

**COORDENADORIA DE ENGENHARIA DA COMPUTAÇÃO**

**GUILHERME PROENÇA CRAVO DA COSTA**

**RENATO DRUZIAN**

**SUMARIZAÇÃO DE NOTÍCIAS**

**Sorocaba/SP**

**2020**

**Guilherme Proença Cravo da Costa**

**Renato Druzian**

**SUMARIZAÇÃO DE NOTÍCIAS**

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.

Orientador: Prof. Johannes Von Lochter

**Sorocaba/SP**

**2020**

FICHA CATALOGRAFICA

C837s

Costa, Guilherme.

Sumarização de notícias / por Guilherme Costa, Renato Druzian. – Sorocaba, SP: [s.n.], 2020.

51f.

Trabalho de Conclusão de Curso (Graduação) – Centro Universitário Facens – Curso de Engenharia de Computação, 2020.

Orientador: Prof.(a) Johannes von Lotcher

1. Inteligência Artificial. 2. Sumarização Extrativa. 3. Sumarização Abstrata. I. Druzian, Renato. II. Centro Universitário Facens. III. Título.

CDD 621.39

ELABORADA PELA “BIBLIOTECA FACENS”

**Guilherme Proença Cravo da Costa**

**Renato Druzian**

**SUMARIZAÇÃO DE NOTÍCIAS**

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.

Orientador: Prof. Johannes von Lotcher

Sorocaba, 26 de Novembro de 2020

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me Johannes Von Lochter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me Jones Artur Gonçalves

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Me André Breda Carneiro

“Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar a onde a maioria não chega, faça o que a maioria não faz.”

Bill Gates

**RESUMO**

Este trabalho sobre Sumarização Extrativa e Abstrata apresentou métodos capazes de produzir resumos automaticamente com um conjunto de notícias coletadas de um site e modelo pré-treinado já criado. O objetivo é gerar textos e frases com auxílio da Inteligência Artificial e Processamento de Linguagem Natural capazes de sintetizar as informações principais para que o tempo de leitura seja menor, e preservar relações sintáticas e semânticas da língua portuguesa. Os resultados da Sumarização Extrativa, são relacionados as frases mais relevantes em relação ao título. Em contra partida a Sumarização Abstrata gera um texto novo com um limite de palavras.

**Palavras-Chave**: Sumarização Extrativa, Sumarização Abstrata, Inteligência Artificial, Processamento de Linguagem Natural.

**ABSTRACT**

This work regarding Extractive and Abstract Summarization outlines the methods capable of automating production of a set of news gathered from a pre-trained website and previously created model. The objective is to generate texts and phrases with the aid of Artificial Intelligence and Natural Language Processing capable of synthesizing core information so that the reading time is shorter, whilst preserving the relationship of the syntactic and semantic within the Portuguese language. The results of the Extractive Summarization, are listed the most relevant phrases in relation to the title. In contrast, Abstract Summarization generates new text with a limit of words.

**Key-words**: Extractive Summarization, Abstract Summarization, Artificial Intelligence, Natural Language Processing.

**LISTA DE ILUSTRAÇÕES**

[Figura 1. Funcionamento do método](#_Toc40185531) *[CboW](#_Toc40185531)* [19](#_Toc40185531)

[Figura 2. Funcionamento do método](#_Toc40185532) *[Skip Gram](#_Toc40185532)* [20](#_Toc40185532)

[Figura 3. Representação de grafo direcionado 24](#_Toc40185532)

[Figura 4. Tabela de grafo direcionado transformado em matriz 25](#_Toc41147217)

[Figura 5. Exemplo de multiplicação da coluna com peso de matriz 25](#_Toc40185532)

[Figura 6. Exemplo de resultado com](#_Toc40185532) *[PageRank](#_Toc40185532)* [26](#_Toc40185532)

Figura 7. Tabela de grafo não direcionado transformado em matriz ....... 26

[Figura 8. Etapas de criação de sumário com](#_Toc40185532) *[PageText](#_Toc40185532)* [27](#_Toc40185532)

[Figura 9. Arquitetura da metodologia](#_Toc40185532) *[Sequence to Sequence](#_Toc40185532)* [29](#_Toc40185532)

[Figura 10. Tipos de Redes Neurais Recorrentes 31](#_Toc40185532)

Figura 11. Exemplo de tradução *Seq2Seq* 32

Figura 12. Arquitetura *LSTM* 34

Figura 13. Arquitetura *GRU* 35

Figura 14. Implementação do *Web Scraping* 38

Figura 15. Modelo *World Embbedings* e manipulação dos títulos 41

[Figura 16. Manipulação das notícias e resultados de semelhança 42](#_Toc40185532)

**LISTA DE QUADROS**

Quadro 1. Resultados do *TextRank* 43

**LISTA DE TABELAS**

Tabela 1. Exemplo de vetor *Bag of Words....* 16

Tabela 2. Otimizador *Adamax* e *batch* 32 e 287 notícias............................ 45

Tabela 3. Otimizador *Adamax* e *batch* 16 e 287 notícias............................ 45

Tabela 4. Otimizador *Adam* e *batch* 32 e 287 notícias................................ 46

Tabela 5. Otimizador *Adam* e *batch* 16 e 287 notícias................................ 46

Tabela 6. Otimizador *Adagrad* e *batch* 32 e 287 notícias............................ 46

Tabela 7. Otimizador *Adagrad* e *batch* 16e 287 notícias............................. 46

Tabela 8. Otimizador *Adam* e *batch* e 2.191 notícias.................................. 47

Tabela 9. Otimizador *Adam* e *batch* 32 e 2.191 notícias............................. 47

Tabela 10. Otimizador *Adam* e *batch* 64 e 2.191 notícias........................... 47

**LISTA DE SIGLAS**

*BoW* *Bag of Words*

*CBoW* *Continuous Bag of Word*

*GloVe* *Global Vectors for Word Representations*

*GRU Gated Recurrent Unit*

*IA Inteligência Artificial*

*Json JavaScript Object Notation*

*LSTM  [Long Short Term Memory](http://deeplearningbook.com.br/arquitetura-de-redes-neurais-long-short-term-memory/)*

*[NLP Natural Language Processing](https://en.wikipedia.org/wiki/Recurrent_neural_network)*

*[RNN](https://en.wikipedia.org/wiki/Recurrent_neural_network)* *[Recurrent Neural Network](https://en.wikipedia.org/wiki/Recurrent_neural_network)*

*Seq2eq*  *Sequence-to-sequence*

**SUMÁRIO**

[1 INTRODUÇÃO 14](#_Toc41145484)

[2 REPRESENTAÇÃO DE TEXTO 16](#_Toc41145485)

[2.1](#_Toc41145486) *[Bag of Words](#_Toc41145486)* [16](#_Toc41145486)

[2.2](#_Toc41145487) *[Word Embeddings](#_Toc41145487)* [17](#_Toc41145487)

[2.3](#_Toc41145488) *[Skip Gram](#_Toc41145488)* [19](#_Toc41145488)

[2.4](#_Toc41145489) *[Cbow](#_Toc41145489)* [20](#_Toc41145489)

[2.5](#_Toc41145490) *[GloVe](#_Toc41145490)* [21](#_Toc41145490)

[3 ABORDAGENS DE SUMARIZAÇÃO 23](#_Toc41145491)

[3.1 Sumarização Extrativa 23](#_Toc41145492)

[3.2 Sumarização Abstrata 29](#_Toc41145492)

[3.2.1](#_Toc41145492) *[Sequence-to-Sequence](#_Toc41145492)* [30](#_Toc41145492)

[3.2.2 Mecanismo de atenção........................................................33](#_Toc41145492)

[3.2.3](#_Toc41145492) *[LSTM](#_Toc41145492)* [e](#_Toc41145492) *[GRU](#_Toc41145492)*[........................................................................34](#_Toc41145492)

[3.2.4 Trabalhos na Literatura 36](#_Toc41145492)

[4 RESULTADOS 38](#_Toc41145493)

[4.1](#_Toc41145492) *[Web Scraping](#_Toc41145492)* [38](#_Toc41145492)

[4.2 Sumarização Extrativa com](#_Toc41145492) *[TextRank](#_Toc41145492)* [41](#_Toc41145492)

[4.3 Sumarização Abstrata com](#_Toc41145492) *[Sequence to Sequence](#_Toc41145492)* [43](#_Toc41145492)

[5 CONCLUSÃO 45](#_Toc41145495)

REFERÊNCIAS.......................................................................................48

# 

# INTRODUÇÃO

O mundo está cada vez mais repleto de informações não-estruturadas, principalmente texto. Mídias sociais, como Twitter e Facebook, tiveram alto crescimento nos últimos anos e influenciam diariamente com opiniões e notícias.

Sites de notícias são fontes provedoras de informações muitas vezes confiáveis, mas o volume de notícias nem sempre é possível de ser acompanhado por uma pessoa ocupada. Rino & Pardo (2003) ”...viajar pelas páginas de notícias a fim de apreender o que é essencial exige tempo, capacidade de identificar o que é relevante, no grande volume de informações disponível, e capacidade de mentalizar, de forma coerente, o conteúdo essencial...”.

Máquinas começaram a ser empregadas para realizar tarefas que antes eram das pessoas, como secretarias que resumem notícias financeiras para os patrões ou agentes de *home brokers* geram *insights* para investidores, que possibilitou a diminuição do tempo de muitos processos.

Em IA, uma das técnicas mais recentes para tratar de sumarização de texto são redes neurais recorrentes. Este método possui aplicações para solucionar algumas análises de sentimento, entidades nomeadas e sumarização de texto.

Primeiramente, para a sumarização de texto, há a coleta das notícias que são utilizadas como entrada na rede neural, nessa coleta são captados os títulos e os textos das notícias onde o título fica sendo nosso parâmetro de comparação para a saída da rede neural.

Posteriormente, na etapa de teste, que consiste em dividir os dados coletados em treino e teste, esses dois grupos podem ter tamanho que for necessário. A divisão mais comum é 70% para treino e 30% para teste, a rede aprenderá com o grupo de treino e o resultado obtido pela mesma será analisado com o grupo de teste.

A rede neural aprenderá lendo o texto da notícia e fará uma ligação com o título, já na etapa de teste lerá as notícias do grupo que ela não conhece os dados e tentará gerar um título, ao término será realizado uma análise dessa saída com o título original, com isso será feito a análise, de acordo com Ferneda (2006) ”As redes neurais artificiais se diferenciam pela sua arquitetura e pela forma como os pesos associados às conexões são ajustados durante o processo de aprendizado”.

Entretanto, deve-se observar que se a maioria das notícias na parte de treinamento possuírem títulos sensacionalistas ou que não condizem com as reais informações apresentadas no texto, será gerado resultados não confiáveis, podendo com isso gerar um algoritmo enviesado, ou seja, que pensa de forma muito parecida com a fonte das notícias.

Para evitar esse problema e garantir melhor assertividade, é necessário um grande volume de dados e de várias fontes, evitando assim criar algum viés na rede neural.

Por fim, a área científica poderá beneficiar-se para melhor inserção de títulos em artigos, assim como resenhas ou textos gerais em instituições acadêmicas, pois a sumarização possibilita que mais textos sejam lidos em menor tempo, sem que haja perda no sentido para que a mensagem seja transmitida.

# REPRESENTAÇÃO DE TEXTO

Texto e palavras como o ser humano compreende, um computador não consegue entender, para isso, é necessário realizar algumas tratativas para que o computador possa compreendê-los.

Realizar apenas o pré processamento do texto, não quer dizer que o computador consiga entender ainda, pois pode-se utilizar sinônimos para manter o mesmo contexto da frase ou texto (KRYSCINSKI *et. al*,2019) (LUO XU, & GUO, 2014).

Outras problematizações encontradas são na análise semântica, de contexto e léxica, esses são relacionados, por exemplo, que robôs humanoides não entenderam sarcasmo.

O campo da inteligência artificial, que estuda a forma como a comunicação e trabalho são realizados para que máquinas possam comunicar-se com as pessoas, é chamado de Processamento de Linguagem Natural que, muitas vezes, é usando na aprendizagem profunda com o uso de redes neurais. Alguns exemplos de uso do *PLN* são assistentes pessoais e *chatbots*.

## Bag of Words

Este método é o mais simples, mas com a possibilidade de resultados satisfatórios dependendo do conjunto de palavras, chamado de vocabulário, onde cada uma delas representa uma amostra de palavra contida no texto, e com esse vocabulário é que realiza-se a análise de outros textos.

Uma análise é iniciada com a criação do vocabulário a partir da primeira amostra, então, cria-se um vetor com 0 e 1, que indica se uma palavra do vocabulário está presente no texto analisado ou não, porém dessa forma é perdido a posição dela na frase e o seu contexto também é alterado (BROWNLEE, 2017).

Para ficar mais claro, abaixo um exemplo com duas frases:

1. A FACENS é nota máxima no Enade
2. A FACENS conseguiu obter nota 5 no Enad

Com essas duas frases pode-se criar o vocabulário, montado da seguinte forma {A, FACENS, nota, conseguiu, obter, máxima, Enade, 5, é, no}. Com essas informações, monta-se o vetor de 1 e 0, que fica de acordo com a Tabela 1.

Tabela 1 – Exemplo de vetor *Bag of Words*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | FACENS | nota | conseguiu | obter | máxima | Enade | 5 | é | no |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

Fonte: Elaborado pelos autores.

Com essa tabela é possível visualizar como seria uma saída do método *BoW*, onde a primeira linha é representada pelo vocabulário, e cada linha a mais na tabela representa uma frase que através de cálculos e análises para fazer comparações, o computador consegue “entender” a frase. .

Pode-se observar mais um problema nesse método, quanto maior for o vocabulário, maior será a tabela criada, uma vez que o total de linhas sempre será a quantidade de palavras no vocabulário.

## Word Embeddings

Este método é amplamente utilizado nos dias atuais, segundo o estudo de Bolukbasi, Chang, Zour, Saligrama, & Kalai (2016), este método representa cada palavra ou alguma frase em vetores de n-dimensões, utilizado como dicionário para agrupar palavras similares.

Algumas das vantagens deste método são análise semântica e de contexto de uma determinada palavra em um texto, outra forma de análise é a verificação do radical, prefixo e sufixo de uma palavra, com isso é possível encontrar sinónimos para uma palavra.

Para este tipo de processamento são usadas redes neurais para identificar e compreender qual o contexto que a palavra está inserida, e com essa possibilidade é possível resolver problemas como recuperação de

informação, classificação de documentos e respostas a perguntas (BOLUKBASI *et. al*, 2016), (KRYSCINSKI *et. al*, 2019), (HUANG *et. al*, 2012).

Analogias entre palavras podem ser criadas de acordo com a distância desses vetores, tais como:

Como todos os métodos anteriores apresentados, esse também possui desvantagem, para que o modelo possa aprender melhor são necessários uma grande quantidade de textos, com isso é possível analisar muitas palavras em diversos contextos e em diferentes posições semântica, quanto maior o vocabulário melhor será o resultado do modelo e maior será o tempo de treino do modelo.

## Skip Gram

Conforme estudo de Chen *et al.* (s.d) o *Skip Gram* é um método baseado em vetores, ou seja, cada palavra é transformada em um vetor e com a finalidade de prever quais são as combinações de palavras disponíveis para uso antes e depois da palavra, dado a mesma como entrada.

Assim, é possível analisar o contexto em que a palavra está inserida para encontrar essas combinações. Outro uso deste método é, dado dois vetores e realizar a soma deles, gerar na saída um terceiro vetor com relevância em relação a soma dos dois vetores, como no exemplo abaixo:

Nesse exemplo, realiza-se a conexão das palavras Brasil e Capital pode-se notar que a saída dele é Brasília.

Outro ponto a ser observado é que, ao treinar um modelo com *Skip Gram*, encontra-se uma combinação de palavras que tenha sentido no contexto.

Dado uma sequência de palavras de treino, , ,…,, o *Skip Gram* tem o objetivo de aumentar a probabilidade de uma palavra fazer sentido no contexto de acordo com a Equação 1.

(1)

Onde tem-se que é o tamanho do vocabulário para treino, quanto maior for , maiores são os exemplos de treino, maior o tempo de treinamento e, consequentemente, melhor o resultado do modelo.

A base da fórmula do *Skip Gram* é definida por , que também é escrita como a Equação 2:

(2)

São as representações dos vetores de entradas e saídas dadas por e da palavra , e é o número de palavras no vocabulário, ou seja, o número total de palavras presentes no conjunto de treino.

## Cbow

No trabalho de Luo, Xu, & Guo (2018) *CBoW* é mais uma técnica de representação de texto que utiliza vetores para simbolizar palavras, esta técnica também faz uso de redes neurais recorrentes para avaliar e prever uma palavra em um determinado contexto.

De uma forma bem simples pode-se dizer que o *CBoW* faz o caminho inverso do *Skip Gram,* onde o último quer, através de uma palavra prever toda a frase, enquanto o *CBoW* possui a frase mas com uma palavra faltando, e o mesmo pretende prever essa palavra faltante.

O uso de redes neurais recorrentes se enquadra, pois as palavra antes são representadas como sendo o passado e as palavra após a lacuna como sendo o futuro, e como lacuna a palavra que deseja-se prever.

Abaixo há duas imagens que mostram de maneira mais clara a diferença entre eles na Figura 1 e Figura 2.

Figura 1 - Funcionamento do método *CBoW*

Uma imagem contendo objeto, relógio

Descrição gerada automaticamente

Fonte: Disponível em: https://arxiv.org. Acesso em 12 mai 2020

Figura 2 - Funcionamento do método *Skip Gram*

Uma imagem contendo objeto, relógio

Descrição gerada automaticamente

Fonte: Disponível em: https://arxiv.org. Acesso em 12 mai 2020

## GloVe

Conforme estudos realizados por Pennington, Socher, & Manning (2014) os Vetores Globais, ou *Global Vectors,* são amplamente utilizados para criar vetores semânticos em representação da linguagem. São utilizados em aplicações de classificação de documentos, respostas automáticas de questões, recuperação de informação e análises textuais.

Existem dois modelos conhecidos para aprendizagem de vetores de palavras, que são:

* Fatorização de matrizes, utilizados para análise semântica;
* Janela de contexto local, como o *Skip Gram* por exemplo

Para continuar no estudo do *GloVe,* ao definir  como representação de uma matriz que conta a ocorrência de uma palavra do conjunto de teste por uma palavra do conjunto de treino e  representa o número de vezes que uma palavra  apareceu no contexto da , de acordo com a Equação 3:

(3)

E por último tem-se a probabilidade de uma palavra ocorrer no contexto  com a Equação 4 :

(4)

A resultante do vetor, , que aponta para a palavra que mais se adequa ao contexto com a Equação 5:

(5)

Onde tem-se que e são vetores de palavras e e são vetores de palavras livres de contexto. Simplificando a Equação 5 - Vetor resultante *GloVe*, tem-se:

(6)

# ABORDAGENS DE SUMARIZAÇÃO

Para a empresa Sciforce [(2019](https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715?source=post_page-----e8439cd54715----------------------)) sumarizar consiste em um resumo que seja conciso e fluente, e manter o significado geral com as informações-chave do texto. Assim, espera-se que o computador tenha melhor desempenho na tarefa do que um humano por conta da imparcialidade.

Sem perceber, a sumarização de textos está cada vez mais presente nas mais diversas áreas de estudo. No meio acadêmico, os cientistas tendem a ler mais resumos de artigos antes mesmo de ler todo o artigo, uma vez que não encontrem o que procuram, procura-se outras fontes.

Para os vestibulandos, ler os resumos dos livros, ou criar os próprios resumos das disciplinas para estudo, e assimilar melhor o conteúdo apenas com pontos mais importantes dos livros ou o que foi ensinado em sala de aula.

Ao entrar em algum site de notícia, a manchete é lida primeiro e seu rápido resumo, e se houver mais interesse em saber detalhes sobre aquela reportagem, entrar na página para conferir na íntegra a notícia.

Para realizar a tarefa de sumarizar um texto, existem arquiteturas, métodos, classes e algoritmos a serem explorados. Neste capítulo será explicado mais a fundo sobre como funcionam e são implementados.

## Sumarização Extrativa

De acordo com Pardo & Rino (2018) a sumarização extrativa é baseada em métodos estatísticos e/ou empíricos, em que é necessário ter o título do texto para realizar comparações entre as palavras e frases do texto com seu título para obter valores comparativos. E nesse processo, a vetorização de palavras e a técnica de similaridade de cosseno auxiliam a encontrar a proximidade das palavras ou frase do título.

Com a similaridade de cosseno, gera-se valores entre 0 e 1, onde valores próximos de 0 possuem baixa similaridade, e próximos à 1 alta, dessa forma, é possível realizar as comparações e encontrar as frases mais relevantes.

O estudo da empresa Sciforce (2019) relata a utilização de frases relevantes no texto para produzir um sumário com subcojunto baseado em frases obtidas do texto original.

Sumarizadores extrativos partem de três princípios de tarefas independentes:

1) Construção de uma representação intermediária do texto de entrada: Há duas formas de abordagem. Representar o texto em formato de tópico para ser interpretado e representado de maneira intermediária, formada com a entrada de dados, e saída depois do processamento. Ou através de indicadores de importância, baseado em uma lista de características de cada frase.

2) Pontuação das sentenças com base na representação: Após a geração de representação intermediária, cada frase recebe um coeficiente entre 0 e 1, que representa maior importância da frase quanto mais próximo e 1 e menor relevância quando mais próximo de 0.

3) Seleção de um resumo composto por uma série de frases: Selecionar um certo número de frases mais importantes para produzir o resumo. Atentar-se para a coerência e minimizar a redundância durante a produção.

Para coletar dados de sites de notícias, a sumarização extrativa possui algoritmos capazes organizar as páginas da *web* em grafos. São conjuntos representados por vértices ou nós, com letras, e arestas, com setas, Figura 3, onde o nó da ponta da seta é a saída e o da origem é a entrada representando uma página *web* conectada a outra como, por exemplo, um *link* de  para (TOWARDS DATA SCIENCE, 2019).

Figura 3 - Representação de grafo direcionado



Fonte: Disponível em: https://towardsdatascience.com. Acesso em 2 set 2020

O algoritmo *PageRank* utiliza grafos, como da Figura 3, chamado de grafo direcionado ou dirigido por causa do fluxo de informações entre os nós serem em uma única direção, para atribuir pesos às páginas da *web e* classificá-las em importância.

Após a construção completa do grafo, os cálculos para atribuir os pesos é dado pela Equação 7.

 (7)

Onde  é o peso de cada página na *web* representado por , um fator de saída de links por , um conjunto de *links* de entrada por , um conjunto e a quantidade de *links* de saída por .

Em termos práticos, para calcular o peso do nó , em destaque na Figura 3, apenas analisando os nós e arestas de  para representá-los em uma matriz e, para simplificar o cálculo, atribuir peso 1 quando houver um *link* entre os nós, de acordo com a ponta da seta, e 0 caso não tenha, descrito na Figura 4.

Figura 4 - Tabela de grafo direcionado transformado em matriz



Fonte: Disponível em:https://towardsdatascience.com/. Acesso em 2 set 2020

É possível observar um *link* em três casos, ,  e .

Em seguida, cada coluna dessa matriz é normalizada de acordo com  da Equação 3, cada linha é somada e multiplicada com peso de todos os nós. Por enquanto, sem levar em consideração o fator de saída , como mostra a Figura 5, o peso de  é igual a 1,5 após uma iteração, ou seja, realizar o processo uma vez.

Figura 5 - Exemplo de multiplicação da coluna com peso de matriz



Fonte: Disponível em:https://towardsdatascience.com/. Acesso em 2 set 2020

Já se o fator de saída  for igual à 0,85 e 10 iterações forem realizadas, o peso de , de acordo com o *PageText*, é 0,34125. Quanto maior for a quantidade de iterações e o tamanho na matriz, mais custoso e demorado é o processo de cálculo. A Figura 6 mostra o valor de cada nó em cada iteração.

Figura 6 - Exemplo de resultado com *PageRank*

Fonte: Disponível em:https://towardsdatascience.com/. Acesso em 3 set 2020

Apenas uma iteração era necessária para atribuir o peso de , porém em casos mais complexos, as iterações podem variar de valor.

Caso o grafo seja não direcionado, ou seja, sem uma única direção entre um nó e outro, a matriz formada que corresponde ao direcionado fica de acordo com a Figura 7 e, o cálculo do peso segue as mesmas etapas anteriores.

Figura 7 - Tabela de grafo não direcionado transformado em matriz



Fonte: Disponível em: https://towardsdatascience.com/. Acesso em 2 set 2020

Joshi (2018) baseado no *PageRank*, o algoritmo *PageText* segue a mesma premissa de classificação, entretanto, pesos são atribuídos à frases de um texto para formar um resumo, também chamado de sumário. Essa análise é feita com quantidade de recomendações da página *web*.

Geralmente, algumas palavras são retiradas dessa análise por não apresentarem relevância significativa no texto, chamadas de *stop words*, como artigos e preposições.

As etapas para criar o sumário partem de 1) uma combinação de textos, 2) separar cada texto em frases individuais, 3) representar as frases em vetores, 4) calcular a similaridade de cada vetor de frase e armazenar em uma matriz, 5) converter a matriz em um grafo não direcionado com as frases sendo nós e as arestas a recomendação e, por fim, 6) as melhores frases classificadas compõem o sumário. A Figura 8 ilustra cada etapa utilizada pelo *PageText*.

Figura 8 - Etapas para criação de sumário com *PageText*



Fonte: Disponível em:<https://www.analyticsvidhya.com/>. Acesso em 2 set 2020

Uma implementação possível é com a linguagem de programação Python, que oferece diversas bibliotecas e pacotes para facilitar algumas etapas.

*Numpy*:auxilia em cálculos numéricos e pode ser aplicada para resolver tarefas como modelos de *Machine Learning*, cálculos matemáticos como

integração e diferenciação, funções de álgebra linear, processamento de imagem e computação gráfica.

*Pandas*: Realiza análises de dados ou estruturas de dados, chamados de *datasets*, de uma forma simples e com alta performance.

*NLTK*: Auxilia em processar dados em Linguagem Natural, ao dividir as frases em *tokens* e separá-las em grupos diferentes como, por exemplo, palavras, números, datas e ponto final e, assim, gerar vetores.

*KeyedVectors*: Para carregar modelos pré-treinados de maneira mais rápida como, por exemplo, solicitações para carregar páginas na *web*.

*Cosine\_similarity*: Calcula a similaridade de cosseno entre as amostras de um ponto à outro, com base no produto escalar normalizado .

*NetworkX*: Criação e manipulação para estudar estruturas, funções e redes mais elaboradas.

*Json*: Representa estrutura de dados em formato de texto com a sintaxe do objeto *JavaScript*.

*TensorFlow:* Utilizada para aprendizado de máquina em diversas tarefas. É um sistema de redes neurais para criar e treinar, com objetivo de realizar detecção, decifrar padrões e correlações.

*Keras*: Com o *TensorFlow* e o uso de redes neurais profundas, torna o processo mais rápido e facilita o uso de módulos e extensões.

Com essas ferramentas e métodos, é possível construir uma estrutura de dados capaz de extrair e sumarizar textos de páginas *web*. No próximo capítulo, será mostrado o algoritmo e resultado para realizar essa tarefa.

## Sumarização Abstrata

No estudo feito por Bing *et. al* (2016) Sumarização Abstrata refere-se a tarefa de gerar títulos e resumos, capturando as ideias principais. Chamada de abstrata devido ao potencial de utilizar palavras que não estão presentes no documento original para gerar o título ou resumo.

Segundo Newtechdojo (2018) A categorização é dividida em três: Aprendizagem Supervisionada baseada em prever dados conhecidos de entrada e saída na etapa de treinamento no processo de aprendizagem, Aprendizagem Não Supervisionada em que apenas os dados de entrada são conhecidos e não há como saber qual será o resultado da saída e Reforço ou Semi-Supervisionado que é uma mescla dos dois anteriores e mais comum em aplicações do mundo real.

## 3.2.1 Sequence to Sequence

Nikolov, Pfeiffer & Hahnloser (2018) *Sequence to Sequence* é uma categorização de modelo utilizada em Redes Neurais para processar o texto e gerar na saída um texto novo e legível para uma pessoa.

Regularmente utilizada em estudos da área de Processamento de Linguagem Natural, ou mais conhecido como NLP, principalmente em traduções. *Encoder-Decoder* é uma arquitetura da rede *Seq2Seq* com diversas variações que um grande volume de dados de entrada são ordenados em sequência e codificados para que métodos e algoritmos sejam capazes de classificá-los em algum contexto e, posteriormente, decodificados pela saída gerando outra sequência, com a tradução ou sumarização legível do texto.

A figura 9, ilustra como arquitetura *Encoder-Decoder* no modelo *Seq2Seq* funciona na sumarização de texto.

Figura 9 – Arquitetura da metodologia *Sequence to Sequence*

**

Fonte: Disponível em: https://github.com/. Acesso em 2 set 2020

Na publicação de Trask (2015) Redes Neurais Recorrentes é um modelo que propaga-se como matriz que definir uma relação entre suas camadas em cada etapa de tempo. Somente a camada oculta baseasse nos dados de entrada. O fluxo de informações e camadas mais simples é descrito abaixo.



Basicamente, a camada oculta tem o efeito de uma memória para armazenar informações da camada anterior.

Em situações mais elaboradas, com apenas uma entrada, é possível criar camadas ocultas a partir da camada de entrada, como no exemplo:



Já se houver mais de uma entrada, criar uma camada de saída com a oculta. Este processo muda o efeito de memória que a camada oculta possui para guardar as informações relevantes recebidas da anterior, já que a combinação entre os dados da camada de entrada com a oculta são de tempos diferentes, como no exemplo:



Outras variações que possuem dois fluxos, percorrendo cada tempo duas vezes, chamadas de bidirecionais. A Figura 10 ilustra com mais detalhes como funciona cada tipo *RNN* em relação a sua arquitetura, representando cada bloco como um vetor, onde vermelho é a camada de entrada, verde oculta e azul saída.

Figura 10 - Tipos de Redes Neurais Recorrentes



Fonte: Disponível em: http://karpathy.github.io/. Acesso em 30 ago 2020

Segundo [Karpathy](http://karpathy.github.io/) (2015) O tipo de *RNN* *one-to-one* é o mais clássico com uma única camada de entrada, saída e oculta de dados. No *one-to-many* uma camada de entrada gera várias de saídas, e cada camada oculta é processada em um período de tempo com uma saída.

Outros modelos possuem várias camadas entradas como o *many-to-one,* queproduz uma sequência de dados na saída após várias leituras na entrada, utilizado para análises de sentimentos.

Já o *many-to-many* pode funcionar de duas maneiras. A primeira que antes de começar a gerar a sequência de saída de imediato, processa alguns períodos de tempo, utilizado em casos de tradução de máquina. Já na segunda, o processamento da sequência de entrada e saída pertencem ao mesmo período de tempo.

Para implementar métodos e algoritmos na sumarização de notícias, algumas ferramentas foram utilizadas durante o decorrer do projeto para contribuir na organização e evitar possíveis problemas.

O Google Collaboratory foi usado para compartilhar códigos de testes. Já o Anaconda Navigator versão 1.9.12 para criar e padronizar os ambientes, minimizando o tempo de seção e Internet instável e a linguagem de programação Python versão 3.7 com a importação de bibliotecas auxiliares.

Os códigos feitos foram armazenados na plataforma GitHub para centralizar as versões do projeto e evitar possíveis perdas de conteúdo.

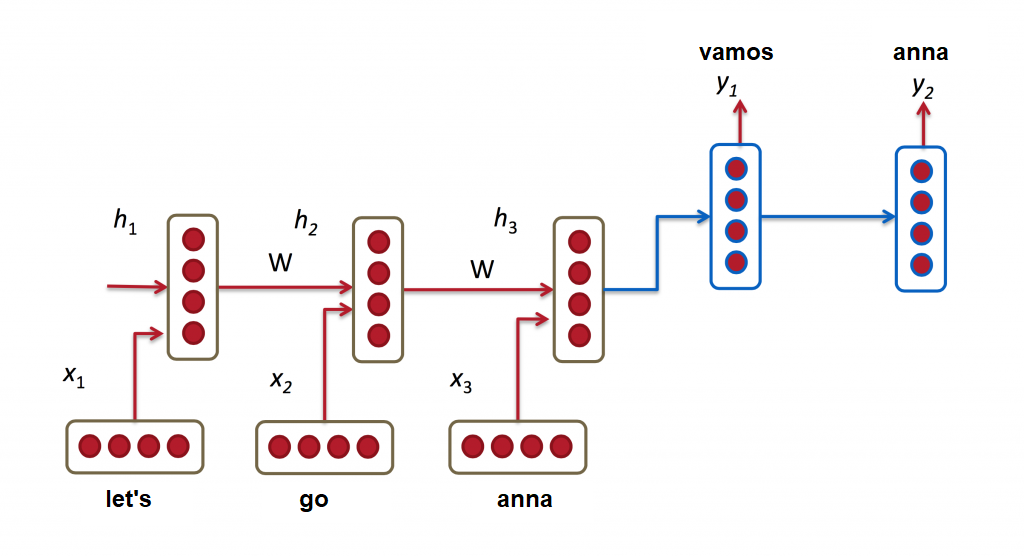
*3.2.2 Mecanismo de atenção*

De acordo com Dantas (2019) a tradução de um idioma para outro em aplicações de aprendizado de máquina têm algumas barreiras, já que o número de palavras pode variar, além de trabalhar com um número limitado de frases e, assim, generalizar padrões da linguagem.

Modelos *Seq2Seq* são amplamente utilizados nesses casos, pois uma sequência de palavras de entrada para o *Encoder* é reduzida em um vetor de tamanho fixo com seu contexto, no qual o *Decoder* gera outra sequência com o que foi traduzido.

A frase “*let’s go anna*” em inglês com três palavras é traduzida para o português “vamos anna” com apenas 2, como na Figura 11.

Figura 11- Exemplo de tradução *Seq2Seq*

**

Fonte: Disponível em:http://www.wildml.com/. Acesso em 11 nov 2020

Entretanto, para grandes sequências, o tamanho fixo é um problema, pois todo o contexto de entrada é considerado e é necessário representar essa dependência ao longo do tempo.

Como solução, o mecanismo de atenção atua para que o *Decoder* “preste atenção” em cada parte da sequência de entrada ao gerar cada

saída.Então, ao invés de codificar em um vetor de contexto toda sequência de entrada, o *Decoder,* através de estados intermediários, gera a saída.

Assim, o tipo possível de mecanismo de atenção é o *[BahdanauAttention](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention)*[, que possui duas formas:](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention)

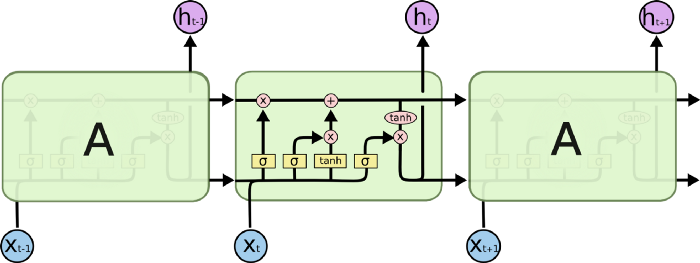
[A primeira de acordo com Bahdanau, Bengio & Cho (2016), apesar de apresentar melhorias mais significativas em frases longas, o mesmo pode ser observado para qualquer tamanho. Sua característica na arquitetura](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention) *[Encoder-Decoder é](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention)* [codificar a frase de entrada em uma sequência de vetores e não apenas um único, de tal forma que, ao escolher um subconjunto desses vetores com a melhor adaptação de contexto, por exemplo, uma tradução de frase é decodificada.](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention)

[Já a segunda, com os estudos de Kingma, & Salimans (2016) a normalização de pesos propoe que, em uma rede neural, os vetores de peso com uma simples reparameterização garantam a descida do gradiente estocástico. O objetivo que é conforme as iterações ocorram, os valores dos parâmetros minimizem a algum ponto de interesse, melhorando a otimização.](https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/BahdanauAttention)

3.2.3 *LSTM* e *GRU*

Para um Filipa (2017) um tipo especial de *RNN* é a *LSTM* que, naturalmente, guarda informações de longos e curtos períodos de tempo, ou seja, garante mais precisão para prever a próxima palavra. Apesar de sua estrutura também ser em cadeia, existem quatro interações nas camadas, como mostra a Figura 12.

Figura 12 - Arquitetura *LSTM*

Fonte: Disponível em: https://www.cetax.com.br/. Acesso em 11 nov 2020

A linha de entrada horizontal superior recebe os dados de longo prazo, como o dataset completo, já a inferior os de curto, como as informações mais recentes. Com estruturas chamadas portas, têm a capacidade de remover ou adicionar informações a camada.

Cada célula dessa rede neural possui três portões sigmóide (σ) que controlam o estado, onde cada um produz 0 que descarta e 1 mantêm a informação.

No início, é definido o que será descartado com a primeira camada sigmóide e, posteriormente, decidir quais novas informações serão armazenadas este estado da célula baseada em um peso .

Na próxima etapa é há duas partes, a primeira em que próxima camada sigmóide decide quais valores serão atualizados a memória de longo prazo e, a segunda, em que uma camada de cria um vetor de novos valores possíveis.

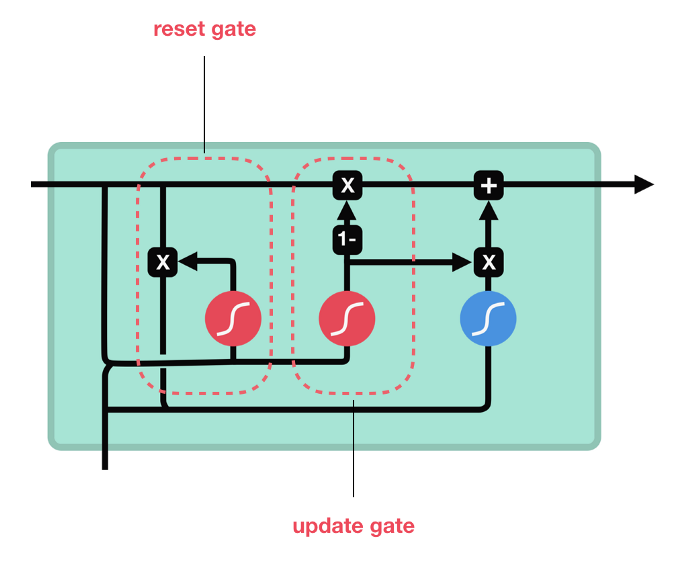
Assim, ao multiplicar o estado anterior que esquece as informações, novos valores podem atualizar o estado, baseado no seu valor.

Por fim, uma versão filtrada da célula utiliza as partes mais relevantes e coloca em um estado  que multiplica a saída da sigmóide anterior com a decisão do que será produzido no tempo atual que gera a previsão em .

Para Phi (2018) a *GRU* apesar de bem semelhante a *LSTM*, sua diferença é por conta de possuir uma camada a menos, com isso, estado

oculto transfere as informações e apenas duas portas, uma de reinicialização “*reset gate*”e outra de atualização ”*uptade gate*” como ilustra a Figura 13.

Figura 13 - Arquitetura *GRU*



## Fonte: Disponível em: [https://towardsdatascience.com.](https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.) Acesso em 11 nov 2020

A função da porta de atualização é informar informações para serem descartadas ou adicionadas da rede. Assim, o de reinicialização decide as informações de tempos anteriores que podem ser esquecidas.

Apesar de trabalhar melhor com tensores, que são matrizes de *n-dimensões*, consumir menos memória e processamento, não se pode dizer se é melhor ou pior que a *LSTM*, pois sequências grandes não são bem processadas pela *GRU* ,ou seja, deve-se analisar a tarefa a ser resolvida e verificar qual mais adequadamente se encaixa.

## 3.2.4 Trabalhos na Literatura

No artigo de Xiang *et. al* (2016) foi proposto aplicar uma *Encoder-Decoder* *Attention* para melhorar o desempenho da sumarização abstrata com dois *datasets* diferentes. O objetivo era melhorar a forma como uma soma abstrata é feita e solucionar dois problemas específicos. Os resultados foram promissores nos dois *datasets*, e um novo foi cogitado para construir um modelo mais robusto em trabalhos futuros.

Já no artigo de Hara, Adelson, & Arora (2017) misturar dois modelos de redes neurais, o *Quasi-Recurrent Neural Network* e *RNN* padrão, para diminuir perdas ao longo do tempo. Entretanto, apenas o *RNN* isolado teve resultados significativos ao produzir

manchetes. Os títulos gerados com a mescla dos modelos apesar de ter apresentado uma sequência de aprendizagem, não obteve bons resultados na tentativa de diminuir perdas com baixo custo, considerado 'falta de *IA*', e todos os modelos apresentaram potencial para melhoria.

E com os estudos de Ding, Soricut, & Zhang (2018) duas arquiteturas de modelos foram criadas, *SHAPED* e *Mix-SHAPED*, com a papel de adaptar o estilo para gerar uma sequência de texto. Ambos os modelos tiveram melhor desempenho na etapa de treinamento se comparados aos que não tinham adaptação, ou seja, a saída gera uma média de estilo e, também, a modelos com estilo único.

# RESULTADOS

## Web Scraping

Para a coleta de dados, um robô para *Web Scraping* em Python foi implementado com a função de extrair e estruturar dados *web* em forma de vetores de palavras e frases para serem analisados. Neste caso, foram coletadas, aproximadamente, 28000 notícias de 3000 páginas *web* do site G1.

Cada notícia foi coletada a partir da página principal do site e a busca pela categoria. Com isso, os dados são salvos em um arquivo *Json* capaz de reconhecer a gramática *PT-BR* e estruturar a divisão de cada notícia em título, categoria e texto como na Figura 14.

Figura 14 - Implementação do *Web Scraping* 

Fonte: Elaborado pelos autores

Os resultados armazenados no arquivo *Json* seguem esta estrutura:

"título": "Brasil chega a 16.792 mortes e se torna 3º do mundo com mais casos",

"categoria": " Bem Estar "

"texto": " O Ministério da Saúde divulgou nesta segunda-feira (18) seu mais recente balanço de mortes e casos confirmados de Covid-19. Os principais dados são: 16.792 mortes, eram 16.118 no domingoEm 24 horas, foram mais 674 novas mortes registradas254.220 casos confirmados, eram 241.080 casos no domingoEm 24 horas, foram mais 13.140 casos De acordo com o ministério, 136.969 pacientes estão em acompanhamento (53,9% do total) e 100.459 estão recuperados (39,5%). PRETOS E PARDOS: Em 4 semanas, mortes de pretos e pardos por Covid-19 passam de 32,8% para 54,8%VACINA: Empresa anuncia resultados 'positivos preliminares' em teste de vacinaDISTANTES: Portugal tem volta às aulas parcial nesta segunda-feiraSÃO PAULO: Câmara de SP aprova antecipação de feriados municipais para aumentar isolamento socialTerceiro em número de casos no mundo Com a atualização, o Brasil tem mais casos confirmados que o Reino Unido. De acordo com o monitoramento da Universidade Johns Hopkins, o Reino Unido tinha, até 19h45, pouco mais de 247 mil casos. No topo da lista ainda aparecem a Rússia (290 mil) e Estados Unidos (1,5 milhão)

## Sumarização Extrativa com TextRank

Para este método, foi utilizado *Json* criado pelo *Web Scraping,* após o carregamento do arquivo *Json,* foram criados dois *arrays*, um contendo os títulos e o outro apenas o corpo das notícias.

Em seguida, foi iniciada a etapa de pré-processamento dos textos, onde algumas tratativas foram feitas para diminuir os ruídos contidos nos textos, alguns dos exemplos para esta etapa são: conversão do texto para minúsculo e remoção de *stop words*.

Para esta sumarização foi utilizado o modelo *GloVe* pré-treinado, de 100 dimensões, com 929.605 palavras pelo Núcleo Interinstitucional de Linguística Computacional (NILC) da Universidade de São Paulo (USP), que foi carregado com a biblioteca *Gensim* utilizando o método *KeyedVector*. Assim, todas as notícias e títulos transformados em uma matriz, onde cada linha representa uma frase da notícia e as colunas representam os valores de cada frase. Em seguida é criado um vetor com a média dos valores das frases das notícias e dos títulos.

Nesta etapa foi utilizado um conjunto menor, aproximadamente, 290 notícias, a Figura 15 mostra a implementação.

Figura 15 - Modelo *World Embbedings* e manipulação dos títulos



Fonte: Elaborado pelos autores

Após o desenvolvimento do dicionário e do cálculo da média, foi aplicado a similaridade de cosseno na matriz criada com os valores médios das notícias e títulos. Com essa matriz de similaridade criada podemos obter as frases que contem o maior e menor valor, onde o maior valor é a frase mais semelhante e o menor valor é menos semelhante, como ilustra a Figura 16.

Figura 16 - Manipulação das notícias e resultados de semelhança



Fonte: Elaborado pelos autores

O Quadro 1 abaixo mostra três exemplos de notícias onde se encontram as frases mais e menos semelhantes e o título da notícia:

Quadro 1 - Resultados do *TextRank*

|  |  |
| --- | --- |
| Título: falta de política clara para conter pandemia atrasa retomada da economia, dizem empresários | |
| Frase mais semelhante: executivos dão mostras de desconforto com a falta de coordenação entre a pasta e a presidência da república | Frase menos semelhante: a saída de nelson teich do ministério da saúde reacendeu o alerta entre empresários sobre o grau de confiança no brasil para o combate à pandemia do novo coronavírus |
| Título: brasil chega a 16.792 mortes e se torna 3º do mundo com mais casos | |
| Frase mais semelhante: os principais dados são: 16 | Frase menos semelhante: o ministério da saúde divulgou nesta segunda-feira (18) seu mais recente balanço de mortes e casos confirmados de covid-19 |
| Título: hospital de referência em natal fecha as portas do ps | |
| Frase mais semelhante: todos os leitos de uti e até os leitos de enfermaria estão ocupados no hospital giselda trigueiro | Frase menos semelhante: em natal, o hospital de referência para covid-19 fechou as portas do pronto-socorro para novos pacientes |

Fonte: Elaborado pelos autores

## Sumarização Abstrata com Sequence to Sequence

O processo de pré preocessamento é o mesmo realizado na sumarização extrativa, aqui a diferença é que foi utilizado métodos da biblioteca *Keras* do *Tensorflow,* outra diferença é que não é utilizado *arrays* e sim *tensores,* que são estruturas de dados mais complexas, e a biblioteca do *Keras* contribui para a aplicação. Assim, cada notícia tem tamanho máximo de 600 palavras, e cada título de 20. Então, 80% do *dataset* é destinado a parte de treino e 20% de teste.

Em sequência, os dados foram divididos da seguinte, utilizamos dois conjuntos de treino que foram divididos da seguinte maneira: entrada com 1.752 notícias de treino e 439 de teste e outro com 229 notícias de treino e 28 para teste.

O conjunto menor foi utilizado para validar algumas parametrizações da redes, e com a melhor parametrização utilizamos no conjunto maior para verificar se existe alguma diferença durante a etapa de treinamento.

Para a configuração da rede, algumas variáveis foram criadas, um *buffer* com o mesmo tamanho dos dados de entrada do treino, *batch* igual a 16 ou 32 que corresponde ao tamanho do lote que será processado, uma variável, *emb\_dim* com valor de 200, *units* com valor de 1024, um vocabulário para as notícias com 600 palavras e com 20 para o título.

Na arquitetura da rede, o processo de *Encoder* há uma camada de *Embedding* e uma camada *GRU* que recebe todas as saídas e o parâmetro *state,* além de possuir um método de inicialização da camada oculta com valores 0.

O modelo *BahdanauAttention* com a arquitetura *GRU* atua para extrair palavras importantes da sentença com uma pontuação ao predizer a palavra-alvo com o contexto atual no vetor intermediário, associando partes da sentença de entrada e todas as palavras relevantes geradas em tempos anteriores. Além disso, é capaz de alinhar a sentença de entrada e de saída que a corresponde (GONÇALVES, 2018).

Já para o processo de *Decoder*, é utilizado uma arquitetura bem similar com de *Encoder,* porém utiliza o parâmetro de *recurrent\_initializer* com o valor de *glorot\_uniform* e *dropout* de 10%.

Para o cálculo do erro da rede é utilizado diferentes otimizadores *Adamax, Adam e Adagrad* e, para o cálculo de perda, a Entropia Cruzada Esparsa.

A função de avaliação é responsável por formatar a predição da frase gerada pela rede, dado que também é fornecida a sentença que gerou a sumarização.

# CONCLUSÃO

Com este trabalho pode-se analisar duas formas de resolver a tarefa de sumarização de textos, uma onde apenas é analisada a correlação entre as frases das notícias com seu título, que torna mais simples o processo porém mais custoso, já que é necessário ter o um modelo de *Word Embedding* carregado durante a execução, e também se faz necessário saber o título da notícia.

Já para o método de Sumarização Abstrata foram executados vários testes, alterando os valores de *batch* e o parâmetro de otimização da rede*,* abaixo a Tabela 2, Tabela 3, Tabela 4, Tabela 5 Tabela 6 e Tabela 7 mostram as comparações analisadas. Os parâmetros de *loss* e *dropout* não foram alterados nestes testes, onde é indicado *loss* como *SparseCategoricalCrossentropy* e *dropout* de 10%.

Tabela 2 – Otimizador *Adamax* e *batch* 32 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adamax |
| Loss (época 100) | 1,4219 |
| Batch\_loss | 1,4058 |
| Batch | 32 |

Fonte: Elaborado pelos autores

Tabela 3 – Otimizador *Adamax* e *batch* 16 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adamax |
| Loss (época 100) | 0,6637 |
| Batch\_loss | 0,6090 |
| Batch | 16 |

Fonte: Elaborado pelos autores

Tabela 4 – Otimizador *Adam* e *batch* 32 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adam |
| Loss (época 100) | 0,0624 |
| Batch\_loss | 0,0514 |
| Batch | 32 |

Fonte: Elaborado pelos autores

Tabela 5 – Otimizador *Adam* e *batch* 16 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adam |
| Loss (época 100) | 0,0191 |
| Batch\_loss | 0,0122 |
|  | 16 |

Fonte: Elaborado pelos autores

Tabela 6 – Otimizador *Adagrad* e *batch* 32 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adagrad |
| Loss (época 100) | 4,2722 |
| Batch\_loss | 4,1938 |
| Batch | 32 |

Fonte: Elaborado pelos autores

Tabela 7 – Otimizador *Adagrad* e *batch* 16 e 287 notícias

|  |  |
| --- | --- |
| Otimizador | Adagrad |
| Loss (época 100) | 4,3139 |
| Batch\_loss | 4,2224 |
| Batch | 16 |

Fonte: Elaborado pelos autores

Com esses resultados é possível comprovar que dos otimizadores testados o que obteve menor *Loss* em 100 épocas e *Batch\_loss* foi o otimizador *Adam* de *batch* 16 com, respectivamente, 0,624 e 0,0514. Já o com pior resultado em ambos os parâmetros foi o *Adagrad*  e *batch* 16 com 4,3139 e 4,2224.

Por fim, para validar os testes anteriores e verificar qual o impacto do treinamento em um conjunto maior de notícias, foi realizado o treinamento da arquitetura com melhor desempenho no *dataset* com 2.191 notícias, em que 1752 fizeram parte do treino e 439 do teste, e os resultados foram inseridos nas Tabela 8, Tabela 9 e Tabela 10.

Tabela 8 - Otimizador *Adam* e *batch* 16 e 2191 notícias

|  |  |
| --- | --- |
| Otimizador | Adam |
| Loss (época 100) | 0,0650 |
| Batch\_loss | 0,0417 |
| Batch | 32 |

Fonte: Elaborado pelos autores

Tabela 9 - Otimizador *Adam* e *batch* 32 e 2191 notícias

|  |  |
| --- | --- |
| Otimizador | Adam |
| Loss (época 100) | 0,0100 |
| Batch\_loss | 0,0024 |
| Batch | 32 |

Fonte: Elaborado pelos autores

Tabela 10 - Otimizador *Adam* e *batch* 64 e 2191 notícias

|  |  |
| --- | --- |
| Otimizador | Adam |
| Loss (época 100) | 0,0258 |
| Batch loss | 0,0172 |
| Batch | 64 |

Fonte: Elaborado pelos autores

Ao analisar os dados, houve um pequeno ganho ao término da última época, porém, em relação ao *Batch\_loss* 32 e 64 tiveram resultados significativamente melhores. Isso mostra que o valor de *batch* tem impacto no resultado final, isso ocorre porque o *batch* é a quantidade de notícia que a rede processará em cada passo, isso indica que, neste caso, analisar muitas notícias no mesmo passo não implica em grandes melhorias ou maiores perdas.

Entretanto, o tempo para treinamento aumentou expressivamente por conta do tamanho do *dataset*, com o tempo de, aproximadamente, 2 horas com *batch* 16 e pouco menos de 2 horas para de 32 e 64, enquanto com um conjunto menor de dados, foi necessário em torno de 10 minutos para cada treino.

Foi realizado testes com um *dataset*, com aproximadamente, 28.000 notícias, mas não foi possível realizar o treinamento pelo Google Colaboratory. Este seria um teste interessante para mostrar a dependência da quantidade de notícias a serem analisadas pela rede.

**REFERÊNCIAS**

ADELSON, P., ARORA, S., & HARA, J. (2017). Abstractive text summarization with Quasi-Recurrent neural netwaorks. p. 7.

Bahdanau, D., Bengio Y.,& Cho, K. (2016) Neural machine translation by jointly learning to align translate., (p. 2)

Bolukbasi, T., Chang, K.-W., Zour, J., Saligrama, v., & Kalai, A. (2016). Man is to computer programmer as woman is to homemaker? Debiasing embeddings. *30th Conference on Neural Information Processing Systems (NIPS 2016)*, (p. 9). Barcelona.

Brownlee, J. (2017). *Machine Learning Mastery*. Fonte: Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-bag-words-model/

Chen, K., Corrado, G., Dean, J., Tomas, M., & Sutskever, I. (s.d.). Distribuited representations of words and phrases and their compositionality., (p. 9).

Dantas, S. (2019) Pav attention explicando o macanismo de atenção. Fonte: Lamfo: https://lamfo-unb.github.io/2019/05/01/Pay-attention-Explicando-o-mecanismo-de-Atencao/

Duarte, G., Matsumoto, F. & Murakami, L. (2019) *Redes Neurais Recorrentes*. Fonte: Turing Talks: <https://medium.com/turing-talks/turing-talks-26-modelos-de-predi%C3%A7%C3%A3o-redes-neurais-recorrentes-439198e9ecf3>

Ding, N., Soricut, R., & Zang., Y. (2018). Shared-Private Encoder-Decoder for Text Style. Adaptation. Association for Computational Linguistics., (p. 9). New Orleans.

Ferneda, E. (2006). *Redes neurais e sua aplicação em sistemas de recuperação de.* Ribeirão Preto.

Filipa, S. (2017) *Compreendendo LSTM Networks*.

Fonte: Cetax: https://www.cetax.com.br/blog/compreendendo-lstm-networks/

Gonçalves, L. (2018) *O que é sequence-to-sequence em deep learning?* Fonte: Medium: https://medium.com/luisfredgs/o-que-%C3%A9-sequence-to-sequence-em-deep-learning-9f8857a423ca

Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. *50th Annual Meeting of the Association for Computational Linguistics.* Jeju: 10.

Kapur, R (2016) Neural Networks & The Backpropagation Algorithm Explained Fonte: Medium: <https://ayearofai.com/rohan-lenny-1-neural-networks-the-backpropagation-algorithm-explained-abf4609d4f9d>

Kingma, P. ,& Salimans, T. (2016). Weight normalization a simple reparameterization to accelerate training of deep neural networks, (p. 8)

Kryscinski, W., Keshar, N. S., McCAnn, B., Xiong, C., & Socher, R. (2019). Neural text summarization: A critical evaluation., (p. 13).

Luo, Q., Xu, W., & Guo, J. (2014). A study on the CBOW model's overfitting and stability., (p. 4). Shangai.

[Meier](https://arxiv.org/search/cs?searchtype=author&query=Meier,+F), F., [Mujika](https://arxiv.org/search/cs?searchtype=author&query=Mujika,+A), A., & [Steger](https://arxiv.org/search/cs?searchtype=author&query=Steger,+A), A., (2018). Approximating Real-Time Recurrent Learning with Random Kronecker Factors., (p. 3).

NewTechDojo .(2018). *List of Machine Learning Algorithms*. Fonte: NewTechDojo: [https://www.newtechdojo.com/list-machine-learning-algorithms/#Reinforcement%20Learning](https://www.newtechdojo.com/list-machine-learning-algorithms/" \l "Reinforcement Learning)

NILC . (2017) *Repositório de Word Embeddings do NILC*.

Fonte: NILC: http://www.nilc.icmc.usp.br/embeddings

Paulus, R., Xiong, C., & Socher, R. (2017). A deep reinforced model for abstractive summarization., (p. 12). Palo Alto.

Pardo, T, & Rino, L. (2018). A sumarização automática de textos principais características e metodologias.

Phi, M. (2018) *Illustrated Guide to LSTM’s and GRU’s: A step by step explanation*. Fonte: Towards Data Science: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Sciforce. (2019). *Towards Automatic Text Summarization Extractive Methods*. Fonte: Medium: <https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715>

Trask, A., (2015). *Anyone Can Learn To Code an LSTM-RNN in Python Part1 RNN*. Fonte: iamtrask: <https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/>