

UNIVERSIDADE FEDERAL DO PARANÁ

DIMMY KARSON SOARES MAGALHÃES

REPRESENTAÇÃO DE TEXTOS E ARQUITETURA DE REDES NEURAI PROFUNDAS
MISTAS USANDO UMA ABORDAGEM DE EVOLUÇÃO GRAMATICAL

CURITIBA PR

2022

DIMMY KARSON SOARES MAGALHÃES

REPRESENTAÇÃO DE TEXTOS E ARQUITETURA DE REDES NEURAIIS PROFUNDAS
MISTAS USANDO UMA ABORDAGEM DE EVOLUÇÃO GRAMATICAL

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Aurora Pozo.

CURITIBA PR

2022

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA CIÊNCIA E TECNOLOGIA

Magalhães, Dimmy.

Representação de textos e arquitetura de redes neurais profundas mistas usando uma abordagem de evolução gramatical. / Dimmy Magalhães. – Curitiba, 2023.

1 recurso on-line : PDF.

Tese (Doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientadora: Profa. Dra. Aurora Trinidad Ramirez Pozo.

1. Redes neurais (Computação). 2. Processamento de dados. 3. Programação genética (Computação). 4. Linguagem natural. I. Ramirez Pozo, Aurora Trinidad. II. Universidade Federal do Paraná. Programa de Pós-Graduação em Informática. III. Título.

Bibliotecário: Nilson Carlos Vieira Júnior CRB-9/1797

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **DIMMY KARSON SOARES MAGALHÃES** intitulada: **REPRESENTAÇÃO DE TEXTOS E ARQUITETURA DE REDES NEURAIS PROFUNDAS MISTAS USANDO UMA ABORDAGEM DE EVOLUÇÃO GRAMATICAL**, sob orientação da Profa. Dra. AURORA TRINIDAD RAMIREZ POZO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa. A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 02 de Setembro de 2022.

Assinatura Eletrônica

05/09/2022 09:31:19.0

AURORA TRINIDAD RAMIREZ POZO
Presidente da Banca Examinadora

Assinatura Eletrônica

05/09/2022 19:52:50.0

ANNE MAGALY DE PAULA CANUTO
Avaliador Externo (UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE)

Assinatura Eletrônica

05/09/2022 13:25:30.0

EDUARDO JAQUES SPINOSA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica

05/09/2022 10:18:20.0

EMERSON CABRERA PARAÍSO
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

Esse trabalho é dedicado, in memoriam, ao meu avô, João Soares. Se cheguei até aqui foi porque me apoiei no ombro dos gigantes.

AGRADECIMENTOS

A realização desse trabalho em muito se deve à colaboração de inúmeras pessoas, às quais transmito os mais sinceros agradecimentos:

- A Professora Dra. Aurora Trinidad Ramirez Pozo pela orientação dedicada e comprometida, e pelo precioso conhecimento transmitido, além da condução acadêmica, sua condução humana foi de extrema importância para esse trabalho;
- Aos meus pais, Joaquim e Socorro Magalhães pela ajuda na minha educação e formação de caráter;
- Ao irmão Jonny Magalhães pelas noites em claro em prol da ciência e do aprendizado mútuo;
- Aos meus amigos do grupo de pesquisa CBIO-Gres que sempre com espírita, eficiente e eficaz colaboração me ajudou a construir cada parte do conhecimento que constitui esse trabalho;
- A todos os amigos presentes nas calorosas discussões;
- A minha filha, Rebeca Magalhães, que com doçura e leveza compreendeu, no alto de seus 6 anos de idade, que o que papai estava fazendo era para o bem do nosso futuro;
- Ao meu filho Theo Magalhães, que nasceu no meio de uma pandemia, enquanto o pai estava se debruçando sobre códigos e teorias;
- A Dayanne de Sousa Magalhães, pelo companheirismo, pela compreensão das tantas ausências e dos intermináveis dias dedicados à computação e a inteligência artificial. Muito obrigado pelo apoio indestrutível e por tantas vezes me mostrar que era possível e tudo iria dar certo. Dedico ainda grande parte dessa conquista a ela, sem seu apoio, NADA seria possível.

RESUMO

A classificação de texto é uma das tarefas do Processamento de Linguagem Natural (do inglês, *Natural Language Processing (NLP)*). Seu principal objetivo é rotular elementos textuais, tais como frases, questões, parágrafos e documentos. Em NLP, muitas abordagens têm alcançado resultados promissores sobre essa tarefa. Abordagens baseadas em Redes Profundas têm sido constantemente usadas neste contexto, em especial as Redes Neurais Profundas (do inglês, *Deep Neural Network (DNNs)*) adicionam a habilidade de extração de características e a criação de um modelo de aprendizagem. A escala crescente e complexidade das arquiteturas DNN colocaram novos desafios para projetar e configurar DNNs. Em termos gerais, este trabalho está alinhado às tarefas de classificação de texto e como melhorar a performance de modelos já consolidados na literatura. Nossa hipótese é que a performance de modelos de classificação de texto pode ser incrementada em duas frentes complementares: 1) Representação de texto como aprimoramento contextual; e 2) Construção de modelos especializados no domínio do texto. Neste contexto, este estudo é dividido em duas partes principais: Primeiramente é apresentado uma abordagem para gerar representações de textos refinadas e semanticamente mais coesas. Para avaliar este modelo realizamos experimentos de classificação de texto consolidados na literatura que colocam à prova o modelo de representação. Em seguida, é apresentado uma aplicação da abordagem baseada em gramática evolutiva para o desenho de DNNs usando modelos baseados em *Convolutional Neural Networks (CNNs)*, *Long Short-Term Memory (LSTM)* e *Graph Neural Networks (GNNs)* aplicadas à tarefa de Classificação de Texto. Propõe-se, portanto, diferentes gramáticas, que foram definidas para capturar as características de cada tipo de rede, propondo também algumas combinações entre elas, analisando seu impacto nas arquiteturas de redes produzidas e no desempenho dos modelos gerados. Os resultados em ambas as frentes alcançam valores animadores e oferecem direções promissoras de pesquisas futuras, pois mostram que as representações de textos ganham significativos pontos semânticos e as arquiteturas projetadas têm desempenho comparável ao de suas contrapartes. Na melhor das arquiteturas, foi possível melhorar os resultados de uma rede neural estruturada manualmente em 8,18 pontos percentuais o projeto de representação de texto proposto neste trabalho.

Palavras-chave: Processamento de Linguagem Natural, Representação de Texto, Programação Genética, Redes Neurais Profundas

ABSTRACT

Text classification is one of the Natural Language Processing (NLP) tasks. Its main objective is to label textual elements, such as phrases, queries, paragraphs, and documents. In NLP, several approaches have achieved promising results regarding this task. Deep Learning-based approaches have been widely used in this context, with deep neural networks (DNNs) adding the ability to generate a representation of the data and a learning model. The increasing scale and complexity of DNN architectures was expected, posing new challenges to designing and configuring DNNs. In general terms, this work is aligned with text classification tasks and how to improve the performance of models already consolidated in the literature. Our hypothesis is that the performance of text classification models can be increased on two complementary fronts: 1) Text representation as contextual enhancement; and 2) Construction of specialized models in the text domain. So, this study is divided into two parts: Firstly we present a new approach to refine a text in order to create a comprehensive text representation model. Secondly, we present an application of a grammar-based evolutionary approach to the design of DNNs using models based on Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), and Graph Neural Networks (GNNs). We propose different grammars, which were defined to capture the features of each type of network, also proposing some combinations, verifying their impact on the produced designs and performance of the generated models. The results on both fronts reach encouraging values and offer promising directions for future research, as they show that text representations gain significant semantic points and the designed architectures have performance comparable to their counterparts. In the best of architectures, it was possible to improve the results of a manually structured neural network by 8.18 percentage points using the text representation project proposed in this work.

Keywords: Natural Language Processing, Text Representation, Genetic Programming, Deep Learning

LISTA DE FIGURAS

2.1	Fluxo genérico de construção de uma representação texto	20
2.2	Representação de palavras no espaço vetorial utilizando <i>word embedding</i> . Adaptado de [81].	25
2.3	Exemplo de análise de sentença por <i>skip-gram</i>	25
3.1	Modelo matemático simples de um neurônio. A ativação da saída da unidade $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, no qual a_i é a ativação de saída da unidade i e $w_{i,j}$ é o peso sobre a ligação da unidade i com a unidade corrente j . Adaptado de [95].	29
3.2	Exemplo de RNA multicamadas	30
3.3	Exemplo de arquitetura genérica de uma CNN	32
3.4	Exemplo de filtro aplicado à entrada de uma camada convolucional.	33
3.5	Exemplo de <i>max pooling</i> e <i>average pooling</i> a uma entrada	34
3.6	Exemplo de conversão de texto em uma representação em matriz	34
3.7	Arquitetura LSTM com <i>forget gate</i> . Fonte: [124]	35
3.8	Arquitetura geral de um <i>transformer</i>	37
3.9	Tipos de camadas de atenção. (esquerda) <i>Scaled Dot-Product Attention</i> . (direita) <i>Multi-Head Attention</i> que consiste em múltiplas camadas de atenção executando em paralelo. Fonte: [117].	38
3.10	Exemplo de funcionamento do mecanismo genérico de atenção	39
3.11	Representação gráfica da arquitetura do modelo BERT. Fonte: [22].	40
3.12	Representação de sentenças utilizando BERT. Adaptado de [22]	41
3.13	Métodos de representação de sentenças utilizando a média das incorporações (<i>embeddings</i>) ou o <i>token</i> de entrada [CLS].	42
4.1	Exemplo de Cruzamento <i>One-point</i>	47
4.2	Exemplo de mutação uniforme, no qual, probabilisticamente alguns atributos são modificados.. . . .	47
4.3	Esquema geral da abordagem proposta. O fluxo básico segue a estrutura tradicional do GA com a adição de uma etapa de mapeamento, necessária para construir a rede a partir da representação de codificação.	48
4.4	Exemplo de uma gramática simplificada para projeto de DNN. Ela define o uso de camadas convolucionais com dois parâmetros (<i>feature</i> e tamanho do <i>kernel</i>), <i>pool</i> máximo ou médio usando os parâmetros padrões e camadas densas com um parâmetro (quantidade de unidades).	49
4.5	Exemplo de genótipo usando a codificação DSGE. Cada lista de inteiros no genótipo está relacionada a uma regra (não-terminal) da gramática.	50
4.6	Processo de mapeamento DSGE para a solução apresentada na Figura 4.5.	51

4.7	Exemplo de <i>One-Point</i> e <i>Gene Crossovers</i> aplicável no DSGE.	52
4.8	A codificação divide a solução de acordo com a forma como a gramática 4.4 está estruturada, agrupando sub listas relacionadas a não-terminais à esquerda e sub listas relacionadas a terminais à direita	53
6.1	Fluxo genérico de interação entre as técnicas BECA e TextDSGE.	61
6.2	Representação dos <i>tokens</i> no espaço vetorial sem nenhuma tratativa ou pré-processamento.	63
6.3	Representação dos <i>tokens</i> no espaço vetorial com remoção de <i>tokens</i> com baixa carga semântica.. . . .	63
6.4	Fluxo genérico do funcionamento do <i>framework</i> BECA. O modelo desenvolve o algoritmo subida da encosta por k interações a fim de encontrar uma representação mais coesa.	64
6.5	Fluxo genérico de representação de texto em grafos usando o BECA	67
6.6	Fluxo genérico do funcionamento da abordagem TextDSGE.	68
6.7	Proposta de gramática para a geração de arquiteturas CNN. As camadas são definidas de acordo com a estrutura Keras (https://keras.io/) ou tipos especiais, como a regra <rate>, que é um valor aleatório dentro do intervalo definido quando selecionado.	69
6.8	Proposta de gramática para a geração de arquiteturas LSTM. Os módulos foram definidos de acordo com o <i>framework</i> Keras (https://keras.io/).	70
6.9	Gramática proposta para gerar uma arquitetura mista usando abordagens CNN e LSTM. Os módulos foram definidos de acordo com o <i>framework</i> Keras.	71
6.10	Gramática proposta para gerar arquiteturas GNN. Os módulos foram definidos de acordo com os <i>frameworks</i> Keras e Spektral (https://graphneural.network/).	72
7.1	Distribuição dos valores para modelos LSTM utilizando representações baseadas em TFIDF, <i>Sentence-BERT</i> e BECA para o dataset Ohsumed	77
7.2	Distribuição dos valores para modelos GCN utilizando representações em grafos baseadas em <i>Sentence-BERT</i> e BECA para o dataset R8.	78
7.3	Exemplo de arquitetura de GNN desenvolvida pelo TextDSGE para o dataset MR.	81
7.4	Distribuição dos valores para modelos TextDSGE utilizando representações baseadas no <i>Sentence-BERT</i> para o dataset MR	82
7.5	Distribuição dos valores para modelos TextDSGE utilizando representações baseadas no BECA para o dataset MR	83
7.6	Comparação entre os modelos baseados em LSTM utilizando todos os tipos de representação para o dataset MR.. . . .	83
7.7	Comparação entre os melhores modelos de representação a partir de cada modelo de classificação para o dataset MR	84
8.1	Visão geral das distribuições de valores para todos os modelos de classificação para todas as representações de textos utilizadas.. . . .	91

8.2	Distribuição de valores para modelos de classificação utilizando todas as representações para o dataset de petições iniciais.	92
8.3	Distribuição dos valores para as três representações usadas, independentemente do modelo de classificação utilizado para o <i>dataset</i> de petições judiciais	92
8.4	Distribuição de valores de acurácia para os modelos de classificação utilizando a representação BECA	93

LISTA DE TABELAS

3.1	Comparativo do modelo BERT com outras técnicas aplicadas a problemas de <i>benchmark</i>	41
5.1	Resumo dos trabalhos relacionados à representação de texto	57
6.1	Métodos de camada de convolução em grafos	72
7.1	Descrição estatística de cada dataset	75
7.2	Parâmetros da geração de representação usando o BECA para todos os datasets .	75
7.3	Acurácia do modelo LSTM sob a perspectiva dos diferentes tipos de representação de texto para cada um dos datasets	76
7.4	Resultados do teste de normalidade Shapiro-Wilk para o modelo LSTM para o dataset Ohsumed	76
7.5	Comparações par-a-par Dwass-Steel-Critchlow-Fligner entre as representações para o dataset Ohsumed.	76
7.6	Acurácia do modelo GCN sob a perspectiva dos diferentes tipos de representação de texto para cada um dos datasets	77
7.7	Resultados do teste de normalidade Shapiro-Wilk para o modelo GCN para o dataset R8	78
7.8	Parâmetros da busca evolutiva utilizados nos experimentos com TextDSGE para todos os datasets	79
7.9	Resultados dos experimentos para cada dataset	80
7.10	Tempo de execução da abordagem TextDSGE por rodada de execução para cada dataset	81
7.11	Resultados do teste de normalidade Shapiro-Wilk para cada modelo para o dataset MR	82
7.12	Comparações par-a-par Dwass-Steel-Critchlow-Fligner entre os melhores modelos para o dataset MR.	84
8.1	Resultados dos Experimentos para o dataset de petições iniciais.	88
8.2	Teste de distribuição normal para modelos de classificação usando dataset de petições iniciais	89
8.3	Teste Shapiro-Walk para os modelos de classificação utilizando a representação BECA.. . . .	89
8.4	Diferença estatística entre os modelos de classificação para o dataset de petições iniciais utilizando a representação BECA	90

LISTA DE ALGORITMOS

1	Algoritmo Genético	45
2	Estratégias de Evolução ($\mu + \lambda$)	46
3	Geração de soluções recursivas - DSGE	50
4	Mapeamento Recursivo de Soluções - DSGE	51
5	Algoritmo particionador de sentenças	65
6	Boostered Encoder for Classification Architecture - BECA	66

LISTA DE ACRÔNIMOS

ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
ASMO	Arab Standardization and Metrology Organization
AutoML	Automated Machine Learning
BERT	Bidirectional Encoder Representations from Transformers
BNF	Backus Naur Form
BSDS500	Berkeley Segmentation Data Set 500
BOW	Bag of Words
CNN	Convolutional Neural Network
CGP	Cartesian Genetic Programming
DINF	Departamento de Informática
DL	Deep Learning
Doc2Vec	Documents to Vectors
DS	Dialogue Systems
DSGE	Dynamic Structured Grammatical Evolution
DNN	Deep Neural Network
EA	Evolutionary Algorithm
EC	Evolutionary Computing
ELMo	Embeddings from Language Models
ES	Evolution Strategy
GA	Genetic Algorithm
GBK	Guojia Biaozhun Kuòzhǎn
GCN	Graph Convolutional Network
GE	Grammatical Evolution
GNN	Graph Neural Network
GP	Genetic Programming
ISBI	International Symposium on Biomedical Imaging
ISCII	Indian Script Code for Information Interchange
LDA	Latent Dirichlet Allocation
LGP	Linear Genetic Programming
LSA	Latent Semantic Analysis
LSTM	Long Short-term Memory
ML	Machine Learning
MT	Machine Translation
NN	Neural Network

NAS	Neuro Architecture Search
NEAT	Neuro Evolution for Augmenting Topologies
NER	Named Entity Recognition
NLP	Natural Language Processing
POS	Part-Of-Speech
PPGINF	Programa de Pós-Graduação em Informática
QA	Question-Answering
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SC	Sentiment Classification
SNNC	Shared Nearest Neighbors-Based Combiner
SL	Supervised Learning
SGD	Stochastic Gradient Descent
SRL	Semantic Role Labeling
TextGCN	Graph Convolutional Network for Text
TF-ICF	Term Frequency - Inverse Corpus Frequency
TF-IDF	Term Frequency - Inverse document frequency
UFPR	Universidade Federal do Paraná
Word2Vec	Words to Vectors

SUMÁRIO

1	INTRODUÇÃO	16
1.1	MOTIVAÇÃO.	17
1.2	OBJETIVOS	17
1.3	ORGANIZAÇÃO DO TRABALHO	18
2	PROCESSAMENTO DE LINGUAGEM NATURAL.	20
2.1	REPRESENTAÇÃO DE TEXTO.	20
2.1.1	Pré-processamento de texto	21
2.1.2	Representação de texto no espaço vetorial	23
2.1.3	Normalização	27
2.2	OBSERVAÇÕES FINAIS.	28
3	REDES NEURAIS ARTIFICIAIS	29
3.1	REDES NEURAIS PROFUNDAS	31
3.1.1	<i>Convolutional Neural Network - CNN</i>	32
3.1.2	<i>Long short-term memory - LSTM</i>	34
3.1.3	<i>Graph Neural Network - GNN</i>	36
3.1.4	<i>Transformers</i>	36
3.2	BERT - <i>BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS</i>	39
3.2.1	Sentence-BERT	41
3.3	OBSERVAÇÕES FINAIS.	43
4	PROJETO DE REDES NEURAIS USANDO EVOLUÇÃO GRAMATICAL	44
4.1	DESENVOLVIMENTO DE ALGORITMOS COM COMPUTAÇÃO EVOLUTIVA	44
4.1.1	Codificação	45
4.1.2	Seleção	46
4.1.3	Cruzamento	46
4.1.4	Mutação	46
4.2	EVOLUÇÃO GRAMATICAL	47
4.3	EVOLUÇÃO GRAMATICAL DINAMICAMENTE ESTRUTURADA.	48
4.3.1	Gramáticas para arquiteturas de Redes Neurais Profundas	49
4.3.2	Operadores Genéticos.	52
4.4	OBSERVAÇÕES FINAIS.	53

5	TRABALHOS RELACIONADOS	54
5.1	PROCESSAMENTO DE LINGUAGEM NATURAL	54
5.1.1	Representação de Texto	54
5.1.2	Classificação de Texto.	56
5.2	DESIGN DE REDES NEURAIIS BASEADOS EM ALGORITMOS EVOLUTIVOS	58
5.2.1	Busca por Arquiteturas Neurais.	58
5.2.2	Modelos baseados em evolução.	58
5.3	OBSERVAÇÕES FINAIS.	59
6	ABORDAGENS PARA REPRESENTAÇÃO E CLASSIFICAÇÃO DE TEXTOS COMO MODELOS OTIMIZADOS	60
6.1	QUESTÕES DE PESQUISA	60
6.2	MODELO BECA	61
6.2.1	BECA	63
6.2.2	Representação de textos utilizando Grafos	66
6.3	MODELO TEXTDSGE.	68
6.3.1	Operadores	68
6.3.2	Gramática CNN.	69
6.3.3	Gramática LSTM	70
6.3.4	Gramática CNN+LSTM	71
6.3.5	Gramática GNN.	71
6.4	OBSERVAÇÕES FINAIS.	73
7	EXPERIMENTOS E RESULTADOS	74
7.0.1	<i>Datasets</i>	74
7.0.2	BECA	75
7.0.3	TextDSGE.	78
7.1	OBSERVAÇÕES FINAIS.	83
8	APLICAÇÃO DO MODELO BECA+TEXTDSGE NO DOMÍNIO JURÍDICO	86
8.1	CONTEXTO JURÍDICO	86
8.2	METODOLOGIA.	87
8.3	EXPERIMENTOS E RESULTADOS	88
8.4	OBSERVAÇÕES FINAIS.	89
9	CONSIDERAÇÕES FINAIS	94
9.1	CONTRIBUIÇÕES E LIMITAÇÕES	94
9.2	DIRECIONAMENTOS FUTUROS	96
9.3	PUBLICAÇÕES	96
	REFERÊNCIAS	98

1 INTRODUÇÃO

A extração de conhecimento de textos é cada vez mais popular nos últimos anos devido a sua onipresença na internet, seja por meio de redes sociais, *e-mails*, bibliotecas digitais ou outros meios [3]. A extração de informações úteis do texto com vários tipos de algoritmos é conhecida como mineração de texto, análise de texto ou aprendizado de máquina com texto. Segundo [115] existem diferentes técnicas de extração de informações de texto disponíveis que são aplicadas a diferentes áreas, tais como: classificação de texto (do inglês, *text classification*), recuperação de informação (do inglês, *information retrieval*), clusterização de documentos (do inglês, *text clustering*), extração de informações (do inglês, *information extraction*), e *web mining*.

A classificação de texto é uma técnica de Processamento de Linguagem Natural (do inglês, *Natural Language Processing*, **NLP**) que consiste em organizar informações atribuindo texto a um conjunto de categorias semânticas. No NLP, a ideia principal é treinar um classificador para aprender com exemplos e agrupá-los automaticamente em categorias. A pesquisa em classificação de texto não é nova, mas foi apenas na última década que as abordagens baseadas em redes neurais ganharam destaque com muitas aplicações bem-sucedidas. Em particular, abordagens baseadas em redes neurais profundas (DNNs) tornaram-se proeminentes na classificação de texto devido à sua capacidade de aprender padrões de dados usando extratores de recursos hierárquicos [25]. Nesse contexto, diversos estudos na literatura utilizam arquiteturas baseadas em DNN, como Rede Neural Convolutiva (CNN)[60], *Long Short-Term Memory* (LSTM)[38] e Rede Neural baseadas em Grafo (GNN)[98]. Estes trabalhos evidenciam a importância do modelo de relação semântica. Encontrar um modelo capaz de refletir essa relação entre as palavras e os documentos repercute diretamente nas tarefas seguintes como a de classificação de textos.

Ainda no contexto da aplicação de redes neurais artificiais, existe uma escalada da complexidade das arquiteturas. O conjunto de parâmetros de configuração, a arquitetura da rede, e a adaptação do modelo para tarefas como classificação de texto representa um desafio para a configuração desses sistemas [80]. Pode-se relacionar a esse desafio a área de Busca de Arquiteturas Neurais (do inglês, *Neural Architecture Search*, **NAS**), o processo de automação da engenharia da arquitetura das redes neurais [25]. A neuroevolução pode ser considerada um subcampo das técnicas NAS, com foco em técnicas baseadas na evolução aplicadas à busca de arquitetura. Sua inspiração vem do processo evolutivo natural dos cérebros, supondo que essas abstrações dos cérebros podem ser evoluídas artificialmente por meio de algoritmos evolutivos [107].

Vários algoritmos evolutivos (EAs) são aplicados ao projeto de DNNs e cada um deles apresenta diferentes abordagens e oferece diferentes ferramentas para realizar essa tarefa. A Evolução Gramatical Estruturada Dinâmica (DSGE) foi recentemente proposta para o *design* de DNN [4, 62, 63]. O DSGE é um algoritmo evolutivo baseado em gramática, proposto como uma expansão do algoritmo tradicional de Evolução Gramatical (GE) [96]. A GE, por sua vez, é uma variação da Programação Genética, juntamente com a Programação Genética Baseada em Árvore (GP) [90]. Como uma técnica baseada em gramática, o espaço de projetos possíveis é definido por meio de uma gramática que contém blocos de construção. Com base nas regras definidas, esses blocos de construção descrevem os vários componentes de uma solução (DNN) que podem ser combinados de várias maneiras para construir uma arquitetura flexível. Este trabalho, portanto, foca na construção de estruturas computacionais capazes de criar relações contextuais fortes entre textos e gerar modelos de classificação de textos aprimorados para tais textos.

Por conseguinte, a seguir, são descritos as motivações, delimitação do escopo e objetivos deste trabalho.

1.1 MOTIVAÇÃO

A pesquisa no processamento de linguagem natural não é nova. Modelos formais de representação de texto datam de 1950 com a publicação de [16]¹, na qual o autor afirmou que a linguagem humana é uma atividade cognitiva previsível, sistemática e lógica, passível de ser representada por um metamodelo. A análise automática de textos se tornou ainda mais popular recentemente por conta da propagação de dados proporcionada pelo advento principalmente da internet. Além de disseminar e popularizar o uso de *e-mails* e redes sociais, a internet auxilia na difusão de inúmeras atividades, tais como médicas, econômicas e jurídicas.

A produção de dados tem alcançado índices nunca registrados. Nos últimos anos têm-se produzido mais dados do que em toda a história da humanidade [92]. Segundo [40] são produzidos em média 2.2 milhões de *terabytes* de novos dados por dia e até o final de 2020 serão 40 trilhões de *gigabytes* de dados em todo o mundo. Grande parte desses dados são produções textuais, como artigos, relatórios, publicações, notícias e *e-mails*, visto que os textos são a forma mais tradicional e simples de transmissão do conhecimento [3].

Muitos domínios do conhecimento, tais como o médico, financeiro e jurídico, têm-se beneficiado da revolução proporcionada pela tecnologia e, conseqüentemente, aumentado sua produção textual. O judiciário brasileiro, por exemplo, tem cerca de 80.1 milhões de processos em seu acervo [27]. Este número era de 74 milhões no ano anterior [26]. Tais dados são justificados principalmente pelo advento de sistemas computacionais para o trato judicial, os quais agilizam e expandem o acesso à justiça. Por outro lado, isto significa que, com a atual demanda, a quantidade de processos judiciais julgados é menor do que a quantidade de processos judiciais novos e este comportamento é análogo ao que acontece nas outras áreas. Desta forma, a análise de textos por técnicas de inteligência artificial tem representado uma das principais formas de minimizar a lacuna entre produção e análise de textos.

Devido ao crescente número de processos judiciais e a busca cada vez mais comum pela justiça, reforça-se o discurso de que a justiça brasileira não possui recursos suficientes para sanar o déficit de baixa de processos judiciais com o *modus operandi atual*. É consenso, portanto, que a tecnologia pode ajudar a amenizar esse quadro, principalmente utilizando técnicas de inteligência artificial, combinando inúmeras tarefas já consolidadas em outras áreas do domínio do conhecimento. A área de NLP é uma forte candidata na otimização do trato jurídico. O uso do processamento de texto, com técnicas de agrupamento, sumarização, extração de informações, classificação de texto pode, em última instância, auxiliar o operador do direito no processo de construção de argumentação jurídica ou mesmo no decisório.

1.2 OBJETIVOS

O objetivo deste trabalho é desenvolver uma metodologia de representação e classificação de textos, utilizando aspectos especializados de pré-processamento, construção de representações de texto refinadas, técnicas otimizadas de contextualização e classificação de textos.

Para tal, foram realizados estudos e experimentos salientando-se os seguintes aspectos

- Analisar as diferentes técnicas de representação textual, propondo aperfeiçoamento na construção de uma nova estrutura;

¹Publicado em 1955 em forma de mimeógrafo, e em definitivo em 1975

- Construir uma abordagem de *design* automática de arquitetura de redes neurais ajustada ao problema de classificação de texto;
- Verificar a aplicabilidade da representação textual e o modelo de classificação de textos propostos no **contexto de classificação de processos jurídicos**.

Neste contexto, é importante elencar os macro-desafios deste trabalho, listados a seguir:

1. A extensão dos algoritmos de representação textuais, tais como Word2Vec[81], Doc2Vec[59], *Sentence-BERT*[93], dentre outros, a fim de gerar uma representação refinada, adaptável ao contexto do domínio de aplicação. Sob a hipótese de que alguns termos do texto carregam informação semântica superior a outros, propõe-se uma abordagem baseada em subida da encosta que pondera e seleciona termos de acordo com sua carga semântica. Para tanto, usou-se uma estrutura de particionamento contextual dos textos e uma seleção de termos automática a fim de criar sentenças sintéticas sem a intervenção de remoção de termos por conhecimento externo ao conjunto de dados. O propósito geral aqui é encontrar mecanismos de representação textual que consigam capturar semântica dos textos independente do domínio de aplicação e da intervenção de especialistas, o que ainda é comum nos modelos tradicionais de representação de texto;
2. A extensão do algoritmo DSGE original [4], propondo uma abordagem gramatical aprimorada para DNNs em que os blocos de construção relacionados são definidos para as arquiteturas particulares para orientar o projeto estrutural. Usa-se codificação indireta para representar as soluções, incluindo um processo de mapeamento que traduz as soluções codificadas em redes atuais. Isso oferece uma maneira simples de cobrir possíveis variações nas arquiteturas e simplifica a combinação de soluções. Por fim, o mecanismo de busca é baseado em um Algoritmo Genético (AG), incluindo operadores de recombinação adaptados e avaliação de aptidão com treinamento tradicional, validação e execução de testes. Nosso objetivo é oferecer uma ferramenta flexível que permita o desenho de redes e também seja suficientemente adaptável a qualquer problema de neuroevolução na classificação de textos.

Espera-se que as abordagens propostas neste trabalho sejam capazes de apresentar representações de textos semanticamente coesas a fim de auxiliar na extração de informações refinadas dos documentos, além de contribuir para a pesquisa de Programação Genética e Gramática Evolutiva. Concomitantemente, procuram estreitar os laços entre a academia e o meio jurídico, impulsionando o caráter inovador do primeiro com a necessidade latente por evolução do segundo.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 7 capítulos. Este capítulo apresenta uma breve introdução do trabalho, no qual são apresentadas as motivações e objetivos gerais.

O Capítulo 2 trata dos principais conceitos e técnicas, necessários para o entendimento do trabalho no que se refere ao processamento de linguagem natural, destacando-se: Pré-processamento e Representação de Texto

O Capítulo 3 trata dos principais conceitos e técnicas relacionados a rede neurais artificiais. São apresentados ainda conceitos e ferramentas relacionadas as redes neurais profundas. Por fim é elaborada uma visão detalhada do algoritmo BERT.

O Capítulo 4 destaca os principais conceitos e técnicas relacionadas à gramática evolutiva, bem como a aplicabilidade de conceitos evolutivos para construção de redes neurais artificiais.

O Capítulo 5 enfoca os trabalhos relacionados, no qual são descritos os estudos realizados na última década sobre processamento de textos, destacando principalmente tarefas de representação de texto, classificação de texto e design de arquiteturas de redes neurais artificiais usando uma perspectiva de gramáticas evolutivas.

O Capítulo 6 apresenta os dois modelos propostos: BECA e TextDSGE. O primeiro é responsável pela geração de representação de texto. O segundo é uma extensão do DSGE tradicional aplicado ao problema de classificação de texto.

No Capítulo 7 são desenvolvidos experimentos que buscam validar as ferramentas propostas e são apresentados os direcionamentos futuros para as abordagens propostas.

Em seguida, o Capítulo 8 descreve uma aplicação no contexto jurídico das abordagens propostas. São desenvolvidos diversos experimentos que buscam validar as ferramentas propostas fora do contexto de *benchmarks*. Neste capítulo são apresentados direcionamentos futuros para as abordagens propostas no contexto jurídico.

Por fim, o Capítulo 9 elenca as contribuições do trabalho apresentado, produções técnicas e trabalhos submetidos e apresentados decorrentes direta ou indiretamente da pesquisa desenvolvida.

2 PROCESSAMENTO DE LINGUAGEM NATURAL

Nesta seção serão descritos os principais aspectos relacionados ao Processamento de Linguagem Natural (do inglês, *Natural Language Processing*, **NLP**) e o fluxo genérico de pré-processamento e representação de textos no espaço vetorial. Serão descritas ainda algumas das principais tarefas relacionadas a NLP. Por fim descrever-se-á o funcionamento do BERT [22], ferramenta base para a construção da abordagem de representação de texto proposta neste trabalho.

2.1 REPRESENTAÇÃO DE TEXTO

Uma das primeiras atividades do processamento de linguagem natural é a capacidade de construção de uma representação abstrata do texto [47]. Todas as tarefas de NLP pressupõem a existência de um modelo computacional capaz de converter o texto escrito em algo inteligível para o computador. Em muitas representações de textos, as palavras têm uma representação binária, isto é, se uma palavra aparece em um texto, ela tem o valor 1, senão 0. Outras representações utilizam uma função normalizada da frequência da palavra. Outras ainda utilizam pesos treinados a partir de uma rede neural artificial. Em geral, as representações de texto têm alta dimensão, são esparsas e não-negativas.

A representação de texto pode ser entendida como uma sequência de etapas a fim de produzir um modelo abstrato e computável dos documentos. A Figura 2.1 mostra um fluxo genérico dessa representação. As etapas em azul são consideradas primordiais para o processo por gerarem artefatos fundamentais para a representação do texto. As etapas em amarelo podem ser consideradas opcionais, visto que em alguns domínios de aplicação, podem ocasionar efeito desvantajoso [3].

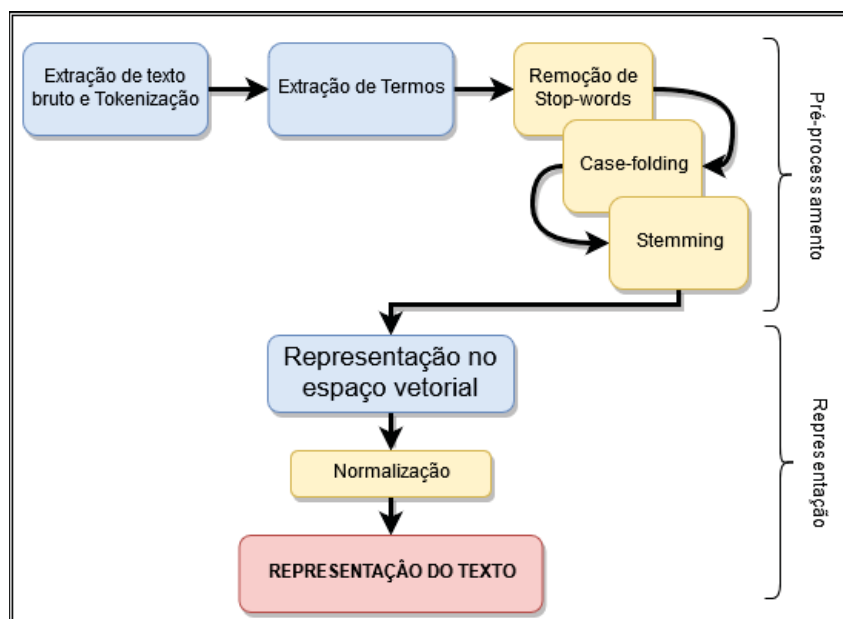


Figura 2.1: Fluxo genérico de construção de uma representação de texto

Segundo [3], o pré-processamento de texto possui alguns pontos que devem ser considerados:

- Algumas ocorrências de palavras têm pouco significado no processo de análise semântica do texto (artigos definidos, por exemplo);
- Algumas palavras são simplesmente variações de gênero ou número de outras palavras e podem ocasionar deflexão de sentido na representação;
- Muitos textos, principalmente oriundos de revisões, mídias sociais e afins, possuem erros de escrita e podem ocasionar alguma distorção;
- Dependendo do domínio de aplicação, algumas etapas de pré-processamento podem ser omitidas.

Ademais, para a representação de texto é comum que o **termo** é a melhor unidade para representação. Embora um documento de texto expresse uma vasta gama de informações, infelizmente, ele não possui uma estrutura especialmente computável. Por outro lado, alguns modelos mais atuais (como BERT [22] e Doc2Vec [59]) buscam uma representação de texto mais genérica, que represente o documento completo.

2.1.1 Pré-processamento de texto

Nesta seção são descritas as principais atividades relacionadas ao pré-processamento de texto (Figura 2.1) e seu impacto nessa representação. Estas atividades são realizadas com os objetivos de: 1) estruturar os textos em elementos computáveis; 2) reduzir a complexidade da representação textual e; 3) atenuar alguns problemas decorrentes de características dos textos, como polissemia.

2.1.1.1 Extração de texto bruto e Tokenization

A primeira etapa do processo de representação é a conversão do texto em elementos computáveis. Isto acontece quando os textos não estão representados por uma sequência de caracteres (por exemplo, conjunto de *bytes*) e segundo [3] os seguintes aspectos devem ser observados nesta conversão:

1. Um texto específico pode apresentar um *encoding* particular, isto é, uma formatação/codificação dos caracteres específica;
2. O idioma de um documento define o conjunto de caracteres a ser usado.

Para tratar estes dois aspectos, a extração de texto bruto deve primeiro obter o *encoding* dos documentos. Em muitos casos, principalmente para idiomas latinos, é usada a codificação Latin 1. Para textos em inglês e outras línguas anglo-saxônicas, é comum o uso de ASCII. O padrão que se adotou na maioria das ferramentas de aprendizado de máquina é o Unicode, que suporta conjuntos de caracteres latinos (com acentos e elementos especiais) como anglo-saxônicos. Entretanto, existem inúmeros outros conjuntos disponíveis, dos quais destacam-se o ASMO 708 para textos arábicos, GBK para chineses e ISCII para variações da língua indiana.

Após o processo de conversão, tem-se a *tokenização* (do inglês, *Tokenization*), que é o processo de converter um documento em uma sequência estruturada de palavras, isto é, uma sequência de *tokens*. Um *token*, por sua vez, é uma sequência contínua de caracteres que possui um significado semântico. Importante determinar que cada palavra em um texto é considerada um *token*, mesmo que ela ocorra mais de uma vez. Por exemplo, considerando a seguinte sentença (adaptado de [3]):

O coelho, depois de comer cenouras por horas, dormiu por um dia inteiro.

Neste caso, os *tokens* são os seguintes:

{ "O", "coelho", ",", "depois", "de", "comer", "cenouras", "por", "horas", ",", "dormiu", "por", "um", "dia", "inteiro" }

Observam-se neste processo de tokenização três pontos importantes: 1) Os *tokens* estão escritos como em sua forma original (letras maiúsculas e minúsculas) mesmo que alguns métodos de tokenização convertam todos os *tokens* em letras minúsculas. 2) Os símbolos gramaticais (como pontos, vírgulas, etc.) são apresentados como *tokens*. 3) Algumas palavras aparecem repetidas (como a palavra "por"). Neste caso, a unificação deve ser realizada (quando necessária).

Neste contexto destacam-se algumas atividades e problemas, tais como: que tipo de separação deve ser adotado no processo de tokenização? É comum que os espaços em branco sejam a base de separação (e foi usado no exemplo anterior), mas alguns problemas decorrem desta ação. Por exemplo, na sentença: "Ayrton Senna, piloto de Fórmula 1, foi tricampeão mundial pela equipe McLaren", o processo de tokenização deve separar os termos "Ayrton Senna" ou "Fórmula 1"? Em muitos casos, dicionários semânticos podem ser usados para realizar o processo de decisão de separação.

Outro ponto é que é possível não criar uma segmentação estrita da sequência de caracteres, mas também extrair sequências de caracteres sobrepostas. Vale ressaltar que é preciso distinguir entre tokenização de baixo nível e tokenização de alto nível. O reconhecimento de frases linguisticamente coerentes é um exemplo de tokenização de alto nível e requer um nível mínimo de processamento linguístico. Em muitos casos, a fase de baixo nível da tokenização básica é seguida por uma fase de alto nível de recriação de *tokens* semanticamente mais significativos da tokenização inicial [3].

O processo de remoção da pontuação pode ser realizado antes ou depois da tokenização. Esta decisão deve ser tomada a partir do uso ou não da tokenização de alto nível, visto que *tokens* podem conter símbolos e a remoção prematura de pontuação pode significar a perda de algum aspecto semântico do *token*. Normalmente, vírgulas, dois pontos e pontos são símbolos removidos sem um critério maior de análise. Dependendo do domínio de conhecimento, como o jurídico, símbolos de interrogação e exclamação podem modificar a carga semântica que uma sentença pode ter. Removê-los acarreta uma mudança clara de sentido.

2.1.1.2 Remoção de stop words

As palavras de parada (do inglês, *stop word*) são geralmente palavras que têm pouco (ou nenhum) poder discriminante para o processo de linguagem natural. Normalmente, pronomes, artigos e preposições são considerados *stop words*. Algumas palavras, ao se realizar uma análise de frequência nos documentos, podem também ser consideradas *stop words*. Para tanto, assume-se o seguinte guia para remoções:

1. Artigos, preposições, conjunções são removidas. Pronomes, em alguns processos de remoção, também podem ser considerados *stop words*;
2. Alguns idiomas possuem *stop words* próprios;
3. A frequência de *tokens* pode ser analisada, e um limiar pode ser adotado para remover uma palavra.

Não é claro que, para todos os domínios de conhecimento, a remoção de *stop words* seja benéfica. É comum a remoção de tais palavras por dois aspectos principais: 1) Capacidade discriminante e 2) Capacidade computacional. Entretanto, [3] propõe que seja analisado cada domínio de aplicação.

2.1.1.3 Case-folding

Dependendo do domínio de conhecimento dos documentos, as palavras de um texto podem ter semântica diferente se estão ou não com letras maiúsculas. Por exemplo: no domínio de revisões de filmes, as palavras *Rose* e *rose* podem ter significados diferentes (uma personagem de um filme, ou o nome de uma flor). O tratamento de *case-folding* deve ser considerado caso a caso, dependendo do domínio de conhecimento. Entretanto, os modelos mais comuns de representação de texto (como o BERT - [22]) consideram todas as palavras como minúsculas, atribuindo sua relevância e representação a partir da avaliação de contexto.

2.1.1.4 Stemming

Stemming é o processo de consolidar palavras relacionadas em um mesmo radical, retirando afixos morfológicos. Isto significa, por exemplo, reduzir expressões que estão no plural para o seu singular, ou mesmo converter tempos verbais para um apenas (por exemplo: "comeram", "comi", "comiam"). As técnicas mais comuns para realizar esta tarefa são descritas a seguir [3]:

- Tabelas de consulta semiautomáticas: são tabelas nas quais as palavras e suas variações são armazenadas em tabelas.
- Retirar sufixos e prefixos: armazenar algumas regras de variação de número, gênero, grau e tempo de algumas palavras.
- Lematização: usa a parte específica do texto para determinar a forma raiz de uma palavra. As regras de normalização dependem de construções específicas do idioma.

2.1.2 Representação de texto no espaço vetorial

Rossi [94] afirma que a qualidade dos resultados de algoritmos de aprendizado de máquina aplicados a textos é diretamente proporcional à qualidade da representação da coleção de textos. Portanto, devido à capacidade de coesão e coerência, é legítima a preocupação com a construção de representações de texto.

Após a conversão dos textos de uma forma não estruturada de dados em estruturada (descrito na Seção 2.1.1), segue a construção de um modelo eficiente de representação computacional sobre o qual os algoritmos de aprendizado de máquina sejam capazes de processar. É comum na literatura que os textos sejam representados no espaço de vetores. Neste tipo de representação, cada documento $d \in D$ é representado por um vetor X , no qual cada dimensão diz respeito a uma característica do documento.

Em muitos modelos de representação no espaço vetorial, uma característica representa a ocorrência ou não de uma palavra em um documento, sendo seu valor positivo caso aquela palavra esteja presente, isto é, $f(w) > 0$ se $w \in D$ [3]. Neste caso, o valor de $f(.)$ é dito binário. Este tipo de valoração, conhecido como *Bag of Words (BoW)*, não é muito comum quando se tem grande quantidade de textos e vocabulário extenso.

O BoW é utilizado quando o algoritmo de aprendizagem de máquina exige estritamente representações binárias. Entretanto, este modelo possui algumas limitações [37]:

- Alta dimensionalidade: cada texto deve ser representado por um vetor de tamanho $|v|$ (tamanho do vocabulário). É importante observar que, mesmo após o uso de técnicas de pré-processamento (tal como remoção de *stop words* 2.1.1.2), esse conjunto pode ser muito grande, contendo centenas de milhares de palavras, o que gera uma representação de grande dimensão;
- Perda de correlação: a simples marcação de existência de uma palavra em um documento faz com que o BoW não consiga construir relações entre palavras;
- Perda de semântica: dado que não existem relações entre palavras adjacentes, informações de semântica são perdidas neste tipo de representação;
- Representação esparsa: é comum que a representação binária gere um conjunto de vetores esparsos, o que dificulta o relacionamento contextual entre documentos.

Devido às limitações apontadas para o BoW, é comum usar-se uma variação deste modelo que utiliza a frequência normalizada dos termos. Este modelo é referenciado como frequência do termo–inverso da frequência nos documentos (do inglês, *frequency-inverse document frequency*, **TF-IDF**) que pode ser definida como uma medida para determinar a frequência relativa de uma palavra em um documento comparada com a proporção inversa daquela palavra em relação a todo o conjunto de documentos [91].

De maneira formal, pode-se definir que dada uma coleção de documentos D , uma palavra w , e um documento $d \in D$, pode-se inferir a Equação 2.1.

$$w_d = f_{w,d} * \log\left(\frac{|D|}{f_{w,D}}\right) \quad (2.1)$$

na qual $f_{w,d}$ é o número de vezes que w aparece em d , $|D|$ é o tamanho do conjunto de documentos, e $f_{w,D}$ é o número de vezes em que a palavra w aparece em D [97]. Desta forma, é possível representar uma palavra através de um vetor que determina sua frequência relacionada com cada documento do *dataset*. Por sua vez, um documento é representado por um vetor de tamanho $|v|$, que representa a quantidade de palavras do vocabulário, e cada característica do documento expressa o valor TF-IDF daquela palavra em relação àquele documento.

O TF-IDF não resolve completamente alguns problemas do BoW, mas a abordagem é comumente usada para ponderar cada palavra em um texto de acordo com o quão única ela é, atribuindo alguma carga semântica para a palavra [51]. Muitos modelos de classificação de texto atualmente utilizam a abordagem TF-IDF como métrica auxiliar para sua abordagem.

Uma alternativa à representação por frequência são as baseadas em incorporação de palavras (do inglês, *word embedding*). Os *word embeddings* são uma forma de representação do vocabulário de um conjunto de documentos capazes de capturar o contexto, semântica, similaridade sintática e relações entre palavras.

Liu [66] afirma que os *word embeddings* tiveram um papel vital na construção de vetores contínuos de palavras baseados no contexto no qual estão inseridas e em grandes *datasets*. A premissa básica do modelo é que cada palavra pode ser representada por um único vetor, que preserva distâncias próximas de outros vetores de outras palavras do mesmo contexto, sintaxe ou semântica. A Figura 2.2 exemplifica o relacionamento semântico entre palavras quando o contexto é considerado.

Formalmente, um *word embedding* é determinado como o mapeamento $V \rightarrow \mathbb{R}^T : w \rightarrow \vec{w}$, que mapeia uma palavra w de um vocabulário V para um vetor real $\vec{w}$ em um espaço vetorial de dimensão T [99]. Os modelos baseados em *word embedding* tem impulsionado pesquisas na área de NLP (destacadas no Capítulo 5).

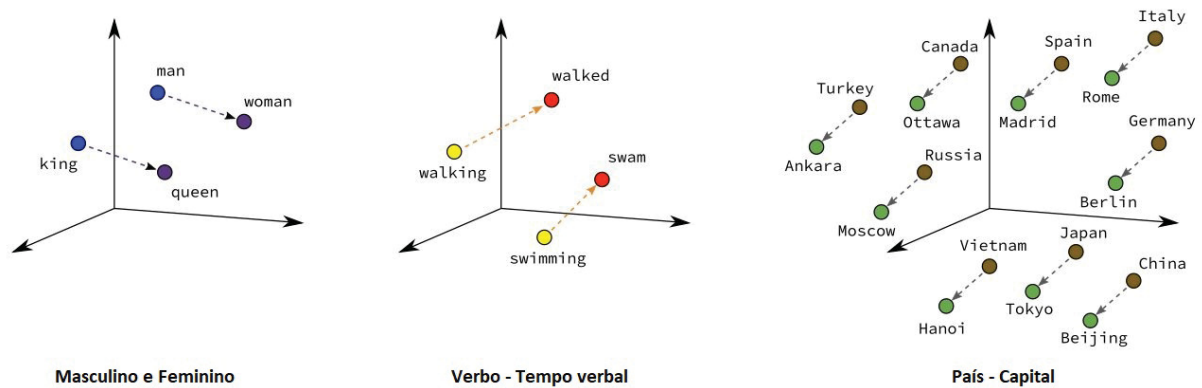


Figura 2.2: Representação de palavras no espaço vetorial utilizando *word embedding*. Adaptado de [81].

Destaca-se neste contexto o trabalho de [81] que propuseram o *Word2Vec*, um modelo computacional de representação de palavras para grandes *datasets* utilizando redes neurais. O principal insumo do trabalho é a utilização de um *skip-gram* para construção da representação e relacionamento de palavras.

Um *skip-gram* é uma das técnicas de aprendizado não-supervisionadas usadas para encontrar as palavras mais relacionadas a uma determinada palavra. Basicamente, *pulam* palavras para se ter uma relação contextual mais ampla de uma sentença. Então, o conjunto $S(k, n)$ de todos os k -skip- n -grams de uma sentença $s = w_1 w_2 w_3 \dots w_m$ é dado pela Equação 2.2.

$$S(k, n) = \left\{ w_{i_1} w_{i_2} \dots w_{i_n} \mid \sum_{j=2}^n (i_j - i_{j-1} - 1 \leq k, i_j > i_{j-1}, \forall j) \right\} \quad (2.2)$$

O conjunto $S(k, n)$ é, então, formado pela tupla de n palavras dentro de uma janela de contexto de tamanho k . É possível observar que todos os n -grams podem ser considerados *0-skip- n -grams*. Desta forma, este modelo representa uma generalização natural de n -grams [3]. A Figura 2.3 exemplifica o funcionamento de um skip-grams com $k = 2$ e $n = 2$.

O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA
O	COELHO	COMEU	A	CENOURA

Figura 2.3: Exemplo de análise de sentença por *skip-gram*

O conjunto final de skip-grams para a Figura 2.3 é dado por: "O COELHO", "O COMEU", "O A", "COELHO COMEU", "COELHO A", "COELHO CENOURA", "COMEU A", "COMEU CENOURA", "A CENOURA".

Formalmente, dado um conjunto de palavras V e seus contextos C^1 , considera-se a probabilidade condicional de uma palavra w como $p(c|w)$ e um dado *dataset* D , o objetivo é encontrar o conjunto de parâmetros θ de $p(c|w; \theta)$ que maximize a probabilidade de todo o *dataset* maximizando a Equação 2.3.

$$\operatorname{argmax}_{\theta} \prod_{w \in D} \left[\prod_{c \in C(w)} p(c|w; \theta) \right] \quad (2.3)$$

na qual $C(w)$ é o conjunto de contextos da palavra w . O modelo *Word2Vec* utiliza uma abordagem de parametrização baseada em redes neurais utilizando a função da Equação 2.4:

$$p(c|w; \theta) = \frac{e^{v_c * v_w}}{\sum_{c' \in C} e^{v_{c'} * v_w}} \quad (2.4)$$

na qual v_c e $v_w \in \mathbb{R}^T$ são vetores que representam o contexto c e a palavra w respectivamente, e C é o conjunto de todos os contextos. Os parâmetros θ são v_{c_i} , v_{w_i} para $w \in V$, $c \in C$, $i \in 1, \dots, T$ (um total de $|C| \times |V| \times T$ parâmetros) [32].

Outra abordagem empregada pelo *Word2Vec* é a utilização de um *Bag of Words* contínuo (do inglês, *Continuous Bag of Words*, **CBOW**). Nesta abordagem, o objetivo é determinar uma palavra w_i dado o contexto C com outras palavras w_j em que $i \neq j$. O objetivo é encontrar um conjunto de parâmetros θ de $p(w_i|w_j; \theta)$ que maximize a probabilidade de todo o *dataset* sob a Equação 2.5.

$$p(w_i|w_j; \theta) = \frac{e^{v_{w_0} * v_{w_j}}}{\sum_{i=1}^V e^{v_{w_i} * v_{w_j}}} \quad (2.5)$$

na qual $v_w \in \mathbb{R}^T$ é vetor que representa a palavra w , V é conjunto de palavras (ou vocabulário). Os parâmetros θ são v_{w_j} para $w \in V$, $j \in 1, \dots, T$ e $j \neq i$ (um total de $|V| \times |V| \times T$ parâmetros) [32].

O modelo de incorporação de palavras é tipicamente construído para ser rápido e escalável sob uma estrutura de dados não rotulados. Entretanto, mesmo o *Word2Vec* tendo revolucionado a forma como as palavras podem ser representadas no espaço de vetores, não é relativamente claro nem otimizada a utilização de técnicas de *word embedding* para representação de sentenças completas [50].

Para resolver este tipo de problema, [59] propuseram o *Doc2Vec* que mapeia parágrafos em um documento para um único vetor usando uma combinação de representação vetorial de palavras e representações dos parágrafos. Basicamente este modelo utiliza o *Word2Vec* como uma sub-rotina adicionando uma representação vetorial da identificação do parágrafo e utiliza técnicas de médias e concatenação para chegar a um vetor final que represente todo o parágrafo. Então, quando o vetor da palavra $w \in V$ é treinado, o vetor do documento $d \in D$ também é treinado, isto garante sua representação numérica.

Recentemente outros modelos têm ganhado destaque na representação de texto, tais como ELMo e BERT. O *Embeddings from Language Models (ELMo)* [87] propõe a combinação de representações de camadas intermediárias de modelos de linguagem bidirecional (biLM), buscando uma representação dependente do contexto e em função de um *token* t_k prever o *token* t_{k+1} . No sentido inverso o modelo é treinado em função da predição do *token* anterior dado um contexto futuro, combinando as técnicas de skip-gram e CBOW. Desta forma, para cada *token* t_{k+1} uma camada do biLM computa um conjunto de $2L + 1$ representações em função da predição de t_k , gerando assim a conjunto de representações para cada *token* em que são agrupadas.

¹Pode ser definido como contexto as n palavras adjacentes dada uma janela deslizando

O *Bidirectional Encoder Representations from Transformers (BERT)* [22] consiste em um modelo de representação de texto projetado para pré-treinar as representações bidirecionais por meio do refinamento conjunto em todos os contextos e em todas as camadas da rede. Os autores, indicam que as representações geradas a partir do BERT tendem a ser mais refinadas do que aquelas geradas apenas através de uma camada de saída de uma rede neural. Formalmente a arquitetura do BERT é definida como um transformador codificador bidirecional de multicamadas descritos em [22] e [117]. Este modelo é discutido com mais detalhes na Seção 3.2.

2.1.3 Normalização

Por fim, a etapa de normalização em processos de aprendizado de máquina diz respeito a adequar os dados. O objetivo da normalização é modificar e adaptar os valores numéricos o que é especialmente útil para modelos em que as características de entrada estão em escalas muito diferentes.

Jayalakshmi et al [41] afirmam que a normalização estatística dos dados pode incrementar a capacidade de generalização de um modelo de aprendizado de máquina. Tendo um grande impacto no tratamento dos dados a serem adequados às etapas de treinamento.

Existem diferentes métodos de normalização de dados, dentre os quais se destacam [41]:

1. Normalização Estatística ou Z-Score: Esta técnica usa a média e o desvio padrão de cada característica para normalizar cada vetor de características de entrada segundo a Equação 2.6.

$$x'_i = \frac{x_i - \mu_i}{\sigma_i} \quad (2.6)$$

em que μ_i é a média dos valores da i -ésima característica e σ_i é o desvio padrão da i -ésima característica e x'_i representa o valor normalizado.

2. Normalização Min-Max: Este tipo de normalização transforma os dados, colocando-os em uma nova escala, frequentemente colocando os valores de cada característica no intervalo $[0, 1]$, representada pela Equação 2.7.

$$x'_i = \frac{x_i - \min(i)}{\max(i) - \min(i)} \quad (2.7)$$

Em que $\max(i)$ e $\min(i)$ representam o maior e menor valores da i -ésima característica, respectivamente.

3. Normalização mediana: O método mediano normaliza cada amostra pela mediana de todos os exemplos de entrada a_i . É uma normalização útil a ser usada quando houver necessidade de calcular a proporção entre duas amostras hibridizadas. A mediana não é influenciada pela magnitude dos desvios extremos, representada pela Equação 2.8.

$$x'_i = \frac{x_i}{\text{median}(a_i)} \quad (2.8)$$

4. Normalização sigmoide: A função de normalização sigmoide é usada para dimensionar as amostras no intervalo $[0, 1]$ ou $[-1, 1]$. Existem vários tipos de funções sigmóides não lineares disponíveis das quais se destaca a Equação 2.9.

$$x' = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.9)$$

Segundo [3], o uso de mecanismos de normalização, em muitas atividades de mineração de texto, tem refletido resultados piores do que o não uso da normalização (em particular utilizando o modelo TF-IDF). É importante observar e comparar os resultados, empregando ou não as funções de normalização. Em contrapartida, alguns modelos de representação de texto, como Doc2Vec, ELmo e BERT já apresentam representações vetoriais normalizadas. O BERT é descrito na Seção 3.2.

2.2 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentados conceitos gerais relacionados ao processamento de linguagem natural, tais como fluxos de representação de texto, exemplificando suas principais atividades. Também foram descritas as tarefas de NLP. Em seguida, foram apresentados alguns modelos de representação textual encontrados na literatura. Estes conceitos são importantes para o decorrer do trabalho haja vista que a construção de representações de texto coesas e semanticamente relacionáveis é premissa para a construção de modelos acurados de classificação de texto.

3 REDES NEURAIS ARTIFICIAIS

Diversas técnicas de aprendizado de máquina (do inglês, *Machine Learning*, **ML**) são utilizadas nas tarefas de classificação de texto (mais exemplos serão descritos na Seção 5.1). As redes neurais é um grupo especial de técnicas de ML baseados em otimização. Neste grupo, um problema de aprendizado é formulado como um problema de otimização em que o objetivo consiste em minimizar (ou maximizar) uma função objetivo [28].

Os modelos baseados em redes neurais artificiais (RNA) são originários da hipótese particular de que a atividade mental consiste basicamente na atividade eletroquímica em redes de células cerebrais chamadas neurônios. Inspirado por essa hipótese, alguns dos trabalhos mais antigos de IA tiveram o objetivo de criar as tais redes neurais artificiais [95]. A Figura 3.1 apresenta um modelo matemático resumido do neurônio proposto por McCulloch e Pitts [79].

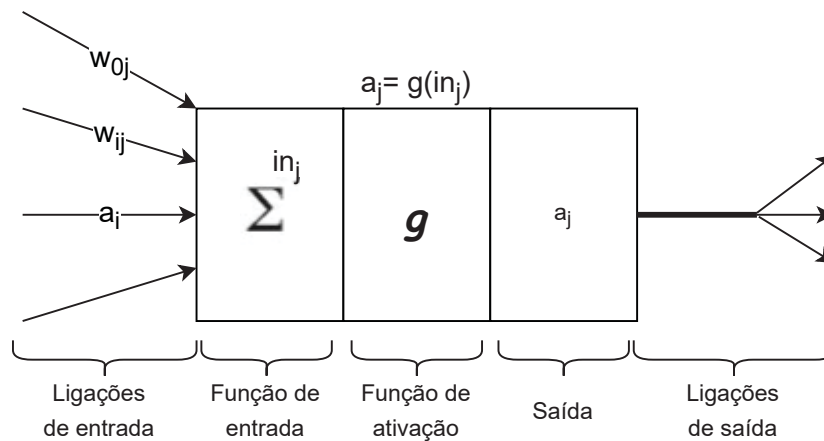


Figura 3.1: Modelo matemático simples de um neurônio. A ativação da saída da unidade $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, no qual a_i é a ativação de saída da unidade i e $w_{i,j}$ é o peso sobre a ligação da unidade i com a unidade corrente j . Adaptado de [95]

Em termos gerais, uma RNA é uma arquitetura computacional distribuída com um conjunto organizado de várias das unidades apresentadas pela Figura 3.1 conectados por ligações direcionadas. Essas unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria das arquiteturas, essas conexões possuem pesos associados ($w_{i,j}$), que ponderam a entrada recebida por cada neurônio da rede. Os pesos podem assumir valores positivos ou negativos, dependendo de o comportamento da conexão ser excitatório ou inibitório, respectivamente. Os pesos tem seus valores ajustados em um processo de aprendizado e codificam o conhecimento adquirido pela rede [28].

Desta forma, uma RNA é caracterizada por dois principais aspectos: Arquitetura e Aprendizado. Primeiramente, conforme já mencionado, a arquitetura de uma rede neurais é a disposição de um conjunto de neurônios artificiais em camadas. Estas são interligadas por um conjunto de conexões ponderadas. Os valores recebidos são ponderados e combinados por uma função matemática g . A saída a_j é a resposta do neurônio para a entrada a_i .

Quando duas ou mais camadas são utilizadas em uma RNA, um neurônio pode receber em seus terminais de entrada valores de saída de neurônios da camada anterior e/ou enviar seu valor de saída para terminais de entrada de neurônio da camada seguinte [28]. A Figura 3.2 ilustra um exemplo de RNA com quatro camadas, cada círculo verde representa um neurônio.

No exemplo, a rede recebe como entrada valores de três atributos de entrada e gera dois valores de saída.

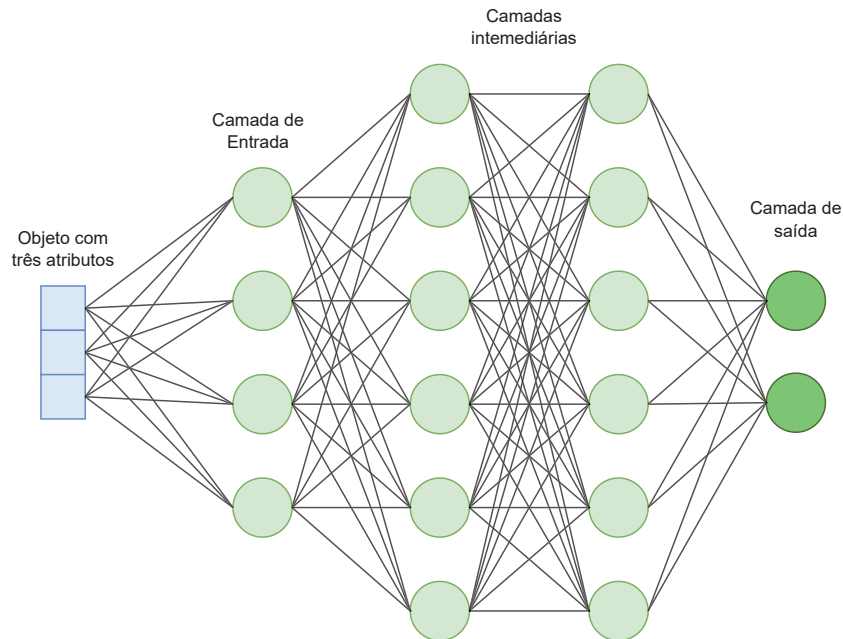


Figura 3.2: Exemplo de RNA multicamadas

O aprendizado em uma RNA se dá pelo ajuste de seus pesos. Várias tem sido as propostas relacionadas a essa área que, em geral, podem ser categorizada como segue [28]:

- Correção de erro: geralmente utilizados em aprendizado supervisionado, procuram ajustar os pesos da RNA de forma a reduzir os erros cometidos pela rede;
- Hebbiano: frequentemente usados no aprendizado não supervisionado, são baseados na regra de Hebb, que diz que, os dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada;
- Competitivo: utilizados em aprendizado não supervisionado, promovem uma competição entre neurônios para definir qual ou quais devem ter seus pesos ajustados. Os neurônios que vencem a competição, em geral, são os que respondem mais fortemente ao objeto apresentado aos seus terminais de entrada;
- Termodinâmico: algoritmos estocásticos baseados em princípios observados na metalurgia.

Destaca-se aqui o algoritmo *back-propagation*, que consiste um algoritmo de treinamento da rede baseado em correção de erro. Ele é constituído da interação de duas fases, uma fase para frente (*forward*) e uma fase para trás (*backward*). Na fase *forward*, cada objeto de entrada é apresentado à rede. Em cada camada, cada neurônio aplica a função de ativação a sua entrada total e produz um valor de saída, que é utilizado como valor de entrada pelos neurônios da camada seguinte. Esse processo continua até que os neurônios da camada de saída produzam cada um seu valor de saída, que é então comparado ao valor desejado para a saída desse neurônio. A diferença entre os valores de saída produzidos e desejados para cada neurônio da camada de saída indica o erro cometido pela rede para o objeto apresentado [28].

O valor do erro de cada neurônio da camada de saída é então utilizado na fase *backward* para ajustar os pesos de entrada. O ajuste prossegue da camada de saída até a primeira camada intermediária [28]. A Equação 3.1 demonstra como é feita a atualização dos pesos de uma rede pelo algoritmo *back-propagation*.

$$w_{jl}(t+1) = w_{jl}(t) + \eta x^j \delta_l \quad (3.1)$$

no qual w_{jl} representa o peso entre o neurônio l e o j -ésimo atributo de entrada ou a saída do j -ésimo neurônio da camada anterior, δ_l indica o erro associado ao l -ésimo neurônio e x^j indica a entrada recebida por esse neurônio (o j -ésimo atributo de entrada ou saída do j -ésimo neurônio da camada anterior) [28].

Como os valores dos erros são conhecidos apenas para os neurônios da camada de saída, o erro para neurônios das camadas intermediárias precisa ser estimado. Dessa forma, esse valor é estimado como a soma dos erros dos neurônios da camada seguinte, cujos terminais de entrada estão conectados a ele, ponderados pelo valor do peso associado a essas conexões. Esse processo é descrito pela Equação 3.2 [28].

$$\delta_l = \begin{cases} g'_a e_l, & \text{se } n_l \in C_{sai} \\ g'_a \sum w_{lk} \delta_k, & \text{se } n_l \in C_{int} \end{cases} \quad (3.2)$$

em que n_l é o l -ésimo neurônio, C_{sai} representa a camada de saída, C_{int} representa uma camada intermediária, g'_a é a derivada parcial da função de ativação g do neurônio e e_l é o erro quadrático cometido pelo neurônio de saída quando sua resposta é comparada à desejada, definido pela Equação 3.3.

$$e_l = \frac{1}{2} \sum_{q=1}^k (y_q - g_q)^2 \quad (3.3)$$

Além de definir o algoritmo de ajustes dos pesos, o *design* de uma rede neural artificial requer a definição de um conjunto de parâmetros, como tamanho do *batch*, o número de épocas, taxa de aprendizado, que determina como os ajustes de peso são calculados, impactando fortemente na qualidade do processo de aprendizado de uma rede.

3.1 REDES NEURAIIS PROFUNDAS

Tradicionalmente, uma RNA é capaz de processar dados naturais em sua forma bruta. Para alcançar resultados de qualidade expressivos, normalmente, técnicas de pré-processamento são aplicadas para gerar um conjunto de características relevantes para os dados originais, que descrevem-nos de uma forma mais discriminante, atrelando a qualidade de uma rede à qualidade deste conjunto de características.

Neste contexto, apresenta-se as Redes Neurais Profundas (do inglês, *Deep Neural Network*, **DNN**). Em termos gerais, as DNN são redes neurais artificiais mais complexas. Embora seja fácil inferir, o termo "profundo" diz respeito a uma abordagem de múltiplas camadas organizadas em profundidade. Essa abordagem foi proposta com o objetivo, entre outros, de fazer com que a própria arquitetura também fosse capaz de aprender qual seria a melhor representação para os dados. Métodos de aprendizado profundo são também métodos de geração de representação, obtidos pela composição de módulos não lineares simples, onde cada um transforma a representação em um nível (começando com a entrada bruta) em uma representação em um nível mais alto, um pouco mais abstrato.

Nas sub-seções seguintes, são descritos resumidamente as abordagens de aprendizado profundas utilizadas neste trabalho.

3.1.1 Convolutional Neural Network - CNN

Uma Rede Neural Convolucional (CNN) é um algoritmo de aprendizado profundo que pode receber uma entrada (em geral, uma imagem), atribuir importância (pesos e vieses) a vários aspectos/objetos da entrada e ser capaz de diferenciar um do outro. O pré-processamento necessário em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são projetados manualmente, com treinamento suficiente, as CNN têm a capacidade de aprender esses filtros/características.

A arquitetura de uma CNN é basicamente dividida em duas partes. Primeiramente, um conjunto de estruturas é responsável pela extração de recursos/características, a partir das entradas, geralmente utiliza camadas convolucionais e de *pooling*. Os recursos/características mais simples são capturados nas primeiras camadas da rede, enquanto características mais complexas são determinadas em camadas mais avançadas da rede. A segunda parte da rede é responsável pela classificação, onde camadas densas e *dropout* são usadas para classificar os dados, usando os recursos aprendidos anteriormente. A Figura 3.3 ilustra o funcionamento geral de uma CNN.

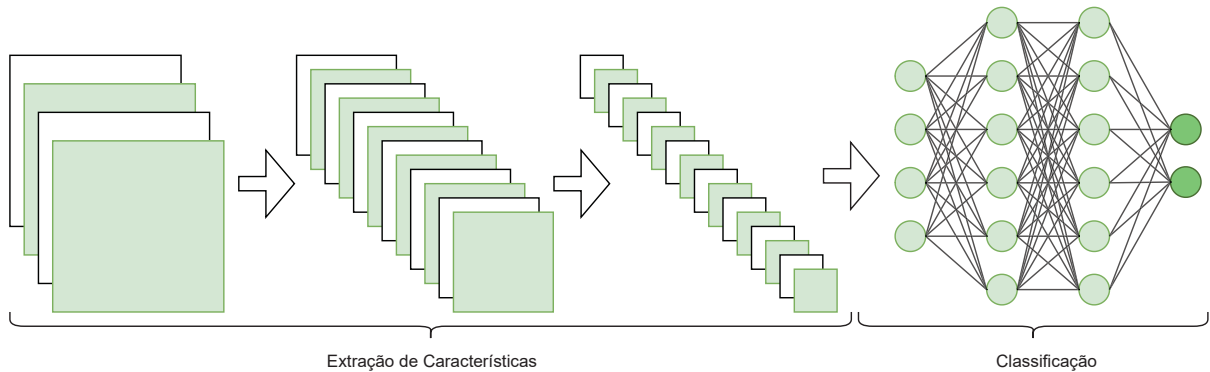


Figura 3.3: Exemplo de arquitetura genérica de uma CNN

A camada de convolução em uma CNN é a responsável pela extração das características. A Equação 3.4 apresenta a formulação matemática da convolução. Em que F é o filtro, I a matriz de entrada, M e N não o número de linhas e colunas, respectivamente.

$$F[m, n] \oplus I[m, n] = \sum_{j=-N}^N \sum_{i=-M}^M F[i, j] * I[m - i, n - j] \quad (3.4)$$

Em abordagens convolucionais, dada um objeto de entrada I com tamanho $M \times N$, um mapa de características é gerado realizando uma operação de convolução $F \oplus I$. Ele usa um filtro F , também conhecido como kernel, que é comumente representado como uma matriz de tamanho fixo e inicializada aleatoriamente. O filtro "desliza" pela entrada da esquerda para a direita, de cima para baixo. Cada elemento no mapa de características é a soma dos produtos de cada elemento do filtro e os elementos correspondentes que este filtro sobrepõe na entrada [112]. Uma representação gráfica de uma convolução é representada na Figura 3.4.

Outra estrutura importante nas redes convolucionais são as camadas de *pooling*. Semelhante à camada convolucional, esta camada é responsável por reduzir o tamanho espacial do recurso. Esta etapa é realizada para diminuir o poder computacional necessário para processar os

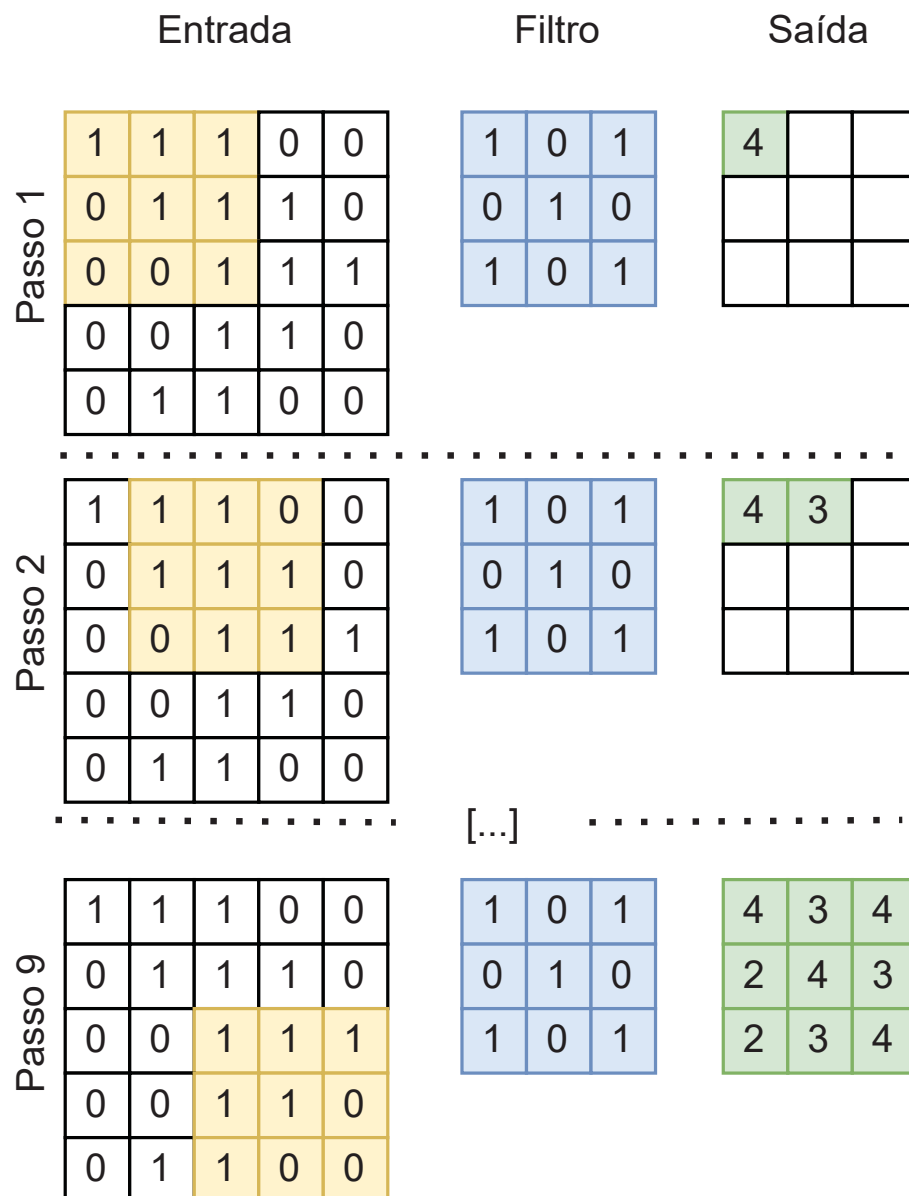


Figura 3.4: Exemplo de filtro aplicado à entrada de uma camada convolucional

dados através da redução de dimensionalidade. Além disso, é útil para extrair características dominantes que são invariantes rotacionais e posicionais, mantendo assim o processo de treinamento efetivo do modelo. Existem dois tipos de métodos de *pooling*: *Max Pooling* e *Average Pooling*. *Max Pooling* retorna o valor máximo da parte da imagem coberta pelo filtro. Por outro lado, o *Average Pooling* retorna a média de todos os valores da parte da imagem coberta pelo filtro. Uma representação gráfica dos dois métodos de *pooling* é representada na Figura 3.5.

A última etapa de uma CNN é uma camada totalmente conectada funcionando como uma rede neural comum com uma camada oculta. Primeiro, é definido o número de neurônios e o número de saídas (em geral, o número de classes do problema de classificação).

As CNN são geralmente usadas em visão computacional, mas recentemente foram aplicadas a várias tarefas de NLP e os resultados foram promissores. A principal adaptação no modelo aplicado a imagens para o modelo aplicado a textos é realizar uma representação

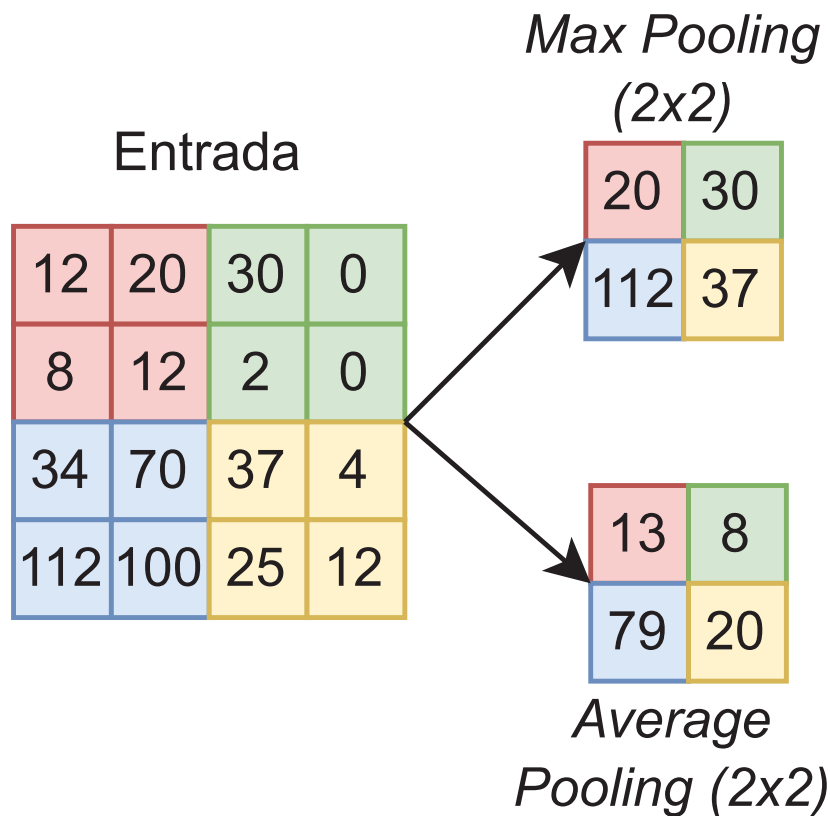


Figura 3.5: Exemplo de *max pooling* e *average pooling* a uma entrada

matricial dos textos, no qual cada palavra de uma sentença é representada por uma linha da matriz. A Figura 3.6 exemplifica essa representação.

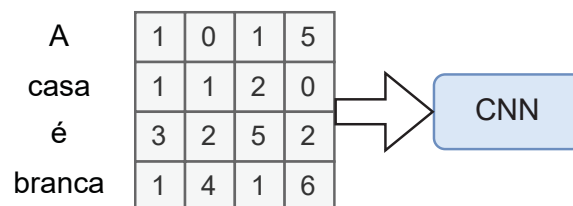


Figura 3.6: Exemplo de conversão de texto em uma representação em matriz

Alguns problemas decorrem dessa implementação, o primeiro deles é a dimensionalidade da representação. Visto que cada sentença tem um número particular de palavras, as representações poderiam variar de tamanho. Uma técnica de *padding* pode ser aplicada aqui para garantir que todas as sentenças tenham representações $M \times N$, sendo M o tamanho da maior sentença do conjunto de dados e N o tamanho da representação de uma única palavra. O segundo problema decorre da alta dimensionalidade, em geral, tem-se um matriz esparsa para representar as sentenças. Algumas soluções estão sendo validadas para esse problema, uma delas é o uso de abordagens de redução de dimensionalidade da matriz de entrada.

3.1.2 Long short-term memory - LSTM

As redes *Long Short-Term Memory* (LSTM) são um tipo de rede neural recorrente capaz de aprender a dependência de ordem em problemas de previsão de sequência. Esse é um

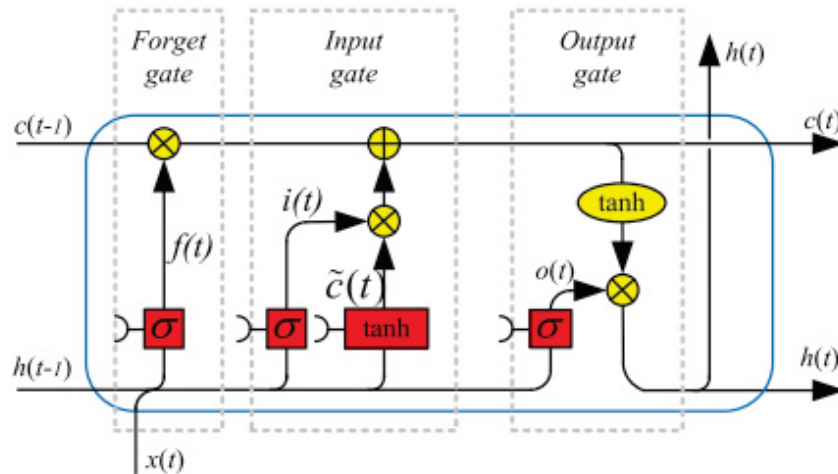


Figura 3.7: Arquitetura LSTM com *forget gate*. Fonte: [124]

comportamento necessário em domínios de problemas complexos, como tradução automática, reconhecimento de fala dentre outros. A LSTM é adequada para classificar, processar e prever séries temporais com intervalos de tempo de duração desconhecida.

As LSTM possui uma arquitetura em cadeia, composta por "células de memória". A Figura 3.7 ilustra um bloco LSTM. Cada um desses blocos possui quatro redes neurais internas (em vermelho) que auxiliam no processo de memorização do fluxo de dados.

No bloco LSTM a informação é retida pelas células e as manipulações de memória são feitas pelos portões (*gates*). Existem três portões:

- *Forget Gate*: Duas entradas: $c(t - 1)$ (estado da célula anterior), $x(t)$ (entrada no momento específico) e $h(t - 1)$ (saída de célula anterior) são alimentadas ao gate e multiplicadas por matrizes de peso, seguidas pela adição do bias. O resultante é passado por uma função de ativação que fornece uma saída binária. Se para um determinado estado de célula a saída for 0, a informação é esquecida e para a saída 1, a informação é retida para uso futuro
- *Input Gate*: responsável pela adição de memória à célula. A informação é regulada usando a função sigmoide que filtra os valores a serem lembrados de forma similar ao *forget gate* usando as entradas $h(t - 1)$ e $x(t)$. Então, um vetor é criado usando a função *tanh* que retorna um valor no intervalo $[-1, +1]$, que contém todos os valores possíveis de $h(t - 1)$ e $x(t)$. Os valores do vetor $i(t)$ e os valores regulados $\tilde{c}(t)$ são multiplicados para obter as informações úteis.
- *Output Gate*: responsável por extrair informações úteis do estado da célula atual para serem repassadas como uma saída para a célula seguinte. Um vetor é gerado aplicando a função *tanh* na célula. Então, a informação é regulada usando a função sigmoide que filtra os valores a serem lembrados usando as entradas $h(t - 1)$ e $x(t)$. Os valores do vetor $o(t)$ e os valores de estado regulados são multiplicados para serem enviados como uma saída da célula corrente e entrada para a próxima célula.

3.1.3 Graph Neural Network - GNN

As redes neurais convolucionais também podem ter sua representação modelada de acordo com um grafo. Em um modelo convolucional em grafos, a função de aprendizagem se dá pela de extração de características em um grafo $G = (V, E)$, em que V é o conjunto de nós e E é o conjunto de arestas que recebe como entrada:

- Uma descrição da característica x_i para todo nó i ; representada na matriz de características X de dimensionalidade $N \times D$ (N é o número de nós, D é o número de características dos objetos de entrada)
- Uma descrição representativa da estrutura do gráfico em forma de matriz, conhecida como matriz de adjacência.

e produz uma saída Z (uma matriz de características $N \times F$, em que F é o número de características de saída para cada nó). Além disso, toda camada da rede neural pode ser descrita por uma função não-linear demonstrada pela Equação 3.5:

$$H^{(l+1)} = f(H^{(l)}, A) \quad (3.5)$$

tal que $H^{(0)} = X$ e $H^{(L)} = Z$, L é o número de camadas. Para modelos específicos, a diferença está somente em como $f(.,.)$ é escolhida e parametrizada e A é uma matriz de adjacência simétrica [55].

Em particular, o modelo proposto por Kipf e Welling [55] é uma rede neural multicamadas que opera diretamente em um grafo e infere vetores de *embedding* de nós baseado nas propriedades de seus vizinhos. Os autores introduziram uma regra simples de propagação em camadas (Equação 3.6) para modelos de redes neurais baseado em uma aproximação de primeira ordem de convoluções espectrais em grafos [36].

$$H^{(l+1)} = \sigma(\bar{D}^{-\frac{1}{2}} \bar{A} \bar{D}^{-\frac{1}{2}} H^{(l)} W^l) \quad (3.6)$$

em que $\bar{A} = A + I_N$ é a matriz de adjacência de um grafo G não direcional com adições de auto-conexões. I_N é a matriz identidade, $\bar{D}_{ii} = \sum_j \bar{A}_{ij}$, e W^l é uma matriz de pesos treinável específica da camada l . $\sigma(.)$ representa uma função de ativação, como a ReLU [84]. H^l in $\mathbb{R}^{N \times D}$ é a matriz de ativação da l -ésima camada. Em particular, para uma rede convolucional em grafos de uma camada, a matriz de características $H^{(1)} \in \mathbb{R}^{N \times D}$ é calculada pela Equação 3.7.

$$H^1 = \sigma(\bar{D}^{-\frac{1}{2}} \bar{A} \bar{D}^{-\frac{1}{2}} X W^0) \quad (3.7)$$

É importante notar que a convolução em um GNN está intimamente ligada à matriz de adjacência (construída a partir do gráfico). Portanto, os valores das arestas presentes na matriz de adjacência podem direcionar a convolução para nichos distintos no espaço vetorial. Em especial, [55] apresentam algumas limitações de seu modelo, em especial, destacamos que sua GNN não suporta naturalmente recursos de borda e se limita a grafos não direcionados (ponderados ou não). Esta é uma limitação importante, pois em muitos casos os argumentos são conectados por relacionamentos direcionados [76].

3.1.4 Transformers

Os *Transformers* são modelos de aprendizado profundo introduzidos em [117] que utiliza o mecanismo de atenção, pesando a influência de diferentes partes dos dados de entrada. A Figura 3.8 ilustra a arquitetura geral de um *transformer*.

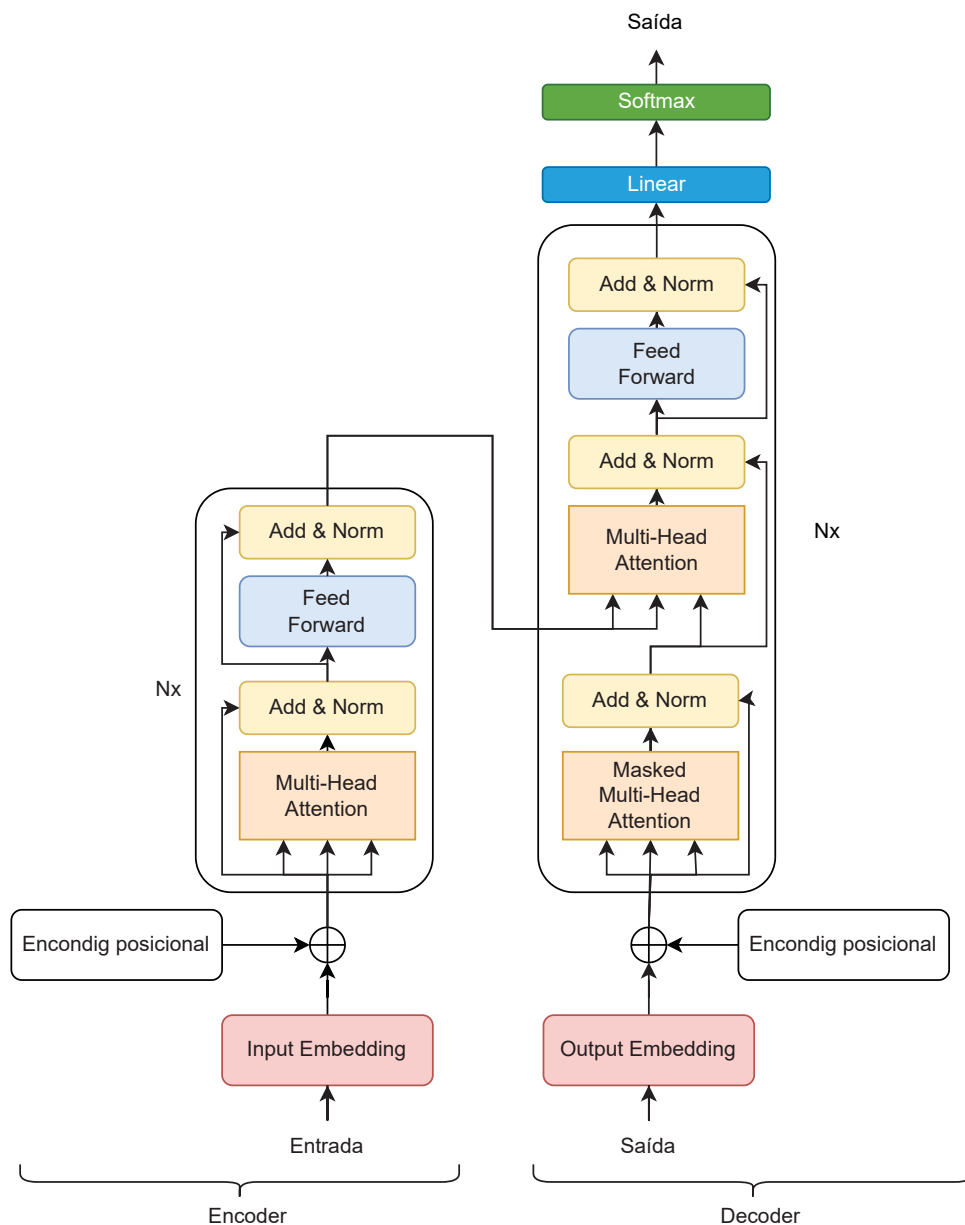


Figura 3.8: Arquitetura geral de um *transformer*

Em síntese, um *transformer* é uma pilha de *encoders* e *decoders*. Um *encoder* (lado esquerdo) é composto por uma pilha de N camadas idênticas. Cada camada tem duas subcamadas. O primeiro é um mecanismo de auto-atenção, e o segundo é uma rede feed-forward simples, totalmente conectada no sentido da posição. Em prol de uma melhor convergência, é empregado uma conexão residual em cada uma das duas subcamadas, seguidas de uma camada de normalização. Ou seja, a saída de cada subcamada é dada por $LayerNorm(x + Sublayer(x))$, em que $Sublayer(x)$ é uma função implementada pela subcamada em si. Para facilitar essas conexões residuais, todas as subcamadas do modelo, bem como a incorporação camadas, produzem saídas de dimensão constante [117].

O *decoder* também é composto por uma pilha de N camadas idênticas. Além das duas subcamadas presentes no *encoder*, o *decoder* insere uma terceira subcamada, que executa atenção sobre a saída da pilha da camada anterior. Semelhante ao *encoder*, emprega-se conexões residuais em torno de cada uma das subcamadas, seguido pela normalização da camada. Também

é modificado a subcamada de auto-atenção para evitar que as posições atendam às posições subsequentes. Cada par de *encoder* e *decoder* funciona em conjunto na arquitetura geral, visto que a saída do primeiro é a entrada da camada de atenção *multi-head* do segundo.

Por fim, a camada de atenção é um função que mapeia uma consulta e um conjunto de pares chave-valor para uma saída, em que a consulta, as chaves, os valores e a saída são todos vetores. A saída é calculada como uma soma ponderada dos valores, onde o peso atribuído a cada valor é calculado por uma função de compatibilidade da consulta com a chave correspondente.

A abordagem proposta por [117] apresenta dois tipos de mecanismos de atenção, ilustrados na Figura 3.9. O primeiro, *Scaled Dot-Product Attention*, consiste em consultas e chaves de dimensão d_k , e valores de dimensão d_v . Calcula-se os produtos escalares da matriz de consulta Q com todas as chaves K , divide cada uma por $\sqrt{d_k}$, e aplica-se uma função *softmax* para obter os pesos. Na prática, calcula-se a função de atenção em um conjunto de consultas simultaneamente, empacotadas em uma matriz Q . As chaves e os valores também são empacotados nas matrizes K e V . A saída desse processo é dada pela Equação 3.8.

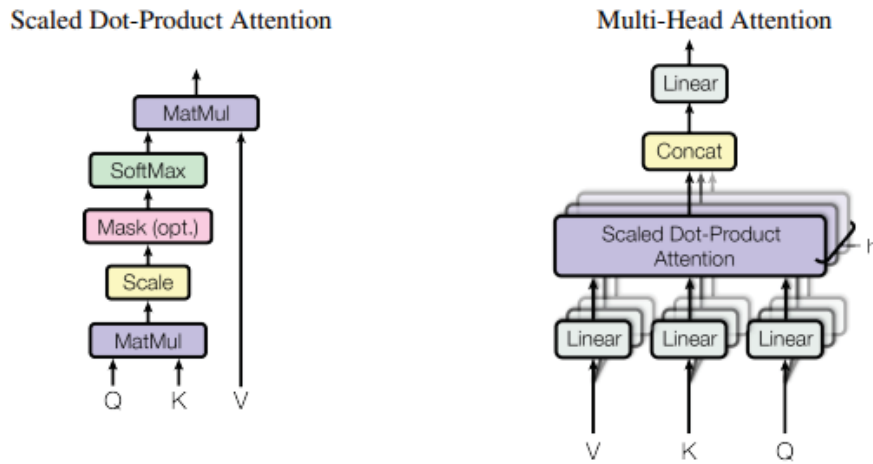


Figura 3.9: Tipos de camadas de atenção. (esquerda) *Scaled Dot-Product Attention*. (direita) *Multi-Head Attention* que consiste em múltiplas camadas de atenção executando em paralelo. Fonte: [117]

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (3.8)$$

Neste contexto, o *Multi-Head Attention* foi projetado para trabalhar linearmente com múltiplas matrizes de consultas, chaves e valores, h vezes, com diferentes projeções lineares para as dimensões d_k , d_k e d_v , respectivamente. Em cada uma dessas versões projetadas de consultas, chaves e valores, executa-se a função de atenção em paralelo, produzindo valores de saída com dimensão d_v . Estas saídas são concatenados e novamente projetadas, resultando nos valores finais.

O *Multi-Head Attention* permite que o modelo atenda conjuntamente a informações de diferentes subespaços de representações em diferentes posições. Este processo é descrito pela Equação 3.9.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \quad (3.9)$$

em que $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$, nas quais as projeções são matrizes paramétricas $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, e $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$.

Supondo a frase "A casa é branca" e sua representação vetorial $X = x_1, x_2, x_3, x_4$, a Figura 3.10 ilustra o funcionamento do mecanismo geral de atenção. Uma camada de atenção treinada associará a palavra "casa" às palavras "é" e "branca" a um peso maior do que a palavra "A".

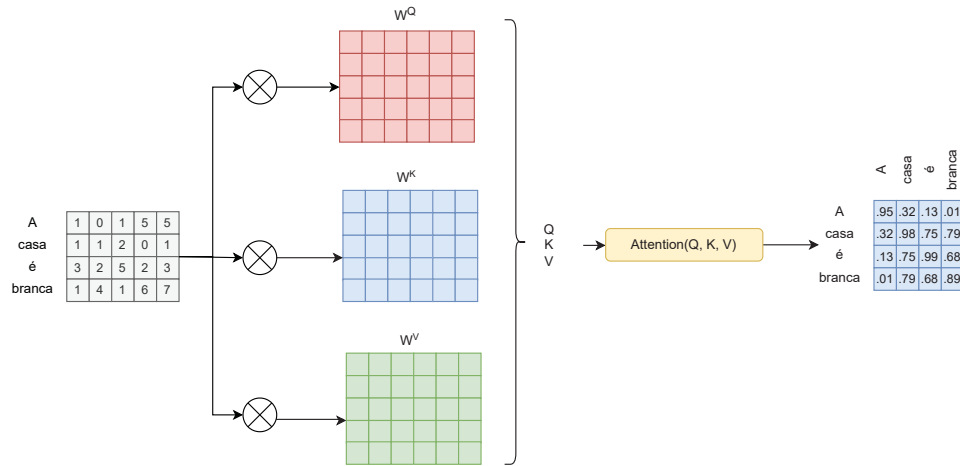


Figura 3.10: Exemplo de funcionamento do mecanismo genérico de atenção

3.2 BERT - *BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS*

O BERT, proposto por [22], ganhou destaque nas atividades de NLP utilizando um modelo pré-treinado de representação para gerar uma representação de texto coesa, alcançando índices de acurácia superiores ao estado-da-arte em tarefas de classificação de texto, em destaque para problemas de *benchmark* como **SQuAD v1.1**¹, **SQuAD c2.0**², e **SWAG** [125].

O modelo BERT recebe como entrada uma sentença e gera vetores para cada palavra da sentença levando em consideração sua posição na sentença. Por exemplo, o vetor para a palavra *coelho* nas frases *O coelho comeu a cenoura* e *Existia um grande coelho na floresta*. O vetor para a palavra *coelho* na primeira frase estará mais próximo do vetor para a palavra *cenoura*.

Em geral, o BERT constrói vetores para palavras como segue:

- O modelo possui um vocabulário de tamanho fixo de palavras e sub palavras (chamadas de *wordpiece embedding*) - qualquer palavra de entrada é mapeada para essas palavras/sub-palavras. Por exemplo, algumas palavras comuns como "A", ou mesmo incomuns como "jurisprudência", "acórdão", estão presentes no vocabulário BERT já que seu modelo foi previamente treinado em um contexto muito grande. Porém, em alguns casos uma palavra pode estar ausente e desta forma ser dividida em elementos conhecidos pelo modelo, podendo chegar ao nível de sílabas ou até caracteres isolados.
- Durante o treinamento, o BERT aprende a representação vetorial para seu vocabulário de tamanho fixo usando o que chamam de *attention heads* (que são essencialmente matrizes de pesos ponderados) e outras matrizes de transformação. O modelo treinado, juntamente com os vetores aprendidos para seu vocabulário é usado durante a avaliação/teste, para construir dinamicamente o vetor para qualquer palavra na frase, primeiro mapeando as palavras para as palavras e sub palavras do vocabulário fixo e codificando a posição de

¹<https://rajpurkar.github.io/SQuAD-explorer/explore/1.1/dev/>

²<https://rajpurkar.github.io/SQuAD-explorer/>

uma palavra na frase de entrada em sua representação. Ao contrário do uso de *gazetteers*, essa formulação permite inserir informações contextuais das outras palavras de uma sentença na representação de uma palavra em específico.

- Dada uma palavra, o modelo procura adaptar a construção dos vetores das outras palavras de uma sentença baseadas na representação da *attention heads*, e em geral, modificando sua própria representação. Por exemplo na frase: *O coelho comeu a cenoura*, tomando-se a palavra *coelho* como uma *attention heads*, a sua representação é uma soma ponderada dos vetores de todas as palavras da frase, incluindo a si mesma. Em essência, o vetor da palavra *coelho* é uma versão aproximada de todas as palavras da sentença por diferentes medidas.

Para realizar este procedimento, O BERT executa duas sub-etapas: *pre-training* e *fine-tuning*. Primeiramente o modelo é treinado com dados não rotulados sob diferentes tarefas de treinamento. Para o *fine-tuning*, o modelo BERT é inicializado com os parâmetros pré-treinados e então todos os parâmetros são ajustados usando dados rotulados previamente [22].

A primeira etapa, *Pre-training*, usa duas subetapas não-supervisionadas: *Masked LM* e *Next Sentence Prediction* [22]. Na primeira tarefa foram criadas máscaras para uma quantidade aleatória de *tokens*, então é realizado um procedimento de predição destas máscaras. Neste caso, os vetores finais desta etapa vão corresponder aos *tokens* da máscara alimentados em uma função *softmax* de saída sobre o vocabulário. A tarefa de *Next Sentence Prediction* é análoga a *Question and Answer*. Aqui o modelo foi treinado para prever uma dada sentença B, baseado em uma prévia sentença A.

A segunda etapa, *Fine-tuning*, usa diretamente o mecanismo de *Transformer* modelando elementos posteriores trocando as entradas e saídas apropriadas. Para aplicativos que envolvem pares de texto, um padrão comum é codificar independentemente os pares de texto antes de aplicar o mecanismo de *self-attention* bidirecional. Em vez disso, o BERT usa o mecanismo de *attention head* para unificar esses dois estágios, pois a codificação de um par de texto concatenado adiciona efetivamente a atenção cruzada bidirecional entre duas frases [22].

A Figura 3.11 representa os procedimentos gerais de pré-treinamento e ajuste fino para o BERT. Para o modelo, é criado um *token* especial chamado [CLS], adicionado no começo de cada exemplo de entrada e outro *token* [SEP] separador (por exemplo, separando perguntas / respostas). Esses *tokens* são treinados em conjunto com a representação das palavras e segundo os autores, o *token* [CLS] pode ser utilizado para representar a sentença completa.

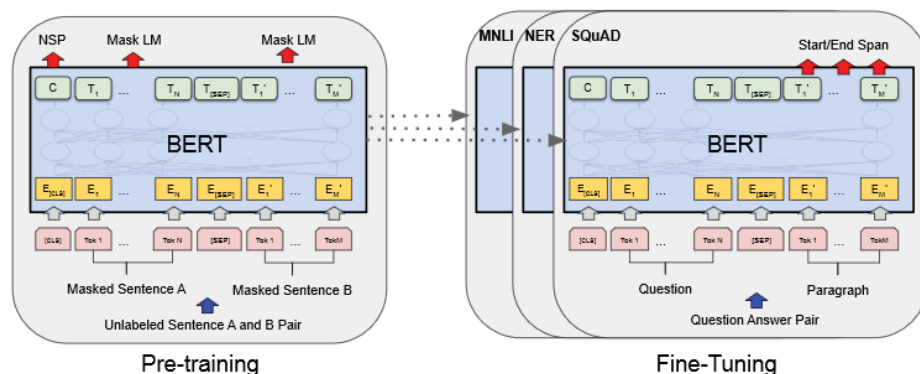


Figura 3.11: Representação gráfica da arquitetura do modelo BERT. Fonte: [22]

A Figura 3.12 exemplifica a representação de uma sentença utilizando o BERT. Primeiramente, um token [CLS] é inserido no início da frase e um token [SEP] é inserido no final de

cada frase. Em seguida, para cada frase é gerado um *embedding* indicador (no exemplo E_A e E_B). Finalmente, uma incorporação posicional é adicionada a cada *token* para indicar sua posição na sequência.

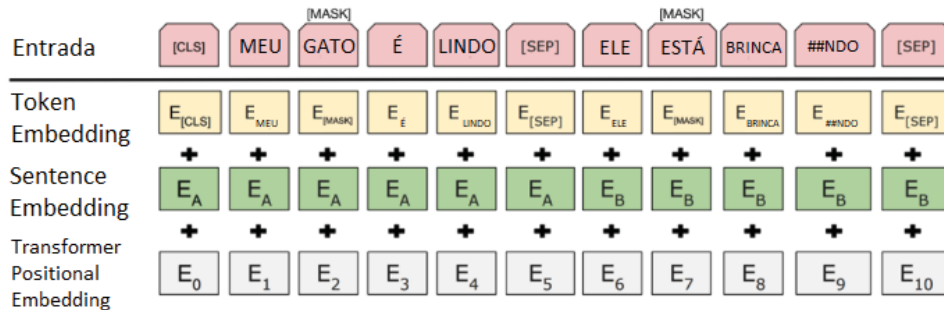


Figura 3.12: Representação de sentenças utilizando BERT. Adaptado de [22]

A principal diferença do BERT ao *Word2vec* é que o último não leva em consideração a ordem das palavras em seu treinamento. Uma implicação prática dessa diferença é que se podem usar vetores *word2vec* treinados em um *dataset* grande diretamente para tarefas posteriores. Tudo de que se precisa são os vetores das palavras. Não há necessidade do próprio modelo que foi usado para treinar esses vetores. No entanto, no caso do BERT, por serem extremamente dependentes do contexto, precisa-se do modelo que foi usado para treinar os vetores mesmo após o treinamento, pois eles geram os vetores para uma palavra com base no contexto e posição.

A arquitetura do BERT segue [117] e foi implementada utilizando Tensorflow [2]. O trabalho reportou dois modelos pré-treinados chamados de **BERT_{BASE}** (com 12 camadas, 768 unidades e 12 unidades de intra-atenção) e **BERT_{LARGE}** (com 24 camadas, 1024 unidades e 16 unidades de intra-atenção), totalizando 110 mil e 340 mil parâmetros respectivamente.

A Tabela 3.1 compara a aplicação dos dois modelos propostos em [22] com outras técnicas já consolidadas. É possível observar uma melhora estatística em relação a todos os outros modelos quando da utilização do BERT.

Tabela 3.1: Comparativo do modelo BERT com outras técnicas aplicadas a problemas de *benchmark*

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

3.2.1 Sentence-BERT

O BERT estabeleceu novos marcos em várias tarefas de classificação de texto. Como saída principal do BERT têm-se vetores atribuídos a cada token identificado. Um método comum para contemplar a representação de uma sentença é mapeá-la para um espaço vetorial de modo que sentenças semanticamente semelhantes estejam próximas. Os pesquisadores começaram a utilizar sentenças individuais no BERT e a derivar *embeddings* de sentenças de tamanho fixo, para, então, usar modelos de classificação mais comuns. O usual nessa abordagem é calcular a média dos vetores de cada token da camada de saída do BERT ou usar a representação do primeiro *token* ([CLS]) como representação vetorial da sentença, a Figura 3.13 exemplifica essa

abordagem. Segundo [93], esta prática comum produz resultados bastante ruins, muitas vezes piores do que a média de *embeddings* utilizando o Word2Vec [81], por exemplo.

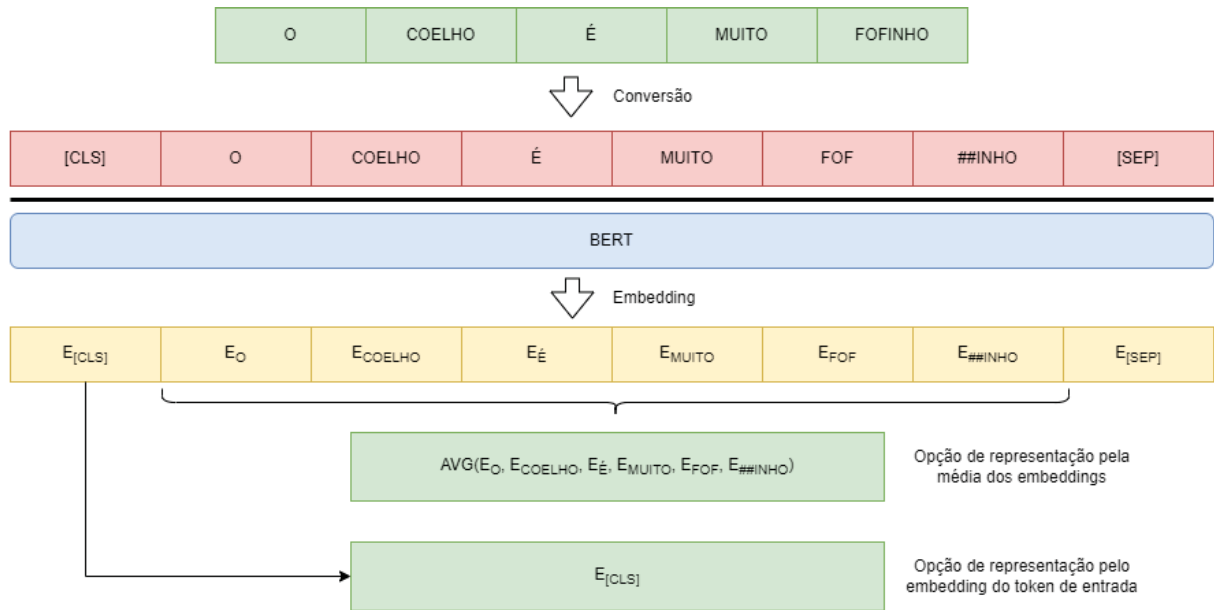


Figura 3.13: Métodos de representação de sentenças utilizando a média das incorporações (*embeddings*) ou o *token* de entrada [CLS].

Para aliviar esse problema, [93] desenvolveu o Sentence-BERT. Essa abordagem utiliza uma arquitetura de rede siamesa que permite que vetores de tamanho fixo para sentenças de entrada possam ser derivados. Usando uma medida de similaridade como cosseno ou distância Manhattan/Euclidiana, sentenças semanticamente semelhantes podem ser encontradas. Essas medidas de similaridade podem se mostrar extremamente eficientes em *hardware* moderno, permitindo que o *framework* seja usado para pesquisa de similaridade semântica tanto em modelos de classificação quanto de agrupamento.

O *Sentence-BERT* adiciona uma camada de *pooling* a suas arquiteturas base (BERT, RoBERT, XLM-RoBERTa) permitindo que sejam geradas representações vetoriais de tamanho fixo. Para sua construção foram experimentadas três estratégias de *pooling*: usando a saída do *token* CLS, calcular a média de todos os vetores de saída (*MEAN strategy*) e calcular um vetor máximo entre vetores de saída (*MAX strategy*). A configuração padrão do *framework* é a *MEAN strategy*.

Para ajustar o modelo, foram criadas redes siamesas³ e redes *triplet*⁴ para atualizar os pesos de forma que os *embedding* de sentenças que são semanticamente significativas possam ser comparadas por similaridade cosseno. Segundo [93], o *Sentence-BERT* é computacionalmente eficiente. Em uma GPU, é cerca de 9% mais rápido que o InferSent [20] e cerca de 55% mais rápido que o Universal Sentence Encoder [2]. O *Sentence-BERT* pode ser usado para tarefas que não são computacionalmente viáveis de ser modeladas com o BERT. Por exemplo, o agrupamento de 10.000 frases com agrupamento hierárquico requer com o BERT cerca de 65 horas, pois cerca de 50 milhões de combinações de frases devem ser computadas. Com o Sentence-BERT, é possível reduzir o esforço para cerca de 5 segundos [93].

³As arquiteturas siamesas consistem em duas ou mais sub-redes idênticas usadas para gerar vetores de características para cada entrada e realizam comparação entre eles

⁴As arquiteturas *triplets* consistem em várias redes paralelamente treinadas que compartilham pesos entre si.

3.3 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentados conceitos gerais relacionados as redes neurais artificiais. Em seguida o conceito de redes neurais profundas. Os conceitos relacionados a *transformers* foram aprofundados para que um elo entre a fundamentação teórica de redes neurais apresentadas e o *framework* BERT fosse criado. Por fim, foi apresentado o *Bidirecional Encoder Representations From Transformers* (BERT) como uma importante ferramenta de representação de texto. Estes conceitos são elementos básicos no decorrer do trabalho.

4 PROJETO DE REDES NEURAIS USANDO EVOLUÇÃO GRAMATICAL

Neste capítulo são descritos os principais aspectos relacionados à evolução gramatical, à arquitetura de redes neurais utilizando uma abordagem baseada em gramática. Em seguida o DSGE é apresentado como uma proposta de evolução gramatical dinâmica capaz de construir arquiteturas de redes neurais artificiais mistas e multicamadas.

4.1 DESENVOLVIMENTO DE ALGORITMOS COM COMPUTAÇÃO EVOLUTIVA

Na ciência da computação, os pesquisadores se esforçam muito para imitar o mecanismo de evolução para desenvolver algoritmos poderosos para problemas de adaptação e otimização, criando o que hoje é chamado de Computação Evolutiva (do inglês, *Evolutionary Computation*, EC) [6]. Em termos técnicos, EC é um sub campo da inteligência artificial, que estuda os solucionadores de problemas de tentativa e erro baseados na população de soluções com um caráter de otimização meta heurístico ou estocástica.

Em termos gerais, uma aplicação de otimização requer encontrar uma configuração $\vec{x} = (x_1, x_2, \dots, x_n) \in M$ de parâmetros, de modo que um determinado critério de qualidade $f : M \rightarrow \mathbb{R}$ (comumente chamada de função objetivo) é maximizado (ou, equivalentemente, minimizada) [73]. As técnicas de EC podem produzir soluções altamente otimizadas em uma ampla gama de configurações de problemas, tornando-as populares na ciência da computação.

Neste contexto encontram-se os algoritmos evolutivos (do inglês, *Evolutionary Algorithms*, EA). Os EA são um conjunto de algoritmos fundamentados no campo da EC. Nestes algoritmos, um grupo inicial de soluções candidatas é gerado e sequencialmente alterado, geração após geração. Em cada nova geração, são produzidas estocasticamente novas soluções e removidas soluções menos aptas. Esse processo é inspirado na evolução natural das espécies, nele os indivíduos mais aptos têm maior chance de transmitir seus genes para as gerações futuras. Alguns conceitos estão intimamente relacionados com esse processo, tais como seleção, reprodução (ou cruzamento) e mutação. O resultado esperado é que a população de indivíduos (ou soluções) seja gradualmente evoluída para um conjunto mais apto ao ambiente (ou problema) de acordo com alguma métrica de aptidão (ou *fitness*).

Técnicas de EC podem produzir soluções altamente otimizadas em uma ampla gama de configurações de problemas, tornando-as populares na ciência da computação. Existem muitas variantes e extensões desta abordagem, adequadas para famílias de problemas e estruturas de dados mais específicas. A maioria dos EAs pode ser dividida em algoritmos geracionais, que atualizam a amostra inteira uma vez por iteração, e algoritmos de estado estacionário, que atualizam algumas soluções da amostra por vez. EAs comuns incluem o Algoritmo Genético (GA) e Estratégias de Evolução (ES); e existem versões geracionais e de estado estacionário para cada uma desses EAs [73]. Os pseudocódigos de GA e ES são mostrados respectivamente nos Algoritmos 1 e 2.

O Algoritmo 1 apresenta o pseudocódigo básico para GA. O parâmetro de entrada *popsiz*e define o tamanho da população desejada. Primeiramente, a população é gerada aleatoriamente (linha 3-5). O algoritmo inicia quando, para cada indivíduo no conjunto de soluções, é executada sua avaliação de aptidão (*AvaliarFitness* - linha 8) mantendo o controle do melhor indivíduo encontrado até o momento. A recombinação começa com uma população vazia de filhos (*Q*), então um operador de seleção (*SelecaoComSubstituicao* - linha 13 e 14) escolherá dois pais da população original. Em seguida, esses pais são cruzados por meio de

um operador *Cruzamento* (linha 15), criando dois novos filhos. Posteriormente, um operador *Mutação* é aplicado (linha 16), provocando pequenas alterações nesses filhos. Este processo é repetido até que a população filha seja preenchida.

Como a população de filhos tem o mesmo número de soluções que a população original, os passos para criar os filhos são repetidos pela metade do tamanho, pois cada iteração cria dois novos indivíduos (linhas 12-16). O algoritmo repetirá as etapas de criar, recombinar e substituir as soluções na população por um número de iterações, chamadas de gerações, e salvar a melhor solução encontrada até o momento, de acordo com o valor de *Fitness*. O algoritmo finalmente termina ao atingir um critério de parada, por exemplo, encontrando a solução ideal ou executando por um número paramétrico de gerações.

Algoritmo 1 Algoritmo Genético

```

1:  $popsiz$  ← tamanho da população desejada
2:  $P \leftarrow \{\}$ 
3: for  $popsiz$  vezes do
4:    $P \leftarrow P \cup \{\text{nova solução aleatória}\}$ 
5:  $Melhor \leftarrow \square$ 
6: repeat
7:   for cada indivíduo  $P_i \in P$  do
8:     AvaliarFitness( $P_i$ )
9:     if  $Melhor = \square$  or  $Fitness(P_i) > Fitness(Melhor)$  then
10:       $Melhor \leftarrow P_i$ 
11:    $Q \leftarrow \{\}$ 
12:   for  $popsiz/2$  times do
13:     Pai  $P_a \leftarrow SelecaoComSubstituicao(P)$ 
14:     Pai  $P_b \leftarrow SelecaoComSubstituicao(P)$ 
15:     Filhos  $C_a, C_b \leftarrow Cruzamento(Copia(P_a), Copia(P_b))$ 
16:      $Q \leftarrow Q \cup \{Mutacao(C_a), Mutacao(C_b)\}$ 
17:    $P \leftarrow Q$ 
18: until  $Melhor$  ser a solução ideal ou o número de rodadas foi atingido
19: return  $Melhor$ 

```

O Algoritmo 2 apresenta o pseudocódigo básico para ES. O processo começa com uma população de λ indivíduos, gerados aleatoriamente (linhas 4 e 5). Em seguida, os seguintes passos são repetidos. Primeiro, a aptidão para cada solução é calculada (linhas 8-11), mantendo o controle do melhor indivíduo encontrado até o momento. Cada um dos indivíduos μ mais aptos consegue produzir filhos λ/μ por meio de mutação (linha 16). Então, todos os filhos substituem os pais. Essas etapas são repetidas até que a solução ideal seja encontrada ou outro critério de parada seja atendido, por exemplo, o tempo de execução. Observe que λ deve ser um múltiplo de μ . Por exemplo, se $\mu = 5$ e $\lambda = 20$.

4.1.1 Codificação

A representação dos dados para uma abordagem evolutiva não segue uma regra específica. A representação geralmente varia de acordo com o problema ou, em alguns casos, de acordo com o algoritmo. Por exemplo, as representações mais comuns são listas de inteiros e/ou pontos flutuantes e, em alguns casos, árvores ou qualquer outra estrutura de dados. As representações binárias são utilizadas quando a presença ou ausência de um atributo deve ser ponderada.

Algoritmo 2 Estratégias de Evolução ($\mu + \lambda$)

```

1:  $\mu \leftarrow$  número de pais selecionados
2:  $\lambda \leftarrow$  número de filhos gerados a partir dos pais
3:  $P \leftarrow \{\}$ 
4: for  $\lambda$  vezes do
5:    $P \leftarrow P \cup \{\text{nova solução aleatória}\}$ 
6:  $Melhor \leftarrow \square$ 
7: repeat
8:   for cada indivíduo  $P_i \in P$  do
9:     AvaliarFitness( $P_i$ )
10:    if  $Melhor = \square$  or  $\text{Fitness}(P_i) > \text{Fitness}(Melhor)$  then
11:       $Melhor \leftarrow P_i$ 
12:   $Q \leftarrow$  os  $\mu$  indivíduos em  $P$  com maior fitness
13:   $P \leftarrow Q$ 
14:  for cada indivíduo  $Q_j \in Q$  do
15:    for  $\lambda/\mu$  vezes do
16:       $P \leftarrow P \cup \{\text{Mutacao(Copia}(Q_j))\}$ 
17:   $P \leftarrow Q$ 
18: until  $Melhor$  ser a solução ideal ou o número de rodadas chegou ao fim
19: return  $Melhor$ 

```

4.1.2 Seleção

O processo de seleção é usado para obter soluções do conjunto populacional e deve ser usado no processo de cruzamento e mutação. Nos algoritmos ES, todas as μ melhores soluções são usadas.

Em termos gerais, os processos de seleção para o algoritmo GA são: Seleção Aleatória (quando um indivíduo é selecionado aleatoriamente do conjunto populacional); Seleção por Roleta (quando o indivíduo é selecionado como chance proporcional ao seu *fitness*) e Seleção por Torneio (quando dois ou mais indivíduos são selecionados aleatoriamente do conjunto populacional e uma 'disputa' entre eles é realizada, e o mais apto é selecionado).

4.1.3 Cruzamento

O operador de cruzamento é responsável por "mesclar" duas soluções em outro conjunto de soluções. Esta mescla irá depender principalmente de como as soluções são representadas. Uma forma simples de realizar esse processo é utilizando o método *One-Point Crossover* (Figura 4.1). Nesta abordagem, dados dois vetores de tamanho n , um ponto p é selecionado entre 1 e $n - 1$. Todos os valores que estão entre as posições 0 e p de um dos pais e $p + 1$ e n do outro pai são concatenados em um novo indivíduo. Os pontos entre 0 e p do segundo pai e $p + 1$ e n do primeiro pai são concatenados no segundo novo indivíduo.

4.1.4 Mutação

A mutação é o operador responsável por injetar variabilidade a uma população, responsável principalmente por retirar o algoritmo de platôs de ótimos locais. Esse operador modifica uma solução filho e promove a chance de gerar soluções que não seriam possíveis com o cruzamento. Uma forma simples de realizar uma mutação em uma representação é a modificação sensível de

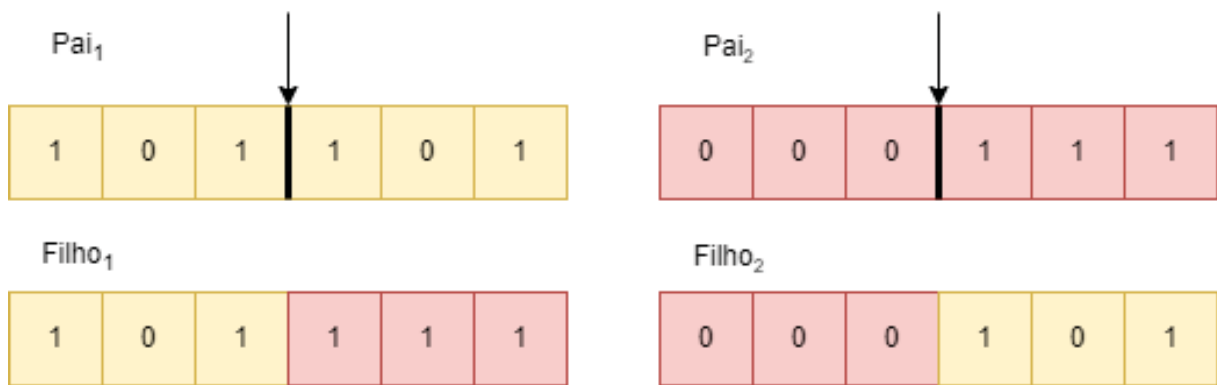


Figura 4.1: Exemplo de Cruzamento *One-point*

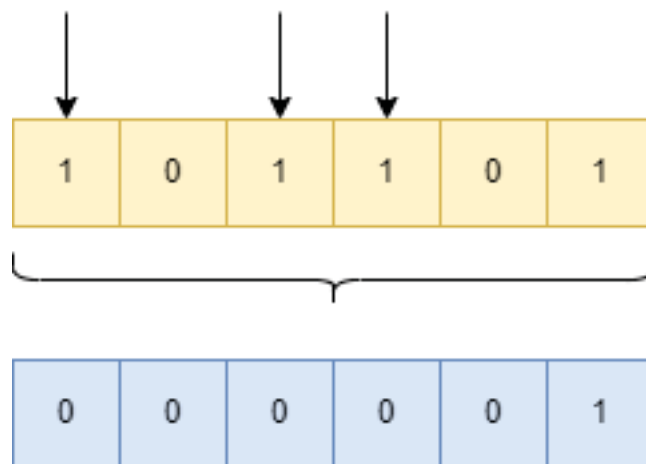


Figura 4.2: Exemplo de mutação uniforme, no qual, probabilisticamente alguns atributos são modificados.

seu valor. Em representações inteiras, bastaria adicionar ou remover uma unidade ao atributo. Em representações booleanas, isso seria possível pela inversão do valor booleano. A Figura 4.2 demonstra um exemplo de mutação.

4.2 EVOLUÇÃO GRAMATICAL

Nos últimos anos os EAs têm sido usados com sucesso em diferentes situações, incluindo otimização, arquitetura ou problemas de aprendizagem [72]. Programação Genética (do inglês, *Genetic Programming*, **GP**) é um ramo da EA que trata da evolução automática de programas ou algoritmos de computador. Uma das variantes mais relevantes do GP é evolução gramatical (do inglês, *Gramatical Evolution*, **GE**) [72].

GE é uma abordagem evolutiva que pode desenvolver programas usando uma linguagem de programação arbitrária [96]. A GE e suas variantes são definidas principalmente por três componentes. No contexto de redes neurais, primeiramente, tem-se o componente 'gramática', na qual são definidas as camadas, parâmetros e regras de estrutura usadas para descrever o espaço de arquiteturas possíveis. Por exemplo, pode-se incluir camadas LSTM [113], Recurrent [33] e Bidirecional [100]. A gramática também pode incluir regras que ajudam a construir a estrutura da rede e podem ser posteriormente complementadas pelo processo de mapeamento.

Em seguida, o mapeamento analisa uma solução codificada, usando a gramática, em uma lista de componentes que descrevem uma rede. Além disso, construir uma rede inclui

lidar com diferentes topologias, camadas e estratégias para conectar as camadas específicas de cada problema. Finalmente, o mecanismo de busca, que geralmente é um algoritmo evolutivo, desenvolve uma população de soluções geradas, selecionadas e recombinadas aleatoriamente até que um critério de parada seja atendido e, em seguida, retornando a melhor solução encontrada. A abordagem geral e seus componentes são mostrados na Figura 4.3.

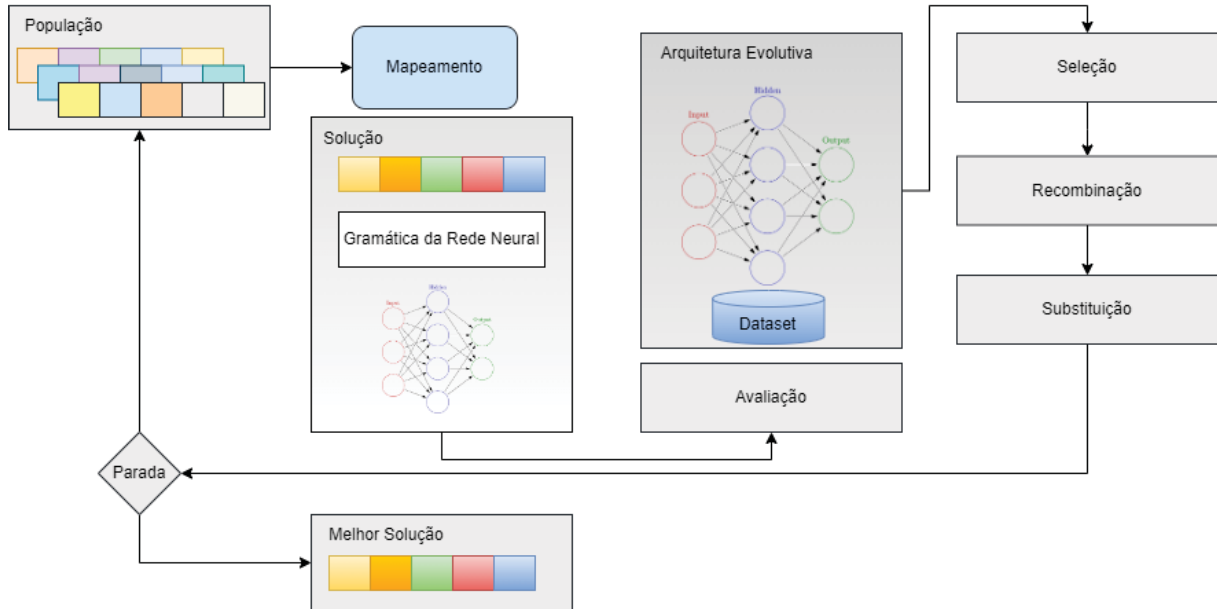


Figura 4.3: Esquema geral da abordagem proposta. O fluxo básico segue a estrutura tradicional do GA com a adição de uma etapa de mapeamento, necessária para construir a rede a partir da representação de codificação.

O algoritmo Evolução de Gramática Estruturada (do inglês, *Structured Grammar Evolution*, **SGE**) [72] propõe uma representação genotípica para GE. Esta representação visa resolver os problemas de redundância e localidade originários na GE tradicional e consiste em uma lista de genes, um para cada símbolo não-terminal. Além disso, um gene é uma lista de inteiros do tamanho do número máximo de expansões possíveis para o não-terminal que o codifica; cada inteiro é um valor no intervalo $[0, NT - 1]$, NT é o número de possibilidades para a expansão do símbolo não-terminal considerado [5].

Portanto, o SGE requer uma etapa de pré-processamento que deve determinar o número máximo de expansões para cada elemento não-terminal. O resultado direto da representação SGE é que, por um lado, a decodificação de indivíduos não depende execução do módulo de arquitetura em si e, portanto, sua redundância é removida; por outro lado, a localidade é reforçada, como os símbolos terminais estão associados diretamente a símbolos não-terminais específicos [5].

O DSGE [70, 5], em especial, foi apresentado como uma versão melhorada do GE tradicional [96] e do SGE [72]. Esta abordagem propõe uma codificação indireta para soluções, afetando como a gramática e o mapeamento interagem para construir os programas, e é mais eficiente durante o processo de busca. Mais detalhes sobre o DSGE na Seção 4.3.

4.3 EVOLUÇÃO GRAMATICAL DINAMICAMENTE ESTRUTURADA

O algoritmo de Evolução Gramatical Dinamicamente Estruturada (do inglês, *Dynamic structured grammatical evolution*, **DSGE**) foi primeiramente apresentado por [70]. Os autores propunham uma abordagem em que: (i) todo o genótipo é usado, ou seja, enquanto em GE o

genótipo codifica a maior sequência permitida, em DSGE o genótipo cresce quando necessário; e (ii) não há necessidade de pré-processar a gramática para calcular os tamanhos máximos de árvore de cada símbolo não-terminal, para que sejam criadas regras de derivação gramaticais intermediárias [70]. Nas próximas subseções são descritos os detalhes e componentes do DSGE e sua aplicação a arquitetura de redes neurais profundas.

4.3.1 Gramáticas para arquiteturas de Redes Neurais Profundas

Uma gramática é composta por um conjunto de símbolos não-terminais N , um conjunto de símbolos terminais T , uma regra de início S , que é o um membro de N , e um conjunto de produções P na forma $A ::= \alpha$, com $A \in N$ e $\alpha \in (N \cup T)^*$, N e T são conjuntos disjuntos [71]. Na arquitetura de DNNs, a gramática contém principalmente definições de camadas, parâmetros e estrutura de construção de blocos. Na Figura 4.4, por exemplo, a regra $\langle \text{conv} \rangle$ inclui os parâmetros $\langle \text{features} \rangle$ e $\langle \text{kernel} \rangle$, cada um com seus possíveis valores. Além disso, $\langle \text{start} \rangle$ é definida como uma regra de início com quatro produções: $\langle \text{conv} \rangle$, $\langle \text{pool} \rangle$, $\langle \text{dense} \rangle$, e “&” (produção vazia). Uma produção como $\langle \text{conv} \rangle \langle \text{start} \rangle$ adiciona a regra $\langle \text{conv} \rangle$ à expansão e também outra instância de $\langle \text{start} \rangle$, que é uma forma comum de uso recursivo das regras para expandir a arquitetura.

$$\langle \text{start} \rangle ::= \langle \text{conv} \rangle \langle \text{start} \rangle \mid \langle \text{pool} \rangle \langle \text{start} \rangle \mid \langle \text{dense} \rangle \langle \text{start} \rangle \mid \&$$

$$\langle \text{conv} \rangle ::= \text{Convolutional } \langle \text{features} \rangle \langle \text{kernel} \rangle$$

$$\langle \text{pool} \rangle ::= \text{Maxpool} \mid \text{Avgpool}$$

$$\langle \text{dense} \rangle ::= \text{Dense } \langle \text{units} \rangle$$

$$\langle \text{features} \rangle ::= 16 \mid 32 \mid 64 \mid 128 \mid 256 \mid 512$$

$$\langle \text{kernel} \rangle ::= 2 \mid 3 \mid 5 \mid 7$$

$$\langle \text{units} \rangle ::= 16 \mid 32 \mid 64 \mid 128 \mid 256 \mid 512 \mid 1024$$

Figura 4.4: Exemplo de uma gramática simplificada para projeto de DNN. Ela define o uso de camadas convolucionais com dois parâmetros (*feature* e tamanho do *kernel*), *pool* máximo ou médio usando os parâmetros padrões e camadas densas com um parâmetro (quantidade de unidades).

4.3.1.1 Codificação

As soluções com DSGE são codificadas como uma relação de listas, e cada lista interna é um conjunto de valores relacionados ao símbolo não-terminal da gramática (Figura 4.5). O tamanho da solução é definido pelo número de símbolos não terminais da gramática. Além disso, cada lista interna tem o número exato de valores para construir um mapeamento completo; estes valores estão entre 0 e o número de opções daquela regra.

4.3.1.2 Inicialização

O Algoritmo 3 é responsável pela inicialização das soluções aleatórias para o DSGE [65]. São necessários cinco parâmetros: a gramática, uma profundidade máxima, o genótipo (inicialmente vazio), um símbolo não-terminal (inicialmente o símbolo do axioma) e a profundidade

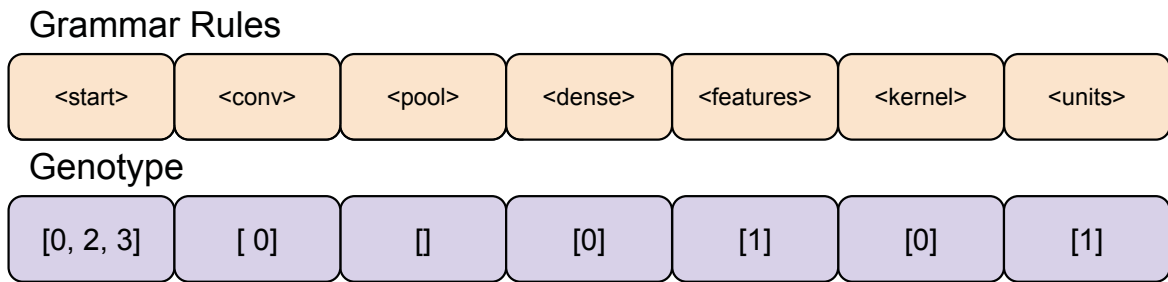


Figura 4.5: Exemplo de genótipo usando a codificação DSGE. Cada lista de inteiros no genótipo está relacionada a uma regra (não-terminal) da gramática.

atual (inicializada como 0). O algoritmo pega o símbolo atual e seleciona uma das possíveis expansões (produção) para ele, neste contexto de inicialização, significa o valor inteiro que representaria aquela expansão (linha 2). Em seguida, verifica se este símbolo é recursivo e se o estado atual do procedimento está respeitando a profundidade máxima, caso contrário, seleciona uma nova expansão sem resultado recursivo (linhas 3-7). A seguir, a expansão é adicionada ao genótipo na sublista que representa o símbolo atual (linhas 8-11). Em seguida, pega os símbolos produzidos por essa expansão e chama esse procedimento recursivamente para cada símbolo não-terminal (linhas 12-15).

Algoritmo 3 Geração de soluções recursivas - DSGE

```

1: procedure CRIAR_SOLUCAO(gramatica, profundidade_maxima, genotipo, simbolo, pro-
   fundidade)
2:   expansao = randint(0, len(gramatica[simbolo])-1)
3:   if e_recursivo(simbolo) then
4:     if expansao in gramatica.recursive(simbolo) then
5:       if profundidade  $\geq$  profundidade then
6:         non_rec = gramatica.non_recursive(simbolo)
7:         expansao = escolha(non_rec)
8:   if simbolo in genotipo then
9:     genotipo[simbolo].append(expansao)
10:  else
11:    genotipo[simbolo] = [expansao]
12:  simbolos_expandidos = gramatica[simbolo][expansao]
13:  for simb in simbolos_expandidos do
14:    if not e_terminal(simb) then
15:      criar_solucao(gramatica, profundidade_maxima, genotipo, simb, profundidade+1)

```

4.3.1.3 Mapeamento

O processo de mapeamento gerencia a tradução da solução codificada, (lista de listas), em uma DNN real. Iniciado do axioma, ele expande a gramática, substituindo o elemento não-terminal mais à esquerda por sua regra de produção. Essa regra escolhida para a substituição é selecionada da lista interna relacionada ao símbolo não-terminal em avaliação. Por exemplo, se o terminal mais à esquerda é <features>, como está na Figura 4.6, tem-se que as opções são entre 16 e 512, divididas em seis opções, então os valores possíveis para a solução irão variar de

0 a 5. A Figura 4.6 apresenta o mapeamento de uma possível solução gerada pela gramática apresentada na Figura 4.4. Seguindo as etapas de mapeamento, o axioma é representado pela regra <start>. A sub lista relacionada a esta regra é [0, 2, 3], e o primeiro valor substituído é o 0, substituindo <start> por <conv> <start>. Este processo é repetido até que todos os valores não-terminais sejam substituídos por valores terminais. O Algoritmo 4 é uma adaptação de [65].

Algoritmo 4 Mapeamento Recursivo de Soluções - DSGE

```

1: procedure MAPEAMENTO(genotipo, gramatica, profundidade_maxima, inteiros_lidos,
   simbolo, profundidade)
2:   fenotipo = “ ”
3:   if simbolo not in inteiros_lidos then
4:     inteiros_lidos[simbolo] = 0
5:   if inteiros_lidos not in genotipo then
6:     genotipo[simbolo] = []
7:   if inteiros_lidos[simbolo] ≥ len(genotipo[simbolo]) then
8:     if profundidade ≥ profundidade_maxima then
9:       gerar_expansao_terminal(genotipo, simbolo)
10:    else
11:      gerar_expansao(genotipo, simbolo)
12:    gen_int = genotipo[simbolo][inteiros_lidos[simbolo]]
13:    expansao = gramatica[simbolo][gen_int]
14:    inteiros_lidos[simbolo] += 1
15:    for simb in expansao do
16:      if e_terminal(simb) then
17:        fenotipo += simb
18:      else
19:        fenotipo += mapeamento(genotipo, gramatica, profundidade_maxima, inteiros_
          lidos, simb, profundidade+1)
20:    return fenotipo

```

Etapa de Derivação	Genótipo
<start>	[[0, 2, 3] [0] [] [0] [1] [0] [1]]
<conv><start>	[[2, 3] [0] [] [0] [1] [0] [1]]
Conv <features><kernel><start>	[[2, 3] [] [] [0] [1] [0] [1]]
Conv 32 <kernel><start>	[[2, 3] [] [] [0] [] [0] [1]]
Conv 32 2 <start>	[[2, 3] [] [] [0] [] [] [1]]
Conv 32 2 <dense><start>	[[3] [] [] [0] [] [] [1]]
Conv 32 2 Dense <units><start>	[[3] [] [] [] [] [] [1]]
Conv 32 2 Dense 32 <start>	[[3] [] [] [] [] [] []]
Conv 32 2 Dense 32 &	[[] [] [] [] [] [] []]

Figura 4.6: Processo de mapeamento DSGE para a solução apresentada na Figura 4.5.

4.3.1.4 Avaliação

Avaliar uma solução requer, primeiramente, que ela seja mapeada, então, a rede neural é construída e executada. Para problemas de classificação, por exemplo, o mais comum é usar a

acurácia alcançada pela rede sobre o conjunto de dados de entrada como a qualidade da rede. Dependendo do problema, diferentes métricas podem ser usadas, por exemplo, a função de perda, métricas de similaridade, tamanho do *batch*, número de épocas, erro quadrático médio (RMSE), dentre outros parâmetros. A acurácia da rede neural é computada quando é executado o treinamento da rede sobre o conjunto de validação. No final do algoritmo de evolução, a rede neural melhor evoluída é treinada novamente, e a acurácia final deve ser computada usando o conjunto de dados de teste.

4.3.2 Operadores Genéticos

Existem quatro operadores básicos aplicáveis em algoritmos evolutivos: seleção, cruzamento, mutação e substituição. O DSGE requer que eles sejam adaptados para trabalhar com codificação apresentada na Seção 4.3.1.1. Os operadores de seleção e substituição seguem o mesmo comportamento apresentado na literatura. Pode ser utilizado, por exemplo o torneio para a seleção e o elitismo para a substituição.

4.3.2.1 Cruzamento

O operador de cruzamento deve ser adaptado. A adaptação segue o mesmo comportamento do cruzamento de um ponto, a partir do qual são criadas duas variações. A primeira, batizada com o nome do *One-Point Crossover* original, começa com dois pais em que um ponto de “corte” aleatório é selecionado criando-se, então, a nova solução combinando a primeira parte de um pai e a segunda parte do outro pai, respeitando a blocos (sub listas). A segunda variante chama-se *Gene Crossover*, que, à semelhança da anterior, seleciona um ponto de “corte”, desta vez realiza-se o corte em cada sub lista individual e depois se cria a nova solução a partir da combinação dos pais. A representação gráfica para os dois cruzamentos é mostrada na Figura 4.7.

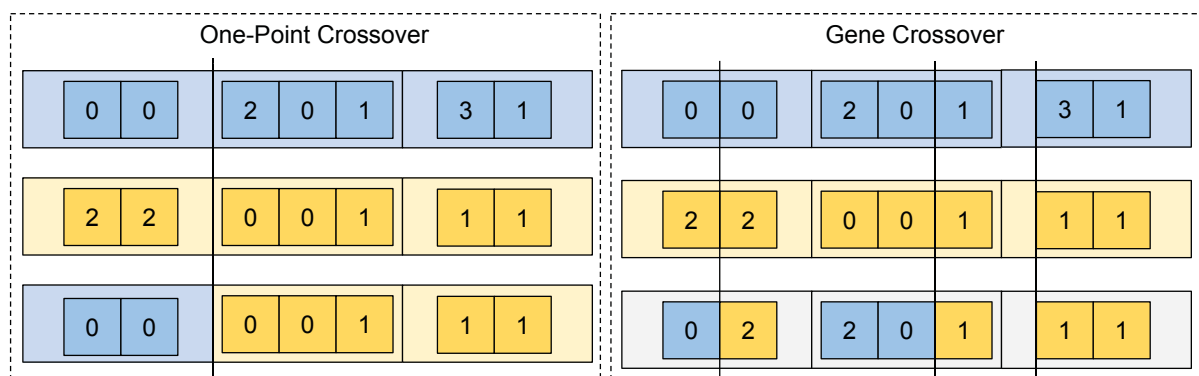


Figura 4.7: Exemplo de *One-Point* e *Gene Crossovers* aplicável no DSGE.

4.3.2.2 Mutação

O operador de mutação também é adaptado para nova codificação da representação. A adaptação segue o comportamento original de *point-mutation*. Um gene aleatório na solução é selecionado para ser substituído por um novo válido. A diferença aqui é que ele pode ser aplicado a todo o genótipo, produzindo, assim, uma maior variedade de alterações, ou restrito a partes do genótipo relacionadas a não-terminais ou terminais (ver Figura 4.8). Quando restrito, o operador pode produzir alterações de maior impacto ao alterar os valores não-terminais ou fazer o ajuste

fino alterando os valores do terminal. Quando um valor não-terminal é substituído, pode produzir uma mudança estrutural que, por exemplo, adiciona três novas camadas à arquitetura. Por outro lado, substituir um valor terminal altera o número de filtros em uma camada convolucional de 256 para 64, por exemplo.

Point-Mutation

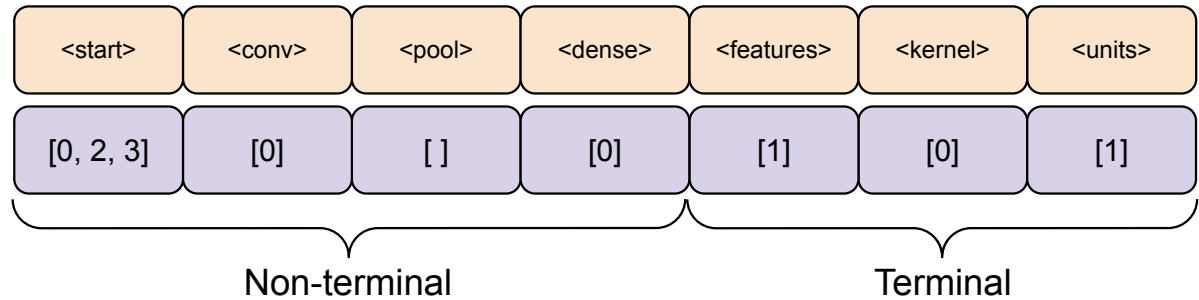


Figura 4.8: A codificação divide a solução de acordo com a forma como a gramática 4.4 está estruturada, agrupando sub listas relacionadas a não-terminais à esquerda e sub listas relacionadas a terminais à direita

Por fim, em cada geração, o DSGE funciona como um Algoritmo Genético convencional, gerando populações de soluções representadas de acordo com o descrito na Seção 4.3.1.1 e 4.3.1.3. O método de cruzamento (Seção 4.3.2.1) é utilizado para gerar novas arquiteturas de redes neurais. Estocasticamente, o operador de mutação (Seção 4.3.2.2) é executado para gerar variabilidade na população. O algoritmo é executado n vezes (ou n gerações), sendo esse valor paramétrico.

4.4 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentados os conceitos fundamentais da abordagem proposta neste trabalho relacionado à arquitetura de redes neurais baseadas em gramática evolutiva. Destacam-se aqui os conceitos de Evolução Gramatical, Evolução Gramatical Dinamicamente Estruturada (DSGE) bem como sua codificação, mapeamento, avaliação e operadores. Os conceitos apresentados são basilares para o decorrer do trabalho visto que os modelos de classificação de texto devem ser construídos dinamicamente a partir de arquiteturas de redes neurais geradas automaticamente.

5 TRABALHOS RELACIONADOS

Neste Capítulo serão discutidos os trabalhos relevantes ao Processamento de Linguagem Natural, principalmente no que se refere à representação e classificação de texto. Além disso, a literatura é visitada sobre a ótica de Evolução Gramatical.

5.1 PROCESSAMENTO DE LINGUAGEM NATURAL

Nos últimos anos, as redes neurais baseadas em representações vetoriais densas produziram excelentes resultados em várias tarefas de NLP [123]. O *Deep Learning* permite o aprendizado de representação automática de vários níveis. [31] descreve os princípios básicos para aplicar redes neurais para NLP de maneira tutorial. Muitos estudos conduzidos em *Deep Learning* mostraram uma boa melhora em termos de acurácia do modelo preditivo em tarefas de NLP, tais como Redes Neurais Convolucionais (CNN), Redes Neurais Recorrentes e Redes Neurais Recursivas.

Podem-se destacar ainda os trabalhos de: [116] que aborda a seleção de características baseadas na probabilidade desta existir em um dado documento, utilizando o cálculo de sua distância para uma representação vetorial do documento, eliminando da representação final palavras distantes dado um limiar paramétrico; [17] propõe o uso de *termsets* (conjunto de termos semanticamente próximos) ao invés de termos indexados para representar um documento utilizando uma ponderação de cada termo em um *termsets* de acordo com a frequência intra-termos e intra-documentos.

Em [52], os autores se apoiam na representação do texto utilizando tensores de 2ª ordem, compostos por um eixo que representa os documentos, outro os termos indexados e um terceiro que representa os conceitos associados, utilizando o *framework* do *Naive Bayes* estendendo seu comportamento para incorporar informação semântica (a partir dos conceitos). Nesta seção serão discutidos alguns trabalhos relacionados à representação e classificação de texto.

5.1.1 Representação de Texto

A pesquisa em representação de texto tem-se intensificado nos últimos anos, visto que muitos trabalhos agora buscam explorar a capacidade das CNN em extrair características refinadas de uma representação. A adaptação das CNN para o contexto de texto pode ser também observada nos trabalhos: [34] que propõem um modelo de representação de texto híbrido que combina uma abordagem Bidirectional Long Short-Term Memory (BiLSTM) com uma CNN; em [101] os autores propõem o BUNOW, uma abordagem que cria um vocabulário indexado, utilizando este índice como característica. Em seguida, a representação do documento é uma concatenação do produto deste índice pela representação vetorial de cada palavra, utilizando, em seguida, uma CNN para gerar características mais refinadas; e [24] usa duas CNN's responsáveis por extrair informações semânticas dos textos em ordem direta, e a segunda aprende a representação do texto na forma inversa.

Explorando a capacidade de classificação baseando-se em vizinhança, alguns autores utilizam o KNN como algoritmo de construção de contexto para texto, é o caso dos trabalhos: [45] que constroem uma matriz de similaridade entre documentos computada a partir da similaridade dos termos contidos no documento, então utiliza uma modificação do KNN para encontrar por votação o rótulo de documento de teste; [46] por sua vez, propõe uma abordagem baseada

num vetor de textos que carrega consigo valores de posicionamento de palavras, estatísticas de frequência e relações gramaticais. Então, uma matriz de similaridade é construída para que uma variação do KNN seja aplicada para classificação.

Zhu et al [1] avaliam a utilização de cinco formas de representação de texto que empregam matrizes de similaridade compostas por relação: 1) entre palavras dos documentos; 2) entre documentos do *dataset*; 3) entre palavras e documentos; 4) palavras dos documentos e entre os próprios documentos; 5) entre palavras dos documentos combinados com matriz de relação das palavras. [103] Propõe duas abordagens de representação baseada na combinação: 1) de palavras com papéis semânticos; 2) de sentenças com base em suas análises semânticas.

Além disso, [81, 59] propuseram duas arquiteturas de modelos para computar representações vetoriais contínuas de palavras de conjuntos de dados muito grandes. Os autores concluíram que é possível treinar vetores de palavras de alta qualidade usando arquiteturas de modelos muito simples, em vez dos modelos populares de redes neurais. Além disso, os autores propuseram o *Word2Vec* como *framework* para incorporação de palavras, então, como uma extensão deste, propuseram o *Doc2Vec*, mapeando cada parágrafo em um documento para um vetor único usando a média de um id de parágrafo e a representação de vetor de palavras.

Podem-se destacar ainda outras pesquisas relacionadas ao *Word2Vec* tais como: [29] propõem uma representação de uma palavra do documento como o produto da sua representação vetorial, dada pelo *Word2Vec*, e sua frequência sobre o documento, dado pelo TF-IDF, criando, assim, uma valoração estatística da representação semântica de cada palavra do documento; [42] assumem que palavras de um mesmo tópico possuem uma distribuição Gaussiana e, desde que cada palavra do documento possa ser definida como um tópico, a representação é então construída com uma mescla de representação por tópicos e baseada no modelo gaussiano; [15] propuseram o TF-ICF (Term Frequency - Inverse Corpus Frequency) que pondera o valor de uma palavra para um dado documento utilizando sua frequência e sua representação vetorial oriunda do *Word2Vec*; [121] integra LDA e *Word2Vec* a fim de criar uma representação híbrida, na qual cada documento é representado por um vetor de dimensão D, em que D representa a quantidade de tópicos relacionados ao *dataset*, sendo o valor de cada dimensão representado pela distância cosseno entre o documento e o dado tópico.

Utilizando o *Doc2Vec*, podem-se destacar as representações de texto em: [74] que aborda a concatenação de representações livre de contexto com representações contextuais, ambas oriundas do *Doc2Vec* para gerar uma representação refinada dos documentos originais; na abordagem de [39] são combinados LDA, *Doc2Vec* e *Topic2Vec* para gerar uma representação vetorial de cada documento; [43] compara representações de textos baseados em BoW, e diferentes representações de textos baseados em: 1) Média dos vetores do *Word2Vec*; 2) Média ponderada por IDF dos vetores do *Word2Vec*; 3) *Doc2Vec*; 4) Média ponderada por TFIDF dos vetores do *Word2Vec*.

Zheng et al [126] propõe uma abordagem denominada *discriminative topic sparse representation* (DTSR), na qual a construção automática de um dicionário de tópicos através do LDA e uma representação esparsa dos documentos que relaciona os documentos aos tópicos encontrados. [9] propõe uma abordagem de representação de palavras capaz de representar a modificação contextual que uma palavra sofre no decorrer do tempo com base nos conceitos de vizinhança mínima e vizinhança absoluta utilizando uma representação baseada em frequência. Na mesma linha [89] utiliza a meta-heurística PSO para construir iterativamente uma representação do texto baseada em uma das seguintes representações de palavras base: 1) Soma de vetores; 2) Soma ponderada por TF-IDF de vetores; 3) Média de vetores; 4) Média ponderada por TF-IDF de vetores. Ainda baseado no fortemente no TFIDF, [102] utiliza uma representação vetorial

esparso baseada em frequência construindo um dicionário incompleto composto por vetores TF correspondentes a todos os documentos de treinamento.

Na abordagem de [85] uma matriz de co-ocorrência de palavras (em geral esparsa) é construída e reduzida de acordo com o teste de Pearson a fim de construir uma matriz mais densa que represente o conjunto de documentos.

O ELMo (do inglês, *Embeddings from Language Models*) [87] é uma representação contextualizada de palavras que contém características complexas do uso de palavras (por exemplo, sintaxe e semântica) e como esses usos variam em contextos linguísticos (ou seja, para modelar polissemia).

Análogo ao Word2Vec, Doc2Vec e ELMo, [86] propuseram o GloVe, um algoritmo de aprendizado não-supervisionado para obter representações vetoriais para palavras. O treinamento é realizado em estatísticas globais agregadas de co-ocorrência de palavra em conjunto com seus documentos, e as representações resultantes mostram subestruturas lineares interessantes do espaço vetorial de palavras.

Recentemente, [122] propôs uma abordagem de um único grafo de texto para um corpus baseado na co-ocorrência de palavras e relações entre palavras do documento. [83] converte uma representação em grafos dos documentos através de combinações entre relações estruturais e relações contextuais dos documentos originais. Nesta linha de representação de textos em grafos, o trabalho de [58] destacou as técnicas para representação de texto: *Semantic Graph reduction method* e *Semantic triplet extraction* que utilizam representações baseadas em co-ocorrência de símbolos representados em grafos não direcionados.

[22] propuseram o BERT (do inglês, *Bidirectional Encoder Representations from Transformers*), baseado em duas etapas de treinamento, utiliza um modelo pré-treinado de representação de texto que tem alcançado e ultrapassado muitos modelos baseados em análise de frequência, e até mesmo os modelos baseados no Word2Vec e Doc2Vec. O BERT é descrito na Seção 3.2 com mais detalhes.

Como uma extensão do BERT, [93] propuseram o *Sentence-BERT*. Esse *framework* fornece um método fácil para calcular representações vetoriais densas para frases, parágrafos e imagens. Os modelos são baseados em *transformers* como o BERT [22], RoBERTa [67] e XLM-RoBERTa [19], entre outros. O modelo alcança alto desempenho em várias tarefas. O texto é incorporado no espaço vetorial de modo que o texto semelhante esteja próximo e possa ser encontrado com eficiência usando a similaridade do cosseno, por exemplo.

A Tabela 5.1 resume os trabalhos apresentados nesta seção.

5.1.2 Classificação de Texto

As redes neurais recorrentes (RNNs) consistem na ideia de processar informações sequenciais e são aplicadas para capturar dependências de palavras e, assim, auxiliar na classificação de textos. Long Short-Term Memory (LSTM) é uma arquitetura de rede neural recorrente amplamente utilizada para esta finalidade. As Redes Neurais Convolucionais (CNNs), por sua vez, reconhecem padrões no texto e destacam características únicas ou distintivas, o que torna esse tipo de arquitetura bastante popular para classificação de texto. Algumas abordagens combinaram RNN e LSTM [8] e RNN e CNN [10] para análise de sentimentos. Estudos recentes também mostraram resultados promissores em modelos de classificação de texto baseados em grafos usando Graph Neural Networks (GNNs) [122].

Em [18], os autores propuseram uma arquitetura baseada em CNN para resolver tarefas de NLP usando uma abordagem de janela. A abordagem cria dependência entre palavras vizinhas utilizando tais janelas. Em [49], foi descrita uma Rede Neural Convolucional Dinâmica (DCNN), que utilizou o agrupamento dinâmico k-Max, operação de agrupamento global sobre sequências

Tabela 5.1: Resumo dos trabalhos relacionados à representação de texto

Autores	Modelo
Yanbu Guo et al [34]	CNN e LSTM
Sharma et al [101]	CNN
Dong et al [24]	CNN
Taeho [45]	KNN
Taeho [46]	KNN
Zhu et al [1]	Similaridade vetorial
Sinoara et al [103]	Similaridade vetorial
Mikolov et al [81]	Word2Vec
Mikolov et al [59]	Doc2Vec
Fan et al [29]	Word2Vec e TF-IDF
Chen et al [15]	TF-ICF
Xue [121]	LDA e Word2Vec
Machhour et al [74]	Doc2Vec
Huaijin et al [39]	LDA, Doc2Vec e Topic2Vec
Zheng et al [126]	LDA
Pita et al [89]	PSO
Sharma et al [102]	TF
Peters et al [87]	ELMo
Pennington et al [86]	GLOVe
Liang et al [122]	TextGCN
Devlin et al [22]	BERT
Reimers et al [93]	Sentence-BERT
Yinhan et al [67]	RoBERTa
Conneau et al [19]	XML-RoBERTa

lineares e convolução entre palavras próximas. As redes neurais recursivas representam uma maneira natural de modelar sequências [123] e têm sido aplicadas para um melhor uso das interpretações sintáticas da estrutura da frase [104].

Recentemente, [122] propôs uma abordagem de um grafo de texto único para um *dataset* baseado em coocorrência de palavras e relações de palavras de documentos. Outros estudos usaram redes convolucionais de grafos (GCNs) para tarefas de NLP: [78] usou GCN para incorporar informações sintáticas em modelos neurais; [11] propôs um modelo para aprendizado gráfico-sequência que promoveu avanços recentes nas arquiteturas de codificador-decodificador neural; e [14] propuseram uma modificação para a arquitetura GCN introduzindo um mecanismo bidirecional para convolução de grafos direcionados, elevando qualidade dos resultados no conjunto de dados usado.

O advento do BERT criou novas perspectivas para a classificação de textos. O BERT foi projetado para pré-treinar representações bidirecionais profundas de um texto não rotulado, condicionando conjuntamente o contexto em todas as camadas. “Como resultado, o modelo BERT pré-treinado pode ser ajustado com apenas uma camada de saída adicional para criar modelos de última geração para uma ampla gama de tarefas, como resposta a perguntas e inferência de linguagem, sem tarefas ou modificações substanciais de arquitetura” [22].

5.2 DESIGN DE REDES NEURAI BASEADOS EM ALGORITMOS EVOLUTIVOS

5.2.1 Busca por Arquiteturas Neurais

Busca por Arquitetura Neurais (do inglês, *Neural Architecture Search*, **NAS**) é o termo usado para categorizar técnicas focadas em automatizar o projeto de redes neurais artificiais (RNAs), um tópico do campo de aprendizado de máquina automático [25].

Os métodos para NAS podem ser categorizados de acordo com o espaço de busca, estratégia de busca e estratégia de estimativa de desempenho utilizada:

- O espaço de busca define o(s) tipo(s) de RNA que pode(m) ser projetado(s) e otimizado(s);
- A estratégia de busca define a abordagem utilizada para explorar o espaço de busca;
- A estratégia de estimativa de desempenho avalia o desempenho de uma possível RNA a partir de seu projeto (sem construí-la e treiná-la).

A neuroevolução pode ser vista como um subcampo do NAS que abrange as técnicas baseadas em algoritmos evolucionários para gerar RNAs, aprendendo blocos de construção de redes neurais (por exemplo, funções de ativação), hiperparâmetros, arquiteturas e métodos para aprender pesos [107]. Os primeiros estudos [82, 120] já mencionavam o uso de algoritmos evolutivos, um algoritmo genético, neste caso, para evoluir as conexões das redes. Em relação aos trabalhos recentes, existe uma variedade de estudos e revisões [107, 25, 114] que abrangem as diferentes abordagens do assunto.

5.2.2 Modelos baseados em evolução

Vários estudos e pesquisas mencionam o uso de abordagens evolucionárias para projetar redes neurais artificiais (RNAs) [82, 120, 107, 25, 114]. As diferentes abordagens incluem aprender blocos de construção, hiperparâmetros, arquiteturas e técnicas de ajuste de peso [107]. Aprender os pesos das redes neurais usando algoritmos evolutivos [30, 23, 88] foram alguns dos primeiros estudos que combinaram busca evolutiva e RNAs. Essas abordagens são relatadas como uma alternativa rápida e robusta para aprender pesos de pequenas redes; no entanto, para redes maiores com dezenas de milhares de parâmetros, eles rapidamente se tornam inviáveis.

A pesquisa de topologia [80, 108, 69, 44] foi outro tópico que os pesquisadores destacaram para o projeto de RNAs. NeuroEvolution of Augmenting Topologies (NEAT), proposto por [108], foi um dos primeiros métodos conhecidos. Além disso, [80] expandiu seu trabalho e propôs o CoDeepNEAT para otimizar arquiteturas de Deep Learning (DL). Além da topologia, permitiu a evolução de outros componentes e hiperparâmetros. Os resultados relatados foram comparáveis aos melhores projetos humanos em *benchmarks* padrão em reconhecimento de objetos e modelagem de linguagem.

Nos últimos anos, as abordagens de GP [111, 63, 68] têm sido aplicadas com mais frequência ao projeto de DNNs. As abordagens baseadas em gramática [5, 63, 21], foco desta pesquisa, são destacadas como flexíveis porque utilizam gramáticas para definir os blocos de construção das redes. [4] propôs o DSGE para evoluir RNAs com mais de uma camada oculta.[64] e [21] adaptaram o DSGE e propuseram uma gramática simétrica para o projeto de U-nets.

Além disso, as abordagens baseadas em GP podem oferecer diferentes opções em relação ao design de DNNs; eles podem usar codificação direta, oferecendo mais controle sobre a solução e os operadores genéticos, ou codificação indireta, que requer pouca ou nenhuma modificação para os operadores de recombinação e, portanto, é mais fácil de usar e modificar para

uma representação simplificada. Além disso, as abordagens GP podem empregar a evolução de arquiteturas e parâmetros e são limitadas apenas pela flexibilidade da gramática e dos processos de construção.

5.3 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentados os trabalhos mais recentes que direcionaram as abordagens propostas aqui. Primeiramente foram apresentados trabalhos que norteiam o processamento de linguagem natural, seja na tarefa de representação de texto, ou seja na classificação de texto. Em seguida, os trabalhos referência na área de Busca por Arquiteturas Neurais e modelos baseados em evolução foram apresentados.

Segundo [109], as representações de texto encontram-se em uma categoria de alta dimensionalidade e extremamente esparsa, que por si só já dificulta o processo de agrupamento. A variação de métricas de distância e similaridade aplicadas aos algoritmos de representação adicionam uma maior variabilidade nas partições base e assim, podem revelar diferentes estruturas (possivelmente mais coesas) dos textos.

Ainda segundo [110], comparativamente, seus experimentos mostraram que representações de texto baseadas em grafos se comportam melhor do que as baseadas puramente em frequência (como TF-IDF), e que neste contexto o K-means e a partição de hipergrafos se adaptam à representação e que métricas como a euclidiana podem não apresentar resultados promissores em alta-dimensão e domínios esparsos.

As abordagens apresentadas para representação de texto demonstram uma captura de contexto acurada com o domínio do dataset. Entende-se que o processo de pré-processamento requer, em primeira instância uma adaptabilidade de conhecimento especialista para de fato criar um conjunto coeso para treinamento, validação e testes dos dados. Nesta pesquisa, diferentemente dos trabalhos citados, que utilizam um processo linear e refinamento dos dados, propõe-se uma abordagem estocástica para geração das representações, adaptável principalmente ao contexto intrínseco do dataset.

Cada abordagem neural oferece diferentes perspectivas para classificação de texto. As arquiteturas de rede neural são um fator essencial no modelo de classificação de texto. Arquiteturas adequadas podem funcionar muito bem para um grupo de conjuntos de dados e falhar em outros, o que torna a construção de um modelo genérico para qualquer banco de dados algo utópico. Nesta pesquisa, diferentemente dos trabalhos citados, que utilizaram arquiteturas pré-definidas para suas redes neurais e com base nas abordagens de gramática evolutiva apresentados por [63], [64] e [21], propõe-se um projeto automático de redes neurais utilizando programação genética aplicadas especialmente na classificação de texto.

6 ABORDAGENS PARA REPRESENTAÇÃO E CLASSIFICAÇÃO DE TEXTOS COMO MODELOS OTIMIZADOS

Nesta seção são apresentadas as questões de pesquisas deste trabalho. Em seguida, as abordagens são apresentadas bem como os experimentos associados que validam a proposta. Por fim, os resultados dos experimentos são discutidos e as questões de pesquisa, revisitadas.

6.1 QUESTÕES DE PESQUISA

Os principais objetivos desta pesquisa são: 1) propor uma abordagem capaz de produzir representações de texto coesas e semanticamente relevantes e 2) produzir redes neurais de alta qualidade, que são adaptadas ao problema de classificação de texto, reduzindo a necessidade de conhecimento especialista na tarefa da arquitetura das redes.

As questões de pesquisa a seguir são propostas para discutir os aspectos individuais que ajudam a abordar o objetivo principal.

- QP1: A abordagem de representação de texto é facilmente adaptada para diferentes tipos de dataset?
- QP2: As representações de texto geradas são competitivas com outros métodos de geração de representação?
- QP3: A abordagem de geração de redes neurais é facilmente adaptada para diferentes tipos de redes?
- QP4: As redes projetadas automaticamente podem ter um desempenho melhor ou semelhante aos projetos criados manualmente?
- QP5: Que novidade essa abordagem traz em comparação com outras técnicas de neuroevolução?

Resolver essas questões permeia encontrar respostas para algumas questões fundamentais apresentadas neste trabalho: *O que são representações de texto coesas?* ou *O que são palavras semanticamente relevantes?*. Como demonstrado, a pesquisa em NLP não é nova, entretanto vem evoluindo a passos largos nos últimos anos. Encontrar modelos que respondam essas perguntas para *benchmark* se tornou habitual na comunidade científica. Entretanto, a aplicação de tais modelos em contextos práticos tem consumido bons recursos de tempo. O que seria uma representação coesa de um texto jurídico, ou de uma sentença, ou ainda de uma petição judicial? A resposta não é clara se não partir de um especialista na área.

Uma representação coesa de um conjunto de textos é tal que a representação matemática de uma sentença está próxima, no espaço vetorial, de outra representação de outra sentença se ambas são semanticamente semelhantes (tratam do mesmo assunto, tem a mesma interpretação, ou interpretação semelhante, ou o sujeito daquela sentença tem uma relação semântica com o sujeito desta sentença). Responder esses questionamentos para espectros mais específicos como o médico ou o jurídico é ainda mais desafiador.

Quando se lê um texto, é comum que alguém fluente na língua leia a sentença sem necessariamente ler palavra por palavra, o cérebro busca encontrar palavras dentro da frase que dê sentido semântico e que o interlocutor consiga captar a informação ali contida. Então,

naturalmente são dadas mais ênfase a algumas palavras em relação a outras. Por exemplo, ao ler esse parágrafo, provavelmente alguns artigos, conjunções ou palavras sem grande contribuição para a semântica foram descartados. Para realizar essa tarefa de forma automática um modelo de inteligência precisa encontrar palavras mais relevantes, isto é, palavras semanticamente mais relevantes. Essa tarefa, mais uma vez é realizada por especialistas em modelos tradicionais, principalmente com o uso da técnica *stopwords removal*. Entretanto, modelos aplicados a domínios mais específicos carecem de um aprofundamento mais lógico e ao mesmo tempo automático, visto que ora uma palavra pode ser relevante em uma sentença, e em outra não.

Para sanar as questões de pesquisa propostas, este trabalho apresenta duas abordagens para tarefas de Processamento de Linguagem Natural. Primeiramente, é descrito o **Boostered Encoder for Classification Architecture - BECA**, responsável por gerar representações de texto refinadas e concisas. Em seguida é descrita a abordagem **TextDSGE - Text Classification using Dynamic Structured Grammar Evolution**, responsável por desenvolver arquiteturas de redes neurais multicamadas, usando estruturas mistas de redes recorrentes, convolucionais ou em grafos. A Figura 6.1 representa em termos gerais como as duas técnicas interagem entre si com o objetivo de gerar um modelo refinado de classificação de texto.

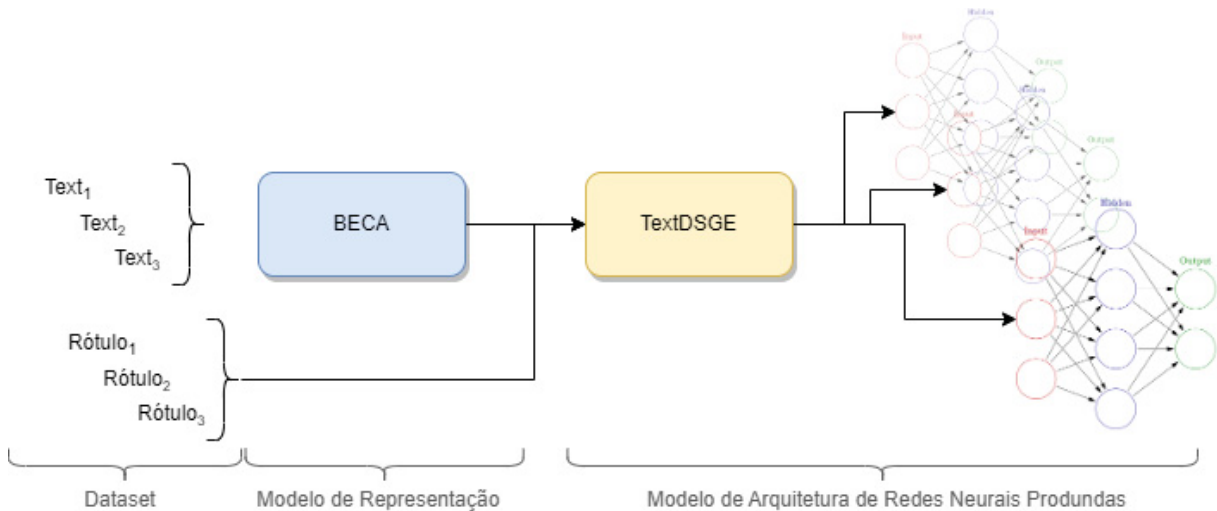


Figura 6.1: Fluxo genérico de interação entre as técnicas BECA e TextDSGE

6.2 MODELO BECA

Nesta seção, propõe-se o *framework* **Boostered Encoder for Classification Architecture - BECA**, especializado na geração de representações de texto coesas. São descritos ainda o funcionamento do Sentence-BERT [93], bem como ele é utilizado como núcleo para geração de representações de texto. Em seguida, os *datasets* selecionados como *benchmarks* são descritos. Por fim, são demonstrados os experimentos e resultados que validam a abordagem.

O BECA é um *framework* que utiliza o motor do *Sentence-BERT* [93] para construir representações de texto refinadas ao domínio de aplicação. É importante frisar que a saída do BECA é uma representação de texto vetorial ou matricial (parametricamente selecionada), é dissociada de qualquer modelo de classificação ou agrupamento, podendo ser acoplada a quaisquer outras propostas. O modelo BECA é descrita detalhadamente na Seção 6.2.1.

Como descrito na Seção 3.2.1, ainda que o Sentence-BERT seja uma evolução significativa no que se refere a representação de texto usando o motor do BERT, alguns pontos podem ser enfatizados. Primeiramente, o modelo está ligado diretamente ao processo de treinamento do

BERT original, carregando consigo informações contextuais daquele dataset. Decorrente disso, dependendo do *dataset* que seja escolhido para execução dos experimentos as representações podem ganhar vieses não desejados.

Concomitante a isso, o *modus operandi* da geração de embeddings para uma palavra (ou sentença) leva em consideração primordialmente um vetor original (decorrente da questão mencionada anteriormente) para cada token conhecido e informações posicionais das palavras. Dessa forma, a representação de duas palavras "coelho" em uma mesma sentença podem ter representações ligeiramente diferentes. Isso de fato não é um problema em si, mas sim uma vantagem do modelo. Decorre que, para gerar informações posicionais e ajustar os valores de cada vetor, o BERT (e por conseguinte o *Sentence-BERT*) consideram todos os *tokens* conhecidos da sentença e isso pode causar um viés ainda maior na representação final da sentença.

Reforça-se que, caso o BERT não conheça um *token*, ele é decomposto em elementos menores até que sejam encontrados vetores primordiais que possam representa-lo. No pior caso, palavras não conhecidas pelo modelo acabam por ser decompostas até seus radicais. Dessa forma, automaticamente, afixos morfológicos são separados de seus radicais e tornam-se novos *tokens*. Como exemplo, volta-se a Figura 3.13, na qual a palavra "FOFINHO" foi separada em dois *tokens* "FOF"(seu radical) e "INHO"(um sufixo morfológico).

Além disso, é importante que atividades de pré-processamento sejam prioritariamente executadas em modelos baseados no BERT por alguns motivos:

- Palavras consideradas *stop words* serão consideradas na geração de representação do *token* CLS;
- Afixos morfológicos, mesmo que separados no processo padrão do BERT/*Sentence-BERT*, serão considerados na representação final, podendo adicionar carga semântica desnecessária;
- Palavras de baixíssimo poder semântico sempre serão consideradas e mesmo que sejam conhecidas pelo modelo do BERT/*Sentence-BERT* adicionarão um viés semântico à representação final.

Suponha a sentença $s = \text{"O COELHO É MUITO FOFINHO"}$, e um processo de construção de representação vetorial de textos σ de acordo com a Equação 6.1.

$$\sigma(v) = \frac{\sum_{i=1}^k v_i}{k} \quad (6.1)$$

em que $v = v_1, v_2, \dots, v_n$ e $k = |v|$, tal que $v_i \in V$ e V é o conjunto de todos os vetores de todas as palavras de um vocabulário idealizado. É possível construir a representação $\sigma(v)$ utilizando todos os *tokens* da sentença original, isto é, $|v| = |T_s|$ ¹ ou utilizando apenas alguns *tokens* de s , isto é, $|v| \leq |T_s|$. As Figuras 6.2 e 6.3 apresentam as distorções que podem acontecer quando considerados todos os *tokens* e quando são removidos (automaticamente ou não) *tokens* semanticamente pobres para o contexto.

Para a tarefa de reduzir o tamanho do conjunto v , pode-se considerar a remoção de *stop words*. É natural incluir essa atividade, principalmente, porque ela é considerada relativamente simples (como demonstrado na Seção 2.1.1.2). O cerne do problema está em encontrar palavras no texto que não foram previstas (no conjunto de *stop words* original) e que carregam baixa carga semântica, agregando pouco ou nenhum valor contextual para a sentença. Nesta lacuna, propõe-se o BECA descrito na Seção 6.2.1.

¹Suponha T_s o conjunto formado pelos *tokens* da sentença s

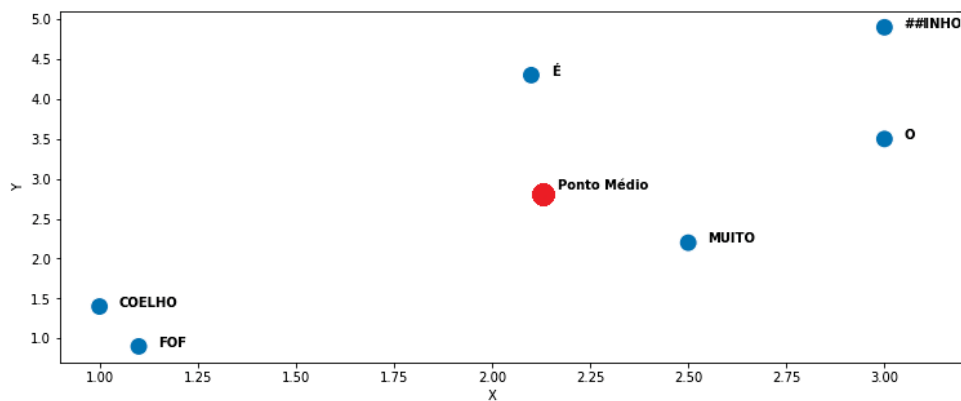


Figura 6.2: Representação dos *tokens* no espaço vetorial sem nenhuma tratativa ou pré-processamento.

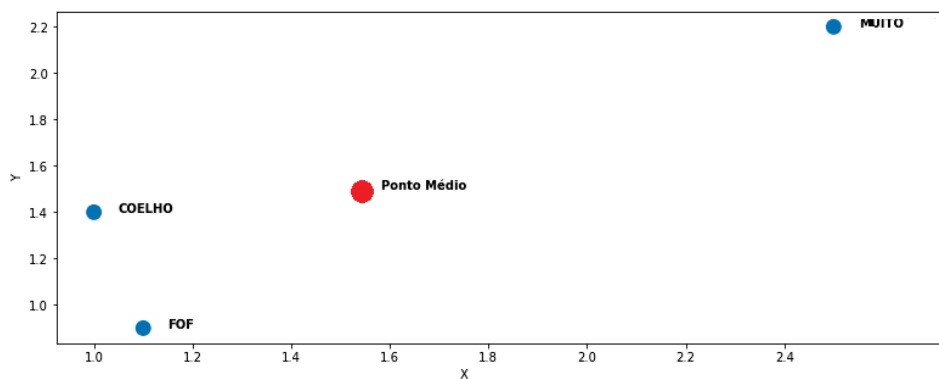


Figura 6.3: Representação dos *tokens* no espaço vetorial com remoção de *tokens* com baixa carga semântica.

6.2.1 BECA

Em termos gerais, a abordagem BECA é descrita na Figura 6.4. O BECA é considerado uma ferramenta não-supervisionada de representação de texto que elimina automaticamente termos/*tokens* pouco expressivos. Como entrada, o *framework* recebe um conjunto de textos não rotulados e não pré-processados. Primeiramente, o conjunto de n textos é apresentado a um módulo de pré-processamento, no qual as atividades descritas na Seção 2.1.1 são executadas. A saída desse módulo é um conjunto de n textos pré-processados (sem *stop words*, sem código html, sem afixos morfológicos etc).

Em seguida, o módulo *Particionador* é executado. O pseudocódigo é apresentado no Algoritmo 5. Este módulo é responsável por separar as palavras de cada sentença em três grupos, denominados aqui de S (singularidade), H (horizonte) e D (disco). Este módulo executa sequencialmente as seguintes tarefas:

1. Sumariza os vocábulos encontrados no conjunto de textos de entrada;
2. Usando o modelo pré-treinado do Word2Vec², gera representações vetoriais para cada vocábulo encontrado no conjunto de textos;
3. Executa o algoritmo de agrupamento hierárquico [48] com o objetivo de encontrar o agrupamento C_E que representa os grupos de palavras semanticamente próximas. O agrupamento é avaliado de acordo com sua métrica *silhouette*, determinando assim um conjunto confiável de grupos;

²<https://code.google.com/archive/p/word2vec/>

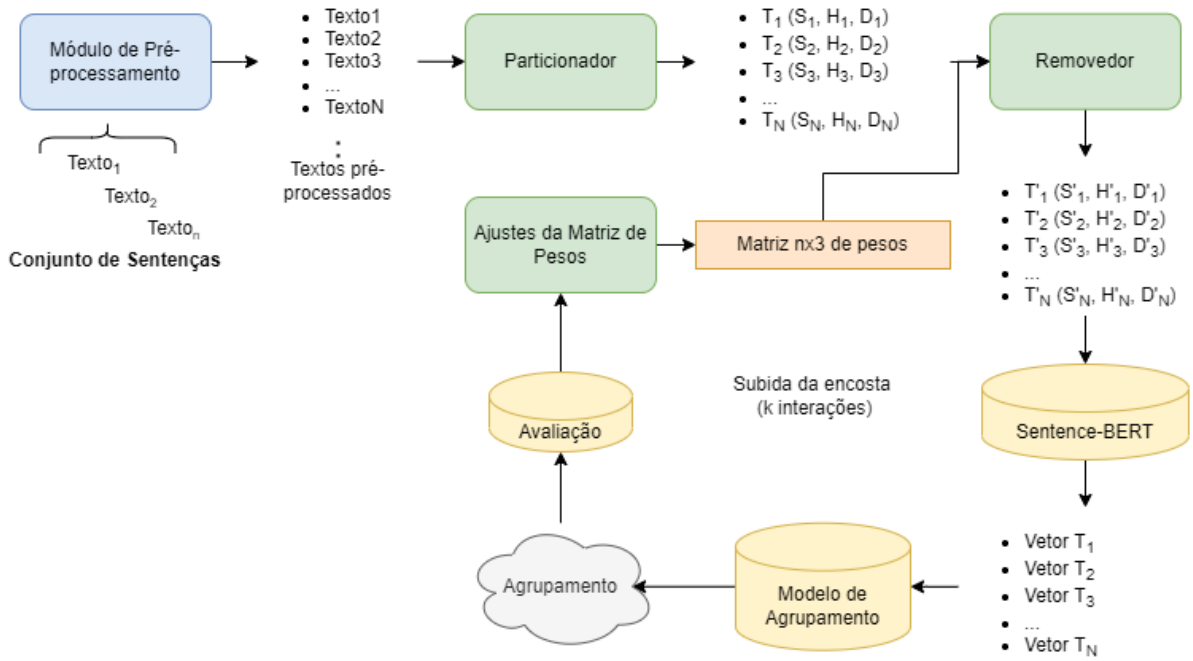


Figura 6.4: Fluxo genérico do funcionamento do *framework* BECA. O modelo desenvolve o algoritmo subida da encosta por k interações a fim de encontrar uma representação mais coesa

4. Calcula o conjunto $P = \{p_1, p_2, \dots, p_c\}$ tal que $c = |C_E|$ e p_x é ponto médio calculado para cada grupo C_E ;
5. Em seguida, seja então $E = \{e_1, e_2, \dots, e_y\}$ o conjunto de todos os *embeddings* do vocabulário encontrado nas sentenças W , para cada vetor e_i , é calculada sua distância cosseno d_i para o ponto médio $P_{m'}$ de seu próprio grupo $C_{m'}$, tal que w_i é um vocábulo do conjunto E e $w_i \in C_{m'}$;
6. Para cada sentença $W_k \in W$, é criada uma tupla $T_k = (S_k, H_k, D_k)$ em que S_k contém as n palavras $w_{ik} \in W_k$ com menor valor de d_i . H_k contém as $n/3$ palavras $w_{ik} \in W_k$ com menor valor de d_i não presentes em S_k . Por fim, D_k contém as x palavras de W_k não presentes no conjunto $S_k \cup H_k$, tal que $S_k \cup H_k \cup D_k \subseteq W_k$. O valor de n é paramétrico no algoritmo.

Para a próxima etapa, considera-se, então, uma matriz de pesos $Q_{k \times 3}$ tal que $\{q \in Q : 0 < q \leq 1\}$, inicialmente com pequenos valores aleatórios. O módulo *Removedor* constrói uma tupla T' derivada de T em que para cada conjunto X_{ik} da tupla $T_i \in T$ e para cada palavra $w_i \in X_{ik}$ remove-a do conjunto derivado com probabilidade q_{ik} . Esse módulo permite modificar o conjunto original retirando algumas palavras da sentença, sob a hipótese de que elas carregam baixa carga semântica.

Em seguida, cada tupla $T'_i \in T'$ remonta uma nova sentença para que o algoritmo Sentence-BERT seja capaz de realizar o *embedding* para cada nova sentença. O passo seguinte usa um modelo de agrupamento paramétrico para realizar o agrupamento dos novos *embeddings*. O resultado do agrupamento é então avaliado sob a ótica da métrica *silhouette*, verificando quão coerentes e coesos estão os grupos gerados.

Algoritmo 5 Algoritmo particionador de sentenças

```

1:  $W \leftarrow$  Conjunto de todas as sentenças
2:  $n \leftarrow$  número de palavras esperado no conjunto de singularidade (S)
3:  $E \leftarrow \text{WORD2VEC}(W)$ 
4:  $C_E \leftarrow \text{AGRUPAMENTO\_HIERARQUICO}(E)$ 
5:  $P \leftarrow \{\}$  Conjunto de Pontos Médios
6:  $D' \leftarrow \{\}$  Conjunto de distâncias de cada palavra para o ponto médio de seu grupo em  $C_E$ 
7:  $T \leftarrow \{\}$  Conjunto de tuplas
8: for cada agrupamento  $c_i \in C_E$  do
9:    $P \leftarrow P \cup \{\text{PONTO\_MEDIO}(c_i)\}$ 
10: for cada embedding  $e_i \in E$  do
11:    $c_e \leftarrow \text{OBTEM\_GRUPO}(e_i, C_E)$ 
12:    $p_e \leftarrow \text{PONTO\_MEDIO}(c_e)$ 
13:    $d_i \leftarrow \text{DISTANCIA}(e_i, p_e)$ 
14:    $D' \leftarrow D' \cup (e_i, d_i)$ 
15: for cada sentença  $W_k \in W$  do
16:    $S_k \leftarrow \text{SINGULARIDADE}(W_k, D')$ 
17:    $H_k \leftarrow \text{HORIZONTE}(W_k, D', S_k)$ 
18:    $D_k \leftarrow \text{DISCO}(W_k, D', S_k, D_k)$ 
19:    $T \leftarrow T \cup (S_k, H_k, D_k)$ 
return  $T$ 

```

Por fim, o módulo de *Ajustes da Matriz de Pesos* é executado. Dado que na métrica *silhouette* o melhor valor é 1 e o pior valor é -1^3 , executa-se o ajuste de pesos segundo a Equação 6.2, adaptada do método de *back-propagation* [61].

$$q_{ik} \leftarrow q'_{ik} - \eta \frac{\delta E}{\delta q} \quad (6.2)$$

isto é, o valor do peso atual q_{ik} será igual anterior q'_{ik} corrigido o valor proporcional ao gradiente⁴. O parâmetro η representa a taxa de aprendizado do algoritmo, controlando a tamanho do passo que se toma na correção do peso. A expressão $\frac{\delta E}{\delta q}$ consiste em computar as derivadas parciais da função de erro E em relação a cada peso da matriz de pesos Q .

Sendo 1 a saída do valor *silhouette* esperada para cada instância (sentença) e y' a saída obtida pelo agrupamento, define-se a função de erro pela equação 6.3.

$$E(y') = \sum_{i=1}^n (1 - y'_i)^2 \quad (6.3)$$

O algoritmo, assim, utiliza a estratégia adaptada do método da subida da encosta para realizar um número k paramétrico de iterações. Por questões de otimização de memória, sempre a melhor estrutura T' é guardada. O Algoritmo 6 descreve os processos da abordagem BECA. Primeiramente o algoritmo gera uma matriz de pesos aleatória (linha 7).

³Valores próximos de 0 indicam grupos sobrepostos. Valores negativos geralmente indicam que uma amostra foi atribuída ao grupo errado, pois um grupo diferente é mais semelhante.

⁴É importante que a função de erro seja diferenciável

Algoritmo 6 Boosted Encoder for Classification Architecture - BECA

```

1:  $k \leftarrow$  número de rodadas do algoritmo
2:  $\eta \leftarrow$  taxa de aprendizagem
3:  $n \leftarrow$  número de palavras esperado no conjunto de singularidade
4:  $W \leftarrow$  Conjunto de sentenças
5:  $N \leftarrow |W|$ 
6:  $FN\_CLUSTER \leftarrow$  Algoritmo de Agrupamento
7:  $Q \leftarrow$  Matriz aleatória de pesos ( $N \times 3$ )
8:  $W' \leftarrow PRE\_PROCESSAMENTO(W)$ 
9:  $T \leftarrow PARTICIONADOR(W')$ 
10:  $MELHOR_T \leftarrow nil$ 
11:  $MELHOR_{SM} \leftarrow -\infty$ 
12: for  $n$  vezes do
13:    $T' \leftarrow REMOVEDOR(T)$ 
14:    $E \leftarrow SENTENCE\_BERT(T')$ 
15:    $C_E \leftarrow FN\_CLUSTER(E)$ 
16:    $V_s \leftarrow AVALIACAO(C_E)$ 
17:    $V_{sm} = MEDIA(V_s)$ 
18:   if  $V_{sm} \geq MELHOR_{SM}$  then
19:      $MELHOR_T \leftarrow T'$ 
20:      $MELHOR_{SM} \leftarrow V_{sm}$ 
21:    $Q \leftarrow ATUALIZA\_PESOS(Q, \eta, V_s)$ 
return  $MELHOR_T$ 

```

6.2.2 Representação de textos utilizando Grafos

Alguns modelos estudados neste trabalho utilizam representação de texto baseados em grafos. Este contexto possui alguns detalhes que devem ser elencados. Em geral, os algoritmos de aprendizado de máquina baseados em grafos empregam dois artefatos de entrada: uma matriz de características (do inglês, *feature matrix*) e uma matriz de adjacência (do inglês, *adjacency matrix*). A primeira representa as características das instâncias de treinamento, validação e teste. Em muitos modelos, como em [122], essa matriz é aleatoriamente gerada e busca-se aprender a representação da instância com o aprendizado do modelo. Em outros casos, essa matriz pode representar características contextuais.

A matriz de adjacência descreve as relações entre os objetos de treinamento, para que seja representado de fato o grafo. Em problemas de classificação de texto, essa matriz em geral representa as relações das palavras de cada sentença a ser classificada. Neste trabalho, quando necessária uma representação baseada em grafos optou-se por estender o trabalho de [122], exceto quando as sentenças devem ser representadas.

Em [122] as sentenças eram representadas por vetores de zeros e apenas as palavras. Neste trabalho propõem-se representações a partir do BECA. Como em [122], o grafo é representado pela matriz de adjacência, no qual cada nó representa uma sentença ou uma palavra. A aresta entre nós de duas palavras é construída usando-se a informação de co-ocorrência formulada pela Equação 6.4. A aresta entre um nó de palavra e um nó de sentença é construída com o emprego da frequência da palavra, pela Equação 6.7.

Na Equação 6.4, $p(i,j)$ representa a razão das palavras i e j sobre o total de janelas de contexto e $p(i)$ representa a razão da palavra i sobre o total de janelas. $W(i,j)$ é o número de

janelas deslizantes no dataset que contém a palavra i . W é o número total de janelas deslizantes no *dataset*.

$$PMI(i, j) = \log \frac{p(i, j)}{p(i)p(j)} \quad (6.4)$$

$$p(i, j) = \frac{W(i, j)}{W} \quad (6.5)$$

$$p(i) = \frac{W(i)}{W} \quad (6.6)$$

Na Equação 6.7, $f_{w,d}$ é o número de vezes que w aparece em d , l_d é o tamanho do conjunto de documentos/sentenças, e $f_{w,D}$ é o número de documentos/sentenças no qual a palavra w aparece em D [97]. Desta forma, é possível representar uma palavra através do vetor que determina sua frequência relacionada com cada documentos(sentenças) no *dataset*.

$$w_d = f_{w,d} * \log\left(\frac{l_d}{f_{w,D}}\right) \quad (6.7)$$

Dessa forma, o BECA é capaz de parametricamente gerar a representação baseada em grafos. A modificação fica por conta da geração adicional da matriz de adjacência $A \in \mathbf{R}^{m \times n}$, tal que $m = |s| + |v|$, em que s é o conjunto de todas as sentenças do conjunto de dados e v é o vocabulário gerado pelo processo de representação vetorial (descrito no Algoritmo 5, linha 3) e n é o tamanho da representação escolhida⁵. A Figura 6.5 mostra um exemplo de como o BECA pode ser adaptado para representações baseadas em grafos.

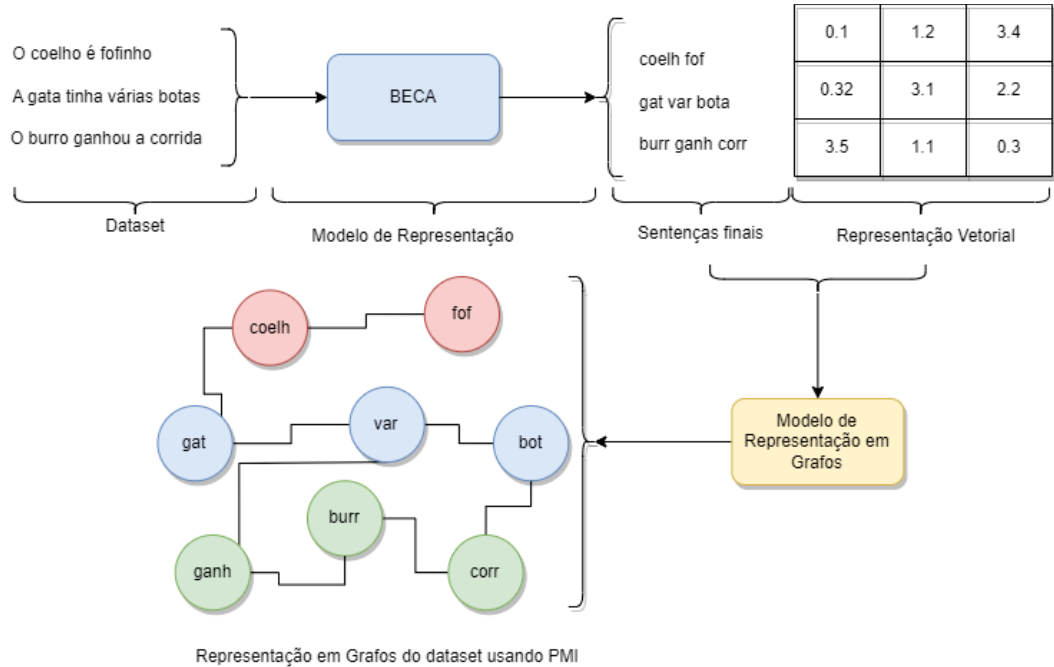


Figura 6.5: Fluxo genérico de representação de texto em grafos usando o BECA

A validação da abordagem BECA é descrita na Seção 7.0.2. Os modelos baseados em representações tradicionais (vetores) ou em grafos são validados utilizando os *datasets* descritos na Seção 7.0.1.

⁵Para o *Sentence-BERT*, esse valor é, por padrão, 768

6.3 MODELO TEXTDSGE

Nesta sessão, propõe-se o *framework* TextDSGE - Text Classification using Dynamic Structured Grammar Evolution, especializado no desenvolvimento de redes neurais aplicadas à classificação de texto. São descritas ainda as gramáticas que permitem a combinação de diferentes estruturas de redes neurais, bem como diferentes modelos de camadas de convolução em grafos. Por fim, são descritos experimentos utilizando o *framework* proposto.

O TextDSGE é um *framework* que emprega o motor do DSGE para construir arquiteturas baseadas em CNN, LSTM e GNN. É importante mencionar que é facultativo o uso do *framework* BECA (proposto na Seção 6.2) neste ponto, visto que ambos podem ser utilizados como módulos independentes.

Em termos gerais, a abordagem TextDSGE é descrita pela Figura 6.6. Como entrada, o *framework* recebe uma representação de textos no espaço vetorial. O módulo seguinte divide o conjunto de entrada em n -fold. Cada *fold* é caracterizado por um conjunto de dados de treinamento, validação e teste para garantir a técnica de *cross-validation*. Para cada execução do *framework* um dos conjuntos de *fold* é selecionado.

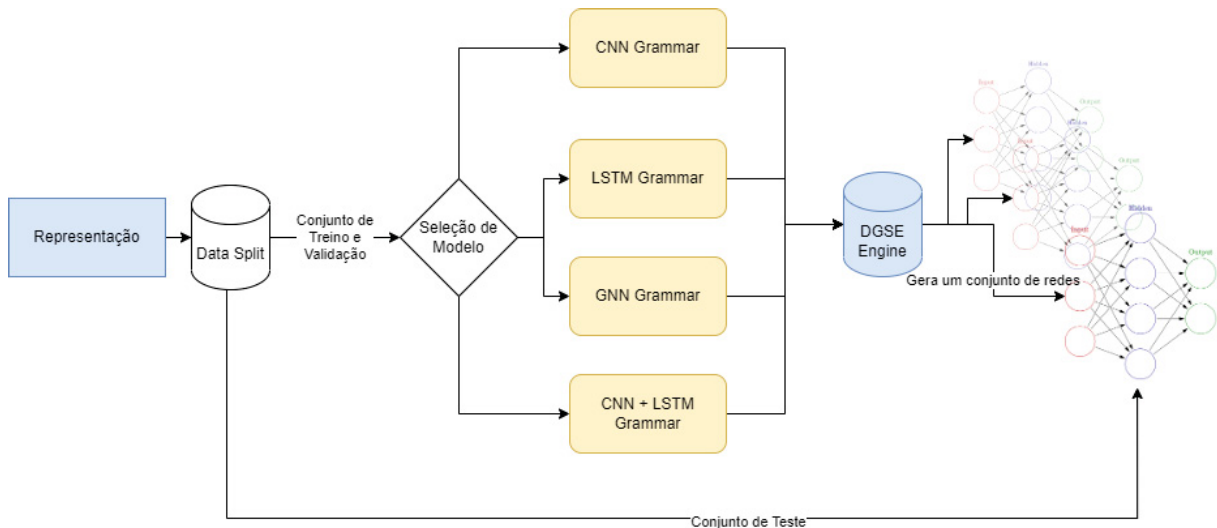


Figura 6.6: Fluxo genérico do funcionamento da abordagem TextDSGE

A seleção da gramática é um elemento paramétrico do *framework*, permitindo, assim, a automação de diferentes rodadas de diferentes algoritmos. Em seguida, a gramática selecionada, o conjunto de treino e o conjunto de validação são inseridos no módulo de DSGE. A saída do TextDSGE é um conjunto paramétrico de redes neurais artificiais às quais é submetido o conjunto de dados de teste.

6.3.1 Operadores

'Half-and-Half': Para a implementação TextDSGE, o operador de mapeamento foi levemente modificado no que se refere ao mapeamento de camadas de tipos diferentes. Quando necessário, programaticamente a saída de uma camada de um tipo (por exemplo, CNN) deve ser mapeada para a entrada de uma camada de outro tipo (por exemplo, LSTM). Isso se deu necessariamente pelas diferenças de implementação de cada um dos tipos de camadas e frameworks utilizados.

Como extensão do DSGE tradicional, na abordagem TextDSGE propõe-se a utilização do operador de recombinação *'Half and Half'* [65] que combina os operadores de Cruzamento e Mutação. Ele trabalha aplicando o cruzamento ou a mutação com uma paramétrica probabilidade.

Por exemplo, se o cruzamento for definido com a probabilidade $p = 0.6$, então, automaticamente, a probabilidade de mutação será ajustada para $m = 1.0 - p$, isto é, 0.4.

Nas seções seguintes são detalhas as gramáticas construídas, com suas características e limitações. Em seguida são descritos os experimentos e resultados alcançados com a abordagem TextDSGE.

6.3.2 Gramática CNN

A construção de uma Rede Neural Convolutacional (CNN) envolve a melhor escolha de quantidades de camadas, camadas de *dropout*, número de unidades, entre outros parâmetros. Usando-se a abordagem baseada em DSGE (descrita na seção 4.3, foi possível definir a gramática da Figura 6.7 focada nessas camadas e hiperparâmetros. Além disso, para se evitar a construção de redes inválidas, foram definidas algumas restrições para construí-las.

```

<cnn> ::= <conv> <c_layer> flatten <d_layer> <dense>

<c_layer> ::= <conv> <c_layer> | <pool> <c_layer> | <c_node> | &

<d_layer> ::= <dense> <d_layer> | <dropout> <d_layer> | <d_node> | &

<c_node> ::= <conv> | <pool>

<d_node> ::= <dense> | <dropout>

<conv> ::= conv <filters> <ksize> <strides> <padding> <activation>

<dense> ::= dense <units> <activation>

<dropout> ::= dropout <rate>

<pool> ::= <pool_type> <ksize> <strides> <padding>

<pool_type> ::= maxpool | avgpool

<activation> ::= relu | selu | elu | tanh | sigmoid | linear

<padding> ::= valid | same

<filters> ::= 16 | 32 | 64 | 128 | 256

<strides> ::= 1 | 2

<ksize> ::= 1 | 2 | 3 | 4

<units> ::= 32 | 64 | 128 | 256 | 512 | 1024

<rate> ::= [0.0,0.5]

```

Figura 6.7: Proposta de gramática para a geração de arquiteturas CNN. As camadas são definidas de acordo com a estrutura Keras (<https://keras.io/>) ou tipos especiais, como a regra <rate>, que é um valor aleatório dentro do intervalo definido quando selecionado.

Ficou decidido que as soluções inválidas produzidas pela gramática proposta seriam penalizadas, já que isso não adiciona custo computacional ao processo. Além disso, um conjunto de restrições foi adicionado à gramática, diminuindo o número de arquiteturas inválidas para cerca de 2% após os testes empíricos. As regras são as seguintes:

- O modelo terá pelo menos uma convolução e uma camada densa;
- As camadas convolucionais e de *pooling* só podem aparecer antes das camadas densas e *dropout*;
- A primeira camada densa é precedida por uma camada de *flatten* ⁶;
- A última camada será sempre um nó denso com ativação *softmax* e o número de unidades definido como o número de classes do conjunto de dados.

6.3.3 Gramática LSTM

As redes neurais LSTM ganharam notoriedade em tarefas relacionadas à NLP. Por isso, foi construída uma gramática (veja a Figura 6.8) focada nas camadas e hiperparâmetros mais comuns usados em arquiteturas LSTM. Algumas restrições para construir as redes e garantir a criação de modelos válidos foram adicionadas.

```

<lstm> ::= <lstm_layer> <layer> <dense>

<layer> ::= <lstm_layer> <layer> | <dropout> <layer> | &

<lstm_layer> ::= lstm <units> <activation> <recurrent_activation>

<dense> ::= dense <units> softmax

<dropout> ::= dropout <rate>

<recurrent_activation> ::= tanh | sigmoid | linear

<activation> ::= relu | selu | elu | tanh | sigmoid | linear

<units> ::= 32 | 64 | 128 | 256 | 512 | 1024

<rate> ::= [0.0,0.5]
```

Figura 6.8: Proposta de gramática para a geração de arquiteturas LSTM. Os módulos foram definidos de acordo com o *framework* Keras (<https://keras.io/>).

Assim como na abordagem utilizando a CNN, as soluções inválidas produzidas pela gramática proposta são penalizadas, já que isso não adiciona custo computacional ao processo. Além disso, um conjunto de restrições foi adicionado à gramática, diminuindo o número de arquiteturas inválidas para cerca de 5% após o teste empírico. As regras são as seguintes:

- O modelo terá pelo menos um LSTM e uma camada densa;
- A última camada será sempre um nó denso com ativação *softmax* e o número de unidades definido como o número de classes do conjunto de dados.

⁶Camada de achatamento de dimensionalidade

6.3.4 Gramática CNN+LSTM

A habilidade gramatical pode ser expandida combinando as duas gramáticas propostas nas Seções 6.3.2 e 6.3.3. A nova gramática proposta (veja a Figura 6.9) une as gramáticas mostradas nas Figuras 6.7 e 6.8. A nova gramática apresentada na Figura 6.9 relaciona-se diretamente com ambas as gramáticas pais. Neste caso, a nova gramática é uma extensão direta das regras de produção $\langle cnn \rangle$ e $\langle lstm \rangle$. A hipótese desta união é, em termos gerais, agregar a capacidade primordial de extração de características que as redes CNN têm, com a capacidade de memória que as redes LSTM possuem.

$$\begin{aligned}\langle mxnn \rangle &::= \langle m_layer \rangle \langle layer \rangle \langle dense \rangle \\ \langle m_layer \rangle &::= \langle cnn \rangle \mid \langle lstm \rangle \\ \langle layer \rangle &::= \langle cnn \rangle \mid \langle lstm \rangle \mid \langle layer \rangle \langle layer \rangle \mid \&\end{aligned}$$

Figura 6.9: Gramática proposta para gerar uma arquitetura mista usando abordagens CNN e LSTM. Os módulos foram definidos de acordo com o *framework* Keras.

Todas as regras de construção da arquitetura propostas nas Seções 6.3.2 e 6.3.3 foram seguidas para esta proposta. Além disso, um conjunto de restrições foi adicionado à nova gramática, diminuindo o número de arquiteturas inválidas para cerca de 10% após os testes empíricos. As regras são:

- O modelo terá pelo menos uma CNN ou LSTM e uma camada densa;
- Foi construído um módulo de ajuste de representação entre camadas CNN e LSTM para manter a correção de transferência de dados.

A interface entre as camadas CNN e LSTM é um ponto a se destacar na abordagem. Vale ressaltar que os modelos baseados em CNN em geral trabalham com uma representação em matriz, e os modelos baseados em LSTM tradicionalmente utilizam vetores como entrada. Neste aspecto, as camadas LSTM foram adaptadas para apenas receberem entradas no formato de matrizes. Ainda sim, em alguns experimentos foram criadas redes neurais inválidas. Para o problema foram levantados duas resolutivas:

- Descartar as redes inválidas e gerar uma rede nova aleatoriamente
- Penalizar a solução com uma rede inválida

Para os experimentos iniciais, foi considerada a primeira solução. Entretanto, essa saída se mostrou ruim, pois em muitos casos genes já evoluídos por muitas gerações eram totalmente perdidos. Para os experimentos finais foi adotada a segunda opção, que garante que os genes da solução ruim sejam mantidos por algumas gerações seguintes.

6.3.5 Gramática GNN

A gramática mostrada na Figura 6.10 combina todos os componentes que podem ser usados para projetar GNNs. Essa gramática é capaz de combinar diferentes camadas convolucionais de grafos em uma única arquitetura de rede neural. Inclui os tipos de camadas

convolucionais e camadas de *dropout*, bem como seus parâmetros, por exemplo, o número de unidades, função de ativação e taxa de *dropout*.

Foram aplicadas restrições semelhantes às apresentadas nas Seções 6.3.2, 6.3.3 e 6.3.4, diminuindo a criação de arquiteturas inválidas ou ineficientes.

```

⟨gnn⟩ ::= ⟨conv⟩ ⟨layer_c⟩ ⟨dense⟩

⟨layer_c⟩ ::= ⟨conv⟩ | ⟨dropout⟩ | ⟨layer_c⟩ ⟨layer_c⟩ | &

⟨dense⟩ ::= conv ⟨conv_type⟩ ⟨units⟩ softmax

⟨conv⟩ ::= conv ⟨conv_type⟩ ⟨units⟩ ⟨activation⟩

⟨conv_type⟩ ::= GraphConv | GraphSageConv | ARMAConv
               | GraphAttention | APPNP

⟨dropout⟩ ::= dropout ⟨rate⟩

⟨units⟩ ::= 32 | 64 | 128 | 256

⟨activation⟩ ::= relu | sigmoid

⟨rate⟩ ::= [0.0, 0.25, 0.5]

```

Figura 6.10: Gramática proposta para gerar arquiteturas GNN. Os módulos foram definidos de acordo com os *frameworks* Keras e Spektral (<https://graphneural.network/>)

O bloco *conv_type* apresentado na Figura 6.10 utiliza diferentes tipos de convoluções amplamente presentes na literatura. Esses tipos de camadas de convolução são apresentados na Tabela 6.1. A principal hipótese é que cada camada convolucional apresentada nesta tabela pode extrair características específicas do conjunto de entrada, destacando diferentes partes do grafo de entrada. A combinação desses fatores pode trazer um ganho de desempenho quando aplicada à classificação de textos. Essa abordagem segue duas etapas: construir uma representação de textos e projetar uma rede neural adaptada ao domínio do conjunto de dados de entrada.

Tabela 6.1: Métodos de camada de convolução em grafos

Camada	Descrição	Autor(es)
GraphConv	Usa matriz de adjacência com auto-loops adicionados e uma matriz de grau para propagar nós discriminantes	[55]
GraphSageConv	Usa uma função para agregar a vizinhança de um nó.	[35]
ARMAConv	Camada convolucional de grafos com base em um filtro de grafo racional	[12]
GraphAttention	Essa camada calcula uma convolução semelhante ao GraphConv. No entanto, ele usa o mecanismo de atenção para pesar a matriz de adjacência em vez de usar o Laplaciano normalizado	[118]
APPNP	Uma camada convolucional de grafo implementando o operador APPNP	[56]

A validação da abordagem TextDSGE é descrita na Seção 7.0.3. Os modelos baseados em representações tradicionais (vetores), em grafos e baseados no BECA (Seção 7.0.2) são apresentados como entrada.

6.4 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentados os modelos propostos neste trabalho. O BECA, primeiro modelo apresentado, recebe como entrada um conjunto de textos não rotulados e usando técnicas de representação de palavras (como Word2Vec) e métodos de agrupamento é capaz de gerar uma representação de texto, utilizando menos palavras do que o dataset original oferece.

Em seguida foi apresentado o TextDSGE, um modelo de evolução de gramática para arquitetura de redes neurais profundas. O modelo foi adaptado para gerar redes neurais convolucionais (CNN's), recorrentes (LSTM's), baseada em grafos (GCN's) e um misto entre CNN e LSTM. A hipótese é que esse mecanismo pode se adaptar melhor a conjuntos de textos de domínios do conhecimento diversos.

7 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os procedimentos de validação dos modelos BECA e TextDSGE apresentados no Capítulo 6. São descritos os *datasets* e as formas de representação utilizadas, parametrização dos algoritmos, resultados encontrados e comparação com modelos de representação e classificação de textos presentes na literatura. Além disso, são desenvolvidos testes de normalidade e estatísticos, usados para decidir se os dados disponíveis suportam suficientemente uma hipótese de eficiência dos modelos apresentados.

7.0.1 *Datasets*

Nos experimentos descritos neste trabalho, foram utilizados quatro *benchmarks* de classificação de texto amplamente testados na literatura: Ohsumed, R8, R52 e Movie Review (MR). O conjunto de dados Ohsumed inclui resumos médicos das categorias MeSH de 1991. Foi utilizado um subconjunto com 7.400 sentenças. O conjunto de dados foi categorizado em 23 classes diferentes representando 23 doenças cardiovasculares, por exemplo:

Infection in total joint replacement. Although a small number of infections in total joint replacements are blood borne from distant sources, most infections appear to have been derived at operation. Strenuous attempts to reduce this risk by cleaning the air in the wound environment, coupled with prophylactic antibiotics [...]

Os conjuntos de dados R8 e R52 são coleções que apareceram no noticiário da Reuters em 1987. Os documentos foram reunidos e indexados a categorias pela própria equipe da Reuters Ltd. R8 é um subconjunto do conjunto de dados original com 7.674 sentenças categorizadas em 8 classes. R52 também é um subconjunto do conjunto de dados original com 9100 sentenças categorizadas em 52 classes. A seguir, está um exemplo de uma frase R52:

Japan india conference cuts gulf war risk charges the japan india pakistan gulf japan shipping conference said it would cut the extra risk insurance surcharges on shipments to iranian and iraqi ports to a minimum three pct from pct on october it said surcharges on shipments of all break [...]

MR é um conjunto de dados usado em experimentos de análise de sentimentos. Ele tem uma coleção de resenhas de filmes rotuladas de acordo com a polaridade geral do sentimento (positiva ou negativa). Este conjunto de dados tem 10.662 revisões. Segue abaixo um exemplo de frase:

The film is a fierce dance of destruction . its flame-like , roiling black-and-white inspires trembling and gratitude.

A Tabela 7.1 resume os valores encontrados para cada *dataset*. As colunas Nós e Arestas representam a quantidade de nós e de arestas de cada dataset usando a representação baseada em grafos apresentada na Seção 6.2.2. O número de palavras é potencialmente diferente do número de nós devido a duas razões principais: 1) nos modelos estudados, cada documento/sentença é também representado por um nó e 2) nem todas as palavras presentes no *dataset* possuem uma representação através de um nó, devido principalmente a razões de baixa frequência.

Tabela 7.1: Descrição estatística de cada dataset

Dataset	Nº Sentenças	Nº Palavras	Nº Nós	Nº Arestas	Nº Classes
MR	10,662	20,473	31,135	1,665,491	2
R8	7,674	6,792	14,466	4,024,898	8
R52	9,100	7,890	16,990	5,118,159	52
Ohsumed	7,400	13,323	20,723	8,384,734	23

7.0.2 BECA

Com o objetivo de validar a abordagem BECA, foram conduzidos os experimentos descritos nessa seção. Os conjuntos de dados de entrada são os descritos na Seção 7.0.1. Para avaliação do modelo foram geradas cinco representações, descritas a seguir:

- representação vetorial utilizando TF-IDF (descrita na Seção 2.1.2);
- representação vetorial utilizando o Sentence-BERT (descrita na Seção 3.2.1)¹
- representação vetorial baseada no BECA (descrita na Seção 6.2.1);
- representação em grafos (seguindo a representação proposta em [122]);
- representação em grafos baseada no BECA (descrita na Seção 6.2.2).

Ainda, para fins de validação, foram selecionados dois modelos de classificação de texto, LSTM² e GCN³. O objetivo principal é, sob as mesmas condições de recursos, verificar quais das representações guardam maior poder contextual e semântico e não verificar quais dos dois modelos é melhor. O algoritmo BECA foi executado com os parâmetros descritos na Tabela 7.2. Realizaram-se 10 execuções independentes utilizando a técnica de *4-folder cross-validation*⁴.

Tabela 7.2: Parâmetros da geração de representação usando o BECA para todos os datasets

Parâmetro	Valor	Parâmetro	Valor
Número de rodadas (k)	30	Taxa de Aprendizado η	$1e^{-4}$
Número de palavras em S	3	Algoritmo de Agrupamento	k-Means [75]

Os resultados para o experimento utilizando as representações vetoriais e o modelo LSTM são apresentados na Tabela 7.3. São reportados a acurácia média do modelo LSTM bem como o desvio padrão (entre parênteses). Os resultados apontam que o *Sentence-BERT* é suficiente para gerar representações coesas na maioria dos *datasets* testados. O detalhe é que para o *dataset* Ohsumed (que é de um domínio mais específico do que os demais), o BECA gerou resultados superiores.

Dessa forma, selecionaram-se os resultados sobre o *dataset* Ohsumed para realizar os testes estatísticos para este experimento. A partir dos resultados é possível realizar o teste

¹Para as representações de texto utilizadas na representação Sentence-BERT e BECA foi utilizado o modelo pré-treinado all-MiniLM-L6-v2, disponível em <https://huggingface.co/nreimers/MiniLM-L6-H384-uncased>

²A rede neural LSTM utilizada aqui segue a estrutura proposta por [7]

³A rede neural GCN utilizada aqui segue a estrutura proposta por [122]

⁴O cross-validation ou validação cruzada é um método de re-amostragem que usa diferentes partes dos dados para testar e treinar um modelo em diferentes iterações. Ele é usado principalmente em configurações em que o objetivo é a previsão e se deseja estimar a precisão com que um modelo preditivo funcionará na prática.

Tabela 7.3: Acurácia do modelo LSTM sob a perspectiva dos diferentes tipos de representação de texto para cada um dos datasets

Representação	Dataset			
	MR	R8	R52	Ohsumed
TFIDF	0.6925 (0.0064)	0.7899 (0.0211)	0.7982 (0.0063)	0.5601 (0.0165)
Sentence-BERT	0.7025 (0.0126)	0.8685 (0.0118)	0.8800 (0.0002)	0.6101 (0.0215)
BECA	0.6989 (0.0235)	0.8645 (0.0144)	0.8781 (0.0134)	0.6205 (0.0105)

de normalidade Shapiro-Wilk, demonstrado na Tabela 7.4, com o objetivo de determinar se a distribuição dos resultados é semelhante a uma distribuição normal. Os resultados mostram que as representações TF-IDF e *Sentence-BERT* tem um valor de p menor que 0.05, isto é, possuem uma distribuição próxima da normal, ao contrário da representação BECA. Conclui-se, portanto, que a distribuição do conjunto de todos os resultados dos modelos não são estatisticamente diferente de uma distribuição normal.

Tabela 7.4: Resultados do teste de normalidade Shapiro-Wilk para o modelo LSTM para o dataset Ohsumed

Modelo: LSTM e Dataset: Ohsumed	
Representação	p -value
TFIDF	0.030
Sentence-BERT	0.006
BECA	0.163

A Figura 7.1 demonstra a distribuição dos valores para modelos baseados no LSTM utilizando representações baseadas em TFIDF, Sentence-BERT e BECA. É possível perceber nessa amostra que a representação BECA alcança valores superiores às outras formulações. Entretanto, a proximidade entre os valores de Sentence-BERT e BECA pode indicar que não existem diferenças estatísticas entre essas duas representações.

Ao se realizarem os teste de Kruskal-Wallis sobre a amostra de resultados apresentados, encontra-se um p -value inferior a 0.001, o que indica que o grupo possui diferenças estatísticas. Por fim, a Tabela 7.5 mostra as comparações par-a-par *Dwass-Steel-Critchlow-Fligner* entre as representações verificadas para o dataset Ohsumed. Ainda que os valores da representação BECA sejam superiores aos outros modelos, é importante notar que ele é estatisticamente igual à representação Sentence-BERT.

Tabela 7.5: Comparações par-a-par Dwass-Steel-Critchlow-Fligner entre as representações para o dataset Ohsumed

Representação Origem	Representação Alvo	p
TFIDF	Sentence-BERT	< .001
TFIDF	BECA	< .001
Sentence-BERT	BECA	0.857

Os resultados para o experimento utilizando as representações em grafo e o modelo GCN são apresentados na Tabela 7.6. São reportados a acurácia média do modelo GCN bem como o desvio padrão (entre parênteses). Os resultados apontam que o BECA possui resultados melhores para todos *datasets* testados. A hipótese sobre a superioridade do BECA nesta base é

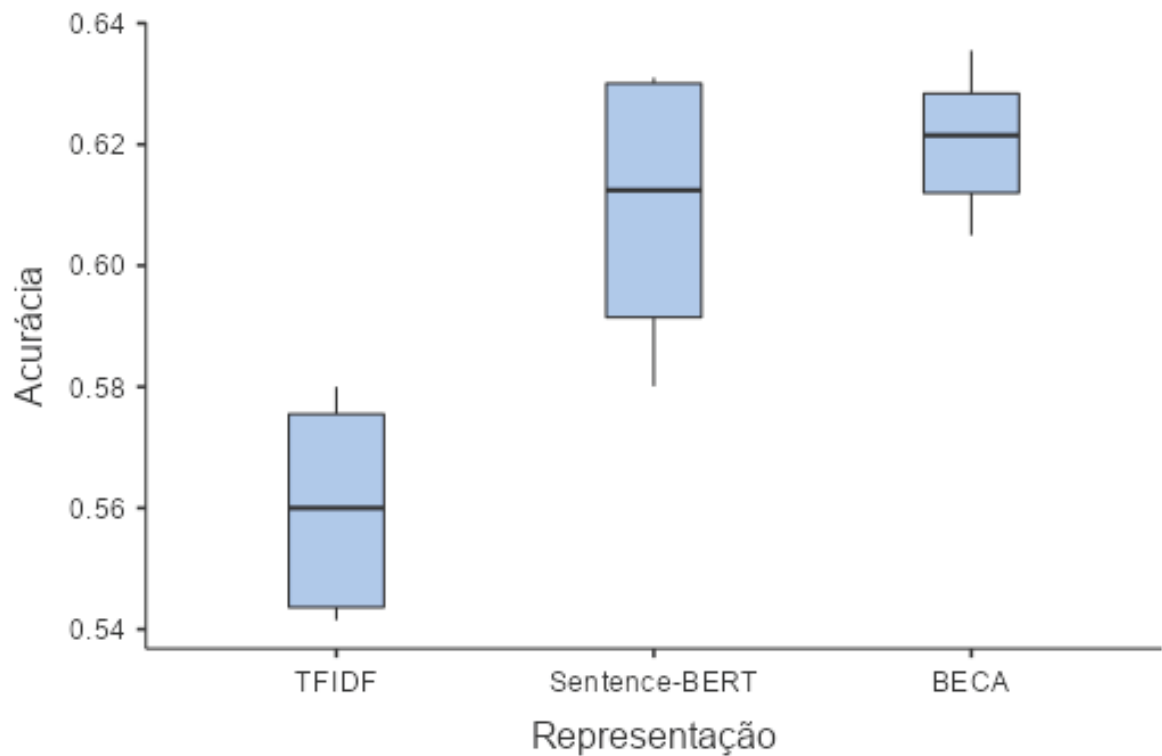


Figura 7.1: Distribuição dos valores para modelos LSTM utilizando representações baseadas em TFIDF, *Sentence-BERT* e BECA para o dataset Ohsumed

que o modelo é capaz de reduzir dinamicamente e eficientemente a quantidade de vocabulários pouco representativos do *dataset*.

Tabela 7.6: Acurácia do modelo GCN sob a perspectiva dos diferentes tipos de representação de texto para cada um dos datasets

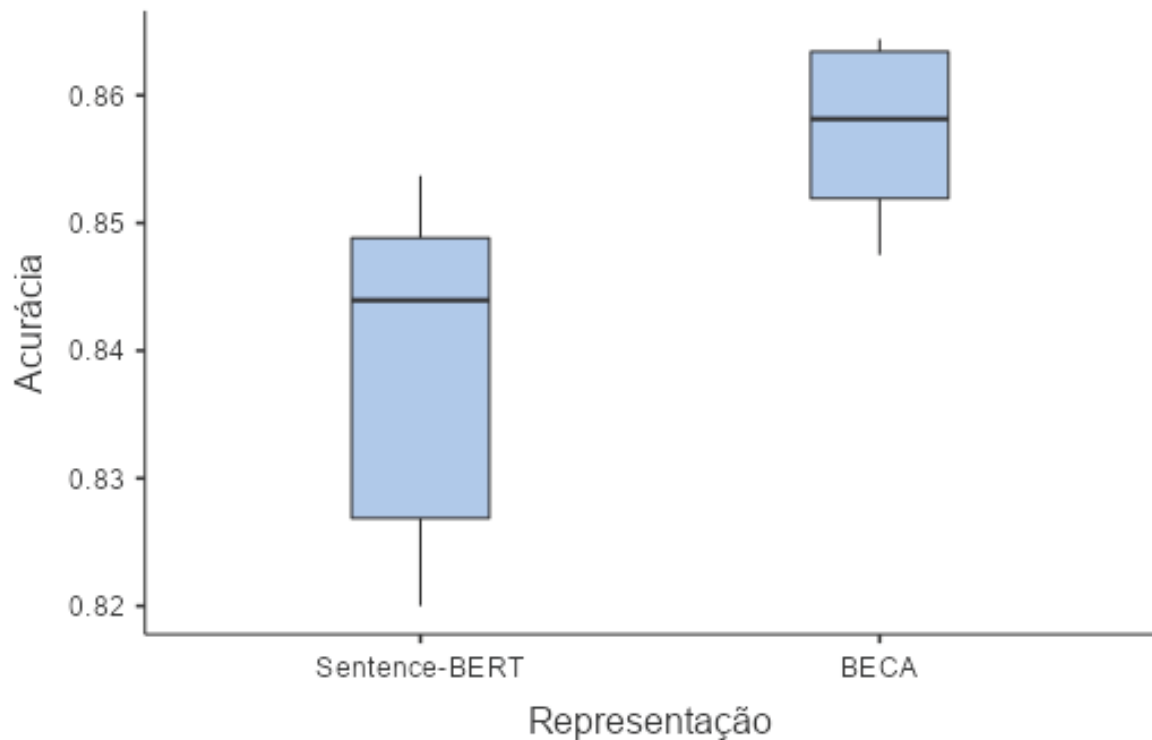
Representação	Dataset			
	MR	R8	R52	Ohsumed
Sentence-BERT	0.7200 (0.0044)	0.8390 (0.0132)	0.8100 (0.0066)	0.5500 (0.0037)
BECA	0.7223 (0.0011)	0.8573 (0.0066)	0.8225 (0.0090)	0.5749 (0.0257)

Como os resultados do BECA no *dataset* R8 foram aparentemente superiores ao *Sentence-BERT*, cabe a verificação estatística dos resultados nesse contexto. Realizando o teste de normalidade Shapiro-Wilk, demonstrado na Tabela 7.7, mais uma vez, os resultados do BECA não seguem uma distribuição normal, o que justifica o teste de Kruskal-Wallis. O teste estatístico aponta um *p-value* de 0.001 para o conjunto de resultados, indicando de fato que existe diferença estatística na amostra de resultados. Nesse contexto, realizando a comparação par-a-par *Dwass-Steel-Critchlow-Fligner* entre os dois modelos encontra-se um *p-value* no valor de 0.002, o que atesta a diferença estatística entre os modelos, evidenciado pela Figura 7.2.

Os resultados apontam um caminho promissor para o BECA como uma melhoria para modelos de representação baseados no *Sentence-BERT*. Mesmo que para alguns experimentos os valores alcançados sejam inferiores, ressalta-se que nestes casos não existe diferença estatística. Por outro lado, quando o BECA supera seu antecessor, consegue alcançar valores de fato

Tabela 7.7: Resultados do teste de normalidade Shapiro-Wilk para o modelo GCN para o dataset R8

Modelo: GCN e Dataset: R8	
Representação	<i>p-value</i>
Sentence-BERT	0.037
BECA	0.163

Figura 7.2: Distribuição dos valores para modelos GCN utilizando representações em grafos baseadas em *Sentence-BERT* e BECA para o dataset R8

estatisticamente superiores. Os resultados do BECA apontam para uma diminuição da necessidade de especialistas na remoção de palavras não relevantes ou com pouca carga semântica. Como demonstrado, o modelo BECA alcança valores de representação significativos por, removendo palavras pouco relevantes, diminui o viés que tais palavras podem causar na representação numérica de uma sentença.

Os modelos de representação aplicados aos *datasets* R8 e R52 indicam que para *datasets* com um vocabulário menor e com um conjunto final de palavras menor, os resultados são mais expressivos. Ressalta-se no entanto, que com uma quantidade menor de palavras é justificável um número de relações maior, como é descrito na Tabela 7.1, nos quais o BECA se mostrou mais vantajoso.

7.0.3 TextDSGE

Com o objetivo de validar a abordagem TextDSGE, foram conduzidos os experimentos descritos nesta seção. Os conjuntos de dados de entrada são os mesmos descritos na Seção 7.0.1. Os dados foram representados de cinco formas:

- representação vetorial utilizando TF-IDF (descrita na Seção 2.1.2);

- representação vetorial utilizando o *Sentence-BERT* (descrita na Seção 3.2.1)
- representação vetorial baseada no BECA (descrita na Seção 6.2.1);
- representação em grafos (seguindo a representação proposta em [122]);
- representação em grafos baseada no BECA (descrita na Seção 6.2.2).

As representações elencadas foram submetidas a diferentes tipos de modelos e arquiteturas de redes neurais (LSTM[7], CNN[119], GNN[122]) além dos modelos gerados pelo TextDSGE. A tabela 7.8 apresenta a configuração da busca evolutiva. Foram realizadas 10 execuções independentes, com uma população de 20 soluções, por 25 gerações. Além disso, os operadores genéticos incluem seleção por torneio, utilizando a recombinação *Half-and-Half* (descrita na Seção 6.3.1), com cruzamento de um ponto e mutação não-terminal e substituição com elitismo.

Durante a evolução do TextDSGE, cada modelo é treinado usando o otimizador Adam [54], com parâmetros padrão, por 50 épocas, e o *fitness* é calculado usando o conjunto de validação. No final, as 10 melhores arquiteturas de redes neurais encontradas são retreinadas com uma taxa de aprendizado de 0,001 e decaimento de peso de $5 * 10^{-4}$, por 500 épocas. A Tabela 7.8 apresenta a configuração da busca evolutiva. Os parâmetros foram determinados por análise empírica de resultados preliminares.

Tabela 7.8: Parâmetros da busca evolutiva utilizados nos experimentos com TextDSGE para todos os datasets

Parâmetro	Valor	Parâmetro	Valor
Tamanho da População	20	Gerações	25
Taxa de Cruzamento	60%	Taxa de Mutação	40%
Tamanho do Torneio	5	Número do Elitismo	25%
Quantidade de Épocas (TextDSGE/Refinamento)	50/500	Número de Execuções	10

Na Tabela 7.9, são exibidos a média de acurácia de todas as rodadas de todos os modelos projetados para esse experimento e, entre parênteses, o desvio padrão alcançado. Em geral, a aplicação de uma representação consolidada, como a do *Sentence-BERT*, melhora substancialmente os resultados dos modelos. Isso pode ser visto nos resultados de um modelo baseado em CNN (usando o TF-IDF como representação para cada sentença) e um modelo usando *Sentence-BERT* para gerar representações para o mesmo modelo.

Nos experimentos realizados, os modelos baseados em *Sentence-BERT* + LSTM tiveram melhor desempenho nos conjuntos de dados R8. Destacou-se que os modelos baseados em LSTM têm grande realce diante do restante dos experimentos e apresenta uma constância bem maior que os demais. Identificou-se que os modelos com arquiteturas construídas pelo TextDSGE apresentaram, em geral, uma melhoria relevante quando comparados aos mesmos modelos usando arquiteturas pré-definidas.

O tempo de execução é uma métrica importante para validação da abordagem. Desconsiderando quaisquer tarefas de geração de representação, a abordagem TextDSGE performa relativamente bem em relação ao tempo. A Tabela 7.10 demonstra o tempo de execução para cada um dos *datasets* testados.

Para exemplificar o uso da abordagem TextDSGE, selecionou-se o dataset MR como exemplo para as próximas inferências. A Figura 7.3 exhibe um exemplo de arquitetura de GNN desenvolvida pela abordagem. Nota-se que existe uma construção de rede neural convolucional em grafos com camadas mistas, isto é, de diferentes fundamentações teóricas. Na literatura,

Tabela 7.9: Resultados dos experimentos para cada dataset

Modelo	Representação	Dataset			
		MR	R8	R52	Ohsumed
CNN	TF-IDF	0.6810 (0.0078)	0.7280 (0.0037)	0.7801 (0.0033)	0.5105 (0.0237)
	Sentence-BERT	0.7000 (0.0081)	0.7520 (0.0125)	0.7570 (0.0055)	0.5710 (0.0110)
	BECA	0.7079 (0.0191)	0.7576 (0.0059)	0.7536 (0.0039)	0.6040 (0.0110)
LSTM	TF-IDF	0.6925 (0.0064)	0.7899 (0.0211)	0.7982 (0.0063)	0.5601 (0.0165)
	Sentence-BERT	0.7025 (0.0126)	0.8685 (0.0118)	0.8800 (0.0002)	0.6100 (0.0215)
	BECA	0.6989 (0.0235)	0.8645 (0.0144)	0.8781 (0.0134)	0.6205 (0.0105)
GCN	Sentence-BERT	0.7200 (0.0044)	0.8390 (0.0132)	0.8100 (0.0066)	0.5500 (0.0037)
	BECA	0.7223 (0.0011)	0.8573 (0.0066)	0.8225 (0.0090)	0.5749 (0.0257)
TextDSGE _{CNN}	TF-IDF	0.6964 (0.0044)	0.7393 (0.0051)	0.7869 (0.0046)	0.5180 (0.0256)
	Sentence-BERT	0.7000 (0.0137)	0.7325 (0.0029)	0.8080 (0.0057)	0.6004 (0.0018)
	BECA	0.7044 (0.0138)	0.7424 (0.0064)	0.8156 (0.0104)	0.6010 (0.0014)
TextDSGE _{LSTM}	TF-IDF	0.7022 (0.0114)	0.7803 (0.0216)	0.8048 (0.0126)	0.5664 (0.0155)
	Sentence-BERT	0.7120 (0.0062)	0.8500 (0.0009)	0.8698 (0.0074)	0.6215 (0.0063)
	BECA	0.7054 (0.0058)	0.8449 (0.0057)	0.8628 (0.0046)	0.6243 (0.0090)
TextDSGE _{GNN}	Sentence-BERT	0.7242 (0.0083)	0.8590 (0.0023)	0.8740 (0.0018)	0.6320 (0.0162)
	BECA	0.7378 (0.0052)	0.8594 (0.0057)	0.8818 (0.0028)	0.6403 (0.0203)
TextDSGE _{CNN+LSTM}	TF-IDF	0.6675 (0.0450)	0.7928 (0.0106)	0.8094 (0.0077)	0.6152 (0.0119)
	Sentence-BERT	0.7520 (0.0095)	0.8543 (0.0080)	0.8611 (0.0031)	0.6280 (0.0057)
	BECA	0.7570 (0.0053)	0.8600 (0.0082)	0.8593 (0.0074)	0.6364 (0.0055)

tipicamente as construções de redes neurais baseadas em grafos utilizam apenas uma técnica de convolução em grafos e são construídas empiricamente por seus arquitetos. A construção é baseada em conhecimento prévio sobre o *dataset*, quantidade de nós, quantidade de arestas. No modelo proposto essas discussões são dirimidas por um modelo automático e evolutivo de construção da rede, o que em geral, constrói redes adaptadas ao domínio do dataset.

Tabela 7.10: Tempo de execução da abordagem TextDSGE por rodada de execução para cada dataset

Modelo	Dataset			
	MR	R8	R52	Ohsumed
TextDSGE (médio)	13h	12h	18h	12h
TextDSGE (melhor)	13h	14h	19h	15h

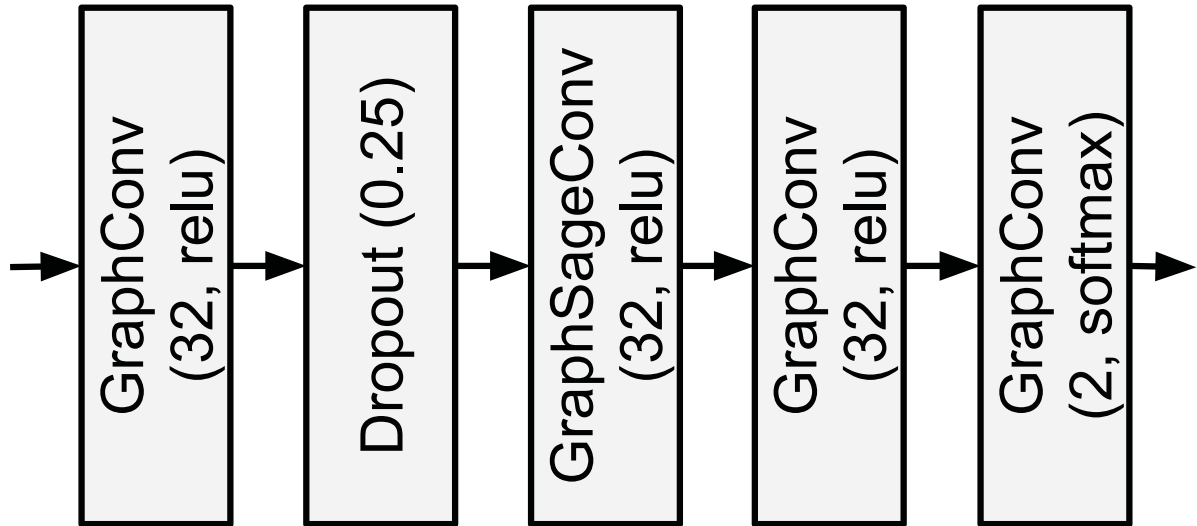


Figura 7.3: Exemplo de arquitetura de GNN desenvolvida pelo TextDSGE para o dataset MR.

Os resultados para o experimento são apresentados na Tabela 7.9. A partir dos resultados é possível realizar o teste de normalidade Shapiro-Wilk, demonstrado na Tabela 7.11. Esse teste tem como objetivo avaliar se uma distribuição é semelhante a uma distribuição normal. A distribuição normal também pode ser chamada de gaussiana. Os resultados mostram que apenas os modelos TextDSGE_{GNN} utilizando o *Sentence-BERT* e $\text{TextDSGE}_{LSTM+GNN}$ utilizando o TF-IDF têm um valor de p menor que 0.05, isto é, possuem uma distribuição próxima da normal. Conclui-se, portanto, que a distribuição de dados dos resultados dos modelos é estatisticamente diferente de uma distribuição normal. Desta forma, para avaliar os resultados, o teste de Kruskal-Wallis se faz necessário [57] a fim de verificar a diferença estatística entre os modelos.

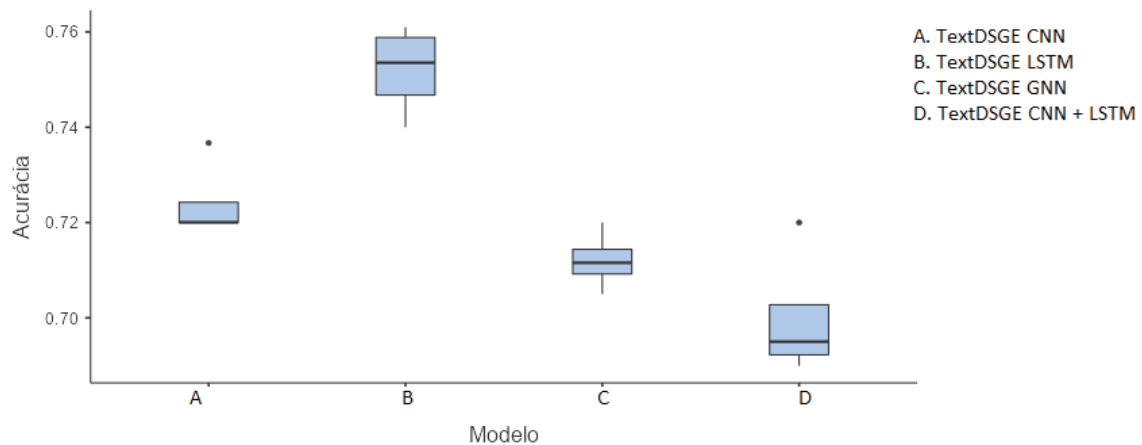
A Figura 7.4 demonstra a distribuição dos valores para modelos baseados no TextDSGE utilizando representações baseadas no *Sentence-BERT*. É possível perceber nessa amostra que o modelo TextDSGE_{LSTM} gera redes neurais superiores às outras formulações. A Figura 7.5 destaca a distribuição dos valores para modelos baseados no TextDSGE utilizando representações baseadas no BECA. Neste caso, percebe-se, nessa amostra, que o modelo $\text{TextDSGE}_{CNN+LSTM}$ gera redes neurais superiores às outras formulações. É importante destacar, para esse cenário, que o uso da representação BECA modificou o comportamento da arquitetura $\text{TextDSGE}_{CNN+LSTM}$, isso se deu, principalmente por que a representação estava muito mais refinada, o que possivelmente potencializou a aplicação das camadas CNN da arquitetura.

Uma vez que os modelos baseados em LSTM se mostraram mais promissores, faz-se necessária a análise comparativa entre os modelos propostos utilizando essa abordagem. A Figura 7.6 exibe um comparativo entre essa abordagem. É possível observar que os modelos baseados em $\text{TextDSGE}_{CNN+LSTM}$ se sobressaem em relação aos outros.

A análise seguinte diz respeito aos melhores resultados entre todos os conjuntos a partir do modelo de classificação de texto. A Figura 7.7 serve este comparativo. É importante frisar que os modelos baseados em TF-IDF, principalmente por não guardarem informação contextual

Tabela 7.11: Resultados do teste de normalidade Shapiro-Wilk para cada modelo para o dataset MR

Dataset: MR		
Modelo	Representação	p-value
CNN	TF-IDF	0.785
	Sentence-BERT	0.695
	BECA	0.119
LSTM	TF-IDF	0.466
	Sentence-BERT	0.903
	BECA	0.877
GCN	Sentence-BERT	0.689
	BECA	0.334
TextDSGE _{CNN}	TF-IDF	0.116
	Sentence-BERT	0.130
	BECA	0.976
TextDSGE _{LSTM}	TF-IDF	0.583
	Sentence-BERT	0.901
	BECA	0.104
TextDSGE _{GNN}	Sentence-BERT	0.001
	BECA	0.011
TextDSGE _{CNN+LSTM}	TF-IDF	0.001
	Sentence-BERT	0.640
	BECA	0.689

Figura 7.4: Distribuição dos valores para modelos TextDSGE utilizando representações baseadas no *Sentence-BERT* para o dataset MR

de seus dados, não figuram entre os melhores resultados. Todavia, os modelos de representação baseados no BECA demonstram excelentes resultados. Vale lembrar que o modelo BECA utiliza internamente o modelo *Sentence-BERT*, o que justifica sua alta performance.

Ao se realizarem os testes de Kruskal-Wallis sobre a amostra de resultados apresentados na Figura 7.7 encontra-se um p -value de 0.001, o que evidencia que esses modelos em conjunto possuem diferenças estatísticas gerais significativas. A Tabela 7.12 mostra as comparações par-a-par entre os modelos testados pela técnica *Dwass-Steel-Critchlow-Fligner pairwise comparisons*.

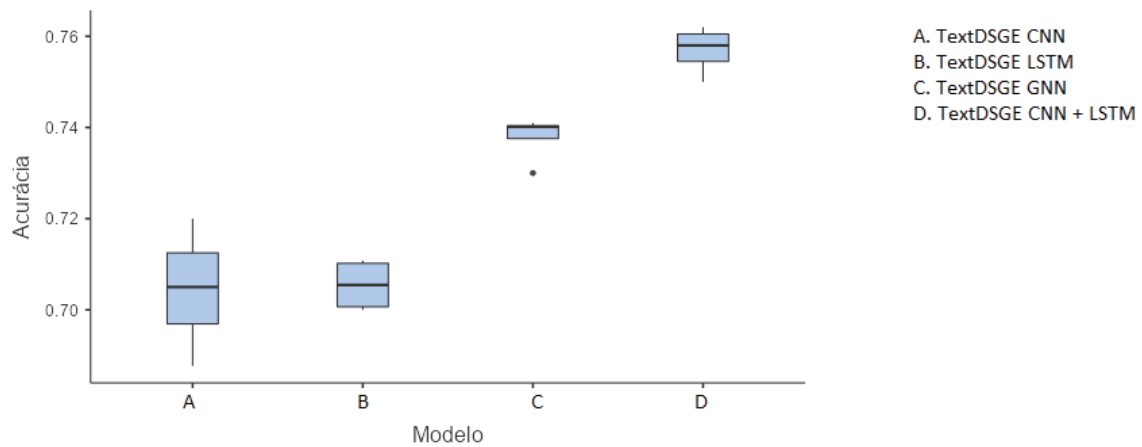


Figura 7.5: Distribuição dos valores para modelos TextDSGE utilizando representações baseadas no BECA para o dataset MR

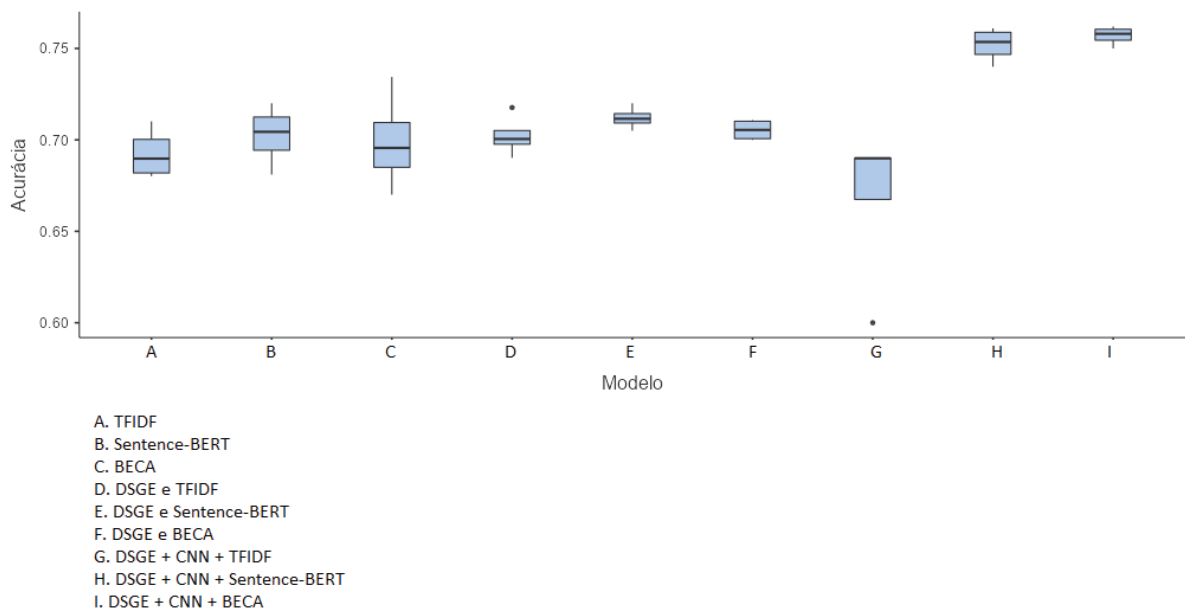


Figura 7.6: Comparação entre os modelos baseados em LSTM utilizando todos os tipos de representação para o dataset MR.

Ainda que os valores de alguns modelos sejam superiores a outros, é importante mencionar que não existem diferenças estatísticas evidentes quando os modelos são comparados par-a-par.

7.1 OBSERVAÇÕES FINAIS

Neste capítulo foram apresentadas as duas abordagens propostas: BECA e TextDSGE. A primeira busca encontrar representações de texto mais coesas e consistentes. A segunda propõe uma abordagem evolutiva para construção de redes neurais profundas (por vezes com camadas mistas) para aplicação na tarefa de classificação de texto.

Para o BECA, os experimentos demonstram que muito ainda tem de ser percorrido para o modelo superar o *status quo* nas tarefas de representação de texto, ainda que, em casos específicos, consiga ser estatisticamente igual aos seus concorrentes. Ressalta-se, no entanto, que em alguns cenários a proposta supera estatisticamente seu antecessor, o *Sentence-BERT*.

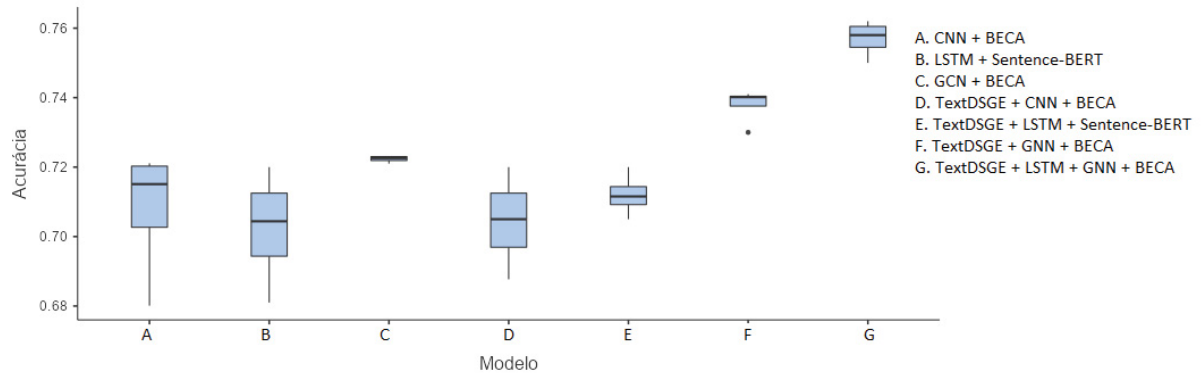


Figura 7.7: Comparação entre os melhores modelos de representação a partir de cada modelo de classificação para o dataset MR

Tabela 7.12: Comparações par-a-par Dwass-Steel-Critchlow-Fligner entre os melhores modelos para o dataset MR

Modelo	Representação	Modelo	Representação	p
CNN	BECA	LSTM	Sentence-BERT	0.991
		GCN	BECA	0.401
		TextDSGE _{CNN}	BECA	0.991
		TextDSGE _{LSTM}	Sentence-BERT	1.000
		TextDSGE _{GNN}	BECA	0.239
		TextDSGE _{CNN+LSTM}	BECA	0.239
LSTM	Sentence-BERT	GCN	BECA	0.239
		TextDSGE _{CNN}	BECA	1.000
		TextDSGE _{LSTM}	Sentence-BERT	0.951
		TextDSGE _{GNN}	BECA	0.239
		TextDSGE _{CNN+LSTM}	BECA	0.239
GCN	BECA	TextDSGE _{CNN}	BECA	0.239
		TextDSGE _{LSTM}	Sentence-BERT	0.239
		TextDSGE _{GNN}	BECA	0.239
		TextDSGE _{CNN+LSTM}	BECA	0.239
TextDSGE _{CNN}	BECA	TextDSGE _{LSTM}	Sentence-BERT	0.951
		TextDSGE _{GNN}	BECA	0.239
		TextDSGE _{CNN+LSTM}	BECA	0.239
TextDSGE _{LSTM}	Sentence-BERT	TextDSGE _{GNN}	BECA	0.239
		TextDSGE _{CNN+LSTM}	BECA	0.239
TextDSGE _{GNN}	BECA	TextDSGE _{CNN+LSTM}	BECA	0.239

Para o TextDSGE, percebe-se que sob as mesmas condições experimentais, e desconsiderando o tempo de execução do algoritmo, a abordagem tem promissores resultados. A proposta supera, em quase todos os cenários testados, os modelos clássicos de construção de redes neurais. O advento de uma metodologia de construção de redes mistas, por exemplo, unindo arquiteturas LSTM e CNN, ou mesmo diferentes tipos de camadas convolucionais de GNN, é uma contribuição relevante do TextDSGE para pesquisa em aprendizado profundo e, principalmente, nas tarefas de classificação de texto.

Revisitando as questões de pesquisa: QP1 perguntou se a abordagem de representação de texto é facilmente adaptada às diferentes datasets. Nos quatro *datasets* foi apresentados, cada um com suas características e domínios particulares, com requisitos e objetivos distintos, foram

alcançados valores de acurácia relevantes. QP2 pergunta se a abordagem de representação de texto proposta é comparável a outras técnicas de representação. Os experimentos descritos na Seção 7.0.2 comparam a abordagem BECA com modelos de representação de textos consolidados na literatura. Os resultados demonstram que a abordagem tem resultados competitivos em diversos cenários.

A QP3 pergunta se a abordagem de geração de redes neurais é facilmente adaptada para diferentes tipos de redes. Como foi demonstrado, o TextDSGE é facilmente adaptável a diferentes tipos de redes. No caso concreto, foram utilizados três diferentes modelos de arquitetura bem como foi possível mesclar diferentes modelos. QP4 questiona se as redes neurais projetadas automaticamente pelo TextDSGE podem ter um desempenho melhor ou semelhante aos projetos criados manualmente. Os experimentos descritos na Seção 7.0.3 respondem ao questionamento na medida em que os resultados apontam uma superioridade estatística das redes geradas quando comparadas com seus pares manuais.

Por fim, a QP5 pergunta sobre a novidade dessa abordagem. Apesar de as abordagens baseadas em gramática serem pouco exploradas, e ainda que abordagens neuro-evolucionais apresentem resultados na literatura superiores aos apresentados, cabe ressaltar que a mescla entre arquiteturas (LSTM + CNN) ou entre teorias convolucionais (diferentes tipos de camadas de convolução em grafos) configura uma novidade no processo de geração de redes neurais a partir de modelos de evolução gramatical. Além disso, as modificações e adaptações do modelo DSGE para utilizar representações de texto consistem em uma inovação do trabalho apresentado. O TextDSGE é então uma evolução natural do DSGE quando expande a sua utilização para modelos e arquiteturas não previstas originalmente.

Ainda sobre a QP5, é importante ressaltar o uso de um modelo de arquitetura de redes neurais para a construção específica de modelos CNN e LSTM, haja vista que determinar o projeto de uma rede não é uma tarefa trivial. Os modelos baseados em TextDSGE_{CNN} e TextDSGE_{LSTM} são capazes de construir redes neurais variavelmente complexas o que tornam a abordagem um aspecto importante para a pesquisa em NLP e Neuro-evolução.

8 APLICAÇÃO DO MODELO BECA+TEXTDSGE NO DOMÍNIO JURÍDICO

Nesta seção, os *frameworks* apresentados na Seção 6 são aplicados em um problema prático e real. São descritos aqui o contexto do problema, as adaptações realizadas e os experimentos executados.

8.1 CONTEXTO JURÍDICO

O judiciário brasileiro tem cerca de 77.1 milhões de processos judiciais em tramitação [26], esse número era de 78.6 milhões no ano de 2018 [27]. Esta redução de quase 2% representa um avanço sem precedentes em toda a série histórica segundo o Conselho Nacional de Justiça. Mesmo assim, o poder judiciário possui uma taxa de congestionamento de 68,5% demonstrando que existe uma crescente busca pelos serviços judiciários e um inevitável acúmulo de processos. Ainda nesta linha, a produtividade na área jurídica cresceu 14,1%, muito pelo avanço na capacitação dos servidores do Poder Judiciário, mas também com a implantação de ferramentas computacionais capazes de agilizar o trâmite, tais como PJe e DataJud.

Existe uma preocupação constante do Poder Judiciário com a baixa deste acervo, o que provoca inúmeras ações, como os mutirões de baixa e conciliação. Essa preocupação se reflete nas esferas estaduais, nas quais é constante a necessidade de evolução das metodologias no trato jurídico. Ressalta-se, neste contexto, que a baixa de processos (a resolução do conflito) é uma área de rica complexidade, em que a expertise jurídica altamente sofisticada se funde com a competência cognitiva e emocional, sendo um processo não estritamente subjetivo, mas que conta com a capacidade de discernimento e análise argumentativa do operador do direito e, em muitos casos, a tomada de decisão é uma combinação entre fatos e argumentos jurídicos.

Devido ao o crescente número de processos judiciais e à busca cada vez mais comum pela justiça, reforça-se o discurso de que a justiça brasileira não possui recursos suficientes para sanar o déficit de baixa de processos judiciais com o *modus operandi* atual, mesmo com o corrente esforço em todas as jurisdições. É consenso, portanto, que a tecnologia pode ajudar a amenizar esse quadro, principalmente utilizando técnicas de inteligência artificial, combinando inúmeras tarefas já consolidadas em outras áreas do domínio do conhecimento. A área de NLP é uma forte candidata na otimização do trato jurídico. O uso do processamento de texto, com técnicas de agrupamento, sumarização, extração de informações e classificação, pode, em última instância, auxiliar o operador do direito no processo de construção de argumentação jurídica ou mesmo no decisório.

A fim de validar as abordagens propostas no Capítulo 6, propõe-se a aplicação dos *frameworks* BECA e TextDSGE em um problema latente no Poder Judiciário: a classificação de petições iniciais. Em termos gerais, uma petição inicial é o início de todo processo jurídico. Portanto, é o documento mais importante de toda a disputa judicial, pois é nela que os fatos são apresentados para o julgador. Neste documento, o advogado leva a juízo os problemas jurídicos de quem está representando [13].

Na petição inicial, o advogado descreve o que a parte deseja, quais são os fatos e os direitos que embasam esse desejo e disponibiliza espaço para que a outra parte se manifeste, por meio da contestação, contrariando ou negando as acusações. É a petição inicial que define a narrativa dos acontecimentos, quais documentos e provas serão necessárias e, principalmente, qual é a **questão abordada** no jurídico pela parte que entrou com o pedido [13].

Atualmente, na maioria dos Tribunais de Justiça é utilizado o sistema eletrônico *Processo Judicial Eletrônico* - PJe. Nos processos submetidos no PJe, o advogado inicia o trâmite do processo inserindo informações básicas das partes (polo ativo - que provoca o processo - e polo passivo - que sofre a ação/processo), determina jurisdição, comarca, juizado, dentre outras informações. Além disso, é competência do advogado informar a Classe Judicial e os Assuntos Processuais envolvidos no processo. Estes elementos em particular são de suma importância para a distribuição, condução e diligências do processo.

Neste contexto, a escolha de classe e assuntos para o processo judicial é de vital importância para a continuidade, organização e celeridade do processo judicial, fazendo com que este consiga seguir o rito condizente com suas características e avaliado pela pessoa mais capacitada para fazê-lo (desde que não fira o princípio do juiz natural. Decorre que esta seleção tão cabal para o processo judicial, por vezes, é negligenciada. Não raro, os processos judiciais são classificados de forma incorreta, seja por:

- Atribuir classe judicial errada ao tema central do processo;
- Atribuir classe judicial/assunto processual genérico demais ao tema central do processo;
- Uso de classes judiciais que são tendenciosas ao trâmite processual;
- Atribuição de assuntos processuais não correlatos, ou pouco discriminantes do processo.

Os problemas apresentados são, em certa medida, diluídos com o processo de conferência pelos secretários de varas e outros servidores que realizam a triagem antes da sua efetiva análise. O que não exclui a existência de processos ainda classificados erroneamente, seja pelo volume de triagem ou por falta de conhecimento prévio. Neste contexto específico, propõe-se abordagem de classificação de petições iniciais.

O objetivo específico desta aplicação é criar um modelo de inteligência artificial baseado em NLP que seja capaz de identificar padrões e conexões semânticas e contextuais entre as petições iniciais que já passaram pelo processo de triagem (isto é, que possuem uma classe e assunto relativamente coesos) e identificar se a questão abordada pelo processo é do tipo **Consignatório**. O foco principal é, então, utilizar as ferramentas BECA e TextDSGE nesta atividade, comparando-as com modelos e metodologias amplamente discutidos neste trabalho.

8.2 METODOLOGIA

Para realizar os experimentos, foi utilizada a metodologia descrita nessa seção. Primeiramente, os dados foram coletados do sistema PJe no total de 6 mil processos judiciais, dentre os quais 3 mil são do assunto jurídico "Empréstimo Consignado" e 3 mil são de assuntos diversos. Para conduzir um experimento mais acurado, foram escolhidas apenas petições iniciais com, pelo, menos 3 mil caracteres, sem conteúdo html e sem imagens.

Os dados foram separados em dois conjuntos idênticos, o primeiro foi pré-processado de acordo com [53] e o segundo foi convertido em representação vetorial utilizando a abordagem BECA. Em seguida, para cada conjunto foram criados *4-folders* contendo cada um deles 1800 instâncias de treino, 600 de validação e 600 de teste. Os conjuntos de instâncias foram representados das seguintes formas:

- representação vetorial utilizando TF-IDF (descrita na Seção 2.1.2);

- representação vetorial utilizando o Sentence-BERT (descrita na Seção 3.2.1)¹
- representação vetorial baseada no BECA (descrita na Seção 6.2.1);
- representação em grafos (seguindo a representação proposta em [122]);
- representação em grafos baseada no BECA (descrita na Seção 6.2.2).

8.3 EXPERIMENTOS E RESULTADOS

Com o objetivo de validar a abordagem BECA e TextDSGE, foram conduzidos os experimentos descritos nesta seção.

Os conjuntos de dados de entrada foram descritos na seção anterior e seguem todos os processamentos descritos na Seção 2.1 e 6.2.2. As representações elencadas foram submetidas a diferentes tipos de modelos e arquiteturas de redes neurais (LSTM[7], CNN[119], GNN[122]) além de modelos gerados pelo TextDSGE. A configuração evolutiva do TextDSGE segue a Tabela 7.8. Foram realizados todos os procedimentos descritos nas Seções 7.0.2.

Na Tabela 8.1 são exibidos a média de acurácia de todas as rodadas de todos os modelos projetados para esse experimento e, entre parênteses, o desvio padrão obtido. Os resultados apontam uma superioridade dos modelos baseados em LSTM, em especial destaca-se a abordagem TextDSGE, capaz de gerar redes LSTM mais ajustadas ao problema de classificação de petições iniciais. Os dados ainda indicam uma margem segura em relação ao uso do BECA como ferramenta de representação de texto. A Tabela 8.2 demonstra o teste de normalidade Shapiro-Wil para modelos de classificação usando *dataset* de petições iniciais. É possível observar que os modelos baseados em CNN e TextDSGE_{LSTM} possuem *p – value* menor que 0.05, indicando uma distribuição normal. Como o restante dos modelos e experimentos não o possuem, existe a indicação de diferença estatística.

Tabela 8.1: Resultados dos Experimentos para o dataset de petições iniciais

Modelo de Classificação	Representação		
	TFIDF	Sentence-BERT	BECA
CNN	0.6376 (0.0451)	0.6431 (0.0443)	0.6476 (0.0437)
LSTM	0.6806 (0.0127)	0.7005 (0.0082)	0.7071 (0.0087)
GNN	0.5468 (0.0103)	0.5517 (0.0115)	0.5554 (0.0117)
TextDSGE _{CNN}	0.6662 (0.0038)	0.6710 (0.0048)	0.6760 (0.0061)
TextDSGE _{LSTM}	0.6741 (0.0079)	0.7057 (0.0091)	0.7121 (0.0085)
TextDSGE _{GNN}	0.5480 (0.0118)	0.5576 (0.0099)	0.5644 (0.0095)
TextDSGE _{CNN+LSTM}	0.6866 (0.0026)	0.6921 (0.0035)	0.6981 (0.0048)

¹Para as representações de textos geradas pelo Sentence-BERT BECA foi utilizado o modelo pré-treinado paraphrase-multilingual-mpnet-base-v2, disponível em <https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2>

Tabela 8.2: Teste de distribuição normal para modelos de classificação usando dataset de petições iniciais

Modelo de Classificação	p-value
CNN	<0.001
LSTM	0.311
GNN	0.315
TextDSGE _{CNN}	0.523
TextDSGE _{LSTM}	0.014
TextDSGE _{GNN}	0.109
TextDSGE _{CNN+LSTM}	0.290

A Figura 8.1 exibe uma visão geral das distribuições de todos os valores para todos os modelos. A Figura 8.2 demonstra a distribuição de valores dos modelos de classificação. Neste caso, fica claro que os modelos LSTM e TextDSGE_{LSTM} têm de fato, valores superiores aos demais. A Figura 8.3 mostra a distribuição dos valores para as três representações usadas, independentemente do modelo de classificação utilizado. A imagem mostra uma superioridade da representação BECA, mas sem larga vantagem, o que pode indicar que todas as representações estejam no mesmo conjunto estatisticamente semelhante.

Observando com mais detalhes a representação BECA, pode-se inferir (a partir da Tabela 8.3, que apenas a CNN possuem uma distribuição normal. Portanto, os modelos baseados no TextDSGE possui alguma diferença estatística entre si, e com os modelos clássicos. A Figura 8.4 demonstra a distribuição de valores de acurácia para os modelos de classificação utilizando a representação BECA. Ressalta-se que os modelos baseados em LSTM mantêm sua boa performance quando comparados com as outras representações.

Tabela 8.3: Teste Shapiro-Walk para os modelos de classificação utilizando a representação BECA.

Modelo de Classificação	p-value
CNN	<0.001
LSTM	0.845
GNN	0.735
TextDSGE _{CNN}	0.762
TextDSGE _{LSTM}	0.307
TextDSGE _{GNN}	0.104
TextDSGE _{CNN+LSTM}	0.678

Por fim, a Tabela 8.4 mostra a diferença estatística entre os modelos de classificação para o dataset de petições iniciais utilizando a representação BECA. Podem-se destacar dois pontos principais: 1) os modelos gerados pelo TextDSGE_{LSTM} têm superioridade em relação aos demais, além de possuírem diferença estatística e 2) A representação BECA é perfeitamente ajustável aos modelos baseados em LSTM como podemos inferir dos resultados, seja utilizando redes neurais LSTM ou geradas pelo TextDSGE, não existindo diferença estatística entre os mesmos.

8.4 OBSERVAÇÕES FINAIS

Neste capítulo, foi apresentada a aplicação das duas abordagens propostas: BECA e TextDSGE sobre um dataset real de petições judiciais. A hipótese é que os modelos são capazes

Tabela 8.4: Diferença estatística entre os modelos de classificação para o dataset de petições iniciais utilizando a representação BECA

Modelo de Classificação Original	Modelo de Classificação Alvo	p-value
CNN	LSTM	0.003
CNN	GNN	0.040
CNN	TextDSGE _{CNN}	0.007
CNN	TextDSGE _{LSTM}	0.003
CNN	TextDSGE _{GNN}	0.040
CNN	TextDSGE _{CNN+LSTM}	0.003
LSTM	GNN	0.003
LSTM	TextDSGE _{CNN}	0.003
LSTM	TextDSGE _{LSTM}	0.823
LSTM	TextDSGE _{GNN}	0.003
LSTM	TextDSGE _{CNN+LSTM}	0.175
GNN	TextDSGE _{CNN}	0.003
GNN	TextDSGE _{LSTM}	0.003
GNN	TextDSGE _{GNN}	0.641
GNN	TextDSGE _{CNN+LSTM}	0.003
TextDSGE _{CNN}	TextDSGE _{LSTM}	0.003
TextDSGE _{CNN}	TextDSGE _{GNN}	0.003
TextDSGE _{CNN}	TextDSGE _{CNN+LSTM}	0.003
TextDSGE _{LSTM}	TextDSGE _{GNN}	0.003
TextDSGE _{LSTM}	TextDSGE _{CNN+LSTM}	0.040
TextDSGE _{GNN}	TextDSGE _{CNN+LSTM}	0.003

de gerar representações de textos coesas (no primeiro caso) e redes neurais ajustadas ao domínio de aplicação (no segundo caso).

Para o BECA, os experimentos demonstram um ajuste fino da representação do texto, ainda que, em casos específicos, consiga ser estatisticamente igual aos seus concorrentes. Ressalta-se, no entanto, que, em alguns cenários, a proposta supera estatisticamente seu antecessor, o *Sentence-BERT*.

Os modelos gerados pelo TextDSGE, em resumo, alcançam resultados melhores que seus concorrentes. Entretanto, um olhar mais acurado sobre os resultados indica que não existe diferença estatística entre os modelos gerados e seus antecessores do mesmo tipo, isto é, por exemplo, redes do tipo LSTM e redes geradas pelo TextDSGE do tipo LSTM.

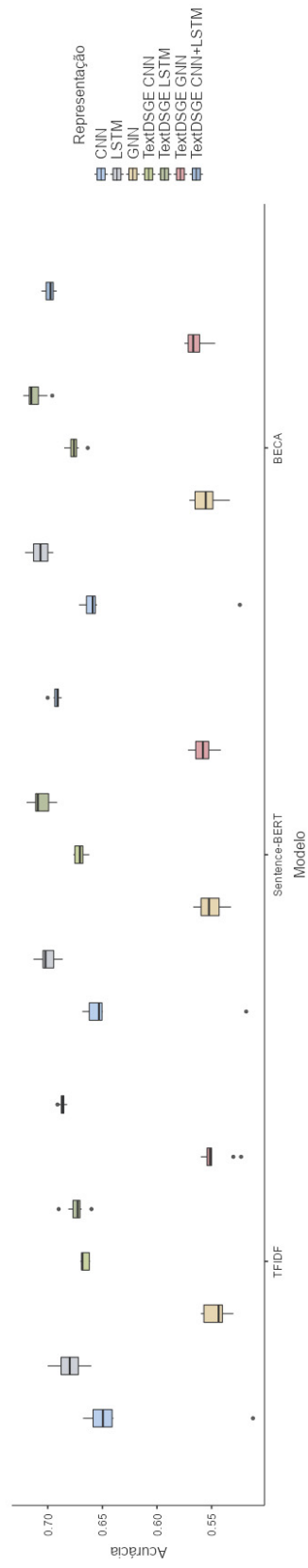


Figura 8.1: Visão geral das distribuições de valores para todos os modelos de classificação para todas as representações de textos utilizadas.

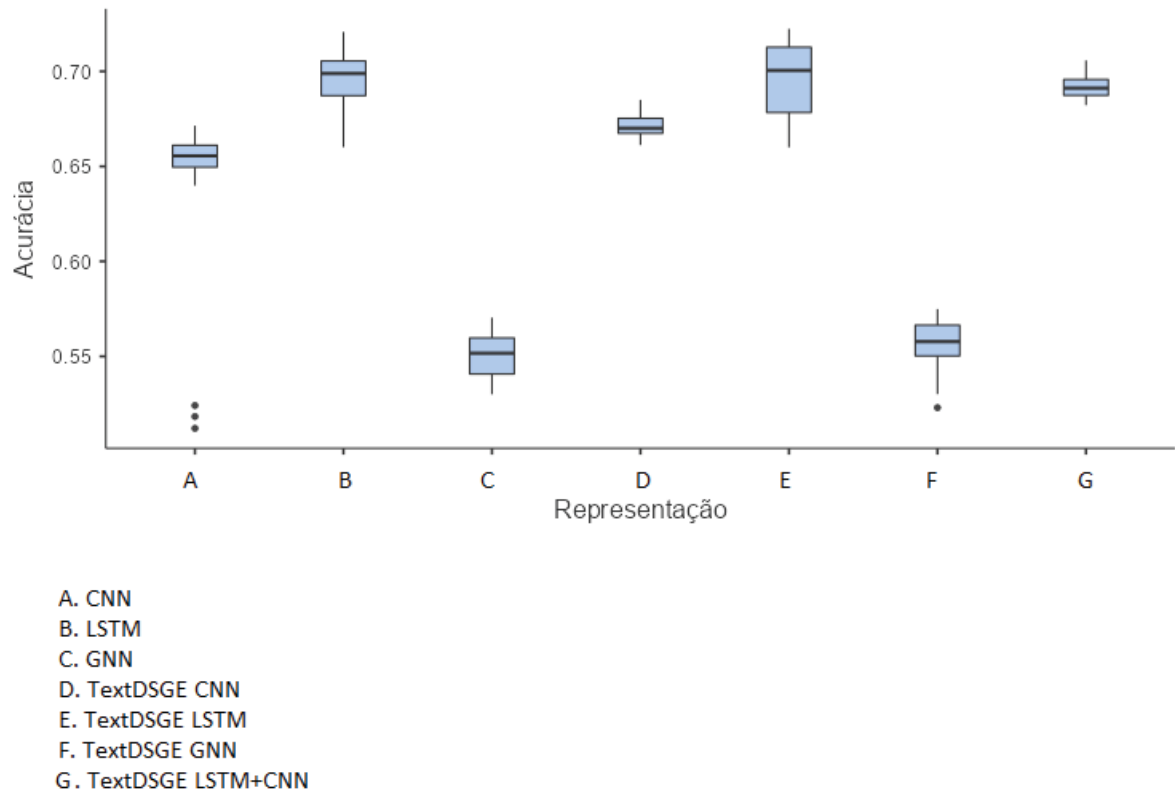


Figura 8.2: Distribuição de valores para modelos de classificação utilizando todas as representações para o dataset de petições iniciais.

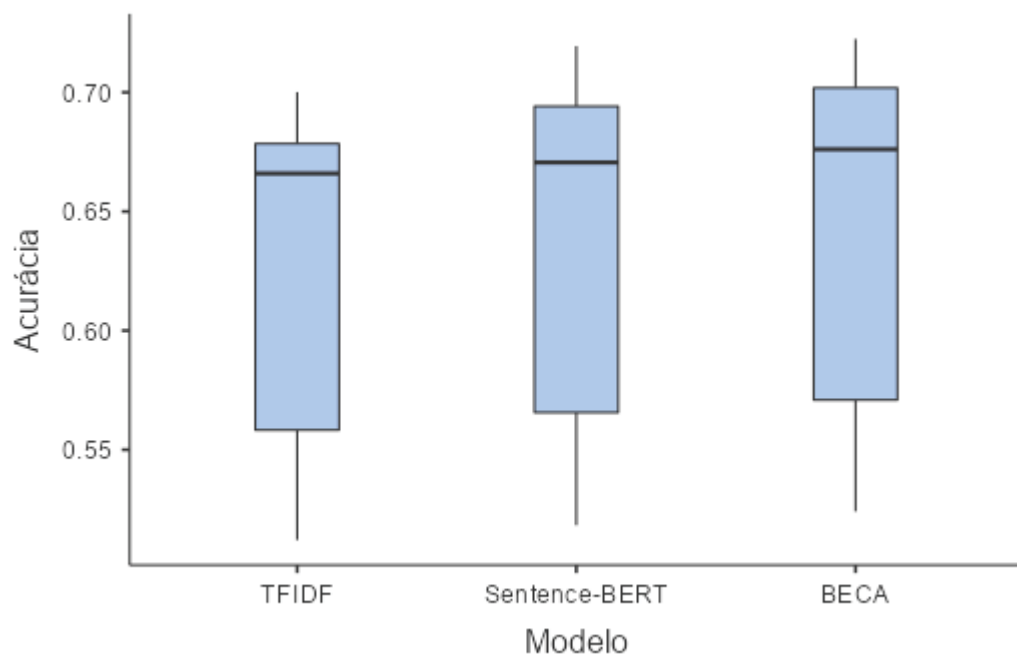


Figura 8.3: Distribuição dos valores para as três representações usadas, independentemente do modelo de classificação utilizado para o *dataset* de petições judiciais

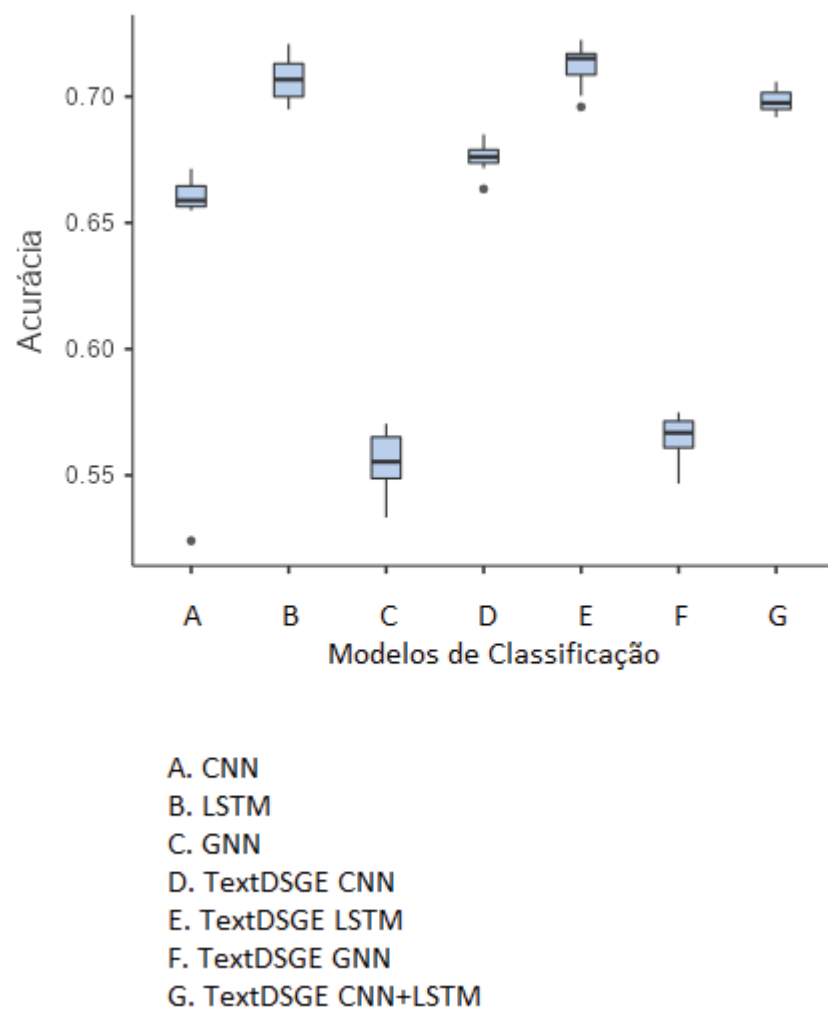


Figura 8.4: Distribuição de valores de acurácia para os modelos de classificação utilizando a representação BECA

9 CONSIDERAÇÕES FINAIS

Nesta seção são elencadas as contribuições do trabalho apresentado, produções técnicas e trabalhos submetidos e apresentados decorrentes direta ou indiretamente da pesquisa desenvolvida.

A NLP é uma área de pesquisa ativa, visto que a maioria das inovações e objetivos da Inteligência Artificial passam pelo estudo do entendimento da fala/escrita humana. Empenhar esforços no desenvolvimento de novas tecnologias, métodos e abordagens é uma tônica nesta área. As Redes Neurais Profundas adicionaram uma nova perspectiva. Mesmo redes neurais originalmente construídas para trabalhar com imagens (como é o caso da CNN) vêm sendo modificadas e adaptadas em prol da pesquisa em NLP.

Neste contexto, estruturas de redes neurais estão sendo continuamente modificadas e ajustadas por especialistas, aumentando a complexidade e o número de hiperparâmetros de tal forma que os ajustes finos estão cada vez mais complexos de ser realizados. Este desafio é parte da motivação para o crescente interesse na aplicação de técnicas para ajudar a projetar redes neurais profundas.

9.1 CONTRIBUIÇÕES E LIMITAÇÕES

No espectro apresentado, este trabalho propõe duas abordagens com o objetivo de estreitar relações entre três áreas de estudo: representação de texto, classificação de texto e projeto de redes neurais profundas. Foram descritos como uma representação de texto, parte precursora de todo modelo de NLP, pode ser construída a partir de diferentes perspectivas. Algumas subtarefas na representação de texto podem, portanto, ser substituídas por métodos inteligentes, tais como a remoção de palavras com baixa carga semântica.

No espectro do desenvolvimento de redes neurais profundas, destaca-se neste trabalho a combinação oportuna de técnicas de combinação existentes usadas para projetar algoritmos e novos *insights* para projeto de redes neurais profundas. Este trabalho explorou ainda o processo de codificação e mapeamento indireto, usado em DSGE, com o esquema evolutivo para geração, recombinação e avaliação de soluções.

É importante frisar que a abordagem TextDSGE é adaptável a problemas de classificação de texto de usando diferentes arquiteturas de redes e não está limitado aos modelos explorados aqui. Os modelos baseados em CNN, LSTM e GNN foram escolhidos, principalmente, por estarem na vanguarda das tarefas de classificação de texto, mas outras estruturas podem e devem ser exploradas a fim de encontrar resultados ainda mais promissores.

Na contramão dos resultados alcançados, salienta-se que o custo computacional das abordagens propostas superam as abordagens tradicionais, independentemente de um conjunto otimizado de parâmetros, equilibrando a quantidade de redes evoluídas.

As abordagens foram experimentadas em diferentes cenários, seja utilizando *benchmarks* (Capítulo 6) ou aplicadas em um problema real (Capítulo 8). Ainda que, em sua maioria, os resultados sejam estatisticamente semelhantes aos seus concorrentes, as abordagens mostram um desempenho superior, passível de reconhecimento. A lista a seguir exhibe as principais contribuições do framework BECA:

- Propôs um conjunto de regras e heurísticas para seleção e geração de representação de texto, de tal forma que diminui a necessidade de domínio do conhecimento do projetista de redes neurais, na medida que o modelo é capaz de determinar o melhor conjunto de palavras que representa uma dada sentença;

- Adicionou um valor semântico mais acurado aos datasets testados. Em termos gerais, o modelo consegue potencializar a carga semântica de algumas palavras de cada sentença, criando um viés cognitivo ao processo de classificação de texto;
- Utilizou técnicas consolidadas na literatura a fim de reduzir o tempo de pré-processamento de modelos textuais e em certa medida utilizou o conhecimento anterior em prol de uma continuada evolução no estudo de NLP;
- Diminuiu a necessidade tácita de especialistas na análise prévia e a posteriori de textos, já que, diferente de modelos passados, os modelos propostos aqui buscam inferir por si o que seriam de fato palavras importantes para a semântica do texto;
- Aplicou as abordagens a diferentes cenários de representação de texto, usando o conceito de *benchmarks* buscou explorar diversos espectros do conhecimento;

A lista a seguir apresenta um conjunto das principais limitações técnicas do framework BECA:

- O custo computacional do pré-processamento, ainda que executado apenas uma vez para um conjunto de modelos, pode ser um impeditivo do seu uso;
- Datasets com instâncias mais curtas podem gerar um viés semântico, tornando palavras com pouca ou nenhuma carga semântica importantes para o contexto da abordagem;
- O modelo está acoplado à ferramenta *Sentence-BERT*, o que o torna pouco maleável neste aspecto.

A lista a seguir apresenta o conjunto das principais contribuições do TextDSGE:

- Propôs uma estratégia baseada em gramáticas para geração de redes neurais profundas que combina regras para camadas, hiperparâmetros e arquiteturas;
- Propôs uma abordagem de geração de redes mistas, seja pela mescla de camadas CNN e LSTM, seja pela mescla de diferentes abordagens de convolução em grafos;
- Aplicou recentes operadores de recombinação explorados em [65] que otimizaram a codificação a partir do DSGE;
- Aplicou a abordagem em diferentes cenários de classificação de texto, mostrando-se flexível em todos eles;
- Demonstrou que é possível gerar automaticamente redes neurais profundas competitivas.

De forma análoga, a seguir são apresentadas as limitações do TextDSGE:

- Algumas redes neurais gerais sofrem de impeditiva complexidade;
- Cada cenário avaliado requer uma profunda análise particular;
- Atualmente, a abordagem não realiza validações ou evoluções dos pesos das redes neurais;
- O custo computacional é um critério questionável no uso da abordagem.

9.2 DIRECIONAMENTOS FUTUROS

Mais experimentos com as abordagens podem ajudar a reduzir os custos relacionados ao treinamento das redes. Cada experimento realizado aqui explorou diferentes recursos da abordagem, como poder de representação, adaptabilidade e flexibilidade. Outros domínios de aplicação (que não as *benchmarks* usados e o espectro jurídico) podem trazer à luz novas experiências, tendências e aplicações.

Realizando uma análise empírica, ficou claro que alguns termos nos experimentos (principalmente nos do Capítulo 8) têm baixa carga semântica, o que pode destacar uma tendência verborrágica dos textos. Esta característica deve ser mais bem explorada em experimentos futuros, principalmente destacando esses resultados como um produto do BECA.

Para o TextDSGE, foi utilizada a métrica de acurácia sobre o conjunto de validação como função de *fitness* para a evolução do DSGE. Outras métricas podem ser exploradas. Além disso, explorar a mistura de redes neurais com diferentes abordagens (tais como unir arquitetura convolucional com as baseadas em grafos) é algo a ser explorado. Finalmente, utilizar técnicas de *transfer learning*, evolução de partes rede neural separadamente e uso de conexões residuais são direcionamentos interessantes para a pesquisa.

9.3 PUBLICAÇÕES

A lista a seguir contém os trabalhos que foram gerados ao longo desta pesquisa, com trabalhos publicados em congressos e revistas de renome, nacionais e internacionais, da área de computação evolutiva e computação aplicada.

- [77] Dimmy K. Magalhães, Aurora T. R. Pozo, Roberto Santana. *An empirical comparison of distance/similarity measures for Natural Language Processing*. Anais do XVI Encontro Nacional de Inteligência Artificial e Computacional 2019.
- [76] Dimmy K. Magalhães, Aurora T. R. Pozo. *Classification of relationship in argumentation using graph convolutional network*. ADBIS, TPD and EDA 2020 Common Workshops and Doctoral Consortium 2020.
- [105] Ana P. Sodré, Luis E. Mochenski, Dimmy K. Magalhães, Aurora T. R. Pozo, Carmem S. Hara, Sidnei Machado. *COVID-19 Portal: Machine Learning Techniques Applied to the Analysis of Judicial Processes Related to the Pandemic*. European Conference on Advances in Databases and Information Systems 2021.
- [106] Ana P. Sodré, Luis E. Mochenski, Dimmy K. Magalhães, Cristina D. Aguiar, Aurora T. R. Pozo, Carmem S. Hara. *Comparing Alternative Storage Models for Words Extracted from Legal Texts*. Anais Estendidos do XXXVI Simpósio Brasileiro de Bancos de Dados 2021.
- [65] Ricardo H. R. Lima, Dimmy K. Magalhães, Aurora T. R. Pozo, Alexander Mendiburu, Roberto Santana. *A Grammar-based GP approach applied to the design of deep neural networks*. Journal of Genetic Programming and Evolvable Machines 2022.
- Dimmy K. Magalhães, Ricardo H. R. Lima, Aurora T. R. Pozo. *Creating Deep Neural Networks for Text Classification Tasks using Grammar Genetic Programming*. Journal of Applied Soft Computing 2022.

- Dimmy K. Magalhães, Aurora T. R. Pozo, Sidnei Machado. *Técnicas de Aprendizado de Máquinas Aplicadas à Classificação de Decisões Judiciais*. Revista de Estudos Empíricos em Direito 2022.

REFERÊNCIAS

- [1] Jianlin , You Fang, Xiaoping Yang, and Qian Wang. Research on text representation model integrated semantic relationship. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2736–2741. IEEE, 2015.
- [2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [3] Charu C. Aggarwal. *Machine Learning for Text*. Springer Publishing Company, Incorporated, 1st edition, 2018.
- [4] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. Towards the evolution of multi-layered neural networks: A dynamic structured grammatical evolution approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 393–400. ACM, 2017.
- [5] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. DENSER: Deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20(1):5–35, 2019.
- [6] Thomas Back and H-P Schwefel. Evolutionary computation: An overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29. IEEE, 1996.
- [7] Xuemei Bai. Text classification based on lstm and attention. In *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*, pages 29–32. IEEE, 2018.
- [8] Anupam Baliyan, Akshit Batra, and Sunil Pratap Singh. Multilingual sentiment analysis using rnn-lstm and neural machine translation. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 710–713. IEEE, 2021.
- [9] Roberto Camacho Barranco, Raimundo F Dos Santos, M Shahriar Hossain, and Monika Akbar. Tracking the evolution of words with time-reflective text representations. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2088–2097. IEEE, 2018.
- [10] Mohammad Ehsan Basiri, Shahla Nemati, Moloud Abdar, Erik Cambria, and U Rajendra Acharya. Abcdm: An attention-based bidirectional cnn-rnn deep model for sentiment analysis. *Future Generation Computer Systems*, 115:279–294, 2021.
- [11] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-sequence learning using gated graph neural networks. *CoRR*, abs/1806.09835, 2018.
- [12] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional ARMA filters. *CoRR*, abs/1901.01343, 2019.

- [13] Guto Brandao, Juliana Pimenta, Greycy Kelly Barreto Marques Dos Santos, Adamo M. Chiambiro, and Claudio Manoel Da Silva. Petição inicial: 5 dicas para elaborar, Sep 2021.
- [14] Alberto Cetoli, Stefano Bragaglia, Andrew D. O’Harney, and Marc Sloan. Graph convolutional networks for named entity recognition. *CoRR*, abs/1709.10053, 2017.
- [15] Da Chen and Hong Wang. Research on short text classification algorithm based on neural network. In *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 1726–1729. IEEE, 2018.
- [16] Noam Chomsky. *The logical structure of linguistic theory*. Springer, 1975.
- [17] Önder Çoban and Selma Aysşe Özel. An evaluation of character level n-gram termsets in text categorization. In *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, pages 1–6. IEEE, 2018.
- [18] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.
- [19] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetraault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8440–8451. Association for Computational Linguistics, 2020.
- [20] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.
- [21] Ricardo Henrique Remes de Lima, Aurora T. R. Pozo, Alexander Mendiburu, and Roberto Santana. Automatic design of deep neural networks applied to image segmentation problems. In Ting Hu, Nuno Lourenço, and Eric Medvet, editors, *Genetic Programming - 24th European Conference, EuroGP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7-9, 2021, Proceedings*, volume 12691 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2021.
- [22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
- [23] Shifei Ding, Chunyang Su, and Junzhao Yu. An optimizing bp neural network algorithm based on genetic algorithm. *Artificial intelligence review*, 36(2):153–162, 2011.
- [24] Chenxi Dong, Haoran Jia, and Cong Wang. Unsupervised learning for semantic representation of short text. In *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 475–478. IEEE, 2018.
- [25] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21, 2019.

- [26] Justiça em Números. Conselho nacional de justiça, 2017.
- [27] Justiça em Números. Conselho nacional de justiça, 2018.
- [28] Katti Faceli, Ana Carolina Lorena, João Gama, André Carlos Ponce de Leon Carvalho, et al. Inteligência artificial: Uma abordagem de aprendizado de máquina. 2011.
- [29] Huilong Fan and Yongbin Qin. Research on text classification based on improved tf-idf algorithm. In *2018 International Conference on Network, Communication, Computer Engineering (NCCE 2018)*. Atlantis Press, 2018.
- [30] David B Fogel, Lawrence J Fogel, and VW Porto. Evolving neural networks. *Biological cybernetics*, 63(6):487–493, 1990.
- [31] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.
- [32] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014.
- [33] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.
- [34] Yanbu Guo, Chen Jin, Weihua Li, Chen Ji, Yuan Fang, and Yunhao Duan. A novel method of text representation on hybrid neural networks. In Qingli Li, Lipo Wang, Mei Zhou, Li Sun, Song Qiu, and Hongying Liu, editors, *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, CISP-BMEI 2017, Shanghai, China, October 14-16, 2017*, pages 1–6. IEEE, 2017.
- [35] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.
- [36] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.
- [37] Bhat S Harish, Devanur S Guru, and Shantharamu Manjunath. Representation and classification of text documents: A brief review. *IJCA, Special Issue on RTIPPR (2)*, pages 110–119, 2010.
- [38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [39] Peng Huaijin, Wang Jing, and Shen Qiwei. Improving text models with latent feature vector representations. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 154–157. IEEE, 2019.
- [40] Gartner Inc. Gartner, inc., 2019.
- [41] T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1):1793–8201, 2011.

- [42] Bingbing Jiang, Zhengyu Li, Huanhuan Chen, and Anthony G Cohn. Latent topic text representation learning on statistical manifolds. *IEEE transactions on neural networks and learning systems*, 29(11):5643–5654, 2018.
- [43] Suqi Jiang, Jason Lewris, Michael Voltmer, and Hongning Wang. Integrating rich document representations for text classification. In *2016 IEEE Systems and Information Engineering Design Symposium (SIEDS)*, pages 303–308. IEEE, 2016.
- [44] Renlong Jie and Junbin Gao. Differentiable neural architecture search for high-dimensional time series forecasting. *IEEE Access*, 9:20922–20932, 2021.
- [45] Taeho Jo. Using k nearest neighbors for text segmentation with feature similarity. In *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pages 1–5. IEEE, 2017.
- [46] Taeho Jo. String vector based knn for text categorization. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 438–443. IEEE, 2018.
- [47] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nedellec and Céline Rouveirol, editors, *Machine Learning: ECML-98, 10th European Conference on Machine Learning, Chemnitz, Germany, April 21-23, 1998, Proceedings*, volume 1398 of *Lecture Notes in Computer Science*, pages 137–142. Springer, 1998.
- [48] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.
- [49] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.
- [50] Tom Kenter, Alexey Borisov, and Maarten de Rijke. Siamese CBOW: optimizing word embeddings for sentence representations. *CoRR*, abs/1606.04640, 2016.
- [51] Aurangzeb Khan, Baharum Baharudin, Lam Hong Lee, and Khairullah Khan. A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, 1(1):4–20, 2010.
- [52] Han-joon Kim, Jiyun Kim, and Jinseog Kim. Semantic text classification with tensor space model-based naïve bayes. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 004206–004210. IEEE, 2016.
- [53] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
- [54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014.
- [55] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [56] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997, 2018.

- [57] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.
- [58] Sreedhi Deleep Kumar, C Sunitha, and Amal Ganesh. Semantic representation of texts in indian languages—a review. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 40–42. IEEE, 2018.
- [59] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.
- [60] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.
- [61] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.
- [62] Ricardo H R Lima and Aurora T R Pozo. A study on auto-configuration of multi-objective particle swarm optimization algorithm. In *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC’17)*, pages 718–725. IEEE, 2017.
- [63] Ricardo H R Lima and Aurora T R Pozo. Evolving convolutional neural networks through grammatical evolution. In *Proceedings of the 2019 Genetic and Evolutionary Computation Conference (GECCO’19)*, pages 179–180. ACM, 2019.
- [64] Ricardo H. R. Lima, Aurora T. R. Pozo, Alexander Mendiburu, and Roberto Santana. A Symmetric grammar approach for designing segmentation models. In *Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC’20)*, pages 1–8. IEEE, 2020.
- [65] Ricardo HR Lima, Dimmy Magalhães, Aurora Pozo, Alexander Mendiburu, and Roberto Santana. A grammar-based gp approach applied to the design of deep neural networks. *Genetic Programming and Evolvable Machines*, pages 1–26, 2022.
- [66] Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2418–2424. AAAI Press, 2015.
- [67] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
- [68] Trevor Londt, Xiaoying Gao, and Peter Andreae. Evolving character-level densenet architectures using genetic programming. *arXiv preprint arXiv:2012.02327*, 2020.
- [69] Pablo Ribalta Lorenzo and Jakub Nalepa. Memetic evolution of deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 505–512, 2018.
- [70] Nuno Lourenço, Filipe Assunção, Francisco B Pereira, Ernesto Costa, and Penousal Machado. Structured Grammatical Evolution: A Dynamic Approach. In *Handbook of Grammatical Evolution*, pages 137–161. Springer, 2018.

- [71] Nuno Lourenço, Francisco Pereira, and Ernesto Costa. Evolving evolutionary algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'12)*, pages 51–58. ACM, 2012.
- [72] Nuno Lourenço, Francisco B. Pereira, and Ernesto Costa. Unveiling the properties of structured grammatical evolution. *Genet. Program. Evolvable Mach.*, 17(3):251–289, 2016.
- [73] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at <http://cs.gmu.edu/~sean/book/metaheuristics/>.
- [74] Hamid Machhour and Ismail Kassou. Concatenate text embeddings for text classification. In *2017 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, pages 206–210. IEEE, 2017.
- [75] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.
- [76] Dimmy Magalhães and Aurora Pozo. Classification of relationship in argumentation using graph convolutional network. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*, pages 60–71. Springer, 2020.
- [77] Dimmy Magalhães, Aurora Pozo, and Roberto Santana. An empirical comparison of distance/similarity measures for natural language processing. In *Anais do XVI Encontro Nacional de Inteligência Artificial e Computacional*, pages 717–728. SBC, 2019.
- [78] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1506–1515. Association for Computational Linguistics, 2017.
- [79] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [80] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzian, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019.
- [81] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
- [82] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing Neural Networks using Genetic Algorithms. In *Proceedings of the International Conference on Genetic Algorithms (ICGA'89)*, volume 89, pages 379–384, 1989.
- [83] Xinchun Ming and Fangyu Hu. Semi-supervised deep network representation with text information. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE, 2017.

- [84] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [85] Aliya Nugumanova, Yerzhan Baiburin, and Kurmash Apaev. A new text representation model enriched with semantic relations. In *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, pages 619–622. IEEE, 2015.
- [86] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [87] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.
- [88] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *ArXiv e-prints*, pages arXiv–1712, 2017.
- [89] Marcelo Pita and Gisele L Pappa. Strategies for short text representation in the word vector space. In *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 266–271. IEEE, 2018.
- [90] Riccardo Poli and John Koza. Genetic programming. In *Search methodologies*, pages 143–185. Springer, 2014.
- [91] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.
- [92] Antonio Regalado. The data made me do it. <https://www.technologyreview.com/s/514346/the-data-made-me-do-it/>, 2013. Accessed: 2019-08-15.
- [93] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.
- [94] Rafael Geraldeli Rossi. *Classificação automática de textos por meio de aprendizado de máquina baseado em redes*. PhD thesis, Universidade de São Paulo, 2016.
- [95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.
- [96] Conor Ryan, John James Collins, and Michael O’Neill. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Proceedings of the European Conference on Genetic Programming (EuroGP’98)*, pages 83–96. Springer, 1998.
- [97] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

- [98] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.
- [99] Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 298–307. The Association for Computational Linguistics, 2015.
- [100] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [101] Neeraj Sharma, Aroor Dinesh Dileep, and Veena Thenkanidiyoor. Text classification using hierarchical sparse representation classifiers. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1015–1019. IEEE, 2017.
- [102] Neeraj Sharma, Anshu Sharma, Veena Thenkanidiyoor, and AD Dileep. Text classification using combined sparse representation classifiers and support vector machines. In *2016 4th International Symposium on Computational and Business Intelligence (ISCBI)*, pages 181–185. IEEE, 2016.
- [103] Roberta A Sinoara, Rafael G Rossi, and Solange O Rezende. Semantic role-based representations in text classification. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2313–2318. IEEE, 2016.
- [104] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
- [105] Ana Sodré, Dimmy Magalhães, Luis Floriano, Aurora Pozo, Carmem Hara, and Sidnei Machado. Covid-19 portal: Machine learning techniques applied to the analysis of judicial processes related to the pandemic. In *European Conference on Advances in Databases and Information Systems*, pages 109–120. Springer, 2021.
- [106] Ana Paula Sodré, Luis Eduardo Mochenski Floriano, Dimmy Magalhaes, Cristina D Aguiar, Aurora Pozo, and Carmem S Hara. Comparing alternative storage models for words extracted from legal texts. In *Anais Estendidos do XXXVI Simpósio Brasileiro de Bancos de Dados*, pages 36–42. SBC, 2021.
- [107] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.
- [108] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [109] Alexander Strehl. *Relationship-based clustering and cluster ensembles for high-dimensional data mining*. The University of Texas at Austin, 2002.

- [110] Alexander Strehl and Joydeep Ghosh. Cluster ensembles: a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.
- [111] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In *Proceedings of the 2017 Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 497–504. ACM, 2017.
- [112] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. Evolving deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.*, 24(2):394–407, 2020.
- [113] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.
- [114] El-Ghazali Talbi. Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys (CSUR)*, 54(2):1–37, 2021.
- [115] Ramzan Talib, Muhammad Kashif Hanif, Shaeela Ayesha, and Fakeeha Fatima. Text mining: techniques, applications and issues. *International Journal of Advanced Computer Science and Applications*, 7(11):414–418, 2016.
- [116] Wenfeng Tian, Jun Li, and Hongguang Li. A method of feature selection based on word2vec in text categorization. In *2018 37th Chinese Control Conference (CCC)*, pages 9452–9455. IEEE, 2018.
- [117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [118] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- [119] Haitao Wang, Jie He, Xiaohong Zhang, and Shufen Liu. A short text classification method based on n-gram and cnn. *Chinese Journal of Electronics*, 29(2):248–254, 2020.
- [120] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990.
- [121] Mu Xue. A text retrieval algorithm based on the hybrid lda and word2vec model. In *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pages 373–376. IEEE, 2019.
- [122] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. *CoRR*, abs/1809.05679, 2018.
- [123] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligence magazine*, 13(3):55–75, 2018.

- [124] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Comput.*, 31(7):1235–1270, 2019.
- [125] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.
- [126] Wenbin Zheng, Yanqiu Liu, Huijuan Lu, and Hong Tang. Discriminative topic sparse representation for text categorization. In *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 454–457. IEEE, 2017.