual a primeira dimensão representa os usuários, a segunda é o mês no qual o usuário fez uma avaliação e a última é o filme ao qual tal avaliação pertence. O valor de uma entrada é a própria avaliação. Isto é:

$$M = [u_0, u_1, ..., u_i, ..., u_{n-1}, u_n]$$

$$u_i = [m_0, m_1, ..., m_t, ..., m_{T-1}, m_T]$$

$$m_t = [a_0, a_1, ..., a_j, ..., a_{k-1}, a_k]$$

$$0 \leq a_j \leq 1$$

Onde $M$ é a matriz que alimentará o modelo neural, $u_i$ é o histórico completo de

avaliações de cada $i$ usuário, $m_t$ é a lista de avaliações para cada filme em um determinado tempo $t$ (nesse caso, mês, entre 0 e 48) e $a_j$ é a avaliação para o filme $j$ entre 0 e 1.

Assim é possível facilmente estruturar a matriz de usuários para passar para o modelo, e, fazer predição em cima de qualquer um dos usuários e futuros usuários que tenham avaliações dentro de um período específico.

#### Meta e Dados de aprendizado

Para fins de facilidade de experimentação desse trabalho foi estabelecido que a meta da predição será as avaliações de um usuário dos últimos dois anos. Para isso, é preciso definir o conjunto de dados de treinamento que será utilizado como entrada para a predição (conjunto X), e o conjunto de dados reais que o modelo gostaria de alcançar com sua predição (conjunto Y).

O X será a modelagem criada anteriormente de todo o histórico do usuário até dois anos antes de acabarem os dados.

O Y será um agregado das avaliações dos itens de um usuário nos últimos dois anos. Dessa forma ele será apenas uma lista de todos os filmes com as avaliações nesses últimos dois anos. Dando prioridade, caso haja duas avaliações para o mesmo filme, à avaliação mais recente.

Isto é, se os últimos anos do dataset forem 2015 e 2016, e, o usuário avaliou em fevereiro de 2016 5 estrelas para o filme A e 4 estrelas para o filme B em março de 2015, em um mesmo vetor estarão as duas avaliações, sem a separação mensal.

Assim será possível para cada usuário fazer a avaliação do modelo, comparando a saída do modelo usando X com o Y que definimos.

### 5.1.3 Construção e Processamento dos Dados

Para construir a matriz necessita-se ter algumas informações básicas: a quantidade de filmes, a quantidade de intervalos de tempo (meses) e o número de usuários usados em um *batch* (ver capítulo 2 Fundamentos). Fazendo uma análise simples é fácil descobrir que no *dataset* existem 17.770 filmes, 47 colunas referentes aos meses mais a coluna do Y que compila dois anos de avaliações, e 480.000 usuários.

Em seguida, para montar a matriz que alimenta o modelo neural deve-se descobrir a qual filme e usuário cada avaliação pertence. Isso é feito iterando sobre as linhas dos arquivos de dados, processando todas as informações e inserindo-as na matriz corretamente. Assim, a matriz $M$ segue a modelagem:

```
1    M[id do usuário][intervalo de tempo][id do filme]
```

Para que seja possível colocar os dados nessa matriz, primeiramente, é necessário criá-la com as dimensões 480.000x48x17.770 e preenchê-la com zeros - que é o padrão para nenhuma avaliação - e então, iterar sobre todas as avaliações realizando o preenchimento.

Inicialmente se descobre o filme ao qual as próximas avaliações se referem, sendo indicado na linha pela presença do caractere ":". Uma vez identificado o ID do filme este valor é salvo na memória, e isso é feito continuamente até que todos os filmes tenham sido percorridos.

Cada uma das avaliações é dividida em três partes: o usuário, a nota e a data da avaliação, separadas por vírgulas. A seguir, a função do *python split()* é chamada para separar os elementos da linha em uma lista, usando, no caso, ","como separador. Assim, o primeiro elemento é o usuário, o segundo a avaliação e o terceiro a data.

Porém, os IDs dos usuários não estão ordenados de 0 a 480.000, são números aleatoriamente distribuídos. Para identificar os usuários de forma fácil serão distribuídos novos IDs. Para isso foi criado um dicionário com o mapeamento de cada usuário com seu novo identificador. Com o auxílio de uma variável contadora i inicializada com o valor 0, percorre-se os usuários e atribui-se o valor de i para o usuário caso seja sua primeira aparição e, em seguida, atualiza o valor da mesma. Caso o usuário já exista, o código identifica o valor previamente atribuído a ele, evitando assim o problema de IDs aleatoriamente distribuídos.

Outro ponto importante é que a data da avaliação está padronizada no formato "YYYY-MM-DD". Uma conversão para período de tempo entre 0 e 48 é necessária, pois este valor precisa estar de acordo com a modelagem da matriz. Para fazer isso basta calcular a diferença, em meses, entre a menor data do *dataset* e a data que se deseja converter. É importante ressaltar que caso uma data esteja entre os dois últimos anos, esta estará obrigatoriamente presente na última coluna, que será o Y.

A última etapa é transformar o valor da avaliação em um valor numérico e a normalizá-lo para o padrão entre 0 e 1.

Feito todas as etapas, a matriz está pronta para ser preenchida com os dados e, consequentemente, pronta para ser submetida ao treinamento do modelo GAN.

### 5.1.4 Problemas

É comum que um tratamento de dados de alto volume tenha problemas, como o encontrado neste trabalho com a alocação de memória RAM. Espera-se que um computador comum não consiga alocar uma matriz de tamanho 480.000x48x17.770, dado que isso são cerca de 409 bilhões de entradas e 400Gb de memória. Além disso, é necessário alocar memória para rodar o modelo preditivo em si.

Para resolver os problemas de memória uma solução é fazer o treinamento da rede em lotes (*batches*), podendo assim treinar o *dataset* inteiro em pedaços menores e evitando que hajam muitos dados alocados em memória ao mesmo tempo. Um valor que funcionou bem para as configurações do experimento foi de mil usuários para cada passo do treinamento pois não ultrapassa o limite da memória física (16Gb de RAM).

Porém, o treinamento em lotes por si só não resolve todos os problemas relacionados à falta de memória, pois antes é preciso fazer o pré-processamento de todo o *dataset* para que ele possa ser usado no treinamento. A primeira solução encontrada foi: para cada mil usuários, procurar todas as avaliações deles nos arquivos. Porém isso é muito

pouco performático, já que os dados estão organizados a nível de filme e não de usuários, sendo necessário percorrer todo o conjunto de dados para garantir que todas as avaliações para dado usuário tenham sido consideradas. Além disso, construir a matriz extraindo as informações de texto para todos os 480 lotes é muito lento, demorando até 5 minutos por lote incluindo extração e treino, totalizando 40 horas no pior caso, o que tomaria muito tempo.

Outra solução é fazer um pré-processamento da matriz e alocá-la em disco ao invés de memória, assim, a matriz estará pronta para o treinamento e não se perde tempo com a extração dos dados em tempo de treinamento. Porém, essa matriz inteira tem o tamanho de 400Gb, o que torna a alocação em disco impraticável. É preciso, portanto, de um meio termo: uma solução que use a alocação em disco com pré-processamento porém não use tanta memória.

A solução final é usar uma tecnologia chamada *Coordinate List* (COO) que serve para alocação eficiente de matrizes esparsas. Como a maior parte da matriz é composto por zeros ela é considerada uma matriz esparsa. Como descrito por Scipy, 2021 A vantagem de utilizar uma COO é que ela guarda apenas os valores não zero da matriz como uma listas de tuplas (linha, coluna, valor) da matriz, ao invés de guardar todos, assim economizando a alocação de espaço para todos os zeros na matriz.

Usar COO para lidar com o problema da matriz esparsa é uma alternativa adequada pois são mais de 17 mil filmes presentes no *dataset*, é razoável assumir que muitos dos usuários não possuam essa quantidade de filmes avaliados. Como o valor padrão para uma avaliação não feita é 0, o uso de COO se torna muito eficiente dado que a maioria dos valores armazenados na matriz é 0. A redução apresentada pelo uso da tecnologia se mostrou excelente uma vez que o espaço necessário para o armazenamento diminuiu em quase 10 vezes, de 400Gb para 4.7Gb.

O processo consiste em fazer o pré-processamento, salvando em disco as matrizes esparsas codificadas em *Coordinate Lists* e separadas em lotes de 1000 usuários. No treinamento, cada lote em memória é carregado e a codificação é desfeita, possibilitando a passagem pelo modelo. Dessa forma o treinamento da rede é feito de forma eficiente, sem o consumo excessivo de memória.

## 5.2 Redes Neurais

A arquitetura das redes serão a recGAN, que foi apresentada no artigo reproduzido por esses experimentos (Homanga Bharadhwaj e Lim., 2018). Ambas as redes são implementadas com *keras*.

### 5.2.1 Geradora

A rede Geradora consiste em 3 camadas:

- GRU de 1000 unidades de ativação *relu*, ativação recorrente *sigmoid* e entrada com o formato 1x17770.
- Densa de 17770 unidades com ativação *tanh*.

- Remodeladora para o formato 1x17770.

Para evitar erros comuns de formatação dos dados em redes neurais é preciso ter muita atenção na arquitetura da rede, uma vez que a saída da rede Geradora será a entrada da rede Discriminadora.

A primeira camada recebe como entrada um único período de tempo de um usuário, e a última camada retorna a predição de quais filmes este usuário pode ou não gostar - ou seja, a recomendação. A recomendação é uma lista de todos os filmes e seus devidos *scores* entre 0 e 1. Quanto mais próximo de 1, mais provável que o usuário goste daquele filme. A rede Geradora é a rede que vai ser de fato utilizada para o propósito de recomendação.

A função *loss* usada para essa rede é *binary crossentropy*.

A implementação em *tensorflow* com *keras* fica:

```
1    GRU(1000, activation='relu', recurrent_activation='sigmoid', input_shape
         =(1,17770,))
2    Dense(17770, activation='tanh')
3    Reshape((1,17770))
```

### 5.2.2 Discriminadora

A rede Discriminadora consiste em apenas 2 camadas:

- GRU de 1000 unidades de ativação *relu*, ativação recorrente *sigmoid* e entrada com o formato 1x17770.
- Densa de 1 unidade com ativação *sigmoid*.

A primeira camada recebe como entrada tanto uma predição da Geradora quanto um dado real do que o usuário de fato avaliou e a última camada dessa rede apresenta um valor que representa a tentativa de discernir se o dado recebido é um dado real ou gerado pela Geradora. Quanto mais perto de 0 maior a chance desse dado ser gerado, quanto mais perto de 1, mais chance de ser real.

A função *loss* usada para essa rede é *binary crossentropy*.

A implementação em *tensorflow* com *keras* fica:

```
1    GRU(1000, activation='relu', recurrent_activation='sigmoid', input_shape
         =(1,17770,))
2    Dense(1, activation='sigmoid')
```

## 5.3 Framework de Redes Neurais Geradoras Adversárias

Para treinar a GAN é preciso que ambas as redes sejam treinadas simultaneamente para que elas convirjam em velocidades semelhantes. Para tal, um fluxo de treinamento precisa ser montado.

### 5.3.1 Treinamento

Para o treinamento existem os conjuntos X e Y de treinamento de um lote, sendo cada lote composto por 1000 usuários. Como citado anteriormente na subseção 5.1.2, o X são as informações históricas das avaliações que o usuário fez até dois anos do final do *dataset* e o Y é o compilado dos dois últimos anos do *dataset* em uma coluna só. Isso foi definido na modelagem dos dados para ser compatível com o treinamento da rede e pode ser alterado caso haja necessidade.

A divisão entre o conjunto de dados de treinamento e de teste foi de 90%, portanto dos 480 mil usuários, 432 mil serão usados para o treinamento e 48 mil usuários para teste da rede. É esperado que a rede treinada tenha uma performance muito semelhante à performance do artigo.

O treinamento ocorre da seguinte forma: Para cada usuário, se itera treinando as redes sobre todos períodos de tempo, usando o estado anterior para gerar o próximo estado. Assim é possível usar a informação temporal para o treinamento do modelo passando do "estado"$x_t$ para $x_{t+1}$ até percorrer todas as avaliações de um determinado usuário, para todos os usuários.

Para avaliar se o modelo está com uma boa performance, durante a avaliação o modelo passa por todo o histórico do usuário, gravando apenas os últimos dois anos. É possível então fazer uma comparação entre o que foi gerado, e o Y definido anteriormente.

Para começar o treinamento é preciso definir duas verdades absolutas que serão usadas no treinamento do Discriminador. Caso seja um lote gerado, a verdade para a Discriminadora será 0, caso seja um lote real, essa verdade é 1 (Goodfellow, 2014). Portanto, por facilidade é preciso criar duas listas com essas verdades, para o *fake* - ou gerado, uma lista de zeros do tamanho do lote de treinamento é criada, para o *valid* - ou real, uma lista de uns do tamanho do lote de treinamento é criada.

São criadas as verdades absolutas:

```
1    valid = np.ones((batch_size, 1))
2    fake = np.zeros((batch_size, 1))
```

Uma vez que todas as configurações estão prontas para o treinamento, o código itera sobre a quantidade de épocas escolhidas para cada lote de usuários. Os valores escolhidos para minimizar o uso de memória foram:

- Lote de usuários: 1000
- Lote de treinamento: 5
- Épocas: 200

Isso significa que para cada lote de 1000 usuários itera-se 200 vezes (épocas) para 5 usuários em cada época. É válido ressaltar que como é uma uma rede recorrente, cada *timestamp* é treinada para tentar avaliar a próxima, fazendo um *feedforward*.

Em cada iteração acontece o treinamento. Inicialmente escolhe-se uma amostra aleatória de 1000 usuários aleatórios, seus respectivos Xs e Ys, e, sobre esses dados acontecerá o treinamento da rede.

Assim são escolhidos os lotes:

```
idx = np.random.randint(0, X_train.shape[0], batch_size)
real_seqs = X_train[idx]
y_seqs = y_train[idx]
```

A rede Geradora então utiliza os dados reais para criar sequências que servirão como entrada para o treinamento da rede Discriminadora:

```
gen_seqs = self.generator.predict(real_seqs)
```

A Discriminadora é treinada nas sequências reais e nas geradas pela Geradora, assim a Discriminadora aprende a discernir entre as duas. A *loss* é capturada apenas para facilitar o acompanhamento da evolução do modelo.

As verdades universais *valid* e *fake* são passadas à discriminadora para que ela possa discernir um valor gerado de um valor real - retornando valores próximos de 1 para as sequências geradas e próximos de 0 para as reais, como citado anteriormente.

O código a seguir mostra a captura da *loss* de monitoramento seguindo essas regras:

```
d_loss_real = self.discriminator.train_on_batch(y_seqs, valid)
d_loss_fake = self.discriminator.train_on_batch(gen_seqs, fake)
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

Em seguida, precisa-se treinar a Geradora. Para isso é usado o conjunto das duas redes para ser possível treinar a Geradora de forma mais fácil. O treinamento é feito de forma muito similar ao treino da Discriminadora, como a Geradora só é treinada sobre os dados de sequências reais só é preciso treinar uma vez.

O treinamento da Geradora é feito da seguinte forma:

```
g_loss = self.combined.train_on_batch(real_seqs, [y_seqs, valid])
```

Como etapa final é importante salvar os modelos usados durante o treinamento para que seja possível voltar no último ponto salvo caso haja algum erro durante o processamento, evitando assim o desperdício de horas de treinamento. Os modelos estão sendo salvos a cada lote de 1000 usuários, porém esse número pode ser diminuído por questões de performance.

Então salva-se o modelo:

```
self.generator.save("generator.h5")
self.discriminator.save("discriminator.h5")
```

### 5.3.2 Ambiente

O treinamento dos 432 mil usuários demorou aproximadamente 16 horas. O computador que rodou tal experimento contém as seguintes especificações:

- Sistema Operacional: Ubuntu 20.04.3 LTS 64-bit
- CPU: AMD® Ryzen 5 5600x 6-core

- GPU: GeForce RTX 2060
- RAM: 16 Gb

Não foi possível fazer o uso otimizado da GPU na execução do código pois o *tensorflow* não suporta a camada GRU com ativação alterada em GPU[1].

Memória foi um problema em todas as etapas do processo, portanto, para rodar o código é necessário ter uma quantia de RAM igual ou superior, ou, tentar valores menores para o número de usuários por lote.

## 5.4 Testes

O teste do modelo é realizado em cima da rede Geradora que tenta predizer quais são os filmes que dado usuário vai assistir e dar avaliação nos dois anos seguintes.

Ele é rodado em apenas 10% da base total de usuários, isto é, 48 mil usuários. Como a memória não suporta um volume muito maior que 100 usuários por teste de métrica, a performance da rede foi testada para a média desses valores.

A métrica usada para o teste da rede se chama *Compute Normalized Discounted Cumulative Gain*[2] (NDCG) da biblioteca *sklearn* em python.

O artigo no qual este trabalho se baseou usa como métrica o NDCG, portanto para fazer a comparação com a performance obtida é importante o uso da mesma métrica. Para $p$ serão usados 3 e 5 usando a notação NDCG@3 e NDCG@5.

## 5.5 Resultados

O resultado esperado é semelhante ao obtido no artigo seguido (quanto maior o valor da métrica, melhor) que são:

- NDCG@3: 0.50
- NDCG@5: 0.47

Também é importante ver como o experimento performa em relação a outros algoritmos. Dessa forma, a seguinte tabela mostra alguns algoritmos utilizados para resolver o problema de recomendação:

| | Experimento | RecGan2 | AutoRec | RNN |
|---|---|---|---|---|
| NDCG@3 | 0.48 | 0.57 | 0.51 | 0.49 |
| NDCG@5 | 0.45 | 0.55 | 0.49 | 0.47 |

As outras duas redes mostradas na comparação são AutoRec por Ouyang, 2014 e RNN por Wu, 2017.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU

[2] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ndcg_score.html

Os resultados obtidos no experimento não são tão próximos do que foi feito no artigo, tendo uma diferença de 0.09 para NDCG@3 e 0.1 para NDCG@5. Vários fatores podem ser atribuídos para essa diferença, como a modelagem dos dados, métodos de limpeza dos mesmos ou até mesmo o método de treinamento e qual foi a condição de parada para a aprendizagem do modelo.

Mesmo que os resultados tenham sido piores do que o artigo, não podemos considerá-los ruins. Pois, ainda assim se assemelham aos resultados de outras redes neurais no estado da arte.

# Capítulo 6

# Conclusão

O estudo de tecnologias no estado da arte é complexo, sobretudo quando se trata das quais começaram a surgir nos últimos dois anos. Por ser muito recente, a quantidade de material o qual se pode basear é muito limitada, dificultando bastante a implementação de alguns detalhes do trabalho. Um bom exemplo disso foi o treinamento de uma camada recorrente sem a possibilidade de utilizar os cálculos em GPU, uma vez que a biblioteca ainda não oferece suporte a essa funcionalidade em camadas modificadas.

Apesar de estar nos primórdios de seus estudos, o modelo de RecGAN demonstrou ser muito promissor dada a performance dos experimentos. Verifica-se que o bom desempenho se dá não apenas pela modelagem comportamental que considera o tempo das ações do usuário, mas também pelos recursos que atenuam os problemas de esparsidade e ruído nos dados.

Um dos maiores desafios do trabalho foi utilizar dados reais nos experimentos. A maior parte do tempo foi dedicada a processar e modelar o *dataset*, principalmente pelas adversidades de limitações de hardware que impediram alguns processamentos. Por conta disso, se mostra essencial ter uma boa base teórica para conseguir manipular grandes quantidades de dados e realizar um trabalho como este.

Outra dificuldade encontrada foi na utilização de GAN, pois os materiais disponíveis para conduzir o passo-a-passo da implementação são destinados a problemas divergentes que não necessariamente se aplicam ao que este estudo se propõe.

Com relação aos resultados obtidos nos experimentos, percebe-se que diferem dos apresentados no artigo de Homanga Bharadhwaj e Lim., 2018, sendo até mesmo pior do que alguns dos outros algoritmos referências no estado da arte. A principal hipótese levantada para essa diferença é que a modelagem foi levemente modificada. É possível que os autores tenham descartado alguns filmes, diminuído a esparsidade do modelo, utilizado outros otimizadores e algoritmos para a função de custo, entre outros detalhes que possam ter ampliado a performance do modelo. Porém, para identificar todos esses fatores seria necessária uma análise mais profunda de cada etapa do processo demonstrado no artigo.

Apesar de não alcançar os resultados ideais, o experimento cumpre seu papel ao

reforçar a importância do uso de GANs e RNNs para executar a predição de classes de usuários. Diversos outros estudos comprovam a eficácia da técnica e o assunto têm ganhado relevância dentro da comunidade acadêmica nos últimos anos, demonstrando uma movimentação positiva em direção a essa tecnologia quando se trata de sistemas de recomendação.

# Anexo A

# Código dos Experimentos

O código usado para a realização dos experimentos está no repositório do GitHub [1]: github.com/vbatistella/2021-tcc

Para executar o treinamento é preciso realizar os seguintes passos:

1. Baixar o conjunto de dados do "Netflix Prize"[2].
2. Copiar os arquivos "combined_data_{N}.txt"para o diretório ‘dataset/archive/‘.
3. Executar o código de pré-processamento dos dados com ‘$ python build_dataset.py‘.
4. Executar o código de treinamento ‘$ python main.py‘

As métricas são avaliadas depois do treinamento apenas. Os modelos treinados não estão disponíveis no repositório devido ao seu tamanho de armazenamento.

O resultado final pode variar devido à aleatoriedade de alguns parâmetros durante o treinamento.

[1] https://github.com/

[2] https://www.kaggle.com/netflix-inc/netflix-prize-data

# Referências

[Abu-Mostafa Y.S. 2015] Lin H Abu-Mostafa Y.S. Magdon-Ismail M. *Supplementary materials for Learning from data*. 1ª ed. 2015 (citado nas pgs. 5, 7).

[Chae 2018] D. Chae. "Cfgan: a generic collaborative filtering framework based on generative adversarial networks". Em: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018) (citado nas pgs. 9, 17).

[*Deep Neural Network* 2021] *Deep Neural Network*. URL: https://www.sciencedirect.com/topics/computer-science/deep-neural-network (acesso em 23/12/2021) (citado na pg. 11).

[Deldjoo 2021] Yashar Deldjoo. "A survey on adversarial recommender systems: from attack/defense strategies to generative adversarial networks". Em: *ACM Computing Surveys* (2021) (citado nas pgs. 2, 12, 15, 16).

[Gaoa 2020] Min Gaoa. "Recommender systems based on generative adversarial networks: a problem-driven perspective". Em: (2020) (citado nas pgs. 10, 17).

[Godoy 2021] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a (acesso em 23/12/2021) (citado na pg. 6).

[Goldberg 1992] D. Goldberg. "Using collaborative filtering to weave an information tapestry". Em: *Comm. ACM* (1992) (citado nas pgs. 9, 15).

[Goodfellow 2014] Ian J. Goodfellow. "Generative adversarial networks". Em: (2014) (citado nas pgs. 7, 8, 19, 27).

[He 2018] X. He. "Adversarial personalized ranking for recommendation". Em: *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval* (2018) (citado na pg. 17).

[Homanga Bharadhwaj e Lim. 2018] Homin Park Homanga Bharadhwaj e Brian Y. Lim. "Recgan: recurrent generative adversarial networks for recommendation systems". Em: *Twelfth ACM Conference on Recommender Systems (RecSys '18)* (2018), pgs. 4–5 (citado nas pgs. 14, 17, 19–21, 25, 31).

[Koren 2009] Yehuda Koren. "Matrix factorization techniques for recommender systems". Em: *IEEE Computer Society* (2009) (citado na pg. 16).

[LeCun 2016] Yann LeCun. *What are some recent and potentially upcoming breakthroughs in deep learning?* 2016. URL: https://quorasessionwithyannlecun.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning-1 (acesso em 15/12/2021) (citado na pg. 2).

[Liu 2019] W. Liu. "Geo-alm: poi recommendation by fusing geographical information and adversarial learning mechanism". Em: (2019) (citado na pg. 17).

[Luo 2018] Shuyu Luo. *Introduction to Recommender System*. Dez. de 2018. URL: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26 (acesso em 15/12/2021) (citado nas pgs. 1, 9).

[Mirza 2014] Mehdi Mirza. "Conditional generative adversarial nets". Em: (2014) (citado nas pgs. 9, 13).

[Mitchell 1997] Tom M. Mitchell. *Machine Learning*. 1ª ed. 1997 (citado nas pgs. 5, 6).

[Netflix 2006] Netflix. *Netflix Prize data*. 2006. URL: https://www.kaggle.com/netflix-inc/netflix-prize-data (acesso em 15/12/2021) (citado na pg. 21).

[Ouyang 2014] Yuanxin Ouyang. "Autoencoder-based collaborative filtering". Em: *International Conference on Neural Information Processing* (2014), pgs. 284–291 (citado nas pgs. 16, 20, 29).

[Phi 2021] Michael Phi. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. URL: https://scipy-lectures.org/advanced/scipy_sparse/coo_matrix.html (acesso em 23/12/2021) (citado nas pgs. 10, 11).

[Reed 1999] Russell Reed. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. 1ª ed. 1999 (citado na pg. 6).

[Rendle s.d.] Steffen Rendle. "Bpr: bayesian personalized ranking from implicit feedback". Em: () (citado na pg. 17).

[Sarwar 2001] Badrul Sarwar. "Item-based collaborative filtering recommendation algorithms". Em: (2001) (citado na pg. 16).

[Scipy 2021] Scipy. *Coordinate Format (COO)*. URL: https://scipy-lectures.org/advanced/scipy_sparse/coo_matrix.html (acesso em 15/12/2021) (citado na pg. 25).

[Szegedy 2013] Christian Szegedy. “Intriguing properties of neural networks”. Em: (2013) (citado nas pgs. 2, 10, 16).

[J. Wang 2017] J. Wang. “Irgan: a minimax game for unifying generative and discriminative information retrieval models”. Em: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017) (citado na pg. 17).

[K. Wang 2017] Kunfeng Wang. “Generative adversarial networks: introduction and outlook”. Em: (2017) (citado na pg. 8).

[WeiboLiu 2017] WeiboLiu. “A survey of deep neural network architectures and their applications”. Em: (2017) (citado na pg. 11).

[Wu 2017] Chao-Yuan Wu. “Recurrent recommender networks”. Em: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (2017), pgs. 495–503 (citado nas pgs. 10, 11, 20, 29).

[Yu 2019] J. Yu. “Generating reliable friends via adversarial training to improve social recommendation.” Em: (2019) (citado na pg. 17).