UNIVERSIDADE ESTADUAL DO MARANHÃO - UEMA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO E SISTEMAS

PAULA MYRIAN LIMA PEDROSO

ESTUDO SEMÂNTICO DE PALAVRAS FORA DO VOCABULÁRIO UTILIZANDO REDES NEURAIS RECORRENTES

SÃO LUÍS - MA
2022

PAULA MYRIAN LIMA PEDROSO

ESTUDO SEMÂNTICO DE PALAVRAS FORA DO VOCABULÁRIO UTILIZANDO REDES NEURAIS RECORRENTES

Dissertação apresentada em cumprimento às exigências do Programa de Mestrado Profissional em Engenharia da Computação e Sistemas na Universidade Estadual do Maranhão como pré-requisito para obtenção do título de Mestre em Engenharia de Computação e Sistemas sob orientação do Prof. Antônio Fernando Lavareda Jacob Júnior e co-orientação do Prof. Fábio Manoel França Lobato.

SÃO LUÍS - MA
2022

Pedroso, Paula Myrian Lima.
Estudo semântico de palavras fora do vocabulário utilizando redes neurais recorrentes / Paula Myrian Lima Pedroso. – São Luís, 2022.
77 f

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia de Computação e Sistemas, Universidade Estadual do Maranhão, 2022.

Orientador: Prof. Me. Antônio Fernando Lavareda Jacob Júnior.

1.PLN. 2.OOV. 3.Redes neurais. 4.Semelhança por cosseno. 5.Marcação gramatical. I.Título.

CDU: 004.8.032.26

Elaborado por Giselle Frazão Tavares - CRB 13/665

PAULA MYRIAN LIMA PEDROSO

ESTUDO SEMÂNTICO DE PALAVRAS FORA DO VOCABULÁRIO UTILIZANDO REDES NEURAIS RECORRENTES

Dissertação apresentada em cumprimento às exigências do Programa de Mestrado Profissional em Engenharia da Computação e Sistemas na Universidade Estadual do Maranhão – UEMA, como requisito para obtenção do título de mestre em Engenharia de Computação e Sistemas.

Aprovado em: 28 / 10 / 2022

BANCA EXAMINADORA

Prof. Antonio Fernando Lavareda Jacob Junior
Orientador - PECS UEMA

Prof. Fábio Manoel França Lobato
Co-orientador - PECS UEMA/UFOPA

Prof. Eveline de Jesus Viana Sá
Membro Interno - PECS UEMA/IFMA.

Prof. Ricardo Marcondes Marcacini
Membro Externo - ICMC/USP

DEDICATÓRIA

Dedico este trabalho ao meu avô Pedro Lima *(in memoriam)*, maior exemplo de um ser humano íntegro e ético.

“Alegrem-se na esperança,
sejam pacientes na tribulação,
perseverem na oração. ”

(Romanos 12:12)

AGRADECIMENTOS

A Deus, nosso eterno pai e soberano. Que sempre me deu forças e esteve comigo em todas as etapas da minha vida. Toda honra e Glória a Ele.

Aos meus pais, José Paulo Gomes Pedroso e Dilzilene Pires Lima Pedroso, por todo apoio, amor e carinho que recebi e continuo recebendo.

Ao meu orientador, Antônio Fernando Lavareda Jacob Junior, pelo apoio, orientação, paciência, atenção e suporte em todas as etapas dessa jornada.

Aos professores Fábio Lobato, Cicero Quarto e Leonardo Nunes pelo apoio e incentivo na vida acadêmica e principalmente por me ajudar no meu crescimento pessoal, profissional e acadêmico.

Aos meus amigos de trabalho, em especial Arthur Silva, pelo apoio e ajuda nas pesquisas e escrita deste trabalho.

E a todas as pessoas que acreditaram e torceram por mim em todos esses anos de graduação e pós-graduação.

RESUMO

O processo de reconhecimento da escrita de textos computacionais por meio do Processamento de Linguagem Natural (PLN) passa por alguns desafios quando há palavras que ainda não foram categorizadas, as quais são denominadas Fora do Vocabulário (OOV). Comumente são conteúdos que fazem uma representação, como gírias locais ou erros de digitação. Estes tipos de conteúdo têm crescido exponencialmente à medida que a Internet popularizou, fazendo com que as pessoas interajam mais assiduamente através de textos. Este trabalho apresenta seis modelos a base de Redes Neurais (RN) para o tratamento dessas palavras desconhecidas na linguagem portuguesa, que são Redes Neurais Recorrentes Simples (RNN), bidirecional RNN (BIRNN), Memória Longa de Curto Prazo (LSTM), bidirecional LSTM (BILSTM), Unidades Recorrentes Fechada (GRU) e bidirecional GRU (BIGRU). Foi realizado treinamentos com os modelos citados utilizando três bases distintas, porém ambas na linguagem portuguesa. Após o treinamento, foi feita uma função que fosse capaz de categorizar as OOVs, criando vetores válidos. Além disso, o significado delas também foi verificado utilizando a semelhança por cosseno e a marcação gramatical. Com todos os testes, foi possível obter uma acurácia de 99,99% com uma das bases utilizando o modelo GRU.

Palavras-chave: PLN; OOV; Redes Neurais; Recorrentes; bidirecional; semelhança por cosseno; marcação gramatical.

ABSTRACT

The process of recognizing computer text writing by means of Natural Language Processing (NLP) goes through some challenges when there are words that have not yet been categorized, which are called Out-of-Vocabulary (OOV). These are commonly content that make a representation, such as local slang or typing error. These types of content have grown exponentially as the Internet has become more popular, causing people to interact more assiduously through text. This paper presents six Neural Network (NN) based models for the treatment of these unknown words in the Portuguese language, which are Simple Recurrent Neural Networks (RNN), bidirectional RNN (BIRNN), Long Short-Term Memory (LSTM), bidirectional LSTM (BILSTM), Gated Recurrent Units (GRU) and bidirectional GRU (BIGRU). The models were trained using three different bases, but both in Portuguese. After training, a function was made that was able to categorize the OOVs, creating valid vectors. In addition, their meaning was also verified using cosine similarity and part-of-speech tagging. With all the tests, it was possible to obtain an accuracy of 99.99% with one of the bases using the GRU model.

Keywords: NLP; OOV; Neural Networks; Recurrent; bidirectional; cosine similarity; part-of-speech tagging.

LISTA DE FIGURAS

Figura 1 — Estrutura básica do ciclo do RNN 23

Figura 2 — Arquitetura de uma RNN 24

Figura 3 — Arquitetura de uma LSTM 25

Figura 4 — Arquitetura de uma GRU 27

Figura 5 — Arquitetura Bidirecional 28

Figura 6 — Fluxo da implementação do experimento 34

Figura 7 — Fluxo para a preparação do Modelo 35

Figura 8 — Fluxo para a predição de Embeddings 40

Figura 9 — Embedding original da palavra “mucura” 42

Figura 10 — Embedding da palavra “mucura” após passar pela função . . 44

Figura 11 — Linha do tempo do treinamento da Base 1 46

Figura 12 — Linha do tempo do treinamento da Base 2 48

Figura 13 — Linha do tempo do treinamento da Base 3 50

LISTA DE TABELAS

Tabela 1 Critérios de Inclusão e Exclusão 16

Tabela 2 Resultado do primeiro processo de triagem 17

Tabela 3 Técnicas de Classificação 18

Tabela 4 Informações detalhadas das bases de dados 32

Tabela 5 Características das bases de dados 32

Tabela 6 Hiperparâmetros de treinamento dos Modelos 38

Tabela 7 Resultados do treinamento na Base 1 45

Tabela 8 Resultados da marcação gramatical utilizando a Base 1 na palavra maroca 47

Tabela 9 Resultados do treinamento na Base 2 47

Tabela 10 Resultados da marcação gramatical utilizando a Base 2 na palavra marocar 49

Tabela 11 Resultados do treinamento na Base 3 49

Tabela 12 Resultados da marcação gramatical utilizando a Base 3 na palavra bofetes 50

Tabela 13 Melhores resultados de cada Base 51

LISTA DE ABREVIATURAS

API Interface de Programação de Aplicação - *Application Programming Interface*

BIGRU *Bidirectional Gated Recurrent Unit*

BILSTM *Bidirectional Long Short-Term Memory*

BIRNN *Bidirectional Recurrent Neural Network*

CE Critérios de Exclusão

CI Critérios de Inclusão

CNN *Convolutional Neural Networks*

CRF *Conditional Random Field*

CRM Gestão de relacionamento com o cliente - *Customer relationship management*

DL Aprendizado Profundo - *Deep Learning*

DOI *Digital Object Identifier*

GRU Unidade Recorrente Fechada - *Gated Recurrent Unit*

IA Inteligência Artificial

LSTM Memória de Curto Prazo Longa - *Long Short-Term Memory*

MIMICK *Mimicking Word Embeddings using Subword*

ML Aprendizado de Máquina - *Machine Learning*

OOV Palavra Fora do Vocabulário - *Out-of-Vocabulary Word*

PLN Processamento de Linguagem Natural

POS Marcação gramatical - *Part-of-speech tagging*

PP Pergunta de Pesquisa

RN Redes Neurais

RNN Rede Neural Recorrente - *Recurrent Neural Network*

RNNs Redes Neurais Recorrentes

RSL Revisão Sistemática da Literatura

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Contextualização e Desafios 13
1.2 Objetivo 14
1.3 Organização do Trabalho 14

2 METODOLOGIA 15
2.1 Revisão Sistemática 15
2.1.1 Planejamento 15
2.1.2 Condução 16
2.1.3 Relato da Revisão 17
2.1.4 Escolha dos Métodos 18

3 REFERENCIAL TEÓRICO 20
3.1 Processamento de Linguagem Natural 20
3.2 Aprendizado Profundo 20
3.3 Embeddings de Palavras 21
3.3.1 Word2Vec 21
3.3.2 Global Vectors for Word Representation (GloVe) 21
3.3.3 Fasttext 22
3.4 Redes Neurais 22
3.4.1 Redes Neurais Recorrentes (RNN) 22
3.4.2 Memória de Curto Prazo Longa (LSTM) 24
3.4.3 Unidade Recorrente Fechada (GRU) 26
3.4.4 Redes Bidirecionais 27

4 ARQUITETURA DO EXPERIMENTO 29
4.1 Implementação 29
4.1.1 Numpy 29
4.1.2 Pickle 30
4.1.3 TensorFlow 30
4.1.4 Keras 30
4.1.5 SpaCy 30
4.1.6 Pandas 31
4.2 Base de Dados 31

4.3 **Função de Similaridade** **33**
4.4 **Marcação Gramatical** **33**
4.5 **Representação da arquitetura** **34**

5 **DESENVOLVIMENTO** **35**
5.1 **Preparação do Modelo** **35**
5.1.1 ***Importação do Corpus de Treinamento*** 36
5.1.2 ***Tratamento do Corpus*** 36
5.1.3 ***Definição do Modelo*** 37
5.1.4 ***Treinamento do Modelo*** 38
5.2 **Predição de embeddings das OOVs** **39**
5.2.1 ***Carregar o Modelo Tratado*** 40
5.2.2 ***Gerador de sequências*** 41
5.2.3 ***Textos de amostras para testes*** 41
5.2.4 ***Função para criar Embeddings*** 43
5.2.5 ***Similaridade*** 44

6 **RESULTADOS** **45**
6.1 **Base 1** **45**
6.2 **Base 2** **47**
6.3 **Base 3** **49**
6.4 **Comparação entre as Bases** **51**

7 **CONSIDERAÇÕES FINAIS** **52**

REFERÊNCIAS **58**

APÊNDICE A **Semelhança de cosseno entre pares de palavras** **59**

APÊNDICE B **Marcação Gramatical** **62**

ANEXO A **Código GRU Utilizado na Preparação do Modelo** **64**

ANEXO B **Código GRU Utilizado na Predição de Embeddings** . . . **67**

1 INTRODUÇÃO

A popularização da internet trouxe uma nova forma de comunicação entre as pessoas. A escrita da web muitas das vezes difere da escrita formal e tradicional que é aplicada em livros. Esse tipo de texto tem novas características, uma vez que a própria escrita assume uma nova forma, identificada por elementos verbais e não verbais. Esse rápido desenvolvimento da área de informação social e técnica, trouxe o aumento contínuo dos números de formato de texto digitais (ZULQARNAIN et al., 2019).

Para que dispositivos tecnológicos processem a linguagem escrita do ser humano a fim de responder às demandas, é necessário utilizar a tecnologia de Processamento de Linguagem Natural (PLN). Ela é um ramo da Inteligência Artificial (IA) que agrupa uma gama de técnicas computacionais motivadas por teoria para a análise automática e representacional da linguagem humana (KANDI, 2018), frequentemente utilizada para fazer a classificação de dados em textos.

1.1 Contextualização e Desafios

O campo que envolve o tratamento de palavras é muito amplo, que vai da tradução automática, reconhecimento de significados, categorização de sentimentos, entre outros (BEYSOLOWII, 2018). Nos dias atuais, existem muitos algoritmos com capacidade de incorporar análises probabilísticas e técnicas de avaliação, com capacidades para definir uma palavra (JETTAKUL et al., 2018).

Por essa razão, há uma grande variedade de tarefas subjacentes e modelos de aprendizado de máquina. Esses modelos pré-treinados possuem limitações quanto às palavras nunca vistas, pois eles não conseguem classificar com exatidão o real significado das palavras chamadas Palavra Fora do Vocabulário - *Out-of-Vocabulary Word* (OOV). O bom tratamento dessas palavras ajuda a não degradar a qualidade dos aplicativos de PLN. Por mais que seja um campo em constante crescimento, o não tratamento das OOVs tem efeito deletério na confiabilidade das análises, pois faz com que haja perda de informações nos cálculos e, consequentemente, nas tomadas de decisões (PARK et al., 2019).

Verifica-se que atualmente existem muitas técnicas de tratamento desses tipos

de palavras. Porém, algumas limitações são percebidas, como por exemplo, o não tratamento de OOVs consideradas variantes de outras OOVs, ou ainda, que sejam OOVs compostas e erros ortográficos (KIM et al., 2018). Tal fato fica evidenciado quando utilizado um domínio textual de treinamento diferente dos utilizados em tarefas de PLN (JETTAKUL et al., 2018).

Trabalhos como os de Kandi (2018), Tang, Tang e Zhu (2020) e Won e Lee (2018) propõe tratamento das palavras utilizando métodos para reconhecê-las com base no contexto em que estão inseridas, produzindo uma incorporação para elas. Porém, esses mesmos trabalhos e outros como os quais também posso citar os de Sharma, Mittal e Vidyarthi (2022) e Chenhao e Chengyao (2019), não fazem o tratamento de OOVs da língua portuguesa. O trabalho de Costa e Paetzold (2018) apresenta um modelo para português que combina redes neurais recorrentes e campos aleatórios condicionais, porém não apresenta se as incorporações encontradas possuem o significado real das palavras.

1.2 Objetivo

À luz desta problemática, este trabalho tem como objetivo principal apresentar modelos para tratamento dessas palavras que ainda não foram caracterizadas. Para isso, será seguido com os seguintes objetivos específicos:

- Identificar qual modelo obteve a melhor precisão;
- Medir a eficácia da incorporação vetorial das OOVs na utilização dos modelos;

e

- Verificar as semelhanças semânticas das OOVs.

1.3 Organização do Trabalho

O trabalho encontra-se organizado da seguinte maneira: no Capítulo 2 mostra a metodologia utilizada para as escolhas do presente estudo, no Capítulo 3 é destacado o referencial teórico utilizado mediante a metodologia escolhida, no Capítulo 4 ressalta a arquitetura do experimento, no Capítulo 5 foca no desenvolvimento e no Capítulo 6 evidencia os resultados. Por fim, no Capítulo 7 estão as conclusões e considerações finais.

2 METODOLOGIA

Atualmente, existem vários algoritmos com capacidade de incorporar análises probabilísticas e técnicas de avaliação, com capacidades para definir uma palavra. Nas variadas pesquisas sobre PLN, a maioria das tarefas é baseada em métodos de nível de palavra porque ela é a menor unidade linguística em línguas naturais (LI et al., 2018).

Porém, os tratamentos a nível de palavras não englobam as palavras desconhecidas, visto que a maioria possui seus próprios significados como variantes, compostos e erros ortográficos (KIM et al., 2018), o qual fica mais evidente quando utilizado um domínio textual de treinamento diferente dos utilizados em tarefas de PLN.

A fim de definir um método eficiente para tratar OOVs, foi realizada uma análise na literatura atual com o intuito de coletar informações pertinentes para servirem de auxílio para este trabalho de tratamento das palavras desconhecidas.

2.1 Revisão Sistemática

Kitchenham et al. (2010) descreve a Revisão Sistemática da Literatura (RSL) como um método que possibilita recuperar, avaliar e interpretar todas as pesquisas disponíveis, relevantes para uma determinada Pergunta de Pesquisa (PP), ou área de tópico ou fenômeno de interesse. Ela passa pelas fases de Planejamento, Condução e Relato da Revisão. O detalhamento de todas as fases é apresentado nas Subseções a seguir.

2.1.1 Planejamento

A etapa do planejamento envolve a identificação da necessidade da revisão, a especificação das PP que devem ser respondidas e criar um protocolo. Ela é uma das principais atividades realizadas no processo de revisão sistemática, pois serve como orientação para o desenvolvimento dos critérios contidos no protocolo, o escopo da revisão e as atividades a serem realizadas nas etapas seguintes (SPOLAÔR et al., 2013).

As PP foram as seguintes:

- PP1: Quais as técnicas que estão sendo usadas para classificar palavras fora do vocabulário?
- PP2: Quais são os tipos de modelos usados para classificação?

- PP3: Qual é o método para avaliar os modelos?

Após essa identificação, outra parte importante do planejamento é identificar as fontes pelas quais os trabalhos científicos serão extraídos para poder auxiliar na busca e recuperação da literatura apropriada (KITCHENHAM et al., 2010). Conforme organização apresentada em (LOBATO et al., 2020), esse processo foi realizado da seguinte forma:

1. Definição das fontes: *ACM Digital Library*[1], *IEEE Digital Library*[2], *ISI Web of Science*[3], *Science@Direct*[4] e *Scopus*[5].
2. Definição dos Critérios de Inclusão (CI) e Critérios de Exclusão (CE): organizados na Tabela 1.
3. Organização dos dados: Foi extraído das fontes características como Título, Código *Digital Object Identifier* (DOI), Ano de Publicação, Nome do periódico, Resumo e Palavras-chave.

Tabela 1 – Critérios de Inclusão e Exclusão

Inclusão	**Exclusão**
CI1 - Publicações realizadas após 2018	CE1 - Capítulo de livro, teses, dissertação, website, livros didáticos
CI2 - Artigos revisados por periódicos e conferências relevantes	CE2 - Domínio de corpus de áudio
CI3 - Publicação que propõe ou compara métodos para tratar palavras fora do vocabulário	CE3 - Reconhecimento de fala
	CE4 - Editoriais e artigos de opinião
	CE5 - Publicações realizadas antes de 2018
	CE6 - Tutorial, poster, report técnico e blogs

Fonte: Elaborado pelos Autores

2.1.2 Condução

Ao realizar uma RSL, é importante manter um registro das palavras-chave e métodos usados na pesquisa de literatura, pois eles precisarão ser identificados poste-

[1] http://portal.acm.org
[2] http://ieeexplore.ieee.org
[3] http://www.isiknowledge.com
[4] http://www.sciencedirect.com
[5] http://www.scopus.com

riormente ao descrever como a pesquisa foi conduzida (RAMDHANI; RAMDHANI; AMIN, 2014).

Os registros que foram buscados neste trabalho foram: informações do banco de dados utilizado para treino (tipo, disponibilidade e quantidade), os modelos utilizados para a representação textual, os modelos utilizados para a classificação textual e as métricas utilizadas para mensurar os treinamentos.

2.1.3 Relato da Revisão

Foi associado a sequência de pesquisa nas fontes citadas anteriormente no processo de planejamento e foram obtidas um total de 11.267 publicações. Após, foi realizada a seleção dos estudos, onde foram removidos os duplicados e aplicados os critérios de exclusão e inclusão, realizando a análise do título, do resumo e das palavras-chaves para definir quais publicações iriam ser aceitas na etapa de avaliação de qualidade. A Tabela 2 mostra os resultados obtidos após a primeira etapa da triagem.

Tabela 2 – Resultado do primeiro processo de triagem

Bases	**Total**	**Avaliação**		
		Aceito	Rejeitado	Duplicado
ACM Digital Library	2.146	25	1.825	296
IEEE Digital LIbrary	584	23	320	241
ISI Web of Science	1.202	6	159	1.037
Science@Direct	5.597	55	5.106	436
Scopus	1.738	97	1.166	475
TOTAL	11.267	206	8.576	2.485

Fonte: Elaborado pelos Autores

Foi realizada a seleção dos estudos, onde foram removidos os duplicados e aplicados os critérios de exclusão e inclusão, realizando a análise do título, do resumo e das palavras-chaves para definir quais publicações iriam ser aceitas na etapa de avaliação de qualidade. Assim como mostrado na Tabela 2, passaram para a próxima etapa 206 trabalhos.

Seguindo, foi feito uma breve leitura nos trabalhos focando na metodologia e nos resultados, com o intuito de classificá-los com pontuações mediante aos critérios de qualidade, onde obtivemos o total de 62 publicações.

Utilizando essas publicações pertinentes com uma análise mais detalhada, a fim de responder às questões de pesquisa, foram identificadas 2 técnicas para classificar palavras fora do vocabulário (PP1), 32 modelos para processamento das palavras (PP2) e 7 métricas para avaliar a eficiência dos modelos (PP3). Os trabalhos que utilizavam

outros modelos fora dos detectados não foram categorizados para essa análise, pois apareciam apenas uma vez ou eram modelos autorais.

As técnicas de classificação são: por contexto e saco de palavras. Há também na literatura modelos que utilizam as duas técnicas de classificação que servem para poder criar vetores para as palavras. Neste processo, foram identificados 4 algoritmos sendo utilizados em mais de cinco trabalhos: *Word2Vec*, *GloVe*, *FastText* e *BERT*. Esses métodos utilizam o *embeddings* de palavras como tipo de representação. Os detalhes estão mostrados na Tabela 3.

Tabela 3 – Técnicas de Classificação

Técnicas de Classificação	**Modelos**	**Quantidade de Trabalhos**
Sequencial e Saco de Palavras	Word2vec	24
	Glove	10
Sequencial	Fasttext	9
	BERT	9

Fonte: Elaborado pelos Autores

Quanto aos modelos de reconhecimento de entidades nomeadas, os principais apresentados são: *Conditional Random Field* (CRF), *Long Short-Term Memory* (LSTM), *Recurrent Neural Network* (RNN), *Bidirectional Long Short-Term Memory* (BILSTM), *Convolutional Neural Networks* (CNN), *Mimicking Word Embeddings using Subword* (MIMICK) e *Gated Recurrent Unit* (GRU). Foi posto nesta RSL os principais modelos que apareciam em no mínimo cinco trabalhos diferentes.

As métricas utilizadas para avaliar os modelos são: *Accuracy*, *BLEU*, *F1-Score*, *Precision*, *ROUGE*, *Recall* e *Loss*. Houve outras medidas de desempenho, porém não apareciam em mais de cinco trabalhos e por esse motivo não foi computado neste estudo.

2.1.4 Escolha dos Métodos

Conforme o objetivo deste trabalho (1.2), foi retirado da RSL trabalhos como os de Kandi (2018), Tang, Tang e Zhu (2020) e Won e Lee (2018), que buscavam reconhecer *embeddings* para OOVs com base no contexto, para poder extrair informações e melhorar a qualidade da classificação de texto com palavras não vistas.

Dentre os reconhecedores de *embeddings* vistos na RSL, foram detectados o *Word2Vec*, *GloVe*, *FastText* e *BERT*. O método escolhido foi o *GloVe*, pois ele aprende construindo uma matriz de co-ocorrência, baseada na técnica de fatoração de matrizes na matriz de contexto de palavras (KHALIFA; SHAALAN, 2019).

O *Word2Vec* não foi escolhido pois ele prevê o contexto de uma determinada palavra dentro do corpus e esses tipos de métodos baseados em corpus geralmente falham em capturar os significados específicos das palavras (MENG et al., 2020).

O *FastText* não trabalha com a palavra completa, mas com a segmentação da palavra. Para o reconhecimento de uma palavra OOV nesse modelo torna-se inviável o aprendizado da semântica de palavras de baixa frequência e também de palavras que não são derivadas de outras (LIAO et al., 2021). Por esse motivo também não foi utilizado.

Quanto ao *BERT*, ele representa a entrada como subpalavras e aprende incorporações para subpalavras (LIAO et al., 2021). Usando um transformador, as palavras mascaradas são então previstas usando palavras não mascaradas ao seu redor (à esquerda e à direita). Esse método pode até atribuir um valor para a palavra OOV, mas pode levar a uma falsa representação dela devido a utilização de pedaços de palavras (WON et al., 2021). Ocorrendo uma inviabilidade parecida com a que foi apontado no *FastText*.

Baseado no estado da arte discutido anteriormente, os modelos de Redes Neurais Recorrentes (RNNs) são os mais adequados para este tipo de tarefa, tornando-se bem presentes nos trabalhos achados durante a RSL. Premjith, Soman e Poornachandran (2018) afirma que as RNNs juntamente com suas versões bidirecionais superaram todas as implementações de nível mundial em termos de precisão, número de parâmetros treináveis e requisitos de armazenamento. Goyal, Gupta e Kumar (2021) confirma quando fala que as RNNs auxiliam no aprendizado essencial para que as informações textuais sejam previstas, sendo ideais para dados sequenciais.

Para verificar o real significado das palavras, será utilizado a medida de similaridade de cosseno, pois é um indicador muito utilizado para caracterizar similaridade semântica entre vetores de palavras (DOVAL; VILARES; GÓMEZ-RODRÍGUEZ, 2020; HU et al., 2019). Além disso, como forma de verificar se a palavra foi devidamente classificada, será feito uma análise da marcação gramatical tendo como base a palavra semelhante à OOV.

3 REFERENCIAL TEÓRICO

Este capítulo está resumido com as informações relevantes que serão utilizadas como base de referenciamento de outras literaturas, tendo como base a metodologia anteriormente explicada.

3.1 Processamento de Linguagem Natural

O PLN é um ramo da IA que ajuda os computadores a entender, interpretar e manipular a linguagem humana através do estudo das interações entre computadores e linguagens naturais humanas (KANDI, 2018).

Os aplicativos de PLN são usados para extrair dados não estruturados e baseados em texto, podendo ser possível organizar e estruturar dados de linguagem para executar tarefas como sumarização automática, tradução, reconhecimento de entidade nomeada, extração de relacionamento, análise de sentimento, reconhecimento de fala e segmentação de tópicos (THANAKI, 2017).

3.2 Aprendizado Profundo

O Aprendizado Profundo - *Deep Learning* (DL) analisa problemas complexos para facilitar o processo de tomada de decisão, pois tenta imitar o que o cérebro humano pode alcançar extraindo recursos em diferentes níveis de abstração, sendo muito utilizado em tarefas de PLN (SULEIMAN; AWAJAN, 2020).

As principais vantagens do DL são que os recursos aprendidos são fáceis de adaptar e rápidos de aprender, em oposição aos recursos projetados manualmente que geralmente tendem a ser muito específicos e exigem muito tempo para projetar e validar (KANDI, 2018).

Vários modelos de DL foram empregados para sumarização abstrata, incluindo Redes Neurais (RN) (SULEIMAN; AWAJAN, 2020). Essas RNs tentam simular o comportamento do cérebro humano, permitindo que ele aprenda com grandes quantidades de dados, podendo impulsionar muitos aplicativos e serviços de IA que melhoram a automação, realizando tarefas analíticas e físicas, sem intervenção humana (DEEP..., 2022).

3.3 Embeddings de Palavras

Um *embedding* é uma representação de um símbolo (palavra, caractere ou frase) em um espaço distribuído de baixa dimensão de vetores de valor contínuo (CHO et al., 2020). Existem algumas vantagens neste espaço contínuo, uma vez que a dimensionalidade é amplamente reduzida e palavras mais próximas em significado estão próximas neste novo espaço contínuo (YEPES, 2017). Isso faz com que os *embeddings* sejam capazes de capturar conhecimento morfológico, sintático e semântico. De um modo geral, existem dois tipos principais de *embedding* de palavras que foram desenvolvidos, a saber incorporação independente de contexto e incorporação dependente de contexto (WANG; NULTY; LILLIS, 2020).

Métodos independentes de contexto são conhecidos como incorporação de palavras clássicas, tendo as suas representações sendo aprendidas de forma única, sem considerar o contexto ao qual a palavra está inserida (WANG; NULTY; LILLIS, 2020). Os algoritmos mais comuns de representação vetorial encontrados, dos que independem de contexto, incluem *Word2Vec* e *Glove* (YEPES, 2017). Os métodos dependentes do contexto aprendem diferentes incorporações para a mesma palavra que depende do contexto em que é usada, o qual pode ser incluído o *Fasttext* (WANG; NULTY; LILLIS, 2020).

Wang, Nulty e Lillis (2020) diz que *word2vec*, *GloVe* e *FastText* são os três *embeddings* clássicos para análise comparativa. A seguir uma breve explicação sobre eles.

3.3.1 Word2Vec

O modelo *Word2vec* é a implementação de duas arquiteturas de modelo de linguagem de RN para aprender representações distribuídas de palavras que tentam minimizar a complexidade computacional Won e Lee (2018).

É um método clássico de incorporação de palavras bem conhecido que aplica as suas arquiteturas através de treinamentos com base em um modelo baseado em predição neural (WANG; NULTY; LILLIS, 2020). OOV pode ser inferido por token desconhecido, treinado antecipadamente pelo *Word2vec* (WON et al., 2021).

3.3.2 Global Vectors for Word Representation (GloVe)

As *embeddings* de palavras baseadas em semântica são tratadas com a limitação de contextos lineares propondo contextos sintáticos derivados de um analisador de dependência, entretanto o método *GloVe* aborda a limitação de contextos locais por meio de estatísticas globais de co-ocorrência palavra a palavra (ZULQARNAIN et al.,

2019; WANG; NULTY; LILLIS, 2020).

O modelo é eficiente em alavancar informações estatísticas obtidas do corpus[6] treinando apenas os elementos diferentes de zero em uma matriz de co-ocorrência (KANDI, 2018).

3.3.3 Fasttext

É uma biblioteca de PLN de código aberto criada pela *Facebook AI Research*[7] que permite que os usuários aprendam com eficiência representações de palavras (KANDI, 2018).

Modelos *FastText* permitem a criação de um algoritmo de aprendizado não supervisionado ou de aprendizado supervisionado para a obtenção de representações vetoriais de palavras pré-treinadas e também estão disponíveis para vários idiomas (WANG; NULTY; LILLIS, 2020).

3.4 Redes Neurais

As RN são uma classe específica e comumente usada de algoritmos de aprendizado de máquina, sendo modeladas a partir do funcionamento do cérebro humano, no qual milhares ou milhões de nós de processamento são interconectados e organizados em camadas (DEEP. . . , 2022).

A rede recorrente é um tipo particular de estrutura de redes neurais artificiais, especialmente aplicada para modelagem sequencial (ZULQARNAIN et al., 2019). Ela é usada para reconhecer a estrutura das palavras a fim de contribuir com sua representação e significado, sendo uma abordagem de classificação (BEYSOLOWII, 2018).

3.4.1 Redes Neurais Recorrentes (RNN)

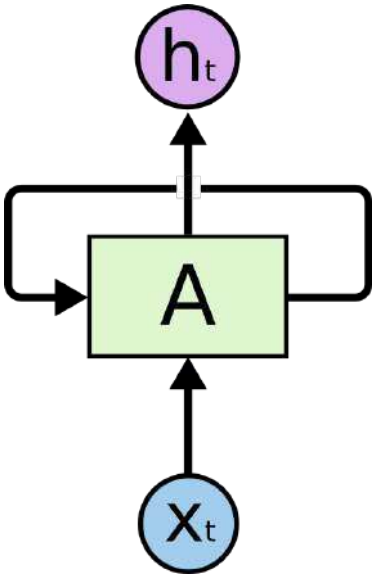
A Rede Neural Recorrente - *Recurrent Neural Network* (RNN) é projetada para trabalhar com dados sequenciais e usa as informações anteriores na sequência para produzir a saída atual. Ela usa esse histórico para prever a saída em cada etapa de tempo (PREMJITH; SOMAN; POORNACHANDRAN, 2018). Suleiman e Awajan (2020) utiliza como exemplo que, em uma frase, o significado de uma palavra está intimamente relacionado ao significado das palavras anteriores.

RNN é uma estrutura geral para modelagem de dados de sequência e é particularmente útil para tarefas de processamento de linguagem natural, funcionando em

[6]Recurso linguístico que consiste em um conjunto grande e estruturado de textos
[7]https://ai.facebook.com/

ciclos que permitem para persistir informações do passado no modelo de rede (KANDI, 2018; DEEP. . . , 2022). A Figura 1 mostra a estrutura básica desse ciclo.

Figura 1 – Estrutura básica do ciclo do RNN

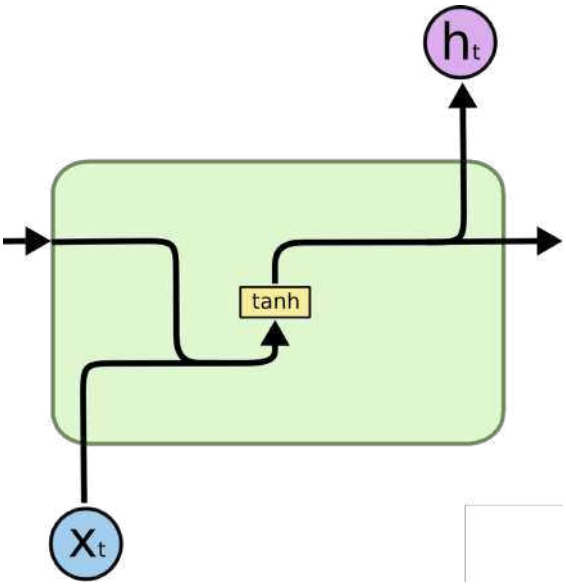


Fonte: (SHEWALKAR, 2018)

Na Figura 1 é mostrado o quadrado central A, que representa uma rede neural, que recebe a entrada X_t na fatia de tempo atual t e fornece o valor h_t como saída. O ciclo mostrado na estrutura permite usar informações anteriores para produzir uma saída para a fatia de tempo atual t. Assim, podemos dizer que a decisão tomada na fatia de tempo $t-1$ afeta a decisão a ser tomada na fatia de tempo t (SHEWALKAR, 2018).

Por padrão, as redes neurais recorrentes possuem a forma de uma cadeia de módulos repetitivos com uma estrutura simples e uma função de ativação. A arquitetura básica de uma RNN está mostrada na Figura 2.

Figura 2 – Arquitetura de uma RNN



Fonte: (MAMANDIPOOR et al., 2020)

A função de ativação é representada na Figura 2 por *tanh*, os valores de entrada e saída é mostrado por X_t e h_t respectivamente. A escolha da função de ativação controlará quão bem o modelo de rede aprende o conjunto de dados de treinamento (MORCHID, 2018).

Uma RNN consiste em um conjunto de estados ocultos que são aprendidos pela rede neural, que pode consistir em várias camadas de estados ocultos, onde estados e camadas aprendem diferentes recursos (SULEIMAN; AWAJAN, 2020). Ou seja, a informação sequencial é preservada no estado oculto da rede recorrente, que consegue passar por muitas etapas de tempo à medida que ela avança em cascata para afetar o processamento de cada novo exemplo (RAGHEB; GODY; SAID, 2021). Nas Figuras 1 e 2 o estado oculto na etapa de tempo t é representado por h_t, que também é o valor de saída.

O problema com esses estados ocultos é que a medida que essa sequência vai se tornando mais longa, as informações ficam mais difíceis para serem transportadas para as próximas etapas. Esse problema é chamado de dissipação de gradiente ou simplesmente de memória de curto prazo (MORCHID, 2018).

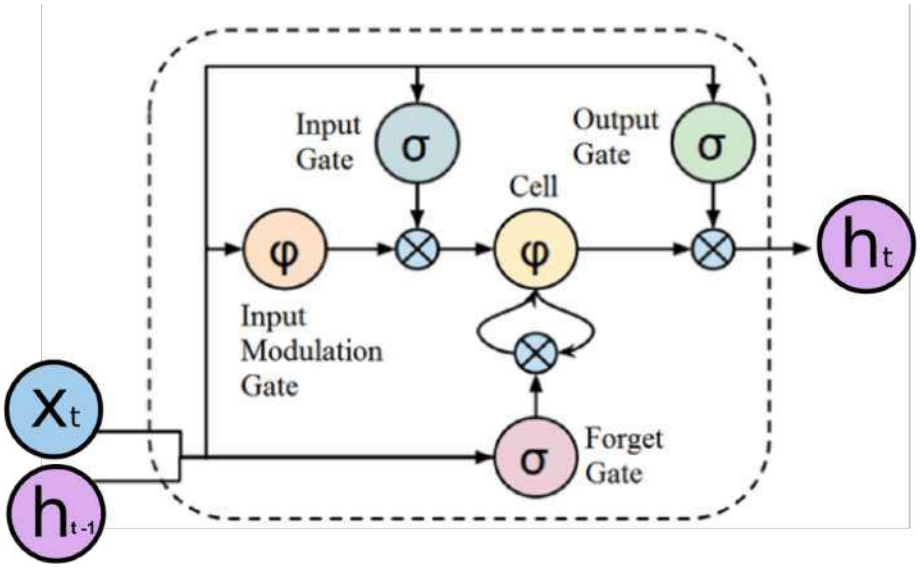
3.4.2 Memória de Curto Prazo Longa (LSTM)

A Memória de Curto Prazo Longa - *Long Short-Term Memory* (LSTM) é um tipo especial de redes neurais recorrentes com células de memória, projetada para resolver o problema da dissipação de gradiente de uma RNN tradicional. Esse método pode

projetar com sucesso dependências temporais de longo prazo apropriadas e recursos de comprimento variável, diminuindo significativamente a necessidade de pré-processar dados (MAMANDIPOOR et al., 2020).

O LSTM funciona da mesma maneira que o RNN em termos de desdobramento, mas ele possui uma estrutura em cadeia que contém diferentes blocos de memória chamados células e é capaz de lembrar o contexto por um longo tempo com a ajuda de alguns circuitos de portas especialmente projetados (PREMJITH; SOMAN; POORNACHANDRAN, 2018). A arquitetura básica do LSTM é mostrado na Figura 3.

Figura 3 – Arquitetura de uma LSTM



Fonte: Feito pelo autor, adaptado de (DEEP..., 2022)

Na Figura 3 é mostrado o X_t e o h_{t-1} como valor de entrada e h_t como a saída. As funções de ativação são representadas por σ como função sigmoide e φ como a função *tanh*, que foi definido por Morchid (2018) como:

- ***Tanh***: Ajuda a regular os valores que fluem pela rede, garantindo que os valores fiquem entre -1 e 1;
- **Sigmóide**: É semelhante à ativação *tanh*, porém ela comprime os valores entre 0 e 1, fazendo com que os valores desapareçam ou sejam mantidos.

Também é mostrado na Figura 3 as portas (*gate*) e a célula (*cell*). Os quatro portões compartilham informações entre si, assim as informações podem fluir em *loops* por um longo período dentro da célula. Eles foram definidos por Suleiman e Awajan (2020) como:

- ***Input Modulation Gate*** **- Portão de Modulação de Entrada**: Ele controla o efeito das informações lembradas nas novas informações. Possui uma função de

ativação φ que é utilizada para gerar as novas informações. A nova informação é formada pela adição da informação antiga ao resultado da multiplicação elemento a elemento da saída das duas redes neurais de porta de memória;

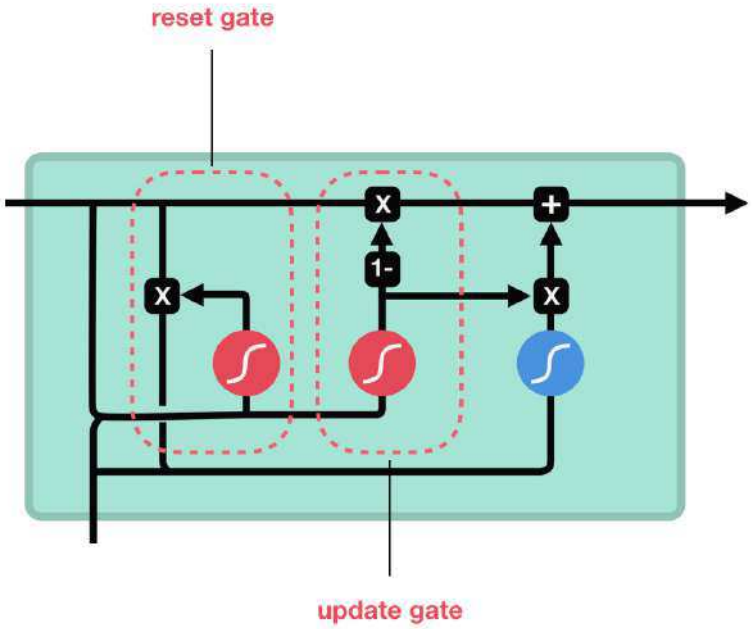
- ***Forget Gate* - Portão de Esquecimento**: Ele é uma rede neural com uma camada e uma função de ativação σ. O valor da função σ determinará se a informação do estado anterior deve ser esquecida ou lembrada. Se o valor do sigmóide for 1, o estado anterior será lembrado, mas se o valor do sigmóide for 0, o estado anterior será esquecido;
- ***Input gate* - Portão de Entrada**: No primeiro passo de tempo, a entrada é um vetor que é inicializado aleatoriamente, enquanto nos passos subsequentes à entrada do passo atual é a saída (conteúdo da célula de memória) do passo anterior. Em todos os casos, a entrada está sujeita à multiplicação elemento a elemento com a saída do portão de esquecimento. O resultado da multiplicação é adicionado à saída da porta de memória atual;
- ***Output gate* - Portão de Saída**: Ela controla a quantidade de novas informações que são encaminhadas para a próxima unidade LSTM. A porta de saída é uma rede neural com uma função de ativação σ que considera o vetor de entrada, o estado oculto anterior, as novas informações e o viés como entrada. A saída da função σ é multiplicada pelo φ da nova informação para produzir a saída do bloco atual.

3.4.3 Unidade Recorrente Fechada (GRU)

A ideia principal por trás da Unidade Recorrente Fechada - *Gated Recurrent Unit* (GRU) é que a ela expõe o conteúdo da memória a cada passo de tempo e equilibra o conteúdo da memória anterior e o nova usando estritamente a integração com vazamento (MORCHID, 2018). Bem semelhante ao LSTM, ele pode ser usado para melhorar a capacidade de memória de uma rede neural, bem como fornecer a facilidade de treinar um modelo, pois decide se tais informações devem ser atualizadas ou não dentro de uma célula (PREMJITH; SOMAN; POORNACHANDRAN, 2018).

A diferença entre o GRU e o LSTM está que o primeiro possui apenas dois portões: um portão de redefinição (*reset gate*) e um portão de atualização (*update date*). A Figura 4 mostra a arquitetura básica do GRU.

Figura 4 – Arquitetura de uma GRU



Fonte: (DEEP..., 2022)

Na Figura 4, de acordo com Premjith, Soman e Poornachandran (2018):

- ***Update Gate* - Portão de Atualização**: É usado para calcular a quantidade de informações passadas a serem armazenadas no estado atual;
- ***Reset Gate* - Portão de Reinicialização**: Decide como mesclar a entrada do passo de tempo com as informações contextuais do passo de tempo.

Assim como o LSTM, o GRU também utiliza as funções sigmóide e *tanh*, representadas na Figura 4 pelo símbolo ς em vermelho e azul, respectivamente. Esse formato ajuda a filtrar seletivamente qualquer informação irrelevante.

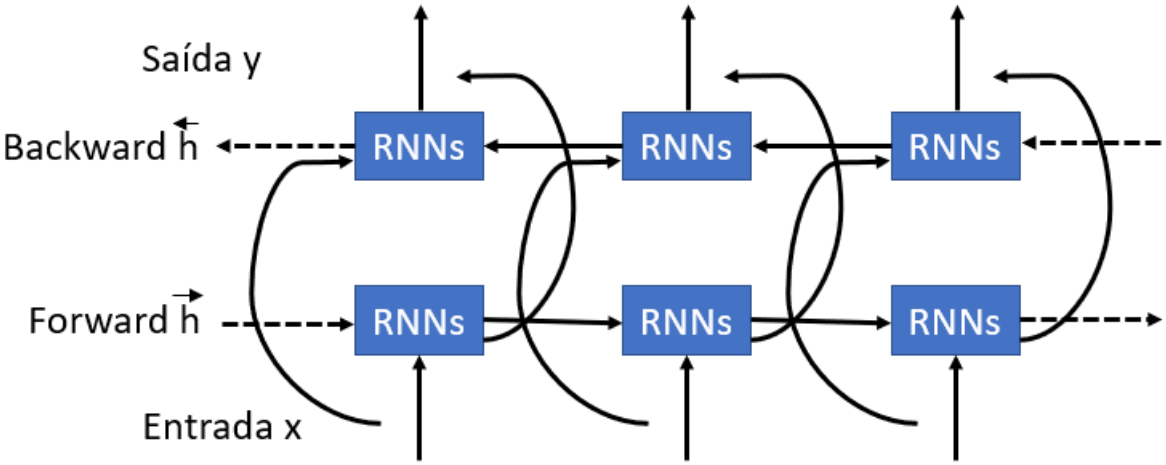
Se compararmos GRU com LSTM, o portão de atualização é a combinação do portão de entrada e de esquecimento e o estado oculto anterior é conectado diretamente ao portão de reinicialização. O que difere esses dois métodos está na exposição do conteúdo da memória, pois como as GRUs não possuem porta de saída, ela expõe todo o seu conteúdo de memória, enquanto no LSTM o conteúdo de memória a ser usado ou visto por outras células da rede é gerenciado pela porta de saída (SHEWALKAR, 2018).

3.4.4 Redes Bidirecionais

As redes bidirecionais possuem uma arquitetura formada por duas camadas: estado oculto para frente (*forward*) e estado oculto para trás (*backward*), as quais são transmitidas para a mesma camada de saída, fazendo uso de informações contextuais de

ambos os lados da sequência (PREMJITH; SOMAN; POORNACHANDRAN, 2018). A Figura 5 mostra o desenho de uma arquitetura bidirecional.

Figura 5 – Arquitetura Bidirecional



Fonte: Feito pelo autor, adaptado de (RAGHEB; GODY; SAID, 2021)

Conforme mostra a Figura 5, os dados são processados em ambas as direções (para frente e para trás) usando duas camadas ocultas de divisão conectadas à mesma camada de saída (RAGHEB; GODY; SAID, 2021).

Os modelos utilizados serão *Bidirectional Recurrent Neural Network* (BIRNN), *Bidirectional Long Short-Term Memory* (BILSTM) e *Bidirectional Gated Recurrent Unit* (BIGRU), ambos explicados anteriormente em suas versões unidirecionais.

4 ARQUITETURA DO EXPERIMENTO

Este capítulo descreve cada ferramenta utilizada para este estudo, sendo buscado junto a literatura as mais utilizadas e que possuem melhor performance para o cumprimento do objetivo deste trabalho.

4.1 Implementação

Python é uma linguagem com código aberto de propósito geral, usado bastante por cientistas de dados em suporte de PLN e Aprendizado de Máquina - *Machine Learning* (ML), que possui bibliotecas de DL em abundância (BEYSOLOWII, 2018).

Thanaki (2017) lista os seguintes pontos do porque o *python* é uma das melhores opções para construir um sistema especialista baseado em PLN:

- O desenvolvimento de protótipos é fácil e eficiente;
- Uma grande variedade de bibliotecas de PLN de código aberto estão disponíveis;
- O apoio da comunidade é muito forte;
- Fácil de usar e menos complexo para iniciantes;
- A otimização do sistema baseado em PLN é menos complexa em comparação com outros paradigmas de programação.

A biblioteca padrão do *python* é muito extensa, oferecendo uma ampla variedade de recursos (PYTHON. . . , 2001). As bibliotecas utilizadas neste trabalho estão listadas a seguir.

4.1.1 Numpy

Ele é usado para tarefas de manipulação de vetores e para definir tipos de dados, como *arrays*, durante a construção do modelo, sendo responsável por construir o *embedding* (KANDI, 2018).

Um *array Numpy* é uma grade de valores, todos do mesmo tipo, indexado por uma tupla de inteiros não negativos, trabalhando em formato de matriz através dos números das dimensões (HARDENIYA et al., 2016).

4.1.2 Pickle

Pickle é um módulo usado para implementar algoritmos de serialização e desserialização em uma estrutura de objeto *python* (KANDI, 2018).

Ele possui o processo pelo qual uma hierarquia de objetos *python* é convertida em um fluxo de *bytes*[8], e depois o processo inverso, pelo qual um fluxo de *bytes* (de um arquivo binário ou objeto semelhante) é convertido novamente em uma hierarquia de objetos (PYTHON. . . , 2001).

4.1.3 TensorFlow

O *TensorFlow* é uma plataforma de ML de código aberto de ponta a ponta, utilizada para DL e é fornecida pelo Google[9] (THANAKI, 2017).

O *TensorFlow* fornece Interface de Programação de Aplicação - *Application Programming Interface* (API)s[10] *Python* e *C++* estáveis, bem como API compatível com versões anteriores não garantidas para outras linguagens.

Ele tem um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que permite que os pesquisadores impulsionam o estado da arte em ML e os desenvolvedores criem e implementem facilmente aplicativos com tecnologia de ML (KERAS, 2017).

4.1.4 Keras

Keras é uma API de aprendizado profundo escrita em *Python*, executada na plataforma de aprendizado de máquina *TensorFlow*, desenvolvido com foco em permitir a experimentação rápida (KERAS, 2017).

Ele fornece implementações de várias arquiteturas de camada, funções objetivas e algoritmos de otimização necessários para construir um modelo e é usado para sequenciar os dados do texto de treinamento (KANDI, 2018).

4.1.5 SpaCy

SpaCy é um mecanismo de processamento natural de código aberto construído para *python* para tarefas de extração de informações em larga escala, usada principalmente para tokenização[11] de texto de entrada e para carregar vetores de palavras treina-

[8] É usado com frequência para especificar o tamanho ou quantidade da memória ou da capacidade de armazenamento de um certo dispositivo.

[9] www.google.com

[10] Um conjunto de normas que possibilita a comunicação entre plataformas através de uma série de padrões e protocolos.

[11] Segmentação de texto em palavras, sinais de pontuação etc.

dos com *GloVe* pré-treinados (KANDI, 2018).

Ele foi projetado especificamente para uso em produção e ajuda a criar aplicativos que processam e entendem grandes volumes de texto, sendo usado para construir extração de informações ou sistemas de compreensão de linguagem natural ou para pré-processar texto para DL (VASILIEV, 2020).

Essa biblioteca permite identificar sobre o que o texto trata, o que as palavras significam no contexto, quais seus relacionamentos, quais empresas e produtos são mencionados ou quais textos são semelhantes entre si, além de determinar se um *token*[12] é substantivo, verbo, adjetivo, advérbio ou outra classe qualquer (SANTANA et al., 2020). Além disso, também é muito eficiente para o processo de encontrar marcações gramaticais (POTA et al., 2019).

4.1.6 Pandas

O *pandas* é uma ferramenta de análise e manipulação de dados de código aberto rápida, poderosa, flexível e fácil de usar, construída sobre a linguagem de programação *python* (HARDENIYA et al., 2016). Destina-se a ser o bloco de construção fundamental de alto nível para fazer análises de dados práticos e do mundo real.

4.2 Base de Dados

Os dados para fazer os treinamentos foram retirados de três bases diferentes. Uma base é autoral, feita através da utilização do algoritmo proposto por Rodrigues, Junior e Lobato (2019). Para fins de identificação, essa base será chamada de **Base 1**[13]. A segunda é uma base proposta por Goldhahn, Eckart e Quasthoff (2012), que para este estudo será chamada de **Base 2**[14]. A terceira base de treinamento foi proposta por Fernandes et al. (2020) e será identificada como **Base 3**[15]. A Tabela 4 mostra os detalhes das bases usadas neste trabalho.

[12] Valor gerado pelo processamento do embedding
[13] https://github.com/paulamyrian/corpusG1.git
[14] https://wortschatz.uni-leipzig.de/en/download/Portuguese
[15] https://github.com/paulamyrian/corpus_tourism_amazon.git

Tabela 4 – Informações detalhadas das bases de dados

Fonte dos Dados	Características	Palavras chaves utilizadas	Autor - DATA
Portal G1 e Folha de São Paulo.	Notícias extraídas dos sites	bumba-boi; muleque; São Luís; pão com ovo;	Rodrigues, Junior e Lobato (2019)
Sites públicos	Textos de notícias e sites Web extraídos em 2011.	N/A	Goldhahn, Eckart e Quasthoff (2012)
TripAdvisor	Comentários de restaurantes da região amazônica	Santarém; Manaus; Belém;	Fernandes et al. (2020)

Fonte: Feito pelo autor.

Conforme descrito na Tabela 4, a Base 1 é formada por notícias dos portais Brasileiros do Portal G1[16] e Folha de São Paulo[17], utilizando as palavras chaves bumba-boi, muleque, São Luís e pão com ovo. A Base 2 foi retirada de uma base já pronta, com pequenos textos extraídos de sites públicos brasileiros. Por fim, a Base 3 é formada por comentários extraídos do site TripAdvisor[18], que são basicamente análises dos usuários sobre restaurantes da região Amazônica (Santarém, Manaus e Belém).

A Tabela 5 mostra as características de cada base de dados.

Tabela 5 – Características das bases de dados

	Base 1	Base 2	Base 3
Quantidade de Palavras	1.016.201	18.479.611	41.960
Quantidade de Caracteres	5.189.965	94.027.772	208.564
Tamanho do Vocabulário	18.997	6.079	3.590
Sequências Totais	159.420	15.878	15.938
Comprimento Máximo da Sequência	2.010	72	229
Tamanho da Base (MB)	6,30	111	0,27

Fonte: Feito pelo autor.

[16] https://g1.globo.com/
[17] https://www.folha.uol.com.br/
[18] https://www.tripadvisor.com.br/

Na Tabela 5 podemos notar que a Base 1 possui um maior tamanho de vocabulário, sequências e comprimento máximo das sequências. Em contrapartida, a Base 2 possui uma maior quantidade de palavras, caracteres e tamanho. Essas informações serão levadas em consideração quando for passado para a etapa de testes, a fim de analisar o comportamento com cada base.

4.3 Função de Similaridade

Em um espaço de incorporação de palavras, a similaridade entre duas palavras pode ser medida através de uma métrica de distância ou similaridade entre os vetores correspondentes no espaço, como a similaridade de cosseno (DOVAL; VILARES; GÓMEZ-RODRÍGUEZ, 2020).

A semelhança de cosseno entre dois vetores é uma medida que calcula o cosseno do ângulo entre eles, fazendo com que seja possível mostrar como duas palavras se relacionam com base no ângulo entre os vetores (KANDI, 2018). A semelhança de cosseno entre os vetores de palavras $A = \{a_1, a_2, \ldots, a_n\}$ e $B = \{b_1, b_2, \ldots, b_n\}$ é calculado como mostrado na Equação 4.1.

$$cos(\boldsymbol{A}, \boldsymbol{B}) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{||\boldsymbol{A}|| \cdot ||\boldsymbol{B}||} \tag{4.1}$$

Basicamente, a distância cosseno entre duas palavras (ou corpus) é calculada pelo produto escalar de dois vetores numéricos. É normalizado pelo produto dos comprimentos do vetor, de modo que valores de saída próximos a 1 indicam alta similaridade (SHARMA; MITTAL; VIDYARTHI, 2022). Uma alta similaridade significa que os vetores estão orientados na mesma direção e o ângulo está próximo de 0 graus e, portanto, o cosseno do ângulo está próximo de 1 (KANDI, 2018).

A função de similaridade de cosseno é muito utilizada em PLN para localização de documentos semelhantes, recuperação de informações, localização de sequência semelhante a um DNA em bioinformática e detecção de plágio (HU et al., 2019). Neste trabalho, essa função será utilizada para verificar quão próximo é a OOV do seu significado, tendo como base os vetores gerados no processo anterior.

4.4 Marcação Gramatical

Marcação gramatical - *Part-of-speech tagging* (POS) é o processo de etiquetagem de cada palavra numa frase, tendo como base a dependência sintática dessa palavra no contexto em que ela aparece (PREMJITH; SOMAN; POORNACHANDRAN, 2018). O objetivo é atribuir um rótulo sintático a cada *token* que ocorre em uma determinada

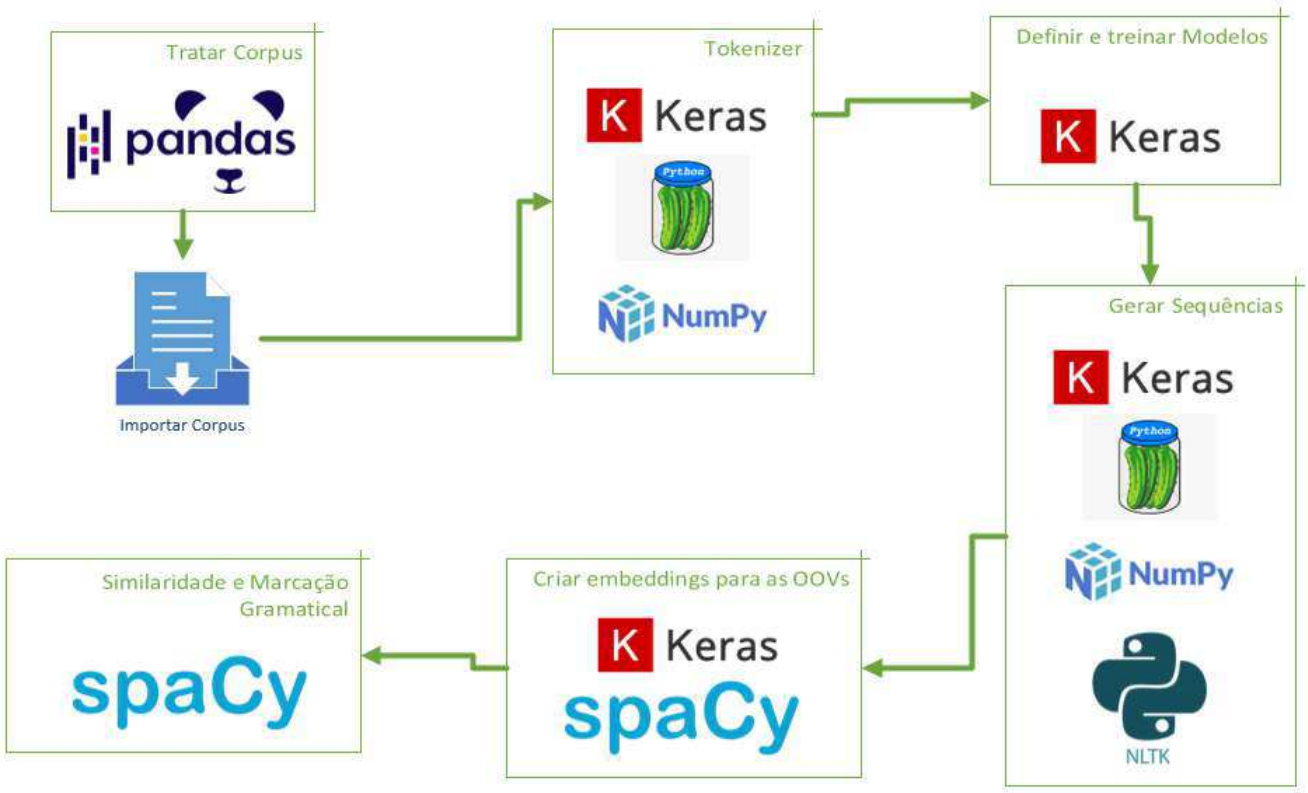
frase (como substantivo, verbo, adjetivo, pontuação, etc.).

Os métodos estatísticos e aqueles baseados em RN obtêm resultados mais eficazes, já que conseguem atingir valores de precisão para etiquetagem de POS em muitos idiomas (POTA et al., 2019).

4.5 Representação da arquitetura

Os passos da implementação, bem como a descrição de cada processo está demonstrado na Figura 6, onde está detalhado todo o processo da utilização das bibliotecas que foram apresentadas anteriormente. Os processos descritos, juntamente com as etapas serão explicadas com mais detalhes no próximo Capítulo.

Figura 6 – Fluxo da implementação do experimento



Fonte: Feito pelo autor

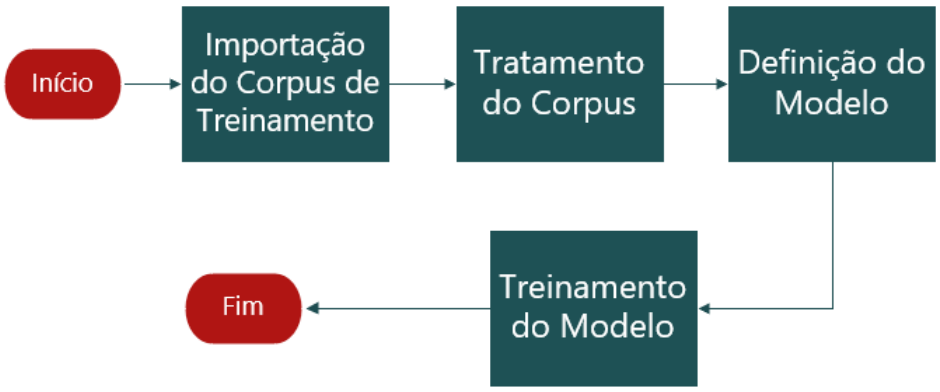
5 DESENVOLVIMENTO

Neste capítulo será mostrado a descrição de como foi realizado o processo de implementação dos modelos[19]. Essa função foi separada em duas etapas: preparação do modelo e predição de *embeddings* das OOVs. Um exemplo com os códigos completos estão nos ANEXO A e ANEXO B.

5.1 Preparação do Modelo

Para poder seguir com o processo de preparação de um modelo de predição de OOV, é necessário seguir uma ordem de processos. A ordem está mostrada na Figura 7.

Figura 7 – Fluxo para a preparação do Modelo



Fonte: Feito pelo autor

Na Figura 7 vemos que a primeira etapa segue uma sequência linear que passa pelo seguinte fluxo: Importação do Corpus de Treinamento, Tratamento do Corpus, Definição do Modelo, Treinamento do Modelo e Gerar sequências com base no modelo. Cada fluxo será detalhado nas subseções a seguir.

[19]Códigos usados: https://github.com/paulamyrian/OOV_usingRNNs.git

5.1.1 Importação do Corpus de Treinamento

Barbosa et al. (2017) define que nesta fase da preparação do modelo, um corpus grande é pré-processado para ser usado na próxima fase, onde esse corpus é *tokenizado*[20] e o texto é codificado como números inteiros.

Para isso, foi utilizado a biblioteca *pandas* (Definição mostrada no Capítulo 3) para poder carregar as bases de dados para fazer um prévio tratamento e uma análise. Esse tratamento é basicamente verificar todas as colunas que a base possui e deixar apenas as colunas pertinentes. O código utilizado para fazer essa limpeza está mostrado no Algoritmo 1.

```
1 import pandas as np
2 df = pd.read_csv('tourism_amazon.csv', encoding='utf-8')
3 df.drop(['status_id', 'link_name', 'status_type','status_link',
    'status_published', 'num_reactions', 'num_comments', 'num_shares',
    'num_likes', 'num_loves', 'num_wows', 'num_hahas', 'num_sads',
    'num_angrys', 'num_special', 'category'], axis=1, inplace=True)
4 df.to_csv('cleandata.csv',index=False)
```

Algoritmo 1: Exemplo de utilização do pandas na Base 3

No Algoritmo 1 vemos que a primeira linha há a importação da biblioteca, na segunda linha a importação do *corpus* (Nesse caso, a Base 3 não tratada), na linha três a função de deletar as colunas indesejadas e por último salvar o arquivo tratado.

5.1.2 Tratamento do Corpus

O tratamento realizado nessa etapa não se parece com o feito na subseção anterior, pois neste fluxo é feito o processo de *tokenização*. Ela quebra a sequência de caracteres em um texto localizando o limite de cada palavra, ou seja, os pontos onde uma palavra termina e outra começa (BARBOSA et al., 2017).

Conforme define Kandi (2018), o *tokenizador* é usado para ajustar o texto de origem para desenvolver o mapeamento de palavras para inteiros exclusivos e em seguida, esses números inteiros podem ser usados para sequenciar linhas de texto.

Essas sequências de textos são divididas em *tokens* antes de ser enviada para a camada de incorporação e para explorar como as incorporações de palavras têm impacto em diferentes conjuntos de dados, Wang, Nulty e Lillis (2020) propõe que seja utilizado corpus diferentes em comprimento e características, justamente para poder ter a análise sobre os impactos na detecção de *embeddings*.

[20] Segmentação de palavras

5.1.3 Definição do Modelo

Após as sequências serem geradas, há uma divisão entre os elementos de entrada e saída, para poder serem usadas em treinamento nos modelos de previsão. Os modelos utilizados para este estudo são: RNN, LSTM, GRU e os seus modelos bidirecionais. O Algoritmo 2 e 3 mostra como é a definição desses modelos.

```
1 model = Sequential()
2 model.add(Embedding(vocab_size,100, input_length=max_length-1))
3 model.add(SimpleRNN(100))
4 model.add(Dense(vocab_size, activation='softmax'))
```

Algoritmo 2: Definição de Modelo - Exemplo do RNN

```
1  # define forward sequence model
2  model_rnn = Sequential()
3  model_rnn.add(Embedding(vocab_size,100, input_length=max_length-1))
4  model_rnn.add(Bidirectional(SimpleRNN(100)))
5  model_rnn.add(Dense(vocab_size, activation='softmax'))
6  # define reverse model
7  rev_model_rnn = Sequential()
8  rev_model_rnn.add(Embedding(vocab_size,100, input_length=max_length-1))
9  rev_model_rnn.add(Bidirectional(SimpleRNN(100)))
10 rev_model_rnn.add(Dense(vocab_size, activation='softmax'))
```

Algoritmo 3: Definição de Modelo Bidirecional - Exemplo do BiRNN

No Algoritmo 2 linha 1 e no Algoritmo 3 linhas 2 e 7, temos a definição do tipo de modelo como sequencial. Como explicado no Capítulo 3, as redes recorrentes são projetadas para trabalhar com dados sequenciais e por esse motivo o modelo tem que ser também sequencial.

Seguindo, temos a linha 2 no Algoritmo 2 e as linhas 3 e 8 no Algoritmo 3 mostrando a definição da camada *embedding* com três argumentos: tamanho do vocabulário nos dados de texto, tamanho do espaço vetorial no qual as palavras serão incorporadas e comprimento das sequências de entrada.

Na definição da RN propriamente dita, temos no Algoritmo 2 linha 3 e no Algoritmo 3 linhas 4 e 9, onde o primeiro define o RNN em sua maneira simples com 100 unidades internas e o segundo define o RNN em sua forma bidirecional com também 100 unidades internas.

Depois, para produzir uma representação oculta de cada palavra na frase, é necessário prever seus rótulos (COSTA; PAETZOLD, 2018). A camada de saída é

uma camada densa composta por um neurônio para cada palavra do vocabulário e ela usa uma função *SoftMax* para garantir que a saída seja normalizada para retornar uma probabilidade (KANDI, 2018; SHEWALKAR, 2018). Essa representação está definida na linha 4 do Algoritmo 2 e nas linhas 5 e 10 do Algoritmo 3.

Essa forma de definição do modelo de processamento está descrita na documentação do Keras (2017) e nos trabalhos de Kandi (2018), Aydoğan e Karci (2020) e Thanaki (2017).

5.1.4 Treinamento do Modelo

O texto codificado para sequências é compilado de acordo com suas respectivas redes neurais. Para executar o processo de treinamento sem problemas, precisamos definir os hiperparâmetros do modelo de DL (GOYAL; GUPTA; KUMAR, 2021). O hiperparâmetros usados estão definidos na Tabela 6.

Tabela 6 – Hiperparâmetros de treinamento dos Modelos

Hiperparâmetro	**Valor**
Número de epochs	200
Tamanho do batch	100
Verbose	2
Taxa de Aprendizagem	0.001
Otimizador	Adam
Tamanho do Embedding	100

Fonte: Feito pelo autor.

A Tabela 6 mostra o otimizador *Adam*, que para Goyal, Gupta e Kumar (2021) é o otimizador que mais gera resultados de última geração. Na biblioteca *Keras*, a taxa de aprendizado padrão do otimizador *Adam* é definida como 0,001.

A quantidade de *epochs* é achada em treinamentos, dependendo do modelo utilizado, a fim de definir um valor que não seja muito grande para achar ruídos e muito pequeno para não aprender o suficiente (DEEP..., 2022). Sendo assim, o número de *epochs* foi definido como 200 com um tamanho de lote de 100, pois Kandi (2018) afirma que esse valor permite maior precisão do modelo a um custo de tempo necessário para treinamento.

A verbose foi definido como 2 porque uma linha tinha que ser definida para cada treinamento com informações sobre acurácia e perda sem a necessidade de mostrar animação (GULLI; PAL, 2017). O Algoritmo 4 mostra a aplicação desses hiperparâmetros na prática.

No Algoritmo 4 na linha 2 podemos ver a aplicação do otimizador, das métricas

```
1 # compile forward sequence network
2 model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
    metrics=['accuracy'])
3 # fit network
4 model.fit(X, y,batch_size=100, epochs=200, verbose=2)
5 # save the model to file
6 model.save('model.h5')
```

Algoritmo 4: Treinamento dos Modelos

e do tipo de função que irá calcular a perda de entropia cruzada entre os rótulos e as previsões, que nesse caso foi utilizada a *Sparse Categorical Crossentropy* pois produz um índice de categoria correspondente mais provável, economizando espaço e tempo (KERAS, 2017).

Na linha 4 é feita a declaração do treinamento através do método *fit*, passando as informações de entrada (X), destino (y), número de amostras (*batch_size*), número de épocas para treinar o modelo e o modo de verbosidade. Por fim, na linha 6, é realizado o processo de guardar as informações do treinamento, através do método *save*, em um arquivo tipo *h5*[21].

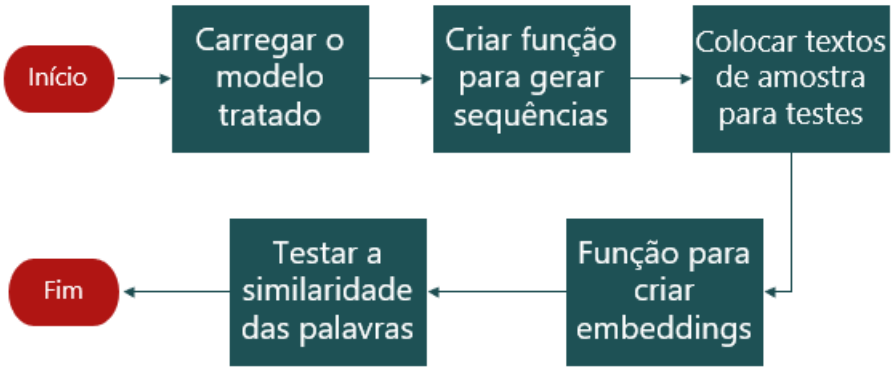
Através das informações do treinamento, é possível gerar o gráfico que mostra a curva de aprendizado e é possível verificar a acurácia e taxa de perda. Elas funcionam de forma contrária: *accuracy* é melhor quanto maior (se aproxima de 100%) e *loss* é melhor quanto menor (se aproxima de 0). Os resultados deste treinamento serão detalhados no próximo Capítulo (6).

5.2 Predição de embeddings das OOVs

Após a primeira etapa, os modelos de predição já estão preparados para poderem ser usados. Dessa maneira, será realizado uma sequência de etapas, conforme é mostrado na Figura 8.

[21] Arquivo de dados salvo no Formato de Dados Hierárquico que possui matrizes multidimensionais de dados científicos

Figura 8 – Fluxo para a predição de Embeddings



Fonte: Feito pelo autor

Na Figura 8 vemos que a segunda etapa segue uma sequência linear que passa pelo seguinte fluxo: Carregar o modelo tratado, criar função para gerar sequências, Colocar textos de amostra para testes, Função para criar embeddings e Testar a similaridade das palavras. Cada fluxo será detalhado nas subseções a seguir.

5.2.1 Carregar o Modelo Tratado

O modelo gerado ao final da primeira etapa (mostrado na subseção 5.1.4) é carregado no começo desta segunda etapa. Esse processo é mostrado no Algoritmo 5.

```
1 # unidirectional
2 model_uni = load_model('model.h5')
3 # bidirectional
4 model_bi = load_model('model_bi.h5')
5 model_revbi = load_model('rev_model_bi.h5')
```

Algoritmo 5: Carregando os Modelos

Nas linhas 2,4 e 5 do Algoritmo 5 podemos ver que a função carrega os modelos, sendo na linha 2 o modelo treinado por uma RNN e nas linhas 4 e 5 o treinado por um modelo bidirecional.

Para ajudar na descoberta das OOVs será utilizado a biblioteca *SpaCy*, pois ele servirá como um componente do aplicativo através de uma segmentação de instruções (*pipeline*) para a linguagem portuguesa (VASILIEV, 2020). Esse carregamento é mostrado no Algoritmo 6.

Na linha 1 do Algoritmo 6 podemos notar que é feito o processo de carregamento de um *pipeline* na linguagem portuguesa otimizado para CPU (SPACY, 2016).

```
1 nlp = spacy.load('pt_core_news_md')
```

Algoritmo 6: Pipeline português

Dessa maneira será integrado os componentes treinados de cada base utilizada ao modelo personalizado.

5.2.2 Gerador de sequências

A função que vai fazer o processo de gerar as sequências recebe um texto inicial (palavras semente), *tokenizador*, o comprimento máximo da sequência e o modelo treinado como entrada. Depois, ela irá prever a distribuição de palavras prováveis e retornar com o vocabulário previsto. Esta função está representada no Algoritmo 7.

```
1 Função gerar_sequencia(modelo, tokenizador, tamanho,
    palavras_semente):
2     for número fixo de palavras do
3         codifica o texto como inteiro
4         prevê probabilidades para cada palavra
5         mapeia o índice de palavras previstas para a palavra
6     end
7     return palavra_prevista;
```

Algoritmo 7: Pseudocódigo da Função Geradora de Sequência

5.2.3 Textos de amostras para testes

Para saber o desempenho do modelo e se ele está aprendendo as coisas certas, será necessário utilizar dados de avaliação, pois o propósito é que ele apresente uma teoria que possa ser generalizada em dados não vistos (SPACY, 2016). Será utilizado textos extraídos do trabalho de Neres e Barros (2011), pois contém muitas frases com palavras desconhecidas e, junto a elas, o significado. Na documentação da biblioteca SpaCy (2016) explica que:

> Ao treinar um modelo, não queremos apenas que ele memorize nossos exemplos – queremos que ele apresente uma teoria que possa ser generalizada em dados não vistos. Afinal, não queremos apenas que o modelo aprenda que esta instância de "Amazon" bem aqui é uma empresa – queremos que ele aprenda que "Amazon", em contextos como este, é provavelmente uma empresa.

Buck e Vlachos (2021) afirma que para obter uma avaliação intrínseca dos métodos proposto é necessário ocorrências naturais das palavras OOV e a documentação do

SpaCy (2016) afirma que para saber o desempenho do modelo e se ele está aprendendo as coisas certas, é preciso dados de avaliação, pois somente assim é possível avaliar se a representação possuirá um sentido real.

Os textos de amostras foram retirados do trabalho de Neres e Barros (2011), que possui um aparato de palavras do dialeto Maranhense e que, consequentemente, são OOVs. Foi escolhido deste as seguintes palavras com seus respectivos significados:

- **“bofetes”**: Soco, tapa ou murro;
- **“escritinho”**: Muito parecido, semelhante ou igual;
- **“maroca”**: Pessoa fofoqueira, que gosta de falar da vida alheia ou de saber o que está acontecendo com os outros;
- **“marocar”**: Observar a vida alheia para tecer comentários com outras pessoas;
- **“mucura”**: Além de referir-se a um animal (vulgo gambá), essa palavra também serve como gíria indicativa de pessoa muito feia.

Foi verificado que todas essas palavras escolhidas não possuíam vetores válidos, assim como mostra a Figura 9 com o exemplo da palavra “mucura”.

Figura 9 – Embedding original da palavra “mucura”

```
nlp.vocab.get_vector('mucura')
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Fonte: Feito pelo autor

A partir destas palavras, foi retirado as frases que as acompanhavam. Neste ponto já é possível saber se após o treinamento os resultados vão ser satisfatórios.

5.2.4 Função para criar Embeddings

A função para criar *embeddings* trabalha junto com a função de gerar sequências. Essas previsões são usadas para definir uma média ponderada das incorporações das palavras previstas, que é atribuída ao vocabulário do modelo pré-treinado. Este método de atribuição de embeddings é usado para que as palavras OOV tenham uma posição razoável no espaço vetorial com base em seu contexto (KANDI, 2018). Esta função está representada no Algoritmo 8.

```
1 Função criar_embedding(documento):
2     for token em documento do
3         if tem OOV no documento then
4             o embedding é atribuído à palavra OOV com base no contexto
5             gerar sequencia
6             o embedding da OOV é atualizado ao modelo pré-treinado
7         end
8     end
```

Algoritmo 8: Pseudocódigo da Função para criar Embeddings para as OOVs

Após as palavras serem convertidas em *tokens*, no conjunto de dados onde será feito o processo de classificação, os valores numéricos das palavras sob a forma de vectores foram retirados e os que são compostos por zeros denotam palavras que são consideradas fora de vocabulário (AYDOĞAN; KARCI, 2020). São essas palavras que passam na estrutura condicional (if) dentro do laço de repetição (for) do Algoritmo 8. Esta etapa é crucial para o bom andamento do processo, porque é dela que será feito o processo de transformação dos vetores.

Após esse processo, as OOVs passam a possuir vetores válidos. Isso é mostrado na Figura 10, tendo como exemplo a palavra “mucura”.

Figura 10 – Embedding da palavra “mucura” após passar pela função

```
nlp.vocab.get_vector('mucura')
```

```
array([ -46.3881   , -104.2602   ,   37.4253   ,   22.917841 ,
          6.49506  ,   13.97748  ,  -37.4319   ,   56.2254   ,
        -16.70856  ,   87.00449  ,    3.016332 ,    0.337161 ,
         48.4539   ,  -83.754005 ,   44.7447   ,  -58.0536   ,
         46.447502 ,   58.2219   ,  103.7025   ,  130.7955   ,
         42.9231   ,   17.124691 ,  -65.5413   ,   49.7574   ,
        -32.83071  ,  -71.4054   ,  -60.3537   ,   88.9713   ,
          1.164438 ,   67.224304 ,   72.2634   ,   66.7755   ,
        178.8699   ,  -97.0926   ,   95.1126   ,   13.56102  ,
         11.083381 ,  -95.9607   ,  -22.20471  , -134.4552   ,
         26.120491 ,   76.9329   ,  -27.65136  ,   29.856419 ,
         22.03674  ,   11.01837  ,   52.437    ,   13.58874  ,
        158.4561   ,   19.29873  ,   98.9373   ,   49.9323   ,
          0.2353956,   61.0434   ,   22.08789  ,  -65.3565   ,
         84.3942   ,   26.36931  ,   -9.433049 ,   88.205696 ,
         19.19973  ,  135.7653   ,   73.9068   ,  119.8626   ,
        -55.6446   ,  -68.9238   , -115.1667   ,   22.94853  ,
        -14.96286  ,  -38.8311   ,  -68.6697   ,    2.373063 ,
         10.5138   ,   90.3639   ,   16.40595  ,  111.5235   ,
```

Fonte: Feito pelo autor

5.2.5 Similaridade

A função utilizada é a similaridade por cosseno (descrita na subseção 4.3). *SpaCy* possui a função que é capaz de comparar dois objetos e fazer uma previsão de quão semelhantes eles são, sendo que a similaridade é sempre subjetiva, dado o contexto em que está inserido e o treinamento utilizado (SPACY, 2016).

O *SpaCy* tem seus próprios vetores de palavras integrados de acordo com cada idioma (VASILIEV, 2020). O *SpaCy* possui o seu próprio método se similaridade por cosseno ($similarity()$). Vale ressaltar que a função de similaridade depende diretamente dos processos anteriores, visto que um correto processo de criação de *embeddings* ajuda na precisão deste.

6 RESULTADOS

O processo de obtenção dos resultados foi padronizado para todas as bases (descrito na subseção 5.1.4), pois será necessário fazer uma medição qualitativa (SULEIMAN; AWAJAN, 2020). Os modelos foram treinados em uma máquina equipada com Intel Core i5, 8GB de RAM e NVIDIA GTX 1650 Ti 4GB GDDR6.

Este capítulo relata um resumo dos resultados separados por base e, ao final, uma comparação entre elas. Para não ficar uma tabela de valores muito extensa, foi colocado nos APÊNDICE A e APÊNDICE B todos os resultados detalhalhos da POS de cada palavra e da semelhança de cosseno entre pares de palavras. Além disso, será detalhado também os resultados do aprendizado dos treinamentos, usando as métricas de *Accuracy* e *Loss*.

6.1 Base 1

Foi realizado o processo de treinamento tendo como *corpus* a Base 1. Na Figura 11 podemos notar que à medida que os *epochs* de treinamento passam a porcentagem de *accuracy* aumenta e a de *loss* diminui.

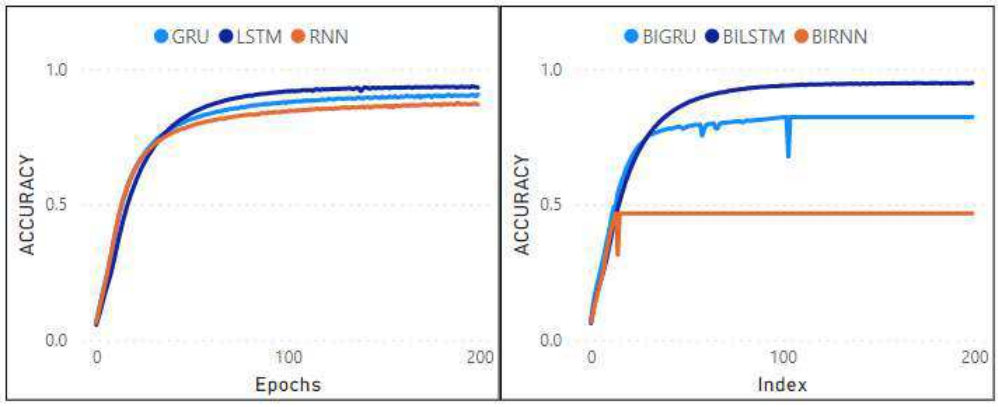
Nesta base, os modelos BIRNN e BIGRU tiveram uma curva de aprendizagem ínfima quando comparada aos outros modelos, conforme demonstra a Figura 11. O modelo BIRNN foi o que teve o pior desempenho, perdendo até para a sua versão unidirecional. Os dados estão detalhados na Tabela 7.

Tabela 7 – Resultados do treinamento na Base 1

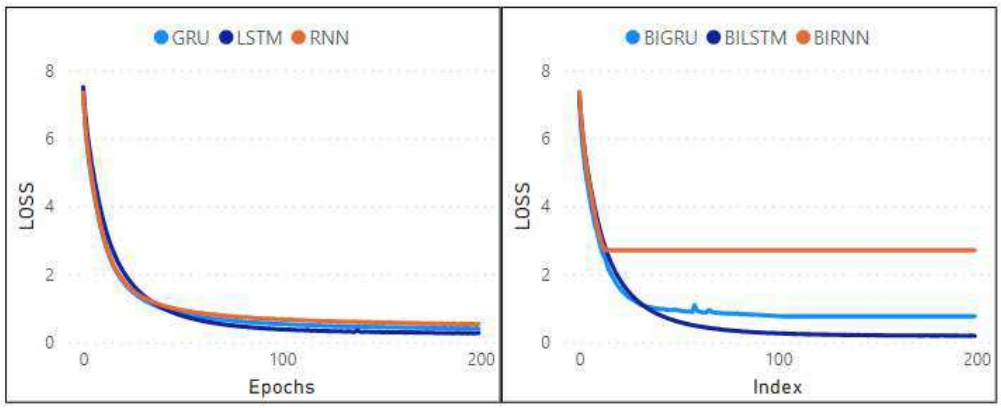
	Accuracy	**Loss**	**Tempo de Treinamento**
RNN	86.89%	0.5398	**3d 9h 10m 17s**
LSTM	93.21%	0.2715	8d 20h 10m 10s
GRU	90.57%	0.3970	11d 3h 25m 19s
BIRNN	46.72%	2.7063	10d 23h 34m 56s
BILSTM	**94.92%**	**0.1841**	25d 2h 33m
BIGRU	82.31%	0.7624	21d 10h 32m 13s

Fonte: Feito pelo autor.

Figura 11 – Linha do tempo do treinamento da Base 1



(a) Accuracy



(b) Loss

Fonte: Feito pelo autor

Os modelos bidirecionais foram os que mais tiveram discrepâncias de valores, pois os modelos unidirecionais ficaram bem próximos na curva de aprendizado e nos valores de *accuracy* e *loss*. Com relação ao tempo que demorou no treinamento, o RNN foi o que obteve o menor tempo e o BILSTM o maior.

Na detecção de similaridade, pegando o exemplo da palavra “maroca”, foi realizada a comparação com duas (2) frases contendo algum significado dela. Segue as frases e os resultados em porcentagem da similaridade:

- Patrícia é a mulher mais **maroca** da rua. Ela sabe da vida de todo mundo.
  - **Fofoquei** com a vizinha sobre o problema do meu irmão. - 48,29%
  - Ela não passa de uma **fofoqueira** que só cuida da vida dos outros e não sabe guardar segredos! - 59,53%

Pegando como exemplo a palavra “maroca”, que possui o significado de “pessoa fofoqueira”, podemos assumir que a marcação gramatical dela é de adjetivo, tendo

como base a sua palavra semelhante. A Tabela 8 mostra como se comportou cada modelo na detecção de POS para a palavra “maroca”.

Tabela 8 – Resultados da marcação gramatical utilizando a Base 1 na palavra maroca

RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
verbo	adjetivo / substantivo	verbo	substantivo	substantivo	substantivo

Fonte: Feito pelo autor.

Como demonstrado na Tabela 8, o modelo treinado em LSTM foi o único que conseguiu marcar gramaticalmente correto a palavra “maroca” (outras palavras e frases estão detalhadas no APÊNDICE B).

6.2 Base 2

O treinamento realizado na Base 2 foi semelhante ao anterior, onde obtivemos uma curva de aprendizado mais próxima, assim como mostrado na Figura 12. É possível notar que os modelos LSTM e BILSTM precisaram de mais *epochs* para poder chegar a uma constância, mais precisamente o primeiro.

Os valores de *accuracy* e *loss* foram muito bons, mais precisamente os modelos GRU e RNN. As taxas de *accuracy* foram tão boas que nenhum modelo fez abaixo de 99% de precisão, sendo que vale ressaltar que o modelo GRU chegou bem próximo de 100%. A Tabela 9 mostra o resultado do treinamento.

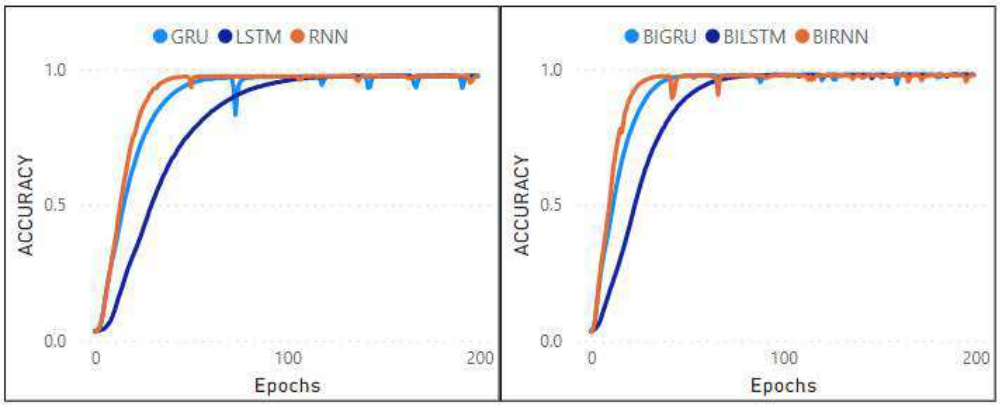
Tabela 9 – Resultados do treinamento na Base 2

	Accuracy	Loss	Tempo de Treinamento
RNN	99.98%	**0.0018**	**31m 56s**
LSTM	99.98%	0.0054	1h 8m 9s
GRU	**99.99%**	0.0039	1h 3m 36s
BIRNN	99.48%	0.0123	2h 29m 59s
BILSTM	99.44%	0.0118	5h 20m 46s
BIGRU	99.45%	0.0098	5h 13m 27s

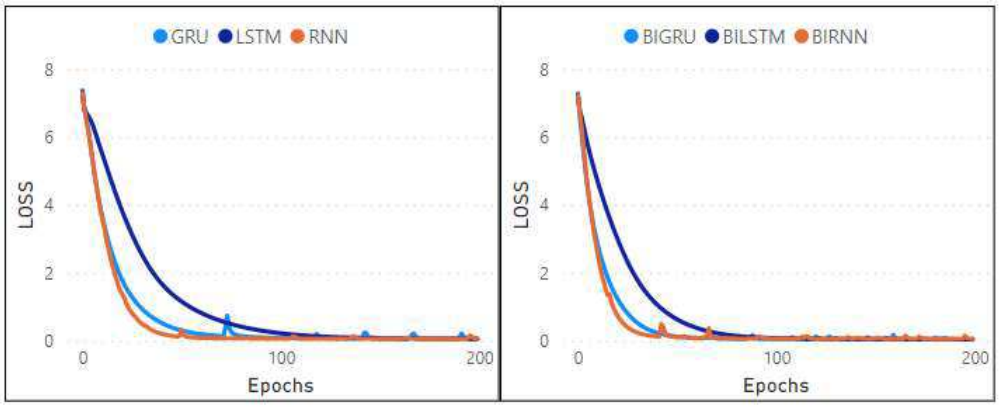
Fonte: Feito pelo autor.

Nesta Tabela 9 também é mostrado que o modelo RNN obteve uma perda ínfima e um tempo menor no seu treinamento. Observa-se que os treinamentos realizados na Base 1 demoravam dias em cada modelo e os da Base 2 não demorou mais de um (1) dia para treinar todos os seis (6) modelos.

Figura 12 – Linha do tempo do treinamento da Base 2



(a) Accuracy



(b) Loss

Fonte: Feito pelo autor

Na detecção de similaridade, pegando o exemplo da palavra “marocar”, foi realizado a comparação com duas (2) frases contendo algum significado dela. Segue as frases e os resultados em porcentagem da similaridade:

- Em alguns bairros, nem se pode falar com os amigos, que logo aparece alguém para **marocar**.
  - **Fofoquei** com a vizinha sobre o problema do meu irmão. - 48,69%
  - Ela não passa de uma **fofoqueira** que só cuida da vida dos outros e não sabe guardar segredos! - 76,91%

Pegando como exemplo a palavra “marocar”, que possui o significado de “fofocar”, podemos assumir que a marcação gramatical dela é de verbo, tendo como base a sua palavra semelhante. A Tabela 10 mostra como se comportou cada modelo na detecção de POS para a palavra “marocar”.

Tabela 10 – Resultados da marcação gramatical utilizando a Base 2 na palavra marocar

RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
numeral	pronome	substantivo	pronome	pronome	verbo

Fonte: Feito pelo autor.

Como demonstrado na Tabela 10, o modelo treinado em BIGRU foi o único que conseguiu marcar gramaticalmente correto a palavra "marocar" (outras palavras e frases estão detalhadas no APÊNDICE B).

6.3 Base 3

Assim como nas outras bases, o treinamento desta não foi diferente. A curva de aprendizado seguiu semelhante à que foi mostrada com a Base 2. Como mostrado na Figura 13, os modelos LSTM e BILSTM foram os que mais demoraram para chegar em uma constância.

Pode-se notar na Figura 13 que a partir dos 100 *epochs* os modelos começaram a ficar bem próximos. Essa afirmação fica bem evidente quando verificamos os resultados mostrados na Tabela 11. A diferença de *accuracy* e *loss* entre eles é muito baixa. Diferença essa que se torna menor da que foi apresentada no treinamento das outras bases.

Tabela 11 – Resultados do treinamento na Base 3

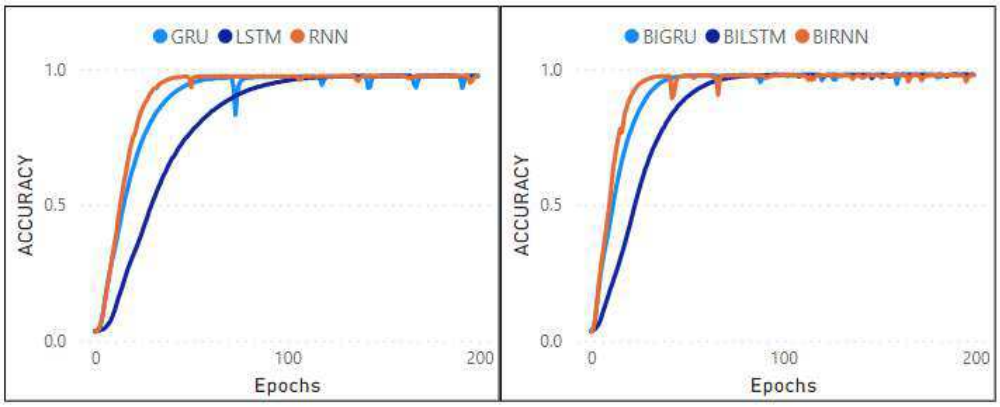
	Accuracy	Loss	Tempo de Treinamento
RNN	97.41%	0.0618	**34m 1s**
LSTM	97.54%	0.0569	2h 13m 21s
GRU	97.50%	0.0615	1h 57m 9s
BIRNN	97.73%	0.0605	5h 52m 42s
BILSTM	**97.96%**	**0.0430**	15h 35m 11s
BIGRU	97.81%	0.0547	15h 46m 58s

Fonte: Feito pelo autor.

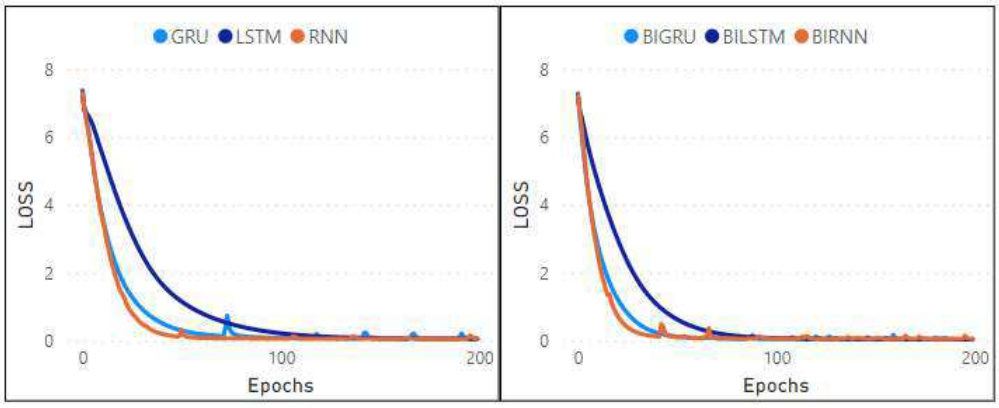
Com relação ao fator tempo de treinamento, podemos notar uma discrepância de valores, principalmente com relação aos modelos BILSTM e BIGRU. Porém, ainda é um tempo menor do que o apresentado nos treinamentos com a Base 1.

Na detecção de similaridade, pegando o exemplo da palavra "maroca", foi realizada a comparação com duas (2) frases contendo algum significado dela. Segue as frases e os resultados em porcentagem da similaridade:

Figura 13 – Linha do tempo do treinamento da Base 3



(a) Accuracy



(b) Loss

Fonte: Feito pelo autor

- Marilda levou uns **bofetes** do marido em plena Rua Grande.
  - Eu queria mesmo era dar um **soco** na tua cara! - 48,76%
  - Todo dia um **tapa** na cara diferente pra eu aprender que ninguém tem a mesma consideração por mim. - 57,77%

Pegando como exemplo a palavra “bofetes”, que possui os significados de “soco”, “tapa” e “murro”, podemos assumir que a marcação gramatical dela é de substantivo, tendo como base a sua palavra semelhante. A Tabela 12 mostra como se comportou cada modelo na detecção de POS para a palavra “bofetes”.

Tabela 12 – Resultados da marcação gramatical utilizando a Base 3 na palavra bofetes

RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
advérbio	advérbio	numeral	substantivo	substantivo	substantivo

Fonte: Feito pelo autor.

Como demonstrado na Tabela 12, os modelos treinados em BIRNN, BILSTM e BIGRU foram os que conseguiram marcar gramaticalmente correto a palavra “bofetes” (outras palavras e frases estão detalhadas no APÊNDICE B).

6.4 Comparação entre as Bases

No decorrer das seções anteriores, foram feitas algumas breves comparações entre as bases, que envolviam todos os resultados dos treinamentos. É notório que os treinamentos utilizando a Base 2 tiveram resultados melhores se comparado com as outras bases. Isso também refletiu no processo de reconhecimento gramatical, pois mostrou que um bom resultado de treinamento ajuda na correta implementação de *embeddings* e no processo de verificação de semelhanças.

Na análise por modelo, a Tabela 13 mostra os melhores de cada base. Quanto ao menor tempo de treinamento, o RNN se saiu melhor em todos. O modelo BILSTM obteve resultados significativos nas Bases 1 e 2, mas não conseguiu nenhum resultado bom na Base 2. O modelo BIGRU não conseguiu destaque em nenhum dos resultados.

Tabela 13 – Melhores resultados de cada Base

Base	**Accuracy**	**Loss**	**Similaridade**	**Marcação Gramatical (POS)**	**Tempo de Treinamento**
1	BILSTM	BILSTM	LSTM	LSTM, BILSTM	RNN
2	GRU	RNN	BIRNN	RNN	RNN
3	BILSTM	BILSTM	BILSTM	GRU, BIRNN, BILSTM, BIGRU	RNN

Fonte: Feito pelo autor.

Quanto aos resultados da semelhança entre os pares de palavras, detalhado no APÊNDICE A, podemos verificar que o treinamento realizado na Base 2 obteve uma maior quantidade de destaques nesta verificação (os resultados bons de cada base ficaram em negrito e os melhores de todas as bases foram grifados de amarelo).

Relembrando as informações de cada base, que foi descrito na Tabela 5, é possível afirmar que a base com o maior tamanho, maior quantidade de palavras e menor comprimento de sequência obteve os melhores resultados.

7 CONSIDERAÇÕES FINAIS

Este trabalho mostrou o desenvolvimento de modelos capazes de reconhecer OOVs, seguindo uma sequência de processos que foram cruciais para o cumprimento do objetivo proposto. A saber, apresentar modelos que tratem palavras fora do vocabulário. Além de mostrar esses modelos, o plano também envolve avaliações intrínsecas e extrínsecas, para verificar a eficiência nas incorporações de *embeddings*. Os modelos escolhidos foram os baseados em redes neurais recorrentes em suas versões unidirecionais: RNN, LSTM e GRU, e bidirecionais: BIRNN, BILSTM e BIGRU. As métricas de avaliação intrínseca: *accuracy* e *loss*, e extrínseca: similaridade e marcação de POS.

O modelo que melhor obteve resultado nos treinamentos foi o BILSTM, obtendo a maior *accuracy* e menor *loss* em duas das três bases. O modelo mais rápido para treinar é o RNN, obtendo um tempo menor em todas as três bases. Com relação à precisão na similaridade, os modelos LSTM (na Base 1), BIRNN (na Base 2) e BILSTM (na Base 3) foram melhores. Quanto à marcação de POS, não houve um destaque maior de um modelo em específico para as três bases.

As três bases utilizadas no treinamento possuíam características diferentes entre si, onde estava envolvido o tamanho do vocabulário, quantidade de palavras, quantidade de caracteres, comprimento das sequências e tamanho da base. Elas foram muito úteis para poder fazer uma análise de qual *corpus* utilizar para tarefas de reconhecimento e tratamento de OOVs na linguagem portuguesa.

Na visão de *corpus*, o que foi melhor em todos os aspectos foi a Base 2. Com ela, foi possível chegar a uma acurácia de 99,99% com o GRU, menores valores de perda (0,0018) e tempo de treinamento (31 minutos e 56 segundos) com o RNN, maior quantidade de valores altos nos processos de similaridade entre pares de palavras. Ressaltando que seu vocabulário e seu comprimento das sequências eram os menores das três bases, porém a quantidade de palavras, caracteres e o tamanho da base era o maior dos três.

Em uma visão geral, todos os modelos treinados em todas as bases conseguiram realizar o reconhecimento das palavras desconhecidas. Também conseguiram fazer a marcação gramatical e a verificação da similaridade das OOVs.

Certo que este trabalho tem uma grande contribuição para o corpo acadêmico,

pois foi possível fazer o tratamento de palavras fora do vocabulário utilizando a linguagem portuguesa e expressões linguísticas da cultura Maranhense, além de verificar, através dos resultados obtidos, qual o melhor *corpus* para fazer o PLN para esse fim.

Como proposta de trabalhos futuros, pretende-se utilizar o aprendizado deste estudo para poder auxiliar em várias aplicações, como o desenvolvimento de detectores de ódio que consideram o regionalismo, enriquecimento das plataformas de análise de dados Gestão de relacionamento com o cliente - *Customer relationship management* (CRM) social e estudos linguísticos computacionais em geral.

REFERÊNCIAS

AYDOĞAN, M.; KARCI, A. Improving the accuracy using pre-trained word embeddings on deep neural networks for turkish text classification. *Physica A: Statistical Mechanics and its Applications*, Elsevier, v. 541, p. 123288, 2020.

BARBOSA, J. et al. Introdução ao processamento de linguagem natural usando python. *III Escola Regional de Informatica do Piauí*, v. 1, p. 336–360, 2017.

BEYSOLOWII, T. *Applied Natural Language Processing with Python: Implementing Machine Learning and Deep Learning Algorithms for Natural Language Processing*. [S.l.]: Apress, 2018.

BUCK, G.; VLACHOS, A. Trajectory-based meta-learning for out-of-vocabulary word embedding learning. *arXiv preprint arXiv:2102.12266*, 2021.

CHENHAO, Z.; CHENGYAO, W. Named entity recognition in steel field based on bilstm-crf model. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2019. v. 1314, n. 1, p. 012217.

CHO, M. et al. Combinatorial feature embedding based on cnn and lstm for biomedical named entity recognition. *Journal of biomedical informatics*, Elsevier, v. 103, p. 103381, 2020.

COSTA, P. d.; PAETZOLD, G. H. Effective sequence labeling with hybrid neural-crf models. In: SPRINGER. *International Conference on Computational Processing of the Portuguese Language*. [S.l.], 2018. p. 490–498.

DEEP Learning Book. In: . Data Science Academy, 2022. Disponível em: <https://www.deeplearningbook.com.br/>. Acesso em: 10 jan. 2022.

DOVAL, Y.; VILARES, J.; GÓMEZ-RODRÍGUEZ, C. Towards robust word embeddings for noisy texts. *Applied Sciences*, MDPI, v. 10, n. 19, p. 6893, 2020.

FERNANDES, L. C. et al. An extensive analysis of online restaurant reviews: a case study of the amazonian culinary tourism. In: IEEE. *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. [S.l.], 2020. p. 81–84.

GOLDHAHN, D.; ECKART, T.; QUASTHOFF, U. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. [S.l.: s.n.], 2012. p. 759–765.

GOYAL, A.; GUPTA, V.; KUMAR, M. A deep learning-based bilingual hindi and punjabi named entity recognition system using enhanced word embeddings. *Knowledge-Based Systems*, Elsevier, v. 234, p. 107601, 2021.

GULLI, A.; PAL, S. *Deep learning with Keras*. [S.l.]: Packt Publishing Ltd, 2017.

HARDENIYA, N. et al. *Natural language processing: python and NLTK*. [S.l.]: Packt Publishing Ltd, 2016.

HU, Z. et al. Few-shot representation learning for out-of-vocabulary words. *arXiv preprint arXiv:1907.00505*, 2019.

JETTAKUL, A. et al. A comparative study on various deep learning techniques for thai nlp lexical and syntactic tasks on noisy data. In: IEEE. *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. [S.l.], 2018. p. 1–6.

JUN, W.; BIN, G.; CHUNHUI, H. Domain neural chinese word segmentation with mutual information and entropy. In: *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City*. [S.l.: s.n.], 2019. p. 75–79.

KANDI, S. M. *Language Modelling for Handling Out-of-Vocabulary Words in Natural Language Processing*. Tese (Doutorado) — Doctoral dissertation, 2018.

KERAS. In: . Keras, 2017. Disponível em: <https://keras.io/>. Acesso em: 17 fev. 2021.

KHALIFA, M.; SHAALAN, K. Character convolutions for arabic named entity recognition with long short-term memory networks. *Computer Speech & Language*, Elsevier, v. 58, p. 335–346, 2019.

KIM, Y. et al. Learning to generate word representations using subword information. In: *Proceedings of the 27th International Conference on Computational Linguistics*. [S.l.: s.n.], 2018. p. 2551–2561.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering–a tertiary study. *Information and software technology*, Elsevier, v. 52, n. 8, p. 792–805, 2010.

LI, C. et al. A computational approach to finding contradictions in user opinionated text. In: IEEE. *2018 IEEE/ACM International Conference on Advances in Social Networks*

Analysis and Mining (ASONAM). [S.l.], 2018. p. 351–356.

LIAO, X. et al. Efficient estimate of low-frequency words' embeddings based on the dictionary: A case study on chinese. *Applied Sciences*, MDPI, v. 11, n. 22, p. 11018, 2021.

LOBATO, F. M. et al. Social crm: A literature review based on keywords network analysis. In: SPRINGER. *International Conference on Business Information Systems*. [S.l.], 2020. p. 237–249.

MAMANDIPOOR, B. et al. Monitoring and detecting faults in wastewater treatment plants using deep learning. *Environmental monitoring and assessment*, Springer, v. 192, n. 2, p. 1–12, 2020.

MENG, W. et al. A semantic-aware representation framework for online log analysis. In: IEEE. *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. [S.l.], 2020. p. 1–7.

MORCHID, M. Parsimonious memory unit for recurrent neural networks with application to natural language processing. *Neurocomputing*, Elsevier, v. 314, p. 48–64, 2018.

NERES, J.; BARROS, L. Maranhão na ponta da língua. 2011.

PARK, S.-S. et al. Robust sentence classification by solving out-of-vocabulary problem with auxiliary word predictor. In: SPRINGER. *Pacific Rim International Conference on Artificial Intelligence*. [S.l.], 2019. p. 338–350.

POTA, M. et al. Multilingual pos tagging by a composite deep architecture based on character-level features and on-the-fly enriched word embeddings. *Knowledge-Based Systems*, Elsevier, v. 164, p. 309–323, 2019.

PREMJITH, B.; SOMAN, K.; POORNACHANDRAN, P. A deep learning based part-of-speech (pos) tagger for sanskrit language by embedding character level features. In: *FIRE*. [S.l.: s.n.], 2018. p. 56–60.

PYTHON Documentation. In: . Python, 2001. Disponível em: <https://www.python.org/doc/>. Acesso em: 15 abr. 2020.

RAGHEB, A. N.; GODY, A.; SAID, T. Comparative study of different types of rnn in speech classification. *The Egyptian Journal of Language Engineering*, Egyptian Society of Language Engineering (ESOLE), v. 8, n. 1, p. 1–16, 2021.

RAMDHANI, A.; RAMDHANI, M. A.; AMIN, A. S. Writing a literature review research paper: A step-by-step approach. *International Journal of Basic and Applied Science*, Insan Akademika, v. 3, n. 1, p. 47–56, 2014.

RODRIGUES, L. D.; JUNIOR, J. L. da S.; LOBATO, F. M. A culpa é dela! e isso o que dizem nos comentários das notıcias sobre a tentativa de feminicıdio de elaine caparroz. In: SBC. *Anais do VIII Brazilian Workshop on Social Network Analysis and Mining*. [S.l.], 2019. p. 47–58.

SANTANA, J. B. d. et al. Desenvolvimento e análise de corpus para reconhecimento de entidades nomeadas em relatórios de inteligência financeira. 2020.

SHARMA, V. K.; MITTAL, N.; VIDYARTHI, A. Context-based translation for the out of vocabulary words applied to hindi-english cross-lingual information retrieval. *IETE Technical Review*, Taylor & Francis, v. 39, n. 2, p. 276–285, 2022.

SHEWALKAR, A. N. Comparison of rnn, lstm and gru on speech recognition data. North Dakota State University, 2018.

SPACY. In: . SpaCy, 2016. Disponível em: <https://spacy.io/>. Acesso em: 17 fev. 2021.

SPOLAÔR, N. et al. A systematic review on experimental multi-label learning. São Carlos, SP, Brasil., 2013.

SULEIMAN, D.; AWAJAN, A. Deep learning based abstractive text summarization: approaches, datasets, evaluation measures, and challenges. *Mathematical problems in engineering*, Hindawi, v. 2020, 2020.

TANG, Y.; TANG, C.; ZHU, C. Resolve out of vocabulary with long short-term memory networks for morphology. In: IEEE. *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. [S.l.], 2020. p. 115–119.

THANAKI, J. *Python natural language processing*. [S.l.]: Packt Publishing Ltd, 2017.

VASILIEV, Y. *Natural Language Processing with Python and SpaCy: A Practical Introduction*. [S.l.]: No Starch Press, 2020.

WANG, C.; NULTY, P.; LILLIS, D. A comparative study on word embeddings in deep learning for text classification. In: *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*. [S.l.: s.n.], 2020. p. 37–46.

WON, M.-S. et al. An embedding method for unseen words considering contextual information and morphological information. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. [S.l.: s.n.], 2021. p. 1055–1062.

WON, M.-S.; LEE, J.-H. Embedding for out of vocabulary words considering contextual and morphosyntactic information. In: IEEE. *2018 International Conference on Fuzzy Theory and Its Applications (iFUZZY)*. [S.l.], 2018. p. 212–215.

YEPES, A. J. Word embeddings and recurrent neural networks based on long-short term memory nodes in supervised biomedical word sense disambiguation. *Journal of biomedical informatics*, Elsevier, v. 73, p. 137–147, 2017.

ZULQARNAIN, M. et al. Efficient processing of gru based on word embedding for text classification. *JOIV: International Journal on Informatics Visualization*, v. 3, n. 4, p. 377–383, 2019.

APÊNDICE A Semelhança de cosseno entre pares de palavras

Semelhanças de cosseno entre pares de palavras

Sentença com OOV	Sentença com Significado Semelhante	Base 1					Base 2					Base 3				
		RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU	RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU	RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
	Eu queria mesmo era dar um soco na tua cara!	-0,1742	0,1731	-0,1249	-0,1053	0,079	-0,1053	0,1281	0,1281	0,1281	0,1625	0,0314	-0,0296	0,3742	0,0238	-0,1758	0,2234	**0,4876**	0,0124
	Todo dia um tapa na cara diferente pra eu aprender que ninguém tem a mesma consideração por mim.	0,0213	**0,3874**	0,0923	0,0723	0,3113	0,0723	0,0600	0,0600	0,0600	**0,3282**	0,1872	0,1701	0,5380	0,2213	-0,1756	0,3964	**0,5777**	0,1155
*Morilda levou uns **bofetes** do marido em plena Rua Grande.*	deu também uma coronhada, além de um murro na moça	0,1175	0,3990	0,1659	0,3964	0,3109	0,3964	0,4528	0,4528	0,4528	**0,4221**	0,2687	0,2904	**0,4227**	0,3307	-0,1167	0,3940	0,2324	0,0367
	Você fica muito parecido com as pessoas que convive	0,3569	0,1647	0,2986	0,2245	**0,5381**	0,2245	0,2670	0,4679	0,4196	0,3206	0,2184	**0,6282**	0,4051	0,3022	0,3020	0,4204	**0,5316**	0,4894
*Alessandra é **escritinho** um ator da Globo.*	Para muitas pessoas a felicidade é semelhante a uma bola.	0,3740	0,3687	0,5011	0,4793	**0,5306**	0,4793	0,5510	**0,5729**	0,4436	0,5405	0,2031	0,4624	0,4399	0,3817	0,3744	0,3190	**0,4501**	0,4309
	Ninguém é igual a ninguém.	0,3385	0,3127	**0,4300**	0,3810	0,4029	0,3810	0,4763	**0,4955**	0,4763	0,4891	0,1409	0,3815	**0,4412**	0,4056	0,3815	0,2998	0,3911	0,4103
	Fofoquei com a vizinha sobre o problema do meu irmão.	0,4458	**0,4829**	0,2269	0,3935	0,3594	0,3935	**0,5748**	0,4063	0,3566	0,3553	0,0385	0,1961	**0,4056**	0,2237	0,3579	0,0423	0,1871	0,2800
*Patrício é a mulher mais **maroca** da rua. Ela sabe da vida de todo mundo.*	Hoje vi Maria e Joana fofocando na esquina. Ela não passa de uma fofoqueira que só cuida da vida dos outros e não sabe guardar segredos!	**0,3220**	0,3071	0,0726	0,2141	0,2011	0,2141	**0,5336**	0,2224	0,2655	0,1496	0,0822	0,1289	0,3450	0,2824	**0,3992**	0,1655	0,1879	0,2964
		0,4788	0,5953	0,2306	0,5067	0,3537	0,5067	**0,7123**	0,4129	0,6381	0,4253	0,1661	0,2434	0,5785	0,2864	0,5474	0,1306	0,4232	0,4575
	A fofoqueira quer saber as novidades.	0,2878	0,3157	0,1798	**0,3472**	0,1305	**0,3472**	**0,4949**	0,1817	0,3957	0,2871	0,0499	0,1393	0,2662	0,1741	0,2803	0,0341	**0,3241**	0,2472
	Fofoquei com a vizinha sobre o problema do meu irmão.	0,1773	0,3698	**0,4550**	0,3428	0,2074	0,3428	0,0833	0,1991	0,2630	**0,4869**	0,0553	0,2550	0,1915	**0,4337**	0,1934	0,1689	0,3093	0,3101
*Em alguns bairros, nem se pode falar com os amigos, que logo aparece alguém para **maracar**.*	Hoje vi Maria e Joana fofocando na esquina. Ela não passa de uma fofoqueira que só cuida da vida dos outros e não sabe guardar segredos!	**0,2416**	0,1127	0,0382	0,1005	-0,0182	0,1005	-0,0158	0,1840	0,1893	**0,3241**	0,0442	0,0686	**0,1427**	0,0579	0,1062	0,1264	0,0705	0,0923
		0,3633	0,5811	0,4512	0,4902	0,3941	0,4902	0,1199	0,5792	0,4576	**0,7691**	0,3253	0,2727	0,4106	**0,4689**	0,3249	0,3344	0,3805	0,4158
	A fofoqueira quer saber as novidades.	0,0644	0,2372	0,0842	**0,2756**	0,1782	**0,2756**	-0,0542	0,2023	0,3366	**0,4448**	-0,0413	0,1695	**0,1468**	0,1054	0,1311	0,0371	0,1681	0,1241
*Tua rua é cheia de **maroca**.*	Fofoquei com a vizinha sobre o problema do meu irmão.	0,3441	**0,3768**	0,1497	0,2858	0,3154	0,2858	**0,3007**	0,2706	0,2405	0,2974	-0,0310	0,1295	0,2952	0,0662	0,2207	-0,0067	0,1032	0,2253
	Hoje vi Maria e Joana fofocando na esquina.	**0,2380**	0,2116	0,0083	0,1150	0,1645	0,1150	**0,4195**	0,1045	0,1720	0,0969	0,0282	0,0755	0,2600	0,1655	**0,3042**	0,1275	0,1232	0,2558
	Ela não passa de uma fofoqueira que só cuida da vida dos outros e não sabe guardar segredos!	0,3441	**0,4641**	0,1337	0,3768	0,2972	0,3768	0,3770	0,2355	**0,5199**	0,3531	0,0821	0,1613	**0,4487**	0,0926	0,3900	0,0704	0,3266	0,3936
	A fofoqueira quer saber as novidades.	0,1773	0,1961	0,1071	**0,2445**	0,0848	**0,2445**	0,1912	0,0341	**0,2961**	0,2310	-0,0143	0,0760	0,1545	0,0256	0,1470	-0,0114	**0,2517**	0,1960
*E a senhora conhece outro além do **mucuru** que também solte um cheiro ruim?*	O gambá é uma verdadeira praga em certas regiões, pois atacam os animais domésticos.	0,2924	0,4657	0,0783	**0,5483**	0,1337	**0,5483**	0,3812	0,2711	0,3288	**0,3876**	0,2713	0,3702	0,2643	0,0632	0,4453	0,2927	**0,5047**	0,2900
	O gambá só libera o líquido com cheiro ruim quando sente ameaçado.	0,2041	0,2541	0,2146	0,2592	0,0566	0,2592	0,3120	0,2374	0,2778	**0,3251**	0,2070	0,2736	0,3264	0,1452	0,2230	0,3070	**0,5899**	0,3589
	A mulher faz uma careta, ficando ainda mais feia	0,3166	0,3505	0,0506	**0,4074**	0,1180	**0,4074**	**0,4047**	0,2478	0,3287	0,3283	0,2010	0,2746	0,2778	0,1493	0,3008	**0,3749**	0,3691	0,2096

	todo aquele que coloca suas verdades como absolutas torna-se feio	0,3862	0,4171	0,0457	**0,5800**	0,2516	**0,5800**	0,4777	0,3887	0,4295	0,4551	0,4576	**0,5130**	0,3810	0,0777	0,4137	0,3989	**0,6761**	0,4397
	Esta bolsa é horrorosa	0,2208	**0,2917**	-0,0410	0,2260	0,0693	0,2260	**0,2113**	0,1877	0,1954	0,1848	0,1282	0,0944	0,1254	0,0698	0,1191	0,1797	**0,2272**	0,1361
	Você é muito feia.	0,2196	**0,2298**	0,0334	0,1659	0,0183	0,1659	0,1990	0,1838	0,1404	**0,2733**	0,1287	0,0483	0,2286	0,1016	0,1340	0,2500	**0,3666**	0,2428
Me bate e me fura, mas não me confunda com essa ***mucura***	O gambá é uma verdadeira praga em certas regiões, pois atacam os animais domésticos.	0,2416	0,4264	0,0391	**0,5027**	0,0970	**0,5027**	0,2590	0,2335	0,2809	**0,3607**	0,2501	0,3450	0,2208	0,0189	0,3906	0,2611	**0,4661**	0,2400
	O gambá so libera o liquido com cheiro ruim quando sente ameaçado.	0,1824	0,2392	0,1940	**0,2427**	0,0416	**0,2427**	0,2562	0,2210	0,2574	**0,3150**	0,1977	0,2651	0,3048	0,1261	0,2011	0,2923	**0,5763**	0,3367
	A mulher faz uma careta, ficando ainda mais feia	0,2824	0,3261	0,0253	**0,3812**	0,0942	**0,3812**	**0,3198**	0,2229	0,2971	0,3113	0,1872	0,2589	0,2478	0,1199	0,2658	**0,3527**	0,3445	0,1773
	todo aquele que coloca suas verdades como absolutas torna-se feio	0,3689	0,4119	0,0345	**0,5922**	0,2409	**0,5922**	0,4303	0,3757	0,4144	0,4496	0,4494	**0,5122**	0,3638	0,0653	0,4019	0,3873	**0,6687**	0,4235
	Esta bolsa é horrorosa	0,1825	**0,2611**	-0,0681	0,1811	0,0417	0,1811	0,1222	0,1595	0,1596	**0,1639**	0,1128	0,0727	0,0940	0,0362	0,0758	0,1563	**0,1974**	0,0991
	Você é muito feia.	0,2029	**0,2197**	0,0215	0,1525	0,0074	0,1525	0,1593	0,1717	0,1259	**0,2661**	0,1221	0,0399	0,2129	0,0877	0,1178	0,2390	**0,3563**	0,2268
Não acredito que o bonitão do Luís esteja saindo com a ***mucura*** *da Catarina.*	O gambá é uma verdadeira praga em certas regiões, pois atacam os animais domésticos.	0,2946	0,4682	0,0776	**0,5326**	0,1344	**0,5326**	0,3746	0,2738	0,3302	**0,3893**	0,2726	0,3675	0,2712	0,0625	0,4393	0,2979	**0,5197**	0,2998
	O gambá so libera o liquido com cheiro ruim quando sente ameaçado.	0,2028	**0,2525**	0,2123	0,2473	0,0545	0,2473	0,3005	0,2379	0,2765	**0,3250**	0,2069	0,2699	0,3332	0,1430	0,2163	0,3110	**0,6055**	0,3688
	A mulher faz uma careta, ficando ainda mais feia	0,2933	0,3270	0,0294	**0,3579**	0,0989	**0,3579**	**0,3390**	0,2311	0,3055	0,3143	0,1915	0,2566	0,2635	0,1258	0,2656	**0,3660**	0,3572	0,1899
	todo aquele que coloca suas verdades como absolutas torna-se feio	0,3655	0,3959	0,0258	**0,5286**	0,2354	**0,5286**	0,4152	0,3752	0,4088	0,4429	0,4506	**0,4948**	0,3714	0,0552	0,3794	0,3915	**0,6755**	0,4305
	Esta bolsa é horrorosa	0,2021	**0,2734**	-0,0580	0,1894	0,0540	0,1894	0,1607	0,1744	**0,1766**	0,1732	0,1205	0,0808	0,1114	0,0508	0,0926	0,1703	**0,2159**	0,1195
	Você é muito feia.	0,1920	**0,2023**	0,0096	0,1179	-0,0043	0,1179	0,1282	0,1640	0,1128	**0,2570**	0,1174	0,0294	0,2103	0,0748	0,0967	0,2367	**0,3519**	0,2211

APÊNDICE B Marcação Gramatical

Base 1

Palavra(s) OOV	Semelhante	Marcação Gramatical do Semelhante	Marcação Gramatical por Modelo					
			RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
bofete / bofetes	Soco. Tapa. Murro.	substantivo	numeral	substantivo	substantivo	substantivo	substantivo	substantivo
escritinho	Muito parecido. Semelhante. Igual.	adjetivo	verbo	substantivo	substantivo	substantivo	advérbio	substantivo
maroca	Pessoa fofoqueira.	adjetivo	verbo	adjetivo / substantivo	verbo	substantivo	substantivo	substantivo
marocar	Observar a vida alheia. Fofocar.	verbo	advérbio	substantivo	pronome	substantivo	verbo	substantivo
mucura	Animal. Gambá.	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo
mucura	Horrorosa. Muito feia. Feio. Feia.	adjetivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo

Base 2

Palavra(s) OOV	Semelhante	Marcação Gramatical do Semelhante	Marcação Gramatical por Modelo					
			RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
bofete / bofetes	Soco. Tapa. Murro.	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo
escritinho	Muito parecido. Semelhante. Igual.	adjetivo	adjetivo	advérbio	advérbio	adjetivo	adjetivo	advérbio
maroca	Pessoa fofoqueira.	adjetivo	adjetivo / advérbio	adjetivo / substantivo	pronome	substantivo	numeral	substantivo
marocar	Observar a vida alheia. Fofocar.	verbo	numeral	pronome	substantivo	pronome	pronome	verbo
mucura	Animal. Gambá.	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo
mucura	Horrorosa. Muito feia. Feio. Feia.	adjetivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo

Base 3

Palavra(s) OOV	Semelhante	Marcação Gramatical do Semelhante	Marcação Gramatical por Modelo					
			RNN	LSTM	GRU	BIRNN	BILSTM	BIGRU
bofete / bofetes	Soco. Tapa. Murro.	substantivo	advérbio	advérbio	numeral	substantivo	substantivo	substantivo
escritinho	Muito parecido. Semelhante. Igual.	adjetivo	advérbio	advérbio	advérbio	advérbio	adjetivo	advérbio
maroca	Pessoa fofoqueira.	adjetivo	adjetivo / substantivo	substantivo	adjetivo/ substantivo	substantivo	substantivo	substantivo
marocar	Observar a vida alheia. Fofocar.	verbo	substantivo	verbo	verbo	verbo	verbo	verbo
mucura	Animal. Gambá.	substantivo	substantivo	substantivo	substantivo	substantivo	adesão*	substantivo
mucura	Horrorosa. Muito feia. Feio. Feia.	adjetivo	substantivo	substantivo	substantivo	substantivo	substantivo	substantivo

ANEXO A Código GRU Utilizado na Preparação do Modelo

```
# -*- coding: utf-8 -*-
"""GRU.ipynb

Original file is located at
    https://github.com/paulamyrian/OOV_usingRNNs
"""

import pandas as pd
import numpy
from numpy import array
import spacy
from spacy.vocab import Vocab
import keras as k
from keras import backend as K
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, load_model
from keras.layers import GRU, Dense, Dropout, Masking, Embedding
import pickle
import nltk
#nltk.download('stopwords')

#reading processed data
data = open('cleandata.csv').read()[:100000]
data

#function for preparing text data into sequences for training
def data_sequencing(data):
    # integer encode sequences of words
    tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts([data])
with open('tokenizer.pkl', 'wb') as f:
  # Save the tokeniser by pickling it
    pickle.dump(tokenizer, f)

encoded = tokenizer.texts_to_sequences([data])[0]
# retrieve vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

# create line-based sequences
sequences = list()
rev_sequences = list()
for line in data.split('.'):
    encoded = tokenizer.texts_to_sequences([line])[0]
    rev_encoded = encoded[::-1]
    for i in range(1, len(encoded)):
        sequence = encoded[:i+1]
        rev_sequence = rev_encoded[:i+1]
        sequences.append(sequence)
        rev_sequences.append(rev_sequence)
print('Total Sequences: %d' % len(sequences))


#find max sequence length
max_length = max([len(seq) for seq in sequences])
with open('max_length.pkl', 'wb') as f:
  # Save max_length by pickling it
    pickle.dump(max_length, f)
print('Max Sequence Length: %d' % max_length)

# pad sequences and create the forward sequence
sequences = pad_sequences(sequences, maxlen=max_length,
                          padding='pre')
# split into input and output elements
sequences = array(sequences)
X, y = sequences[:,:-1],sequences[:,-1]
```

```
    return X,y,max_length,vocab_size

#returning forward and reverse sequences
#along with max sequence
#length from the data

X,y,max_length,vocab_size = data_sequencing(data)

# define forward sequence model
model_2 = Sequential()
model_2.add(Embedding(vocab_size,100,
                      input_length=max_length-1))
model_2.add(GRU(100))
model_2.add(Dense(vocab_size, activation='softmax'))
print(model_2.summary())

# compile forward sequence network
model_2.compile(loss='sparse_categorical_crossentropy',
                optimizer='adam', metrics=['accuracy'])
# fit network
model_2.fit(X, y,batch_size=100, epochs=200, verbose=2)
# save the model to file
model_2.save('model_2.h5')
```

ANEXO B Código GRU Utilizado na Predição de Embeddings

```
# -*- coding: utf-8 -*-
"""modeloUNI (GRU).ipynb


Original file is located at
    https://github.com/paulamyrian/OOV_usingRNNs
"""

#importing libraries
import spacy
from spacy.vocab import Vocab
import numpy
from numpy import array
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM, Bidirectional
import pickle
import nltk

# generate a sequence using a language model
def generate_seq(model, tokenizer, max_length, seed_text):
    if seed_text == "":
        return ""
    else:
        in_text = seed_text
        n_words = 1
        n_preds = 5 #number of words to predict for the seed text
        pred_words = ""
```

```
        # generate a fixed number of words
        for _ in range(n_words):
            # encode the text as integer
            encoded = tokenizer.texts_to_sequences([in_text])[0]
            # pre-pad sequences to a fixed length
            encoded = pad_sequences([encoded], maxlen=max_length,
                                    padding='pre')
            # predict probabilities for each word
            proba = model.predict(encoded, verbose=0).flatten()
            #take the n_preds highest probability classes
            yhat = numpy.argsort(-proba)[:n_preds]
            # map predicted words index to word
            out_word = ''

            for _ in range(n_preds):
                for word, index in tokenizer.word_index.items():
                    if index == yhat[_] and word not in stopwords:
                        out_word = word
                        pred_words += '␣' + out_word
                        #print(out_word)
                        break


        return pred_words

# load the model
model = load_model('model_2.h5')

#load tokeniser and max_length
with open('tokenizer.pkl', 'rb') as f:
    tokenizer = pickle.load(f)

with open('max_length.pkl', 'rb') as f:
    max_length = pickle.load(f)

#loading stopwords to improve relevant word predictions
#stopwords= open('stopwords ').read().split()
```

```
stopwords = nltk.corpus.stopwords.words('portuguese')

#load spacy Model
nlp = spacy.load('pt_core_news_md')

#Find and set embeddings for OOV words
def set_embedding_for_oov(doc):
    #checking for oov words and adding embedding
    for token in doc:
        if token.is_oov == True:
            before_text = doc[:token.i].text

            pred_before = generate_seq(model, tokenizer, max_length-1,
                                       before_text).split()

            embedding = numpy.zeros((300,))

            i=len(before_text)
            print('Words predicted from forward sequence model:')
            for word in pred_before:
                print(word)
                embedding += i*nlp.vocab.get_vector(word)
                i= i*.5

            nlp.vocab.set_vector(token.text, embedding)
            print(token.text,nlp.vocab.get_vector(token.text))

#function to find most similar words
def most_similar(word):
    by_similarity = sorted(word.vocab, key=lambda w: word.similarity(w),
                           reverse=True)
    return [w.orth_ for w in by_similarity[:10]]

"""# Testes com as palavras

## Frase 1
"""
```

```
doc1 = nlp('Marilda␣levou␣uns␣bofetes␣do␣marido␣em␣plena␣Rua␣Grande.')
set_embedding_for_oov(doc1)

nlp.vocab.get_vector('bofetes')

#BOFETE      Soco. Tapa. Murro.

doc2 = nlp('Eu␣queria␣mesmo␣era␣dar␣um␣soco␣na␣tua␣cara!')

doc3 = nlp('Todo␣dia␣um␣tapa␣na␣cara␣diferente␣pra␣eu␣aprender␣que'
           'ningu m␣tem␣a␣mesma␣considera  o␣por␣mim.')

doc4 = nlp('deu␣tamb m␣uma␣coronhada,␣al m␣de␣um␣murro␣na␣mo a')

print(doc1, "<->", doc2, doc1.similarity(doc2))
print(doc1, "<->", doc3, doc1.similarity(doc3))
print(doc1, "<->", doc4, doc1.similarity(doc4))

nlp('bofetes').similarity(nlp('soco'))

nlp('bofetes').similarity(nlp('tapa'))

nlp('bofetes').similarity(nlp('murro'))

nlp('bofete').similarity(nlp('soco'))

most_similar(nlp(doc1))

for token in doc1:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

print(doc1[3].morph)
print(doc1[3].pos_)
```

```
for ent in doc1.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

"""## Frase 2"""

doc1 = nlp('Alessandro␣ ␣escritinho␣um␣ator␣da␣Globo.')
set_embedding_for_oov(doc1)

nlp.vocab.get_vector('escritinho')

#ESCRITINHO      Muito parecido. Semelhante. Igual.

doc2 = nlp('Voc ␣fica␣muito␣parecido␣com␣as␣pessoas␣que␣convive')

doc3 = nlp('Para␣muitas␣pessoas␣a␣felicidade␣ ␣semelhante␣a␣uma␣bola.')

doc4 = nlp('Ningu m␣ ␣igual␣a␣ningu m.')

print(doc1, "<->", doc2, doc1.similarity(doc2))
print(doc1, "<->", doc3, doc1.similarity(doc3))
print(doc1, "<->", doc4, doc1.similarity(doc4))

nlp('escritinho').similarity(nlp('muito␣parecido'))

nlp('escritinho').similarity(nlp('parecido'))

nlp('escritinho').similarity(nlp('semelhante'))

nlp('escritinho').similarity(nlp('igual'))

most_similar(nlp(doc1))

for token in doc1:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)
```

```
print(doc1[2].morph)
print(doc1[2].pos_)

for ent in doc1.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

"""## Frase 3"""

doc1 = nlp('Patr cia␣ ␣a␣mulher␣mais␣maroca␣da␣rua.'
                  'Ela␣sabe␣da␣vida␣de␣todo␣mundo.')
set_embedding_for_oov(doc1)

doc2 = nlp('Em␣alguns␣bairros,␣nem␣se␣pode␣falar␣com␣os␣amigos,'
           'que␣logo␣aparece␣algu m␣para␣marocar.')
set_embedding_for_oov(doc2)

doc3 = nlp('Tua␣rua␣ ␣cheia␣de␣maroca.')
set_embedding_for_oov(doc3)

nlp.vocab.get_vector('maroca')

nlp.vocab.get_vector('marocar')

#MAROCA      Pessoa fofoqueira, que gosta de falar da vida alheia,
#ou de saber o que est  acontecendo com os outros.
#MAROCAR      Observar a vida alheia para tecer coment rios com outras pe

doc4 = nlp('Fofoquei␣com␣a␣vizinha␣sobre␣o␣problema␣do␣meu␣irm o.')

doc5 = nlp('Hoje␣vi␣Maria␣e␣Joana␣fofocando␣na␣esquina.')

doc6 = nlp('Ela␣n o␣passa␣de␣uma␣fofoqueira␣que␣s ␣cuida␣da␣vida␣dos'
           '␣outros␣e␣n o␣sabe␣guardar␣segredos!')

doc7 = nlp('A␣fofoqueira␣quer␣saber␣as␣novidades.')

print(doc1, "<->", doc4, doc1.similarity(doc4))
```

```
print(doc1, "<->", doc5, doc1.similarity(doc5))
print(doc1, "<->", doc6, doc1.similarity(doc6))
print(doc1, "<->", doc7, doc1.similarity(doc7))
print(doc2, "<->", doc4, doc2.similarity(doc4))
print(doc2, "<->", doc5, doc2.similarity(doc5))
print(doc2, "<->", doc6, doc2.similarity(doc6))
print(doc2, "<->", doc7, doc2.similarity(doc7))
print(doc3, "<->", doc4, doc3.similarity(doc4))
print(doc3, "<->", doc5, doc3.similarity(doc5))
print(doc3, "<->", doc6, doc3.similarity(doc6))
print(doc3, "<->", doc7, doc3.similarity(doc7))

nlp('maroca').similarity(nlp('fofoqueira'))

nlp('marocar').similarity(nlp('fofocar'))

nlp('marocar').similarity(nlp('conversar'))

nlp('marocar').similarity(nlp('falar'))

nlp('maroca').similarity(nlp('observador'))

most_similar(nlp(doc1))

most_similar(nlp(doc2))

most_similar(nlp(doc3))

for token in doc1:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

for token in doc2:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)
```

```
for token in doc3:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

print(doc1[5].morph)
print(doc1[5].pos_)

print(doc2[17].morph)
print(doc2[17].pos_)

print(doc3[5].morph)
print(doc3[5].pos_)

for ent in doc1.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

for ent in doc2.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

for ent in doc3.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

"""## Frase 4"""

nlp.vocab.get_vector('mucura')

doc1 = nlp('E␣a␣senhora␣conhece␣outro␣al m␣da␣mucura␣que'
           'tamb m␣solte␣um␣cheiro␣ruim?')
set_embedding_for_oov(doc1)

doc2 = nlp('Me␣bate␣e␣me␣fura,␣mas␣n o␣me␣confunda␣com␣essa␣mucura')
set_embedding_for_oov(doc2)

doc3 = nlp('N o␣acredito␣que␣o␣bonit o␣do␣Lu s␣esteja'
           'saindo␣com␣a␣mucura␣da␣Catarina.')
```

```
set_embedding_for_oov(doc3)

nlp.vocab.get_vector('mucura')

#MUCURA      Al m de referir-se a um animal, essa palavra tamb m serve c

doc4 = nlp('O gamb     uma verdadeira praga em certas regi es,'
           'pois atacam os animais dom sticos.')

doc5 = nlp('O gamb  so libera o liquido com cheiro ruim quando'
           'sente amea ado.')

doc6 = nlp('A mulher faz uma careta, ficando ainda mais feia')

doc7 = nlp('todo aquele que coloca suas verdades como absolutas'
           'torna-se feio')

doc8 = nlp('Esta bolsa     horrorosa')

doc9 = nlp('Voc     muito feia.')

print(doc1, "<->", doc4, doc1.similarity(doc4))
print(doc1, "<->", doc5, doc1.similarity(doc5))
print(doc1, "<->", doc6, doc1.similarity(doc6))
print(doc1, "<->", doc7, doc1.similarity(doc7))
print(doc1, "<->", doc8, doc1.similarity(doc8))
print(doc1, "<->", doc9, doc1.similarity(doc9))
print(doc2, "<->", doc4, doc2.similarity(doc4))
print(doc2, "<->", doc5, doc2.similarity(doc5))
print(doc2, "<->", doc6, doc2.similarity(doc6))
print(doc2, "<->", doc7, doc2.similarity(doc7))
print(doc2, "<->", doc8, doc2.similarity(doc8))
print(doc2, "<->", doc9, doc2.similarity(doc9))
print(doc3, "<->", doc4, doc3.similarity(doc4))
print(doc3, "<->", doc5, doc3.similarity(doc5))
print(doc3, "<->", doc6, doc3.similarity(doc6))
print(doc3, "<->", doc7, doc3.similarity(doc7))
```

```
print(doc3, "<->", doc8, doc3.similarity(doc8))
print(doc3, "<->", doc9, doc3.similarity(doc9))

nlp('mucura').similarity(nlp('horrorosa'))

nlp('mucura').similarity(nlp('muito␣feia'))

nlp('mucura').similarity(nlp('feio'))

nlp('mucura').similarity(nlp('feia'))

nlp('mucura').similarity(nlp('gamb '))

nlp('mucura').similarity(nlp('bicho'))

most_similar(nlp(doc1))

most_similar(nlp(doc2))

most_similar(nlp(doc3))

for token in doc1:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

for token in doc2:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

for token in doc3:
    print(token.text, token.lemma_, token.pos_,
          token.tag_, token.dep_,
            token.shape_)

print(doc1[7].morph)
```

```
print(doc1[7].pos_)

print(doc2[12].morph)
print(doc2[12].pos_)

print(doc3[11].morph)
print(doc3[11].pos_)

for ent in doc1.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

for ent in doc2.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

for ent in doc3.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```