37º Simpósio Brasileiro de Bancos de Dados

Tópicos em Gerenciamento de Dados e Informações: Minicursos do SBBD 2022

De 19 a 23 de setembro de 2022

Realização
SBC
Cebd SBC Comissão Especial de Banco de Dados

Patrocínio
neo4j
DuckDB
hurb
stone tech

Agências de fomento
FAPERJ
CAPES

37º Simpósio Brasileiro de Bancos de Dados

37th Brazilian Symposium on Databases

Tópicos em Gerenciamento de Dados e Informações: Minicursos do SBBD 2022

Organizadores:

Ticiana L. Coelho da Silva

Eduardo Ogasawara

Damires Souza

Sérgio Lifschitz

Porto Alegre

Sociedade Brasileira de Computação – SBC

2022

SBBD 2022

BÚZIOS
RIO DE JANEIRO

Organização do SBBD 2022

SBBD 2022 Organization

Program Chair

Damires Souza (IFPB, Brazil)

Short Papers Chair

Daniel Kaster (UEL, Brasil)

Demos and Applications Chair

Elaine Sousa (USP, Brasil)

WTDBD Chair

Eduardo C. de Almeida (UFPR, Brasil)

Short Courses Chair

Ticiana L. Coelho da Silva (UFC, Brasil)

Tutorials Chair

Ricardo Ciferri (UFSCar, Brasil)

Workshops Chair

Carlos Eduardo Pires (UFCG, Brasil)

WTAG Chair

Rafaelli Coutinho (CEFET/RJ, Brasil)

Proceedings Chair

Eduardo Ogasawara (CEFET/RJ, Brasil)

GENERAL ORGANIZATION

SBBD General Chair

Sérgio Lifschitz (PUC-Rio, Brasil)

Realização SBC Cebd Comissão Especial de Banco de Dados
Patrocínio neo4j DuckDB hurb stone tech
Agências de fomento FAPERJ CAPES

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Banco de Dados (37. : 19 – 23 set. 2022 : Búzios)
Tópicos em Gerenciamento de Dados e Informações [recurso eletrônico] : Minicursos do SBBD 2022 / organização: Ticiana L. Coelho da Silva ... [et al.]. Dados eletrônicos. – Porto Alegre: Sociedade Brasileira de Computação, 2022.
82 p. : il. : PDF ; 15MB

Modo de acesso: World Wide Web.
Inclui bibliografia
ISBN 978-85-7669-511-0 (e-book)

1. Computação – Brasil – Simpósio. 2. Banco de Dados. I. Silva, Ticiana L. Coelho da. II. Ogasawara, Eduardo. III. Souza, Damires. IV. Lifschitz, Sérgio. V. Sociedade Brasileira de Computação. VI. Título.

CDU 004.6(063)

Ficha catalográfica elaborada por Jéssica Paola Macedo Müller – CRB-10/2662

Biblioteca Digital da SBC – SBC OpenLib

Índices para catálogo sistemático:

1. Ciência e tecnologia dos computadores : Informática : Dados – Publicação de conferências, congressos e simpósios etc. ... 004.6(063)

Sinopse

O presente livro do XXXVII Simpósio Brasileiro de Bancos de Dados (SBBD 2022) inclui três capítulos escritos pelos autores dos minicursos selecionados e apresentados na edição do evento realizado de 19 a 23 de setembro de 2022. Os capítulos abordam conteúdos relacionados a Processamento de Linguagem Natural, Projeto de Banco de Dados NoSQL e Uso de Meta-learning em Tarefas de Aprendizado Profundo. O comitê de programa de minicursos foi composto pelas professoras Ticiana L. Coelho da Silva (UFC), Damires Yluska de Souza Fernandes (IFPB) e Anne Magály de Paula Canuto (UFRN), sob coordenação da primeira.

A qualidade dessa edição é devida essencialmente aos autores e revisores dos trabalhos submetidos. Expressamos nossos fortes agradecimentos pelas contribuições e discussões durante o SBBD 2022.

Para mais informações sobre o SBBD 2022, visite http://sbbd.org.br/2022, o site desta edição do evento.

Realização
SBC
Cebd SBC Comissão Especial de Banco de Dados
Patrocínio
neo4j
DuckDB
hurb
stone tech
Agências de fomento
FAPERJ
CAPES

Synopsis

This book of the XXXVII Brazilian Symposium on Databases (SBBD 2022) includes three book chapters written by the authors of the selected short courses and presented in the edition of the event held from September 19 to 23, 2022. The chapters address issues related to Natural Language Processing, NoSQL Database Design, and Using Meta-Learning in Deep Learning tasks. The short course program committee was composed of professors Ticiana L. Coelho da Silva (UFC), Damires Yluska de Souza Fernandes (IFPB), and Anne Magály de Paula Canuto (UFRN), under the coordination of the former.

The richness of this issue can be mainly credited to the authors and reviewers. We thank them a lot for their insightful contributions and discussions during SBBD 2022.

You can find more information about SBBD 2022, visiting the http://sbbd.org.br/2022/ website of the event.

Sumário

sbbd:01

Capítulo 1

Processamento de Linguagem Natural

Helena Caseli, Cláudia Freitas e Roberta Viola

Abstract

Natural language processing (NLP) is an interdisciplinary research area, mainly involving Computing and Linguistics, which aims to process verbal, written or spoken language produced by human beings. This chapter focus on processing texts written in Brazilian Portuguese, with the presentation of toolkits and specific resources for that language. Here we will cover pre-processing steps, corpus annotation and generation of representations and language models most used today in important applications present in our daily lives such as conversational agents (chatbots), virtual assistants and semantic analyzers capable of returning, for example, the polarity (valence) of a post on a social network.

Resumo

O processamento de linguagem natural (PLN) é uma área de pesquisa interdisciplinar, envolvendo principalmente a Computação e a Linguística, que visa processar a linguagem verbal, escrita ou falada, produzida pelos seres humanos. Este capítulo foca no processamento de textos escritos em português do Brasil, com a apresentação de toolkits e recursos específicos para esse idioma. Aqui serão abordadas etapas de pré-processamento, anotação de corpus e geração de representações e modelos de linguagem mais utilizados na atualidade em importantes aplicações presentes em nosso dia-a-dia como os agentes conversacionais (chatbots), os assistentes virtuais e os analisadores semânticos capazes de retornar, por exemplo, a polaridade (valência) de uma postagem em uma rede social.

1.1. Introdução

Este minicurso tem como propósito apresentar uma introdução ao Processamento Automático das Línguas Naturais (PLN). O PLN pode ser situado como uma subárea da computação e da linguística (McShane e Nirenburg 2021), onde o objetivo principal é desenvolver modelos computacionais e recursos linguísticos úteis para a automatização do processamento das línguas humanas. Esses modelos podem ser produzidos como uma "atividade fim" para explicar um fenômeno linguístico, ou como uma "atividade meio",

onde visa-se um produto ao final. Neste curso, o foco está no processamento de textos escritos em português do Brasil, utilizando recursos disponíveis livremente, com vistas a gerar um produto de PLN.

No PLN, palavras interessam enquanto unidades portadoras de *sentido*. No entanto, o sentido das palavras é instável, isto é, pode variar ao longo do tempo, do espaço, de grupos de falantes, de diferentes áreas do conhecimento. Por isso, na prática, é limitada a busca por uma representação única e verdadeira do sentido (veja-se por exemplo (Kilgarriff 1997)).

A análise linguística não é linear, embora uma apresentação linear (dos morfemas às palavras, das palavras às frases, das frases aos textos; ou da morfologia à sintaxe, da sintaxe à semântica etc) funcione bem em termos didáticos. A frase abaixo[1] é um bom exemplo desta não linearidade:

> Com estas palavras, André Coruja, [...] além de *quebrar o gelo* **que** havia esfriado o clima, devolveu ao recinto a eloquência necessária para que a sessão continuasse [...]

Para compreender o sentido da frase, precisamos que *quebrar o gelo* seja entendido como uma unidade de sentido (algo próximo a *descontrair*), mas também precisamos que *quebrar o gelo* seja entendido como três palavras distintas, pois apenas o *gelo* havia esfriado o clima. Ou seja, para a semântica, precisamos que *quebrar o gelo* seja uma unidade indecomponível, mas para a sintaxe precisamos da decomposição, pois precisamos de *gelo* com o seu sentido literal.

Semântica e sintaxe são dois dos níveis de processamento das línguas naturais originalmente definidos para operacionalizar o entendimento e a automatização. Contudo, é curioso perceber como essas dimensões da língua se articulam de maneira orgânica no PLN feito nos últimos anos: os vetores de palavras contextuais[2] (*contextual embeddings*) materializam a instabilidade e multiplicidade de sentidos, e o processamento multicamadas, profundo, com a não-linearidade.

Metalinguagens linguísticas (classes como *verbo*, *sujeito*, *agente*) são produtos humanos, situados historicamente e motivados por problemas específicos (que nunca foram o processamento automático de uma língua). Esse aspecto pode funcionar como um estímulo para experimentações relativas à classificação linguística.

Línguas humanas são sistemas complexos, abertos e dinâmicos. De fato, uma das características das línguas que as fazem serem robustas não é a sua precisão, mas, pelo contrário, a sua vagueza. E por vagueza entendemos aqui a existência de limites difusos, pouco precisos, sobretudo no que se refere ao sentido. Um rápido exercício: que palavra nos é mais "útil": *coisa* ou *neurônio*? *Tomar* (tomar um banho, um suco, um porre, uma surra, vergonha, juízo, as dores ...) ou *esverdear*? Línguas humanas sofrem interferências diversas, havendo sempre uma tensão entre o seu caráter regular e

[1] O trecho foi retirado de <http://www.overmundo.com.br/overblog/rock-paraense-no-diva>

[2] Também conhecidos como modelos de linguagem contextualizados (*contextualized language models*) dos quais BERT (Devlin et al. 2019) é um dos representantes mais populares.

previsível (em *desfazer*, *deslegitimar* e *desleal*, temos o elemento *des-* como um indicativo de reversão ou negação, e essa regularidade nos permite produzir e compreender uma palavra que nunca ouvimos antes, como *desestender* em "Preciso desestender a roupa do varal") e seu caráter irregular e idiossincrático (uma pizza *desgostosa* não é uma pizza sem sabor ou ruim).

Regularidade e irregularidade são igualmente parte das línguas, de qualquer língua, como demonstrado empiricamente pela lei de Zipf, nomeada em homenagem ao linguista George Kingsley Zipf (1902–1950). O que esta lei demonstra é que os dados linguísticos (ou, as palavras em um texto) têm uma distribuição desigual: sempre temos poucos casos com muita frequência (regularidade) e muitos casos de frequência baixíssima (irregularidade).

Uma consequência dessa característica é que abordagens de PLN baseadas exclusivamente em regras, por serem definidas com base na regularidade, estão fadadas a deixar muita coisa de fora. Se a intenção é poder trabalhar com um conjunto de textos inéditos, a irregularidade precisa ser considerada e tratada da maneira adequada. Por outro lado, lembrar que há regularidade e previsibilidade ajuda se a ideia for trabalhar em um ambiente de linguagem controlada.

Este minicurso visa, portanto, trazer uma apresentação do PLN nesse novo cenário em que os modelos automáticos baseados em contextos, frequências e, principalmente, redes neurais estão avançando o estado-da-arte no PLN e tornando o processamento de grandes quantidades de textos uma tarefa factível, gerando modelos úteis para além das portas da academia.

Esta seção está organizada como segue. Na seção 1.1.1 são apresentados os níveis originais do PLN. Na seção 1.1.2 serão listadas as principais abordagens utilizadas na área (simbólica, estatística, neural). Na seção 1.1.3 são listados os recursos geralmente utilizados, como os *corpora* (conjuntos de textos), as *word embeddings* e diversos recursos lexicais (como a WordNet). A seção 1.2 descreve as etapas de pré-processamento de textos usualmente realizadas para converter uma sequência de caracteres em informações úteis para aplicações de PLN. A seção 1.3 traz os principais formalismos da atualidade para representação de dados textuais. O aprendizado supervisionado é o tema da seção 1.4, com atenção especial ao processo de preparação dos dados a serem usados nesse aprendizado. Por fim, a seção 1.5 descreve como modelos neurais pré-treinados para o português podem ser refinados para algumas aplicações de PLN.

1.1.1. Níveis do Processamento de Linguagem Natural

O propósito do PLN é tornar os computadores aptos a processar a língua natural, para uma discussão aprofundada veja (McShane e Nirenburg 2021). Por *processar*, aqui, entende-se uma gama de aptidões próprias dos seres humanos que queremos incorporar às máquinas: entender, gerar, extrair conhecimento útil, comunicar-se, entre outros.

A grande quantidade de dados produzida e disponibilizada diariamente, em língua natural, só reforça a relevância e a urgência por processar esses dados e transformá-los em conhecimento. Há uma estimativa de que cerca de 80-90% dos dados produzidos está na forma de dados não estruturados (como áudio, vídeo, foto, texto), sendo as produções

textuais representantes de grande parte desses dados.[3] A esse fato soma-se o de que, cada vez mais, os dados textuais disponíveis são os gerados pelo próprio usuário em postagens de redes sociais e produções textuais despretensiosas e livres de qualquer rigor linguístico, o que traz novos e maiores desafios do que os que existiam nos primórdios do PLN.

Nesses primórdios, o PLN foi definido para ser processado em níveis: (1) Fonético/fonológico, preocupado na especificação e estudo dos sons das línguas; (2) Morfológico, interessado na definição e processamento das unidades linguísticas; (3) Sintático, responsável pela investigação e criação das regras que ordenam as unidades linguísticas e definem como elas se relacionam; (4) Semântico, que visa o estudo dos significados e (5) Pragmático/Discursivo, preocupado com a influência do contexto (e intenção do falante) na linguagem.

Assim, convencionou-se que as pesquisas e seus produtos, no PLN, fossem associados aos níveis acima. Um Reconhecedor de Voz (*Automatic Speech Recognition*, ASR) por exemplo, que é um sistema capaz de converter sinais de voz em texto (palavras), ou sua contraparte que converte as palavras em sinais de voz por meio de um Sintetizador de Voz (também referenciado como *text-to-speech*), são considerados produtos do nível fonético/fonológico apesar de hoje serem gerados por modelos neurais de muitas camadas nas quais as divisões em níveis linguísticos inexistem.

Nesse contexto, determinar a qual nível uma pesquisa ou um produto de PLN pertence tem sido uma tarefa cada vez mais difícil. Na realidade, essa dificuldade sempre existiu, pois a língua não deve ser vista como algo a ser processado dentro de caixinhas bem-definidas e sem sobreposição. Ao se considerar o exemplo do *quebrar o gelo* apresentado anteriormente, ficou claro que a divisão entre morfologia, sintaxe e semântica é muito nebulosa e não determinística uma vez que não é uma tarefa fácil se determinar onde começa e termina uma unidade linguística (papel da morfologia), dado o contexto, (papel da sintaxe) a fim de se determinar o significado naquela ocorrência (papel da semântica).

1.1.2. Abordagens mais utilizadas no Processamento de Linguagem Natural

Outra caracterização bastante usual nas pesquisas no PLN é a de associá-las a uma das abordagens mais utilizadas: simbólica (linguística), estatística ou neural.

A abordagem simbólica foi a primeira a ser proposta, quando o PLN surgia juntamente com as primeiras implementações computacionais a partir de 1950. Naquela época, os especialistas tinham em mãos seus conhecimentos sobre o processamento da língua natural e utilizaram meios simbólicos de expressar explicitamente esse conhecimento em formatos que as máquinas pudessem processar. Nesse momento, surgiram os léxicos de palavras, as primeiras gramáticas processáveis por computador e algumas bases de conhecimento.

Com os avanços na disponibilização de dados textuais, em especial os primeiros *corpora*[4], dados começaram a surgir em abundância permitindo que a estatística fosse usada para o reconhecimento de padrões (baseado em frequência e probabilidade) que

[3]<https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data>

[4]*Corpora* é o plural de *corpus*. *Corpus* é o nome que se dá a uma coleção de textos.

deram origem aos primeiros modelos estatísticos. Diversos algoritmos de aprendizado de máquina (AM) foram empregados para a geração dos modelos estatísticos. Hoje, tais modelos são denominados de modelos de AM tradicionais ou *shallow*. É desta época a primeira versão do Google Tradutor[5], que foi estatístico de 2006 a 2016 até ser substituído pela versão neural atualmente em uso. De fato, a abordagem tradicional de AM reinou soberana entre os anos de 1980 e 2010 quando as redes neurais artificiais roubaram a cena em diversas aplicações computacionais e também no PLN.

A abordagem neural (*deep learning* ou aprendizado profundo) é hoje a mais usualmente aplicada para se alcançar resultados de estado-da-arte no PLN. Embora, em sentido estrito, a abordagem neural também seja estatística, convencionou-se tratá-las como abordagens distintas. Isso ocorre porque, mesmo também sendo utilizado para reconhecer os padrões recorrentes, o aprendizado profundo lida com várias camadas de unidades de processamento capazes de aprender a partir de grandes quantidades de dados sem que sejam explicitamente programadas. Esse fato faz com que, na prática, o aprendizado profundo seja considerado uma abordagem distinta da estatística. Em outras palavras, enquanto os algoritmos tradicionais de AM especificam como o aprendizado deve ocorrer, nas redes neurais esse aprendizado é realizado em camadas de unidades de processamento (neurônios artificiais) de modo que não é possível saber, ao certo, qual dado de entrada levou a qual padrão aprendido. Além da falta de interpretabilidade do modelo, o poder computacional e a quantidade de dados que os modelos neurais demandam para se alcançar resultados tão impressionantes são limitantes para diversas aplicações e instituições.

Assim, uma alternativa que tem sido empregada com bastante sucesso na atualidade é utilizar modelos neurais pré-treinados a partir de quantidades de dados e poder computacional não disponíveis a todos, e realizar um refinamento (*fine-tuning*) de tais modelos para tarefas específicas para as quais dados e poder computacional são considerados acessíveis.

1.1.3. Recursos mais comuns no Processamento de Linguagem Natural

No aprendizado de máquina, seja ele *shallow* ou *deep*, um recurso essencial são os dados usados no treinamento dos modelos. No caso do PLN, o principal tipo de dado utilizado é um *corpus*. Um *corpus* é uma coleção de textos, por exemplo, um conjunto de *tweets* ou uma coletânea de obras literárias.[6]

Um *corpus* pode ser: comparável, quando trata de uma coleção de textos que versam sobre um mesmo assunto; paralelo, quando trata-se de versão em uma língua e sua(s) tradução(ões) para outra(s) língua(s); ou alinhado, quando além de ser paralelo o *corpus* possui indicações de qual sentença ou palavra fonte (no texto original), por exemplo, é a tradução de qual palavra ou sentença alvo (no texto traduzido). Além disso, também pode-se tipificar o *corpus* em relação à quantidade de línguas que ele abrange como: monolíngue, bilíngue ou multilíngue. Um exemplo de *corpus* paralelo português-inglês e português-espanhol é a coletânea de artigos de divulgação científica da revista Pesquisa

[5]<https://translate.google.com.br/>

[6]Grafias possíveis para a palavra *corpus* são sua versão no latim, em itálico (cujo plural é *corpora*) ou sua versão acentuada córpus que serve tanto para o singular quanto para o plural, em português.

FAPESP[7] conhecido como Corpus FAPESP[8] (Aziz e Specia 2011). Outros *corpora* para o português também estão disponíveis em repositórios como o Metatext[9].

Outro recurso bastante útil no PLN na atualidade são as *word embeddings* ou vetores de palavras. As *embeddings* são representações vetoriais estáticas onde cada palavra é representada como um ponto em um espaço *n* dimensional. Por exemplo, *embeddings* de 300 dimensões usam 300 valores reais para representar uma dada palavra. Essas representações podem ser aprendidas a partir de *corpus* por meio de contagem de frequências – como o GloVe (Pennington et al. 2014) – ou modelos neurais – como o Word2Vec (Mikolov et al. 2013). Por meio dessas representações vetoriais é possível encontrar similaridades sintáticas e semânticas inferidas no aprendizado com base no contexto de ocorrência das palavras no *corpus* usado no treinamento. Algumas *word embeddings* podem ser encontradas no repositório do NILC[10] (Núcleo Interinstitucional de Linguística Computacional).

Um avanço em relação às representações estáticas que são as *word embeddings* são os modelos de linguagem contextualizados como o BERT (Devlin et al. 2019), RoBERTa (Liu et al. 2019) e GPT-3 (Brown et al. 2020). Nesse caso, diferente das *word embeddings* onde cada palavra só possui um vetor que a representa, cada ocorrência da palavra tem sua própria representação. Uma das principais vantagens dos modelos de linguagem contextualizados é que palavras polissêmicas como *banco* podem ter seus diferentes significados preservados, enquanto nos vetores estáticos essa polissemia se perde quando apenas uma representação (a mais frequente) é preferida pelo modelo. Um dos modelos de linguagem mais famosos para o português do Brasil é o BERTimbau (Souza et al. 2020) disponível no repositório do HuggingFace[11].

Além dos *corpora* e dos recursos derivados deles por meio de aprendizado de máquina – *word embeddings* e modelos de linguagem contextualizados – há uma ampla gama de recursos linguístico-computacionais simbólicos e muito ricos, como as wordnets.

As wordnets são, como o nome indica, redes (*nets*) de palavras (*words*). Wordnets são consideradas bases de dados *lexicais*, porque contêm informações relacionadas ao léxico de uma língua[12]. Diferentemente de dicionários, que também lidam com palavras, wordnets não estão organizadas por palavras, mas por *synsets*. Synsets são conjuntos (*sets*) de sinônimos (*synonyms*) que se relacionam entre si por meio de relações semânticas como hiperonímia e hiponímia (relações hierárquicas), meronímia (parte – todo), dentre outras. As palavras *berrar* e *gritar*, por exemplo, integram um mesmo *synset*. Este *synset*, por sua vez, não é representado por uma outra palavra ou conceito, mas por um código. Por exemplo:

```
00912473-v {berrar, gritar}
```

O *synset* acima, por sua vez, é hipônimo – isto é, está um nível abaixo – do *synset*

[7]<https://revistapesquisa.fapesp.br/>

[8]<http://www.nilc.icmc.usp.br/nilc/tools/Fapesp%20Corpora.htm>

[9]<https://metatext.io/datasets-list/portuguese-language>

[10]<http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc>

[11]<https://huggingface.co/neuralmind/bert-base-portuguese-cased>

[12]O léxico é a parte da língua que diz respeito às palavras. De forma simplificada, podemos entender uma língua como a combinação de um léxico (as palavras) e regras (a gramática).

mais genérico:

```
00941990-v {proferir, falar, dizer, verbalizar, conversar}
```

Criada originalmente para a língua inglesa (a WordNet de Princeton)[13], existem atualmente iniciativas de diferentes línguas para diferentes wordnets, e um dos desafios é manter o alinhamento entre os *synsets* dessas diferentes línguas. A língua portuguesa tem algumas wordnets ou recursos similares, como OpenWordNet-PT[14], OntoPT[15], PULO[16], TeP[17], PAPEL[18]. (Santos et al. 2010) e (Oliveira et al. 2015) trazem comparações entre os recursos e seus dados podem ser livremente utilizados (veja-se a página de cada projeto para mais informações).

1.2. Pré-processamento: De caracteres a tokens e "palavras"

O texto pode ser visto, inicialmente, como uma sequência de caracteres, ou seja, de símbolos reconhecidos por humanos e computadores. A combinação desses caracteres é que dá forma ao conhecimento por eles representado. Caracteres são combinados em *tokens* (sequências de caracteres delimitados por espaços) que podem ser unidades linguísticas como as palavras.

Na junção de caracteres para a formação de *tokens* uma ferramenta computacional simples e útil são as expressões regulares. Por meio das expressões é possível definir padrões de caracteres capazes de interpretar a sequência de entrada da maneira desejada e extrair dela *trechos* relevantes para o processamento que se deseja. Por exemplo, em um *chatbot*, as expressões regulares podem ser usadas para identificar palavras-chave, como em (1), ou padrões correspondentes a um número de pedido (2) ou a uma data (3)[19]:

(1) `reclamação|quebrado|trocar`

(2) `[A-Z]{3}-[0-9]{3}-[0-9]{3}`

(3) `[0-9]{2}/[0-9]{2}{/[0-9]{4}}?`

Uma das etapas do PLN que se beneficia das expressões regulares é a tokenização. A tokenização (*tokenization*) é o processo de transformar a sequência de caracteres de entrada em unidades que façam sentido para o processamento: os *tokens*. No caso da língua portuguesa, em que o critério gráfico de delimitação costuma funcionar bem[20], palavras são geralmente delimitadas por espaço em branco ou símbolos de pontuação (, - ; : etc.) esse processo não é tão complexo quanto línguas aglutinativas, como o alemão,

[13]<http://wordnet.princeton.edu/>

[14]<http://wn.mybluemix.net/>

[15]<http://ontopt.dei.uc.pt/>

[16]<http://wordnet.pt/>

[17]<http://www.nilc.icmc.usp.br/tep2/index.htm>

[18]<https://www.linguateca.pt/PAPEL/>

[19]Em uma expressão regular o símbolo "|" separa as opções; "[" e "]" delimitam grupos de caracteres; "{" e "}" são usados para agrupar um padrão ou, quando há um número *n* entre eles, *n* indica o número de vezes que o padrão que os antecede se repete e "?" indica que o padrão que o antecede pode ou não ocorrer.

[20]Dissemos "costuma funcionar bem" porque o critério gráfico baseia-se na ortografia, e a ortografia é conhecidamente a dimensão mais arbitrária de uma língua, como vemos em *embaixo* (uma palavra) e *em cima* (duas palavras).

ou nas quais a delimitação não é tão explícita.

No exemplo apresentado na seção 1.1, os *tokens* seriam como os listados a seguir:

Com	estas	palavras	,	André	Coruja	,
além	de	quebrar	o	gelo	que	havia
esfriado	o	clima	,	devolveu	ao	recinto
a	eloquência	necessária	para	que	a	sessão
continuasse	.					

Um conceito relacionado ao de *token* é o *type*. Enquanto *token* se refere a cada ocorrência, o *type* contabiliza a quantidade de *tokens* únicos independente do número de ocorrências no *corpus*. Assim, por exemplo, no exemplo anterior, apesar de termos 30 *tokens*, há apenas 25 *types*, pois os *tokens* ",” (3 ocorrências), “o” (2 ocorrências), “que” (2 ocorrências) e “a” (2 ocorrências) são contabilizados apenas uma vez na contagem de *types*. Desse modo, a quantidade de *types* é usada como um indicativo do tamanho do vocabulário de um *corpus* e, consequentemente, de sua cobertura linguística.

Alcançar 100% de cobertura linguística em uma aplicação de PLN é algo utópico, porque o léxico de uma língua é muito amplo e novas palavras surgem em uma velocidade muito maior do que a de atualização dos recursos e modelos. Contudo, muitas das raras e novas palavras compartilham semelhanças com as mais frequentes e já existentes. Essas semelhanças são informações valiosas para se determinar o significado da rara/nova palavra. No exemplo apresentado na seção 1.1 para a palavra *desestender*, a regularidade do uso do elemento *des* como um indicativo de reversão ou negação traz informações valiosas sobre a palavra como um todo.

Considerando essa característica intrínseca das línguas naturais, uma estratégia bastante utilizada pelos modelos de linguagem contextualizados da atualidade é a de dividir uma palavra desconhecida em partes conhecidas no que se acostumou chamar de tratamento de subpalavras (*subwords*). Por exemplo, a técnica de *Byte-Pair Encoding* (BPE), inspirada nas técnicas de compressão de dados, junta os pares de caracteres mais frequentes até ter um vocabulário de tamanho desejado. O tokenizador é treinado com base no conjunto de sequências de caracteres mais frequentes e caracteres restantes, e usado para tokenizar novos textos. Outra técnica de tratamento de subpalavras é o WordPiece usado pelo BERT. Nesse caso, ao invés da frequência, escolhe-se juntar o par de símbolos que maximiza a probabilidade. Seguindo uma abordagem *bottom-up* diferente da dos demais, a Unigram inicializa um vocabulário base com uma grande quantidade de caracteres, sub-palavras e palavras e, durante o treinamento, o vocabulário é progressivamente reduzido, mantendo somente os símbolos mais relevantes, até obter o tamanho desejado.

Subsequente (ou juntamente) à tokenização podem ocorrer outros processamentos que consideram os *tokens* como *lexemas*: a lematização e a radicalização. Um lexema é uma unidade abstrata de significado que corresponde a um conjunto de formas relacionadas. Por exemplo, *bonito*, *bonita*, e *bonitos* são exemplos de lexemas. O *lema* é a forma canônica, dicionarizada, escolhida por convenção para representar um lexema. No caso dos lexemas listados anteriormente, o lema é *bonito*. O *radical* é a palavra básica, sem afixos flexionais, como é o caso de *bonit* para os exemplos de lexemas apresentados

anteriormente.

Esses processamentos podem ser relevantes ou não, depende da aplicação. Para algumas aplicações de PLN não é interessante saber a forma flexionada da palavra (o que aumenta considerável e às vezes desnecessariamente o tamanho do vocabulário), mas sim apenas sua forma canônica (lema) ou raiz. Já para outras aplicações, os resultados são melhores quando se considera todas as formas superficiais (flexionadas) do *corpus* sem quaisquer processamentos adicionais além da tokenização.

No exemplo apresentado na seção 1.1, os lemas correspondentes aos *tokens* retornados pelo processamento com o Spacy[21], uma das ferramentas mais utilizadas para pré-processamento de textos, são[22]:

Com	**este**	**palavra**	,	André	Coruja	,
além	de	quebrar	o	gelo	que	**haver**
esfriar	o	clima	,	**devolver**	ao	recinto
o	eloquência	**necessário**	**parir**	que	**o**	sessão
continuar	.					

Por fim, outro conceito relacionado ao processamento de *tokens* é o de *stopwords*. Em diversas aplicações de PLN é interessante desconsiderar algumas palavras que pouco acrescentam ao conteúdo do texto, como preposições, determinantes, conjunções etc. Essas palavras são conhecidas como *stopwords*. É importante notar que, além das palavras clássicas mencionadas, as *stopwords* podem variar de acordo com o contexto específico com o qual estamos trabalhando. Por exemplo, se estivermos tratando um *corpus* de SMS com o intuito de extrair informações acerca das operadoras e assuntos mais comumente abordados por esse meio de comunicação, termos como "SMS" ou "mensagem" se tornam irrelevantes para a análise, dada sua alta frequência e baixa relevância. Assim, tratá-los como *stopwords* nessa análise específica pode servir como uma boa estratégia para a tarefa/aplicação de PLN em questão.

No exemplo apresentado na seção 1.1, os *tokens* resultantes após desconsiderar as *stopwords* comumente definidas no português, novamente usando o Spacy, seriam como os listados a seguir:

–	–	palavras	–	André	Coruja	–
–	–	quebrar	–	gelo	–	havia
esfriado	–	clima	–	devolveu	–	recinto
–	eloquência	necessária	–	–	–	sessão
continuasse	–					

O código usado para esses processamentos usando o Spacy, e processamentos similares usando o NLTK[23], está disponível em: <https://github.com/brasileiras-pln>

Assim como a utilização do lema ou da forma flexionada depende da aplicação, a eliminação de *stopwords* depende do que se pretende com o texto. *Dar tempo* é diferente de *dar um tempo* e *camisa de força* é diferente de *camisa* e *força* usados isoladamente.

[21] <https://spacy.io/>

[22] Note que o Spacy lematizou equivocadamente a preposição *para* como se fosse o verbo *parir*.

[23] <https://www.nltk.org/>

Além disso, é de suma importância a consideração da língua específica com a qual se está trabalhando e suas idiossincrasias. A maioria dos recursos disponíveis para tratamento e processamento dos *corpora* se baseia principalmente em línguas de maior prestígio socioeconômico, como o inglês, de modo que devemos entender a aplicação de tais recursos e adaptá-los, se necessário, às características da língua com a qual estivermos lidando.

Ressaltamos que os processamentos que apresentamos aqui podem servir como ferramentas bastante úteis para tarefas de PLN. No entanto, devemos sempre lembrar que estamos trabalhando com as línguas humanas, ou seja, com um material que produz significados e sentidos na comunicação e que deve ser entendido com todas as sua complexidades e peculiaridades. Portanto, tais processamentos não devem ser aplicados como uma rotina de tratamento prescritiva e rígida. O trabalho do especialista em PLN com a análise qualitativa dos dados e objetivos é indispensável para a tomada de decisão em cada uma das etapas da construção dos modelos, desde a escolha do *corpus* e seu pré-processamento.

1.3. Representação de palavras: *one-hot-encoding*, *word embeddings* e modelos de linguagem contextualizados

Avançando um pouco mais no nosso entendimento do que é o PLN, nesta seção falaremos sobre as principais formas de representação do conteúdo textual, todas relacionadas ao processamento das unidades linguísticas considerando seus contextos e suas frequências de ocorrência.

O tipo mais tradicional de representação de palavras é o *one-hot encoding*. Nesse tipo de representação, uma palavra é representada em um vetor cuja dimensão é a mesma do tamanho do vocabulário que ela representa. Cada palavra é, então, representada como um vetor binário (contendo 0s e 1s) que tem 1 em apenas uma posição. A Figura 1.1 ilustra essa ideia. Para representar uma sentença, as posições das palavras que ocorrem na sentença recebem o valor 1 enquanto as demais ficam com valor 0.

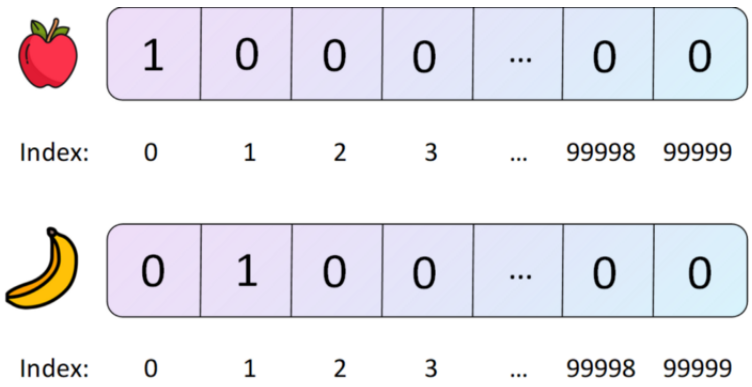


Figura 1.1. Representação vetorial *one-hot encoding*

Fonte: <https://towardsdatascience.com/deep-learning-for-nlp-word-embeddings-4f5c90bcdab5>

Uma das grandes limitações desse tipo de representação é sua esparsidade, uma

vez que, para cada palavra, apenas uma posição será preenchida com 1 desperdiçando, assim, as demais posições que poderiam ser usadas para representar outros traços dessa palavra. Outra limitação é que, em uma representação *one-hot encoding*, a similaridade entre palavras semanticamente próximas como *maçã* e *banana* não é facilmente capturada pois o contexto de ocorrência similar entre elas não é representado no vetor.

A necessidade de considerar o contexto na geração das representações de palavras surgiu com a *hipótese distributiva* formulada na década de 1950 por Joss (1950), Harris (1954) e Firth (1957) apud (Jurafsky e Martin 2021). De acordo com essa hipótese, assume-se que palavras semelhantes ocorrem em contextos similares. Foi com base nesta hipótese que surgiram representações vetoriais (*embeddings* ou vetores de palavras).

Os primeiros modelos vetoriais (também conhecidos como modelos distribucionais) se baseavam em matrizes de co-ocorrência que relacionam ocorrências de termos com documentos ou outros termos, como: (i) a matriz de frequência termo-documento, que associa a frequência de ocorrência de termos em documentos (sentenças); (ii) a matriz de frequência termo-termo, que associa a frequência de ocorrência entre termos e (iii) a matriz TF-IDF, que representa um termo em função da relação de sua frequência de ocorrência (TF, *term frequency*) e a frequência inversa de documentos nos quais ele ocorre (IDF, *inverse document frequency*). Embora tenham trazido avanços na representação ao se considerar o contexto de ocorrência das palavras, esses modelos vetoriais tradicionais ainda são esparsos (contêm muitos 0s) e não escaláveis (à medida que o número de documentos e o vocabulário cresce, a dimensão das matrizes torna-se um gargalo).

Como alternativa, surgiram as formas de representação contínuas (também conhecidas como vetores densos) que são vetores numéricos de dimensão menor do que o tamanho vocabulário (geralmente entre 50 e 1000) com valores reais em cada uma dessas posições. Por meio desses vetores é possível realizar cálculos espaciais facilitados para encontrar a similaridade entre palavras. Essas representações têm sido as mais utilizadas na atualidade devido aos bons resultados em aplicações de PLN que, segundo (Jurafsky e Martin 2021), se devem a sua capacidade de generalização (evitando o *overfitting*) e captura de sinônimos. Tais representações podem ser divididas em: estáticas (mais conhecidas como *word embeddings*) e dinâmicas (modelos de linguagem contextualizados). Enquanto os modelos de representação estática aprendem uma *embedding* fixa para cada palavra no vocabulário, os modelos dinâmicos geram um vetor para cada contexto de ocorrência das palavras.

Assim como as demais representações vetoriais, as *word embeddings* são aprendidas a partir de um grande *corpus* por meio de métodos de contagem, como o GloVe (Pennington et al. 2014), ou neurais, como o Word2Vec (Mikolov et al. 2013) e o FastText (Bojanowski et al. 2016). Na geração das *word embeddings*, os contextos de ocorrência das palavras vão sendo considerados para se calcular os valores numéricos que populam cada uma das n dimensões usadas na representação. Idealmente, é como se cada uma das dimensões representasse algum traço semântico/sintático da palavra sendo representada. A Figura 1.2 ilustra essa ideia.

Neste exemplo, traços semânticos – ser vivo (*living being*), felino (*feline*), humano (*human*) e realeza (*royalty*) – e sintáticos – gênero (*gender*), verbo (*verb*) e plural – aparecem como significado idealizado para as dimensões usadas na representação das

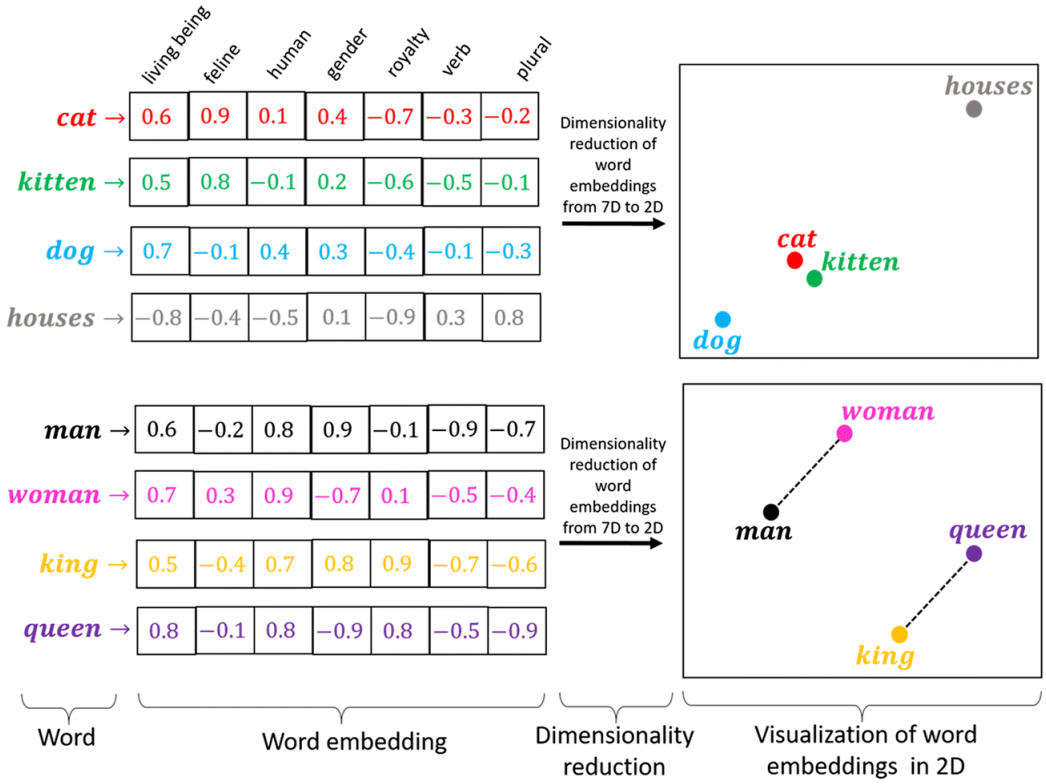


Figura 1.2. Representação vetorial *word embedding*

Fonte: <https://medium.com/@hari4om/word-embedding-d816f643140>

palavras *gato* (*cat*), *gatinho* (*kitten*), *cachorro* (*dog*) e *casas* (*houses*). Com base na distância espacial entre as representações dessas palavras é possível recuperar a similaridade entre palavras próximas, como *gato* e *gatinho*, por exemplo. Vale ressaltar, contudo, que na prática não é possível saber ao certo o que está sendo representado ou capturado por cada uma das dimensões das *word embeddings*.

Códigos para manipulação e criação de *word embeddings* estão disponíveis em: <https://github.com/brasileiras-pln>

Embora tenham representado um grande avanço na área de PLN, as *word embeddings* também têm limitações. A principal delas é conhecida como confluência de significados. Uma vez que apenas uma representação é gerada para a mesma forma superficial de uma palavra, os diversos significados dessa mesma forma superficial são "misturados" na mesma *embedding* perdendo-se, assim, informação linguística valiosa.

Assim, em 2017, com o surgimento da arquitetura neural *transformer* proposta inicialmente para modelos de tradução automática, diversos modelos de linguagem contextualizados (ou vetores de palavras dinâmicos ou *embeddings* contextualizadas) foram propostos. Nesses modelos, cada palavra é representada por um vetor diferente cada vez que um contexto de ocorrência diferente é detectado, ou, como bem colocado por (Jurafsky e Martin 2021): "enquanto os vetores estáticos representam os significados dos *types* (entradas do vocabulário), os vetores dinâmicos representam os significados dos *tokens* (instâncias de um *type* um contexto específico)". São tantas as arquiteturas neurais

propostas para gerar esses modelos que a Figura 1.3 provavelmente já está defasada.

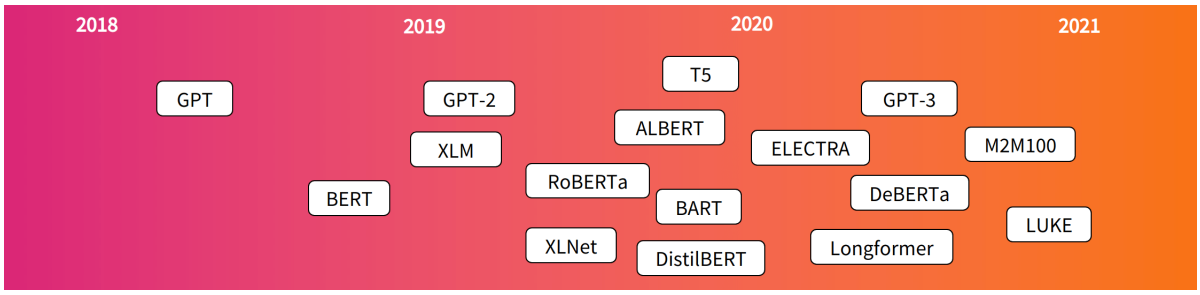


Figura 1.3. Linha do tempo dos modelos *transformers* mais populares em Março 2022

Fonte: <https://huggingface.co/blog/bert-101>

O PLN usando modelos de linguagem contextualizados envolve duas etapas: pré-treinamento e refinamento ou ajuste fino (*fine-tuning*). No pré-treinamento, imensas quantidades de dados e muito poder computacional (em máquinas dotadas de várias GPUs) são usados para gerar um modelo computacional capaz de abstrair características da linguagem. O pré-treinamento é, portanto, o processo de aprender algum tipo de representação de significado das palavras (ou sentenças) a partir do processamento de grandes *corpora* (Jurafsky e Martin 2021). Esse pré-treinamento é realizado de modo auto-supervisionado usando estratégias como *Masked Language Modeling* (MLM) ou *Next Sentence Prediction* (NSP).

No MLM o objetivo é prever uma palavra que foi artificialmente substituída por um *token* especial: `[MASK]`. Por exemplo, considerando o exemplo da seção 1.1, a palavra *quebrar* poderia ser mascarada como apresentado a seguir:

> Com estas palavras, André Coruja, além de `[MASK]` o gelo que havia esfriado o clima ...

Assim, a partir de um grande *corpus*, alguns *tokens* são selecionados aleatoriamente para serem mascarados, outros são substituídos por *tokens* aleatórios e os demais mantidos sem alteração. No BERT, 15% dos *tokens* no *corpus* de treinamento são selecionados para esse processo. Destes, 80% são mascarados, 10% são substituídos por *tokens* aleatórios e 10% permanecem inalterados (Jurafsky e Martin 2021). O objetivo da estratégia de MLM é tenta prever qual é a palavra que foi mascarada considerando os contextos de ocorrência tanto à esquerda quanto à direita (*bidirectional encoder*). Essa mesma ideia pode ser estendida para mascarar sequências de *tokens* como ocorre no SpanBERT (Joshi et al. 2020).

Enquanto o foco da estratégia de treinamento MLM é predizer as palavras com base em suas vizinhanças (contexto à esquerda e à direita), a NSP visa prever a próxima sentença. Essa estratégia é bastante útil para aplicações de PLN nas quais o relacionamento entre as sentenças é importante, como na detecção de paráfrases. Na NSP, o pré-treinamento é feito a partir de pares de sentenças que estão relacionadas entre si (como sentenças adjacentes) ou não. Segundo (Jurafsky e Martin 2021), no BERT, 50% dos pares de treinamento são positivos e os outros 50% são pares gerados aleatoriamente.

Nesse caso, o objetivo é dizer se o par de sentenças é um par positivo ou não. Para tanto, dois novos *tokens* especiais são usados: `[CLS]`, que indica o início do par de sentenças; e `[SEP]`, que é usado para separar as sentenças e marcar o final do par. Adaptando o exemplo da seção 1.1, poderíamos ter algo do tipo:

> `[CLS]` Com estas palavras, André Coruja, além de quebrar o gelo que havia esfriado o clima, `[SEP]` devolveu ao recinto a eloquência necessária para que a sessão continuasse. `[SEP]`

Vale mencionar que essas estratégias são de aprendizado não supervisionado, ou seja, os dados de entrada são dados puros (brutos), sem qualquer tipo de anotação (rotulação) da tarefa de PLN que se deseja aprender/modelar. Diversos modelos de linguagem pré-treinados estão disponíveis no Hugging Face[24].

A partir de um modelo pré-treinado o que se faz é realizar o ajuste fino (*fine-tuning*) para uma tarefa específica usando um conjunto menor de dados rotulados. Para (Jurafsky e Martin 2021), o *fine-tuning* é um processo que continua (estende) o pré-treinamento de um modelo de linguagem contextualizado, geralmente adicionando camadas neurais de classificação e/ou ajustando parâmetros, a fim de realizar alguma tarefa fim como etiquetagem morfossintática ou análise de sentimentos. Exemplos de *fine-tuning* de modelos pré-treinados para o português do Brasil, em tarefas de PLN, são apresentados na seção 1.5.

1.4. Aprendizado supervisionado

Nesta seção falaremos de uma importante parte do PLN que é o enriquecimento do conjunto de treinamento (*corpus*) por meio da anotação. A anotação é feita para evidenciar algum fenômeno linguístico que se deseja identificar/processar automaticamente por meio de um modelo computacional.

1.4.1. Anotação de *corpus* e a criação de *datasets* linguísticos

Chamamos de *datasets* (linguísticos), ou *corpora* anotados padrão ouro, a uma coleção de documentos (textos) anotados. Anotar um texto, por sua vez, significa adicionar informação linguística a palavras ou porções de textos maiores. Esta informação linguística pode ser de natureza bastante variada, e alguns tipos mais comuns de anotação são:

- classes de palavras (chamada anotação de POS, do inglês *part of speech*): anotação que atribui a cada palavra uma etiqueta do tipo substantivo, verbo, adjetivo etc.
- sintaxe: anotação que atribui a cada palavra uma etiqueta do tipo sujeito, objeto direto, adjunto adverbial etc.
- polaridade: anotação que atribui a cada palavra ou conjunto de palavras uma etiqueta do tipo positivo, negativo ou neutro conforme o seu valor semântico/cultural: *defeito*, *cabeça dura*, *quebrar* e *doença* são negativos; *amor*, *perfeito*, *adorar* e *inteligente* são positivos, e *surpresa*, *computador* e *normal* são neutros.

[24]<https://huggingface.co/blog/bert-101>

- entidades nomeadas: anotação que atribui a cada palavra ou conjunto de palavras uma etiqueta semântica, que pode ser genérica (como PESSOA, LUGAR, ORGANIZAÇÃO) ou específica (como ROCHA para termos como anidrita, basalto e folhelho considerando entidades do domínio geológico).

- correferência: anotação que relaciona duas formas que se referem à mesma entidade. Na frase abaixo[25] os índices *i* e *ii* sinalizam as correferências, que podem estar restritas a uma frase ou levar em conta parágrafos ou o texto inteiro

  > A tenista americana *Serena Williams*$_i$, de 40 anos, anunciou na terça-feira (9) que vai se aposentar das quadras [...]. Apesar de não deixar claro a data de aposentadoria, *ela*$_i$ sugeriu que o *US Open*$_{ii}$, que ocorre entre 29 de agosto e 11 de setembro deste ano, pode ser o último *torneio*$_{ii}$ da sua vitoriosa carreira profissional.

- anotação de relações: similar à anotação de correferência, mas leva em conta uma variedade de relações semânticas (inclusão, localização etc), e não apenas a relação de *identidade*;

- anotação de similaridade semântica: anotação que indica o grau de semelhança entre duas porções de um texto – por exemplo, entre duas frases. Em geral, o grau de semelhança pode ser de três tipos:

  1. acarretamento - o sentido de uma frase está incluído no sentido de outra.
  2. contradição - o sentido de uma frase contradiz o sentido de outra.
  3. neutro - a frase hipótese pode ser verdadeira dada a frase premissa.

Datasets codificam o desempenho humano em uma determinada tarefa. Por isso, para que um *dataset* seja útil no aprendizado automático, é importante que a anotação nele contida tenha sido feita (ou revista) por pessoas. Por isso, também é possível fazer referência a este tipo de material como um *corpus* padrão-ouro (*gold standard*).

1.4.2. Relevância de datasets no aprendizado supervisionado

Um dataset – ou *corpus* anotado – é um recurso linguístico. E sua relevância no aprendizado supervisionado se deve a três motivos.

O aprendizado de máquina precisa de exemplos do que será aprendido. E apesar dos avanços das redes neurais e do aprendizado profundo, bons exemplos continuam necessários, com a vantagem de agora ser possível oferecer aos algoritmos uma quantidade menor de exemplos, graças ao ajuste fino (*fine-tuning*) apresentado na seção 1.3.

O segundo motivo que torna *datasets* fundamentais em diversas tarefas de PLN é que eles facilitam o processo de avaliação de um sistema e de comparação entre sistemas. Isto porque se a anotação codifica, no *corpus*, a compreensão humana sobre algo, e o que queremos das máquinas em certas tarefas é que elas reproduzam esta compreensão

[25] O trecho foi retirado de <https://www.nexojornal.com.br/expresso/2022/08/11/O-que-Serena-Williams-fez-que-nenhum-outro-tenista-conseguiu>

humana, então a melhor maneira de saber o quão bom é um resultado é comparando-o com o entendimento humano.

Por fim, tecnologias /algoritmos podem ser independentes de língua, mas recursos linguísticos, não. Ou seja, para garantir um bom aprendizado em uma tarefa de identificação de entidades, por exemplo, adianta pouco treinar o modelo que seja efetivo para a língua portuguesa em um *dataset* criado para uma língua diferente do português.

1.4.3. Criação de um *dataset* linguístico: planejamento e esquema de anotação

Um dos primeiros passos na criação de um *dataset* linguístico é a definição do conjunto de etiquetas que será usado na anotação. O conjunto de etiquetas de anotação é chamado *tagset*, e subjacente a um *tagset* há um esquema de anotação, isto é, uma forma de descrever e organizar as classes (etiquetas) que o compõem.

O conjunto de etiquetas pode vir diretamente de uma teoria ou pode apenas se inspirar em algum modelo teórico; pode ser a simplificação de uma teoria ou a convergência de diferentes teorias que se debruçam sobre o mesmo fenômeno. Além disso, as etiquetas podem ser criadas tendo em vista uma determinada tarefa/aplicação em mente, como a anotação de polaridades e a anotação de entidades. Assim, a anotação pode codificar tanto aspectos estritamente linguísticos (como classes de palavras) como aspectos mais genéricos e dependentes de uma boa capacidade de interpretação de textos (caso da anotação de polaridades) ou o conhecimento específico de algum domínio, como geologia, direito ou medicina. Neste caso, a participação de especialistas das respectivas áreas nos projetos de anotação é crucial.

Excetuando-se o cenário em que todas as categorias de anotação já estão dadas de antemão por alguma teoria, e as situações em que se deseja replicar (para uma outra língua ou para um outro *corpus*) uma determinada anotação que já existe, nos demais contextos será necessário criar um esquema de anotação.

Definir um conjunto de etiquetas (um conjunto de classes) e sua forma de aplicação reflete uma maneira de ver a tarefa. Por isso, quanto mais bem definido o problema (a tarefa), mais chances de sucesso. Caso seja necessário criar um esquema de anotação, devemos logo responder às seguintes perguntas:

- Qual o objetivo da anotação?
- A que ela serve?

Um aspecto importante de um esquema de anotação é que ele favoreça a generalização. A terminologia de uma área, embora contenha termos específicos daquele domínio, não é exatamente um conjunto de entidades desse mesmo domínio: é importante que esses diferentes termos sejam agrupados em classes mais amplas, ou seja, os termos podem ser instâncias de categorias de anotação. Por exemplo, “conglomerado”, “lamito” e “arenito” são termos da Geologia. Na anotação da frase (a) precisaremos de uma classe ampla que os agrupe e generalize – por exemplo, uma classe ROCHA.

> A formação é constituída por conglomerados, arenitos conglomeráticos, arenitos e lamitos.

O processo de anotação segue as seguintes etapas:

1. Levantamento bibliográfico sobre o que já existe relacionado à questão, em termos teóricos (descrição linguística, por exemplo) e aplicados (existem anotações do mesmo tipo, ou diretamente relacionadas? Quais os problemas enfrentados?);
2. Elaboração de um esquema ou modelo de anotação, que contém as primeiras generalizações acerca do fenômeno observado, isto é, a primeira proposta de etiquetas (categorias);
3. Aplicação dessas etiquetas a uma amostra mais ampla;
4. Refinamento progressivo do esquema de anotação;
5. Observação dos casos em que as generalizações não se aplicam (com a ressalva de que as irregularidades devem estar igualmente marcadas no *corpus*).

Com relação às etiquetas, as seguintes dicas também podem ajudar:

1. A existência de uma etiqueta do tipo MISCELÂNEA, ou OUTROS, é útil para os casos não previstos. Esta etiqueta pode incluir casos que serão especificados no futuro;
2. Admitir a indeterminação, isto é, que duas ou mais etiquetas estejam igualmente adequadas, no mesmo contexto. A possibilidade de anotação múltipla permitiria que as etiquetas AVISO, AMEAÇA e CONSELHO sejam atribuídas à frase *Se eu fosse você, deixaria a cidade imediatamente*, por exemplo.

1.4.4. Avaliação da anotação

No PLN, temos duas maneiras de aferir a qualidade das anotações: comparar entre si as anotações produzidas pelos anotadores (humanos), ou comparar essas anotações e um gabarito (e este gabarito foi produzido por alguém). Como nem sempre existe um gabarito à disposição, a comparação entre análises de diferentes anotadores acaba sendo a solução adotada. Assim, a ideia de uma anotação correta (supostamente fornecida pelo gabarito) é substituída pela ideia de uma anotação consistente (todos os anotadores analisaram os fenômenos da mesma maneira). Este procedimento de avaliação e verificação da anotação humana é chamado de Concordância Inter-Anotadores (*Inter annotator agreement*) (Artstein 2017).

1.4.5. Considerações finais sobre o processo de anotação

Em resumo, um projeto cuidadoso de anotação deverá levar em conta:

- Clareza quanto ao fenômeno que será anotado (que se reflete em um bom esquema de anotação);
- Escolha do *corpus* adequado (um *corpus* composto por relatórios de pesquisa é pouco adequado para a anotação de ironia, por exemplo);

- um conjunto de etiquetas e um esquema de anotação;
- instruções para a aplicação das etiquetas (documentação), que contenham tanto os casos gerais quanto as exceções;
- Avaliação da anotação (concordância entre anotadores);
- Verificação quanto à eficiência da anotação (por exemplo, o tempo levado e o nível de treinamento/conhecimento necessário por parte de quem vai anotar).

1.5. Aplicações de PLN com modelos pré-treinados

Por fim, esta última seção apresenta o *fine-tuning* de modelos computacionais estado-da-arte para o português em algumas aplicações de PLN.

1.5.1. Análise de sentimentos em avaliações de produtos da B2W (Americanas s.a.)

Uma das aplicações mais conhecidas do PLN é a análise de sentimentos (ou mineração de opiniões) que visa extrair o sentimento (ou a polaridade) em sentenças que expressam opiniões. A classificação, neste caso, pode ser em emoções (gratidão, alívio, remorso, etc.) como ocorre no GoEmotions (Demszky et al. 2020) ou em polaridade/valência (positiva, negativa ou neutra).

Nesta seção, vamos ilustrar o uso de um modelo pré-treinado para o português, o BERTimbau[26] (Souza et al. 2020), na análise de polaridade/valência usando o *corpus* B2WReviews[27], um recurso disponível livremente que contém mais de 130.000 avaliações (*reviews*) de clientes sobre produtos vendidos pela B2W (atual Americanas S.A.). De acordo com as informações na página deste *corpus*, essas avaliações foram coletadas do site Americanas.com nos primeiros 5 meses de 2018. A Figura 1.4 ilustra um trecho desse *corpus* com os campos de informação que serão usados no ajuste do modelo: `review_text` (texto da avaliação escrita pelo usuário), `overall_rating` (quantidade de 1 a 5 estrelas atribuídas pelo usuário para o produto adquirido) e `polaridade` (novo campo inserido neste exemplo para mapear `overall_rating` para polaridades negativa, neutra ou positiva).[28]

A partir das informações textuais (textos das avaliações) o modelo pré-treinado do BERTimbau foi usado no *fine-tuning* para classificação em três classes possíveis (polaridade): positiva, negativa ou neutra. Para determinar essas classes, fizemos o mapeamento da nota atribuída pelos clientes (`overall_rating`) aos produtos que adquiriram como: 4 e 5 (positiva), 3 (neutra) e 1 e 2 (negativa). Desse modo, vale ressaltar que a anotação dos dados foi realizada pelos próprios autores das avaliações não sendo, portanto, uma anotação *gold standard* realizada seguindo critérios rigorosos.

A análise de sentimentos é um exemplo de tarefa de classificação de uma sequência (*sequence classification*), uma vez que toda a sequência de entrada é processada para

[26]<https://github.com/neuralmind-ai/portuguese-bert>

[27]<https://github.com/b2w-digital/b2w-reviews01>

[28]É importante mencionar que os textos são textos gerados pelos usuários e portanto, podem conter abreviações comuns na linguagem coloquial (como *td* e *hj*), ausência de acentos (como *maos*) e erros de grafia (como em *ofereçe*), entre outros.

review_text	overall_rating	polaridade
A TV chegou em minha casa dia 21/12/2017 liguei todos os fios e td certo, chegou hj dia 01/01/2018 a TV não quer maos ligar, nem acende o led, lamentável acaba a confiança na marca!!! Não recomendo.	1	negativa
a mochila nao esta fechando direito por isso nao recomendo se meu filho nao tivesse deixado suja ia devolver	2	negativa
Produto ofereçe o que prometeu no anúncio, entrega muito rapida bem antes do prazo	3	neutra
Estou contente com a compra entrega rápida o único problema com as Americanas é se houver troca ou devolução do produto o consumidor tem problemas com espera.	4	positiva
Excelente produto, por fora em material acrílico super resistente e por dentro em adamantio, faz milagre com qualquer bebida. Sugiro aproveitarem a promoção antes que acabe.	5	positiva

Figura 1.4. Trecho do *corpus* B2WReviews apresentando apenas as informações utilizadas no treinamento

que seja gerada a saída (neste caso, a polaridade). No BERT, o *token* especial `[CLS]` é colocado no início de todas as sequências gerando representações para elas (y_{CLS}) e os pesos da rede (W_C) são ajustados para que o vetor de saída gerado represente a qual das classes uma dada sequência de entrada está associada. Desse modo, para classificar uma nova sequência de entrada, ela é processada pelo modelo pré-treinado para gerar a representação correspondente (y_{CLS}) que é, então, associada à classe correspondente com base nos pesos aprendidos durante o treinamento (W_C). O ajuste fino dos valores em W_C requer um treinamento supervisionado com base em sequências de entrada rotuladas com a classe correta.

No nosso caso, depois de 3 épocas de treinamento, usando um pouco menos de 8.000 instâncias do *corpus*, o modelo ajustado alcançou os resultados apresentados na Figura 1.5 para o conjunto de teste (com cerca de 2.000 instâncias). De acordo com os valores é possível notar que o modelo se sai muito bem na classificação das classes positiva e negativa, com precisões próximas a 90%, mas tem um desempenho bem pior na classificação da classe neutra, com apenas 44% de precisão. No geral, o valor da média harmônica entre precisão e cobertura (F1) foi de 84%.

Embora não seja o mesmo tipo de texto (avaliações de produtos), por curiosidade verificamos qual a polaridade que esse modelo gera para a sentença de exemplo da seção 1.1. O resultado é apresentado na Figura 1.6.

```
              precision    recall  f1-score   support

    Negativa       0.88      0.90      0.89       476
      Neutra       0.44      0.35      0.39       240
    Positiva       0.90      0.93      0.91      1228

    accuracy                           0.85      1944
   macro avg       0.74      0.73      0.73      1944
weighted avg       0.84      0.85      0.84      1944

F1:  0.8420356252433361 Accuracy:  0.8482510288065843
```

Figura 1.5. Resultados do *fine-tuning* para a análise de polaridade

texto	**polaridade**
Com estas palavras, André Coruja, além de quebrar o gelo que havia esfriado o clima, devolveu ao recinto a eloquência necessária para que a sessão continuasse.	positiva

Figura 1.6. Polaridade atribuída à sentença de exemplo da seção 1.1 usando o modelo ajustado com o B2WReviews

1.5.2. Etiquetagem morfossintática usando o *corpus* Mac-Morpho

Além da categorização textual realizada para sentenças (*sequence classification*), apresentada na seção anterior, outra aplicação bastante comum em PLN é a categorização de *tokens* (*sequence labelling*). Para ilustrar tal aplicação, nesta seção será descrito o uso do BERTimbau para a etiquetagem morfossintática (*part-of-speech tagging*). Para tanto, o *corpus* utilizado é o Mac-Morpho[29] (Aluísio et al. 2003). A Figura 1.7 ilustra uma sentença desse *corpus* acompanhada de suas etiquetas morfossintáticas.[30]

O Mac-Morpho é um *corpus* de textos em português do Brasil anotados com etiquetas morfossintáticas (*part-of-speech tags*). Ele é composto por 1,1 milhões de palavras derivadas de artigos do jornal brasileiro Folha de São Paulo[31]. Como tal, esse *corpus*, diferente do usado na seção anterior, possui textos que seguem a norma culta da língua portuguesa e não apresentam erros ortográficos e gramaticais. A anotação deste *corpus* foi realizada inicialmente de modo automático pelo *parser* PALAVRAS (Bick 2000) e, então, revisada por 4 anotadores.

Diferente do processo de classificação de sequência (adotado no *fine-tuning* para a análise de sentimentos), aqui o vetor de saída está associado a cada *token* e não mais a uma sequência completa. Depois de 3 épocas de treinamento o modelo ajustado alcançou os resultados apresentados na Figura 1.8 para o conjunto de teste, ou seja, uma acurácia quase perfeita de 98%. Aqui vale ressaltar que o desempenho dos modelos computacionais varia muito entre as tarefas de PLN, uma vez que tarefas mais objetivas como a etiquetagem morfossintática de *tokens* tendem a ser mais fáceis de generalizar do que as

[29] <http://nilc.icmc.usp.br/macmorpho/>

[30] A lista completa as etiquetas utilizadas na anotação desse *corpus* está disponível em: <http://nilc.icmc.usp.br/macmorpho/macmorpho-manual.pdf>.

[31] <https://www.folha.uol.com.br/>

Ainda ADV	em PREP	dezembro N	de PREP	1990 N	, PU
foi V	editada PCP	a ART	famosa ADJ	289 N	, PU
que PRO-KS	modificava V	a ART	sistemática N	da PREP+ART	arrecadação N
do PREP+ART	ITR NPROP	e KC	alterava V	suas PROADJ	alíquotas N
. PU					

Figura 1.7. Trecho do *corpus* Mac-Morpho onde cada *token* está associado a uma etiqueta morfossintática

tarefas mais subjetivas como a análise de sentimentos.

A título de ilustração, o modelo ajustado gerou as etiquetas de *part-of-speech* para a sentença de exemplo da seção 1.1 como apresentado na Figura 1.9.

```
              accuracy                           0.98     33729
             macro avg       0.96      0.94      0.95     33729
          weighted avg       0.98      0.98      0.98     33729
```

Figura 1.8. Resultados do *fine-tuning* para a etiquetagem morfossintática

Com PREP	estas PROADJ	palavras N	, PU	André NPROP	Coruja NPROP	, PU
além PREP	de PREP	quebrar V	o ART	gelo N	que PRO-KS	havia V
esfriado PCP	o ART	clima N	, PU	devolveu V	ao PREP+ART	recinto N
a ART	eloquência N	necessária ADJ	para PREP	que KS	a ART	sessão N
continuasse V	. PU					

Figura 1.9. Etiquetas geradas para a sentença de exemplo da seção 1.1 usando o modelo ajustado com o Mac-Morpho

1.5.3. Tradução automática en-pt usando TED *corpus*

A última aplicação de *fine-tuning* de um modelo pré-treinado envolvendo o português do Brasil apresentada neste documento é a de tradução automática. Neste caso, a partir de pares de sentenças paralelas (sentenças que são a tradução umas das outras), o modelo pré-treinado é ajustado para ser capaz de gerar uma sequência de *tokens* alvo (de saída) a partir de uma sequência de *tokens* fonte (de entrada).

Para tanto, aqui optamos por partir de um modelo pré-treinado do T5[32] e realizar um ajuste fino com o TED[33] *corpus* para o qual pares de sentenças paralelas português-inglês são apresentados na Figura 1.10.

Sentença em português	**Sentença em inglês**
Este é o caminho para tratar as pessoas como adultos responsáveis.	This is the way we treat people as responsible adults.
Ela morava na rua havia mais de 20 anos, tinha problemas mentais e era uma alcoólatra inveterada.	She had been on the street for 20-plus years, had mental health issues and was a severe alcoholic.
Isso é na antiga União Soviética entre o Cazaquistão e o Usbequistão, um dos maiores mares interiores do mundo.	This is in the former Soviet Union in between Kazakhstan and Uzbekistan, one of the great inland seas of the world.
Há muitos anos, eu comecei a usar o termo "interseccionalidade" para lidar com o fato de que muitos de nossos problemas de justiça social, como racismo e sexismo, frequentemente se sobrepõem, criando múltiplos níveis de injustiça social.	Many years ago, I began to use the term "intersectionality" to deal with the fact that many of our social justice problems like racism and sexism are often overlapping, creating multiple levels of social injustice.
Havia isso onde todas essas coisas em minha mente podiam estar nesse lugar lá fora, grande e elétrico, existir e escapar, e eu sabia, a partir daí, que eu queria fazer isso pelo resto da vida, fosse algo remunerado ou não. Eu tinha essa paixão e precisava de ferramentas.	There was this place where all these things in my head could go into the great, electric somewhere-out-there and exist and escape, and I knew from this moment on, I wanted to do this for the rest of my life, whether I was going to get paid for it or not.

Figura 1.10. Pares de sentenças paralelas do TED *corpus*

Neste caso, usamos um modelo de linguagem de sequência-para-sequência (ou `seq2seq`) que é o indicado para tarefas de PLN nas quais a entrada é uma sequência de *tokens* e a saída também é uma sequência de *tokens*, mas de tamanho arbitrário; e não mais uma classificação como nos casos anteriores. Assim, modelos pré-treinados `seq2seq` são úteis para aplicações como a tradução automática, a sumarização automática e a geração de legendas para imagens.

Depois de três épocas de treinamento, o modelo ajustado apresentou um BLEU[34] (Papineni et al. 2002) de 0,46 que trouxe uma tradução bastante razoável para a sentença de exemplo da seção 1.1, como ilustrado na Figura 1.11. Neste caso, fazendo uma comparação da saída de nosso modelo ajustado com aquela gerada pelo Google tradutor, temos apenas duas palavras diferentes (destacadas em negrito): o sobrenome Oruja, que nosso modelo traduziu de modo equivocado (o correto seria manter o original); e o termo *re-*

[32]<https://huggingface.co/Helsinki-NLP/opus-mt-ROMANCE-en>

[33]<https://object.pouta.csc.fi/OPUS-TED2020/v1/tmx/en-pt_br.tmx.gz>

[34]O BLEU é uma medida de avaliação automática baseada na co-ocorrência de n-gramas (sequências de n *tokens*) em comum entre a saída gerada por um sistema automático e uma ou mais sentenças de referência.

cinto traduzido para *compound* pelo nosso modelo e para *venue* (uma tradução melhor) pelo Google.

Fonte	Com estas palavras, André Coruja, além de quebrar o gelo que havia esfriado o clima, devolveu ao recinto a eloquência necessária para que a sessão continuasse.
Alvo (nosso modelo)	With these words, André **Oruja**, in addition to breaking the ice that had cooled the climate, returned to the **compound** the eloquence necessary for the session to continue.
Alvo (Google)	With these words, André **Coruja**, in addition to breaking the ice that had cooled the climate, returned to the **venue** the eloquence necessary for the session to continue.

Figura 1.11. Tradução gerada para a sentença de exemplo da seção 1.1 usando o modelo ajustado com o TED *corpus* e o Google tradutor

1.6. Considerações finais

O processamento automático das línguas naturais (ou PLN) é uma área interdisciplinar que tem as línguas humanas como objeto de estudo e processamento. Os processamentos realizados com as técnicas, ferramentas e recursos do PLN podem ser o produto final (como em corretores ortográficos, tradutores e sumarizadores automáticos, ferramentas de auxílio à escrita, etc.) ou o produto intermediário para enriquecimento de dados que poderão ser utilizados em uma aplicação final (como ocorre com os etiquetadores morfossintáticos e os analisadores sintáticos e semânticos, por exemplo).

Saber quanto do PLN é necessário para uma aplicação computacional exige conhecimento das características e peculiaridades da língua natural sendo processada (**conhecimento da língua**), bem como do produto final que se deseja desenvolver (**propósito**). Hoje, com a disponibilização livre de códigos, ferramentas computacionais, recursos linguísticos e modelos pré-treinados, implementar soluções de PLN se tornou uma tarefa bem menos custosa. Dados também estão cada vez mais disponíveis, principalmente aqueles gerados pelas pessoas comuns, em suas comunicações e interações corriqueiras em redes sociais e postagens online. Contudo, códigos e dados sozinhos não são capazes de produzir uma aplicação de PLN robusta. O conhecimento da língua e do propósito para seu processamento devem sempre ser a base de qualquer desenvolvimento em PLN.

Referências

Aluísio et al. 2003 ALUÍSIO, S. et al. An account of the challenge of tagging a reference corpus for brazilian portuguese. In: MAMEDE, N. J. et al. (Ed.). *Computational Processing of the Portuguese Language*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 110–117. ISBN 978-3-540-45011-5.

Artstein 2017 ARTSTEIN, R. Inter-annotator agreement. In: *Handbook of linguistic annotation*. [S.l.]: Springer, 2017. p. 297–313.

Aziz e Specia 2011 AZIZ, W.; SPECIA, L. Fully automatic compilation of a Portuguese-English parallel corpus for statistical machine translation. In: *STIL 2011*. Cuiabá, MT: [s.n.], 2011.

Bick 2000 BICK, E. *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Århus: University of Arhus, 2000.

Bojanowski et al. 2016 BOJANOWSKI, P. et al. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Brown et al. 2020 BROWN, T. et al. Language models are few-shot learners. In: LAROCHELLE, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. v. 33, p. 1877–1901. Disponível em: <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf>.

Demszky et al. 2020 DEMSZKY, D. et al. GoEmotions: A dataset of fine-grained emotions. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. p. 4040–4054. Disponível em: <https://aclanthology.org/2020.acl-main.372>.

Devlin et al. 2019 DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423>.

Joshi et al. 2020 JOSHI, M. et al. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, v. 8, p. 64–77, 01 2020. ISSN 2307-387X. Disponível em: <https://doi.org/10.1162/tacl_a_00300>.

Jurafsky e Martin 2021 JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. [s.n.], 2021. Draft of December 29, 2021. Disponível em: <https://web.stanford.edu/~jurafsky/>.

Kilgarriff 1997 KILGARRIFF, A. I don't believe in word senses. *Computers and the Humanities*, Springer, v. 31, n. 2, p. 91–113, 1997. ISSN 00104817.

Liu et al. 2019 LIU, Y. et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. Cite arxiv:1907.11692. Disponível em: <http://arxiv.org/abs/1907.11692>.

McShane e Nirenburg 2021 MCSHANE, M.; NIRENBURG, S. *Linguistics for the Age of AI*. The MIT Press, 2021. ISBN 9780262363136. Disponível em: <https://doi.org/10.7551/mitpress/13618.001.0001>.

Mikolov et al. 2013 MIKOLOV, T. et al. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, v. 2013, 01 2013.

Oliveira et al. 2015 OLIVEIRA, H. G. et al. As wordnets do português. *Oslo Studies in Language*, v. 7, n. 1, p. 397–424, 2015.

Papineni et al. 2002 PAPINENI, K. et al. Bleu: a method for automatic evaluation of machine translation. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, 2002. p. 311–318. Disponível em: <https://aclanthology.org/P02-1040>.

Pennington et al. 2014 PENNINGTON, J.; SOCHER, R.; MANNING, C. GloVe: Global vectors for word representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1532–1543. Disponível em: <https://aclanthology.org/D14-1162>.

Santos et al. 2010 *Relações semânticas em português: comparando o TeP, o MWN.PT, o Port4NooJ e o PAPEL*. [S.l.]: Associação Portuguesa de Linguística: Lisboa, 2010. 681–700 p.

Souza et al. 2020 SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. BERTimbau: Pretrained BERT Models for Brazilian Portuguese. In: CERRI, R.; PRATI, R. C. (Ed.). *Intelligent Systems*. Cham: Springer International Publishing, 2020. p. 403–417. ISBN 978-3-030-61377-8.

sbbd:02

Capítulo 2

Projeto de Bancos de Dados NoSQL

Angelo Augusto Frozza, Geomar André Schreiner and Ronaldo dos Santos Mello

Resumo

Bancos de Dados (BDs) NoSQL são uma tecnologia relativamente recente para gerenciamento de dados cuja principal motivação é um melhor atendimento dos desafios do Big Data, como volume, variedade e velocidade. Centenas de sistemas gerenciadores de BDs (SGBDs) NoSQL estão hoje disponíveis e alguns deles já ocupam posições elevadas no ranking de SGBDs populares mundialmente. NoSQL engloba uma família de modelos de dados não-relacionais: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos. Considerando essa popularidade crescente e variedade de modelos de dados, torna-se pertinente a definição e adoção de uma metodologia de projeto para BDs NoSQL quando se deseja um esquema de base para a manipulação dos dados. Este capítulo apresenta uma visão geral dos modelos de dados NoSQL e uma proposta para projeto lógico de BDs NoSQL nos quatro modelos de dados supracitados, a partir de uma modelagem conceitual definida no modelo entidade-relacionamento estendido (modelo EER). Discute-se também a implementação deste projeto lógico em dois SGBDs NoSQL de amplo uso na indústria: MongoDB e Neo4j.

2.1. Apresentação

2.1.1. Objetivo

Este capítulo apresenta uma metodologia para o projeto lógico de BDs NoSQL. Em termos de projeto físico, considera-se dois SGBDs NoSQL: *MongoDB* e *Neo4j*.

2.1.2. Relevância para o Público-Alvo

A relevância deste capítulo se justifica pelo fato de BDs NoSQL [Sadalage and Fowler 2013, Atzeni et al. 2014] serem uma tecnologia relativamente recente para o gerenciamento de dados e, desta forma, os cursos de graduação na área de Ciência da Computação geralmente não apresentam essa tecnologia em suas disciplinas da área de BD. Disciplinas na área de BD enfatizam o modelo de dados relacional, porém, NoSQL compreende uma

família de novos modelos de dados não-relacionais que vêm crescendo em termos de utilização pela indústria. Assim sendo, ao final deste capítulo o público-alvo terá adquirido um conhecimento inovador na área de BD.

Além disso, o projeto de BDs NoSQL é um tópico pouco explorado na literatura de BD. Por este motivo, o conteúdo desse capítulo se torna também interessante para alunos de pós-graduação, pesquisadores e profissionais entusiastas que buscam inovação na gestão dos seus dados. Os autores deste capítulo têm trabalhado em pesquisas relacionadas a este tópico nos últimos anos [Lima and Mello 2015, Lima and Mello 2016, Schreiner et al. 2019, Frozza and Mello 2020, Santana and Mello 2020, Schreiner et al. 2020, Silva and Mello 2021].

2.1.3. Autores

Os autores deste capítulo são todos professores com vários anos dedicados ao ensino e pesquisa na área de BD. Seus curriculos resumidos são os seguintes:

- **Angelo Augusto Frozza** é doutor em Ciência da Computação pela Universidade Federal de Santa Catarina (UFSC) e professor efetivo no Instituto Federal Catarinense (IFC), Campus Camboriú. Sua área de atuação é BD e seus principais tópicos de pesquisa são data warehouse, BDs geográficos e modelagem de dados. CV Lattes: *http://lattes.cnpq.br/5878372087019892*.
- **Geomar André Schreiner** é doutor em Ciência da Computação pela UFSC e professor na Universidade do Oeste de Santa Catarina (UNOESC) em Chapecó. Sua área de atuação é BD e seus principais tópicos de pesquisa são BDs NoSQL, BDs NewSQL, interoperabilidade de dados e particionamento de dados. CV Lattes: *http://lattes.cnpq.br/0776438722468291*.
- **Ronaldo dos Santos Mello** é doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS) e professor titular na UFSC. Coordena o Grupo de pesquisa em BD da UFSC e o PET Computação da UFSC. Sua área de atuação é BD e seus principais tópicos de pesquisa são modelagem de dados, restrições de integridade, integração e interoperabilidade de dados, BD NoSQL e BD NewSQL. CV Lattes: *http://lattes.cnpq.br/5011370918857999*.

2.1.4. Organização do Capítulo

Este capítulo está organizado em mais 4 seções. A Seção 2 apresenta uma visão geral de BDs NoSQL. A Seção 3 detalha uma metodologia para projeto lógico de BDs NoSQL e a Seção 4 exemplifica a implementação de um projeto lógico nos SGBDs MongoDB e Neo4j. Por fim, a Secão 5 é dedicada às considerações finais.

2.2. Introdução a BDs NoSQL

Esta seção apresenta a origem dos BDs NoSQL, suas principais propriedades e modelos de dados. O texto tem por base o livro *NoSQL Distilled* [Sadalage and Fowler 2013].

2.2.1. Um Pouco de História

O termo "NoSQL"foi utilizado pela primeira vez em 1998, por Carlos Strozzi, para apresentar sua proposta de BD relacional de código aberto que não implementava a linguagem SQL[1]. Porém, esse SGBD não teve nenhuma influência sobre o desenvolvimento dos BDs que seguem o paradigma NoSQL.

No final dos anos 2000, a Big Table, do Google, e o DynamoDB, da Amazon, inspiravam o desenvolvimento de inúmeros projetos alternativos para armazenamento de dados, sendo que isso tornou-se assunto nas melhores conferências de software da época.

Em 2009, Johan Oskarsson, um desenvolvedor de software de Londres, foi a São Francisco (EUA) para um evento sobre *Hadoop*. Ele também estava interessado em conhecer mais sobre esses novos SGBDs não relacionais, distribuídos e de código aberto que estavam surgindo e, como estava com pouco tempo, resolveu organizar uma reunião para que todos pudessem apresentar seus trabalhos para quem estivesse interessado em conhecê-lo. Assim, participaram representantes dos projetos Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB e MongoDB. Johan procurava um nome para identificar essa reunião, o qual teria que ser uma boa *hashtag* para o *Twitter* (curto, fácil de lembrar e sem muitos similares no Google). Ele usou seus contatos para pedir sugestões e, entre as recebidas, escolheu usar "*NoSQL Meetup*".

Atualmente, o termo NoSQL é aplicado apenas em SGBDs desenvolvidos a partir do século XXI e que possuem as principais características dos BDs NoSQL. Para [Sadalage and Fowler 2013], o termo NoSQL é entendido como ”*Not only SQL*” e remete a SGBDs que não seguem o modelo de dados relacional ao mesmo tempo que não pretendem substituir o modelo relacional, mas sim, ser uma opção quando é necessária uma maior flexibilidade na estruturação do BD. Michael Stonebreaker cita duas razões principais que motivam a troca de um SGBD relacional por um SGBD NoSQL: *(i)* necessidade de um *desempenho* melhor; e *(ii)* necessidade de maior *flexibilidade* [Stonebraker 2010].

Os BDs NoSQL foram projetados para gerenciar grandes volumes de dados e apresentam 5 propriedades principais [Schreiner et al. 2015]: *(i)* Escalabilidade horizontal; *(ii)* Armazenamento distribuído; *(iii)* Interface de acesso simples; *(iv)* Alta disponibilidade; e *(v)* Esquemas flexíveis ou mesmo ausência de esquema.

2.2.2. Modelos de dados NoSQL

BDs NoSQL podem ser categorizados pelo modelo de dados adotado: *chave-valor, orientado a documentos, orientado a colunas* e *orientado a grafos*. Os três primeiros são conhecidos como *modelos NoSQL baseados em chave*, pois compartilham o princípio de recuperar dados a partir de uma chave de entrada, enquanto diferem em termos de acesso aos componentes internos dos dados [Atzeni et al. 2014]. Esses modelos de dados são

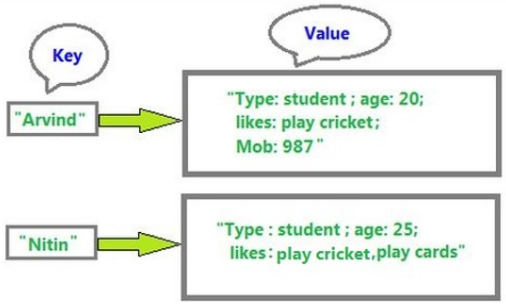
[1]NoSQL Relational Database Management System,`http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page`

detalhados a seguir.

2.2.2.1. Chave-valor

O modelo de dados *chave-valor* é o mais simples, similar a uma estrutura de indexação, como uma *hash table*. Ele mantém um conjunto de pares chave-valor, sendo o valor acessado por meio da chave única (ver Figura 2.1). Um valor pode conter um conteúdo simples ou complexo, mas esse conteúdo é tratado como um componente único, ou seja, uma "caixa-preta". Devido a isso, assume-se que qualquer valor no modelo de dados chave-valor tem um domínio atômico [Sadalage and Fowler 2013, Atzeni et al. 2014].

Figura 2.1. Representação do modelo de dados chave-valor



A API de acesso é simples, contendo as operações *get(key)*, *put(key, value)* e *delete(key)*. Não suporta definição de esquemas, relacionamentos entre os dados e não possui uma linguagem de consulta. Exemplos de SGBDs NoSQL chave-valor populares são o *Redis* e o *Voldemort*.

2.2.2.2. Orientado a documentos

O modelo *orientado a documentos* define uma coleção de documentos, sendo cada documento acessado por uma chave única. Um documento é composto por atributos atômicos ou atributos complexos com conteúdo estruturado principalmente a partir do uso dos construtores JSON para *objeto* e *array* que podem, por sua vez, abrigar valores atômicos ou complexos (ver Figura 2.2) [Sadalage and Fowler 2013, Atzeni et al. 2014].

O modelo orientado a documentos é mais adequado para a representação de objetos complexos. SGBDs que seguem esse modelo possuem APIs de acesso proprietárias e/ou linguagens de consulta simples. Não suporta relacionamento entre dados. Exemplos de SGBDs NoSQL orientado a documentos são o *MongoDB* e o *CouchDB*.

2.2.2.3. Orientado a colunas

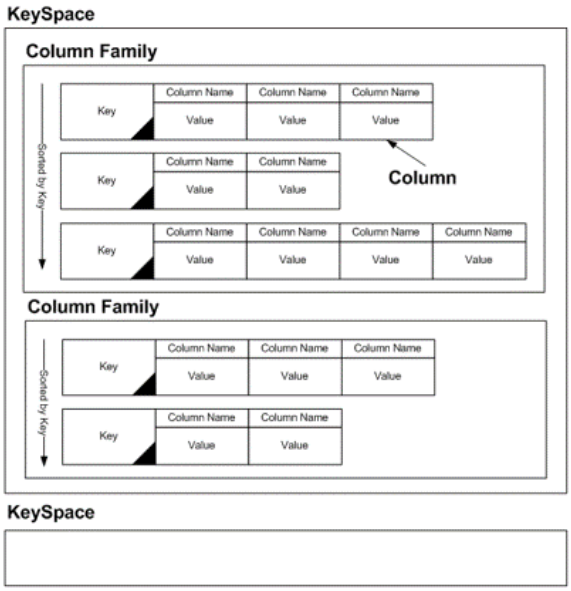
O modelo de dados *orientado a colunas* organiza os dados com base em um modelo distribuído em colunas, semelhante ao modelo relacional, mas com esquema flexível. Novas colunas podem ser definidas no momento do armazenamento. Por causa disso,

Figura 2.2. Exemplo de um documento (formato JSON)

```
{ "_id":"discussion_tables",
  "_rev":"D1C946B7",
  "Sunrise":true,
  "Sunset":false,
  "FullHours":[1,2,3,4,5,6,7,8,9,10],
  "Activities": [
     {"Name":"Football", "Duration":2, "DurationUnit":"Hours"},
     {"Name":"Breakfast", "Duration":40, "DurationUnit":"Minutes",
           "Attendees":["Jan", "Damien", "Laura", "Gwendolyn", "Roseanna"]}
   ]      }
```

instâncias de dados de uma mesma entidade podem ter um número diferente de colunas e alguns BDs dessa classe suportam o conceito de supercoluna. Ao mesmo tempo, colunas são agrupadas em famílias, o que auxilia o particionamento vertical e a otimização física (ver Figura 2.3). A criação de novas tabelas e novas famílias de colunas só é possível em tempo de projeto de esquema, quando os dados não podem ser acessados [Atzeni et al. 2014].

Figura 2.3. Representação do modelo orientado a colunas.

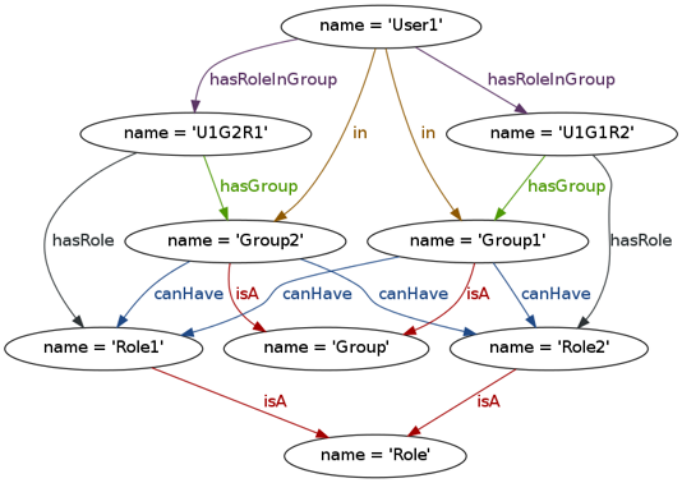


Comparando com BDs relacionais, o conceito de *keyspace* é equivalente ao conceito de *BD*, uma *família de colunas* é equivalente a uma *tabela* e um *conjunto de colunas* é equivalente a um *registro*. Uma coluna possui um nome e um valor, um conjunto de colunas é acessado por uma chave e instâncias de dados (registros) podem ter quantidade de colunas diferentes. Alguns SGBDs possuem suporte a colunas multivaloradas e supercolunas. O acesso aos dados é por meio de APIs proprietárias e/ou linguagens de consultas simples e não suportam relacionamentos entre dados. Alguns exemplos de SGBDs NoSQL orientados a colunas são o *Cassandra* e o *HBase*.

2.2.2.4. Orientado a grafos

O modelo de dados *orientado a grafos* consiste basicamente de vértices conectados por arestas [Sadalage and Fowler 2013]. Essa estrutura é indicada para armazenar pequenos registros que possuem interconexões complexas, como é o caso de dados em redes sociais. Um vértice representa um objeto do mundo real, enquanto uma aresta representa uma relação entre dois vértices. Propriedades podem ser atribuídas para vértices e arestas e são formadas por pares chave-valor, sendo que o valor é restrito a um determinado tipo de dados (Figura 2.4).

Figura 2.4. Representação do modelo orientado a grafos



O acesso aos dados é possível através de APIs proprietárias e/ou linguagens de consultas simples. Exemplos de SGBDs NoSQL orientados a grafos são o *Neo4j* e o *InfiniteGraph*.

2.3. Projeto lógico de BDs NoSQL

2.3.1. Projeto clássico de BD

O projeto de um BD é parte integrante do desenvolvimento de um sistema informatizado que necessita lidar com um grande volume de dados. Seus principais objetivos são a representação adequada dos dados operacionais da Organização e o armazenamento/acesso eficientes a estes dados [Henry Korth and Silberschatz 2019]. Uma metodologia clássica de projeto de BD apresenta 3 etapas que definem esquemas de dados em diferentes níveis de abstração a partir do levantamento dos requisitos de dados da Organização [Heuser 2009, Elmasri and Navathe 2011]:

- *projeto conceitual*: modelagem de dados de alto nível de abstração, ou seja, independente do(s) modelo(s) de dados do(s) SGBD(s) alvo. Seu objetivo é o entendimento dos dados e seus relacionamentos dentro da Organização, bem como a validação deste entendimento junto aos usuários da Organização. O modelo clássico utilizado para a especificação de um esquema conceitual é o modelo Entidade-Relacionamento Estendido (modelo EER);

- *projeto lógico*: conversão do esquema conceitual em esquema(s) válido(s) no(s) modelo(s) de dados do(s) SGBD(s) alvo, como por exemplo, o modelo relacional e o modelo orientado a documentos. Nesta etapa aplicam-se regras de conversão visando gerar esquema(s) de dados eficientes para fins de armazenamento, assim como acesso pelas principais operações de manipulação de dados;

- *projeto físico*: esta etapa realiza a especificação do(s) esquema(s) lógico(s) no(s) SGBD(s) alvo, ou seja, realiza-se a implementação das estruturas de dados e dos recursos para garantia de integridade, segurança e otimização de acesso utilizando a linguagem de definição de dados (DDL) do(s) SGBD(s). Exemplos no contexto de BDs relacionais são a definição de tabelas, índices, visões e gatilhos.

A Figura 2.5 mostra uma modelagem conceitual utilizando o modelo EER (diagrama à esquerda) e uma possível modelagem lógica relacional (diagrama à direita) para o domínio de um *Museu*. Fatos do mundo real (ou entidades) são representados por retângulos nomeados (p.ex., *Obras*) enquanto relacionamentos entre esses fatos são representados por losangos nomeados (p.ex., *exposição*) e possuem cardinalidades associadas (p.ex., uma obra pode estar exposta em um único salão e um salão pode abrigar várias obras). Entidades e relacionamentos podem conter atributos (p.ex, *título* da obra e *posição* da exposição, respectivamente), sendo possível definir atributos compostos (p.ex., *endereco*), multivalorados (p.ex., *diasAtendimento*) e identificadores (p.ex., *CNPJ* do fornecedor). Ainda, uma entidade pode se especializar em uma ou mais entidades que podem apresentar propriedades adicionais (atributos e/ou relacionamentos), como é o caso de uma obra, que pode ser uma *Pintura* ou uma *Escultura*. Por fim, uma entidade associativa (p.ex., *restauração*) denota um relacionamento que é promovido a uma entidade para que possa se relacionar com outras entidades.

A modelagem lógica relacional, por sua vez, é gerada através da aplicação de regras de conversão. Basicamente, uma entidade se torna uma tabela (p.ex., *Pais*) e

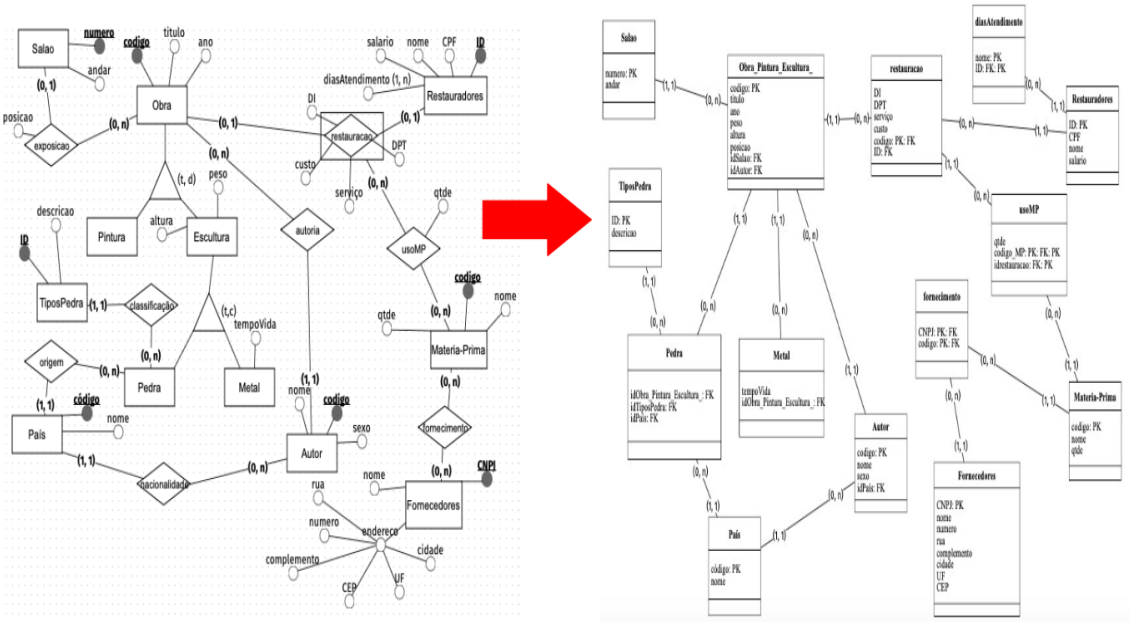


Figura 2.5. Exemplo de modelagem conceitual e uma correspondente modelagem lógica

um relacionamento pode ser representado por uma chave estrangeira (p. ex., *idTipoPedra* em Pedra), gerar uma tabela própria (p.ex., *usoMP*) ou gerar uma tabela resultante da sua fusão com os dados das entidades envolvidas. Entidades envolvidas em um relacionamento de especialização podem ser fundidas em uma única tabela (p.ex., *Obras_Pinturas_Esculturas*), gerar tabelas para cada entidade ou ainda tabelas apenas para as entidades especializadas (p.ex., *Pedra* e *Metal* no caso de Esculturas). Atributos multivalorados e entidades associativas geralmente geram tabelas.

A Figura 2.6 exemplifica parte da especificação de um projeto físico para a modelagem lógica apresentada na Figura 2.5 utilizando a linguagem DDL da SQL. Ilustra-se a definição da tabela *Salao* com seus atributos e restrições, a definição de um índice para acelerar buscas por *cidade* na tabela *Fornecedores* e uma visão que permite visualizar dados de obras e autores conjuntamente.

2.3.2. Adequação do Projeto Clássico de BD a BDs NoSQL e o Modelo de Agregados

Conforme descrito na Seção 2.2.2, os BDs NoSQL são uma família de modelos lógicos de dados. Assim sendo, a etapa de projeto lógico deve ser capaz de converter um esquema EER para um ou mais desses modelos NoSQL. Para evitar uma quantidade excessiva de regras de mapeamento para cada modelo NoSQL, uma estratégia para minimizar a complexidade do projeto lógico pode ser a adoção de um modelo intermediário que seja capaz de abstair características comuns de mais de um desses modelos NoSQL.

Neste trabalho consideramos o modelo de agregados, conforme sugerido por Sadalage & Fowler [Sadalage and Fowler 2013], como modelo intermediário. Desta forma, o projeto lógico clássico de um BD pode ser desmembrado em 2 etapas: *(i) projeto lógico de alto nível*; e *(ii) projeto lógico de baixo nível*. A primeira converte um esquema EER para um esquema baseado no modelo de agregados e a segunda converte o esquema

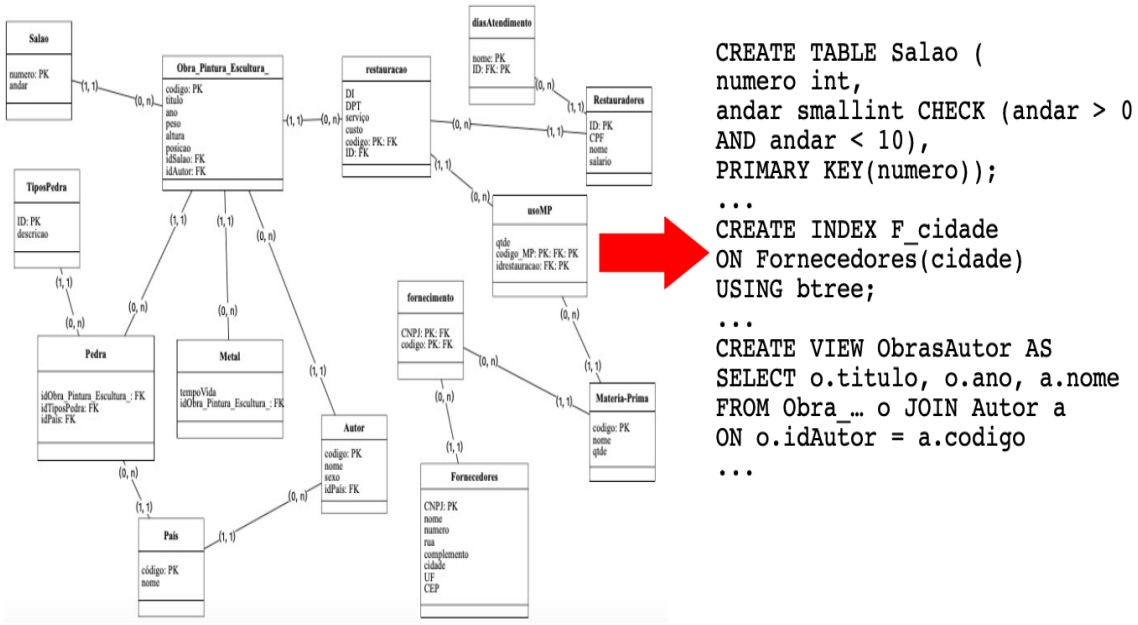


Figura 2.6. Exemplo de mapeamento de uma modelagem lógica para uma modelagem física

de agregados (ou parte dele, caso seja desejada a persistência do esquema conceitual em mais de um BD NoSQL) para o esquema de um BD NoSQL.

O modelo de agregados é adequado à abstração de dados de alto nível para 3 modelos de dados NoSQL: orientado a documentos, orientado a colunas e chave-valor. Isso por que o modelo de agregados é capaz de representar objetos complexos que possuem uma chave de acesso (identificador). Essas características de representação estão presentes nesses 3 modelos.

O modelo de agregados foi formalizado por Lima & Mello [Lima and Mello 2015]. Neste trabalho também foi definida uma notação gráfica para ele, conforme exemplificado na Figura 2.7 para dados no domínio de um Museu.

Os conceitos do modelo de agregados são: *(i) coleção; (ii) bloco; (iii) atributo; (iv) relacionamento de agregação; (v) relacionamento de referência;* e *(vi) restrição de disjunção*. O primeiro representa um objeto de um mundo real (geralmente uma entidade) e possui um identificador (atributo com prefixo *ID_*). Uma coleção pode ser referenciada por outra (ou pela mesma) coleção. Exemplos são *Obra* e *Salão* na Figura 2.7.

Um bloco é um componente interno a uma coleção ou a outro bloco, e pode ter uma cardinalidade associada, indicando que várias ocorrências dele podem existir. Um exemplo na Figura 2.7 é o bloco opcional *autoria*. Tanto blocos quanto coleções podem ter atributos que descrevem seus dados, podendo eles ser mono ou multivalorados. Um exemplo é o atributo monovalorado *nacionalidade* no bloco autoria e o atributo multivalorado *horários* na coleção Salão.

Os dois tipos de relacionamento presentes no modelo de agregados são agregação e referência. O primeiro é representado pelos blocos aninhados em coleções ou outros blocos, indicando a existência de um objeto interno a outro, como é o caso de *Pintura*

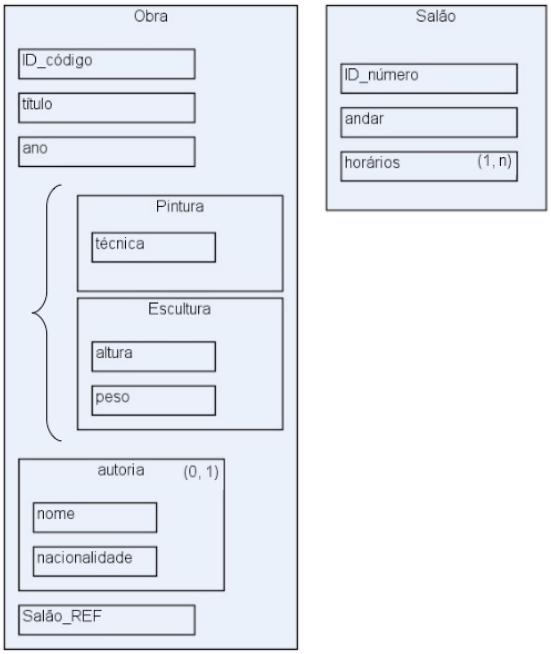


Figura 2.7. Exemplo de esquema de dados representado no modelo de agregados

e *Escultura* na coleção *Obra*. O segundo representa relacionamentos entre coleções e é definido por um atributo com sufixo *_REF*. Um exemplo na Figura 2.7 é o atributo *Salao_REF*, indicando o salão onde a obra pode estar exposta.

Por fim, uma restrição de disjunção indica blocos mutuamente exclusivos em uma coleção, sendo uteis para representar especializações disjuntas em um diagrama EER. Blocos mutuamente exclusivos são denotados no modelo de agregados por um ”abre chave” envolvendo todos os blocos que devem ser disjuntos. Um exemplo na Figura 2.7 são os blocos *Pintura* e *Escultura* na coleção *Obra*.

A seguir detalha-se as duas etapas do projeto lógico de um BD NoSQL. Elas foram propostas no trabalho de Lima [Lima 2016].

2.3.3. Projeto Lógico de Alto Nível para BDs NoSQL

A etapa de projeto lógico de alto nível converte um esquema conceitual representado no modelo EER para um esquema de agregados. Ela executa 2 subetapas, nesta ordem: *(i) conversão de hierarquias; (ii) conversão de relacionamentos*. Estas conversões tomam como base as regras de mapeamento para projeto lógico de BDs relacionais [Heuser 2009].

A primeira subetapa analisa cada hierarquia de especialização presente no esquema EER, através de um percorrimento *bottom-up* dos seus níveis hierárquicos, e aplica uma dentre as seguintes regras de conversão para cada nível:

1. *ênfase na entidade genérica*: gera-se uma única coleção para representar o nível hierárquico. O nome da coleção é o nome da entidade genérica e seus atributos são os atributos de todas as entidades presentes no nível. Um atributo especial *tipo*

é definido para indicar qual(is) entidade(s) do nível (genérica e/ou especializadas) está(ão) sendo instanciada(s)[2]. Um exemplo é mostrado na Figura 2.8 (a);

2. *ênfase nas entidades especializadas*: gera-se uma coleção para cada entidade especializada. Seus atributos incluem os atributos da entidade genérica. Um exemplo é mostrado na Figura 2.8 (b);

3. *ênfase na hierarquia*: gera-se uma única coleção para todo o nível representando a entidade genérica, bem como blocos aninhados a esta coleção representando as entidades especializadas. Caso a especialização seja disjunta, uma restrição de disjunção é definida. Um exemplo é mostrado na Figura 2.8 (c).

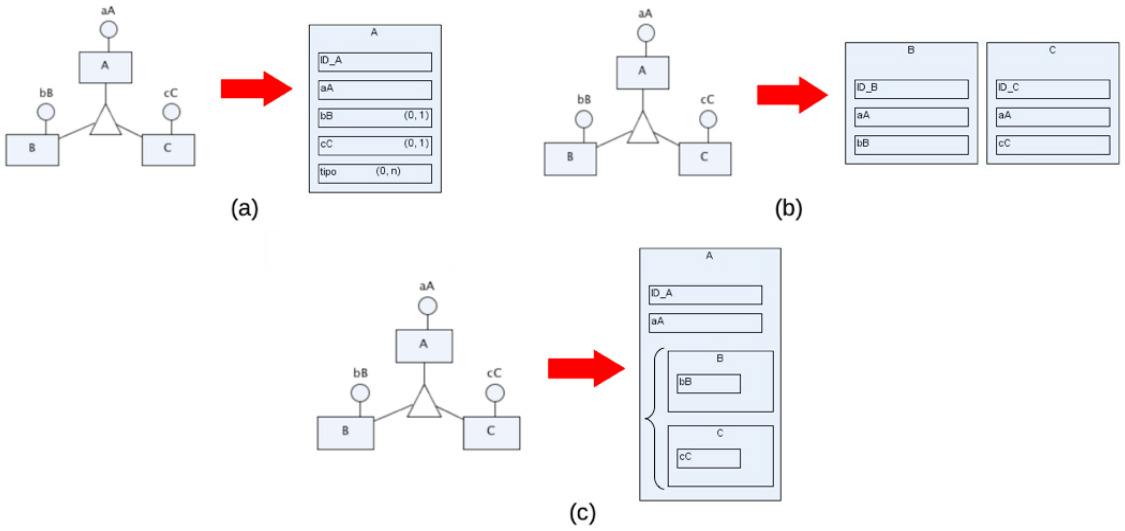


Figura 2.8. Conversão de hierarquias em um esquema EER

A escolha por uma ou outra alternativa depende de fatores como o *tamanho do esquema*, a *carga de trabalho* típica (*workload*) da aplicação e a quantidade de *restrições de integridade* que devem ser definidas para respeitar o esquema gerado. Por exemplo, a alternativa 1 gera um esquema mais enxuto e menos complexo, sendo mais adequada a situações onde a maioria das instâncias no nível hierárquico são da entidade genérica ou quando a maioria dos atributos e relacionamentos pertencem à entidade genérica. Entretanto, ela requer alguns controles de integridade associados ao tipo da instância. Conforme ilustra a Figura 2.8 (a), se o tipo da instância for B, então o atributo *bB* deve ter valor enquanto que o atributo *cC* deve ser NULL. A alternativa 2, por sua vez gera um esquema com mais coleções, sendo indicada para um *workload* que acessa principalmente as instâncias das entidades especializadas ou quando a maioria dos atributos e relacionamentos pertencem à elas. Ela também não está livre de restrições de integridade caso a especialização seja disjunta. Por fim, a alternativa 3 se mostra interessante quando qualquer instância do nível hierárquico é relevante para o *workload*.

[2]Depende do tipo de especialização definida para o nível hierárquico no esquema EER: total/parcial e disjunta/compartilhada.

A segunda subetapa analisa os relacionamentos do esquema EER, através de uma ordenação baseada nas suas cardinalidades máximas (prioridade para relacionamentos 1-1 seguido de relacionamentos 1-n e, por fim, relacionamentos m-n), e aplica uma dentre as seguintes regras de conversão:

1. *fusão*: gera-se uma coleção que une os atributos do relacionamento e das entidades envolvidas nele. Um exemplo é mostrado na Figura 2.9 (a);

2. *aninhamento*: gera-se uma coleção C_x que representa uma das entidades envolvidas no relacionamento. A outra entidade participante é representada como um bloco aninhado à C_x. Um exemplo é mostrado na Figura 2.9 (b);

3. *referência*: gera-se uma coleção para cada uma das entidades participantes do relacionamento. Pelo menos uma destas entidades abriga o relacionamento, que pode ser representado apenas como um atributo de referência (relacionamento sem atributos) ou um bloco aninhado (relacionamento com atributos). Um exemplo é mostrado na Figura 2.9 (c).

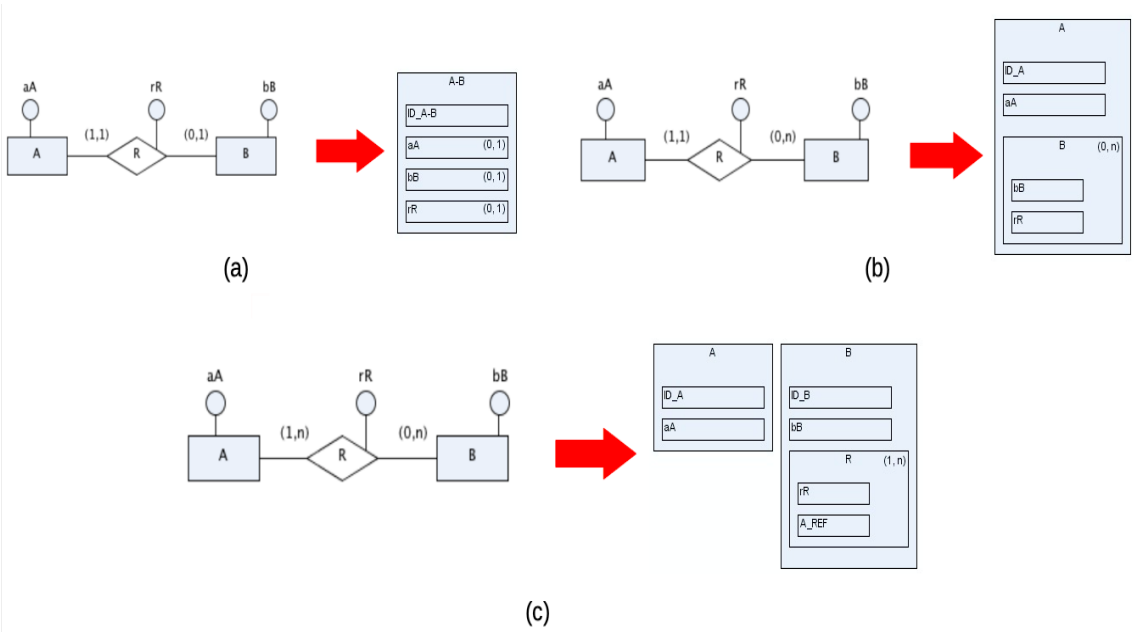


Figura 2.9. Conversão de relacionamentos em um esquema EER

A Tabela 2.1 sumariza as recomendações de aplicação destas alternativas de conversão conforme a cardinalidade do relacionamento. A alternativa *fusão* é indicada para relacionamentos 1-1, em particular relacionamentos obrigatórios, pois as instâncias estão sempre conectadas. Para casos 1-1 com opcionalidades envolvidas pode-se adotar outras alternativas, pois a fusão pode ser uma estratégia inadequada se poucas instâncias das entidades envolvidas participam do relacionamento no mundo real.

Relacionamentos 1-n podem ser convertidos pelas alternativas de *aninhamento* e *referência*. A primeira é sugerida quando uma entidade E_x se relaciona com cardinalidade

Tabela 2.1. Conversão de relacionamentos de um esquema EER baseada na cardinalidade

cardinalidade	fusão	aninhamento	referência
(1,1)-(1,1)	X		
(1,1)-(0,1)	X	X	
(0,1)-(0,1)	X	X	X
(1,1)-(1,n) / (1,1)-(0,n)		X	X
(0,1)-(0,n)		X	X
(1,n)-(1,n) / (1,n)-(0,n) / (0,n)-(0,n)			X

máxima 1 de forma obrigatória, denotando que E_x está sempre atrelada a uma única entidade E_y e pode, assim, estar aninhada à coleção que representa E_y. A segunda alternativa é mais indicada quando há opcionalidades em ambos os casos e poucas instâncias de E_x participam do relacionamento no mundo real.

Por fim, relacionamentos m-n são convertidos através da alternativa de *referência* e, neste caso, sugere-se que o bloco aninhado contendo o relacionamento seja posicionado na coleção correspondente à entidade E_i que ou possui maior probabilidade de participar do relacionamento ou E_i é a entidade mais solicitada no *workload* da aplicação para fins de navegação pelo relacionamento. Cabe observar aqui que, diferente da alternativa de conversão para BDs relacionais, relacionamentos m-n não geram uma coleção própria. Essa alternativa não é considerada aqui para evitar a geração de mais uma coleção no BD NoSQL e também por que o modelo de agregados permite definir blocos aninhados a coleções com cardinalidade associada, o que é capaz de simular um relacionamento m-n em uma direção e sentido específicos. Nada impede que o projetista do BD defina blocos aninhados contendo o relacionamento em ambas as coleções geradas para as entidades envolvidas caso esse cenário seja mais favorável para o *workload* da aplicação. Este raciocínio de conversão pode ser aplicado também a relacionamentos n-ários, ou seja, relacionamentos ternários ou superiores. Nestes casos, blocos com referências para as coleções correspondentes às entidades envolvidas são definidos.

2.3.4. Projeto Lógico de Baixo Nível para BDs NoSQL

Esta subetapa é responsável pela conversão do esquema de agregados para esquemas representados nos modelos chave-valor, orientados a colunas e/ou orientados a documentos. Estes 3 modelos de dados NoSQL são centrados em acesso via chave e podem suportar conteúdos com estruturas aninhadas, apresentando uma forte similaridade com o modelo de agregados. Nesta etapa, cada coleção do esquema de agregados é analisada e convertida em uma estrutura adequada a um modelo de dados NoSQL.

A Tabela 2.2 sumariza o mapeamento de uma coleção para o modelo chave-valor. Este mapeamento é mais simples em função da simplicidade do modelo chave-valor. Neste caso, o ID da coleção se torna a chave de um esquema chave-valor e o conteúdo completo da coleção deve ser desaninhado e serializado gerando um conjunto de pares chave-valor para ser inserido no componente valor.

A Figura 2.10 mostra um exemplo deste mapeamento considerando novamente o

Tabela 2.2. Conversão de agregados para chave-valor

modelo de agregados	modelo chave-valor
ID da coleção	chave
conteúdo da coleção	valor

domínio de um Museu. Supondo o esquema no modelo de agregados na parte esquerda da figura e uma instância de *Obra* neste esquema com identificação *O145* referente à pintura *Abaporu* de *Tarsila do Amaral*, ela se torna uma instância em um BD chave-valor, conforme mostrado à direita da figura. Blocos internos são desaninhados e serializados, como é o caso de *autoria*. Conteúdos multivalorados, como o atributo *horários* em *Salão*, são igualmente serializados.

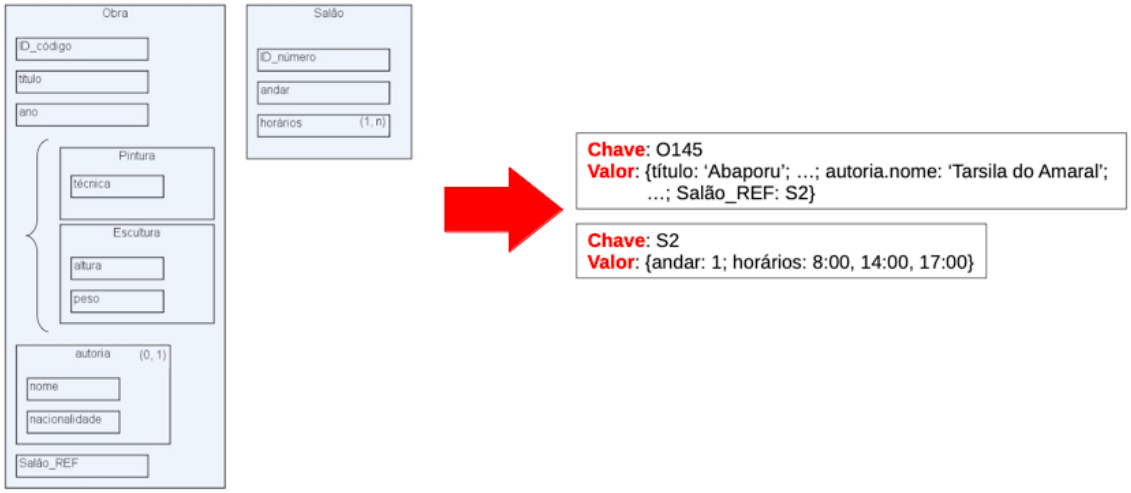


Figura 2.10. Exemplo de conversão de um esquema de agregados para um esquema chave-valor

A Tabela 2.3 sumariza o mapeamento de uma coleção para o modelo orientado a colunas. Este modelo de BD NoSQL, por ser mais robusto, permite um mapeamento mais detalhado. Entretanto, devido à falta de homogeneidade nas implementações deste modelo nos vários SGBDs orientados a colunas, algumas recomendações de mapeamento podem ser implementadas de formas diferentes, como são os casos do mapeamento de atributo multivalorado e bloco. No primeiro caso, se o SGBD suportar colunas multivaloradas, o mapeamento é trivial. Senão, o conteúdo do atributo deve ser serializado. No segundo caso, se o SGBD suportar supercolunas, o mapeamento é igualmente trivial. Caso contrário, o conteúdo do bloco deve ser desaninhado e serializado.

Um exemplo de conversão é mostrado na Figura 2.11, supondo o suporte à colunas multivaloradas e supercolunas. As instâncias exemplo são as mesmas mencionadas anteriormente. Basicamente, cada coleção se torna uma família de colunas.

Por fim, a Tabela 2.4 sumariza o mapeamento de uma coleção para o modelo orientado a documentos. Este modelo de BD NoSQL é o que apresenta maior afinidade com o modelo de agregados.

Um exemplo de conversão é mostrado na Figura 2.12 considerando que o SGBD

Tabela 2.3. Conversão de agregados para orientado a colunas

modelo de agregados	modelo orientado a colunas
coleção	família de colunas
ID da coleção	chave da família de colunas
atributo simples	coluna
atributo de referência	coluna
atributo multivalorado	coluna multivalorada ou coluna com conteúdo serializado
bloco	supercoluna ou conteúdo desaninhado e serializado

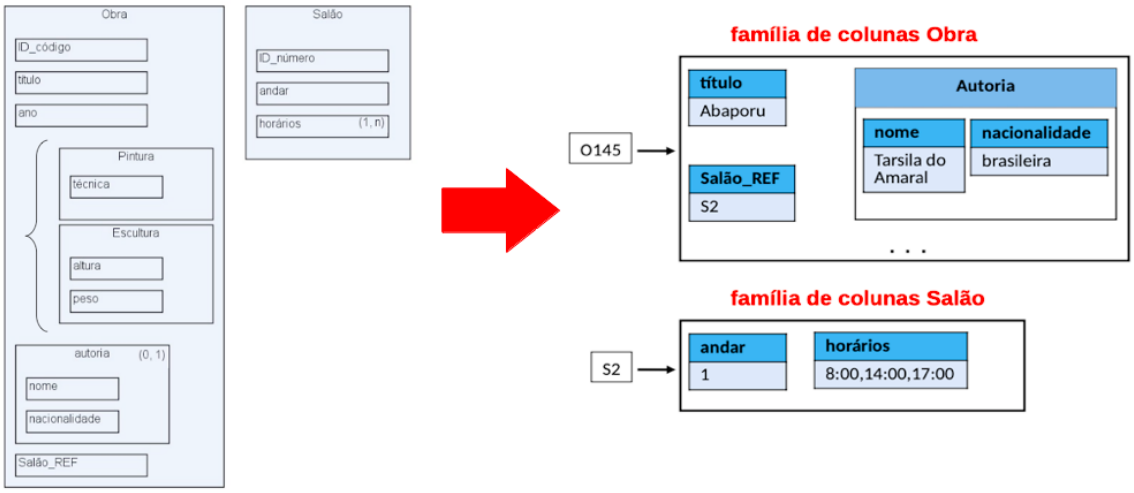


Figura 2.11. Exemplo de conversão de um esquema de agregados para um esquema orientado a colunas

Tabela 2.4. Conversão de agregados para orientado a documentos

modelo de agregados	modelo orientado a documentos
coleção	documento
ID da coleção	chave do documento
atributo simples	atributo simples
atributo de referência	atributo simples
atributo multivalorado	atributo do tipo lista
bloco	atributo do tipo objeto

orientado a documentos segue o formato JSON para fins de armazenamento. Os recursos do modelo de dados JSON, como atributos do tipo lista e do tipo objeto, permitem uma conversão bastante direta de uma instância complexa do modelo de agregados para um objeto JSON.

Na sequência aborda-se o projeto lógico para BDs orientados a grafos.

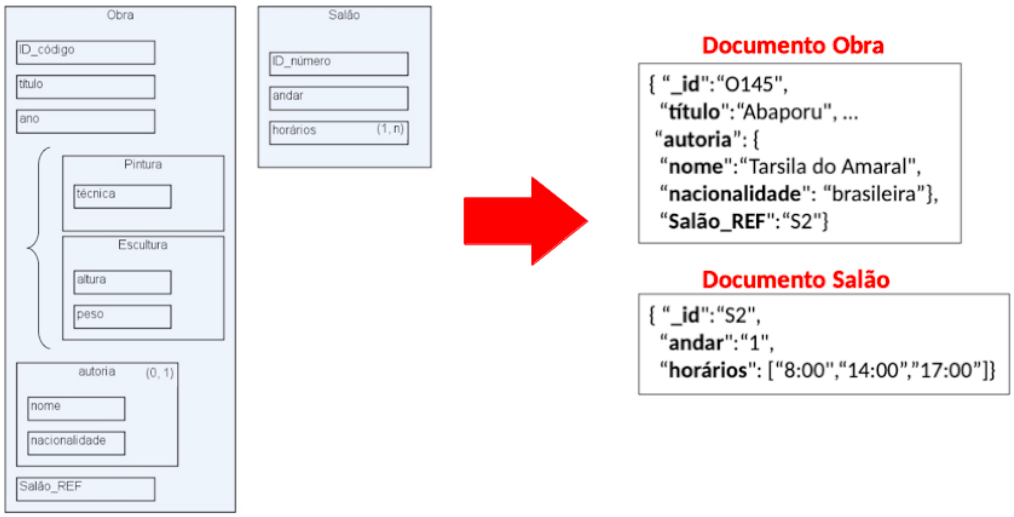


Figura 2.12. Exemplo de conversão de um esquema de agregados para um esquema orientado a documentos

2.3.5. Projeto Lógico para BDs NoSQL Orientado a Grafos

O modelo de dados orientado a grafos se diferencia dos demais modelos de dados NoSQL pelo foco das suas consultas, que são centradas em relacionamentos ao invés de acessos via chave. Em geral, seus atributos possuem domínios simples, com alguns SGBDs oferecendo suporte a atributos do tipo *array*. Desta forma, ao invés de considerar um mapeamento intermediário para o modelo de agregados, sugere-se um mapeamento direto de um esquema EER para um esquema orientado a grafos, considerando que um esquema EER já é uma estrutura em forma de grafo.

A Tabela 2.5 apresenta o mapeamento de um esquema EER para um esquema orientado a grafo. Basicamente, uma entidade se torna um tipo de vértice e um relacionamento binário um tipo de aresta, sendo que ambos podem conter propriedades, e os nomes de entidades e de relacionamentos binários se tornam os rótulos dos vértices e das arestas, respectivamente.

Tabela 2.5. Conversão de um esquema EER para um BD orientado a grafos

modelo EER	modelo orientado a grafos
entidade	vértice
atributo simples	propriedade
atributo composto	vértice
atributo multivalorado	propriedade *array* ou vértice
relacionamento binário	aresta
relacionamento n-ário	vértice

Arestas em BDs orientados a grafos geralmente são unidirecionais. Logo, a decisão pelos vértices de origem e de destino podem ser guiados pelo *workload* típico da aplicação, ou seja, qual sentido é mais frequente nas buscas. Caso ambos os sentidos sejam frequentemente percorridos em buscas, deve-se criar duas arestas com sentidos opostos entre os vértices em questão.

Com relação aos atributos do modelo EER, atributos simples tornam-se propriedades de vértices ou de arestas. Atributos compostos, não suportados em geral pelos SGBDs orientados a grafos, devem gerar novos vértices no grafo conectados ao vértice que representa a entidade com o atributo composto. Já atributos multivalorados podem ser convertios em propriedades do tipo *array*, caso o SGBD orientado a grafo o suporte, ou devem gerar vértices para representar cada valor, além de arestas que os conectem ao vértice que representa a entidade.

Por fim, sugere-se que um relacionamento n-ário R_n seja mapeado para um vértice no grafo. Isso evita a manutenção de muitas referências em cada vértice representativo de cada entidade E_i participante de R_n. Cada E_i deve ter arestas para as ocorrências de R_n que participa.

A literatura carece de uma metodologia efetiva para o projeto lógico de um BD orientado a grafo. A metodologia aqui proposta se baseia nas recomendações do recente trabalho de Silva & Mello [Silva and Mello 2021] e nas regras de conversão presentes na Tabela 2.5. O processo de conversão é composto de 3 etapas: *(i) conversão de entidades; (ii) conversão de hierarquias*; e *(iii) conversão de relacionamentos*. A primeira etapa analisa entidades não envolvidas em hierarquias de especialização, gerando um tipo de vértice para cada uma delas. A segunda etapa analisa hierarquias de especialização de entidades (percorrimento *bottom-up*), sendo possível fusionar a entidade genérica com uma ou mais entidades especializadas em um único tipo de vértice. Por fim, os relacionamentos são analisados e convertidos em tipos de aresta com uma cardinalidade associada. Também é possível fusionar entidades conectadas por relacionamentos 1-1 em um único tipo de vértice para reduzir o tamanho do esquema, caso grande parte das instâncias destas entidades participam do relacionamento.

A Figura 2.13 ilustra um exemplo de conversão. O esquema EER mostrado na Figura 2.13 (a) é um esquema simplificado para o domínio de um Museu. Um possível esquema lógico orientado a grafo é mostrado na Figura 2.13 (b). As entidades *Salão*, *Autor* e *TiposPedra* são tratadas pela primeira etapa gerando os respectivos tipos de vértices.

Na sequência, a hierarquia de especialização é analisada iniciando pelo nível de *Escultura* e suas especializações. Neste caso, optou por fusionar esse nível em um único tipo de vértice (Escultura) considerando que a especialização é compartilhada e para redução do tamanho do esquema. O nível superior é então tratado, gerando um tipo de vértice para cada entidade (*Obra*, *Pintura* e *Escultura*) e arestas conectando-os.

Por fim, analisa-se os relacionamentos. Optou-se pela fusão da entidade *Zelador* com a entidade *Salão* pelo fato do relacionamento ser do tipo 1-1 obrigatório. Os demais relacionamentos podem ser convertidos considerando o *workload* da aplicação para determinar a melhor direção e sentido da aresta. No caso do relacionamento entre *Salão* e *Obra* optou-se por gerar 2 arestas com sentidos diferentes supondo que este relacionamento é percorrido com frequência em ambos os sentidos.

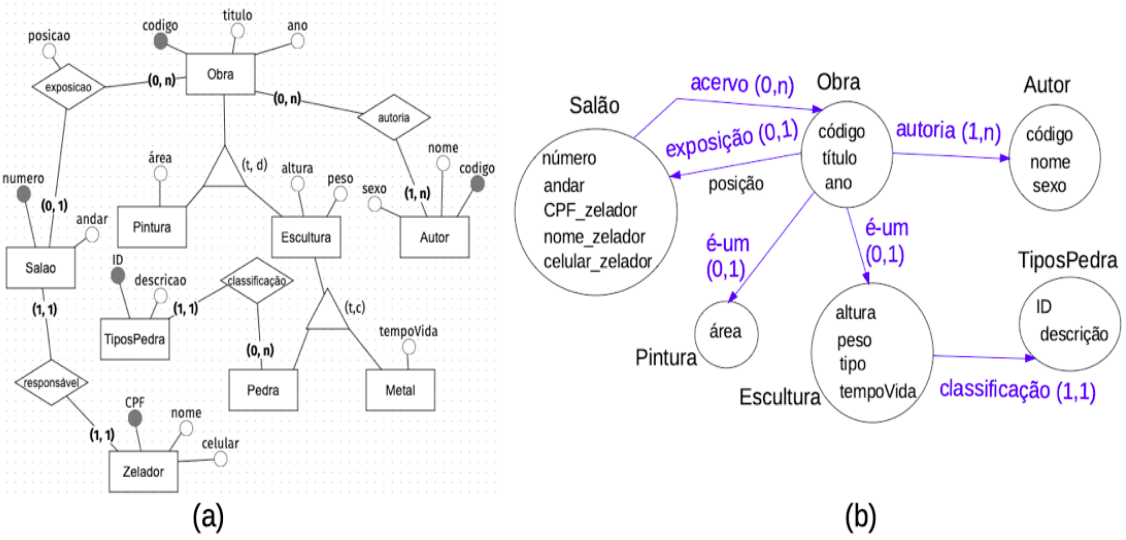


Figura 2.13. Exemplo de conversão de um esquema EER para um esquema orientado a grafos

2.4. Implementação de um Projeto lógico de BD NoSQL

A implementação do projeto lógico é a etapa que realiza a transformação do projeto lógico em um projeto físico específico de um SGBD. Enquanto na modelagem conceitual o foco é esboçar a relação entre os conceitos do domínio, o modelo lógico foca no armazenamento considerando as peculiaridades de cada modelo de dados. O modelo físico de um BD é ainda mais específico considerando as características do SGBD que será utilizado para o armazenamento dos dados. Desta forma, nesta etapa, são realizadas escolhas que definem como se dará o acesso aos dados e como otimizar esse processo.

O projeto de um BD relacional permite que o usuário seja capaz de combinar diferentes tabelas com junções para gerar o resultado de uma consulta. O projeto de um BD NoSQL, entretanto, não permite, de maneira geral, junções de diferentes estruturas de dados. Esta limitação faz com que os dados sejam organizados de tal maneira que uma junção, por exemplo, não seja necessária.

Durante a implementação do esquema lógico em um SGBD NoSQL é necessário que se leve em consideração o *workload* das principais consultas. As consultas mais comuns devem ser utilizadas como guia na transformação do esquema lógico. Durante a transformação considera-se constantemente o *tradeoff* do custo efetivo de consultar os dados que estão armazenados através de agregados ou a necessidade de combiná-los em memória através de operações realizadas pela aplicação. Desta forma, é importante manter em mente que o esquema deve atender de maneira efetiva as consultas mais frequentes.

Outro ponto importante a ser considerado na implementação do esquema lógico é o SGBD NoSQL a ser utilizado. Essa é uma decisão complexa que envolve um estudo das características básicas da aplicação. A escolha de uma modelo de dados inadequado para um determinado problema, ou mesmo, um SGBD inadequado pode levar a solução proposta a problemas de escalabilidade no geral. Cada modelo de dados e cada SGBD atendem um conjunto de requisitos e um conjunto de aplicações distinto. Por exemplo, se a modelagem dos dados em questão possui alto nível de relacionamentos, é provável que um SGBD orientado a grafo seja uma boa opção. Ou mesmo, se a aplicação exige um grande número de buscas por atributos e os dados não tem muitos relacionamentos, um SGBD orientado a grafos pode não ser uma boa opção, sendo um SGBD orientado a documentos mais adequado. A escolha do modelo e do SGBD é muito dependente dos requisitos, e é importante que essa escolha seja realizada de maneira consciente para que sejam evitados possíveis problemas posteriores.

Nas próximas seções são discutidas duas implementações de esquemas lógicos nos SGBDs NoSQL MongoDB e Neo4J por serem os mais populares no mundo NoSQL. Como base é utilizado o esquema EER para um domínio de uma rede social apresentado na Figura 2.14. O domínio é uma rede social simples onde um *Perfil* possui pode postar, comentar e curtir (*like*) um *Post*. Adicionalmente, para cada *perfil* são armazenadas informações pessoais (*Sobre*). Para cada um dos SGBDs NoSQL apresentados é proposto um mapeamento lógico e físico para o domínio em questão.

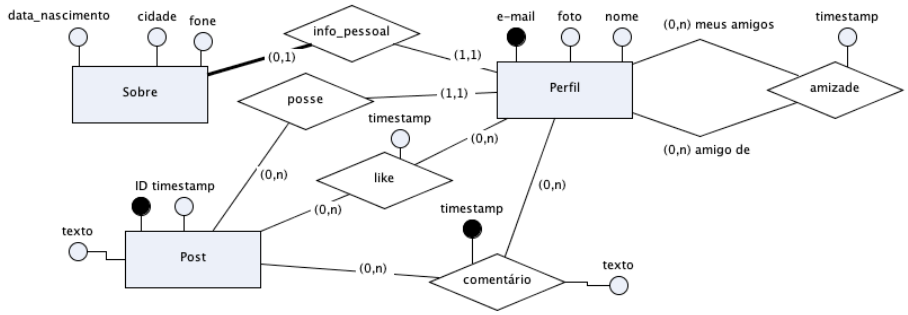


Figura 2.14. Exemplo de esquema EER para uma rede social

2.4.1. Implementação no SGBD MongoDB

O MongoDB é um SGBD NoSQL cujo modelo de dados é orientado a documentos. Ele provê alta disponibilidade, escalabilidade e uma forma flexível de armazenar os dados. Segundo o *ranking* do *DB-Engine*[3], o Mongo DB é o quinto SGBD mais utilizado na atualidade. Sua estrutura de dados é baseada no formato *JSON*, armazenando seus dados em documentos *BSON* (*JSON* binário).

A estrutura interna de um BD em *MongoDB* pode ser definida através de um conjunto de coleções de documentos. Cada coleção armazena um conjunto de documentos que, em geral, possui uma similaridade estrutural ou semântica. Analogamente ao modelo relacional, uma coleção de documentos pode ser vista como uma tabela do BD relacional [Sadalage and Fowler 2013]. Por sua vez, cada documento é composto por um conjunto de atributos e valores, análogo a uma tupla.

Ao adotar o *MongoDB* alguns comportamentos internos do SGBD devem ser considerados. Diferente de um BD relacional tradicional, o *MongoDB* não suporta transações ACID. Para este SGBD as propriedades ACID são consideradas apenas em nível de documento, ou seja, durante a inserção de um documento ou ele insere o documento por inteiro ou não insere nada. Quando a inserção envolve mais de um documento, cada documento é tratado como um elemento independente.

Outro ponto importante a ser considerado é a estrutura de armazenamento. O *MongoDB* possui uma arquitetura distribuída do tipo *master* e *worker*. Qualquer nodo pode tratar demandas de leitura, mas todas as escritas são executadas no *master* para depois serem replicadas para os nodos *workers*. Desta forma, a consistência aplicada, por padrão, é eventual tendo em vista que se um dado *x* foi escrito no nodo mestre e ainda não foi replicado para um *worker* e este atender uma requisição de leitura de *x*, ele responderá com os dados desatualizados. É importante ressaltar que o tempo de replicação dos dados entre os nodos tem uma latência baixa, mas mesmo assim, não é desprezível para alguns domínios de aplicação.

O MongoDB também suporta níveis maiores de consistência usando uma parâmetro de escrita denominado de *WriteConcern*[4]. O parâmetro define o número de réplicas que deve receber o dado de maneira síncrona. Por exemplo, em um *cluster* composto de

[3] `https://db-engines.com/en/ranking`

[4] `https://www.mongodb.com/docs/manual/reference/write-concern/`

3 nodos (1 *master* e 2 *workes*) se o *write concern* for 2, o MongoDB irá gravar os dados no *master* e em pelo menos um *worker* e só então considerar que a operação foi realizada. Aumentando o *write concern* é possível atingir um alto nível de consistência, porém, a latência de atualização entre as réplicas pode vir a ser um gargalo.

Para a implementação do esquema lógico no MongoDB todas as características supracitadas devem ser consideradas. A escolha de um SGBD para o armazenamento dos dados vem recheada de prós e contras e é importante que o projetista esteja ciente delas para realizar a escolha correta. Além disso, o usuário deve conhecer o *workload* da aplicação pois a modelagem física em um sistema NoSQL é muito pautada nas consultas a serem executadas.

Não é possível criar um esquema físico para o MongoDB que possa responder com alto desempenho qualquer consulta. Um dos motivos disso é a falta de algoritmos de junção, fazendo com que para unir dados de diferentes coleções de documentos sejam tratados pela própria aplicação. Desta forma, ao modelar deve-se levar em consideração quais são as consultas com maior custo e quais são mais executadas e armazenar os dados de maneira que estas tenham o melhor desempenho possível.

Considerando o exemplo do domínio da rede social (Figura 2.14) um possível esquema lógico é apresentada na Figura 2.15. Ele considera que é comum, por exemplo, consultas que por um *post* e seus comentários bem como o número de *likes* de um *post*. Porém, na estrutura proposta, torna-se mais complexo realizar consultas como buscar todos os *posts* de um determinado usuário ou o nome do usuário que realizou um comentário.

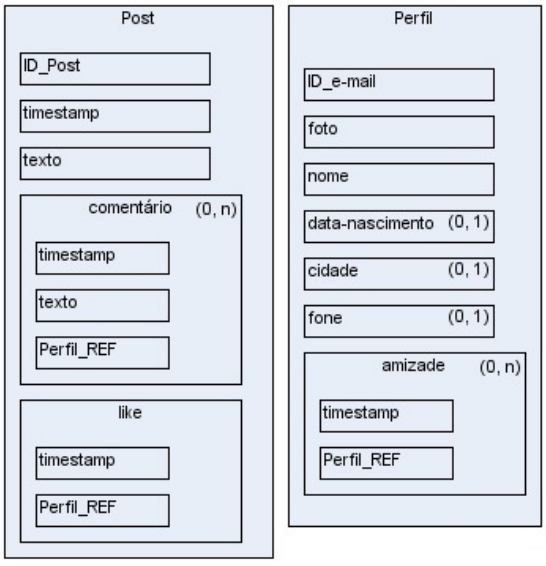


Figura 2.15. Exemplo de modelo lógico baseado em agregado.

É possível ver Figura 2.15 na que cada *Post* possui um id, um texto e conjuntos de comentários e *likes*. Cada comentário possui um texto e a referência do perfil que o criou. No Mongo DB não existe o conceito de chave estrangeira. Assim, esse atributo refere-se unicamente ao código de um documento visto com um valor inteiro qualquer.

O SGBD não realiza nenhuma verificação de integridade referencial. Apenas o usuário compreende a semântica desta ligação.

Este armazenamento obriga o usuário a implementar um algoritmo de junção para exibir o nome de um usuário que efetuou um comentário em um *post* específico. Este processo é oneroso e deve ser considerado durante a etapa de modelagem. Se esta é uma consulta comum ao sistema uma forma de mitigar a junção seria junto ao comentário já armazenar o nome do usuário que o fez. Isto aumenta a redundância de dados mas evita gasto com processamento de uma junção. Este tipo de *trade-off* deve ser considerado constantemente no processo de modelagem para que o melhor modelo possível seja gerado. Um bom esquema deve ser abrangente o suficiente para suportar uma grande gama de consultas, porém otimizado para as consultas mais comuns.

A linguagem de consulta do MongoDB se difere muito do padrão SQL. Como o SGBD trabalha com o formato JSON, seu *shell* de consulta é baseado na linguagem *Javascript*. Neste minicurso não são abordadas a instalação e configuração para o MongoDB. São apresentados alguns comandos básicos da sua *Application Programming Interface (API)* de acesso para inserção de dados e posterior consulta.

Para inserir dados no MongoDB a sua API oference os comandos *insertOne* e *insertMany*. O primeiro recebe como parâmetro um único documento a ser inserido. Já o segundo recebe um vetor com vários documentos a serem inseridos. A Figura 2.16 apresenta um exemplo de inserção de dados, sendo que $db.perfil$ representa o BD atual *db* e a coleção de documentos *perfil* onde o documento Json é inserido.

```
db.perfil.insertOne({
  "_id": 2,
  "nome": "Violet Baudelaire",
  "data_nascimento": new Date("1994-08-01"),
  "cidade": "Dream of Stone",
  "amizade": [1,3,5,8,7]
});
```

Figura 2.16. Exemplo *insert*.

Para consulta a dados, a API do MongoDB utiliza o comando *find*. Este comando recebe dois parâmetros. O primeiro é um documento que contém a busca a ser realizada e o segundo a projeção dos dados (atributos retornados). O documento de busca geralmente define operadores a serem executados, permite o uso de conectores lógicos *and* e *or*, assim como operadores lógicos > (gt), < (lt), >= (gte), <= (lte) e diferente (ne).

A Figura 2.17 apresenta um exemplo de uma busca de dados no MongoDB. Ela lista perfis ($db.perfil.find$) que sejam válidos para o predicado informado. O predicado considera um operador lógico *and* representado pelo vetor de todos os filtros conectados. Cada filtro deve explicitar sobre qual atributo deve ser executado, qual o tipo de comparação a ser feita e qual o valor usado na comparação. No caso do exemplo, a consulta retorna todos os perfis cujo nome seja *Violet Baudelaire* e tenha data de nascimento superior a $01/01/2000$.

Apesar da diferença em relação à SQL, o comando de consulta da API do MongoDB possui um alto poder de expressão, permitindo a definição de diversas consultas

```
db.perfil.find({
  $and: [
    {"nome": "Violet Baudelaire"},
    {"data_nascimento": {
        $gte: new Date("2000-01-01")
      }
    }]}, {}
});
```

Figura 2.17. Exemplo *busca*.

complexas, com a ressalva que consultas com junções não são suportadas, como mencionado anteriormente. Uma consulta particularmente complexa para ser resolvida é listar o nome de todos os amigos de um determinado perfil. Para defini-la, deve-se buscar as informações de cada *id* de amigo armazenado no vetor e realizar a junção com a própria coleção de documentos *perfil*. Se expandirmos a consulta para mostrar o nome dos amigos dos amigos de um dado perfil então a consulta fica inviável. Por outro lado, esse tipo de consulta é plenamente viável em um BD orientado a grafo, como o Neo4j, cuja implementação é detalhada a seguir.

2.4.2. Implementação no SGBD Neo4j

O Neo4J é um SGBD orientado a grafos. Sua estrutura interna é otimizada para o armazenamento de dados que possuam um alto número de relacionamentos. Diferente do MongoDB, que pode ser adaptado e aplicado em diversas gamas de problemas, o Neo4J é otimizado para o armazenamento de dados que possuam muitos relacionamentos.

O modelo de dados do Neo4J é baseado majoritariamente em dois conceitos: *vértices* e *arestas*. Um vértice é definido por um *label*, um ou mais tipos e um conjunto de atributos armazenado no formato JSON. Já uma aresta é um relacionamento que conecta dois vértices. Cada aresta possui um vértice de origem, um vértice de destino, um *label*, um ou mais tipos e um conjunto de atributos também no estilo JSON.

Diferente da maioria dos BDs NoSQL, o Neo4J possui suporte às propriedades ACID. Quando um conjunto de dados é adicionado ou atualizado em conjunto, todas as operações devem ser executadas até o fim, ou nenhuma delas é efetivada. Porém, para que o ACID seja melhor controlado, o Neo4J não permite o particionamento dos dados.

O Neo4J utiliza a linguagem *Cypher* para executar suas operações. Cypher é uma linguagem declarativa imperativa com alto poder de expressão e que possui semelhanças com a SQL. Considerando o domínio da rede social (Figura 2.14), um possível esquema lógico é apresentado na Figura 2.18. Diferente do esquema orientado a documentos apresentado anteriormente, o modelo de BD orientado a grafo é focado nos relacionamento dos dados. Desta forma, os relacionamentos *postar*, *comentar* e dar *like* são mapeados diretamente para arestas que unem vértices do tipo *Perfil* com *Post*. Além disso, o autorelacionamento de um perfil (*amizade*) também é mapeado para uma aresta.

O modelo de dados orientado a grafo permite que o projetista defina um conjunto de atributos para cada vértice e aresta. O Neo4j permite que sejam definidos filtros por valores de atributos, entretanto, este tipo de consulta não é tão otimizado quanto buscas

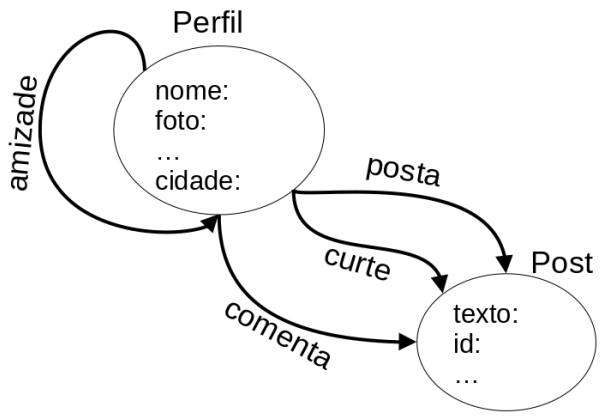


Figura 2.18. Exemplo de esquema lógico para um BD em grafo

envolvendo navegação pelas arestas.

A inserção de dados no Neo4J é realizada através do comando *CREATE* da linguagem Cypher. Para tanto, elementos delimitados por ( ) são considerados vértices e elementos delimitados por $-[]->$ são arestas. Existem duas formas de realizar a inserção dos dados. A primeira realiza a criação de todos os elementos diretamente em um único comando. Um exemplo é apresentado nas 2 primeiras linhas da Figura 2.19. O comando cria dois vértices (*Violet* e *Sunny*), ambos do tipo *Perfil*. O tipo serve para indicar vértices que possuem a mesma estrutura e semântica.

```
CREATE  (s:Perfil {name: "Sunny"}),
        (v:Perfil {name: "Violet"});


MATCH   (s:Perfil {name: "Sunny"}),
        (v:Perfil {name: "Violet"})
CREATE (s)-[:rel]->(v)
```

Figura 2.19. Exemplos de inserção de dados na linguagem Cypher

Já a segunda alternativa utiliza o comando *MATCH* além do comando CREATE. O comando MATCH é utilizado para buscar elementos em um grafo. No exemplo da Figura 2.19, busca-se 2 vértices do tipo *Perfil*, filtrados pelo atributo *nome*, e define-se uma aresta entre os dois vértices, caso eles sejam encontrados.

A tipagem (*labels*) dos vértices e das arestas em um grafo é um importante recurso na definição de consultas. Cada vértice ou aresta pode possuir mais de um tipo, o que permite uma maior flexibilidade de representação. A Figura 2.20 apresenta uma representação gráfica de um conjunto de dados armazenado no Neo4J. Vértices de mesma cor são de um mesmo tipo. Perceba também que as arestas são todas unidirecionais. O Neo4J não suporta arestas bidirecionais.

Conforme descrito anteriormente, a consulta a dados na linguagem Cypher é definida pelo comando *MATCH*. Este comando visa encontrar um conjunto (mesmo que unitário) de vértices e arestas que satisfaçam suas restrições. Para cada vértice ou aresta

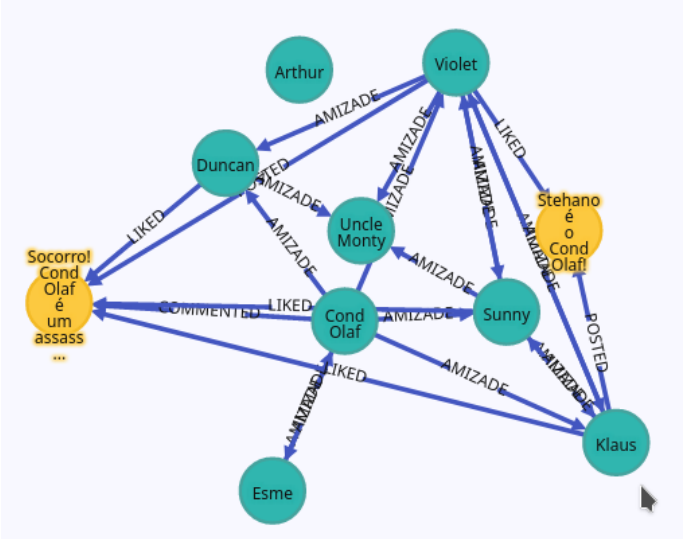


Figura 2.20. Exemplo de uma instância de grafo no Neo4j

envolvido na consulta é possível atribuir um *alias* facilitando operações complexas que exijam filtros com operadores *and* e *or*.

```
MATCH (s:Perfil)-[:AMIZADE]->(y)
WHERE s.nome = 'Sunny'
RETURN y.nome, y.data_nascimento
      '

MATCH (s:Perfil)-[:AMIZADE]->()-[:AMIZADE]->(y)
WHERE s.nome = 'Sunny'
RETURN y.nome
```

Figura 2.21. Exemplos de consultas na linguagem Cypher

A Figura 2.21 apresenta 2 consultas em Cypher. A primeira retorna o nome e a data de nascimento de todos os vértices que possuem um relacionamento do tipo amizade com o vértice de nome *Sunny*. Apesar de uma consulta relativamente simples, este exemplo ilustra a estrutura básica de uma consulta na linguagem Cypher. A primeira parte da consulta é destinada a busca por um padrão (*MATCH*), a segunda parte é o predicado da consulta (*WHERE*) e, por fim, o resultado da consulta (*RETURN*).

Já a segunda consulta é um pouco mais complexa. Ela realiza a seleção dos vértices (y) que possuem um relacionamento de amizade com algum nodo que já possui relacionamento de amizade com o vértice *Sunny*. Em outras palavras, a consulta retorna o nome dos amigos dos amigos de *Sunny*. Esta é uma consulta que teria um alto custo de execução em outros modelos de dados NoSQL, entretanto, no modelo orientando a grafos é relativamente simples devido à sua ênfase no percorrimento por relacionamentos entre dados.

2.5. Considerações Finais

Este minicurso apresenta uma proposta de metodologia para projeto lógico de BDs NoSQL a partir de um esquema conceitual EER. Esta metodologia contempla todos os 4 modelos de dados NoSQL e também discute como a implementação de um esquema lógico para NoSQL pode ser implementada em 2 SGBDs de amplo uso: MongoDB (orientado a documentos) e Neo4j (orientado a grafos).

Esta metodologia é uma contribuição para a área de modelagem de dados NoSQL, sendo um guia detalhado do processo de mapeamento de esquemas conceituais para projetistas que desejam utilizar SGBDs NoSQL para o armazenamento e acesso aos seus dados. A literatura de BD é carente de tal metodologia detalhada. A proposta aqui apresentada se baseia no projeto de BD clássico e na adaptação das regras de conversão EER-relacional. Como em qualquer projeto de BD para um dado modelo lógico e físico de dados, a metodologia proposta apresenta diversas alternativas de conversão de conceitos de um esquema EER para estruturas correspondentes nos modelos de dados NoSQL, cabendo ao projetista decidir pelas alternativas que melhor contemplam as necessidades de armazenamento e acesso aos seus dados.

O *Grupo de BD da UFSC*[5] vêm trabalhando em soluções informatizadas para o projeto de BDs, com destaque para o protótipo *brModeloNext*, que está sendo estendido para suportar a modelagem lógica de agregados e consequente geração de esquemas físicos em alguns SGBDs NoSQL. Espera-se que em breve esta ferramenta esteja estável e disponível para uso pela comunidade acadêmica para ensino e prática de projeto de BD NoSQL, bem como para demais interessados.

[5] *http://lisa.inf.ufsc.br/; https://github.com/gbd-ufsc*

Referências

[Atzeni et al. 2014] Atzeni, P., Bugiotti, F., and Rossi, L. (2014). Uniform Access to NoSQL Systems. *Information Systems*, 43(SI):117–133.

[Elmasri and Navathe 2011] Elmasri, R. and Navathe, S. (2011). *Fundamentals of Database Systems*. Addison-Wesley, 6 edition.

[Frozza and Mello 2020] Frozza, A. and Mello, R. (2020). JS4Geo: A Canonical JSON Schema for Geographic Data Suitable to NoSQL Databases. *GeoInf.*, 24(4):987–1019.

[Henry Korth and Silberschatz 2019] Henry Korth, S. S. and Silberschatz, A. (2019). *Database System Concepts*. McGraw-Hill Education, 7 edition.

[Heuser 2009] Heuser, C. A. (2009). *Projeto de Banco de Dados*. Bookman, 6 edition.

[Lima 2016] Lima, C. (2016). Projeto Lógico de Bancos de Dados NoSQL Documento a Partir de Esquemas Conceituais Entidade-Relacionamento Estendido (EER). Master's thesis, PPGCC-UFSC.

[Lima and Mello 2015] Lima, C. and Mello, R. (2015). A Workload-driven Logical Design Approach for NoSQL Document Databases. In *XVII ACM iiWAS*, pages 73:1–73:10.

[Lima and Mello 2016] Lima, C. and Mello, R. (2016). On Proposing and Evaluating a NoSQL Document Database Logical Design Approach. *Int. J. Web Inf. Syst.*, 12(4):398–417.

[Sadalage and Fowler 2013] Sadalage, P. J. and Fowler, M. (2013). *NoSQL Distilled : A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.

[Santana and Mello 2020] Santana, L. H. Z. and Mello, R. (2020). An Analysis of Mapping Strategies for Storing RDF Data into NoSQL Databases. In *XXXV ACM SAC*, pages 386–392.

[Schreiner et al. 2020] Schreiner, G., Duarte, D., and Mello, R. (2020). Bringing SQL Databases to Key-based NoSQL Databases: A Canonical Approach. *Computing*, 102(1):221–246.

[Schreiner et al. 2015] Schreiner, G. A., Duarte, D., and Mello, R. (2015). SQLtoKeyNoSQL: A Layer for Relational to Key-based NoSQL Database Mapping. In *XVII ACM iiWAS*, pages 74:1–74:9. ACM.

[Schreiner et al. 2019] Schreiner, G. A., Duarte, D., and Mello, R. (2019). When Relational-Based Applications Go to NoSQL Databases: A Survey. *Information*, 10(7):241–263.

[Silva and Mello 2021] Silva, T. H. V. and Mello, R. (2021). A Rule-based Conversion of an EER Schema to Neo4j Schema Constraints. In *XXXVI SBBD*, pages 181–192.

[Stonebraker 2010] Stonebraker, M. (2010). SQL Databases vs. NoSQL Databases. *Commun. ACM*, 53(4):10–11.

sbbd:03

Capítulo 3

Uso de *Meta-Learning* em Tarefas de Aprendizado Profundo

Luis Gustavo Coutinho do Rêgo, Bárbara Stéphanie Neves Oliveira, Lucas Peres Gaspar, João Araújo Castelo Branco, José Antônio Fernandes de Macêdo

Abstract

Deep Learning is a subarea of Machine Learning that uses neural networks with successive layers of data representation, which allow the performance of more complex tasks. Generally, these models produce better results when working with large volumes of data. However, in some situations, obtaining labeled data in large quantities for training these networks is not feasible. The meta-learning strategy aims to mitigate this problem, enabling learning models to learn quickly from other models initially trained for different tasks. This work introduces some meta-learning techniques, focusing on their use with Deep Learning models to solve tasks with fewer data.

Resumo

Aprendizado Profundo é uma subárea de Aprendizado de Máquina que utiliza redes neurais com sucessíveis camadas de representação dos dados, as quais permitem a realização de tarefas mais complexas. Em geral, esses modelos produzem melhores resultados quando trabalhados com grandes volumes de dados. Entretanto, em algumas situações, não é possível obter dados rotulados em grande quantidade para o treinamento dessas redes. A estratégia de meta-learning visa mitigar esse problema, fazendo com que modelos de aprendizagem consigam aprender, de forma rápida, a partir de outros modelos inicialmente treinados para diferentes tarefas. Este trabalho introduz algumas técnicas de meta-learning, focando em seu uso com modelos de Aprendizado Profundo para a resolução de tarefas com quantidade de dados reduzida.

3.1. Introdução

Nos últimos anos, o Aprendizado Profundo, do inglês *Deep Learning* (também conhecido como Aprendizagem Profunda ou redes neurais profundas), impulsionou um rápido progresso em campos diversos como Visão Computacional e Processamento de Linguagem

Natural [Neves Oliveira et al. 2022a]. Dada a aplicabilidade desse aprendizado de representação em vários níveis por meio de muitas camadas de transformações, o Aprendizado Profundo substitui não apenas modelos superficiais de *pipelines* tradicionais de Aprendizado de Máquina, como também o processo trabalhoso da engenharia de *features*.

Grande parte do progresso recente do Aprendizado Profundo foi desencadeado por uma abundância de dados provenientes de sensores e aplicações da *Web*. No entanto, apesar de muitos avanços, ainda há desafios a serem resolvidos, como precisar de grandes quantidades de dados para obter bons desempenhos, além de conseguir assimilar novos padrões presentes nos dados de forma rápida e menos custosa [Huisman et al. 2021]. À medida que o uso de modelos de Aprendizado Profundo aumentou, as dificuldades para contornar essas limitações e treinar esses algoritmos originaram um aumento no interesse por estudos de *meta-learning*.

Abordagens de *meta-learning* [Thrun 1998] são usadas para melhorarem soluções de modelos de aprendizagem ao replicar a capacidade de adaptação da mente humana em diferentes tarefas. Um dos aspectos interessantes da inteligência humana é sua capacidade em adaptar-se rapidamente a novos problemas e padrões com apenas poucas informações. Por exemplo, uma pessoa que sabe pilotar um carro manual terá facilidade em aprender a pilotar um carro automático. Isto deve-se ao fato de que humanos conseguem utilizar suas experiências passadas em contextos similares para aprender a realizar algo novo [Lake et al. 2017].

O conceito chave da estratégia descrita é: assim como as pessoas, modelos de aprendizagem também devem conseguir aproveitar o conhecimento obtido de tarefas passadas [Schmidhuber 1987] para adequarem-se mais rapidamente a novas tarefas utilizando poucas informações (dados). Dessa forma, abordagens de *meta-learning* visam "aprender como modelos de aprendizagem aprendem", ao invés de apenas generalizarem os dados de uma tarefa específica, tal como as abordagens tradicionais operam.

Essa solução tem sido utilizada em diversos cenários e obtido bons resultados quando comparada às soluções convencionais. No cenário de sistemas de recomendação, técnicas de *meta-learning* podem mitigar o problema de *cold-start* de uma recomendação, uma vez que não se possuem informações prévias sobre um determinado usuário [Wang et al. 2022]. No contexto de trajetórias, a tarefa de planejamento de rotas em terrenos desconhecidos pode beneficiar-se dos conceitos de *meta-learning* dada a falta de informações e o baixo poder computacional de equipamentos embarcados contidos em veículos autônomos usados em experimentações [Visca et al. 2022].

Nos últimos anos, sugiram diversas estratégias para o uso de *meta-learning*. Uma forma de conseguir classificá-las é fazendo uma distinção sobre a metodologia de aprendizado que pode ser dividida em três tipos: abordagens baseadas em métricas, modelos e otimização. A Figura 3.1 mostra o desempenho de algumas dessas técnicas para a tarefa de classificação de imagens para o conjunto de dados *miniImageNet* [Vinyals et al. 2016].

O objetivo do presente trabalho é apresentar os conceitos básicos de *meta-learning* e como utilizar suas abordagens em tarefas de Aprendizado Profundo. Em resumo, este capítulo está organizado da seguinte forma: a Seção 3.2 apresenta a motivação por trás de alguns conceitos do Aprendizado Profundo e expõe as definições iniciais relacionadas ao

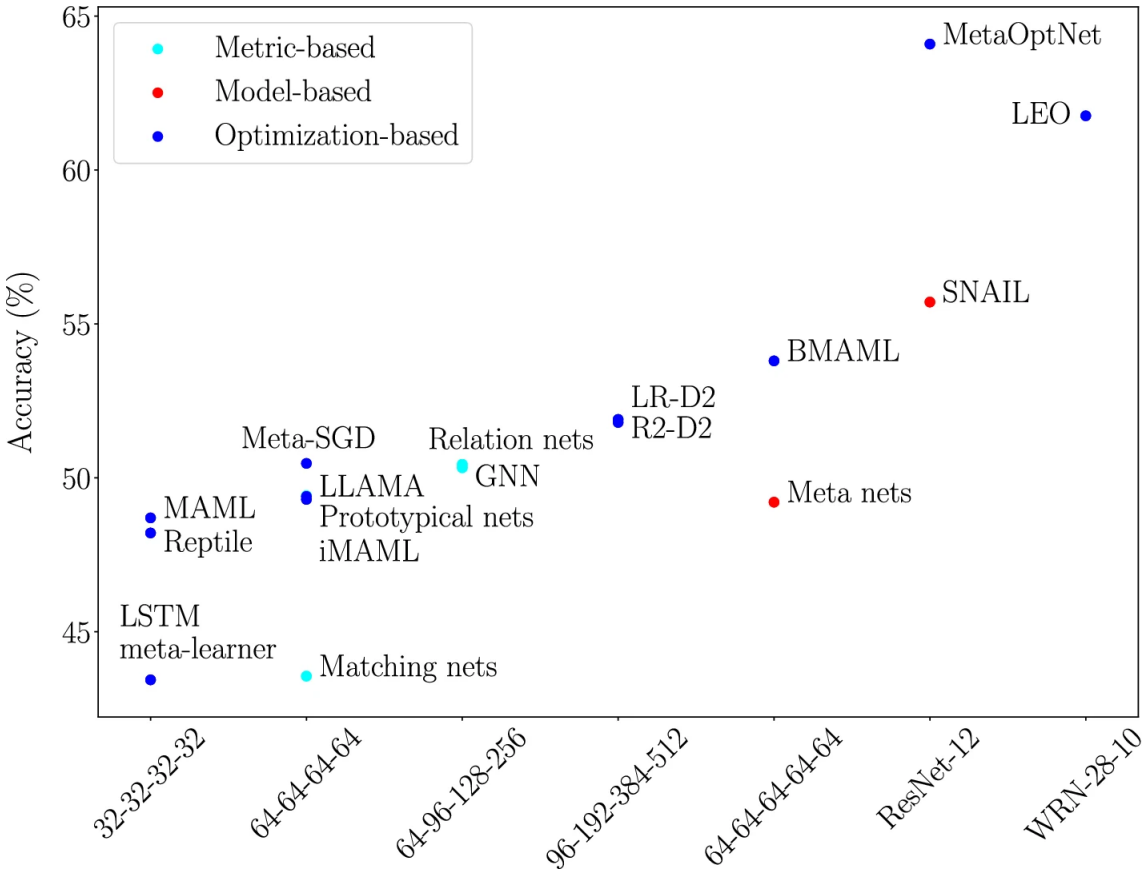


Figura 3.1. Comparativo da acurácia (eixo y) entre as abordagens do *meta-learning* para o conjunto de dados miniImageNet utilizando diferentes redes como base (eixo x) [Huisman et al. 2021].

campo de *meta-learning*. A Seção 3.3 mostra as diferentes abordagens de como resolver problemas com *meta-learning*. A Seção 3.4 apresenta um exemplo prático do uso de técnicas de *meta-learning* para a resolução de uma tarefa de classificação de imagens e, por fim, as Seções 3.5 e 3.6 apresentam algumas limitações, desafios de pesquisa e as considerações finais, respectivamente.

3.2. Fundamentação Teórica

Esta seção revisa os conceitos básicos sobre Aprendizado Supervisionado e os princípios relacionados à *meta-learning* necessários para o bom entendimento deste trabalho.

3.2.1. Aprendizado Supervisionado

A maneira usual de fazer com que uma máquina realize alguma tarefa envolve um(a) programador(a) e um *software* que contém regras a serem seguidas para transformar dados de entrada em respostas apropriadas. O Aprendizado de Máquina inverte esse processo: a máquina analisa os **dados de entrada** e as **respostas correspondentes** e descobre quais que devem ser as regras.

No Aprendizado de Máquina, essa definição trata-se de um tipo bastante espe-

cífico de aprendizagem chamada de **aprendizagem supervisionada**. Existem diferentes maneiras de construir algoritmos de Aprendizado de Máquina, cada uma com suas próprias vantagens e desvantagens. Assim, neste trabalho, apenas o aprendizado supervisionado, um dos tipos mais básicos, é referenciado para entendimento do conceito de *meta-learning*.

Tal como no Aprendizado de Máquina, o Aprendizado Profundo também possui diferentes tipos de aprendizagem. Para entender um pouco as diferenças de como o aprendizado supervisionado funciona entre modelos tradicionais e profundos, considere três fatores importantes:

1. **Dados de entrada.** Se a tarefa for identificar imagens de animais (tarefa comumente chamada de classificação de imagem), os dados de entrada serão as próprias imagens dos animais. A Figura 3.2 ilustra uma tarefa de classificação de imagens, em que os dados de entrada são imagens de formigas e joaninhas.

2. **Dados de saída que informam a que classes pertencem os dados de entrada.** Para a tarefa de classificação de imagens, são esperados valores como classes ou rótulos[1] que identificam as imagens de entrada em tipos específicos, como formiga e joaninha, dispostas em grupos com cores diferentes na Figura 3.2. A junção dos dados de entrada com os dados de saída em forma de pares (dados anotados) é comumente referenciada na literatura como conjunto de dados de treinamento (*training set*).

3. **Uma forma de medir se o modelo está realizando um bom trabalho.** Isso é necessário para determinar a distância entre o que o modelo julga ser uma certa imagem (se é uma formiga, por exemplo. O que o modelo julga ser correto é chamado de *classe predita*) e o que é realmente esperado (e se, na verdade, for uma joaninha? A informação correta sobre o que é uma imagem vinda do conjunto de dados é chamada de *classe verdadeira*). A medição é usada como um sinal de *feedback* para ajustar a maneira como o modelo funciona. Essa etapa de ajuste é o que é conhecido como aprendizado. Um modelo entende "como aprender" quando consegue generalizar o total de classes preditas com as classes verdadeiras.

Um sistema de aprendizado supervisionado é treinado em vez de explicitamente programado sendo apresentado a muitos dados e respostas relevantes para encontrar estrutura estatística que eventualmente permite que o sistema crie regras para automatizar a tarefa. Por exemplo, na Figura 3.2 a ilustração demonstra que o treinamento do modelo de classificação é realizado usando como entrada o conjunto de dados formado por muitas imagens de formigas e joaninhas. Em seguida, o conhecimento adquirido pelo modelo sobre as imagens é testado com uma nova imagem não vista anteriormente.

[1]Apesar de classes e rótulos servirem para o mesmo propósito, esses conceitos possuem algumas diferenças entre si: uma classe é uma categoria onde é possível agrupar instâncias dos dados (por exemplo, identificar quais imagens possuem apenas formigas, joaninhas, etc.), enquanto um rótulo é uma categoria que permite diferenciar os dados, ou seja, uma instância pode ser associada a um ou mais rótulos (por exemplo, identificar se uma imagem possui uma formiga e uma joaninha). Neste capítulo, os problemas apresentados lidam apenas com classes.

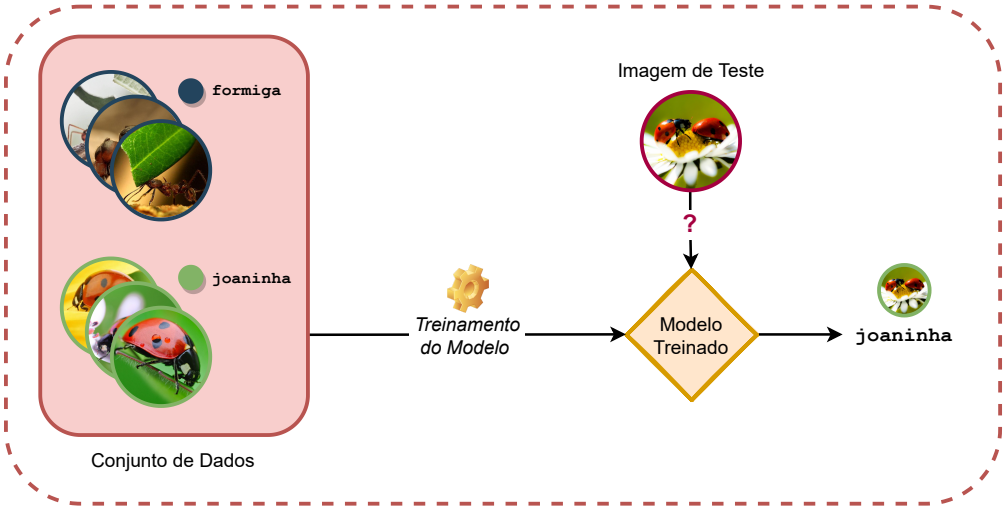


Figura 3.2. Aprendizado supervisionado para o problema de classificação de imagens. O conjunto de dados formado por imagens pertencentes às classes `formiga` e `joaninha` é usado como entrada para um modelo de classificação. Após treinado, uma nova imagem que não pertence ao conjunto inicial usado para treinar o modelo (imagem de teste) é passada para o classificador treinado. O modelo realiza uma predição correta, informando que a imagem pertence à classe `joaninha`.

O problema central em Aprendizado de Máquina e Aprendizado Profundo é transformar dados significativamente. Em outras palavras, aprender representações úteis dos dados de entrada disponíveis que aproximem das saídas esperadas. O Aprendizado Profundo oferece uma nova visão sobre o aprendizado de representações a partir do uso de camadas sucessivas de representações cada vez mais significativas por meio de redes neurais. Não só isso, mas o principal motivo que influencia na escolha desses tipos de modelos é o fato de exigirem pouca engenharia de *features* [Neves Oliveira et al. 2022b], podendo elencar automaticamente as características mais importantes presentes nos dados (como as linhas, texturas, antenas de uma formiga, patas de um gato, etc., em uma tarefa de classificação de imagens), conforme a arquitetura utilizada.

Uma prática será feita adiante com um modelo de rede neural profunda para ilustrar o funcionamento dessas camadas, mas como representações aprendidas podem parecer visualmente? Observe a Figura 3.3 que possui um exemplo de rede neural com várias camadas que transformam uma imagem de entrada de um dígito. As camadas da rede transformam a imagem do dígito em representações cada vez mais diferentes da imagem original e cada vez mais informativas sobre o resultado.

Na Seção 3.4, uma prática será feita com uma tarefa de classificação de imagens utilizando tanto Aprendizado Profundo convencional[2] quanto *meta-learning*. Nas duas abordagens, será usada como base um tipo de arquitetura de redes profundas, chamada de Redes Neurais Convolucionais (CNNs, de *Convolutional Neural Networks*), que começaram a obter ótimos resultados em competições de classificação de imagens entre os

[2]Neste trabalho, o termo "convencional" é usado para referenciar qualquer problema que não utilize conceitos de *meta-learning*. Por exemplo, tarefa de classificação *convencional* com Aprendizagem Profunda.

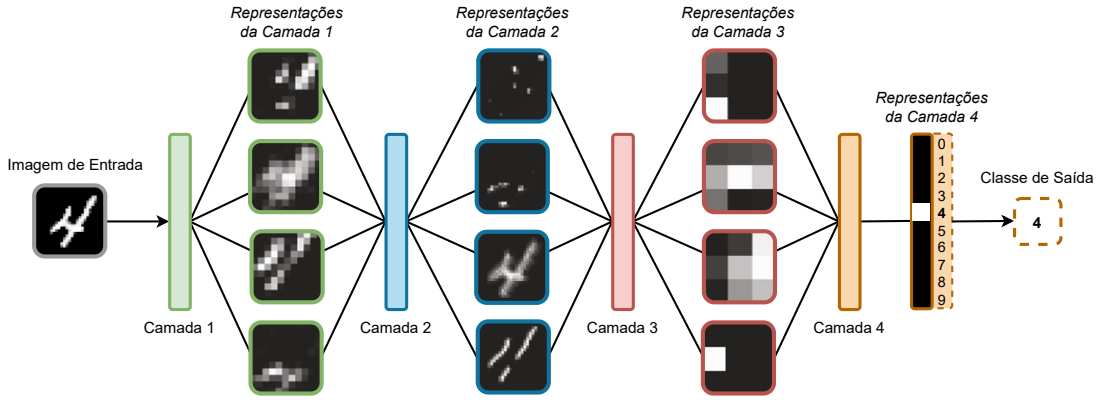


Figura 3.3. Representações de dados aprendidas por um modelo de classificação de imagens de dígitos. Adaptado de [Chollet 2021].

anos de 2011 e 2015. Mais especificamente, será usada a *ResNet* [He et al. 2016], uma modificação mais moderna das redes convolucionais.

Contudo, por que CNNs obtiveram bastante sucesso no problema de classificação de imagens? Para entender genericamente como CNNs funcionam, primeiro considere que o objetivo dessas arquiteturas é identificar algo, seja um ser vivo ou um objeto, em uma imagem. Desta forma, parece razoável afirmar que o modelo não deve preocupar-se com a localização precisa de um certo objeto ou ser na imagem. Idealmente, o modelo deve explorar um conhecimento extra sobre as imagens, isto é, precisa saber algumas das características principais que definem o que queria identificar.

Pelo fato das CNNs serem arquiteturas renomadas, a presença delas na literatura é bastante abrangente. O objetivo do restante desta subseção é apenas introduzir seus conceitos básicos intuitivamente e mostrar como usar uma CNN simples de forma prática. [Chollet 2021, Dumoulin and Visin 2016, Zhang et al. 2021] são alguns dos livros e artigos que possuem uma base sólida, da teoria à prática, sobre CNNs.

Como Usar uma Arquitetura Simples Baseada em CNN. A Figura 3.3 representa o funcionamento das camadas de um modelo de classificação de imagens de dígitos manuscritos. Será se é possível construir algo parecido (ou melhor) como o modelo ilustrado na Figura 3.3 com uma CNN?

O problema em questão consiste em classificar imagens de dígitos manuscritos em escala cinza (28×28 *pixels*) em dez (10) classes compostas por números de zero (0) a nove (9). O conjunto de dados utilizado é o MNIST [3], um clássico da comunidade de Aprendizado de Máquina, que existe há quase tanto tempo quanto o próprio campo, composto por 60.000 imagens de treinamento e mais 10.000 imagens de teste. O Programa 3.1 mostra como carregar o conjunto de dados MNIST através da biblioteca *Keras*. `train_images` e `train_classes` formam o conjunto de treinamento, dados com os quais o modelo aprenderá, e o modelo será então testado com o conjunto de teste, formado por `test_images` e `test_classes`. As imagens são codificadas como matrizes

[3] `http://yann.lecun.com/exdb/mnist/`

NumPy e as classes são uma matriz de dígitos que variam de 0 a 9. As imagens e as classes têm uma correspondência de um para um e foram montadas pelo Instituto Nacional de Padrões e Tecnologia (o NIST no MNIST) na década de 1980.

```
import numpy as np
from tensorflow.keras.datasets import mnist

(train_images, train_classes), (test_images, test_classes) =
 ↪  mnist.load_data()

print(train_images.shape, train_classes.shape)
# ((60000, 28, 28), (60000,))

print(test_images.shape, test_classes.shape)
# ((10000, 28, 28), (10000,))

print("Classes do MNIST =
 ↪  {}".format(np.unique(train_classes.tolist() +
 ↪  test_classes.tolist())))
# Classes do MNIST = [0 1 2 3 4 5 6 7 8 9]
```

Programa 3.1. Carregando o conjunto de dados MNIST com o *Keras*.

O fluxo a partir desse momento será o seguinte: primeiro, a CNN será alimentada com os dados de treinamento, `train_images` e `train_classes`. A rede então aprenderá a associar imagens e classes. Por fim, será solicitado à rede que produza predições para `test_images` e será verificado se essas predições correspondem às classes de `test_labels` (ao invés de testar para apenas uma imagem, como é demonstrado na Figura 3.3). O Programa 3.2 constrói uma arquitetura simples baseada em camadas CNN, formada apenas com camadas `Conv2D` (convolução) e `MaxPooling2D` (*pooling*).

Observe que no Programa 3.2 o modelo da CNN está configurado para processar entradas de tamanho `(28, 28, 1)` (altura da imagem, largura da imagem e canais da imagem, respectivamente)[4], que é o formado das imagens do MNIST. É possível ver no código o formato das saídas de cada camada: para a maioria delas, um tensor de tamanho três (`kernel_size=3`) na forma `(altura, largura, canais)`. As dimensões de largura e altura tendem a diminuir à medida que se passa pelas camadas. O número de canais é controlado pelo primeiro argumento `filters` passado nas camadas de convolução (filtros de tamanho 32, 64 ou 128).

Após a última camada de convolução, a saída possui a forma `(3, 3, 128)`, que consiste em um mapa de *features* 3×3 com 128 filtros. O próximo passo é usar essa saída como entrada em um classificador densamente conectado (`Dense`), que processa vetores

[4]As camadas de convolução operam sobre tensores de nível três (3) chamados de mapas de *features*, que possuem dois eixos espaciais (altura e largura), bem como um eixo de profundidade (também chamado de canal). Para uma imagem RGB, a dimensão do eixo de profundidade é 3, pois a imagem possui três canais de cores: vermelho, verde e azul. Para uma imagem em preto e branco, como os dígitos MNIST, a profundidade é 1 (níveis de cinza). Além disso, canais podem ser usados para representar filtros [Chollet 2021].

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(28, 28, 1))

# Camada/bloco 1
conv32 = layers.Conv2D(filters=32, kernel_size=3,
 ↪  activation="relu")(inputs) # Saída (26, 26, 32)
pool1 = layers.MaxPooling2D(pool_size=2)(conv32) # Saída (13, 13,
 ↪  32)

# Camada/bloco 2
conv64 = layers.Conv2D(filters=64, kernel_size=3,
 ↪  activation="relu")(pool1) # Saída (11, 11, 64)
pool2 = layers.MaxPooling2D(pool_size=2)(conv64) # Saída (5, 5,
 ↪  64)

# Camada/bloco 3
conv128 = layers.Conv2D(filters=128, kernel_size=3,
 ↪  activation="relu")(pool2) # Saída ( 3, 3, 128)
flatten = layers.Flatten()(conv128) # Saída (1152,)

# Camada 4
outputs = layers.Dense(10, activation="softmax")(flatten) # Saída
 ↪  (10,)
cnn_model = keras.Model(inputs=inputs, outputs=outputs)
```

Programa 3.2. Inicializando uma CNN simples.

de 1 dimensão (1D). Como a saída atual é um tensor de 3 dimensões, a camada `Flatten` é usada para converter 3D para 1D antes de passar para a camada `Dense`, que possui 10 saídas possíveis e uma função de ativação *softmax*[5].

O Programa 3.3 treina a CNN criada. Antes é feito um pré-processamento para padronizar os dados na forma que o modelo espera e dimensioná-los para que todos os valores estejam no intervalo $[0, 1]$. Como a saída do modelo pode conter 10 diferentes classes, é usada a função de perda (ou *loss*) *categorical crossentropy*, em especial, a versão esparsa `sparse_categorical_crossentropy`, já que as classes tratam-se de números inteiros. A CNN obtém uma acurácia de aproximadamente 99% sobre o conjunto de teste. Note que o modelo criado possui a mesma quantidade de camadas que o da Figura 3.3.

3.2.2. *Meta-Learning*

Modelos treinados utilizando arquiteturas de Aprendizado Profundo, em geral, necessitam de uma grande quantidade de dados de entrada para obterem bons resultados. Seres

[5]A função *softmax* é usada como função de ativação na camada de saída em modelos que predizem uma distribuição de probabilidade multinomial, como para os que lidam com mais de duas classes.

```
# Padronização dos dados
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255

# Treinamento do modelo
cnn_model.compile(optimizer="rmsprop",
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])
cnn_model.fit(train_images, train_classes, epochs=5,
 ↪  batch_size=64) # São usadas 5 épocas para treinamento

# Teste do modelo
test_loss, test_acc = cnn_model.evaluate(test_images,
 ↪  test_classes)
print("Acurácia do teste = {:.3f}".format(test_acc))
# Acurácia do teste = 0.991
```

Programa 3.3. Treinando e avaliando a CNN com o conjunto de dados de imagens MNIST.

humanos, no entanto, realizam tarefas completamente novas com apenas poucos exemplos ou conseguem aprender uma nova habilidade com poucas instruções caso possuam conhecimento prévio de uma habilidade similar. Por exemplo, uma pessoa A consegue identificar outra pessoa B em uma fotografia tendo observado poucas imagens da pessoa B previamente; ou, um aluno que sabe tocar violão precisará de pouca ajuda para aprender a tocar uma guitarra.

Para tentar simular o aprendizado rápido de novas tarefas baseado em conhecimentos prévios de outras tarefas distintas, modelos de *meta-learning* foram desenvolvidos para tentar mimetizar o aprendizado humano. Esses modelos são treinados utilizando não um conjunto de dados convencionais, como imagens e suas classes em um problema de classificação de imagens, mas sim tarefas inteiras treinadas em conjuntos de dados diferentes, como classificadores de imagens de carros, classificadores de motos e classificadores de bicicletas. Modelos treinados com base em um conjunto de tarefas poderão se adaptar a novas tarefas similares, como classificação de imagens de ônibus, uma vez que já possuem conhecimento prévio de tarefas similares. Por conta disso, modelos de *meta-learning* também são conhecidos como modelos capazes de "aprender a aprender".

A Figura 3.2 exemplifica como ocorre, de forma geral, o treinamento de um modelo convencional de classificação de imagens de dois tipos específicos de insetos. Nesse cenário, imagens das duas classes são fornecidas como entrada para a arquitetura e um modelo é gerado, podendo categorizar novas imagens que lhes forem fornecidas. A Figura 3.4 exemplifica como acontece o treinamento de um meta-modelo: em um primeiro momento, tarefas distintas de classificação de animais (Tarefas T_1, T_2 e T_3) são treinadas isoladamente (fase de aprendizado base, ou *base learning*, em inglês) e os resultados da soma das suas funções de perda são utilizadas como resultado da função de perda no

treinamento do meta-modelo (fase de meta-aprendizado, ou *meta-learning*, em inglês). Uma vez que o meta-modelo tenha sido treinado, ou seja, que tenha aprendido sobre o conhecimento gerado pelos modelos das tarefas anteriores, pode-se realizar o processo de *fine-tuning* em uma nova tarefa (T_4) para que o meta-modelo seja apto a classificar os novos tipos de animais da nova tarefa.

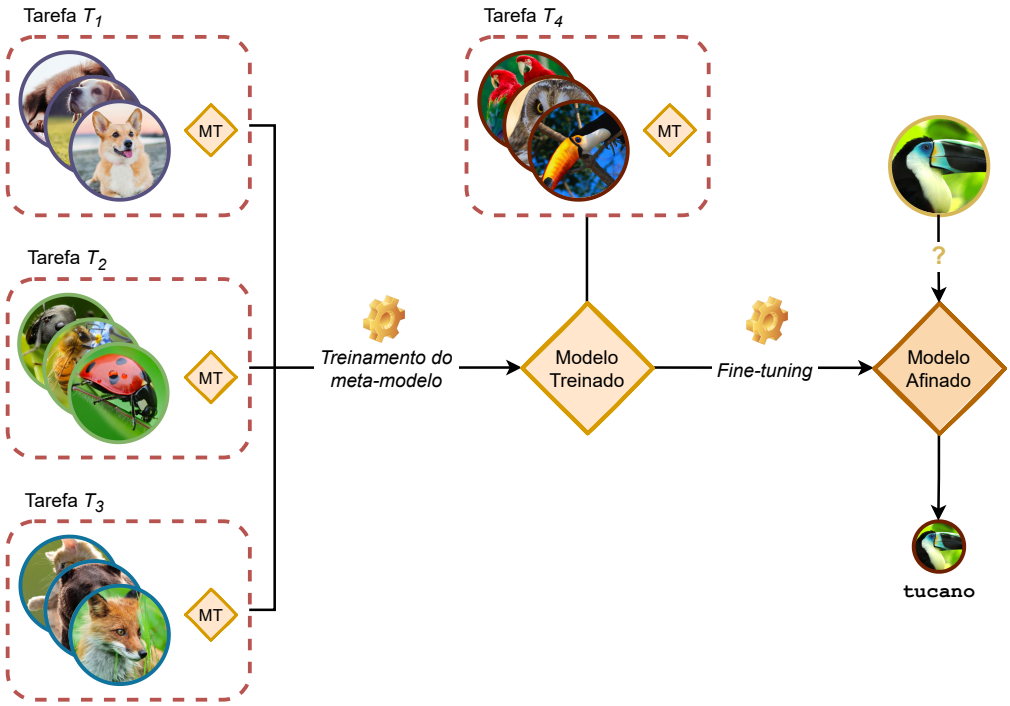


Figura 3.4. Treinamento de um meta-modelo utilizando diferentes tarefas e posterior *fine-tuning* com uma tarefa inédita. A sigla MT representa o modelo convencional treinado para cada tarefa individualmente. O esquema mostra que o meta-modelo foi treinado com diferentes tarefas de classificação de animais: raças de cães (Tarefa T_1), insetos (Tarefa T_2) e mamíferos no geral (Tarefa T_3). O meta-modelo, que até então possui um conhecimento prévio das Tarefas T_1, T_2 e T_3, é treinado novamente com uma Tarefa T_4 que classifica tipos de aves. Por fim, o modelo é testado com uma nova imagem que antes do *fine-tuning* só poderia ser predita pela Tarefa T_4.

3.2.2.1. Definição

O processo de *meta-learning* utiliza um conjunto de tarefas, extraído de um grupo maior de tarefas semelhantes entre si, e consiste em duas fases principais [Upadhyay et al. 2021]:

1. **Meta-treinamento.** Nesta fase, o treinamento ocorre em dois níveis distintos: (i) o aprendizado interno, que treinará meta-modelos para cada uma das tarefas utilizadas como entrada; e (ii) o aprendizado externo, que treinará o meta-modelo baseado nos valores das funções de perda das tarefas do nível de aprendizado interno. O mo-

delo final gerado na fase de meta-treinamento consegue generalizar o aprendizado realizado em todas as tarefas de entrada fornecidas.

2. **Meta-teste.** Também conhecida por fase de *fine-tuning*, a fase de meta-teste utiliza o modelo treinado na fase anterior como conhecimento prévio para adaptá-lo a uma nova tarefa não vista anteriormente.

3.2.2.2. Organização de Tarefas para *Meta-Learning*

Uma das formas mais comuns de modelar tarefas utilizadas como entrada para o treinamento de meta-modelos é através da divisão *N-way*, *K-shot*, que funciona da seguinte forma: cada tarefa deverá possuir N classes distintas e K exemplos para cada classe em seu conjunto de dados de treinamento (também chamado de conjunto de suporte ou *support set*). A Figura 3.5 exemplifica algumas divisões *N-way*, *K-shot* do conjunto de dados de treinamento para tarefas de classificação de imagens. Para o conjunto de dados de teste da tarefa (também chamado de conjunto de consulta ou *query set*), nenhuma restrição é estabelecida.

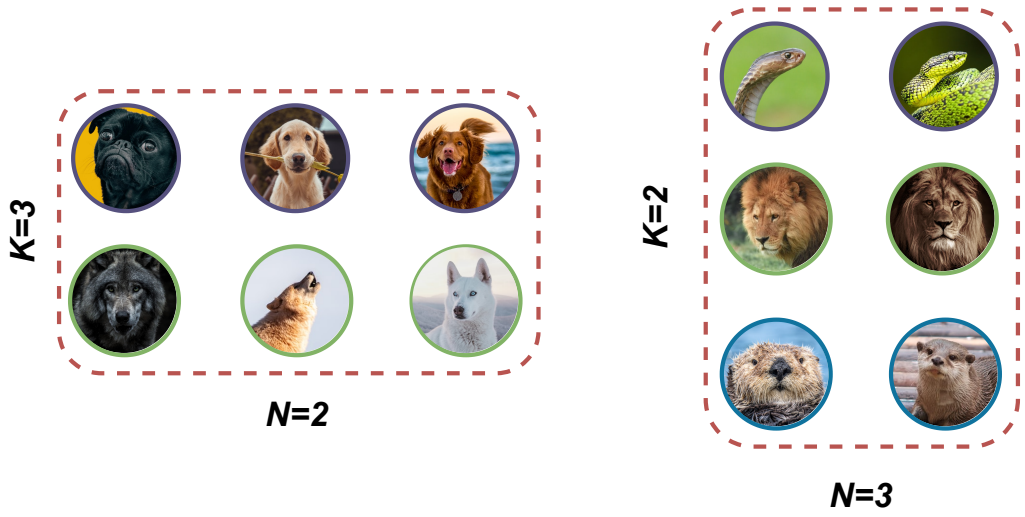


Figura 3.5. Diferentes exemplos de conjuntos de dados *N-way*, *K-shot* para modelagem de tarefas de meta-modelos.

Para o treinamento e teste de um meta-modelo, ou seja, um meta-treinamento e um meta-teste, respectivamente, exemplos de tarefas (também chamadas de episódios) são usadas com uma configuração *N-way*, *K-shot*. A Figura 3.6 ilustra como ocorre a divisão de tarefas no treinamento de um meta-modelo.

Em domínios mais específicos, a quantidade de dados disponível não é suficiente para o treinamento de modelos de Aprendizado Profundo com *meta-learning*. Em geral, esses conjuntos de dados possuem um subconjunto rotulado manualmente por algum especialista de domínio. Ou seja, ao mapear esse cenário para uma tarefa com configuração *N-way*, *K-shot*, têm-se valores de N e K muito pequenos. Tarefas desse tipo são chamadas de problemas de *few-shot learning* (ou aprendizagem com poucos exemplos, em tradução livre).

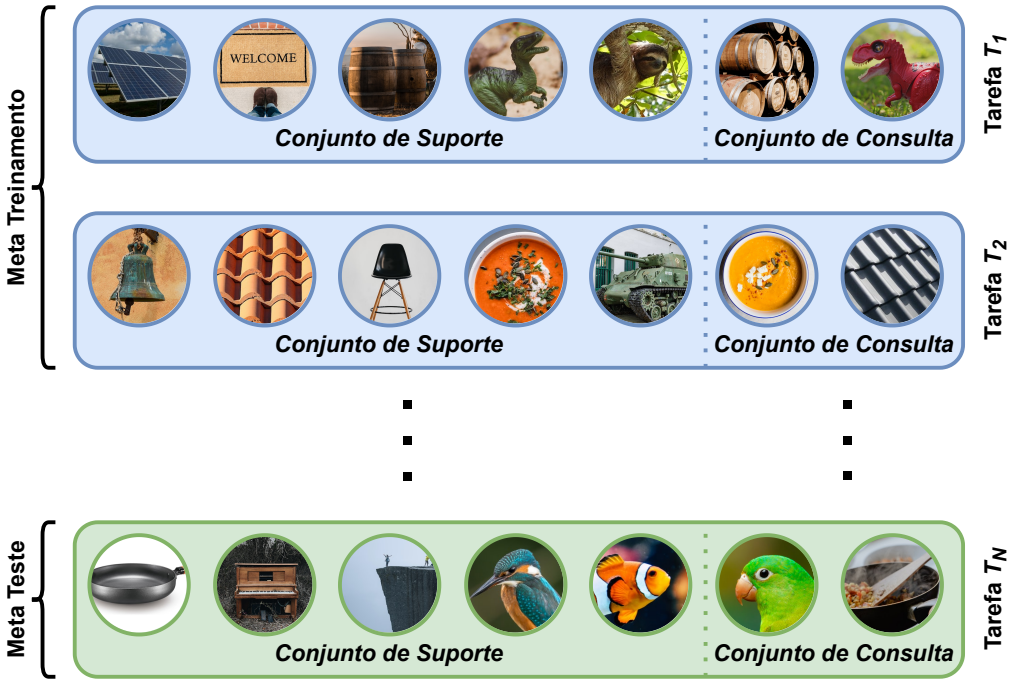


Figura 3.6. Tarefa de classificação *N-way*, *K-shot*, onde $N = 5$ e $K = 1$. Adaptado de [Huisman et al. 2021].

3.3. Técnicas de *Meta-Learning* Aplicadas a Aprendizado Profundo

Uma vez que a estrutura de tarefas é definida, existem algumas técnicas que podem ser utilizadas para o treinamento do meta-modelo. Essas diferentes abordagens variam com base em como o meta-modelo se adapta perante às tarefas do treinamento. Essas abordagens podem ser classificadas em três categorias, baseadas em:

1. **Métricas.** A abordagem baseada em métricas geralmente utiliza técnicas não paramétricas (por exemplo, *k*-vizinhos mais próximos) para generalizar as tarefas utilizadas para treinamento do meta-modelo.

2. **Modelos.** Também conhecida como abordagem "caixa preta", seus métodos treinam uma rede neural inteira, com alguns dados de treinamento no conjunto de suporte e um conjunto inicial de meta-parâmetros. Depois, guardam uma representação interna e fazem as predições no conjunto de consultas.

3. **Otimização.** A abordagem baseada em otimização trata a parte de adaptação do processo como um problema de otimização e utiliza algoritmos tradicionais para a atualização dos parâmetros do meta-modelo, como o gradiente descendente.

Nesta seção, cada uma das abordagens descritas brevemente acima é apresentada em detalhe, como também alguns dos algoritmos mais utilizados.

3.3.1. Abordagem Baseada em Métricas

A primeira abordagem baseada em métricas visa encontrar um **espaço vetorial** que possa representar bem as características dos dados de diversas tarefas distintas. Assim, novas tarefas que possuem uma pequena quantidade de exemplos podem ser aprendidas apenas comparando o novo conjunto de dados com as representações vetoriais previamente obtidas, das quais se sabem as classes. Quanto maior for a semelhança entre um novo dado representado e uma representação presente no espaço vetorial, maior é a chance de que o dado tenha a mesma classe que a da representação.

Formalmente, a ideia é encontrar o meta-conhecimento ω na forma de um modelo com parâmetros θ (a qual será referida como $f_\theta : \mathrm{R}^N \rightarrow \mathrm{R}^M$). Dessa forma, é possível criar um *kernel de similaridade*, permitindo que duas representações vetoriais $\mathbf{x}_1$ e $\mathbf{x}_2$ possam ser comparadas, onde representações mais similares pertencem a um mesmo grupo. Logo, a predição de novos dados dá-se comparando novas representações $\mathbf{x}$ com representações vetoriais $\mathbf{x}_i$ já conhecidas.

Na Figura 3.7, as retas representam $f_\theta(\mathbf{x})$, onde $\mathbf{x}$ é uma representação. Os dados em vermelho representam dados cujos rótulos são conhecidos (conjunto de suporte), denotado por S, enquanto o dado em azul representa um novo dado. A partir dessa projeção, é possível identificar a qual elemento de suporte o novo dado mais se assemelha, permitindo então sua classificação.

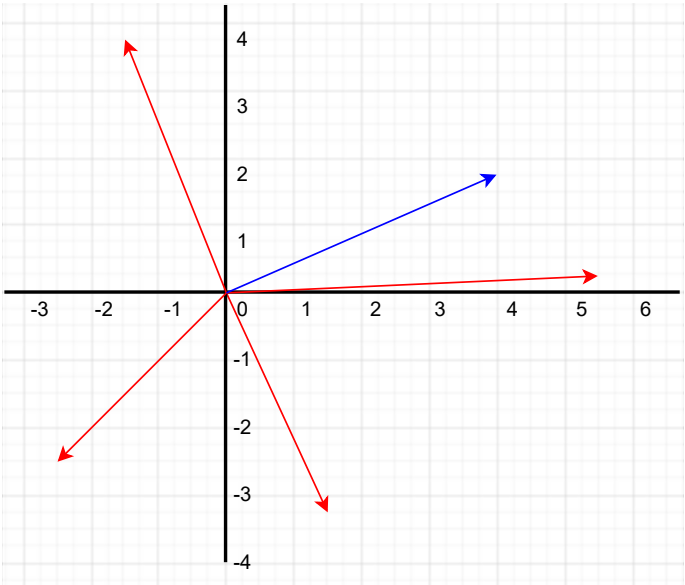


Figura 3.7. Ilustração dos métodos baseados em métricas. O vetor azul indica a representação vetorial de um novo dado, enquanto os vermelhos, dados conhecidos.

Nos métodos baseados em métricas, o aprendizado é tido como **não paramétrico**: o modelo f_θ não sofre nenhum tipo de modificação quando lida com novas tarefas, pois ele é o meta-conhecimento aprendido pelo método. Os dados apenas são projetados e comparados. Dada uma tarefa $T = (\mathrm{D}_T^{tr}, \mathrm{D}_T^{te})$, uma nova representação vetorial $\mathbf{x} \in \mathrm{D}_T^{te}$ e uma medida de similaridade $s_\theta = s(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_j))$, a distribuição probabilística de $\mathbf{x}$ sobre as classes Y existentes pode ser encontrada pela Equação 1, onde y_i é um vetor *one-*

hot (vetor de zeros com o valor um na posição da respectiva classe). A partir do resultado da equação, a classe predita será aquela com maior valor.

$$p_\theta(Y|\mathbf{x}, \mathrm{D}^{tr}) = \sum_{(\mathbf{x}_i, y_i) \in \mathrm{D}^{tr}} s_\theta(\mathbf{x}, \mathbf{x}_i) y_i \tag{1}$$

As principais vantagens do uso de técnicas baseadas em métricas são que (i) a ideia de realizar predições baseadas em similaridade é conceitualmente simples e (ii) elas podem ser rápidas no teste quando as tarefas são pequenas, pois as redes não precisam fazer ajustes específicos na arquitetura [Huisman et al. 2021]. No entanto, quando as tarefas de meta-teste tornam-se mais distantes das tarefas usadas no meta-treinamento, essas técnicas são incapazes de entender novas informações específicas das tarefas. Além disso, conforme as tarefas aumentam, as comparações de pares podem se tornar computacionalmente mais caras. Por fim, a maioria das técnicas baseadas em métricas depende da presença de dados anotados, o que as tornam inaplicáveis para configurações fora do aprendizado supervisionado.

Existem diversos métodos baseados em métricas: redes siamesas [Koch et al. 2015], *matching networks* [Vinyals et al. 2016], *Graph Neural Network* [Garcia and Bruna 2017], dentre outras. Este trabalho focará apenas na *prototypical networks* [Snell et al. 2017].

3.3.1.1. *Prototypical Networks*

O método baseia-se na ideia de que é mais fácil realizar classificações quando as classes estão representadas no mesmo espaço vetorial dos dados. Dado que f_θ realiza uma projeção M-dimensional, o objetivo é encontrar um ponto $\mathbf{c}_k$ (conhecido como protótipo ou *prototype*) para representar cada classe. Uma forma intuitiva de representar o protótipo $\mathbf{c}_k$ é encontrando o centro geométrico dos elementos que pertencem àquela classe, conforme representado na Figura 3.8. Comumente, os modelos f_θ são redes neurais, por isso o nome *prototypical networks*.

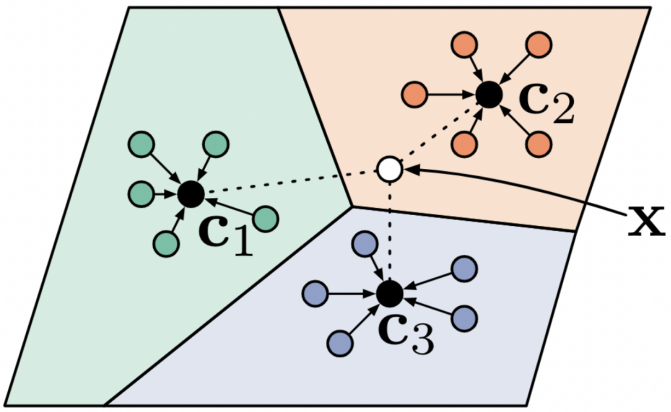


Figura 3.8. Exemplo da técnica *prototypical networks*. As superfícies coloridas representam os dados de suporte projetados no espaço R^M utilizando f_θ. Os pontos pretos representam os protótipos de cada classe. Um novo dado x é classificado conforme o protótipo mais próximo. Fonte: [Snell et al. 2017].

Os protótipos podem ser calculados conforme a Equação 2. Para cada representação no conjunto de suporte de cada classe, o protótipo é caracterizado pelo seu centroide e S_k representa o subconjunto dos exemplos do conjunto de suporte S associados à classe k.

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\theta(\mathbf{x}_i) \tag{2}$$

A implementação do cálculo dos protótipos encontra-se no Programa 3.4. Inicialmente, as representações são projetadas no espaço vetorial utilizando o meta-modelo f_θ. Em seguida, é criada uma matriz para representar todos os centroides, inicializadas com zeros (0's). Para cada classe, são selecionadas as representações projetadas que pertencem aquela classe, sendo tirada sua média. A matriz dos centroides é, por fim, retornada.

```
def compute_prototypes(model,X,y):
    projection = model.predict(X, verbose=0)
    centroids = np.zeros((y.shape[1], projection.shape[1]))
    for i in range(y.shape[1]):
        # Obtém a representação para a classe i
        idx = y.argmax(axis=1) == i
        # Computa a média
        centroids[i,:] = projection[idx].mean(axis=0)

    return centroids
```

Programa 3.4. Cálculo dos protótipos de um conjunto de representações X e classes y, utilizando um modelo para projetar os dados.

A partir dos protótipos encontrados e de uma medida de distância $d : \mathrm{R}^N \times \mathrm{R}^M \rightarrow [0, +\infty)$, a probabilidade de um novo ponto $\mathbf{x}$ pertencer a cada classe conhecida dá-se pela aplicação da função exponencial normalizada (ou *softmax*) sobre o negativo da distância de $\mathbf{x}$ para o protótipo $\mathbf{c}_k$ de cada classe, conforme apresentado na Equação 3. Consequentemente, a classe predita será a de maior probabilidade.

$$p_\theta(y = k|\mathbf{x}) = \frac{\exp(-d(f_\theta(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\theta(\mathbf{x}), \mathbf{c}_{k'}))} \tag{3}$$

O Programa 3.5 apresenta o cálculo da probabilidade presente na Equação 3. Nele, é possível informar um conjunto inteiro de representações a serem classificadas. Isso é possível, pois a biblioteca *NumPy* permite a organização de matrizes para realizar múltiplas operações simultaneamente. Nesse caso, a matriz dos protótipos é $K \times M$, enquanto a matriz das projeções dos dados de consulta é $Nq \times M$, onde Nq é a quantidade de representações. Os *reshapes* realizados no código permitem que cada uma das Nq representações seja operada com cada um dos K protótipos. Ao tirar a norma, obtém-se um resultado de $Nq \times K \times 1$, que indica a distância euclidiana entre cada representação e protótipo. Por fim, é possível aplicar o *softmax* para cada linha da matriz negativa das distâncias

(transformando ela em uma medida de similaridade), resultando na probabilidade das representações.

```
def predict_proba(model, prototypes, query):
    projections = model(query)

    projections = tf.reshape(projections,
     ↪  (projections.shape[0],1,projections.shape[1],1))
    prototypes =
     ↪  prototypes.reshape(1,prototypes.shape[0],prototypes.shape[1],1)

    distances = tf.norm(projections - prototypes, axis=2, ord=np.inf)
    distances = tf.reshape(distances, (len(query), len(prototypes)))

    probabilities = keras.activations.softmax(-distances, axis=1)
    return probabilities
```

Programa 3.5. Cálculo das probabilidades de cada elemento do conjunto de consulta pertencer a cada classe a partir de seus protótipos e um modelo para projetar os dados.

O processo do aprendizado do meta-modelo dá-se por minimizar a log-probabilidade negativa $J(\theta) = -\log p_\theta(y = k|\mathbf{x})$ da classe k esperada, utilizando o gradiente descendente estocástico (ou algum outro método de gradiente). O Programa 3.6 demonstra uma implementação desse aprendizado.

```
def task_episode(model,
             support, support_labels,
             query, query_labels, optimizer):

    prototypes = compute_prototypes(model, support, support_label)
    loss = None
    with tf.GradientTape() as tape:
        probabilities = predict_proba(model, prototypes, query)
        right_probabilities =
         ↪  tf.reduce_sum(probabilities*query_labels, axis=1)
        neg_log_right_prob = -tf.math.log(right_probabilities)
        loss = tf.reduce_mean(neg_log_right_prob)

    gradient = tape.gradient(loss,model.trainable_variables)
    optimizer.apply_gradients(
        zip(gradient, model.trainable_variables))
```

Programa 3.6. Implementação do processo de aprendizado de uma tarefa utilizando *prototypical networks*.

3.3.2. Abordagem Baseada em Modelos

Enquanto a abordagem baseada em métricas visa encontrar um espaço vetorial que permita comparar bem as características dos dados, a abordagem baseada em modelos tenta fazer com que, durante o meta-treinamento, as tarefas mudem de alguma forma a **estrutura do modelo** manipulado internamente. Como o meta-modelo em questão possui um estado interno e uma representação da tarefa, quando apresentado a uma nova tarefa, ele deve conseguir capturar informações específicas das tarefas consideradas relevantes para usá-las em seguida.

Vale ressaltar que, como o próprio meta-modelo realiza operações em seu estado interno, eles são considerados "caixa-preta", reduzindo sua explicabilidade para os resultados obtidos. Além disso, como o meta-modelo deve reter informações de tarefas específicas e passadas, é necessário utilizar um mecanismo de memória, que pode ser interno ou externo.

As vantagens de usar a abordagem baseada em modelos inclui a flexibilidade da dinâmica interna dos sistemas e sua aplicabilidade mais ampla em comparação com a maioria das técnicas baseadas em métricas. No entanto, métodos baseados em modelo são frequentemente superados pelos baseados em métricas, podem não ter um bom desempenho quando apresentados a conjuntos de dados maiores, e generalizam menos para tarefas mais discrepantes entre si do que técnicas baseadas em otimização [Huisman et al. 2021].

Existem diversos algoritmos de *meta-learning* baseados em modelos, como o *Simple Neural Attentive Learner* (SNAIL) [Mishra et al. 2017], que propõe uma arquitetura de rede neural para agir como mecanismo de memória e a *Conditional Neural Processes* (CNP) [Garnelo et al. 2018] que não necessita de um módulo de memória. Esta seção irá focar na *Meta-Networks* [Munkhdalai and Yu 2017] e na abstração utilizada para seu funcionamento.

3.3.2.1. *Meta-Networks*

O método consiste em utilizar redes neurais tanto para aprender uma nova tarefa quanto para generalizar seus conceitos para novas tarefas. *Meta-Networks* são formadas por duas redes neurais, como mostra a Figura 3.9: uma atuando no espaço vetorial proveniente das tarefas (generalizando um novo conceito) e a outra no espaço da tarefa (atividade para extração de *features* dos dados da tarefa). Essas redes possuem o nome de *meta-learner* e *base-learner*, respectivamente.

O fluxo de aprendizagem inicia-se no *base-learner*, o qual executará seu aprendizado sobre a tarefa, enviando um *feedback* para o *meta-learner* na forma de um relatório, contendo meta-informações sobre o aprendizado para as tarefas executadas nele.

Observe que o *feedback* enviado do *base-learner* para o *meta-learner* tem o intuito de mostrar o quão bem o *base-learner* está aprendendo sobre a tarefa. Ou seja, poderá ser passado o valor da função de perda como meta-informação para quantificar isso, assim como métricas mais adequadas para o domínio da aplicação.

Em seguida, o *meta-learner*, de posse do *feedback*, deverá aprender um mapea-

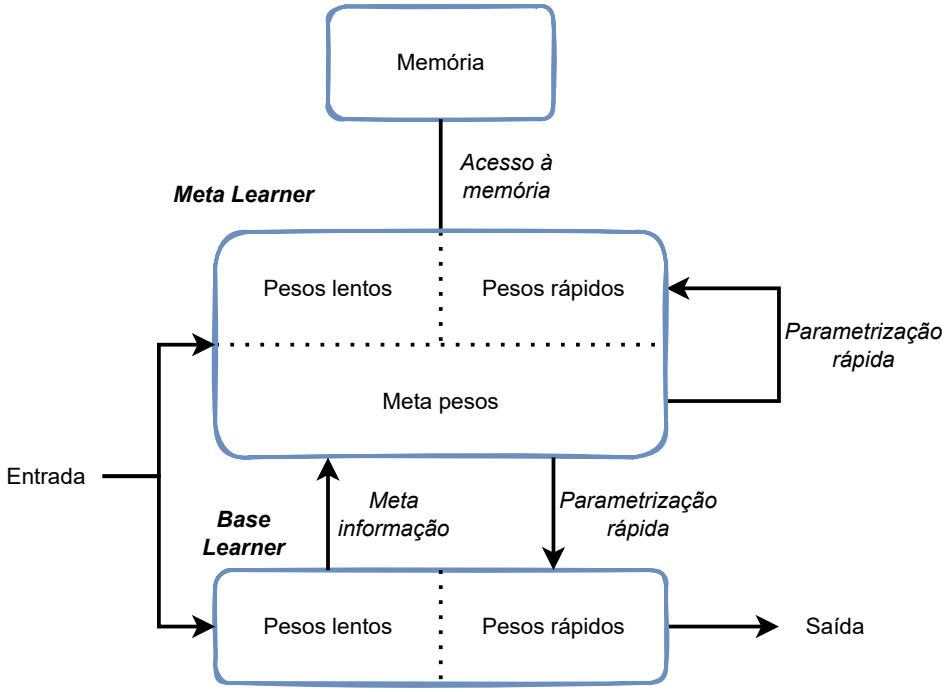


Figura 3.9. Arquitetura de uma *Meta-Network*.

mento desta informação à sua camada de pesos rápidos. Por fim, uma vez que sua camada de pesos provenientes da matriz de memória é atualizada, ele gerará uma representação de vetores indexada por cada tarefa vista para, então, realizar o treinamento sobre ela.

Como ele está operando no espaço vetorial, o *meta-learner* suporta aprendizado contínuo sobre tarefas agnósticas, atualizando seus próprios parâmetros e os do *base-learner* utilizado[6].

As seções a seguir apresentam mais detalhes sobre os componentes descritos anteriormente.

3.3.2.2. *Meta-Learner*

Este componente é composto por uma função de representação vetorial e duas funções de geração rápida de pesos. Considere m e d como as funções de geração rápida de pesos e u a função de representação vetorial. A função m é responsável por aprender o mapeamento do *feedback* fornecido pelo *base-learner* à camada de pesos rápidos.

Além disso, a função m deverá armazenar os pesos obtidos na matriz de memória M, indexada pela representação vetorial de tarefas fornecida pela função u. A função de aprendizagem u é parametrizada pela camada de pesos lentos em conjunto com a camada de pesos rápidos. Desta forma, ela consegue capturar as funções de perda de cada tarefa passada.

[6]Para mais detalhes sobre a implementação de uma *Meta-Network*, ver o repositório do seu criador [Munkhdalai and Yu 2017]

Finalmente, após os gradientes serem atualizados na matriz de memória M, é realizado uma espécie de "sorteio" de T tarefas para compor o treino da função de aprendizado d. Essa etapa é muito importante, pois, como o treino ocorre sequencialmente, a ordem das tarefas passadas pode influenciar nos resultados obtidos.

A rede representada por d precisa aceitar um tamanho de entrada variável. Para tal, uma MLP pode ser usada para sumarizar os gradientes, entretanto, os experimentos realizados por [Munkhdalai and Yu 2017] demonstraram uma fraca convergência da *Meta-Network* quando configurada desta forma, sendo preferível utilizar uma rede *Long Short-Term Memory* (LSTM) [Hochreiter and Schmidhuber 1997] para esta atividade.

3.3.2.3. *Base-Learner*

Diferentemente do *meta-learner*, o *base-learner*, denotado por b, possui uma única função de aprendizado, cujo objetivo consiste em aprender e extrair atributos da tarefa. Entretanto, ao contrário de redes neurais convencionais, essa é parametrizada tanto pela camada de gradientes rápidos quanto pela camada de gradientes lentos.

O interessante desse componente de aprendizado é que seus parâmetros são atualizados em momentos diferentes. Os gradientes lentos são atualizados durante o treinamento de b, enquanto os gradientes rápidos são atualizados pelo *meta-learner* a cada entrada da rede. A Figura 3.9 ilustra, de uma perspectiva mais geral, como ocorre essa troca de informações entre os componentes da *Meta-Network*.

Segundo [Munkhdalai and Yu 2017], é possível instanciar um *base-learner* mais simples, com apenas uma das camadas de gradiente, seja ela a rápida ou lenta, porém, durante seus experimentos, concluiu-se que são necessárias as duas camadas de gradientes para a *Meta-Network* convergir devidamente.

A Figura 3.10 mostra como uma MLP pode ser instanciada para ser utilizada em uma *Meta-Network*. Uma CNN modificada para atender os dois gradientes também pode ser utilizada e, segundo [Munkhdalai and Yu 2017], a CNN apresentou melhores resultados que a MLP proposta inicialmente para uso em *Meta-Networks*.

3.3.3. Abordagem Baseada em Otimização

Modelos de Aprendizado Profundo convencionais atualizam seus parâmetros utilizando o algoritmo de retro-propagação de seus gradientes. Entretanto, em muitas aplicações, são necessárias grandes quantidades de dados para esse treinamento. As técnicas de *meta-learning* baseadas em otimização focam em **otimizar meta-parâmetros** para que novas tarefas sejam aprendidas rapidamente, mas com uma pequena quantidade de dados.

A maioria das técnicas utiliza otimizações de dois níveis: a primeira otimização, também chamada de otimização interna, otimiza os parâmetros das tarefas (utilizando o gradiente descendente, por exemplo) que serão utilizadas para otimizar um meta-modelo, como em um modelo convencional; já a segunda otimização, também chamada de otimização externa, utiliza o resultado das funções de custo das tarefas de entrada e generaliza o aprendizado das tarefas utilizadas no otimizador interno.

Dentre as diversas técnicas de *meta-learning* disponíveis baseadas em otimização

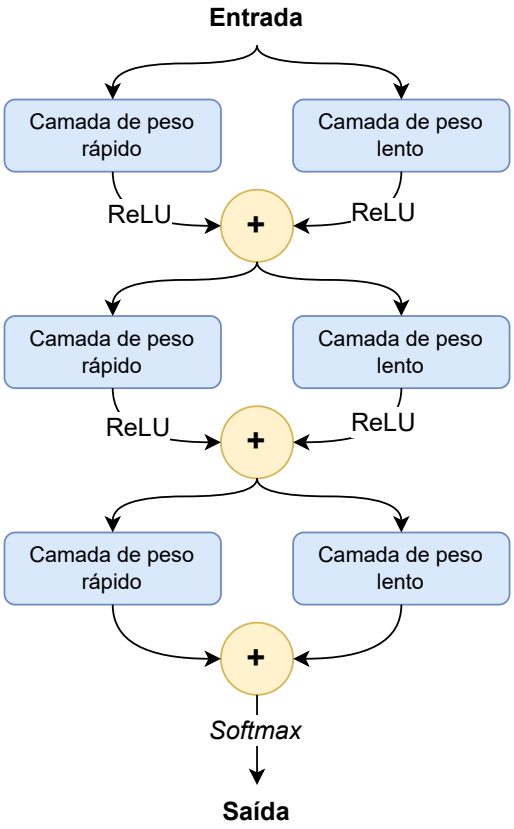


Figura 3.10. MLP utilizada por [Munkhdalai and Yu 2017] em seus experimentos com *base-learner*.

de parâmetros, como meta-aprendizado utilizando LSTMs [Ravi and Larochelle 2017], Meta-SGD [Li et al. 2017] e *Reptile* [Nichol and Schulman 2018], esta seção focará no algoritmo MAML[7] [Finn et al. 2017] por sua simplicidade, alto desempenho e facilidade de aplicação.

O algoritmo MAML utiliza procedimentos de otimização simples, como o gradiente descendente estocástico, tanto em sua fase de aprendizado interno (aprendendo tarefas individualmente), quanto em sua fase de aprendizado externo (aprendendo seu meta-modelo). A ideia principal do MAML é aprender bons parâmetros na sua fase de treinamento que ofereçam uma rápida adaptação para novas tarefas. A Figura 3.11 exemplifica como o algoritmo MAML funciona: o parâmetro θ foi otimizado baseado na otimização individual das tarefas que geraram gradientes $\nabla\mathcal{L}_1$, $\nabla\mathcal{L}_2$ e $\nabla\mathcal{L}_3$. À vista disso, novas tarefas adaptaram facilmente seus parâmetros θ_1^*, θ_2^* e θ_3^* a partir de θ.

A Figura 3.12 apresenta um fluxograma para o algoritmo MAML, que recebe como entrada um conjunto de tarefas e a quantidade de passos de treinamento. Inicialmente, é criado um meta-modelo com pesos inicializados aleatoriamente, representado no fluxograma por θ. Em seguida, uma cópia desse modelo θ para cada uma das tarefas $(\theta_1, \theta_2, \cdots, \theta_N)$ do conjunto de treinamento é criada. Cada uma das tarefas irá treinar seu modelo utilizando como dados de entrada o seu próprio conjunto de suporte e a quanti-

[7] *Model Agnostic Meta-Learning*, ou, em português, Meta-Aprendizado Agnóstico à Modelos.

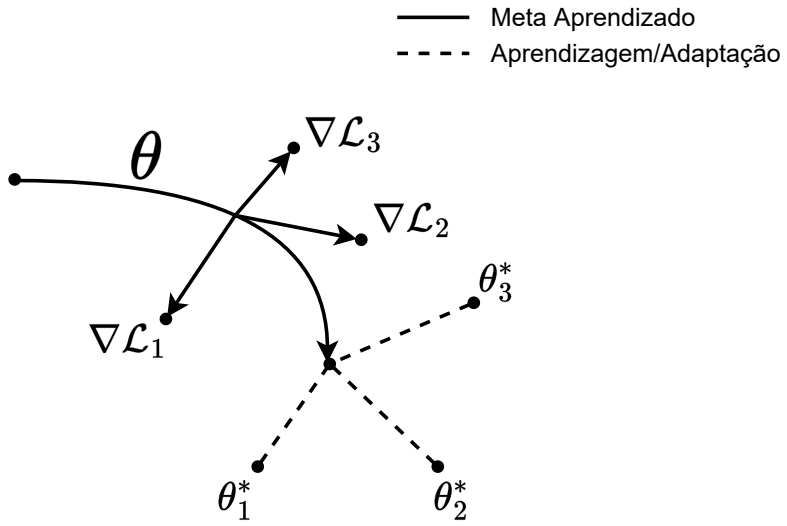


Figura 3.11. Intuição sobre como o treinamento de meta-modelos baseados em otimização funcionam. Adaptado de [Finn et al. 2017].

dade de passos de treinamento informada inicialmente, gerando novos pesos atualizados $(\theta_1', \theta_2', \cdots, \theta_N')$.

As tarefas não possuem nenhuma relação entre si em suas fases de treinamento, assemelhando-se a treinamentos de modelos convencionais. Uma vez que os novos pesos foram computados, a função de perda é calculada para cada tarefa utilizando seus respectivos conjuntos de consulta. O meta-modelo utilizará a soma de todas as funções de perda de todas as tarefas treinadas como sua própria função de perda, indicando o quão útil os pesos iniciais no começo do treinamento foram para cada tarefa individualmente. Com a meta-função de perda calculada, os pesos do meta-modelo θ são atualizados e o fluxograma pode ser repetido quantas vezes forem necessárias.

Dada a sua importância dentre as técnicas de meta-aprendizagem baseadas em otimização, o algoritmo MAML serviu como inspiração para muitos outros trabalhos, como LLAMA [Grant et al. 2018], PLATIPUS [Finn et al. 2018] e BMAML [Yoon et al. 2018]. Entretanto, o MAML possui suas desvantagens: o algoritmo tem custos muito elevados com relação ao tempo de treinamento e consumo de memória, uma vez que precisa computar derivadas de alta-ordem para várias tarefas simultaneamente.

3.4. Uso de Meta-Learning em Classificação de Imagens

Esta seção possui uma aplicação prática de todos os conceitos abordados até aqui no capítulo. A Figura 3.13 apresenta um fluxograma do processo de aplicação de uma das técnicas de *meta-learning* abordadas na Seção 3.3: o algoritmo MAML, considerando desde a criação de tarefas do tipo *N-way*, *K-shot* a partir de um conjunto de dados de imagens, até o *fine-tuning* do meta-modelo treinado para uma nova tarefa.

Descrição do Problema. Alguns modelos mais recentes para classificação de imagens, como VGG-16 e *ResNet*50, treinados com centenas de milhares de exemplos, conseguem resolver bem diversas tarefas de classificação de imagens mais genéricas, como distinguir

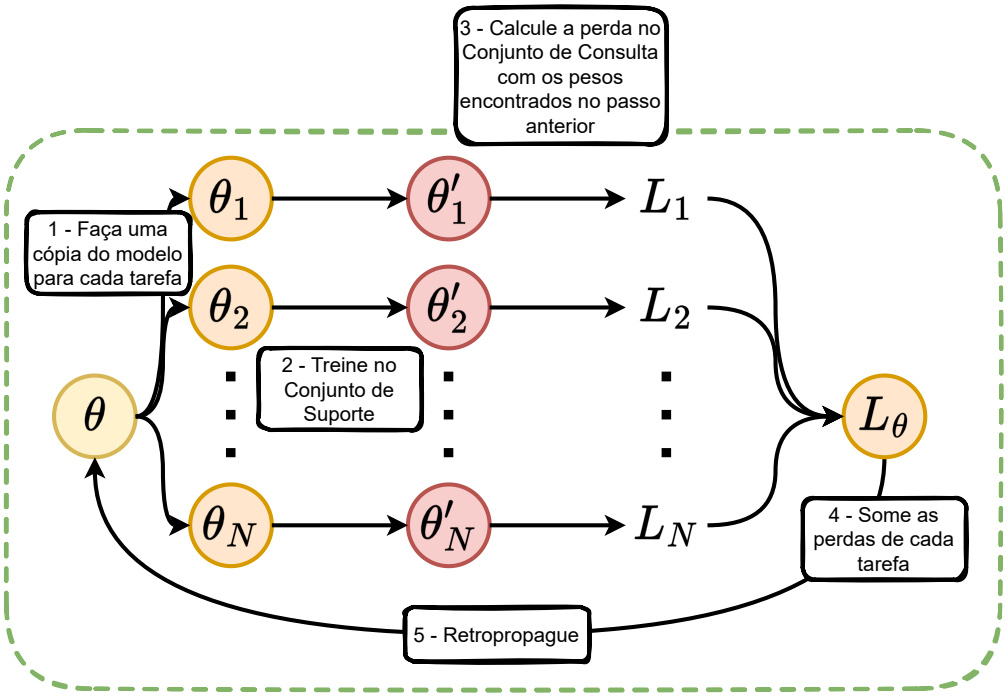


Figura 3.12. Diagrama que representa o algoritmo MAML.

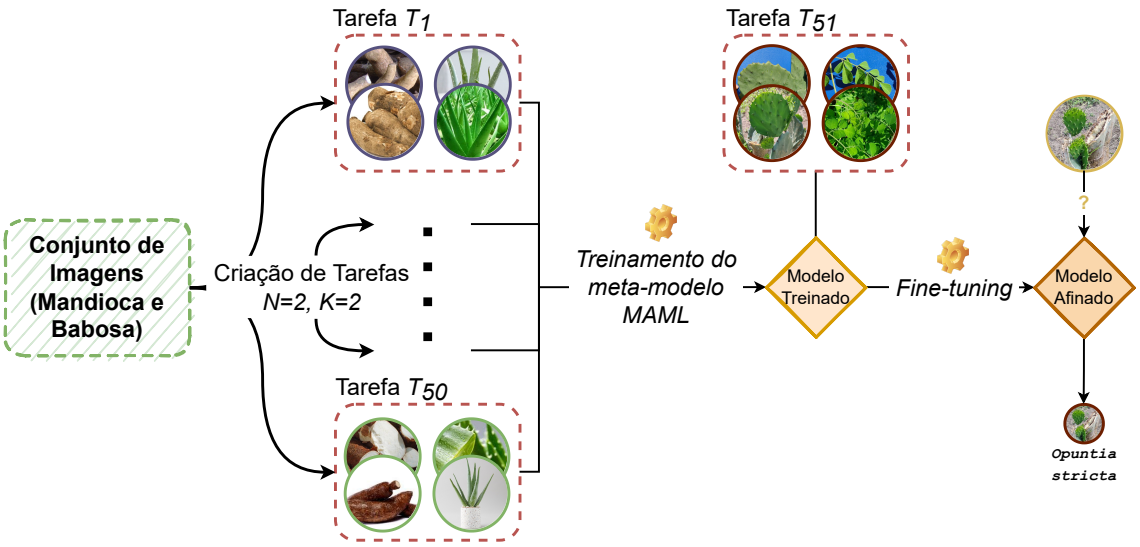


Figura 3.13. *Pipeline* completo para o treinamento do modelo e *fine-tuning* de um meta-modelo classificador de plantas forrageiras.

gatos de cachorros, ou leões de cavalos. Entretanto, para domínios específicos, esses modelos podem não fornecer bons resultados.

No contexto da Agricultura de Precisão, classificar imagens de diferentes tipos de plantas forrageiras é uma tarefa importante, pois esses resultados podem auxiliar agricultores na tomada de decisões com relação aos seus cultivos. Infelizmente, não existem opções de conjuntos de dados com quantidades de exemplos relevantes para o treinamento

de, por exemplo, CNNs para classificação de diferentes tipos de forrageiras.

Conjuntos de Dados Utilizados. Para resolver essa tarefa, um pequeno conjunto de dados de plantas forrageiras foi criado, contendo seis classes e aproximadamente quinze imagens por classe, e anotado manualmente. Também foi utilizado um subconjunto do conjunto de dados *Plants Type Datasets*[8] contendo 862 imagens do tubérculo mandioca e 1.000 imagens da planta babosa.

Para treinar o meta-modelo, foram criadas cinquenta tarefas do tipo 2-*way*, 2-*shot* a partir do conjunto de dados de mandiocas e babosas. Também foi criada uma tarefa do tipo 2-*way*, 2-*shot* com duas classes diferentes de plantas forrageiras para o processo de *fine-tuning*.

Treinamento do Meta-Modelo e *Fine-Tuning*. O Programa 3.7 apresenta o código da criação do meta-modelo que utiliza o algoritmo MAML e as definições de tarefas aplicadas nele. Para a criação do meta-modelo, o modelo base foi uma rede `resnet18` e o otimizador `Adam`. O código apresentado foi escrito utilizando a linguagem *Python* e o *framework Flash*[9]. O Programa 3.8 mostra como instanciar e executar o componente que fará o treinamento do meta-modelo com as tarefas de classificação de mandioca e babosa com o auxílio de uma GPU (caso disponível) e como é feito o processo de *fine-tuning* utilizando a nova tarefa de classificação de forrageiras.

```
model = ImageClassifier(
    backbone="resnet18",

    # Caso queira utilizar a estrategia prototypical networks, use
     ↪  a chave 'prototypicalnetworks' abaixo
    training_strategy="maml",

    pretrained=False,

    # Definição das tarefas que serão utilizadas neste modelo
    training_strategy_kwargs={"epoch_length": 50,
     ↪  "meta_batch_size": 2, "num_tasks": 50, "test_num_tasks":
     ↪  50, "ways": datamodule_fruits.num_classes, "shots": 2,
     ↪  "test_ways": 2, "test_shots": 1},

    optimizer=torch.optim.Adam,
    learning_rate=0.001,
)
```

Programa 3.7. Criação do meta-modelo que utiliza o algoritmo MAML para tarefas de classificação de plantas (mandioca e babosa).

Resultados. A Tabela 3.1 mostra a comparação de dois modelos de classificação de plantas forrageiras a partir de métricas comuns para este tipo de tarefa. Para o modelo con-

[8] https://www.kaggle.com/datasets/yudhaislamisulistya/plants-type-datasets

[9] https://lightning-flash.readthedocs.io/

```
trainer = flash.Trainer(
    max_epochs=50,
    precision=16,
    accelerator="ddp_shared",
    gpus=int(torch.cuda.is_available()),
    logger=logger,
)

trainer.fit(model, datamodule=datamodule_fruits)

trainer.finetune(model, datamodule=datamodule_forrageiras,
 ↪  strategy="no_freeze")
```

Programa 3.8. Treinamento do meta-modelo com os dados de mandioca e babosa e *fine-tuning* da tarefa de classificação de forrageiras.

vencional (*RestNet*-18), a tarefa de classificação de forrageiras foi treinada considerando uma pequena quantidade de dados. O meta-modelo que usa o MAML foi primeiramente treinado com o conjunto de dados de babosas e mandiocas, e um processo de *fine-tuning* foi aplicado para a tarefa de classificação de forrageiras.

Modelo	**Acurácia**	**Precisão**	**Revocação**	**F_1-*score***
ResNet-18 + MAML	0.88	0.86	0.91	0.86
ResNet-18	0.59	0.49	0.58	0.51

Tabela 3.1. Resultados dos dados de teste para o modelo convencional *ResNet*-18 e para o modelo gerado após o *fine-tuning* do meta-modelo criado com o algoritmo MAML e a arquitetura *ResNet* como base (*ResNet*-18 + MAML).

Conclui-se que, para uma baixa quantidade de dados, o meta-modelo *ResNet*-18 + MAML em comparação com o *ResNet*-18 convencional generaliza bem o aprendizado para tarefas de classificação de babosas e mandiocas, obtendo bons resultados para a tarefa de classificação de forrageiras após o processo de *fine-tuning*.

3.5. Limitações, Aplicações e Desafios

A presente seção aborda as limitações dos métodos de *meta-learning*, apresenta algumas aplicações em diferentes áreas e cobre desafios que acompanham esta técnica em ascensão. Além disso, são apresentadas algumas das áreas de pesquisa que merecem serem exploradas por esses métodos.

3.5.1. Limitações e Desafios

Muitas aplicações de *meta-learning* apresentam bons resultados quando utilizam uma distribuição de tarefas com alto nível de similaridade como entrada. Quando utilizadas distribuições com tarefas distintas, o conflito de gradientes entre as tarefas pode ocasionar um baixo grau de generalização por parte do meta-modelo [Yu et al. 2020].

Outro desafio que as técnicas de *meta-learning* enfrentam são as potenciais baixas

quantidades de tarefas disponíveis para que o meta-modelo generalize bem para uma nova tarefa. Além disso, os meta-modelos podem não generalizar bem para tarefas que estão fora da distribuição utilizada como treinamento.

Um último desafio a salientar é o custo computacional e de tempo do treinamento dos meta-modelos por conta dos treinamentos aninhados tanto para tarefas quanto para meta-modelo e, a depender da abordagem utilizada, cálculos matemáticos diferenciais envolvidos.

3.5.2. Aplicações

Cenários de tarefas com pouca quantidade de dados de treinamento são extremamente desafiadores, principalmente para redes neurais profundas, onde o conjunto de dados é fator determinante no desempenho de um modelo. Em geral, quando tais modelos são treinados com conjuntos de dados pequenos, obtém-se um *overfitting* do modelo ou sua não-convergência. Abordagens baseadas em *meta-learning* são úteis em diversas áreas, pois todas podem sofrer com os mesmos problemas de disponibilidade de dados.

Visão Computacional. As aplicações mais comuns nessa área envolvem tarefas como reconhecimento de imagens [Snell et al. 2017], detecção de objetos [Kang et al. 2019], segmentação de objetos [Dong and Xing 2018], e geração de imagens [Zakharov et al. 2019] e vídeos [Wang et al. 2019]. Infelizmente, os resultados obtidos com técnicas de *meta-learning* aplicadas à Visão Computacional ainda não se equiparam ao treinamento de modelos de Aprendizado Profundo com uma grande quantidade de dados, mas mostram-se úteis por utilizarem conjuntos de dados pequenos, como um conjunto de poucas imagens radiológicas para detecção de doenças pulmonares.

Processamento de Linguagem Natural (PLN). Pesquisas vêm sendo desenvolvidas em tarefas relacionadas a modelagem de linguagem natural, como classificação de textos [Bao et al. 2019], tradução automática de textos [Gu et al. 2018] e reconhecimento de fala [Hsu et al. 2020]. Um dos grandes desafios enfrentados no treinamento de tarefas de PLN é o baixo volume de dados rotulados para treinamento em domínios específicos, como textos relacionados à segurança pública ou notícias, e línguas extintas em tarefas de tradução de textos.

Bem-Estar Social. Técnicas de *meta-learning* podem ser utilizadas em tarefas que melhoram o estado de bem-estar social e que possuem uma quantidade muito escassa de dados, como classificação de imagens médicas [Maicas et al. 2018, Mirikharaji et al. 2019], descoberta de medicamentos [Altae-Tran et al. 2017], predições do comportamento de pandemias [Panagopoulos et al. 2021] e identificação de *fake news* [Salem et al. 2021].

3.5.3. Diferença Entre *Meta-Learning* e Outras Áreas

Além de *meta-learning*, outras técnicas, como *transfer learning* e *multi-task learning*, também aproveitam da ideia de utilizar conhecimentos prévios para o treinamento de novas tarefas de forma mais rápida ao invés de treinar uma nova tarefa do zero. Como os limites entre essas áreas nem sempre são claros, esta subseção apresenta uma explicação breve de cada uma e mostra uma comparação entre elas, ressaltando suas vantagens e desvantagens.

Transfer Learning. Nessa abordagem, explora-se o que já foi aprendido em uma determinada tarefa base, que pode já possuir um bom modelo treinado em um volume de dados relevante, para melhorar o aprendizado de uma nova tarefa que, por exemplo, não possui dados suficientes para treinar um bom modelo. A transferência do aprendizado pode se dar de várias formas, como utilizando um *feature extraction*, que reutiliza camadas e parâmetros do modelo treinado na tarefa base, ou um *fine-tuning* dos parâmetros de um modelo pré-treinado para uma nova tarefa. Considere a seguinte situação como exemplo de *transfer learning*: suponha que uma tarefa base de classificação de espécies de passarinhos que possui um grande volume de dados rotulados. Uma tarefa alvo, com pouca quantidade de dados rotulados, pode ser outra tarefa de classificação de pinguins e avestruzes. O conhecimento aprendido no treinamento da tarefa base pode ser transferido, reutilizando algumas camadas do modelo treinado e adicionando uma nova camada de classificação e, se feito corretamente, pode melhorar o desempenho da tarefa alvo[10].

Multi-Task Learning. Nessa abordagem, um conjunto de tarefas base são utilizadas no treinamento de um modelo único, baseado em uma arquitetura personalizada, que consiga generalizar bem o suficiente para várias tarefas em simultâneo. Um exemplo de *multi-task learning* seria o treinamento de um modelo utilizando como dados de entrada imagens de paisagens da natureza. Esse modelo deve conseguir generalizar bem para diversas tarefas envolvendo esse conjunto de dados, como segmentação semântica de pixeis, classificação da imagem, estimativa de profundidade, etc. Diferente do *transfer learning*, a abordagem do *multi-task learning* aprende várias tarefas simultaneamente, enquanto a primeira treina uma tarefa base inicialmente e, em seguida, transfere o aprendizado para uma tarefa alvo.

Meta-Learning. Também conhecido como um conjunto de algoritmos que cria modelos que "aprendem a aprender", esse paradigma visa o aprendizado de uma determinada tarefa alvo, explorando o aprendizado adquirido a partir de um conjunto de tarefas-base treinadas individualmente. Um exemplo do uso de *meta-learning* seria o treinamento de uma tarefa de classificação de gatos e cachorros utilizando o meta-conhecimento adquirido a partir do treinamento de outras tarefas-base, como uma tarefa de classificação de coelhos e lebres e uma classificação de leões e tigres. Quando *meta-learning* e *transfer learning* são comparadas, a maior diferença surge no procedimento de otimização dos modelos: nas tarefas que envolvem *transfer learning*, não existe um meta-objetivo a ser otimizado, ao contrário do *meta-learning*, que otimiza um meta-objetivo a partir da otimização das tarefas-base utilizadas na fase de treinamento. Já *meta-learning* comparado com *multi-task learning*, a mesma diferença com relação à otimização de modelos citada é anteriormente repetida; além disso, *multi-task learning* consegue lidar com tarefas heterogêneas, ao contrário do *meta-learning*, que lida apenas com tarefas homogêneas (como, por exemplo, apenas tarefas de classificação binária ou tarefas de regressão).

O Quadro 3.1 apresenta um resumo dos três paradigmas apresentados, enfatizando suas vantagens e desvantagens principais.

[10] *Transfer learning* parece um pouco com o processo de *fine-tuning*, mas possuem duas perspectivas diferentes: *transfer learning* ocorre quando um modelo desenvolvido para uma tarefa é reutilizado para outra tarefa, enquanto *fine-tuning* é uma abordagem que treina novamente um modelo treinado para se adequar a uma nova tarefa.

Paradigma	Vantagens	Desvantagens
Transfer Learning	- Precisa de uma menor quantidade de dados para a tarefa alvo, - Bom para extração de features.	- Modelos pré-treinados tendem a se ajustar demais à tarefa alvo, - Em geral utiliza modelos base complexos, com milhões de parâmetros, nem sempre necessários para tarefas alvo, - Concentra-se apenas em melhorar o desempenho da tarefa alvo.
Multi-task Learning	- Consegue lidar com tarefas heterogêneas e entradas multimodais, - Visa melhorar o desempenho de todas as tarefas ao mesmo tempo.	- Exige um grande conjunto de dados para treinamento, - Ao adicionar uma nova tarefa, é necessário treinar novamente a rede, pois pode exigir alterações de arquitetura.
Meta Learning	- Fácil adição de novas tarefas, - Menor quantidade de dados de treinamento necessária para uma nova tarefa. - Generalização robusta entre tarefas, - Utiliza um objetivo de meta learning (otimização de dois níveis).	- Funciona apenas para tarefas homogêneas, pois as tarefas de base e alvo devem corresponder, - Concentra-se apenas em melhorar o desempenho da tarefa alvo.

Quadro 3.1. Comparativo entre *Transfer Learning*, *Multi-task Learning* e *Meta-Learning*. Adaptado de [Upadhyay et al. 2021].

3.6. Conclusão

O presente capítulo apresentou como técnicas de *meta-learning* podem ser utilizadas no contexto de algoritmos de Aprendizado Profundo. Inicialmente, foram apresentados alguns conceitos básicos sobre Aprendizado Profundo e algumas arquiteturas comuns para tarefas de classificação de imagens[11], como as Redes Neurais Convolucionais (CNNs). Em seguida, o conceito de *meta-learning* foi definido, com suas diferentes fases, e a principal forma de organização de tarefas a serem utilizadas pelos métodos (*N-way*, *K-shot*) foi apresentada. Uma comparação entre outros tipos de aprendizado também foi introduzida (*Transfer Learning*, *Multi-task Learning* e *Meta-Learning*), enfatizando os aspectos positivos e negativos de cada uma delas. Por fim, alguns dos diversos métodos de *meta-learning* foram apresentados, divididos em métodos baseados em métricas, modelos e otimização.

[11]Tarefa escolhida para exemplificar como *meta-learning* funciona ao longo do trabalho.

Referências

[Altae-Tran et al. 2017] Altae-Tran, H., Ramsundar, B., Pappu, A. S., and Pande, V. (2017). Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293.

[Bao et al. 2019] Bao, Y., Wu, M., Chang, S., and Barzilay, R. (2019). Few-shot text classification with distributional signatures. *arXiv preprint arXiv:1908.06039*.

[Chollet 2021] Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

[Dong and Xing 2018] Dong, N. and Xing, E. P. (2018). Few-Shot Semantic Segmentation with Prototype Learning. In *BMVC*, volume 3.

[Dumoulin and Visin 2016] Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[Finn et al. 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[Finn et al. 2018] Finn, C., Xu, K., and Levine, S. (2018). Probabilistic model-agnostic meta-learning. *Advances in neural information processing systems*, 31.

[Garcia and Bruna 2017] Garcia, V. and Bruna, J. (2017). Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*.

[Garnelo et al. 2018] Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR.

[Grant et al. 2018] Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*.

[Gu et al. 2018] Gu, J., Wang, Y., Chen, Y., Cho, K., and Li, V. O. (2018). Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437*.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hochreiter and Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hsu et al. 2020] Hsu, J.-Y., Chen, Y.-J., and Lee, H.-y. (2020). Meta learning for end-to-end low-resource speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7844–7848. IEEE.

[Huisman et al. 2021] Huisman, M., Van Rijn, J. N., and Plaat, A. (2021). A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541.

[Kang et al. 2019] Kang, B., Liu, Z., Wang, X., Yu, F., Feng, J., and Darrell, T. (2019). Few-shot object detection via feature reweighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8420–8429.

[Koch et al. 2015] Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille.

[Lake et al. 2017] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

[Li et al. 2017] Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

[Maicas et al. 2018] Maicas, G., Bradley, A. P., Nascimento, J. C., Reid, I., and Carneiro, G. (2018). Training medical image analysis systems like radiologists. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 546–554. Springer.

[Mirikharaji et al. 2019] Mirikharaji, Z., Yan, Y., and Hamarneh, G. (2019). Learning to segment skin lesions from noisy annotations. In *Domain adaptation and representation transfer and medical image learning with less labels and imperfect data*, pages 207–215. Springer.

[Mishra et al. 2017] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2017). A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*.

[Munkhdalai and Yu 2017] Munkhdalai, T. and Yu, H. (2017). Meta networks. In *International Conference on Machine Learning*, pages 2554–2563. PMLR.

[Neves Oliveira et al. 2022a] Neves Oliveira, B. S., do Rêgo, L. G. C., Peres, L., da Silva, T. L. C., and de Macêdo, J. A. F. (2022a). Processamento de linguagem natural via aprendizagem profunda. *Sociedade Brasileira de Computação*.

[Neves Oliveira et al. 2022b] Neves Oliveira, B. S., do Rêgo, L. G. C., Peres, L., da Silva, T. L. C., and de Macêdo, J. A. F. (2022b). Processamento de Linguagem Natural via Aprendizagem Profunda. *Sociedade Brasileira de Computação*.

[Nichol and Schulman 2018] Nichol, A. and Schulman, J. (2018). Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4.

[Panagopoulos et al. 2021] Panagopoulos, G., Nikolentzos, G., and Vazirgiannis, M. (2021). Transfer graph neural networks for pandemic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4838–4845.

[Ravi and Larochelle 2017] Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *International Conference on Learning Representations*.

[Salem et al. 2021] Salem, F. K. A., Al Feel, R., Elbassuoni, S., Ghannam, H., Jaber, M., and Farah, M. (2021). Meta-learning for fake news detection surrounding the syrian war. *Patterns*, 2(11):100369.

[Schmidhuber 1987] Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München.

[Snell et al. 2017] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

[Thrun 1998] Thrun, S. (1998). Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer.

[Upadhyay et al. 2021] Upadhyay, R., Phlypo, R., Saini, R., and Liwicki, M. (2021). Sharing to learn and learning to share-fitting together meta-learning, multi-task learning, and transfer learning: A meta review. *arXiv preprint arXiv:2111.12146*.

[Vinyals et al. 2016] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. *Advances in neural information processing systems*, 29.

[Visca et al. 2022] Visca, M., Powell, R., Gao, Y., and Fallah, S. (2022). Deep meta-learning energy-aware path planner for unmanned ground vehicles in unknown terrains. *IEEE Access*, 10:30055–30068.

[Wang et al. 2022] Wang, C., Zhu, Y., Liu, H., Zang, T., Yu, J., and Tang, F. (2022). Deep meta-learning in recommendation systems: A survey. *arXiv preprint arXiv:2206.04415*.

[Wang et al. 2019] Wang, T.-C., Liu, M.-Y., Tao, A., Liu, G., Kautz, J., and Catanzaro, B. (2019). Few-shot video-to-video synthesis. *arXiv preprint arXiv:1910.12713*.

[Yoon et al. 2018] Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. *Advances in neural information processing systems*, 31.

[Yu et al. 2020] Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

[Zakharov et al. 2019] Zakharov, E., Shysheya, A., Burkov, E., and Lempitsky, V. (2019). Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9459–9468.

[Zhang et al. 2021] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into Deep Learning. *arXiv preprint arXiv:2106.11342*.