UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALISSON MARIANO DE SALES

# COMPARAÇÃO EXTRÍNSECA DE ALGORITMOS DE *WORD EMBEDDING* NA SIMPLIFICAÇÃO LÉXICA DE TEXTO

TRABALHO DE CONCLUSÃO DE CURSO

**MEDIANEIRA**

**2017**

ALISSON MARIANO DE SALES

# COMPARAÇÃO EXTRÍNSECA DE ALGORITMOS DE *WORD EMBEDDING* NA SIMPLIFICAÇÃO LÉXICA DE TEXTO

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Bacharel em Computação".

Orientador: Prof. Dr. Arnaldo Candido Junior

**MEDIANEIRA**

**2017**

Ministério da Educação
**Universidade Tecnológica Federal do Paraná**
Diretoria de Graduação e Educação Profissional
Coordenação do Curso de Ciência da Computação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

---

## TERMO DE APROVAÇÃO

## COMPARAÇÃO EXTRÍNSECA DE ALGORITMOS DE *WORD EMBEDDING* NA SIMPLIFICAÇÃO LÉXICA DE TEXTO

Por

**ALISSON MARIANO DE SALES**

Este Trabalho de Conclusão de Curso foi apresentado às 09:00h do dia 23 de Novembro de 2017 como requisito parcial para a obtenção do título de Bacharel no Curso de Ciência da Computação, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

---

Prof. Dr. Alan Gavioli
UTFPR - Câmpus Medianeira

---

Prof. Dr. Arnaldo Candido Junior
UTFPR - Câmpus Medianeira

---

Prof. Me. Jorge Aikes Junior
UTFPR - Câmpus Medianeira

A folha de aprovação assinada encontra-se na Coordenação do Curso.

# RESUMO

SALES, Alisson Mariano. COMPARAÇÃO EXTRÍNSECA DE ALGORITMOS DE *WORD EMBEDDING* NA SIMPLIFICAÇÃO LÉXICA DE TEXTO. 67 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2017.

O advento das áreas de Inteligência Artificial tem proporcionado o avanço e a criação de soluções aplicadas às mais diversas áreas. Com o Processamento de Linguagem Natural isso não está sendo diferente, nos últimos cinco anos as pesquisas sobre os algoritmos de representação vetorial e captura semântica das palavras obtiveram grandes resultados. Chamados também de *word embeddings*, esses algoritmos agregam benefícios que métodos anteriores não disponibilizavam. Visando a necessidade de maior estudo sobre esses novos algoritmos, como Skip-Gram, Glove e CBOW e, ao mesmo tempo, observando a importância da automatização de simplificação léxica em benefício de pessoas em aprendizagem do português, disléxicos, portadores de afasia, entre outros, desenvolveu-se neste trabalho um simplificador léxico utilizando-se dessas representações. Esse simplificador utilizou-se também de Rede Neural Artificial e alguns dicionários para criar simplificações. Nos experimentos realizados, gerou-se três contribuições, sendo elas: um simplificador capaz de auxiliar um falante proficiente no processo de simplificação léxica, uma estrutura de rede neural com tendência ao aprendizado automatizado e a comparação extrínseca dos algoritmos. Como melhor algoritmo, nas observações realizadas, o Wang2vec CBOW obteve os melhores resultados para a atividade de simpificação léxica.

**Palavras-chave:** computação semântica, processamento de linguagem natural (computação), inteligência artificial.

# ABSTRACT

SALES, Alisson Mariano. EXTRINSIC COMPARISON OF WORD EMBEDDING ALGORITHMS IN TEXT LEXICAL SIMPLIFICATION. 67 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2017.

The advent of Artificial Intelligence has provided the advance and the creation of solutions applied to the most diverse areas. Within Natural Language Processing this has not been different, in the last five years, the studies of algorithms for vector representation and semantic retrieval of words have shown great advances. Also called *word embeddings*, these algorithms add benefits that earlier methods did not provide. Aiming at the need to further study these new algorithms, such as Skip-Gram, Glove and CBOW, and at the same time, noting the importance of the automation of lexical simplification for the benefit of Portuguese learners, dyslexics, aphasia, among others, this work proposes the development of a lexical simplifier using these representations. This simplifier also used a Artificial Neural Network and some dictionaries to create simplifications. There were three main contributions observed in the experiments carried out: a simplifier capable of assisting a proficient speaker in the lexical simplification process, an artificial neural network structure with a tendency to automated learning and the extrinsic comparison of the algorithms. The algorithm Wang2vec Continuous Bag-of-Words performed the best results for the lexical simplification activity during this work's experiments.

**Keywords:** semantic computing, natural language processing (computer science), artificial intelligence.

Aos meus pais, Jupira Maria Mariano Sales e Hermínio de Sales, que com o incondicional apoio, proporcionaram, sob as adversidades da vida, minha educação.

## AGRADECIMENTOS

Agradeço aos mestres que ao longo de todas as disciplinas da graduação compartilharam do seu conhecimento, me proporcionando enfim, desenvolver este trabalho final.

Agradeço ao Nathan Hartmann pela disponibilização antecipada das bases de *embeddings*.

Ao meu orientador, Professor Arnaldo Candido Junior, agradeço sua demasiada atenção e entusiasmo em me guiar adentro dos tortuosos caminhos do conhecimento. Sem seu comprometimento em ensinar, este trabalho não seria possível.

Agradeço também aos meus amigos da UTFPR que compartilharam de dúvidas e soluções durante o trajeto de aprendizado e desenvolvimento deste projeto final.

Aos meus amigos vivendo em outras cidades que sinergeticamente me apoiaram: muito obrigado!

Também agradeço aos meus irmãos e diversos familiares que torceram e, através de palavras e atitudes de carinho e atenção, me encorajaram.

Enfim, agradeço aos meus pais pelos constantes esforços em prol dos meus objetivos.

You shall know a word by the company it keeps! (Firth, 1957)

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| CBOW | Continuous Bag-of-Words |
| GloVe | Global Vectors |
| IA | Inteligência Artificial |
| MLP | Multilayer Perceptron |
| NILC | Núcleo Interinstitucional de Linguística Computacional |
| NLTK | Natural Language Toolkit |
| PLN | Processamento de Linguagem Natural |
| ReLU | Rectified Linear Unit |
| RNA | Rede Neural Artificial |
| S1 | Simplificador 1 |
| S2 | Simplificador 2 |

**SUMÁRIO**

# 1 INTRODUÇÃO

A Inteligência Artificial (IA) tem apresentado grandes avanços na última década. Tópicos que antes eram considerados ficção científica, atualmente se tornaram possíveis. Um exemplo é o projeto YOLO (REDMON et al., 2016), que a partir de uma sequência de imagens consegue identificar objetos em tempo real. A IA é uma área difícil de definir, visto que para defini-la é necessário antes definir o que é inteligência (RUSSELL; NORVIG, 2013). Diferentes autores utilizaram definições distintas para definir IA (SCHALKOFF, 1990; RICH; KNIGHT, 1990; WINSTON, 1992; HAUGELAND, 1985). Em linhas gerais, a IA busca reproduzir ou se inspirar nas capacidades inteligentes do ser humano para realizar simulação em sistemas computacionais.

Processamento de Linguagem Natural (PLN) é uma subárea da IA e busca, também, gerar informações a partir de uma base escrita em linguagem natural, como a língua portuguesa. A língua natural apresenta características que dificultam a sua interpretação pelo computador. Ambiguidade e sua constante mutação são exemplos dessas características (RUSSELL; NORVIG, 2013).

Existe, dentro dos estudos de PLN, a ideia que palavras são semelhantes quando aparecem em contextos semelhantes, ou seja, é possível identificar uma palavra a partir do contexto ao qual ela se insere (JURAFSKY; MARTIN, 2016). Vetores semânticos capturam esse conceito, e buscam reduzir uma palavra a um vetor n-dimensional. Uma forma de converter uma palavra em um vetor é a técnica de *word embedding* (incorporação de palavras, em tradução livre).

Outra subárea de estudo importante da IA são as Redes Neurais Artificiais (RNA). Essas redes se inspiram no funcionamento dos neurônios humanos para criar uma forma de aprendizado que se distingue da forma que os programas computacionais convencionais computam. Os neurônios artificiais são unidades simples de processamento que, dentro de uma rede, são capazes de aprender a partir das experiências de treinamento (HAYKIN, 2009).

Este trabalho propõe o uso de *word embeddings*, dicionários linguísticos e RNA para a construção de um software de simplificação lexical, que sugere a troca de palavras difíceis por palavras com maior frequência dentro da língua portuguesa. Averigua-se em seguida se

as substituições sugeridas preservam as características gramaticais e semânticas, e ao mesmo tempo torna o texto resultante mais simples de ser lido. Ao final é realizada uma comparação extrínseca dos algoritmos de *word embedding* Skip-Gram e Continuous Bag-of-Words (CBOW) do pacote word2Vec, Global Vectors (GloVe), e também Skip-Gram e CBOW do pacote Wang2vec e do pacote fastText, na tarefa de simplificação léxica.

## 1.1 OBJETIVOS GERAL E ESPECÍFICOS

O objetivo geral deste trabalho foi desenvolver uma comparação de algoritmos de *word embedding* na tarefa de simplificação léxica assistida por uma RNA e uma base de dados psicolínguisticos. Os objetivos específicos foram:

- Utilizar base de *word embeddings*, com 50 dimensões, gerada pelo Núcleo Interinstitucional de Linguística Computacional (NILC, Universidade de São Paulo câmpus São Carlos) para Skip-Gram, CBOW e GloVe para a seleção de palavras candidatas a ser versão simplificada;
- Desenvolver *scripts* para entrada dos textos sujeitos à simplificação;
- Integrar recursos e bibliotecas de processamento de língua natural para o uso dos textos;
- Elencar as palavras candidatas mais suscetíveis a serem utilizadas na simplificação através de dicionário de frequência e lematizador de palavras;
- Anotar uma base de dados com 500 amostras cada, de palavras complexas e simples, sendo 250 positivas e 250 negativas, para o Skip-Gram, CBOW e GloVe;
- Desenvolver e treinar diferentes arquiteturas de rede neural para classificação das simplificações, com auxílio do dicionário psicolinguístico;
- Comparar os resultados obtidos das sugestões com avaliação das palavras sugeridas por meio dos diferentes algoritmos de *word embedding*.

## 1.2 JUSTIFICATIVA

No campo de PLN, pode-se observar nos últimos anos os bons resultados advindos dos métodos distribucionais. Os métodos distribucionais dão significado a uma palavra a partir do seu contexto, ou seja, das palavras ao seu redor. Chamado de semântica vetorial, pois a palavra é transformada em um vetor, essa metodologia trabalha com vetores esparsos (grande maioria dos valores é igual a zero) e também com vetores densos (menor dimensionalidade e menos valores zerados) (JURAFSKY; MARTIN, 2016). Os vetores densos têm apresentado melhores resultados na obtenção de sinônimos, quando comparado aos vetores esparsos (JURAFSKY; MARTIN, 2016).

Como um de seus objetivos, o Processamento de Linguagem Natural busca melhorar a comunicação humano-humano (JURAFSKY; MARTIN, 2009). É muito comum deparar-se com textos que abusam de palavras antiquadas e que já possuem sinônimos que facilmente as substituiriam. Também é visto, em muitas oportunidades, a necessidade de mudar o público de publicações, como tornar o texto com uma linguagem mais infantil, ou um conteúdo especializado e torná-lo mais acessível ao público leigo (por exemplo, um texto jurídico). Não nativos dentro do processo de aprendizagem da língua portuguesa e portadores de deficiência cognitiva também são pessoas que se beneficiam dos textos em versões simplificadas devido às suas condições. A construção de um software, que faça essa simplificação a partir de sugestão de palavras mais comuns, pode melhorar a comunicabilidade dos textos produzidos e, por consequência, pode abranger um público maior. Como resultado, também pode atingir um dos objetivos do Processamento de Linguagem Natural que é melhorar a comunicação humano-humano.

## 1.3 ORGANIZAÇÃO DO DOCUMENTO

Esta seção apresenta a estrutura de organização deste trabalho. No Capítulo 2 apresenta-se o tema de Processamento de Língua Natural abrangendo história e seu contexto dentro da IA e também dos níveis de estudo da língua. Seguindo o capítulo, uma descrição da atividade de simplificação léxica é feita. O tema de RNA também é fundamentado com a

apresentação geral e das arquiteturas, formas de aprendizado, funções. O estudo feito sobre o tema de semântica vetorial é apresentado logo em seguida, com uma descrição geral e também dos diferentes algoritmos utilizados neste trabalho. Ao fim do Capítulo 2, alguns trabalhos relacionados a este são apresentados. Os materiais e métodos a serem utilizados na produção deste trabalho são apresentados no Capítulo 3. No Capítulo 4 são apresentados os resultados obtidos a partir dos materiais e métodos propostos e também a discussão deles. Por fim, é realizada uma conclusão de todo o trabalho aqui descrito no Capítulo 5.

# 2 FUNDAMENTAÇÃO TEÓRICA

Será desenvolvido neste capítulo o referencial teórico dos temas em estudo, abrangendo IA e o campo de PLN. Aprofunda-se nas áreas de semântica vetorial e seus distintos algoritmos, também é apresentada fundamentação para o tema de RNA, bem como, para simplificação léxica.

## 2.1 PROCESSAMENTO DE LÍNGUA NATURAL

A IA é uma área de estudo muito vasta e abrange os mais diversos temas. As técnicas estudadas em suas diferentes áreas podem contribuir entre si para o crescimento do campo no geral. Com o PLN não é diferente, a utilização de técnicas de IA e o estudo de forma científica da língua tem resultado em diversas aplicações industriais e comerciais (KIBBLE, 2013). Alguns exemplos comuns de aplicações são: tradutores automáticos, corretores ortográficos e ferramentas de *autocomplete*. Esta seção discorrerá sobre o histórico da PLN e sobre os níveis de estudo da língua, que se mostra como um importante tópico para o entendimento dos estudos realizados neste trabalho.

### 2.1.1 Histórico

O processamento de linguagem natural é uma tarefa que se distingue do processamento de dados. Jurafsky e Martin (2009) utilizam-se do programa do Unix "wc" para estabelecer uma comparação, onde diz que uma contagem de bytes é um processamento de dados, enquanto a contagem de palavras é um processamento de linguagem, pois já tem conhecimento

do conceito de palavras. No entanto, os primeiros marcos da evolução do processamento de linguagem natural são datados na década de 50 com os estudos na área de tradução automatizada e o movimento inicializado por Warren Weaver e sua carta conhecida como "Weaver Memorandum" (SILVA et al., 2007). Weaver movimentou a comunidade a respeito do tema, o que culminou no que se tem como o primeiro tradutor automático lançado. O sistema de tradução automática, desenvolvido pela IBM e a Georgetown University, continha um vocabulário de 250 palavras para a tradução do russo para inglês (HUTCHINS, 1994).

Jurafsky e Martin (2009) escrevem que a Teoria da Linguagem Formal é resultado dos estudos inicializados por Turing em 1943, o que foi a base, no início da década seguinte, para o conceito de autômatos. Chomsky (1956), fundamentando-se em outras pesquisas realizadas, introduziu pela primeira vez a ideia das máquinas de estados finitos e a gramática livre de contexto. De forma independente e em paralelo a Chomsky, Backus e Naur realizaram as mesmas averiguações durante a construção da linguagem de programação ALGOL (JURAFSKY; MARTIN, 2009).

Para Silva et al. (2007), a revolução na definição da linguagem formal acontece no lançamento do livro *Syntatic Structures* de Chomsky, em 1957, onde são apresentados dois níveis de estruturação sintática: a *d-structure* e a *s-structure*. A *d-structure* diz a respeito às regras de geração de linguagem e é uma camada mais profunda, ou seja, um conjunto finito de produções que, trabalhando sobre o vocabulário de uma língua, é capaz de gerar infinitos enunciados linguísticos, estes chamados de *s-structure*. Essa formalização desenvolvida por Chomsky é comumente denominada como gerativo-transformacional (SILVA et al., 2007).

### 2.1.2 Níveis de estudo da língua

Existem alguns conhecimentos da língua que são estruturais na construção de processadores de linguagem. Em exemplificação, Jurafsky e Martin (2009) utilizam o sistema HAL 9000 do filme "2001: A Space Odyssey", de Stanley Kubrick, para estabelecer alguns aspectos da língua. Ele escreve que o robô deve ter conhecimentos de morfologia, sintaxe, semântica composicional, pragmática, diálogo, entre outros, para poder realizar com sucesso as atividades que ele realiza no filme. Esses diferentes conhecimentos são chamados de níveis de estudo da língua. A divisão dos níveis de estudo são distintas para pesquisadores de diferentes áreas e se correlacionam aos campos e subcampos de interesse de cada um

(CANDIDO JUNIOR, 2013). Para este trabalho, escolheram-se os níveis conforme Jurafsky e Martin (2009):

- Fonético-Fonológico: estudo da pronúncia das palavras e a forma acústica que seus sons são percebidos;
- Morfológico: estudo do significado dos componentes que formam as palavras (morfemas);
- Sintático: estudo da estrutura e ordem das palavras, bem como suas relações;
- Semântico: estudo do significado das palavras e de suas relações;
- Pragmático: estudo do que é expresso além do significado explícito de uma sentença, ou, em outras palavras, das intenções do elemento comunicador;
- Discursivo: estudo da linguística em perspectiva de sentenças e unidades maiores que palavras.

Nas averiguações que se realizarão neste trabalho, os níveis de estudo de interesse, são os níveis morfológico (em particular, simplificação léxica), sintático (em particular, gramaticalidade) e semântico (em particular, semanticalidade).

## 2.2 SIMPLIFICAÇÃO LÉXICA

A simplificação léxica é uma entre algumas formas de simplificação textual, assim como a sintática e de discurso. Max (apud CANDIDO JUNIOR, 2013) escreve que a simplificação textual é uma maneira de diminuir a complexidade de um texto e mesmo assim manter as características semânticas e de informação, aumentando assim a inteligibilidade do texto. A simplificação léxica, por sua vez, é a substituição de palavras de baixo uso dentro da língua por aquelas com maior utilização (URANO, 2000). No exemplo a seguir, adaptado de Urano (2000), a substituição de “diligente” por “trabalhador” exemplifica uma simplificação léxica:

- Original: Todo mundo sabe que Paulo é **diligente** e gentil com os outros;
- Simplificado: Todo mundo sabe que Paulo é **trabalhador** e gentil com os outros.

Shardlow (2014) faz uma análise da relação entre as palavras “simples” e “complexo” para os estudos de simplificação textual, pois a simplicidade é relativa à palavra original, ou seja, uma palavra simplificada deve ser menos complexa que a original, porém pode ainda ser complexa quando observada de um contexto mais leigo. No exemplo adaptado do autor, a

expressão “perna fraturada” é uma versão simplificada para “tíbia fraturada” mas também pode ser a versão complexa de “perna quebrada”, quando analisado em outro contexto.

A simplificação léxica aumenta a acessibilidade dos textos em geral. Um exemplo de esforço dentro da área de simplificação textual já existente é a Simple English Wikipedia[1], que é uma versão do Wikipédia com artigos escritos com uma linguagem mais simples para atender pessoas menos letradas. Outro exemplo é a lei federal 10098/2000 que busca garantir o direito de acesso à informação e comunicação ao cidadão fornecendo uma linguagem simplificada em portais e páginas governamentais (CANDIDO JUNIOR, 2013). Dentre os públicos que podem se beneficiar de um texto simplificado, citados por Candido Junior (2013), estão pessoas em alfabetização, falantes não nativos, pessoas leigas na utilização de material técnico e portadores de afasia.

A simplificação léxica é tipicamente resumida em quatro etapas, segundo Shardlow (2014). A primeira delas é a identificação das palavras complexas dentro do texto, ou seja, aquelas que idealmente devem ser substituídas por outras de maior simplicidade. Na segunda etapa, deve-se identificar as palavras candidatas à substituição e é interessante localizar mais de uma palavra candidata para que possa encontrar a melhor nas etapas seguintes. A terceira etapa é responsável por refinar as candidatas à substituição para que concentre-se nas com maior proximidade de sentido dentro daquele contexto. Por fim, deve-se identificar a candidata mais simples e realizar a substituição da palavra original de maior complexidade.

## 2.3 REDES NEURAIS ARTIFICIAIS

As Redes Neurais Artificiais (RNAs), comumente referenciadas como redes neurais, são uma abordagem computacional bio-inspirada no cérebro humano. Da mesma forma em que o cérebro humano aprende a partir de experiências, as RNAs também buscam criar modelagens para tarefas (HAYKIN, 2009). Uma das possíveis definições para a área é apresentada a seguir.

> Uma rede neural é um processador distribuído massivamente paralelo composto de unidades de processamento simples que tem uma propensão natural para armazenar conhecimento experiencial e torná-lo disponível para uso. Ele se assemelha ao cérebro em dois aspectos:
>
> 1. O conhecimento é adquirido pela rede a partir do seu ambiente através de um processo de aprendizagem;

[1] http://simple.wikipedia.org/

> 2. As forças de ligação interneurônio, conhecidas como pesos sinápticos, são usadas para armazenar o conhecimento adquirido.
>
> Tradução própria de Haykin (2009)

Na apresentação histórica, realizada por Carvalho (2017), é apontado que as primeiras publicações na área de RNAs foram realizadas na década de 40 e abordavam temas como modelagem de redes neurais na simulação de máquinas, modelo de redes de auto-organização e também o método de aprendizado supervisionado utilizando Perceptron (detalhado na Seção 2.3.4). Já na década de 60 e 70, o autor destaca os trabalhos com atuação em redes neurais de visão, memória, controle e auto-organização. Por fim, são citadas as realizações da década de 80, que foram a criação do *backpropagation* (detalhado na Seção 2.3.4) e também os estudos sobre redes simétricas para otimização.

As RNAs são fundamentalmente compostas por neurônios artificiais, como o da Figura 1. Cada neurônio dentro de uma rede apresenta três elementos básicos (HAYKIN, 2009): um conjunto de sinapses, um agente somador e uma função de ativação. No modelo da Figura 1, o conjunto de sinapses é representado pelas $n$ entradas de $x_i$, onde cada sinapse advém ou da camada de entrada ou de um outro neurônio. O valor da entrada da sinapse é imediatamente multiplicado pelo seu peso $p$. O agente somador é responsável por acumular todas as sinapses atualizadas ($x \times p$) em $u$. Por fim, tem-se a função de ativação ($\varphi$) que é responsável pela propagação da amplitude do sinal. Ocultado da Figura 1, os *bias* atuam como modificadores do valor $u$ e sua aplicação resulta numa transformação afim[2] à soma ponderada das entradas pelos pesos (HAYKIN, 2009). O valor do *bias* deve ser adicionado a $u$, resultando assim em $v$. A Equação 1 apresenta a fórmula que resulta na saída $y$ do neurônio.

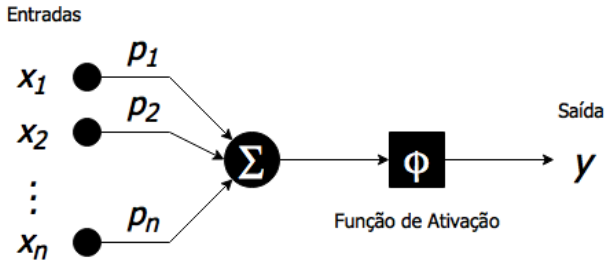



**Figura 1 – Modelo de um neurônio artificial**

**Fonte: Adaptado de Ferneda (2006)**

$$y = \varphi(b + \sum_{k=1}^{n} x_k p_k) \tag{1}$$

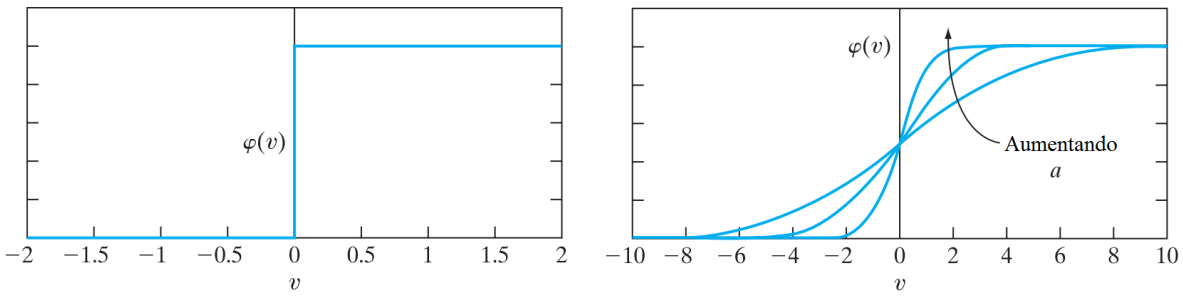
[2]Tranformação afim é uma tranformação que mantém a colinearidade e a proporção das distâncias.

A simulação do cérebro humano se dá com a combinação de vários neurônios artificiais, como o da Figura 1, bem como as trocas sinápticas realizadas por eles (FERNEDA, 2006). Os modelos de RNA adotam diferentes regras para atualizar os pesos (e *bias*) e aprender a partir de exemplos entrados na rede (CARVALHO, 2017). Essencialmente, a arquitetura de uma RNA é organizada em camadas que se distinguem entre si como: entrada, oculta/intermediária e saída. A camada de entrada não contém neurônios, é apenas a camada que introduz os padrões à rede e uma rede pode conter diversos pontos de entrada. A camada oculta pode ocorrer diversas vezes dentro de uma arquitetura de RNA e possuir diferenciadas quantidades de neurônios para cada uma. Ela é responsável pela maior parte do processamento e é definida como a extratora de características do treinamento (CARVALHO, 2017). A camada de saída é responsável pela entrega dos resultados obtidos nas camadas anteriores.

As RNAs podem ser arquitetadas em diversas formas e dentre as possíveis modificações estão a escolha da função de ativação, arquitetura e forma de aprendizado.

### 2.3.1 Função de Ativação

A função de ativação é muito importante para a RNA, pois é ela que definirá a propagação da saída do neurônio. Haykin (2009) apresenta duas funções básicas que podem ser utilizadas como função de ativação, a degrau e a sigmóide, conforme apresentado na Figura 2.




(a) Função degrau (b) Função sigmóide

**Figura 2 – Gráficos para as funções de ativação.**

**Fonte: Adaptado de Haykin (2009)**

No trabalho desenvolvido por McCulloch e Pitts em 1943, como escreve Haykin (2009), o autor definiu a função degrau como “tudo ou nada”, por sua característica binária. A Equação 2 apresenta a função, onde $v$ indica um valor de domínio da função.

$$y = \begin{cases} 1, \text{ se } v \geqslant 0 \\ 0, \text{ se } v < 1 \end{cases} \tag{2}$$

Outra função de ativação utilizada é a sigmóide esta função é uma suavização da função degrau e tem o benefício de ser contínua, logo diferenciável em todos os seus pontos (HAYKIN, 2009). A função é apresentada na Equação 3, onde parâmetro *a* indica a curvatura do declive do gráfico, sendo que quanto mais próximo ao infinito esse for, mais parecida com a função degrau ela ficará.

$$y = \frac{1}{1 + e^{-av}} \tag{3}$$

Existem outras funções que podem ser utilizadas como função de ativação, como a função rampa, tangente hiperbólica, Rectified Linear Unit (em tradução literal, Unidade Linear Retificada) (ReLU), entre outras.

### 2.3.2 Arquitetura

Projetar uma RNA demanda identificar uma arquitetura apropriada à solução do problema em questão. Dentro das arquiteturas possíveis estão as *feedforward* de camada única, *feedforward* com múltiplas camadas e também as recorrentes.

A arquitetura mais simples é a *feedforward* de camada única. Nessa estrutura não se tem camadas intermediárias, apenas a camada de entrada e a de saída e as sinapses ocorrem apenas da primeira para a segunda (HAYKIN, 2009).

A segunda arquitetura se diferencia da primeira na quantidade de camadas intermediárias, por isso é denominada *feedforward* com múltiplas camadas. Adicionando camadas à RNA, a capacidade de recuperar informações estatísticas aumenta (HAYKIN, 2009). Neste modelo, cada camada subsequente à primeira camada intermediária tem sua sinapse de entrada adquirida da saída dos neurônios artificiais da camada anterior, como apresentado na Figura 3.

A conexão entre as camadas também pode ser configurada diferentemente. Quando todo neurônio de uma camada está conectado a todo neurônio da camada subsequente, para todas as camadas, é dito que esta rede é completa. Por outro lado, quando existem conexões faltantes é dita incompleta.

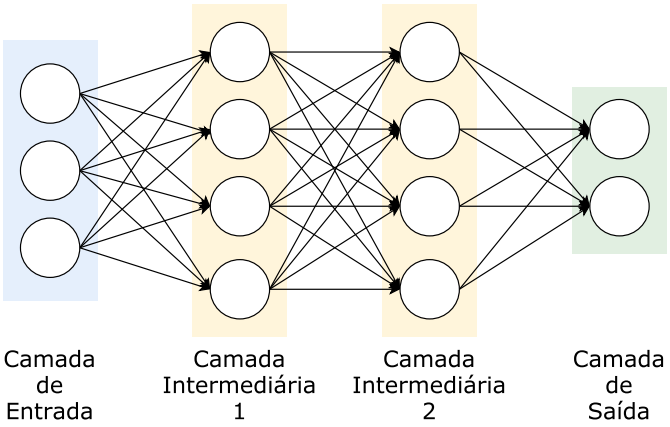



**Figura 3 – Exemplo de RNA *feedforward* multicamadas**

**Fonte: Adaptado de http://cs231n.github.io/neural-networks-1/**

Outra forma de arquitetar uma RNA é quebrando a ideia sequencial ao qual o sinal é transmitido. A arquitetura recorrente utiliza-se de realimentação, ou seja, a saída de um neurônio pode ser a entrada de outro na mesma camada ou anterior, dependendo se a rede é multicamadas ou não (FINOCCHIO, 2014). É considerada uma rede neural recorrente toda rede com ao menos uma conexão de realimentação, como a Figura 4.

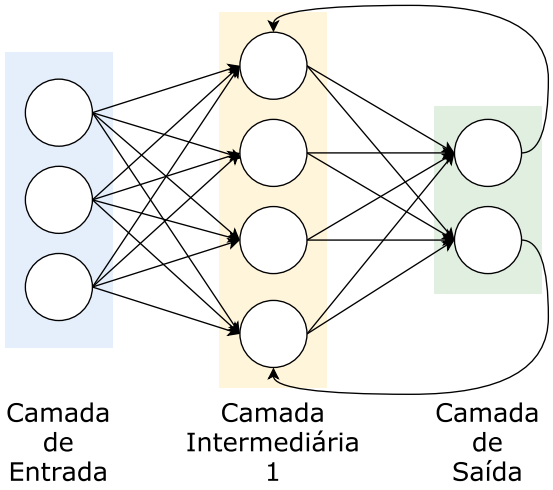



**Figura 4 – Exemplo de RNA recorrente com uma camada intermediária**

**Fonte: Autoria própria**

### 2.3.3 Formas de Aprendizado

Uma das características mais importantes das RNAs é sua capacidade de extrair regras, que são os seus pesos, e contraria-se dos sistemas computacionais tradicionais que necessitam de ter suas regras bem delimitadas (FINOCCHIO, 2014). Essas regras são criadas e ajustadas durante o treinamento da RNA, onde exemplos reais são apresentados à rede. Diz-se que uma rede está treinada para resolver uma classe de problemas quando esta apresenta soluções gerais para eles (FINOCCHIO, 2014). Haykin (2009) divide a forma de aprendizado de uma rede em duas categorias: aprendendo com um professor e aprendendo sem um professor. Nessa divisão o aprendizado realizado com a assistência de um professor é dito aprendizado supervisionado e já a segunda divide-se em outras duas categorias, aprendizado por reforço e não supervisionado. Para

No aprendizado supervisionado, tem-se a ideia de um professor detentor das respostas corretas para as entradas feitas na rede (HAYKIN, 2009). Ao gerar uma reposta a uma dada entrada, a rede calcula o quão errado essa resposta apresentada é e então faz uma correção dos pesos. Essa correção é realizada em sentido contrário ao da entrada, ou seja, inicia da camada de saída em direção à entrada. O treinamento e, consequentemente, a atualização dos pesos acontecem até que esta rede consiga atingir níveis aceitáveis para o conjunto de treinamento (FINOCCHIO, 2014). A partir do momento em que a rede consegue emular o conhecimento do professor, este é deixado, e os pesos fixados, implicando assim na criação de memória de longo prazo. A rede então está pronta para solucionar problemas do ambiente real de forma independente (HAYKIN, 2009).

Ao contrário do anterior, o aprendizado sem o auxílio de um professor não possui exemplos que possam nortear o aprendizado da rede. No primeiro tipo, o aprendizado por reforço, a rede utiliza-se de um agente externo chamado de crítico que irá receber um sinal de entrada e então propagar correções conforme a heurística adotada (HAYKIN, 2009). Sutton e Barto (2016) escrevem que as duas principais características que distinguem o aprendizado por reforço dos demais aprendizados são a procura por tentativa e erro, pois identifica ações que maximizam as recompensas dentro de um ambiente, e recompensas atrasadas, pois as ações da rede geram benefícios momentâneos e também para as mesmas situações nos futuro.

Para o aprendizado não supervisionado, não se tem professor e nem um agente que auxilie a correção durante o aprendizado. Neste tipo de aprendizado tem-se apenas entradas e a rede deve identificar dentro desta, subconjuntos com elementos similares. A modificação dos pesos são feitas conforme a diferença dos resultados obtidos entre esses conjuntos similares,

pois são esperados resultados semelhantes (FINOCCHIO, 2014).

### 2.3.4 Algoritmo Backpropagation em Redes Perceptron de Multicamadas

A implementação de RNAs de camada única limita a formação de uma representação interna da rede, os padrões apresentados à entrada serão mapeados diretamente à saída (FINOCCHIO, 2014). Nesse cenário, a rede fica impossibilitada de criar novos padrões e mapear novos conhecimentos e acaba não gerando saídas dissimilares à entrada. Logo, como verifica-se em Haykin (2009), RNAs de camada única estão limitadas em classificar apenas os padrões que são linearmente separáveis. Um problema clássico desta questão é o problema do "ou" exclusivo (XOR, sigla em inglês), onde é possível verificar necessidade de duas camadas para conseguir classificar as saídas dessa verificação lógica (FINOCCHIO, 2014).

Por algum tempo, o limitador do avanço das RNAs de múltiplas camadas, foi a complexidade em fazer a correção dos pesos e então criar modelos eficientes, pois exigia muitos recursos computacionais para se realizar essa atualização (ROJAS, 1996). Um dos modelos de RNAs de multicamadas mais utilizados são as redes Perceptron de Multicamadas (MLP). São características da MLP, segundo Haykin (2009): cada neurônio na rede tem uma função de ativação que não é linear e que é diferenciável, a rede conterá ao menos uma camada intermediária e a rede terá um nível alto de conectividade entre os neurônios. Estas redes são vastamente utilizadas em conjunto com o algoritmo de treinamento Backpropagation. Este algoritmo tornou, em 1986, o treinamento de redes de multicamadas, como a MLP, possíveis, pois diminuiu a complexidade do algoritmo de atualização de pesos (FINOCCHIO, 2014).

O treinamento utilizando-se do algoritmo do Backpropagation ocorre em duas etapas principais (CARVALHO, 2017). Na primeira etapa é apresentado à rede um padrão para camada de entrada e, como uma MLP utiliza a forma de aprendizado *feedforward*, o sinal é propagado até chegar na camada de saída. Após a primeira etapa, é calculado um erro da saída da rede com a saída desejada e propagado em sentido reverso, corrigindo então cada um dos pesos (CARVALHO, 2017).

Na abordagem do algoritmo de Backpropagation a minimização da função que calcula o erro, em relação aos pesos e *bias*, é a solução para a modelagem do problema do mundo real (ROJAS, 1996). É importante que a função utilizada para calcular o erro seja contínua e diferenciável, pois é necessário calcular o gradiente para realizar a correção (ROJAS, 1996).

Na primeira etapa do algortimo Backpropagation, no passo para frente (*forward pass*),

entra-se o sinal na camada de entrada, este sinal é processado pelos Perceptrons, como na Equação 1, sendo que a saída de um Perceptron é a entrada de outro, até que chegue à camada de saída e tenha o resultado para esta etapa. A segunda etapa, passo para trás (*backward pass*), tem como primeiro objetivo calcular os erros para cada neurônio de saída, em relação ao resultado esperado (MAZUR, 2015). Na Equação 4 faz-se o cálculo do erro empírico genérico ($e$) para uma saída desejada e uma saída obtida de um neurônio ($n$).

$$e(n) = desejado(n) - obtido(n) \tag{4}$$

O grande objetivo do algoritmo é minimizar a função do erro, para que após ter realizado o treinamento da rede, os novos padrões apresentados à ela possam gerar os mesmos resultados dos apresentados durante o treinamento (ROJAS, 1996). Neste momento, deve-se então propagar o erro começando pela camada de saída e indo em direção à camada de entrada (FINOCCHIO, 2014). Os pesos internos à RNA vão sendo corrigidos conforme a sua contribuição para o erro. O treinamento se estende enquanto a RNA não apresentar resultados que atendam à modelagem do problema.

### 2.3.5 Regularização

As RNAs tendem a criar modelagens de complexas relações presentes nas bases de dados, porém diversas dessas relações não se assemelham ao problema real (SRIVASTAVA et al., 2014). O estado que uma RNA atinge, quando aprende além do modelo similar ao mundo real, mas também informações específicas da base de dados de treinamento, é chamado de *overfitting*. Como forma de contenção desse comportamento, utiliza-se algoritmos de regularização como *dropout* e L2, utilizados neste trabalho.

O *dropout*, conforme apresentado por Srivastava et al. (2014), desabilita randomicamente e temporariamente parte dos neurônios e suas conexões. Os autores afirma que os algoritmos de *backpropagation* são capazes de se adaptar e aprender conforme os neurônios são desabilitados durante o treinamento. Ainda, é apresentado no trabalho que, com a utilização da regularização *droupout*, obteve-se bons resultados em redes de reconhecimento de voz, classificação de documentos, etc.

Outra forma de regularização é a L2, em linhas gerais, a proposta deste método é diminuir o impacto dos pesos das conexões no aprendizado e assim diminuir o *overfitting* (MCCAFFREY, 2017). A forma que a L2 atua é diminuindo a influência dos pesos no erro

final, aplicando apenas uma fração do peso. Em outras palavras, pode-se entender a L2 como uma forma de esquecimento do aprendizado durante o treinamento, este por sua vez diminui o *overfitting*.

## 2.4 SEMÂNTICA VETORIAL

As averiguações no campo de semântica vetorial, ou métodos distribucionais, tiveram início na década de 50, onde Harris observou que as palavras podem ser distinguidas a partir do contexto ao qual se inserem (CLARK, 2014). Embora Harris tenha introduzido a noção da semântica vetorial, foi com Firth, em 1957, que o embasamento dos estudos dos métodos distribucionais modernos se consolidou, pois se estava interessado na relação do comportamento das palavras em relação ao contexto ao qual ela está inserida. Logo, os métodos distribucionais são definidos como a extração do significado de uma palavra a partir da distribuição das palavras que formam o seu contexto (JURAFSKY; MARTIN, 2016).

Uma das formas vetoriais mais básicas de representação de uma palavra é o vetor de codificação *one-hot* (um-quente, em tradução literal). Essa representação binária tem seu nome importado da área de Circuitos Digitais e cada palavra é representada como um vetor sobre $\mathbb{R}^{|V|\times 1}$ (CHAUBARD et al., 2015), onde $|V|$ é o tamanho do vocabulário. No vetor, cada posição representa uma palavra do vocabulário. Para um vocabulário, cada palavra é convertida em um vetor no qual sua respectiva posição vale 1 enquanto as demais posições valem 0. No exemplo a seguir, adaptado de Qu et al. (2016), são mostrados 3 casos de *one-hot* de diferentes dimensionalidades:

$$\underbrace{[0,1,0,0,0,0,0]}_{\text{Dia da Semana=Terça- Feira}} \qquad \underbrace{[0,1]}_{\text{Gênero=Masculino}} \qquad \underbrace{[0,0,1,0,\ldots,0,0]}_{\text{Cidade=Londres}}.$$

Expandindo o exemplo, o vetor *one-hot* para Segunda-feira seria $[1,0,0,0,0,0,0]$, já o vetor para gênero feminino, cujo o vocabulário tem tamanho dois, seria $[1,0]$.

Apesar do benefício de sua simplicidade, a codificação *one-hot* não consegue identificar similaridades entre as palavras. Em um exemplo adaptado de Goldberg (2016), "cachorro" é tão dissimilar a "gato" quanto é a palavra "pensando". Outra desvantagem do uso desses vetores é sua esparsidade. Nas ferramentas de RNA, vetores de alta dimensionalidade e muito esparsos são mais trabalhosos de serem processados (GOLDBERG, 2016), pois necessitam atualizar muitos pesos (JURAFSKY; MARTIN, 2016), além da chamada maldição

da dimensionalidade (RUSSELL; NORVIG, 2013).

Na tentativa de melhorar a captura de similaridade semântica entre as palavras e também diminuir a dimensionalidade dos vetores, surgiram as representações vetoriais densas. Esses vetores densos tem como aspecto armazenar as características entre as palavras em um vetor de baixa dimensão (geralmente, entre 50 e 300 posições). Nesse caso, palavras com características parecidas vão gerar vetores semelhantes (GOLDBERG, 2016). Essa representação de vetores densos é chamada de modelo espaciais de vetores (*vector space models*) e, basicamente, existem dois métodos diferentes que os geram, o método baseado em contagem e o método preditivo (TENSOR FLOW, 2017). O primeiro faz a contagem de coocorrência de uma palavra e seus contextos dentro de um *corpus*[3] e então calcula algumas estatísticas para agrupar informações em um vetor (TENSOR FLOW, 2017).

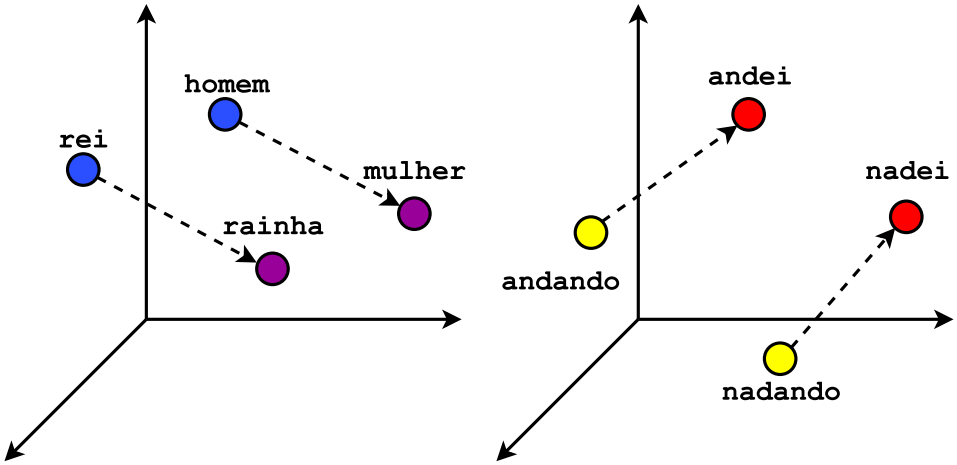
Segundo Baroni et al. (2014), o método baseado em contagem é o mais antigo e sempre apresentou a necessidade de aplicar transformações para melhorar o seu desempenho. Modificações como reponderamento dos pesos e técnicas de redução de dimensionalidade são exemplos citados pelos autores. O método preditivo é uma forma supervisionada de criação dos vetores que, ao invés de compor uma entrada com os contextos das palavras, faz uma predição desses contextos (BARONI et al., 2014). Neste sentido, um contexto geralmente são as palavras na vizinhança de uma palavra de interesse em uma dada sentença. No estudo elaborado por Baroni et al. (2014), foi averiguado a superioridade dos métodos preditivos na maioria das tarefas de semântica lexical propostas, como relacionamento semântico e detecção de sinônimos, porém o autor ressalvou que há um vasto número de parâmetros possíveis para os métodos baseados em contagem e que em outros cenários poderiam melhorar o desempenho.

Os vetores densos também são comumente chamados de *word embeddings*, e este trabalho utilizará três algoritmos diferentes para a sua geração: dois modelos preditivos, Skip-Gram e CBOW (MIKOLOV et al., 2013a; MIKOLOV et al., 2013b), e um modelo baseado em contagem, GloVe (PENNINGTON et al., 2014). Tanto o método Skip-Gram quanto o CBOW foram apresentados por Mikolov et al. (2013a) e lançados em um pacote de software chamado Word2vec. A principal vantagem desses métodos não é a capacidade de predizer vetores a partir do contexto e sim a velocidade com que o algoritmo é treinado, sua eficiência e por terem disponibilizada uma plataforma online com diversos recursos (JURAFSKY; MARTIN, 2016). Além desses três modelos, serão utilizadas duas abordagens de otimização dos algoritmos do Word2vec (Skip-Gram e CBOW): Wang2vec (LING et al., 2015) e fastText (BOJANOWSKI et al., 2016).

Esses modelos são capazes de capturar múltiplos níveis de similaridade e então

[3]Um corpus pode ser definido como um conjunto de textos escritos que servem como base de análise.

expressar diferentes relações semânticas entre as palavras (MIKOLOV et al., 2013a). A Figura 5 demonstra alguns exemplos de relações interessantes que podem ser capturadas, como a relação masculino-feminino (a) e a relação de tempo verbal (b). Também é possível realizar operações como vetor(“rei”) - vetor(“homem”) + vetor(“mulher”), que resultam em um vetor bem próximo do vetor da palavra “rainha” (MIKOLOV et al., 2013a).




(a) masculino-feminino (b) tempo verbal

**Figura 5 – Visualização da informação semântica entre palavras.**
**Fonte: Adaptado de Tensor Flow (2017)**

Uma outra forma de identificar as relações semânticas entre as palavras é utilizando, por exemplo, a distância euclidiana, ou a similaridade do cosseno, entre os vetores (PENNINGTON et al., 2017). Em um exemplo de *embeddings* gerados com o algoritmo GloVe e tendo a palavra “sapo” como alvo, a distância euclidiana revelou algumas palavras de uso não frequente, mas que estavam na vizinhança, como “litoria” e “leptodactylidae” (Figura 6).

(a) litoria (b) leptodactylidae

**Figura 6 – Ilustração das palavras próximas à palavra “sapo”.**
**Fonte: (a) https://pixabay.com/en/finger-coral-tree-frog-8341/**
**(b) http://www.pybio.org/2341/leptodactylidae/**

### 2.4.1 CBOW

O algoritmo CBOW prediz uma palavra (palavra objetivo) a partir de um contexto oferecido, ou seja, sabendo-se que as palavras {“O”, “gato”, “o”, “rato”}, por exemplo, ocorreram em uma janela de contexto (sem a necessidade de preservar a ordem de corrência), o algoritmo deve prever uma palavra adequada, que no exemplo anterior uma das possibilidades poderia ser a palavra objetivo “pegou”, formando assim “O gato pegou o rato” (CHAUBARD et al., 2015). Mikolov et al. (2013a) descrevem o modelo em três camadas básicas: entrada, projeção e saída, conforme a Figura 7. Na camada de entrada têm-se $2L$ vetores *one-hot*, onde $L$ é usado para definir o tamanho da janela de contexto e indica o número de palavras anteriores e posteriores à palavra objetivo. A camada de projeção é o *word embedding* da palavra objetivo. E na camada de saída encontra-se um vetor com o tamanho do vocabulário $V$ utilizado, onde cada elemento deste vetor é uma probabilidade para uma palavra diferente do vocabulário.

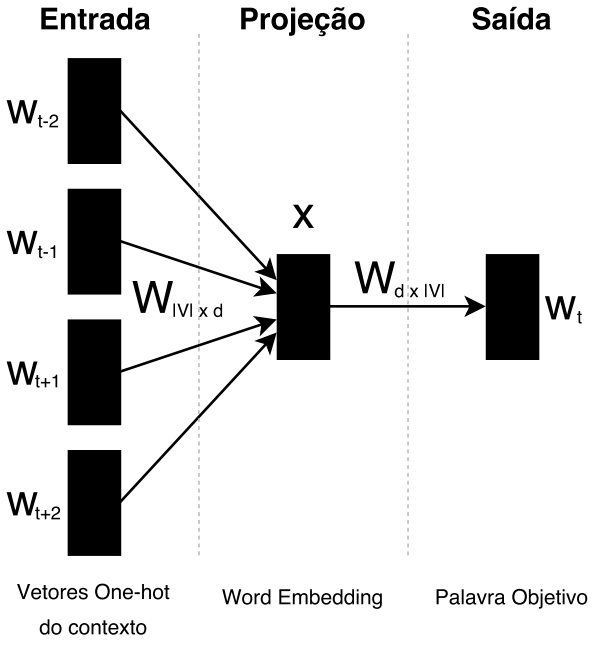



**Figura 7 – Modelo CBOW $L = 2$**
**Fonte: Adaptado de Mikolov et al. (2013a)**

Sendo $t$ a posição da palavra $w$ dentro de um *corpus* $T$, o vetor *one-hot* da palavra objetivo é $w_t$ e seu contexto é dado pela representação vetorial *one-hot* como $w_{t-L}, \ldots, w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}, \ldots, w_{t+L}$.

O CBOW utiliza-se de duas matrizes de pesos, $W$ e $W'$. $W$ é utilizada para gerar o

*word embedding* da camada de projeção e $W'$, para gerar o vetor com probabilidades na camada de saída.

A matriz $W$ tem $|V|$ linhas, onde $|V|$ é o tamanho do vocabulário, e $d$ colunas, onde $d$ é a dimensão escolhida para o vetor de *embedding*, logo $W_{|V|\times d}$. Cada linha $i$ em $W$ representa uma palavra diferente do vocabulário $V$ na forma vetorial de *embedding*. A matriz $W'$ é muito semelhante à anterior, porém tem $d$ linhas e $|V|$ colunas, sendo que cada coluna $j$ possui os valores que serão utilizados para calcular as pontuação para o vetor de saída. Para a inicialização dessas matrizes é recomendado pelos autores do algoritmo gerar números randômicos uniformemente distribuídos entre $-\frac{1}{2d}$ e $\frac{1}{2d}$.

Primeiramente, na transição da camada inicial para a de projeção, cada vetor *one-hot* do contexto é multiplicado pela matriz $W$, como apenas um elemento de cada vetor é diferente de zero, o resultado será o vetor de *embedding* $y$ correspondente àquela palavra no vocabulário. A forma com que o CBOW faz a entrada das $2L$ palavras do contexto é realizando a média dos vetores embutidos, conforme a Equação 5. O resultado dessa equação é o *word embedding* $x$, correspondente a todo o contexto de entrada.

$$x = \frac{y_{t-L} + ... + y_{t-1} + y_{t+1} + ... y_{t+L}}{2L} \tag{5}$$

Na segunda etapa é realizada a transição da camada de projeção para a de saída. Multiplica-se o vetor $x$ e a matriz $W'$, resultando em um vetor $v$ de dimensão $1x|V|$, onde cada $k$ elemento é uma pontuação para a palavra correspondente em $V$. Por esses valores não serem probabilidades e sim pontuações, aplica-se então a função Softmax sobre cada $k$ elemento em $v$, conforme a Equação 6. Resulta-se então em um vetor $\hat{w}_t$.

$$P(w_t|w_{contexto}) = \frac{e^{(v_k)}}{\sum_{m=1}^{|V|} e^{(v_m)}} \tag{6}$$

Após realizada a normalização dos valores, o algoritmo necessita aprender atualizando os pesos das matrizes $W$ e $W'$. Para mensurar o erro obtido a partir da saída desejada ($w_t$) e da saída obtida ($\hat{w}_t$), calcula-se então a entropia cruzada ($H$) entre ambas, como a Equação 7 apresenta (CHAUBARD et al., 2015).

$$H(\hat{w}_t|w_t) = -\sum_{m=1}^{|V|} w_{t(m)} log(\hat{w}_{t(m)}) \tag{7}$$

Uma vez que $w_t$ é um vetor no formato *one-hot*, apenas a sua $k$ posição será 1, isso torna possível a simplificação da função $H$ (Equação 8).

$$H(\hat{w}_t|w_t) = -log(\hat{w}_{t(k)}) \tag{8}$$

Por fim, a função objetivo $J$ a ser minimizada pode ser obtida através substituição de $\hat{w}_t$ por $P(w_t|w_{contexto})$, resultando então na Equação 9 e simplificada para a Equação 10. Utiliza-se do gradiente descendente para realizar a atualização dos pesos.

$$J = -log\frac{e^{(v_k)}}{\sum_{m=1}^{|V|} e^{(v_m)}} \tag{9}$$

$$J = v_k + log\sum_{m=1}^{|V|} e^{(v_m)} \tag{10}$$

### 2.4.2 Skip-Gram

O algoritmo Skip-Gram funciona de uma forma muito semelhante ao CBOW, pois prediz um contexto, dentro de uma janela de tamanho predeterminado, como é apresentado pela Figura 8. Por exemplo, a partir de uma palavra dada "pegou", deve-se prever as palavras {"O", "gato", "o" e "rato"}. Aumentando-se a janela de palavras do contexto, aumenta-se a complexidade do algoritmo (MIKOLOV et al., 2013a). Jurafsky e Martin (2016) escrevem que o Skip-Gram é a imagem do método CBOW, descrito na Seção 2.4, mas apresenta comportamento diferente para tarefas específicas.

Utilizando a mesma ideia apresentada para o CBOW, o Skip-Gram também implementa duas matrizes, $W_{|V|\times d}$ e $W'_{d\times|V|}$. Outra característica é a separação em camadas de entrada, projeção e saída. A camada de entrada será um vetor *one-hot* de uma palavra $w$ na posição $t$ do *corpus* $T$, escolhida para a obtenção dos $2L$ vetores de contexto. A transição da camada de entrada para a camada de projeção é a multiplicação do vetor *one-hot* pela matriz $W$, resultando no *embedding* $x$, conforme a Equação 11.

$$x = w_t \times W_{|N|\times d} \tag{11}$$

Para realizar a transição da camada de projeção para a camada de saída é necessário projetar os $2L$ vetores com pontuações, aqui denominados de $v_{t-L},...,v_{t-1},v_{t+1},...,v_{t+L}$. Como será multiplicado o mesmo valor $x$ pela mesma matriz $W'$ para todos os $2L$ vetores, todos resultarão no mesmo vetor $v$. Subsequentemente, necessita-se transformar as pontuações em probabilidades utilizando a função Softmax, conforme a Equação 12, assim como é feito para o CBOW, onde $k$ e $m$ são as posições de elementos dentro do vetor $v$.

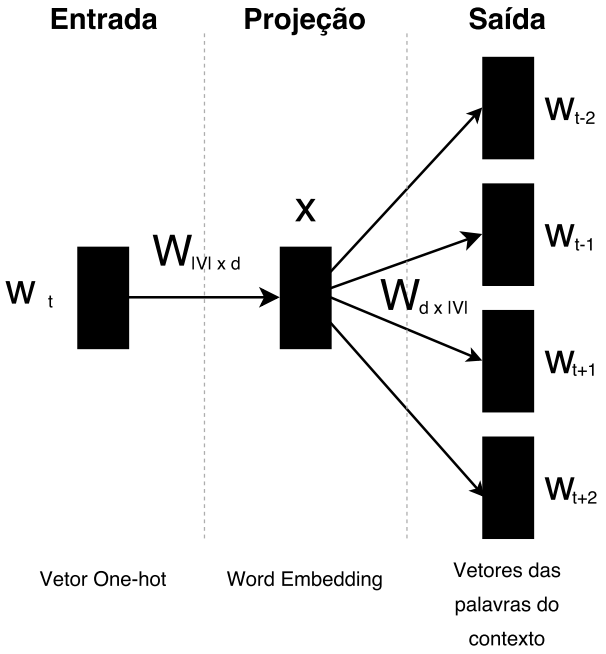



**Figura 8 – Modelo Skip-Gram**

**Fonte: Adaptado de Mikolov et al. (2013a)**

$$P(w_{contexto}|w_t) = \frac{e^{(v_k)}}{\sum_{m=1}^{|V|} e^{(v_m)}} \tag{12}$$

O aprendizado do modelo Skip-Gram se dá da mesma maneira que o CBOW, a diferença é que o erro é calculado usando o produto entre as probabilidades de cada palavra da janela de contexto. A Equação 13 apresenta a função objetivo $J$ do modelo Skip-Gram, onde $w_{contexto}$ é expandido e a Equação 14 apresenta já com o produtório responsável pelas probabilidades.

$$J = -logP(w_{t-L}, ..., w_{t-1}, w_{t+1}, ..., w_{t+L}|w_t) \tag{13}$$

$$J = -log \prod_{m=0, m\neq L}^{2L} P(w_{t-L+m}|w_t) \tag{14}$$

Faz-se então a substituição de $P$ pela Equação 12, resultando então na Equação 15, onde $r$ itera pelos elementos de cada vetor, e $m$ itera pelos vetores da janela $L$.

$$J = -log \prod_{m=0, m\neq L}^{2L} \frac{e^{(v_{k,m})}}{\sum_{r=1}^{|V|} e^{(v_r)}} \tag{15}$$

Por fim, expandindo-se a Equação 15 tem-se a função objetivo J final (Equação 16).

$$J = -\sum_{m=0,m\neq L}^{2L} e^{(v_{k,m})} + 2L \times log \sum_{r=1}^{|V|} e^{(v_r)} \quad (16)$$

Com a função objetivo estabelecida, pode-se então calcular os gradientes respectivos de cada peso das matrizes e atualizá-los via gradiente descendente (CHAUBARD et al., 2015).

### 2.4.3 Amostras Negativas

O processo de aprendizagem e atualização das matrizes de pesos dos algoritmos preditivos descritos neste trabalho (CBOW, Skip-Gram), pode apresentar um custo muito elevado. No exemplo de McCormick (2017), utilizando-se um vetor com 300 dimensões em um vocabulário de 10 mil palavras, será necessário a atualização de duas matrizes com 3 milhões de pesos cada. Tratando essa necessidade de melhora no rendimento do algoritmo, Mikolov et al. (2013b) apresentam a ideia de amostras negativas. A proposta central desta abordagem é, para cada iteração de treinamento, não atualizar todos os pesos, mas sim apenas uma amostra deles (RONG, 2014). O objetivo então será se aproximar das amostras com palavras de contexto/objetivo de interesse e se distanciar das palavras que não fazem parte da vizinhança (amostras negativas) (JURAFSKY; MARTIN, 2016).

Nos experimentos de Mikolov et al. (2013b), verificou-se que para pequenos *corpora*[4] um número de 5 a 20 amostras negativas apresentaram bons resultados, enquanto que para *corpora* maiores, entre 2 e 5 palavras. Para conseguir a probabilidade $P$ de uma palavra ($w_i$) pertencer ao conjunto de amostras negativas, utiliza-se da frequência das palavras $f(w_i)$, quantidade de aparições da palavra dentro do *corpus*, elevada a $3/4$ (MCCORMICK, 2017), que é um peso que se demonstrou superior em comparação a outras formas de otimização (MIKOLOV et al., 2013b). Por fim, divide-se a frequência da palavra pela soma das frequências de todas as palavras do vocabulário, como demonstra a Equação 17.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_i)^{3/4}} \quad (17)$$

[4]Plural de *corpus*

### 2.4.4 Wang2vec

Os modelos preditivos Skip-Gram e CBOW do Word2vec desconsideram a ordem das palavras, conforme já apresentado. Para muitas tarefas da área de PLN, como etiquação morfológica (*part-of-speech tagging*), análise sintática de dependências (*dependecy parsing*) (LING et al., 2015) e também para os objetivos deste trabalho, que considera a sintaxe como uns dos critérios para uma boa simplificação, a ordem das palavras é de grande relevância. Visando essas necessidades, Ling et al. (2015) propuseram duas modificações aos algoritmos CBOW e Skip-Gram a fim de considerar a ordem das palavras para a geração dos *embeddings*.

A alteração realizada no algoritmo do Skip-Gram se dá na matriz de transição da camada de projeção para a camada de saída. Por exemplo, para a frase "o gato pulou no colchão", um possível padrão de treinamento para o Skip-Gram classico é a entrada *one-hot*(pulou) e saída esperada *one-hot*(o) + *one-hot*(gato) + *one-hot*(no) + *one-hot*(colchão). Já na versão Wang2vec, a saída desejada muda para -2*one-hot*(o) + -1*one-hot*(gato) + 1*one-hot*(no) + 2*one-hot*(colchão). Isso força a rede a aprender a ordem na qual o contexto aparece.

Basicamente, o mesmo processo ocorre para a versão CBOW do modelo Wang2vec, apenas trocando-se entrada com saída desejada.

### 2.4.5 fastText

A forma com que os algoritmos Word2vec constroem seus vetores não levam em consideração a estrutura interna das palavras. Para línguas com uma riqueza morfológica, como o português, onde um verbo pode ter diversas flexões (fala, falam, falamos, ...), algumas formas ocorrem raramente, gerando assim, vetores de baixa qualidade. Para melhorar a representação vetorial dessas palavras, Bojanowski et al. (2016) propuseram uma forma de cálculo dos *embeddings* onde considera-se a nível grafema (letras e símbolos) ou invés de lexemas (palavras e *tokens*). Para isso, é utilizada uma RNA recorrente do tipo Long Short-Term Memory (HOCHREITER; SCHMIDHUBER, 1997). A entrada são os caracteres, um por vez, e a saída desejada é *one-hot* da palavra em questão. Na camada oculta é formado o *embedding*.

### 2.4.6 GloVe

Ao contrário dos algoritmos preditivos já descritos, o GloVe utiliza-se de estatísticas sobre uma matriz de coocorrência de palavras para gerar os *embeddings* (JURAFSKY; MARTIN, 2016). Para Pennington et al. (2014), os métodos preditivos se apresentam em desvantagem quando comparado aos métodos baseados em contagem, como o GloVe, pois estes utilizam os dados estatísticos de coocorrência das palavras dentro de um texto inteiro enquanto o primeiro apenas faz diversas análises locais. Por outro lado, os autores afirmam que embora as estatísticas de coocorrência de uma palavra dentro de um *corpus* seja a maior fonte de informação para os métodos não supervisionados de aprendizado de representação de palavras, ainda segue inexplicado o processo da captura de semântica.

A Tabela 1 exemplifica a estrutura típica de uma matriz de coocorrência. Utiliza-se de um vocabulário genérico constituído apenas das palavras “flor”, “jardim”, “casa”, “nação”. Os números representam a quantidade de vezes que uma palavra ocorreu no contexto de outra, percebe-se que a matriz mantém simetria, como “jardim”-“flor” e “flor”-“jardim” tem coocorrência.

**Tabela 1 – Exemplo de matriz de coocorrência.**

| | flor | jardim | casa | nação |
|---|---|---|---|---|
| flor | X | 5 | 3 | 0 |
| jardim | 5 | X | 4 | 1 |
| casa | 3 | 4 | X | 2 |
| nação | 0 | 1 | 2 | X |

**Fonte: Autoria própria**

Pennington et al. (2014) apresentam o método utilizando-se de uma matrix $X$ com $i$ linhas e $j$ colunas, onde $i$ e $j$ são as palavras do vocabulário e uma entrada $X_{ij}$ é o número de vezes que uma palavra $j$ ocorre no contexto de uma palavra $i$. Ainda, entende-se por $X_i$ a soma das $k$ colunas da palavra na linha $i$, como $X_i = \sum_k X_{ik}$. Por fim, tem-se como a probabilidade de uma palavra $j$ aparecer no contexto de uma palavra $i$ o valor de $P(j|i)$, que é dado por $P(j|i) = P_{ij} = \frac{X_{ij}}{X_i}$.

Alguns aspectos semânticos das palavras podem ser identificados utilizando as probabilidades ($P_{ij}$) e a razão entre elas (PENNINGTON et al., 2014). Para o exemplo da Tabela 2, faz-se uma observação onde $i = gelo$ e $j = vapor$ e varia-se o contexto $k$. A razão $P_{ik}/P_{jk}$ tem um valor grande quando o contexto $k$ é próximo apenas de $i$, como “sólido” (8, 9 na Tabela 2), e um valor baixo quando é próximo apenas de $j$, como a palavra “gás” (0,085). O

valor da razão se aproxima de 1 quando a palavra é próxima de $i$ e $j$, como "água", ou quando são distantes de ambas, como "moda". As razões das probabilidades, portanto, é o que mantém as informações das palavras, ao invés das probabilidade sozinhas (RUDER, 2016) e é o que deve ser mantido em um vetor (PENNINGTON et al., 2014).

**Tabela 2 – Probabilidades modelo GloVe.**

| Probabilidade e Razão | k = sólido | k = gás | k = água | k = *moda* |
|---|---|---|---|---|
| $P(k\|gelo)$ | $1,9x10^{-4}$ | $6,6x10^{-5}$ | $3,0x10^{-3}$ | $1,7x10^{-5}$ |
| $P(k\|vapor)$ | $2,2x10^{-5}$ | $7,8x10^{-4}$ | $2,2x10^{-3}$ | $1,8x10^{-5}$ |
| $P(k\|gelo)/P(k\|vapor)$ | $8,9$ | $8,5x10^{-2}$ | $1,36$ | $0,96$ |

**Fonte: Adaptado de Pennington et al. (2014)**

A Equação 18 descreve uma função sobre os *embeddings*. Como exemplo, ela deve receber com $w_i$ = *embedding*("gelo"), $w_j$ = *embedding*("vapor") e $\tilde{w_k}$ uma palavra de contexto (exemplo: "sólido").

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{18}$$

Existem diferentes formas de implementar a função F. Como os *embeddings* são vetores e o espaço vetorial é linear, os autores (PENNINGTON et al., 2014) optam por usar a diferença entre os vetores $w_i$ e $w_j$. Além disso, tendo em vista que a saída da função é linear, os autores aplicam o produto escalar da diferença prévia com o contexto $\tilde{w_k}$. O resultado é ilustrado na Equação 19. Outra constatação é a de que um lado da equação está lidando com valores escalares e o outro com vetores. Para solucionar isso, primeiramente é realizado o produto vetorial entre $(w_i - w_j)$ e $\tilde{w}$.

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{19}$$

Para que a operação linear de diferença vetorial do lado esquerdo da equação 19 corresponda a divisão no lado direito, uma abordagem é aplicar função exponencial natural no lado direito, tanto no nominador, quanto no denominador (já que o resultado de divisão pode ser calculado aplicando-se a diferença de seus expoentes $e^a/e^b = e^{(a-b)}$). Os autores mostram que a busca pelos melhores vetores pode ser feita resolvendo-se o problema de otimização apresentada pela Equação 20, para $f$ conforme Equação 21, onde é recomendado inserir parâmetros de *bias* $b_i$ e $\tilde{b}_k$, além utilizar-se os parâmetros $x_{max} = 100$ e $\alpha = 3/4$.

$$J = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij})(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - log(X_{ij}))^2 \tag{20}$$

$$f(x) = \begin{cases} (x/x_{max})^{\alpha}, \text{ se } x < x_{max} \\ 1, \text{ caso contrário} \end{cases} \tag{21}$$

## 2.5 TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns exemplos de trabalhos correlatos a este. Tem-se como objetivo elucidar aspectos de magnitude do problema, bem como justapor a abordagem utilizada neste trabalho em relação a outras soluções.

Com o objetivo de realizar simplificação para terminologia científica, Kim et al. (2016) introduziram o SimpleScience, que realiza a simplificação léxica utilizando-se de uma base, denominada SimpleSciGold. Essa base contém simplificações para termos científicos que foram anotadas também por humanos. O software utiliza-se do algoritmo Skip-Gram para a criação de *embeddings* e apesar de obter resultados satisfatórios, suas melhores configurações não são suficientes para fazê-lo ser usado de forma completamente automatizada.

Dentro dos estudos de simplificação automatizada para o português brasileiro, encontra-se o trabalho realizado por Aluísio et al. (2008). Esse trabalho foca em criar recursos básicos para o desenvolvimento de sistemas de simplificação textual. Chamado de PorSimples, o projeto não limita-se apenas à simplificação léxica e produz, também, o primeiro manual de simplificação sintática para o português brasileiro.

Na tentativa de desenvolver um sistema de simplificação léxica a partir de limitados recursos linguísticos e de forma não supervisionada, Glavaš e Štajner (2015) apresentaram a solução LIGHT-S. Neste sistema, só é possível realizar simplificação palavra-a-palavra e necessita apenas de um grande *corpus* de linguagem comum. O sistema faz uso do algoritmo GloVe para a criação dos *word embeddings* e identifica alguns parâmetros arbitrários para a seleção da palavra simplificada. Esta abordagem, segundo o autores, diminui uma das barreiras que a simplificação léxica encontra, que é a necessidade de notações simplificadas, como o Simple Wikipedia, e que ainda não existe para muitas línguas.

# 3 MATERIAIS E MÉTODOS

Serão apresentados neste capítulo os materiais e métodos que foram utilizados para as averiguações deste trabalho. Primeiramente, serão apresentados os materiais e subsequentemente os métodos que os utilizam.

## 3.1 MATERIAIS

Os materiais utilizados na realização deste trabalho foram:

- Bases de *embeddings* criadas utilizando diferentes algoritmos: estas bases foram treinadas por Hartmann et al. (2017) e estão sendo utilizados em sete diferentes algoritmos: Word2vec Skip-Gram e CBOW, GloVe, Wang2vec Skip-Gram e CBOW e fastText Skip-Gram e CBOW. Para cada algoritmo são disponibilizados vetores com 50, 100, 300, 600 e 1000 dimensões. Para este trabalho foram utilizadas apenas as bases de 50 dimensões. Os *corpora* utilizados para o treinamento consistem 17 *corpus* diferentes com cerca de 1,4 bilhões de tokens advindos de múltiplos gêneros textuais;
- Linguagem de Programação Python[1] na versão 3.6.1: linguagem interpretada, dinâmica e de código aberto;
- Biblioteca Tensor Flow[2] para Python na versão 1.0: biblioteca de código aberto desenvolvida pela equipe de Aprendizado de Máquina do Google para computação numérica e aplicada à RNAs profundas. Esta biblioteca utiliza-se de diagramas de fluxos na forma de grafos direcionados para abstrair os processos, sendo os nós as operações matemáticas e as arestas os dados dispostos em matrizes multidimensionais, chamadas de tensores;
- Biblioteca NumPy[3]: biblioteca escrita em Python, de código aberto e que permite

[1] https://www.python.org/
[2] https://www.tensorflow.org/
[3] http://www.numpy.org/

manipulação de matrizes e diversas funções matemáticas em alto nível;

- Biblioteca Keras[4] 2.0.6: biblioteca de alto nível para desenvolvimento de redes neurais em Python. Esta biblioteca funciona também sobre a biblioteca do Tensor Flow como *wrapper* e tem como objetivo facilitar a realização de experimentos;
- Dicionário psicolinguístico para português brasileiro (PB): este dicionário é resultado do trabalho realizado por Salles et al. (2017) e disponibiliza dados de concretude, imaginabilidade, familiaridade e idade de aquisição das palavra através de um *corpus* para 26.874 palavras do português brasileiro. Com o âmbito de tornar claro o conteúdo deste dicionário, a Tabela 3 mostra as palavras nas posições extremas do dicionário para cada um dos quatro atributos utilizados. Lê-se o menor valor como o menos concreto, menos imaginável, menos familiar ou menor idade de aquisição da palavra. Para o maior valor, deve-se seguir a mesma ideia. As palavras contidas neste dicionário estão em sua forma lematizada. Para o uso neste trabalho, os valores foram alterados utilizando a normalização pelo desvio padrão (RASCHKA, 2015);

**Tabela 3 – Amostras das extremidades do conteúdo do dicionário psicolinguístico.**

| | menor valor | maior valor |
|---|---|---|
| concretude | transcedente | pescoço |
| imaginabilidade | quadrângulo | gosto |
| familiaridade | discriminatório | bebê |
| idade de aquisição | comer | suplência |

**Fonte: Autoria própria.**

- Dicionário de frequência do Corpus Brasileiro (CEPRIL, LAEL, PUCSP, Fapesp)[5]: este dicionário faz parte do projeto Corpus Brasileiro e foi desenvolvido pelo Grupo de Estudos de Linguística de Corpus (GELC) do Centro de Pesquisas, Recursos e Informação de Linguagem (CEPRIL) e pelo Programa de Pós-Graduação em Linguística Aplicada (LAEL) da Pontifícia Universidade Católica de São Paulo (PUC-SP);
- Natural Language Toolkit[6] (NLTK): plataforma de código aberto construída para desenvolvimento, em Python, de trabalhos utilizando dados de língua natural. Possui uma grande gama de *corpora* e recursos como classificador, tokenizador, lematizador, *tagger*, *parser* e *stemmer*;
- Biblioteca gensim: desenvolvida por Řehůřek e Sojka (2010): esta biblioteca Python de código aberto é voltada à modelagem semântica não supervisionada de textos. Oferece diversos recursos relacionados à *word embeddings*;

[4] https://keras.io/
[5] http://corpusbrasileiro.pucsp.br
[6] http://www.nltk.org/

- Textos do Wikipédia: foram extraídos seis textos de um pequeno *corpus* de artigos em destaque (na seção de capa). O *corpus* é composto por 12 diferentes domínios (esporte, ecônomia, cultura, entre outros) sendo utilizado para compor a base de anotação;
- Lematizador[7]: ferramenta que realiza a etiquetação e lematização de palavras do português brasileiro desenvolvida pelo NILC.

## 3.2 MÉTODOS

Para a realização dos objetivos propostos neste trabalho, dividiu-se a estrutura geral em dois simplificadores, conforme a Figura 9. O primeiro simplificador, Simplificador 1 (S1), foi responsável por realizar o pré-processamento do texto, eliminando palavras de baixo valor semântico para a simplificação, como artigos e preposições, selecionar palavras do espaço de *embeddings*, realizar filtragem utilizando o dicionário de frequência e por fim gerar uma lista contendo as palavras originais e as palavras candidatas. O segundo simplificador, Simplificador 2 (S2), é uma rede neural treinada a partir de uma base anotada com exemplos positivos e negativos do simplificador 1 que verificou apenas as boas simplificações. Isso é feito com os *embeddings* da palavra original e da candidata à simplificação, além de dados sobre essas palavras extraídos do dicionário psicolinguístico. Os métodos utilizados para a realização deste trabalho são descritos em quatro etapas:

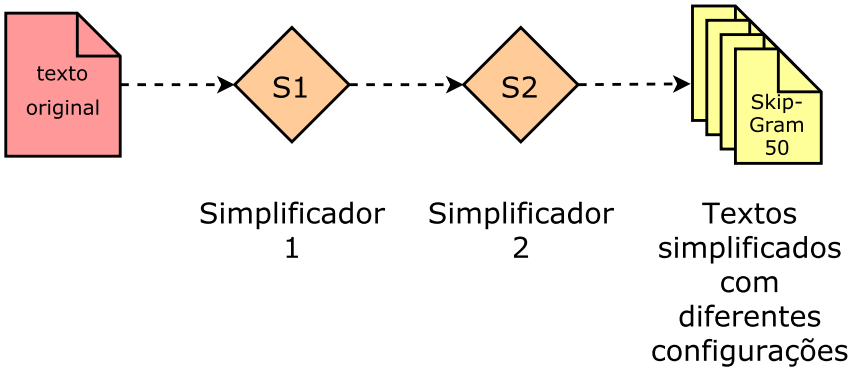



**Figura 9 – Visão geral do processo de simplificação.**

**Fonte: Autoria própria.**

**Etapa 1:** Foi desenvolvido nesta etapa o simplificador léxico 1. Este *script* foi desenvolvido

[7]http://www.nilc.icmc.usp.br/nilc/index.php/tools-and-resources

em Python e é responsável por:

1.1 Tokenizar o texto de entrada e remover *stop-words*, conforme a Tabela 4. Utilizam-se os recursos da biblioteca NLTK para a realização desta etapa;

**Tabela 4 – Excerto ilustrativo das etapas do simplificador 1.**

| Etapa | Excerto |
|---|---|
| entrada | onde nota-se uma fixação pelo olhar dúbio de "cigana oblíqua e dissimulada" de Capitu |
| tokenização | [onde], [nota-se], [uma], [fixação], [pelo], [olhar], [dúbio], [de], ["], [cigana], [oblíqua], [e], [dissimulada], ["], [de], [Capitu] |
| remoção de *stop-words* | [onde], [nota-se], [fixação], [olhar], [dúbio], [cigana], [oblíqua], [dissimulada], [capitu] |

**Fonte: Autoria própria.**

1.2 Desconsiderar palavras originais que não estão na base de *embeddings*;

1.3 Desconsiderar palavras originais com frequência igual ou acima de 3.000 ocorrências, como exemplificado na Tabela 5. Essa heurística foi aplicada para evitar que as 20.000 palavras mais frequentes sejam simplificadas. De acordo com Cervatiuc (2008), o número de palavras flexionadas que um estudante nativo do idioma inglês conhece varia de 13.500 e 20.000. Considerando que os verbos da língua portuguesa apresentam uma variedade de flexão maior, estima-se que um corte ainda mais alto poderia ser feito.

**Tabela 5 – Frequência das palavras originais.**

| Palavra original | Frequência | Situação |
|---|---|---|
| onde | 567.182 | eliminada |
| nota-se | 15.469 | eliminada |
| fixação | 21.898 | eliminada |
| olhar | 49.626 | eliminada |
| dúbio | 434 | mantida |
| cigana | 771 | mantida |
| oblíqua | 1.209 | mantida |
| dissimulada | 562 | mantida |
| capitu | 939 | mantida |

**Fonte: Autoria própria.**

1.4 Buscar as 20 palavras candidatas mais similares a cada palavra original selecionada. Para isso, é utilizada a base de *embeddings* específica (CBOW, Skip-Gram ou GloVe). Utiliza-se a função disponibilizada pela biblioteca gensim "most_similar_cosmul". A Tabela 6 apresenta uma busca similar, onde altera-se apenas a quantidade de palavras consultadas para quatro;

**Tabela 6 – Palavra original com suas quatro palavras candidatas.**

| Original | Candidatas |
|---|---|
| dúbio | ambíguo, rebuscado, insólito, discreto |
| cigana | judaica, mestiça, africana, hispânica |
| oblíqua | achatada, rugosa, cónica, asp000 |
| dissimulada | supersticiosa, sórdida, direcu, cristalizada |
| capitu | inesita, mariazinha, iaiá, zadig |

**Fonte: Autoria própria.**

1.5 Eliminar palavras candidatas de baixa frequência. As palavras candidatas são filtradas de modo que reste apenas aquelas que são mais frequentes do que a original. Além disso, a candidata deve ter pelo menos 500 ocorrências a mais que a original no dicionário de frequências. Essa heurística foi definida empiricamente com o propósito de evitar substituições de palavras originais por palavras candidatas mais difíceis (Tabela 7);

**Tabela 7 – Filtragem das palavras candidatas pela frequência.**

| Original | Frequência | Candidata | Frequência | Situação |
|---|---|---|---|---|
| dúbio | 434 | ambíguo | 2018 | mantida |
| | | rebuscado | 178 | eliminada |
| | | insólito | 767 | eliminada |
| | | discreto | 5718 | mantida |
| cigana | 771 | judaica | 4873 | mantida |
| | | mestiça | 1276 | mantida |
| | | africana | 7851 | mantida |
| | | hispânica | 963 | eliminada |
| oblíqua | 1209 | achatada | 559 | eliminada |
| | | rugosa | 1030 | eliminada |
| | | cónica | 5 | eliminada |
| | | asp000 | 0 | eliminada |
| dissimulada | 562 | supersticiosa | 131 | eliminada |
| | | sórdida | 401 | eliminada |
| | | direcu | 0 | eliminada |
| | | cristalizada | 731 | eliminada |
| capitu | 939 | inesita | 30 | eliminada |
| | | mariazinha | 168 | eliminada |
| | | iaiá | 182 | eliminada |
| | | zadig | 12 | eliminada |

**Fonte: Autoria própria.**

1.6 Lematizar palavras originais e candidatas para realizar consulta na base de dados psicolinguística, conforme exemplo na Tabela 8. Para esta etapa utiliza-se o lematizador do NILC;

1.7 Utilizar versão não flexionada da palavra original e de cada candidata para consultar as

**Tabela 8 – Pares original e candidata com suas versões lematizadas.**

| Original | Candidatas | Original Lematizada | Candidata Lematizada |
|---|---|---|---|
| dúbio | ambíguo | dúbio | ambíguo |
| dúbio | discreto | dúbio | discreto |
| cigana | judaica | cigano | cigano |
| cigana | mestiça | cigano | mestiço |
| cigana | africana | cigano | africano |

**Fonte: Autoria própria.**

informações psicolinguísticas no dicionário;

1.8 Calcular a distância Manhattan (REINELT, 1991) entre o vetor da palavra original e da palavra candidata;

1.9 Buscar *embeddings* das palavras anterior e posterior à palavra original. Utiliza-se a ordem do texto após já removidas as *stop-words*;

1.10 Gerar arquivo com os pares de original e candidata, informação psicolinguística das duas palavras, distância Manhattan e os vetores das palavras anterior e posterior (Tabela 9). Um exemplo completo da saída do S1 pode ser visto no Apêndice B.

**Tabela 9 – Atributos de saída do Simplificador 1.**

| Descrição | Quantidade |
|---|---|
| *embedding* da palavra original | 50 |
| *embedding* da palavra candidata | 50 |
| informações psicolinguísticas da palavra original | 4 |
| informações psicolinguísticas da palavra candidata | 4 |
| *embedding* da palavra vizinha anterior | 50 |
| *embedding* da palavra vizinha posterior | 50 |
| distância Manhattan | 1 |
| **Total** | **209** |

**Fonte: Autoria própria.**

**Etapa 2:** nesta etapa foi construída uma base anotada de exemplos de palavras simplificadas. A anotação dos exemplos foi realizada considerando o contexto ao qual a palavra estava inserido, sendo este verificado por um estudante de graduação nativo da língua portuguesa. Para a geração dos casos analisados, utilizou-se a base de textos do Wikipédia e identificou-se cerca de 500 casos para cada algoritmo de WE em estudo (totalizando 1.500 anotações em três conjuntos de treinamento), onde aproximadamente 50% destes são casos positivos e os outros negativos, conforme pode ser conferido na Tabela 10. Esta etapa se faz necessária para que se pudesse fazer o estudo do classificador do Simplificador 2;

**Etapa 3:** foi desenvolvido, nesta etapa, um segundo simplificador léxico (S2). Ele consiste em uma RNA, desenvolvida a partir do Keras executado sobre o Tensor Flow, e da linguagem

**Tabela 10 – Composição do *corpus* na criação da base anotada de simplificação.**

| | CBOW | | | Skig-Gram | | | GloVe | | |
|---|---|---|---|---|---|---|---|---|---|
| Título do artigo | - | + | total | - | + | total | - | + | total |
| Beyonce | 20 | 23 | 43 | 20 | 22 | 42 | 21 | 20 | 41 |
| Brasil | 80 | 88 | 168 | 84 | 78 | 162 | 90 | 72 | 162 |
| Bohemian Rhapsody | 29 | 16 | 45 | 14 | 12 | 26 | 16 | 14 | 30 |
| Dom Casmurro | 0 | 0 | 0 | 24 | 3 | 27 | 46 | 37 | 83 |
| Ginástica | 6 | 5 | 11 | 5 | 9 | 14 | 0 | 3 | 3 |
| Maneirismo | 78 | 98 | 176 | 74 | 89 | 163 | 63 | 72 | 135 |
| Trachylepis Atlantica | 16 | 15 | 31 | 10 | 11 | 21 | 10 | 13 | 23 |
| Wicca | 21 | 30 | 51 | 23 | 27 | 50 | 16 | 20 | 36 |
| **Totais** | 250 | 275 | 525 | 254 | 251 | 505 | 262 | 251 | 513 |

**Fonte: Autoria própria.**

Python. A RNA realiza a aprovação da simplificação sugerida pelo S1. Os dados de entrada desta rede são dados de acordo com as informações apresentadas no item (j). A camada de saída possui apenas um neurônio que deverá aprovar ou não a simplificação. O treinamento dessa RNA se deu de forma supervisionada com a base gerada na etapa 2;

**Etapa 4:** a última etapa consistiu na comparação extrínseca dos diferentes algoritmos de *word embedding*, a partir dos resultados dos dois simplificadores. Os parâmetros utilizados para averiguar a qualidade da simplificação foram: (a) as características semânticas, se a simplificação mantém a semântica da palavra original; (b) simplicidade, se a palavra sugerida realmente é mais simples que a original; e (c) gramaticalidade da palavra, se a palavra candidata pode realmente substituir a original sem quebrar a estruturação sintática da sentença.

Neste capítulo foram apresentados os materiais (Seção 3.1) e métodos (3.2) para a realização dos objetivos propostos por este trabalho.

# 4 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados os resultados obtidos para os dois simplificadores propostos (S1 e S2) para os diferentes algoritmos de *word embedding* em estudo.

## 4.1 SIMPLIFICADOR 1

O primeiro simplificador é capaz de identificar boas sugestões para realizar a simplificação, diminuindo bastante o escopo de seleção de candidatas. No exemplo da Tabela 11, pode-se verificar que dentre as oito possíveis palavras sugeridas para a palavra "imemorial", três poderiam simplificar a palavra original dado o contexto do qual a palavra foi extraída: "o desenho tem uma origem imemorial [...]". Observa-se que a anotação pode ser subjetiva e que o crivo do anotador foi usado para resolver subjetividade.

**Tabela 11 – Candidatas à simplificação da palavra "imemorial".**

| Candidata | Anotação |
|---|---|
| indefinida | + |
| budista | - |
| desconhecida | + |
| imutável | - |
| incerta | + |
| efêmera | - |
| desconhecido | - |
| infinita | - |

**Fonte: Autoria própria.**

Assim como já estudado por Hartmann et al. (2017) e Souza (2016), os algoritmos de *word embedding* são capazes de capturar similaridades semânticas e sintáticas, e para a tarefa de simplificação léxica é importante que a palavra candidata atenda a estes dois requisitos. Nessa tarefa o Simplificador 1, além de usufruir desta característica dos *embeddings*, ainda exclui as

palavras cuja lematização da candidata é igual à da original, para que não se tenha flexões de uma mesma palavra como sugestão. A Tabela 12 apresenta um exemplo onde todas as palavras candidatas que resultaram do S1 para a palavra "retomado" são sintaticamente corretas, mas não são exemplos positivos de simplificação, apesar de armazenar algum nível de similaridade semântica à palavra original.

**Tabela 12 – Candidatas à simplificação da palavra "retomado".**

| Candidata |
|---|
| mantido |
| analisado |
| executado |
| implementado |
| discutido |
| iniciado |

**Fonte: Autoria própria.**

Outra observação que pode ser feita é quanto à similaridade semântica das palavras recuperadas dos *embeddings*, que muitas vezes agregam um valor de similaridade muito boa, como é o caso da relação capital–país, mas que não é necessariamente uma versão simplificada. Para estes casos não foi identificada nenhuma forma de seleção, logo, como apresentado na Tabela 13, sugestões errôneas são realizadas.

**Tabela 13 – Candidatas à simplificação da palavra "luteranismo".**

| Candidata |
|---|
| protestantismo |
| cristianismo |
| islã |
| judaísmo |
| marxismo |
| catolicismo |
| iluminismo |
| anarquismo |

**Fonte: Autoria própria.**

Um cenário onde a utilização de *word embeddings* se mostra promissor é na identificação de hiperônimos. A relação de hiperonímia entre palavras acontece quando uma palavra expressa de forma mais genérica uma ou várias outras palavras. Como exemplo, "animal" é hiperônimo de "cachorro" e "elefante". Na atividade de simplificação é muito comum utilizar dicionários de sinônimos para identificar as palavras menos complexas, esses dicionários não trazem relações de hiperonímia, logo, a utilização de *word embeddings* se mostra promissora para a seleção de candidatas com essa característica. Na Tabela 14 é possível

observar que dentre as candidatas recuperadas para o artigo “Dom Casmurro” com o algoritmo Skip-Gram, para a palavra “antologia”, três são hiperônimos.

**Tabela 14 – Candidatas à simplificação da palavra “antologia”.**

| Candidata | Relação |
|---|---|
| ópera | - |
| poesia | hiperonímia |
| ilustração | - |
| coleção | hiperonímia |
| peça | - |
| obra | hiperonímia |
| biografia | - |

**Fonte: Autoria própria.**

O número de sugestões geradas pelo Simplificador 1 para cada algoritmo e cada artigo do Wikipédia utilizado é apresentado na Tabela 15. Vale ressaltar que o comportamento do lematizador é baseado em contexto e pode, em situações ambíguas, identificar a versão flexionada diferente para uma mesma palavra, o que pode influenciar no número de pares sugeridos por cada algoritmo.

**Tabela 15 – Quantitade de sugestões elencadas por algoritmo e texto de entrada.**

| Título do artigo | CBOW | Skip-Gram | GloVe |
|---|---|---|---|
| Beyoncé | 1090 | 883 | 1483 |
| Bohemian Rhapsody | 651 | 432 | 710 |
| Brasil | 1586 | 1367 | 1612 |
| Dom Casmurro | 1457 | 1101 | 1449 |
| Ginástica | 1211 | 982 | 998 |
| Maneirismo | 2344 | 1920 | 1999 |
| Trachylepis Atlantica | 562 | 407 | 684 |
| Wicca | 2062 | 1505 | 1931 |
| Total | 10.996 | 8.597 | 10.866 |

**Fonte: Autoria própria.**

## 4.2 SIMPLIFICADOR 2

Para avaliar o segundo classificador, primeiramente foi feita uma calibração dos parâmetros da RNA com base no algoritmo Skip-Gram (Seção 4.2.1). Em seguida, foi feita

uma comparação entre Skip-Gram, CBOW e GloVe (Seção 4.2.2). Na Seção 4.2.3 é realizada uma comparação a partir de bases modificadas, incluindo também os algoritmos Wang2vec e fastText.

Deve-se observar aqui um problema ocorrido durante os testes iniciais, que ocasionou valores de acerto abaixo de 50%. As bases anotadas foram construídas buscando anotar um exemplo negativo e um positivo para cada palavra original. Antes de realizar os treinamentos, a base foi sorteada de forma que as amostras não ficassem em nenhuma ordem específica. Esta métrica de organização fez com que os pares ficassem separados, ou seja, uma amostra positiva poderia estar na base de treinamento, enquanto a amostra negativa daquela mesma palavra original poderia estar na base de teste. Como ambas as amostras compartilham vários atributos (*embedding* da palavra original, informações psicolinguísticas da palavra original e *embeddings* das palavras vizinhas), a rede não estava conseguindo aprender. A solução utilizada para este problema foi manter os pares na mesma base, seja de treino ou de teste. Com essa modificação os valores de acurácia melhoraram.

### 4.2.1 Calibração da RNA para o algoritmo Skip-Gram

O processo de calibração da RNA foi realizado com o objetivo de obter um melhor modelo inicial para que fosse possível realizar a comparação com as bases geradas pelos diferentes algoritmos de *embeddings*. A motivação ao uso do Skip-Gram foi devido este ser mais difundido. Foram testados nove modelos diferentes, conforme a Tabela 16, onde alterou-se quantidade de camadas e neurônios, tamanho do lote de treinamento, quantidade de época e valores para algoritmos de regularização.

Todos os experimentos foram realizados utilizando a função de ativação ReLU. Para os parâmetros não apresentados, foram utilizados os valores padrão utilizados pelo Keras na versão 2.0.6. As camadas são todas completas, ou seja, os neurônios de uma camada se conecta a todos os neurônios da camada seguinte. A camada de saída é composta sempre por um único neurônio sigmoide logístico (que aprova ou reprova a candidata a simplificação).

Durante os experimentos, observou-se muita oscilação na acurácia. Para minimizar este efeito, cada modelo foi treinado sete vezes, com base em diferentes sementes aleatórias, descartando-se o melhor e o pior valor de acurácia. Para os cinco valores restantes, calculou-se então a média. Os valores de cada treinamento utilizados para o cálculo da média podem

**Tabela 16 – Arquiteturas de calibração.**

| Arquitetura | Camadas | Quantitade de épocas | Tamanho de lote |
|---|---|---|---|
| 1 | Camada 1: 100 neurônios | 5000 | 75 |
| 2 | Camada 1: 100 neurônios, regularização L2 de 0,05 | 5000 | 75 |
| 3 | Camada 1: 100 neurônios, regularização *dropout* de 0,05 | 5000 | 75 |
| 4 | Camada 1: 100 neurônios, regularização L2 de 0,05 | 1000 | 75 |
| 5 | Camada 1: 300 neurônios, regularização *dropout* de 0,03<br><br>Camada 2: 500 neurônios, regularização L2 de 0,03 | 1000 | 100 |
| 6 | Camada 1: 500 neurônios, regularização *dropout* de 0,05<br><br>Camada 2: 700 neurônios, regularização L2 de 0,03, regularização *dropout* de 0,05<br><br>Camada 3: 300 neurônios, regularização *dropout* de 0,05 | 200 | 20 |
| 7 | Camada 1: 500 neurônios, regularização *dropout* de 0,05<br><br>Camada 2: 700 neurônios, regularização L2 de 0,03, regularização *dropout* de 0,05<br><br>Camada 3: 300 neurônios, regularização *dropout* de 0,05 | 1000 | 20 |
| 8 | Camada 1: 200 neurônios, regularização L2 de 0,02, regularização *dropout* de 0,02 | 1000 | 40 |
| 9 | Camada 1: 100 neurônios, regularização L2 de 0,03, regularização *dropout* de 0,01 | 1000 | 75 |

**Fonte: Autoria própria.**

ser visto no Apêndice A. Um objetivo secundário dos testes realizados foi de investigar a melhor técnica de regularização. Devido a isso, testou-se diferentes valores para L2 e *dropout*. Adicionalmente, também variou-se o tamanho do lote de aprendizado e o número de épocas, buscando-se a melhor configuração para convergência eficiente.

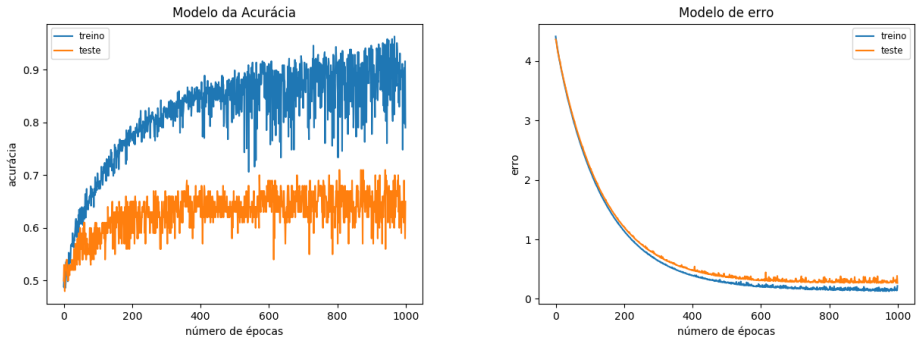
Os treinamentos foram executados em duas versões diferentes: Acurácia Média A, com 109 atributos, sem palavras vizinhas; Acurácia Média B, com 209 atributos contando os *embeddings* das palavras vizinhas. A base é dividade em 100 amostras para teste e 405 para treinamento. Os valores são apresentados na Tabela 17.

**Tabela 17 – Acurácias médias das arquiteturas de RNA para a base do Skip-Gram.**

| Arquitetura | Acurácia A (%) | Acurácia B (%) |
|---|---|---|
| 1 | 52,8 | 46,4 |
| 2 | 56,4 | 60,4 |
| 3 | 59,6 | 57,6 |
| 4 | 59,6 | 59,2 |
| 5 | 60,6 | 55,4 |
| 6 | 60,6 | 57,8 |
| 7 | 59,2 | 53,8 |
| 8 | 61,0 | 56,4 |
| 9 | **64,2** | **64,6** |

**Fonte: Autoria própria.**

A Arquitetura 9 obteve os melhores valores de acurácia, tanto para o cenário onde utiliza-se os vetores das palavras vizinhas quanto para o cenário onde esses não são utilizados. São apresentados na Figura 10 um excerto de treinamento do melhor resultado obtido, onde pode-se ver o comportamento do aprendizado ao longo das épocas treinadas.




(a) acurácia (b) perda por erros quadráticos

**Figura 10 – Acurácia e perda para Arquitetura 9 para base com palavras vizinhas.**

**Fonte: Autoria própria**

### 4.2.2 Treinamento para diversas bases

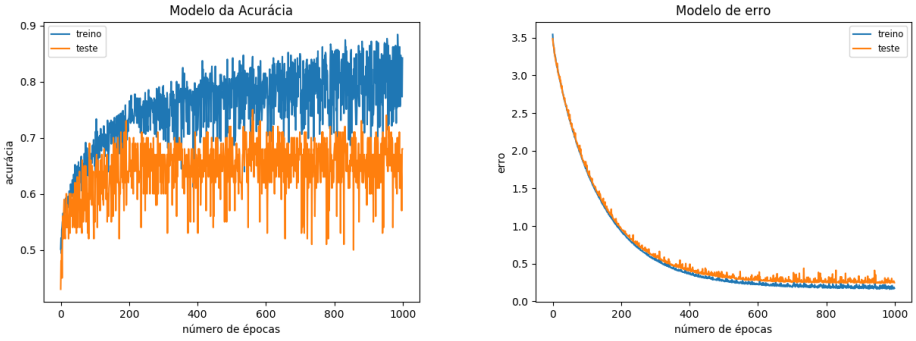
A partir dos resultados da calibração da RNA, selecionaram-se três modelos para cada um dos dois cenários (com vizinhos e sem vizinhos). Os dois melhores resultados e o pior. A partir das três arquiteturas referentes a esses valores, realizou-se então o treinamento utilizando as bases do CBOW e GloVe.

Na Tabela 18 é possível verificar, para o cenário onde as palavras vizinhas não são utilizadas, que o Skip-Gram tem o melhor e o pior valor de acurácia. Na Figura 11 é apresentado um exemplo do comportamento da acurácia e perda ao longo das épocas de treinamento o melhor resultado.

**Tabela 18 – Acurácia Média A para as Arquiteturas 1, 8 e 9 nas diferentes bases.**

| Arquitetura | Skip-Gram (%) | CBOW (%) | GloVe (%) |
|---|---|---|---|
| 1 | 52,8 | 54,0 | 56,0 |
| 8 | 61,0 | 57,8 | 56,0 |
| 9 | **64,2** | 59,0 | 62,8 |

**Fonte: Autoria própria.**




(a) acurácia (b) perda por erros quadráticos

**Figura 11 – Acurácia e perda para Arquitetura 9 para base sem palavras vizinhas.**

**Fonte: Autoria própria**

Para o cenário onde as palavras vizinhas são consideradas, verifica-se na Tabela 19 que o Skip-Gram novamente tem o melhor e o pior resultado.

Analisando todo os cenários, o melhor resultado obtido continua sendo com a Arquitetura 9 para o algoritmo Skip-Gram, onde obteve-se acurácia de 64,6%.

**Tabela 19 – Acurácia Média B para as Arquiteturas 1, 2 e 9 nas diferentes bases.**

| Arquitetura | Skip-Gram (%) | CBOW (%) | GloVe (%) |
|---|---|---|---|
| 1 | 46,4 | 53,2 | 56,0 |
| 2 | 60,4 | 58,2 | 60,4 |
| 9 | **64,6** | 58,8 | 57,6 |

**Fonte: Autoria própria.**

### 4.2.3 Avaliação de arquitetura a partir de variação da base do Skip-Gram

No trabalho apresentado por Hartmann et al. (2017), pode-se verificar que os algoritmos Wang2vec e fastText obtiveram bons resultados quando aplicados às tarefas específicas de PLN. Com essa motivação, porém com a ressalva de não ter disponível uma base de dados desenvolvida puramente a partir desses algoritmos, propõe-se nesta seção a alteração da base original gerada a partir do algoritmo Skip-Gram. Nesta alteração, elimina-se os *embeddings* das palavras vizinhas, e substituem-se os *embeddings* das palavras original e candidata pelos vetores gerados pelos algoritmos Wang2Vec e fastText. Para efeito de comparação, também faz-se o mesmo procedimento para as bases do GloVe e CBOW. As bases criadas a partir da modificação da base gerada com o Skip-Gram serão identificadas com o símbolo "+" ao final. Ao fim da substituição tem-se então seis novas bases mais a base original do Skip-Gram, sendo elas: Wang2Vec Skip-Gram+, Wang2Vec CBOW+, fastText Skip-Gram+, fastText CBOW+, CBOW+ e GloVe+.

Para realizar o teste nessas novas bases, escolheu-se uma arquitetura com resultados médios no processo de calibração. O objetivo desta escolha é evitar uma arquitetura que seja muito específica ao Skip-Gram e que possa apontar bons resultados para os outros algoritmos também.

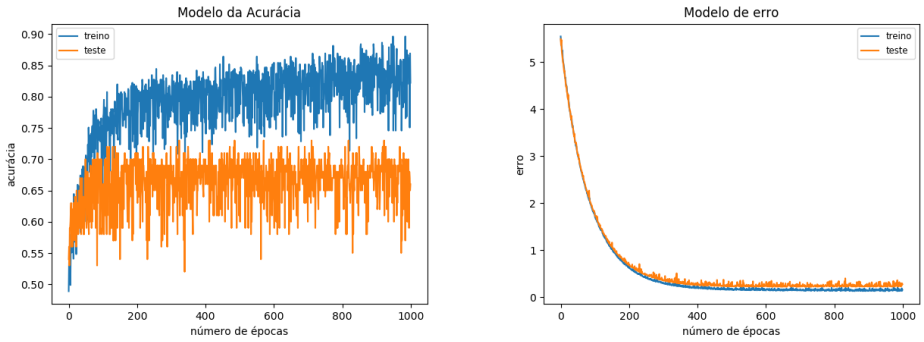
Na Tabela 20 pode-se verificar as acurácias médias obtidas para cada nova base. Cada média é obtida da mesma forma da Seção 4.2.2 e equivalem à Acurácia A (sem as palavras vizinhas).

Observa-se nos novos resultados obtidos que a base alterada com os *embeddings* do Wang2Vec CBOW obteve o melhor resultado entre as bases alteradas e o melhor resultado geral dentre todas as análises deste trabalho. Na Figura 12, pode-se verificar um exemplo do

**Tabela 20 – Acurácia Média A para a Arquitetura 4 nas bases modificadas.**

| Base | Acurácia média (%) |
|---|---|
| Skip-Gram (original) | 59,6 |
| CBOW+ | 60,8 |
| GloVe+ | 52,0 |
| Wang2Vec Skip-Gram+ | 60,2 |
| Wang2Vec CBOW+ | **66,4** |
| fastText Skip-Gram+ | 65,8 |
| fastText CBOW+ | 52,0 |

**Fonte: Autoria própria.**




(a) acurácia (b) perda por erros quadráticos

**Figura 12 – Acurácia e perda para Arquitetura 4 para base sem palavras vizinhas.**

**Fonte: Autoria própria**

comportamento da acurácia e perda deste modelo ao longo das épocas de treinamento.

## 4.3 DISCUSSÃO DOS RESULTADOS

Os resultados obtidos deste trabalho podem ser analisados de duas formas diferentes: da perspectiva do S1 e do S2.

Quando observados os resultados do S1, pode-se verificar que o grande uso potencial dele é a identificação de palavras candidatas já que os *embeddings* capturam diversos níveis de similaridade diferentes entre as palavras. Somando-se as regras de seleção, com base em um dicionário de frequência, é possível diminuir consideravelmente o escopo de palavras sugeridas. Nesse âmbito, apesar do S1 não funcionar como uma ferramenta de simplificação 100% automatizada, este pode ser utilizado como ferramenta de auxílio para leitores proficientes da

língua na simplificação de seus textos.

Outra forma de observação dos resultados que pode ser feita é utilizando o S2. A utilização da base do Skip-Gram para a calibração, pode ter desenvolvido uma tendência maior ao modelo, já que, nos resultados para as bases sem alteração, este obteve o melhor e pior valor. Esse argumento é sustentado pelo fato de muitos dos cenários de treinamento das arquiteturas para a base do GloVe não apresentarem convergência, isto é, sua acurácia se manteve constante desde o início do treinamento (em torno de 56%), embora a função de perda continuou a decrescer. Observa-se também que em todos os melhores modelos apresentados, utilizou-se a normalização L2, algumas vezes associada com *dropout*. Embora os resultados sejam modestos, percebe-se uma tendência ao aprendizado.

Na comparação extrínseca dos algoritmos, obteve-se, nos experimentos realizados, que o Wang2vec CBOW atinge melhores valores de acurácia, chegando a 66,4% mesmo com uma base não desenvolvida a partir das simplificações sugeridas com *embeddings* do mesmo.

# 5 CONSIDERAÇÕES FINAIS

## 5.1 CONCLUSÃO

Neste trabalho foi apresentada uma abordagem para a simplificação léxica de textos utilizando a tecnologia dos *word embeddings* para a etapa de seleção de candidatas, bem como realizada uma comparação extrínseca dos principais algoritmos de geração desses vetores, a partir do treinamento de uma RNA em conjuntos de treinamentos especificamente criados para a tarefa. Este estudo consistiu em uma abordagem inicial para o português brasileiro e ainda elenca diversos pontos de melhoria a serem explorados.

Construiu-se, durante o desenvolvimento deste trabalho, três bases balanceadas com 500 amostras de simplificações capturadas da estrutura do S1. Essas bases tornaram possível o treinamento de diferentes arquiteturas de RNA para a classificação das simplificações.

As principais contribuições são:

(a) Um primeiro simplificador para apoiar leitores proficientes na adaptação de seus textos para o público alvo em foco. Essa funcionalidade é uma alternativa ao uso dos dicionários rígidos de sinônimos, assim como o software Facilita (WATANABE et al., 2010);

(b) Um segundo simplificador, que os baixos valores de acurácia obtidos nos melhores modelos desestimulam o uso 100% automatizado, porém indicam que existe uma tendência ao aprendizado automático através dos modelos, já que um classificador aleatório deve ter acurácia de 50% no problema em estudo. Como não havia modelo de RNA base para iniciar os estudos, acredita-se que se pode melhorar bastante os resultados com outros conjuntos de parâmetros. Adicionalmente, os modelos se beneficiariam em um aumento no conjunto de treinamento, já que foram anotados apenas três conjuntos (para Skip-Gram, GloVe e CBOW) com 500 instâncias cada e uma adaptação no primeiro foi feita para acomodar os modelos Wang2vec e fastText. A natureza do problema de simplificação léxica é muita vezes subjetiva. Em virtude dessa subjetividade e da

dimensionalidade do problema, é interessante que se amplie a base anotada para que obtenha-se melhores valores de acurácia dos modelos de RNA;

(c) Por fim, uma terceira contribuição do trabalho foi uma análise extrínseca dos modelos de *embeddings*, originalmente concebidos para capturar similaridade semântica e, neste trabalho, utilizados em uma tarefa diferenciada, a simplificação léxica. Durante a análise, foi feita uma comparação dos diversos modelos apresentados, sendo que nos experimentos realizados para tarefa específica de simplificação, foi verificado que o maior valor obtido foi do modelo Wang2vec CBOW, que chegou a 66,4% de acurácia. É interessante ressaltar que a RNA utilizada não foi otimizada para este modelo, e que, além disso, o conjunto de treinamento também não foi anotado com base em *embeddings* desse modelo.

## 5.2 TRABALHOS FUTUROS

Pode-se em trabalhos futuros aplicar diversas melhorias aos simplificadores aqui apresentados:

- O processo pode ser otimizado com eliminação de candidatas com o mesmo radical utilizando ferramentas de *stemming*: neste trabalho utilizou-se um lematizador. Esta melhoria identificaria melhor palavras candidatas com a mesma estruturação morfológica da palavra original. Isto evitaria sugestões do S1 para substituir uma determinada conjugação de um verbo por outra do mesmo verbo. Nota-se, que, em teoria, o modelo fastText tem condições de fornecer subsídios à rede para esta tarefa, uma vez que ele considera grafia das palavras para gerar *embeddings*, o que é conveniente nas conjugações verbais, cuja grafia é quase sempre próxima;
- Outra otimização seria durante a utilização do dicionário de frequências, quanto mais atual e mais abrangente este dicionário for, mais alinhado à realidade serão as palavras candidatas;
- Outra melhora é em relação aos cortes utilizados. O primeiro corte foi usado para eliminar palavras originais com frequência maior que 3.000, pois possivelmente já são simples o bastante. O segundo corte permitiu eliminar palavras candidatas que possuam pelo menos 500 ocorrências a mais do que a original. Pode-se realizar testes para otimizar estes valores;

- Um estudo que pode ser realizado é a verificação dos indicadores que fizeram o modelo Wang2vec CBOW obter os melhores resultados neste trabalho. Identificando características do algoritmo que, por ventura, beneficiem a atividade de simplificação léxica e contrapondo ao algoritmo CBOW clássico do Word2vec;
- A base é modesta dado o escopo do trabalho. Ela pode ser aumentada com mais anotação manual e melhorada especialmente para os modelos Wang2vec e fastText;
- A base poderia ser melhorada por meio de técnicas de aprendizado semi-supervisionado, a partir de trigramas de palavras extraídos de grandes *corpora*. Por exemplo, considerando-se um o primeiro trigrama ($w_1$, $w_2$, $w_3$) e um segundo ($w_1$, $w_4$, $w_3$) pode-se criar uma instância positiva na qual $w_2$ é substituída por $w_4$, já que as extremidades são as mesmas. Isso pode ser feito desde que a primeira palavra seja mais rara do que a segunda ou que, por exemplo, simplificador S1 inclua $w_4$ como candidata para $w_2$. As instâncias negativas poderiam ser criadas substituindo $w_2$ por uma palavra aleatória qualquer que não aparece no contexto de $w_1$, $w_3$. Uma RNA treinada sobre esta grande base, pode ser testada sobre a base pequena balizada por humano.

# REFERÊNCIAS


ALUÍSIO, S. M. et al. Towards brazilian portuguese automatic text simplification systems. In: **Proceedings of the Eighth ACM Symposium on Document Engineering**. New York, NY, EUA: ACM, 2008. p. 240–248.

BARONI, M.; DINU, G.; KRUSZEWSKI, G. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In: **Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Baltimore, Maryland, EUA: Association for Computational Linguistics, 2014. p. 238–247.

BOJANOWSKI, P. et al. Enriching word vectors with subword information. **CoRR**, abs/1607.04606, 2016.

CANDIDO JUNIOR, A. **Análise bidirecional da língua na simplificação sintática em textos do português voltado à acessibilidade digital**. 225 p. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação - USP, São Carlos, Brasil, 2013.

CARVALHO, A. P. de Leon F. de. **Redes Neurais Artificiais**. 2017. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/index.htm>. Acesso em: 24 de abril de 2017.

CERVATIUC, A. **ESL Vocabulary Acquisition: Target and Approach**. 2008. Disponível em: <http://iteslj.org/Articles/Cervatiuc-VocabularyAcquisition.html>. Acesso em: 10 de Outubro de 2017.

CHAUBARD, F.; MUNDRA, R.; SOCHER, R. **CS 224D: Deep Learning for NLP**. 2015. Disponível em: <http://cs224d.stanford.edu/lecture_notes/LectureNotes1.pdf>. Acesso em: 17 de abril de 2017.

CHOMSKY, N. Three models for the description of language. **IRE Transactions on Information Theory**, v. 2, p. 113–124, 1956.

CLARK, S. **Vector Space Models of Lexical Meaning**. 2014. Disponível em: <http://www.cl.cam.ac.uk/~sc609/>. Acesso em: 15 de abril de 2017.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, SciELO Brasil, v. 35, n. 1, p. 25–30, 2006.

FINOCCHIO, M. A. F. **Noções de Redes Neurais Artificiais**. 2014. Disponível em: <http://paginapessoal.utfpr.edu.br/mafinocchio/labsi-laboratorio-de-seguranca-e-iluminacao/redes-neurais-artificiais>. Acesso em: 06 de maio de 2017.

GLAVAŠ, G.; ŠTAJNER, S. Simplifying lexical simplification: Do we need simplified corpora. In: **Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing**. Beijing, China: Association for Computational Linguistics, 2015. p. 63–68.

GOLDBERG, Y. A primer on neural network models for natural language processing. **Journal of Artificial Intelligence Research**, v. 57, p. 345–420, 2016.

HARTMANN, N. et al. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. In: **Proceedings of the XI Brazilian Symposium in Information and Human Language Technology and Collocated Events**. Uberlândia, Minas Gerais, Brasil: Sociedade Brasileira de Computação, 2017. p. 122–132.

HAUGELAND, J. **Artificial Intelligence: The Very Idea**. Cambridge, MA, EUA: Massachusetts Institute of Technology, 1985.

HAYKIN, S. **Neural Networks and Learning Machines**. 3. ed. [S.l.]: Prentice Hall, 2009.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, EUA, p. 1735–1780, 1997.

HUTCHINS, J. The georgetown-ibm demonstration, 7th january 1954. In: **MT News International**. [S.l.: s.n.], 1994. p. 15–18.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 2. ed. Upper Saddle River, NJ, EUA: Prentice-Hall, Inc., 2009.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 2016. Disponível em: <https://web.stanford.edu/~jurafsky/slp3/>. Acesso em: 01 de abril de 2017.

KIBBLE, R. **Introduction to Natural Language Processing**. 2013.

KIM, Y.-S. et al. Simplescience: Lexical simplification of scientific terminology. In: **Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing**. Austin, Texas, EUA: Association for Computational Linguistics, 2016. p. 1066–1071.

LING, W. et al. Two/too simple adaptations of word2vec for syntax problems. In: **Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**. Denver, Colorado, EUA: Association for Computational Linguistics, 2015. p. 1299–1304.

MAX, A. Writing for language-impaired readers. In: **Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Text Processing**. Mexico City, México: Springer-Verlag, 2006. p. 567–570.

MAZUR, M. **A Step by Step Backpropagation Example**. 2015. Disponível em: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>. Acesso em: 10 de maio de 2017.

MCCAFFREY, J. D. **L2 Regularization and Back-Propagation**. 2017. Disponível em: <https://jamesmccaffrey.wordpress.com/2017/02/19/l2-regularization-and-back-propagation/>. Acesso em: 30 de Outubro de 2017.

MCCORMICK, C. **Word2Vec Tutorial Part 2 - Negative Sampling**. 2017. Disponível em: <http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>. Acesso em: 15 de maio de 2017.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013.

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: **Proceedings of the 26th International Conference on Neural Information Processing Systems**. Lake Tahoe, Nevada, EUA: Curran Associates Inc., 2013. p. 3111–3119.

PENNINGTON, J.; SOCHER, R.; MANNING, C. Glove: Global vectors for word representation. In: **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)**. Doha, Catar: Association for Computational Linguistics, 2014. p. 1532–1543.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. **GloVe: Global Vectors for Word Representation**. 2017. Disponível em: <https://nlp.stanford.edu/projects/glove/>. Acesso em: 19 de maio de 2017.

QU, Y. et al. Product-based neural networks for user response prediction. In: **2016 IEEE 16th International Conference on Data Mining (ICDM)**. [S.l.: s.n.], 2016. p. 1149–1154.

RASCHKA, S. **Python machine learning**. [S.l.]: Packt Publishing Ltd, 2015.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Seattle, WA, EUA: [s.n.], 2016. p. 779–788.

ŘEHŮŘEK, R.; SOJKA, P. Software Framework for Topic Modelling with Large Corpora. In: **Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks**. Valletta, Malta: ELRA, 2010. p. 45–50.

REINELT, G. Tsplib—a traveling salesman problem library. **ORSA journal on computing**, INFORMS, v. 3, n. 4, p. 376–384, 1991.

RICH, E.; KNIGHT, K. **Artificial Intelligence**. 2. ed. [S.l.]: McGraw-Hill Higher Education, 1990.

ROJAS, R. **Neural Networks: A Systematic Introduction**. Berlim, Alemanha: Springer Berlin Heidelberg, 1996.

RONG, X. word2vec parameter learning explained. **arXiv preprint arXiv:1411.2738**, 2014.

RUDER, S. **On word embeddings - Part 3: The secret ingredients of word2vec**. 2016. Disponível em: <http://sebastianruder.com/secret-word2vec/index.html>. Acesso em: 15 de maio de 2017.

RUSSELL, S. J.; NORVIG, P. **Inteligência Artificial: Tradução da 3a Edição**. 3. ed. Rio de Janeiro, RJ, Brasil: Elsevier Brasil, 2013.

SALLES, L. A. et al. A lightweight regression method to infer psycholinguistic properties for brazilian portuguese. 2017.

SCHALKOFF, R. **Artificial Intelligence: An Engineering Approach**. [S.l.]: McGraw-Hill, 1990. (Schaums Outline Series in Computers).

SHARDLOW, M. A survey of automated text simplification. **International Journal of Advanced Computer Science and Applications**, v. 4, n. 1, 2014.

SILVA, B. C. D. da et al. **Introdução ao Processamento das Línguas Naturais e Algumas Aplicações**. 2007.

SOUZA, S. **Estudo de Modelos de Word Embeddings**. Bacharelado — Universidade Tecnológica Federal do Paraná, Medianeira, Paraná, Brasil, 2016.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **J. Mach. Learn. Res.**, v. 15, p. 1929–1958, 2014.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2016. Disponível em: <http://incompleteideas.net/sutton/book/the-book-2nd.html>. Acesso em: 08 de maio de 2017.

TENSOR FLOW. **Vector Representations of Words**. 2017. Disponível em: <https://www.tensorflow.org/tutorials/word2vec>. Acesso em: 17 de abril de 2017.

URANO, K. **Lexical Simplification and Elaboration: Sentence Comprehension and Incidental Vocabulary Acquisition**. 79 p. Mestrado — The University of Hawaii, Honolulu, Hawaii, EUA, 2000.

WATANABE, W. M. et al. Adapting web content for low-literacy readers by using lexical elaboration and named entities labeling. In: **Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)**. New York, NY, USA: ACM, 2010. p. 8:1–8:9.

WINSTON, P. H. **Artificial Intelligence**. 3. ed. Boston, MA, EUA: Addison-Wesley Longman Publishing Co., Inc., 1992.

# APÊNDICE A – VALORES PARCIAIS DOS TREINAMENTOS

**Treinamentos de calibração sem palavras vizinhas**

| Arquitetura 1 | | Arquitetura 2 | | Arquitetura 3 | |
|---|---|---|---|---|---|
| perda | acurácia | perda | acurácia | perda | acurácia |
| 0,7431 | 0,4400 | 0,3814 | 0,4800 | 0,3394 | 0,5300 |
| 0,4979 | 0,4700 | 0,3280 | 0,5600 | 0,3432 | 0,5700 |
| 0,4696 | 0,5200 | 0,3258 | 0,5600 | 0,2976 | 0,5800 |
| 0,4800 | 0,5200 | 0,3765 | 0,5600 | 0,3336 | 0,5900 |
| 0,4669 | 0,5600 | 0,3898 | 0,5700 | 0,3003 | 0,6100 |
| 0,3400 | 0,5700 | 0,3308 | 0,5700 | 0,2559 | 0,6300 |
| 0,3962 | 0,5900 | 0,3411 | 0,6000 | 0,2636 | 0,6500 |
| 0,4509 | 0,5280 | 0,3502 | 0,5640 | 0,3061 | 0,5960 |

| Arquitetura 4 | | Arquitetura 5 | | Arquitetura 6 | |
|---|---|---|---|---|---|
| 0,3396 | 0,5300 | 0,2961 | 0,5600 | 0,3591 | 0,5800 |
| 0,3088 | 0,5900 | 0,3074 | 0,5700 | 0,3741 | 0,5900 |
| 0,2947 | 0,5900 | 0,2943 | 0,6000 | 0,3598 | 0,6000 |
| 0,2582 | 0,5900 | 0,3101 | 0,6000 | 0,3701 | 0,6000 |
| 0,2759 | 0,6000 | 0,2812 | 0,6300 | 0,3606 | 0,6000 |
| 0,2607 | 0,6100 | 0,2863 | 0,6300 | 0,3690 | 0,6400 |
| 0,2715 | 0,6300 | 0,2856 | 0,6400 | 0,3588 | 0,6400 |
| 0,2797 | 0,5960 | 0,2959 | 0,6060 | 0,3667 | 0,6060 |

| Arquitetura 7 | | Arquitetura 8 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| 0,3189 | 0,5500 | 0,3519 | 0,5600 | 0,2495 | 0,6100 |
| 0,3097 | 0,5700 | 0,3542 | 0,5900 | 0,2653 | 0,6200 |
| 0,3226 | 0,5700 | 0,3006 | 0,5900 | 0,2554 | 0,6200 |
| 0,3062 | 0,5900 | 0,3001 | 0,6000 | 0,2612 | 0,6400 |
| 0,2942 | 0,6100 | 0,3377 | 0,6200 | 0,2622 | 0,6600 |
| 0,2826 | 0,6200 | 0,2761 | 0,6500 | 0,2537 | 0,6700 |
| 0,2806 | 0,6400 | 0,3073 | 0,6600 | 0,2631 | 0,6800 |
| 0,3031 | 0,5920 | 0,3137 | 0,6100 | 0,2596 | 0,6420 |

**Treinamentos de calibração com palavras vizinhas**

| Arquitetura 1 | | Arquitetura 2 | | Arquitetura 3 | |
|---|---|---|---|---|---|
| perda | acurácia | perda | acurácia | perda | acurácia |
| 0,6573 | 0,4100 | 0,3210 | 0,5500 | 0,3642 | 0,5300 |
| 0,6828 | 0,4200 | 0,3411 | 0,5800 | 0,3370 | 0,5700 |
| 0,7575 | 0,4300 | 0,3315 | 0,6100 | 0,3232 | 0,5700 |
| 0,7263 | 0,4800 | 0,3272 | 0,6100 | 0,3381 | 0,5800 |
| 0,6427 | 0,4900 | 0,3277 | 0,6100 | 0,4023 | 0,5800 |
| 0,5396 | 0,5000 | 0,3015 | 0,6100 | 0,3427 | 0,5800 |
| 0,4952 | 0,5500 | 0,3264 | 0,6100 | 0,3465 | 0,5900 |
| 0,6698 | 0,4640 | 0,3258 | 0,6040 | 0,3487 | 0,5760 |

| Arquitetura 4 | | Arquitetura 5 | | Arquitetura 6 | |
|---|---|---|---|---|---|
| 0,3655 | 0,5500 | 0,4530 | 0,5200 | 0,3733 | 0,5600 |
| 0,3539 | 0,5700 | 0,3175 | 0,5400 | 0,3730 | 0,5600 |
| 0,3258 | 0,5700 | 0,3338 | 0,5500 | 0,3926 | 0,5700 |
| 0,2867 | 0,6000 | 0,3206 | 0,5500 | 0,3711 | 0,5700 |
| 0,3113 | 0,6000 | 0,3253 | 0,5600 | 0,3721 | 0,5900 |
| 0,2937 | 0,6200 | 0,3188 | 0,5700 | 0,3717 | 0,6000 |
| 0,3397 | 0,6200 | 0,3179 | 0,6200 | 0,4085 | 0,6300 |
| 0,3143 | 0,5920 | 0,3232 | 0,5540 | 0,3761 | 0,5780 |

| Arquitetura 7 | | Arquitetura 8 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| 0,3758 | 0,5200 | 0,3319 | 0,5300 | 0,2849 | 0,6200 |
| 0,3845 | 0,5200 | 0,3200 | 0,5400 | 0,2792 | 0,6300 |
| 0,3925 | 0,5200 | 0,4119 | 0,5600 | 0,2708 | 0,6400 |
| 0,4009 | 0,5300 | 0,3804 | 0,5600 | 0,2685 | 0,6500 |
| 0,3930 | 0,5300 | 0,4408 | 0,5700 | 0,2757 | 0,6500 |
| 0,3173 | 0,5900 | 0,3633 | 0,5900 | 0,2964 | 0,6600 |
| 0,3501 | 0,6100 | 0,4214 | 0,5900 | 0,2720 | 0,6800 |
| 0,3776 | 0,5380 | 0,3833 | 0,5640 | 0,2781 | 0,6460 |

**Treinamentos CBOW sem palavras vizinhas**

| Arquitetura 1 | | Arquitetura 8 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| perda | acurácia | perda | acurácia | perda | acurácia |
| 0,5304 | 0,4700 | 0,3563 | 0,5200 | 0,3274 | 0,5500 |
| 0,3497 | 0,5100 | 0,3660 | 0,5600 | 0,2686 | 0,5700 |
| 0,3744 | 0,5400 | 0,3420 | 0,5700 | 0,2735 | 0,5900 |
| 0,3443 | 0,5500 | 0,3101 | 0,5800 | 0,2809 | 0,5900 |
| 0,3594 | 0,5500 | 0,3103 | 0,5900 | 0,2688 | 0,5900 |
| 0,4039 | 0,5500 | 0,3275 | 0,5900 | 0,2699 | 0,6100 |
| 0,3204 | 0,6000 | 0,2940 | 0,6000 | 0,2815 | 0,6400 |
| 0,3663 | 0,5400 | 0,3312 | 0,5780 | 0,2723 | 0,5900 |

**Treinamentos GloVe sem palavras vizinhas**

| Arquitetura 1 | | Arquitetura 8 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2857 | 0,5400 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2925 | 0,5600 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2683 | 0,5700 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2384 | 0,6600 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2466 | 0,6700 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2367 | 0,6800 |
| 0,4400 | 0,5600 | 0,2411 | 0,6500 | 0,2563 | 0,7000 |
| 0,4400 | 0,5600 | 0,4400 | 0,5600 | 0,2565 | 0,6280 |

**Treinamentos CBOW com palavras vizinhas**

| Arquitetura 1 | | Arquitetura 2 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| 0,4259 | 0,5000 | 0,3230 | 0,5200 | 0,3211 | 0,5500 |
| 0,5090 | 0,5100 | 0,3171 | 0,5500 | 0,3108 | 0,5800 |
| 0,4019 | 0,5300 | 0,3188 | 0,5700 | 0,3571 | 0,5800 |
| 0,4814 | 0,5400 | 0,3524 | 0,5900 | 0,2974 | 0,5900 |
| 0,4502 | 0,5400 | 0,3100 | 0,6000 | 0,3085 | 0,5900 |
| 0,4774 | 0,5400 | 0,3340 | 0,6000 | 0,3137 | 0,6000 |
| 0,4811 | 0,5500 | 0,3248 | 0,6000 | 0,3065 | 0,6100 |
| 0,4640 | 0,5320 | 0,3265 | 0,5820 | 0,3175 | 0,5880 |

**Treinamentos GloVe com palavras vizinhas**

| Arquitetura 1 | | Arquitetura 2 | | Arquitetura 9 | |
|---|---|---|---|---|---|
| 0,4074 | 0,5800 | 0,3733 | 0,4800 | 0,3767 | 0,4700 |
| 0,4358 | 0,5600 | 0,3188 | 0,5300 | 0,3740 | 0,5300 |
| 0,4370 | 0,5600 | 0,3189 | 0,5600 | 0,3363 | 0,5500 |
| 0,4400 | 0,5600 | 0,2844 | 0,5800 | 0,2867 | 0,5800 |
| 0,4400 | 0,5600 | 0,2395 | 0,6600 | 0,2764 | 0,6000 |
| 0,4400 | 0,5600 | 0,2282 | 0,6900 | 0,2883 | 0,6200 |
| 0,4400 | 0,5600 | 0,2251 | 0,6900 | 0,2817 | 0,6600 |
| 0,4386 | 0,5600 | 0,2780 | 0,6040 | 0,3123 | 0,5760 |

**Treinamentos CBOW e GloVe com base modificada**

| CBOW | | GloVe | |
|---|---|---|---|
| 0,4800 | 0,5200 | 0,4800 | 0,5200 |
| 0,3200 | 0,5700 | 0,4800 | 0,5200 |
| 0,2762 | 0,6000 | 0,4800 | 0,5200 |
| 0,2545 | 0,6200 | 0,4800 | 0,5200 |
| 0,2755 | 0,6200 | 0,4800 | 0,5200 |
| 0,2530 | 0,6300 | 0,4800 | 0,5200 |
| 0,2670 | 0,6300 | 0,4800 | 0,5200 |
| 0,2758 | 0,6080 | 0,4800 | 0,5200 |

**Treinamentos fastText com base modificada**

| Skip-Gram | | CBOW | |
|---|---|---|---|
| 0,3188 | 0,5500 | 0,4800 | 0,5200 |
| 0,2635 | 0,6300 | 0,4800 | 0,5200 |
| 0,2435 | 0,6400 | 0,4800 | 0,5200 |
| 0,2399 | 0,6500 | 0,4800 | 0,5200 |
| 0,2464 | 0,6800 | 0,4800 | 0,5200 |
| 0,2440 | 0,6900 | 0,4800 | 0,5200 |
| 0,2330 | 0,7200 | 0,4800 | 0,5200 |
| 0,2475 | 0,6580 | 0,4800 | 0,5200 |

**Treinamentos Wang2vec com base modificada**

| Skip-Gram | | CBOW | |
|---|---|---|---|
| 0,4800 | 0,5200 | 0,4800 | 0,5200 |
| 0,4800 | 0,5200 | 0,2875 | 0,6100 |
| 0,4800 | 0,5200 | 0,2310 | 0,6600 |
| 0,3075 | 0,5900 | 0,2772 | 0,6600 |
| 0,2352 | 0,6900 | 0,2324 | 0,6900 |
| 0,2316 | 0,6900 | 0,2392 | 0,7000 |
| 0,2091 | 0,7200 | 0,2238 | 0,7200 |
| 0,3469 | 0,6020 | 0,2535 | 0,6640 |

# APÊNDICE B – EXEMPLO DE SAÍDA DO SIMPLIFICADOR 1

| | |
|---|---|
| **1. Identificador da amostra** | brasil_g50_3305_9 |
| **2. Posição da palavra no texto** | 3305 |
| **3. Palavra original** | represália |
| **4. Palavra candidata** | repressão |
| **5. Classe** | 1 |

| **6. Informações psicolínguisticas** | | |
|---|---|---|
| | **represália** | **repressão** |
| **concretude** | -1,1921325610 | -1,6766584060 |
| **imaginabilidade** | -0,3664549763 | 0,7814221363 |
| **familiaridade** | -0,9361115569 | -1,0280573356 |
| **idade de aquisição** | 0,7530846315 | 0,6327743370 |

| **7. *Embedding* da palavra "represália"** | | | | |
|---|---|---|---|---|
| 0,231011003 | -0,293195993 | 0,259074003 | -0,992981017 | -0,206073999 |
| 0,257400990 | -0,209634006 | 0,202647999 | -0,690985978 | -0,077803001 |
| 0,509540975 | 0,414790004 | -0,416382998 | 0,036338001 | -1,052263975 |
| 0,078230001 | 0,170130000 | -0,650497019 | 0,151208997 | 0,812367022 |
| -0,004120000 | 0,249467000 | -0,065104999 | -0,085998997 | -0,340249002 |
| -0,570400000 | 0,391961008 | -0,531952977 | -0,758114994 | -0,927106977 |
| -0,500531018 | 0,315961987 | -0,110701002 | 0,273119986 | -0,417551994 |
| 0,167241007 | 1,199056029 | -0,242768005 | 1,188128948 | 0,133662999 |
| -0,054889999 | 0,133057997 | 0,294723004 | -0,667788982 | -0,133839995 |
| -0,516049981 | 0,230278000 | 0,701497018 | -0,156810999 | 0,584890008 |

| **8. *Embedding* da palavra "repressão"** | | | | |
|---|---|---|---|---|
| 0,9766680002 | -0,3575069904 | 0,1110949963 | -2,2273859978 | -0,0195719991 |
| -0,3557640016 | -0,2708750069 | 0,3898999989 | -0,7538480163 | -0,5362650156 |
| 0,0688449964 | -0,0313210003 | -0,9413980246 | 0,0686250031 | -0,1187610030 |
| -0,5296090245 | -0,0801139995 | -0,7097730041 | 0,9168670177 | 0,5568209887 |
| 0,1699900031 | 0,6594539881 | -1,2119909525 | 0,0470039994 | 0,3391410112 |
| -0,4676899910 | 0,1622709930 | -0,7090340257 | -0,3277580142 | 0,1123289987 |
| -0,1839070022 | -0,1483629942 | -0,7337020040 | 0,7138919830 | 0,0401949994 |
| 0,0506719984 | 0,8238250017 | -0,9807569981 | 0,9623320103 | 0,3047809899 |
| 0,3876900077 | 1,2446479797 | 0,2863180041 | -1,0629199743 | 0,2438260019 |
| -1,0087159872 | 0,4902290106 | 1,0036560297 | -1,0581690073 | 0,0245849993 |

| **9. *Embedding* da palavra vizinha 1** | | | | |
|---|---|---|---|---|
| 0,156290993 | -0,060926002 | -0,399794012 | -3,360342979 | -0,499485999 |
| 0,458884001 | 0,500075996 | -0,351559997 | -0,583468020 | -0,050381001 |
| 0,241864994 | -0,480159998 | 0,090331003 | 0,915189028 | -0,298501998 |
| -0,260194987 | 0,327892989 | 0,542298019 | 0,459583014 | 0,293790013 |
| 0,623341024 | 0,654250026 | 0,351285011 | -0,596518993 | 0,297104001 |
| 0,275393993 | 0,107697003 | -0,195360005 | 0,299337000 | 0,869534016 |
| -0,512543023 | -0,719923019 | 0,097332999 | 0,365307987 | 0,604771018 |
| -0,079158999 | 0,141944006 | -0,338667005 | -0,673910975 | 0,009053000 |
| 0,269982994 | -0,498012990 | -0,753140986 | -0,597724020 | 0,164846003 |
| 0,331115007 | 0,208120003 | -0,126036003 | -0,053831998 | -0,229479998 |

| **10. *Embedding* da palavra vizinha 2** | | | | |
|---|---|---|---|---|
| 0,331759006 | 0,431746989 | -0,173583999 | -0,442717999 | -0,619530976 |
| 0,850477993 | 1,119259953 | 0,249578997 | 0,031575002 | 0,781768978 |
| 0,765788019 | 0,852522016 | 1,141818047 | -0,337706000 | 0,897650003 |
| 0,392506003 | 0,299591005 | -0,758888006 | 0,274980992 | -1,020558000 |
| 0,368523002 | 0,840247989 | -0,194325998 | -1,346554995 | 0,691073000 |
| -0,815589011 | 0,341259986 | -0,259285986 | 0,526562989 | 0,734402001 |
| -0,233840004 | -0,246547997 | 1,250498056 | 0,166508004 | -0,446754009 |
| 0,403308004 | -0,509458005 | -1,262490034 | 0,308566988 | -0,184283003 |
| 0,440061986 | 0,323428988 | -0,100815997 | 0,811703026 | -0,116227001 |
| -0,344453007 | -0,140193000 | 0,229957998 | -0,499778986 | 0,023061000 |

| **11. Distância Manhattan** | 21,38164508 |
|---|---|