Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal Catarinense
Campus Rio do Sul

MATHIAS ARTUR SCHULZ

DETECÇÃO DE *FAKE NEWS* A PARTIR DE REDES NEURAIS ARTIFICIAIS

Rio do Sul
2020

MATHIAS ARTUR SCHULZ

DETECÇÃO DE *FAKE NEWS* A PARTIR DE REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de graduação em Ciência da Computação do Instituto Federal Catarinense – Campus Rio do Sul para obtenção do título de bacharel em Ciência da Computação.
Orientador: Daniel Gomes Soares, Msc.

Rio do Sul

2020

AGRADECIMENTOS

Agradeço primeiramente a Deus por me conceder todas as oportunidades e estar constantemente me amparando.

Agradeço ao meu pai Artur Schulz, minha mãe Simone Sedlacek Schulz e meu irmão Thiago Schulz por todo o apoio e compreensão.

Agradeço também aos meus professores que estiveram constantemente me ensinando e ao meu orientador e professor Daniel Gomes Soares por ter aceito esse desafio e por ter me ajudado na elaboração deste trabalho.

Agradeço aos meus amigos, em especial ao Thiago Nicolau Fortunato e sua esposa Sirlene Frech Fortunato, por todo o apoio e dicas durante a minha trajetória acadêmica.

RESUMO

As falsas notícias são histórias fabricadas e construídas com base em notícias verídicas, no entanto possuem como objetivo enganar seus leitores a partir de informações que são falsas intencionalmente. Não são novas no cotidiano das pessoas e, além disso, a internet alavancou o seu crescimento e facilitou a forma como elas são disseminadas. As falsas notícias compartilhadas garantem poder, vantagem e podem causar nos leitores julgamentos precipitados, preconceitos, violência verbal e discriminação. Neste cenário, o combate das falsas notícias pode ocorrer de diversas formas, como o aumento da capacidade dos indivíduos em avaliar as notícias, por meio de características das notícias e sites que avaliam a veracidade das informações e a busca por mudanças estruturais que diminuam o acesso dos indivíduos às falsas resenhas. Dentre as técnicas disponíveis para a detecção de falsas notícias, estão as Redes Neurais Artificiais (RNAs), que são modelos computacionais inspirados no sistema nervoso de seres vivos. A finalidade deste trabalho é desenvolver um modelo de detecção de falsas notícias em sites de notícias brasileiros disponíveis nas mídias sociais por meio da técnica de RNA. Para a obtenção dos objetivos deste trabalho, foi realizada a revisão bibliográfica da literatura de modo a identificar os modelos que têm sido utilizados para a detecção de falsas notícias, as variáveis utilizadas e a forma de conversão dos textos para representação numérica. Foi realizada a modelagem e treinamento das redes Perceptron Multicamadas (MLP) e Memória de Longo Prazo (LSTM) com diversas configurações de variáveis de entrada e parâmetros iniciais. Por fim, é analisado e apresentado o desempenho das redes modeladas por meio de índices estatísticos e gráficos, no qual a melhor configuração para o modelo MLP apresentou um RMSE de 0.0439 e um F1 Score de 98.32% e a melhor configuração para o modelo LSTM apresentou um RMSE de 0.0551 e um F1 Score de 96.27%. Dessa forma, o modelo MLP obteve, neste trabalho, um desempenho melhor quando comparado ao modelo de aprendizado profundo LSTM.

Palavras-chave: Detecção. Falsas Notícias. Redes Neurais Artificiais.

ABSTRACT

False news is a story fabricated and built on the basis of true news, however they aim to deceive your readers from information that is intentionally false. They are not new in people's daily lives and, moreover, the internet has leveraged their growth and facilitated the way they are disseminated. The false news shared guarantees power, advantage and can cause readers to make hasty judgments, prejudices, verbal violence and discrimination. In this scenario, the fight against false news can occur in several ways, such as increasing the capacity of individuals to evaluate the news, through features of the news and websites that evaluate the veracity of the information and the search for structural changes that reduce access from individuals to false reviews. Among the techniques available for detecting false news, there are Artificial Neural Networks (ANNs), which are computational models inspired by the nervous system of living beings. The purpose of this work is to develop a model for detecting false news on Brazilian news sites available on social media using the ANN technique. To obtain the objectives of this work, a literature review of the literature was carried out in order to identify the models that have been used for the detection of false news, the variables used and the way of converting the texts for numerical representation. The modeling and training of the Multilayer Perceptron (MLP) and Long Short Term Memory (LSTM) networks was performed with several configurations of input variables and initial parameters. Finally, the performance of the modeled networks is analyzed and presented using statistical and graphical indices, in which the best configuration for the MLP model presented a RMSE of 0.0439 and a F1 Score of 98.32% and the best configuration for the LSTM model presented a RMSE of 0.0551 and a F1 Score of 96.27%. Thus, the MLP model obtained, in this work, a better performance when compared to the LSTM deep learning model.

Key words: Detection. Fake News. Artificial Neural Networks.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN – Convolutional Neural Network

FNC – Fake News Challenge

GRU – Grated Recurrent Units

IFCN – International Fact-checking Network

LSTM – Long Short Term Memory

MLP – Multilayer Perceptron

NLTK – Natural Language Toolkit

PNL – Processamento de Linguagem Natural

RF – Random Forest

RNA – Artificial Neural Network

RNN – Recurrent Neural Network

STF – Supremo Tribunal Federal

SVM – Support Vector Machine

USP – Universidade de São Paulo

VDCNN – Very Deep Convolutional Networks

RMSE - Root Mean Square Error

SUMÁRIO

1 INTRODUÇÃO

As falsas notícias, também chamadas de *fake news*, não são novas no cotidiano das pessoas, entretanto a internet alavancou o seu crescimento e facilitou a forma como elas são disseminadas. São fabricadas e construídas com base em notícias atuais, entretanto possuem diferenças quanto a suas intenções (BAUM *et al.*, 2018), no qual possuem como principal objetivo enganar seus leitores a partir de notícias que são falsas intencionalmente (TANDOC; LIM; LING, 2017).

Segundo o dicionário de Cambridge, *fake news* são falsas histórias que parecem ser notícias e se encontram espalhadas na internet ou usando outras mídias, geralmente criadas para influenciar opiniões políticas ou utilizadas como piada (FAKE NEWS, 2020). Se tornam populares principalmente devido a sua inserção em diversas áreas e tópicos, como: política, doenças, vacinas, nutrição e valores das ações (BAUM *et al.*, 2018).

A criação de *fake news* são motivadas a partir de dois fatores principais, que são: O financeiro, a partir do desenvolvimento de histórias falsas e que instigam as pessoas à leitura, propagam rapidamente a notícia e geram lucros ao autor a partir dos cliques na notícia; E ideológico, promovendo ou desfavorecendo pessoas específicas por meio da criação de falsas notícias (TANDOC; LIM; LING, 2017).

A internet e as mídias sociais são os principais canais para a disseminação de notícias falsas, principalmente a partir da facilidade da criação de novos sites. Além disso, não apenas fornecem um meio para publicar essas histórias, mas também oferecem ferramentas para promover ativamente a sua divulgação (BAUM *et al.*, 2018).

Para o combate de *fake news*, é possível realizar uma separação em duas categorias: A primeira forma está relacionada ao aumento da capacidade dos indivíduos em avaliar as notícias, buscando características que levem ao levantamento de um relato falso. Por exemplo, a partir de sites que avaliam a veracidade das informações, como o *PolitiFact* e *Snopes*; E a segunda categoria é a busca por mudanças estruturais que diminuam o acesso dos indivíduos às falsas resenhas (BAUM *et al.*, 2018).

Segundo uma pesquisa realizada pelo site de notícias e entretenimento *BuzzFeed*, no ano de 2016, as *fake news* de jornais enganavam os adultos americanos cerca de 75% das vezes. (TANDOC; LIM; LING, 2017). Os indivíduos tendem a aceitar as notícias com maior facilidade a partir de informações que se alinhem com suas respectivas crenças (como

partidárias e ideológicas) e também por meio de notícias que agradem a eles (BAUM *et al.*, 2018).

Um grande desafio para os indivíduos é a identificação, compreensão e a distinção a partir do olho humano entre notícias reais e falsas (VERMA; MITTAL; DAWN, 2019). Com isso, Redes Neurais Artificiais (RNAs) podem ser utilizadas para a detecção de *fake news*, a partir de diversos modelos existentes. A detecção de *fake news* também é um obstáculo para os métodos de análise utilizados atualmente, como métodos de aprendizado tradicionais e modelos de aprendizado profundo. Entretanto, grande parte dos modelos atuais não são capazes de identificar notícias recentes de eventos críticos em termos de tempo (ADBULLAH; QAWASMEH; TAWALBEH, 2019).

Considerando os recentes avanços das RNAs, o propósito principal deste trabalho é desenvolver um modelo de detecção de falsas notícias (*fake news*) em sites de notícias disponíveis nas mídias sociais por meio da técnica de Redes Neurais Artificiais, utilizando a abordagem de aprendizado profundo (*Deep Learning*). A pesquisa realizou também um comparativo entre dois modelos, uma utilizando o aprendizado profundo e outra não utilizando. Essa comparação visa constatar ou não, a vantagem do aprendizado profundo sobre a abordagem tradicional na identificação de *fake news*.

1.1 PROBLEMATIZAÇÃO

A detecção manual de uma *fake news* pode ser realizada de algumas formas, por exemplo buscando outras fontes e sites, modificando o algoritmo de busca *PageRank* da Google, possibilitando a busca a partir da importância e confiança das referências e não apenas a partir da quantidade de referências utilizadas. Entretanto, também é possível utilizar informações das notícias para sua classificação, como fontes, títulos, textos e imagens. Dessa forma, as RNAs permitem a detecção de falsas notícias a partir da diferença na estrutura do texto (MARUMO, 2018).

Existem diversos desafios para detectar *fake news*, em grande parte dos casos, elas se baseiam em histórias verdadeiras com poucos detalhes falsos, dificultando seu reconhecimento e enganando os leitores. As falsas notícias possuem uma grande complexidade e podem ser separadas em três tipos de notícias, que são: sátira, fraude e propaganda, no qual a formação dessas falsas histórias ocorrem por meio de palavras subjetivas, intensificadas e de cobertura, com a intenção de introduzir uma linguagem vaga, dramatizante ou sensacionalista (TIM, 2018).

A classificação de uma notícia é trabalhosa e complexa, além de necessitar bastante tempo do especialista. Com isso, para a resolução desse problema é possível a utilização Aprendizado de Máquina que realiza a classificação de forma mais rápida e, em alguns casos, mais precisa que os seres humanos. Contudo, a classificação de texto é complexa e abstrata, por isso o *Deep Learning* (subárea do aprendizado de máquina) pode ser um grande facilitador para esse problema (MARUMO, 2018).

A aplicação de abordagens para detecção de falsas notícias é trabalhosa, principalmente devido à complexidade das notícias, além disso outro fator determinante é a busca e criação de um conjunto de dados que possua notícias reais e falsas disponíveis na língua portuguesa, devido ao fato de grande parte dos estudos já realizados nessa área utilizarem como base a língua inglesa.

De acordo com estudos já realizados pelos autores citados acima, a detecção de *fake news* pode ser realizada a partir de diversas abordagens, como o aprendizado profundo. No entanto, grande parte dos autores focam na língua inglesa como base para a detecção, dessa forma surge a seguinte pergunta de pesquisa: É possível obter um resultado favorável na detecção de *fake news* utilizando como base a língua portuguesa?

1.1.1 Solução Proposta

A proposta deste trabalho é desenvolver um modelo de detecção de *fake news* baseado em Redes Neurais Artificiais para as áreas de interesse utilizando como conjunto de dados notícias reais e falsas. Neste problema, serão comparados dois modelos de RNAs, um que utiliza a abordagem de aprendizado profundo e outro que não a utiliza. Além disso, a adequação de uma topologia de RNA a um problema particular de detecção inclui diversos complicadores, dentre os quais:

1. Obtenção de notícias reais e falsas na língua portuguesa;
2. Tratamento dos dados;
3. Conversão dos dados para representação numérica;
4. Definição das variáveis de entrada;
5. Algoritmo de treinamento;
6. Algoritmo de otimização;
7. Função de ativação;
8. Número de camadas intermediárias;
9. Número de neurônios nas camadas intermediárias da rede.

1.1.2 Delimitação do Escopo

O estudo apresentará a modelagem de duas Redes Neurais Artificiais para a detecção de *fake news*. Será considerada a Rede Neural Recorrente LSTM (*Long Short Term Memory*) utilizando a abordagem de aprendizado profundo e a Rede Neural Artificial MLP (*Multilayer perceptron*) utilizando a abordagem tradicional. Além disso, a aplicação estará restrita à área de estudo selecionada, que consiste em notícias escritas na língua portuguesa.

1.1.3 Justificativa

O grande número de indivíduos divulgando notícias teve um grande crescimento nos últimos anos, principalmente nas mídias sociais. Esse aumento levou a um grande número de notícias difíceis de serem classificadas como verdadeiras ou falsas (ADBULLAH; QAWASMEH; TAWALBEH, 2019). Além disso, segundo Jacob, diretor administrativo do site de verificação de fatos *BOOM*, de Mumbai, o ano de 2019 foi o ano mais movimentado para verificadores de fatos até agora, atingindo o maior pico de falsas notícias criadas (CHATURVEDI, 2019).

Segundo Chaturvedi (2019) a disseminação de *fake news* não apresenta sinais de declínio, mesmo com algumas medidas realizadas pelos governos e plataformas das mídias sociais. A entidade sem fins lucrativos *Check4Spam*, responsável por verificar a veracidade das notícias nas mídias sociais, relata que, no ano de 2019 o número de mensagens recebidas para verificação teve um aumento de mais de 20%, subindo de 4.000 mensagens no ano de 2018 para aproximadamente 5.000 a 6.000 postagens por mês no ano de 2019, sendo um dos principais impulsionadores os eventos que ocorrem durante o ano.

Para a detecção de *fake news*, redes neurais genéricas também podem ser utilizadas, assim como são empregadas para diversas outras aplicações, como visão computacional e reconhecimento de fala. Contudo, elas não oferecem um excelente desempenho e não funcionam muito bem para grandes aplicações, com isso é necessário o uso de estratégias específicas. O *Deep Learning* permite o uso de técnicas especializadas para a modelagem de linguagem natural e processamento de dados sequenciais (BENGIO; GOODFELLOW; COURVILLE, 2015).

Segundo Deng e Yu (2013) o *Deep Learning* é uma classe de técnicas do aprendizado de máquina, no qual apresentam muitas camadas intermediárias para o processamento de informações não lineares, extração e transformação de recursos supervisionados ou não supervisionados e, além disso, análise e classificação de padrões.

A carência por formas de detecção de *fake news* na língua portuguesa, viabilizou a elaboração da investigação no tema. O presente trabalho auxiliará a comunidade online na detecção de falsas notícias, prevenindo a disseminação de notícias criadas apenas com a intenção espalhar falsas histórias.

Levando em consideração as afirmações dos autores citados, pretende-se através do presente trabalho contribuir para a literatura por meio da modelagem de um modelo de RNA para detecção de *fake news* e de um comparativo de tal modelo em vários níveis de profundidade.

1.2 OBJETIVOS

Esta seção descreve os objetivos geral e específicos do trabalho.

1.2.1 Objetivo geral

Desenvolver um modelo de detecção de falsas notícias em mídias sociais por meio da técnica de Redes Neurais Artificiais.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho estão descritos abaixo.

1. Investigar quais modelos têm sido utilizados no problema da detecção de falsas notícias;
2. Definir as principais variáveis que possam auxiliar na detecção de falsas notícias;
3. Obter conjuntos de dados para treinamento, teste e validação dos modelos;
4. Modelar e treinar os modelos em diversas configurações para detecção de falsas notícias no local de estudo selecionado;
5. Comparar as detecções realizadas pelos modelos com os dados de validação;

1.3 METODOLOGIA

Esta seção apresenta a metodologia empregada para o desenvolvimento do trabalho.

1. Revisão bibliográfica: Esta etapa objetiva o aprofundamento teórico acerca de detecção de *fake news* e Redes Neurais Artificiais aplicadas a este problema. A

pesquisa bibliográfica foi realizada em livros, teses, dissertações, monografias e artigos de periódicos;

2. Revisão sistemática da literatura: Esta etapa identificou o estado da arte na área de detecção de *fake news*. Delimitou quais modelos e métodos têm sido aplicados e identificou as características das detecções, como métricas de erro e variáveis de entrada utilizadas;
3. Tratamento dos dados: Com os dados obtidos realizou-se a normalização dos dados através de técnicas de Processamento de Linguagem Natural;
4. Modelagem das Redes Neurais Artificiais: Nesta etapa, são modeladas e treinadas diferentes configurações de RNAs para realizar a detecção das *fake news* nas áreas de interesse.
5. Comparação das RNAs: Obtenção do menor erro de cada modelo utilizado. Com isso, avaliação e comparação do desempenho das RNAs a partir das métricas de erro. As detecções realizadas por estas redes são comparadas com os dados reais sobre *fake news*.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são abordados conceitos relacionados às áreas de Redes Neurais Artificiais (RNAs) e *fake news*, os quais são fundamentais para a realização e clareza deste trabalho.

2.1 FAKE NEWS

No século XX, as propagandas em massa foram para os nazistas uma ótima forma de disseminar falsas notícias a seu favor e obter a atenção dos alemães. Antes do amplo uso da internet e das mídias sociais, divulgar uma *fake news* era mais complexo, pois prejudicava a reputação da publicação, possuía um alto custo de distribuição e o autor era penalizado (TRAUMANN, 2017).

No entanto, Traumann (2017) relata que atualmente os custos de divulgação de falsas notícias por meio das mídias sociais são pequenos e/ou inexistentes, a probabilidade do autor ser descoberto é menor e, além disso, indivíduos que produzem esses tipos de notícias não estão preocupados com sua reputação.

O conceito de notícias falsas surgiu como uma expressão para artigos que são falsos de forma intencional e verificada, parecem autênticos, no entanto possuem falsos fatos com o objetivo de atrair e enganar os leitores. Além disso, *fake news* são fabricadas imitando a forma do conteúdo de uma notícia, mas não conseguem copiar o processo e o propósito organizacional (RIBEIRO; ORTELLADO, 2018).

De acordo com Ribeiro e Ortellado (2018), as *fake news* são provenientes de diversas formas de comunicação, dentre os quais pode ser: Artigos de notícias, opiniões, piadas e sarcasmo, assim como rumores, memes, boatos, entre outros. Entretanto, é importante observar alguns casos que não são considerados falsas notícias, por exemplo:

a) Erros não intencionais em reportagens;
b) Rumores não provenientes de um artigo de notícia;
c) Teorias da conspiração;
d) Falsas afirmações de políticos.

Para uma tentativa de combate às *fake news* no Brasil, o Supremo Tribunal Federal (STF) lançou, em Junho de 2019, o Painel Multissetorial de Checagem de Informações e Combate a *Fake News*, no qual reuniu jornais, agências, associações de juízes e

tribunais superiores para a criação de sites e projetos com o objetivo de investigar e analisar as informações que circulam nas mídias sociais (BARBOSA; SANTI, 2019).

De acordo com uma pesquisa realizada pelo instituto Ipsos em 2018, 62% dos brasileiros acreditam em notícias que na verdade são falsas notícias. Com isso, é possível analisar que poucas pessoas buscam saber mais sobre as notícias lidas e verificar se a fonte e as informações da notícia são realmente verdadeiras (BARBOSA; SANTI, 2019).

Segundo Barbosa e Santi (2019), as notícias falsas compartilhadas garantem poder e vantagem, em muito casos o conteúdo causa julgamento precipitado, preconceitos, violência verbal e discriminação. Desta forma, podem ser utilizadas como aparato político e de poder para disseminar ideologias, favorecer determinadas pessoas e denegrir a imagem de outras, como políticos e outros agentes.

Para determinados indivíduos nas redes sociais, notícias que se alinhem a sua visão e ponto de vista, fazendo com que atinja as expectativas, pensamentos e preconceitos, independente da veracidade já é vista como uma notícia verdadeira. Com isso, a notícia passa a ser vista do ângulo subjetivo e emocional, que leva a aprovação de falsas informações e a manipulação do leitor a mudar seu pensamento sobre determinados temas, como na área política e científica (BARBOSA; SANTI, 2019).

2.1.1 Impactos das Fake News

Delmazo e Valente (2018) relatam uma pesquisa realizada pelo site *Buzzfeed News*, no qual apresentou que nos três últimos meses da campanha para as eleições presidenciais dos Estados Unidos, no ano de 2016, as principais as notícias falsas do Facebook eram mais populares e aceitas do que as principais notícias de veículos de comunicação, como o *The New York Times*, *Washington Post*, *Huffington Post*, *NBC News*, entre outros.

Além disso, as 20 notícias falsas com melhor performance na rede social possuíram 8.711.000 compartilhamentos, comentários e reações (DELMAZO; VALENTE, 2018). Dentre as principais notícias falsas que repercutiram, destacam-se por exemplo: "*Wikileaks* confirma que Clinton vendeu armas para o Estado Islâmico" e "Papa Francisco choca o mundo e apoia Donald Trump" (SPINELLI; SANTOS, 2018).

De acordo com Spinelli e Santos (2028), um caso de destaque ocorreu em uma pizzaria da Carolina do Norte, no qual um homem de 28 anos entrou atirando com o objetivo de investigar por conta própria uma teoria da conspiração fictícia que se propagou

rapidamente durante as eleições, no qual a pizzaria estava mantendo um cativeiro de tráfico sexual de crianças com o auxílio financeiro do Partido Democrata, felizmente nenhuma pessoa se feriu.

No Brasil, de acordo com uma pesquisa realizada pelo Grupo de Pesquisa em Políticas Públicas de Acesso a Informação da Universidade de São Paulo (USP), durante a semana anterior a votação de abertura do processo de Impeachment da então presidenta Dilma Rousseff, as cinco notícias mais compartilhadas no Facebook, três dessas eram falsas (DELMAZO; VALENTE, 2018).

Segundo Delmazo e Valente (2018), quando uma notícia é acessada, um dos grandes impactos está relacionado a qualidade da leitura, é necessário prestar uma atenção especial ao que está sendo lido, para que os artigos não fiquem descontextualizados em relação às suas fontes e que os fatos não se misturem livremente com ficção.

Uma reportagem divulgada no portal G1 no dia 19 de julho de 2018, realizada pelo programa Profissão Repórter, da Rede Globo, realizou uma investigação sobre notícias falsas que circulam na internet, a equipe entrou em contato e entrevistou Carlos Afonso, administrador do site Ceticismo Político. O site havia publicado notícias falsas sobre a vereadora Marielle Franco, assassinada no dia 14 de março de 2018, associando ela ao tráfico de drogas (HIGINO, 2019).

De acordo com Carlos, as fontes eram obtidas de uma desembargadora no Facebook, entretanto a desembargadora não conhecia Marielle e as informações eram baseadas na postagem de uma amiga. Depois de um tempo o Facebook tirou do ar todas as páginas relacionadas ao site. Entretanto, o texto do Ceticismo Político foi compartilhado mais de 360 mil vezes e a desembargadora apagou sua postagem (HIGINO, 2019).

Um boato disseminado nas redes sociais do Brasil, relatava que a ex-primeira dama Marisa Letícia, morta em 3 de fevereiro de 2017, estaria viva e viajando pela Itália. Além disso, outra *fake news* afirmava que o viúvo Marisa Letícia, o ex-presidente Luiz Inácio Lula da Silva, teria solicitado pensão referente ao salário de Marisa como servidora do Congresso Nacional, no valor de R$ 68 mil (DELMAZO; VALENTE, 2018).

Spinelli e Santos (2028) destacam uma pesquisa do *Pew Research Center*, no qual as notícias falsas confundem a interpretação dos fatos e eventos atuais para 64% dos americanos entrevistados. Diversas *fake news* são disseminadas nas mídias sociais, os exemplos acima são uma pequena amostra dessas notícias e os riscos reais que elas representam para a sociedade contemporânea.

2.1.2 Formas de combate utilizadas para as fake news

Sites responsáveis por disseminar notícias falsas estão a todo vapor na produção, devido principalmente aos cliques da audiência, que geram lucros, e a divulgação de *fake news* acaba sendo incentivada pela publicidade (SPINELLI; SANTOS, 2018).

De acordo com Higino (2019), o jornalismo possui como objetivo pesquisar, organizar, coletar e analisar dados ou fatos e transmitir as informações obtidas para a sociedade, por meio do jornal impresso, rádio, TV ou internet. Por meio dessas informações recebidas cada indivíduo aprimora seu conhecimento sobre determinado assunto.

Com as redes sociais, muitas pessoas podem fazer jornalismo e tornam o diploma de jornalismo opcional. *Fake news* podem ser produzidas e divulgadas por pessoas civis, instituições de variados âmbitos da sociedade e até pessoas públicas, com grande proporção de divulgação. Dessa forma, os jornalistas precisam conquistar novamente a credibilidade da sociedade, um momento em que as *fake news* são espalhadas com mais rapidez e facilidade do que as informações que foram checadas antes de serem publicadas (HIGINO, 2019).

Delmazo e Valente (2018) relatam que com o grande impacto das *fake news*, o Facebook a partir ano de 2016 iniciou o desenvolvimento de barreiras para a disseminação dessas notícias, como: Fim do patrocínio de publicações falsas; Possibilidade dos usuários denunciarem falsas notícias; Parceiras com verificadores de fatos; Alteração da linha do tempo para reduzir a disseminação e o impacto das *fake news*; Maior dificuldade para criação de contas falsas.

No ano de 2017, o Facebook identificou e fechou 470 perfis russos que teriam gasto US$ 100.000 em anúncios entre junho de 2015 e maio de 2017. Além disso, também foi lançado e disponibilizado uma ferramenta sobre dicas de como identificar notícias falsas (DELMAZO; VALENTE, 2018).

Segundo Delmazo e Valente (2018), uma das iniciativas realizada pelo Google, mídias tradicionais de diversos países e outras organizações foi a criação do *First Draft News*, com o objetivo de apresentar notícias adequadas aos usuários e combater as notícias falsas. Além disso, o *First Draft News* também realiza o monitoramento de eleições, como as que ocorreram nos Estados Unidos em 2016 e no Reino Unido, França e Alemanha em 2017.

Outra iniciativa realizada pelo Google foi a implementação de um selo de verificação nas páginas, junto com outras 115 organizações de verificação de fatos, a partir do selo é possível visualizar se a notícia é falsa a partir da checagem das organizações parceiras (DELMAZO; VALENTE, 2018).

Segundo Delmazo e Valente (2018), a disseminação de *fake news* afeta a distinção do que é real na sociedade e o que é falso, ocasionando uma ameaça não apenas ao jornalismo, mas também à democracia. Deste modo, um desafio é que a própria audiência contraponha as falsas notícias, compreenda os métodos de apuração jornalística e que compartilhe e usufrua de veículos de comunicação que informam com credibilidade e precisão.

2.1.3 Checagem de fatos

No jornalismo, a notícia é o principal produto, que é sustentado a partir da necessidade de percepção dos seres humanos, o que acontece na cidade, no país, do outro lado do mundo. Além disso, o jornalismo possui um compromisso com a verdade, desta forma existe consentimento entre os jornalistas sobre a importância de apurar bem os fatos, com exatidão, equidade e verdade (DELMAZO; VALENTE, 2018).

De acordo com Delmazo e Valente (2018), a verificação dos fatos (*fact-checking*), desenvolvida de acordo com os procedimentos já especificados na reportagem jornalística, permite que a política não se afaste do que deveria ser, que não se afaste da sua relação com a verdade. O *fact-checking* possui preocupação com a transparência, credibilidade, busca pela diversidade de personalidades checadas e uma política clara de erros.

Duas grandes iniciativas para a verificação de fatos são o *Fact Checker* e o *PolitiFact*. O *Fact Checker* é uma coluna que teve início em setembro de 2007, especificamente para a campanha presidencial de 2008 e, no ano de 2011, o jornal *Washington Post* a estabeleceu como permanente. O *PolitiFact*, lançado em 2007, é um projeto independente do jornal da Flórida *Tampa Bay Times*, sendo adquirido pelo *Poynter Institute* no ano de 2018, uma escola sem fins lucrativos para jornalistas (VETRITTI, 2019).

Segundo Vetritti (2019), nos Estados Unidos, com o lançamento do site *Factcheck.org*, o gênero *fact-checking* começou a conquistar reconhecimento, audiência e constituir uma nova forma de fazer jornalismo na era digital. Além disso, no ano de 2009, o *PolitiFact* recebeu o Prêmio *Pulitzer* pela cobertura da campanha presidencial de 2008, sendo a primeira vez em que o prêmio foi concedido a uma iniciativa que surgiu online.

Delmazo e Valente (2018) destacam que os responsáveis pela checagem de fatos em todo mundo possuem uma norma internacional, o *International Fact-checking Network* – IFCN, que possuí um código de princípios, uma conferência global realizada todo ano e um dia internacional, o dia do *fact-checking* em 2 de abril, depois do dia da mentira.

O IFCN, possui em abril de 2020, um total de 76 signatários verificados de acordo com código de princípios da IFCN, sendo duas agências do Brasil, Agência Lupa e Aos Fatos. Todas as agências credenciadas devem estabelecer compromissos de apartidarismo, equidade, transparência das fontes, detalhes sobre métodos utilizados e correções francas e amplas (IFCN, 2020).

O IFCN possui 12 signatários verificados em renovação, que são avisados um mês antes da data de vencimento com um período de três meses para concluir seu processo de renovação, estando sujeito a uma nova avaliação e aprovação pelo conselho consultivo da IFCN em referência ao código de princípios. Um dos 12 signatários em renovação, sendo o único do Brasil, é a agência Estadão Verifica. Além disso, o IFCN possui 15 signatários expirados, sendo um do Brasil, a Agência Público – Truco (IFCN, 2020).

A primeira agência de notícias do Brasil a se especializar na técnica jornalística mundialmente conhecida como *fact-checking* é a agência Lupa, que se encontra ativa desde novembro de 2015. Possui como objetivo acompanhar o noticiário de política, economia, cidade, cultura, educação, saúde e relações internacionais, realizando correções e divulgando dados corretos quando necessário (LUPA, 2015).

A agência Lupa já produziu checagens em formatos de texto, áudio e vídeo e, além disso, realizou verificações em jornais, revistas, rádios, sites, canais de televisão e redes sociais, tanto no Brasil quanto no exterior. Possui como uma de suas missões estimular o debate público, por meio de dados e informações precisas baseadas em fontes oficiais que possam ser checadas. Dessa forma, a Lupa não utiliza fontes anônimas em seu trabalho e permite que seus leitores tenham acesso a todos os bancos de dados utilizados para as checagens (LUPA, 2015).

A agência Aos Fatos, possui bases no Rio de Janeiro e em São Paulo e é mantida por uma equipe de profissionais multidisciplinares e multitarefas. Possui como objetivo acompanhar declarações de políticos e autoridades de expressão nacional verificando se estão falando a verdade. Aos Fatos é financiada por meio do seu programa de apoiadores, o Aos Fatos Mais. Além disso, parcerias editoriais e projetos de tecnologia incubados são realizados a partir do Aos Fatos Lab (AOS FATOS, 2020).

Dessa forma, com base em todos os impactos que uma *fake news* pode causar e as poucas formas de detecção dessas falsas notícias na língua portuguesa, as Redes Neurais Artificiais possuem um grande potencial nessas detecções, devido ao fato de existirem diversos trabalhos correlatos com o foco na detecção, entretanto em outras línguas, e o fato das RNAs serem muito boas trabalhando com textos.

2.2 REDES NEURAIS ARTIFICIAIS

Uma Rede Neural Artificial (RNA) é um modelo que simula o comportamento e o funcionamento do cérebro humano (FINOCCHIO, 2014). É capaz de realizar problemas altamente complexos por meio de um processamento distribuído entre várias pequenas unidades interligadas, chamadas de neurônios (FURTADO, 2019). Porém, com uma quantidade de neurônios muito menor quando comparado ao cérebro humano (FINOCCHIO, 2014).

Segundo Furtado (2019), as conexões entre os neurônios possuem um determinado valor, denominado peso da conexão ou sinapse, no qual determinada o grau de conectividade entre os neurônios. Além disso, cada neurônio realiza um processamento de forma isolada e paralela aos outros, com isso o resultado é encaminhado aos próximos neurônios através das conexões.

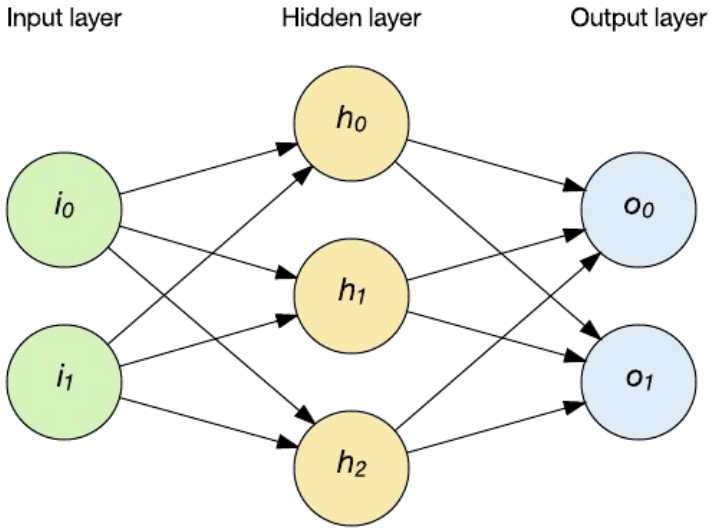
A aptidão de uma RNA na resolução de um problema é determinada através da sua arquitetura, no qual é especificado a quantidade e forma de ligação dos neurônios, assim como o peso das conexões e o número de camadas (FURTADO, 2019).

Geralmente, a forma de resolução de um problema é baseada no método empírico, a partir dos processos de treinamentos e modificações graduais para adaptar e aperfeiçoar o modelo a solução do problema (FURTADO, 2019).

Segundo Nelson (2017), os dados que serão processados são passados às unidades de entrada, os valores são multiplicados pelos respectivos pesos e passados em sequência às unidades com as quais possuem conexões diretas. Além disso, as redes neurais normalmente são organizadas em camadas, com cada camada possuindo uma ou mais unidades, e as saídas das unidades são combinadas, utilizando a função de ativação, e usadas como entradas das unidades da próxima camada.

Redes neurais comuns se baseiam na arquitetura denominada como *feedforward*, no qual representa um fluxo de informações unidirecional, não existindo nenhum tipo de ciclo (NELSON, 2017), como apresentado na figura abaixo, uma rede *feedforward* totalmente conectada com uma camada oculta e uma camada de saída (HAYKIN, 2009).

Figura 1 – Rede neural feedforward



Fonte – Jones (2017)

Furtado (2019) relata que as Redes Neurais Artificiais possuem três fases principais para resolução de um problema: Treinamento, no qual a partir de padrões de entrada o modelo treina, tenta resolver o problema e gera padrões de saída; Teste, os padrões de entrada são apresentados ao modelo e as saídas obtidas são comparadas às saídas desejadas; Aplicação, o modelo é utilizado para resolver determinado problema.

A primeira rede neural relatada em documentos datam o ano de 1943, escritos por McCulloch e Pitts. No qual propuseram um modelo de neurônio com uma unidade de processamento binária, sendo um modelo simples, mas de grande importância para os próximos anos (FINOCCHIO, 2014).

No ano de 1949, Donald O. Hebb apresentou uma hipótese de como a força das sinapses no cérebro se modificam em resposta à experiência dos seres vivos. As conexões entre células que são ativadas ao mesmo tempo tendem a se fortalecer e as outras conexões tendem a se enfraquecer. Com isso, Donald O. Hebb foi o primeiro que propôs uma lei de aprendizagem para as sinapses dos neurônios e auxiliou como inspiração para outros pesquisadores (FINOCCHIO, 2014).

De acordo com Finocchio (2014), em 1957 surgiu o primeiro neuro computador que obteve sucesso, chamado Mark I Perceptron, desenvolvido por Frank Rosenblatt, Charles Wightman e outros pesquisadores. O neuro computador possui como principal objetivo o reconhecimento de padrões. Entretanto, redes básicas como a perceptron, possuem algumas limitações básicas, apesar de serem capazes de executar as operações booleanas ‘e’ e ‘ou’ por

exemplo, não são capazes de implementar outras regras lógicas simples, como o caso do ‘ou exclusivo’.

Finocchio (2014) descreve que nos anos seguintes houve uma queda de financiamentos para pesquisas sobre redes neurais, sendo um dos motivos devido a alta expectativa criada e os baixos resultados obtidos. Entretanto, no ano de 1980, houve um crescimento do número de pesquisas realizadas e aplicações em sistemas reais, tudo isso devido a diversos fatores, dentre os principais podem ser citados:

a) Neurofisiologistas com maior conhecimento sobre o processamento de informações nos organismos vivos;
b) Novas tecnologias permitindo maior potencial computacional e baixo custo, o que possibilitou simulações e testes mais complexos;
c) Novas teorias que serviram de base para o desenvolvimento de novos algoritmos.

Assim como o cérebro humano, as RNAs possuem capacidade de interagir com o meio externo e adaptar-se a ele. Esse motivo é o que permite que essas redes sejam utilizadas em diversas áreas de aplicação (FINOCCHIO, 2014), como engenharia, economia, agronomia, medicina. Resolvendo problemas como: Extração de características; Classificação; Categorização; Estimativa; Previsão; entre outros problemas (FURTADO, 2019).

2.2.1 Neurônio biológico

O sistema nervoso é o responsável por coordenar as atividades dos vários tipos de tecidos e órgãos dos seres vivos, no qual cada um possui determinadas funções e responsabilidades. Como o tecido nervoso, no qual é constituído por dois componentes principais, que são os neurônios e as neuróglias (FURTADO, 2019).

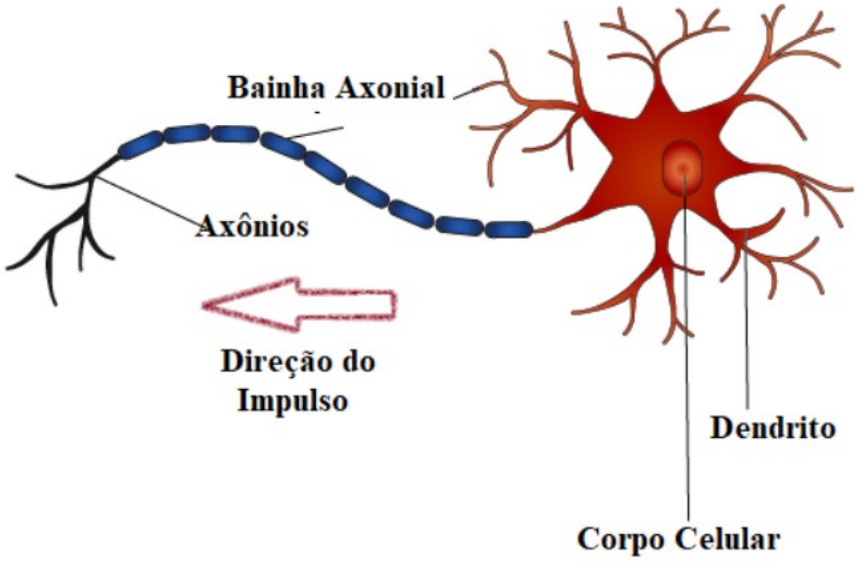
De acordo com Furtado (2019), a neuróglia é composta por diversos tipos celulares que se apresentam entre os neurônios, sua função é a sustentação, nutrição e defesa nos neurônios. Além disso, também possui outras duas funções, na vida embrionária, no qual estabelece sinapses adequadas para o crescimento dos dendritos e axônios. Na vida adulta, no qual possibilita a formação de circuitos independentes que auxiliam os impulsos a serem espalhados corretamente.

Os neurônios são células que reagem a estímulos do meio no qual se encontram, por meio da alteração na diferença de potencial elétrico existente na superfície interna e externa da membrana celular. Por meio desses estímulos, a mudança de potencial é propagada

a outros neurônios, músculos e glândulas (FURTADO, 2019). Os neurônios possuem, desta forma, um papel essencial no funcionamento, comportamento e raciocínio dos seres vivos (FINOCCHIO, 2014).

A figura abaixo apresenta a estrutura de um neurônio, que é composto por um corpo celular, onde se encontra o núcleo do neurônio. Do corpo celular partem os dendritos, responsáveis por receber estímulos. O axônio é um prolongamento responsável por transmitir os impulsos com informações provenientes do corpo celular a outros neurônios, músculos e glândulas (FURTADO, 2019).

Figura 2 – Estrutura de um neurônio biológico



Fonte – Furtado (2019)

O envio e recebimento dos impulsos entre os neurônios dependem das sinapses, que são uma estrutura no qual ocorre o contato entre um axônio de um neurônio e um dendrito de outro neurônio, podendo ser excitatórias ou inibitórias (FURTADO, 2019).

De acordo com Finocchio (2014), o cérebro humano, uma das partes do sistema nervoso, é responsável pelo controle das funções e dos movimentos do organismo, no qual é composto por aproximadamente 100 bilhões de neurônios e cada neurônio se encontra conectado a outros (aproximadamente 100 neurônios) através de sinapses, formando uma grande rede neural biológica, permitindo uma grande capacidade de processamento e armazenamento de informações.

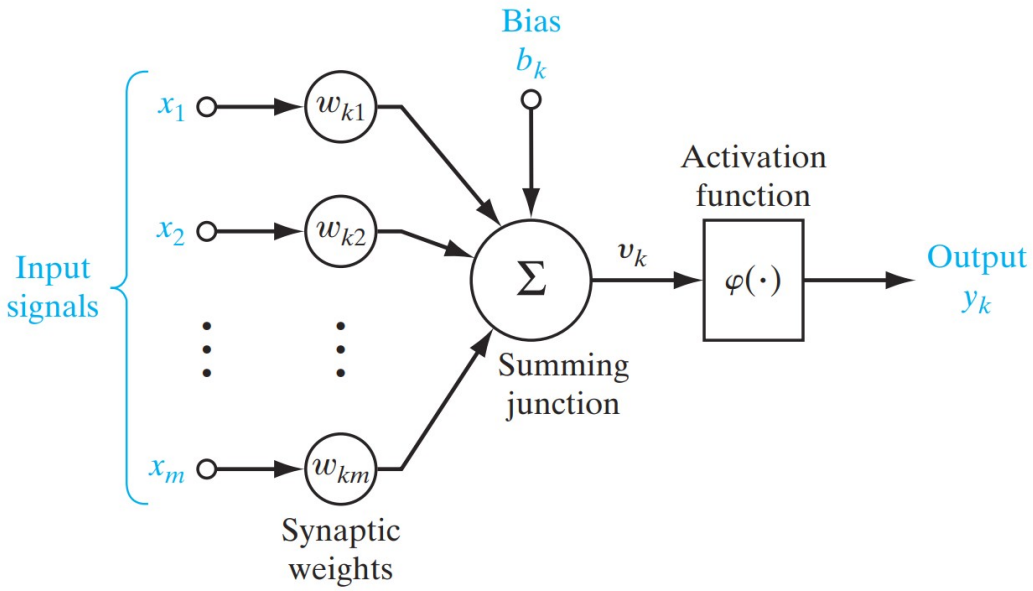
2.2.2 Neurônio Artificial

Um neurônio artificial é uma estrutura lógico-matemática que tem por objetivo simular o comportamento e funcionamento de um neurônio biológico, realizando o processamento a partir de diversas entradas e fornecendo saídas (FURTADO, 2019).

Comparando um neurônio biológico com um artificial, como pode ser observado na figura abaixo, observa-se três elementos básicos, que são:

1. Entrada e Pesos: Os dendritos seriam os sinais de entradas (*Input signals*, representados por $x_1, x_2, \ldots, x_m$), no qual se comunicam com o corpo celular através de canais que possuem pesos sinápticos (*Synaptic weights*, representados por $w_{k1}, w_{k2}, \ldots, w_{km}$), neste caso os pesos são as sinapses do neurônio biológico (FURTADO, 2019). Cada entrada possui um peso ou força própria, um sinal x_m na entrada da sinapse m conectado ao neurônio k é multiplicado pelo peso sináptico w_{kj} (Neste caso, o k simboliza o neurônio em questão e m refere-se a sinapse à qual o peso se refere) (HAYKIN, 2009).
2. Função Soma: Após os estímulos serem captados, são processados a partir da função soma (*Summing junction*) (FURTADO, 2019). No qual, é realizada a soma dos sinais de entrada, ponderados pelas respectivas forças sinápticas do neurônio, representando uma combinação linear (HAYKIN, 2009).
3. Função de Ativação: Responsável por limitar a amplitude da saída de um neurônio, limita a faixa de amplitude permitida do sinal de saída para algum valor finito.

Figura 3 – Estrutura de um neurônio artificial



Fonte – Haykin (2009)

De acordo com figura acima, observa-se também que a função soma possui um viés aplicado externamente, o limiar de ativação chamado de Bias, representado por b_k . Permite aumentar ou diminuir a entrada líquida da função de ativação, dependendo se é positivo ou negativo, respectivamente (HAYKIN, 2009).

Segundo Haykin (2009), o modelo da figura acima, na forma matemática pode ser representado a partir das equações 1 e 2 apresentadas abaixo.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

$$y_k = \phi(u_k + b_k) \tag{2}$$

No qual, de acordo com a figura 3 e as equações 1 e 2: Os valores de $x_1, x_2, \ldots, x_m$ são os sinais de entrada; os valores representados por $w_{k1}, w_{k2}, \ldots, w_{km}$ são os pesos sinápticos; u_k é o resultado da combinação linear dos sinais de entrada e os pesos; b_k é o *bias*; A fórmula 2 representa a função de ativação; e y_k é o resultado do neurônio.

2.3 RNA'S UTILIZADAS

Neste capítulo é apresentado as Redes Neurais Artificiais utilizadas neste trabalho. Primeiramente é apresentado a rede neural MLP e posteriormente é apresentado a rede neural recorrente LSTM.

2.3.1 Multilayer Perceptron

De acordo com Gualda (2008), as redes neurais artificiais são diferenciadas quando comparadas a outras técnicas, devido ao aprendizado que elas possuem. No qual, os valores são propagados pela rede através de pesos de conexão entre os neurônios, os pesos de conexão são ajustados através de um método de aprendizado, por exemplo o método de retropropagação.

As RNAs que possuem duas camadas, sendo uma camada de entrada e outra de saída possuem um desempenho muito limitado, o que torna necessário a adição de uma camada intermediária. Além disso, uma rede com três camadas é suficiente para representar qualquer função ou problema de classificação (GUALDA, 2008).

De acordo com Rodriguez (2019), a dificuldade das redes neurais em resolver problemas linearmente não separáveis, por exemplo, o XOR (ou exclusivo), é uma das principais limitações dos modelos *perceptrons*. Com isso, novas estratégias foram utilizadas para combater esse problema, como adicionar camadas intermediárias a partir da criação da rede *multilayer perceptron* – MLP.

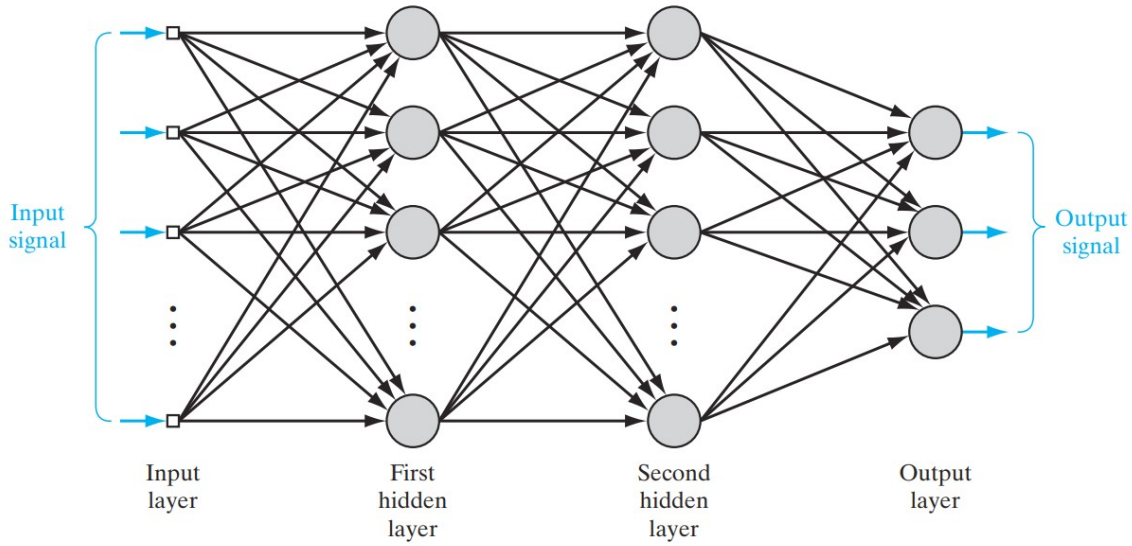
As redes MLP possuem um poder computacional muito maior do que redes sem camadas intermediárias e permite a solução de problemas onde as classes são não-linearmente separáveis, adicionando camadas ocultas para o processamento e utilizando uma função de ativação não linear (GUALDA, 2008).

O ajuste dos pesos nas camadas ocultas é realizado através do aprendizado supervisionado, para um conjunto de amostra de entrada existe uma respectiva saída. Além disso, cada neurônio de uma camada oculta ou de saída é projetado para calcular a saída de um sinal por meio de uma função não linear e calcular o valor do gradiente para minimizar o erro a partir do algoritmo de retropropagação (*backpropagation*) (RODRIGUES, 2019), baseado numa regra de aprendizagem que ajusta o erro durante o treinamento (GUALDA, 2008).

A figura abaixo apresenta a arquitetura de uma rede *multilayer perceptron* com uma camada de entrada, duas camadas ocultas e uma camada de saída (HAYKIN, 2009). A

partir desta configuração, cada neurônio está ligado com todos os outros das camadas vizinhas em uma comunicação unidirecional, contudo neurônios da mesma camada não se comunicam (GUALDA, 2008). Os estímulos / sinais são apresentados à rede pela camada de entrada e são propagados para frente sem nenhuma alteração (*feedforward*) (RODRIGUES, 2019).

Figura 4 – Arquitetura de uma rede multilayer perceptron com duas camadas intermediárias



Fonte – Haykin (2009)

Haykin (2009) descreve de acordo com observações da figura acima, os seguintes pontos: Os sinais de entrada (*input signal*) chegam a partir da camada de entrada (*input layer*) da rede e se propagam adiante, neurônio por neurônio. Na camada de saída (*output layer*) resultam os sinais de saída, que desempenham uma função útil na saída da rede e de acordo como atravessam a rede, o sinal é calculado como uma função das entradas e pesos associados aplicados a cada neurônio. Um sinal de erro se origina no neurônio de saída da rede e se propaga para trás (camada por camada) através da rede.

2.3.2 Redes Neurais Recorrentes

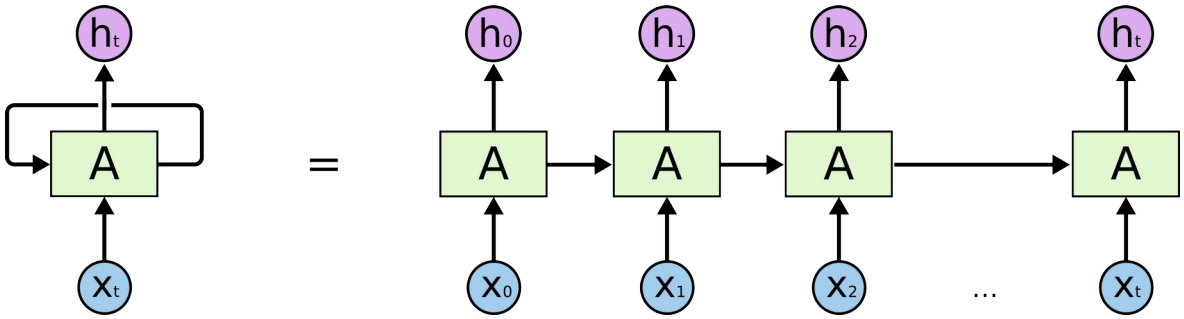
Os seres humanos ao realizarem, por exemplo, a leitura de um livro, entendem cada palavra com base na compreensão das palavras anteriores. Os pensamentos possuem persistência, ou seja, as palavras lidas não são ignoradas e não são jogadas fora e iniciado o pensamento do zero novamente. Redes neurais tradicionais não possuem essa capacidade de persistência das informações, desta forma surgem as redes neurais recorrentes, redes com

loops que possibilitam que as informações persistam e sejam passadas de uma etapa da rede para a próxima (OLAH, 2015).

Segundo Rodriguez (2019), redes neurais artificiais mais simples, como a *multilayer perceptron*, recebem os sinais a partir da camada de entrada e atravessam a rede até a camada de saída (*feedforward*). Entretanto, em uma rede neural recorrente a camada oculta pode receber tanto informações da camada de entrada quanto informação dela mesma na interação anterior (*feedback*), o que permite, por exemplo, trabalhar com modelagens temporais e espaciais.

De acordo com Olah (2015), redes neurais recorrentes são redes com *loops*, entretanto também podem ser vistas como várias cópias da mesma rede, cada uma passando uma mensagem para um sucessor. Ou seja, uma parte da rede neural analisa uma entrada X_t , com isso um resultado h_t é obtido como resposta da etapa atual, mas também é persistido e passado para a próxima etapa da rede, como apresentado na figura abaixo.

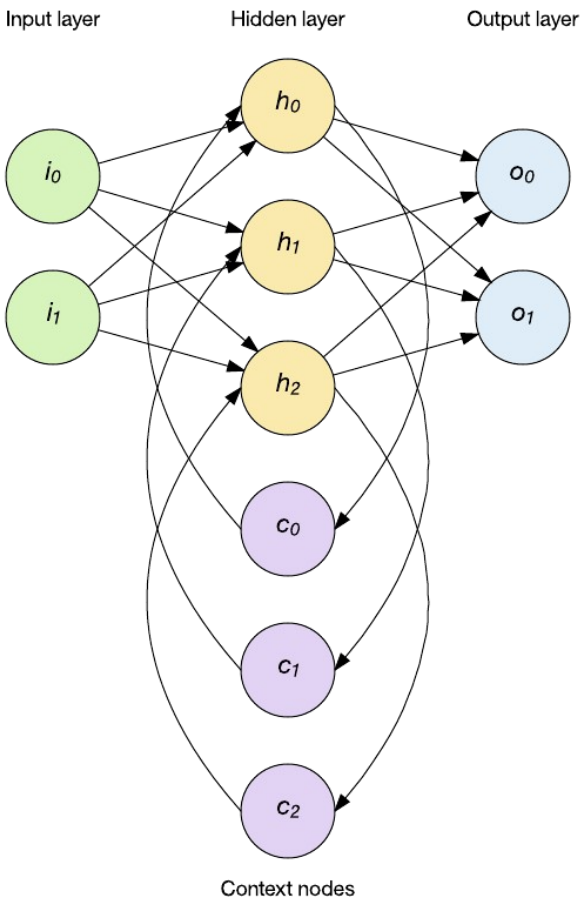
Figura 5 – Funcionamento de uma rede neural recorrente



Fonte – Olah (2015)

Dessa maneira, a informação não flui em um único sentido. Além disso, a saída da rede não depende apenas da entrada corrente, mas também das entradas anteriores, como pode ser visualizado na figura abaixo, onde c_0, c_1 e c_2 são informações dos próprios neurônios da iteração anterior. O efeito prático disto é a existência de memória de curto prazo na rede (NELSON, 2017).

Figura 6 – Rede neural feedback



Fonte – Jones (2017)

O processo de *feedback* permite que o estado oculto passe a funcionar como uma memória de curto prazo através dos *feedbacks* existentes na rede. Com isso, o estado oculto pode fornecer informações para a camada de saída a fim de realizar uma previsão bem como fornecer informações para o estado oculto do próximo passo (RODRIGUEZ, 2019). Redes neurais recorrentes podem criar modelos mais complexos, com uma compreensão mais difícil e com a capacidade de resolver uma gama maior de problemas (NELSON, 2017).

Segundo Olah (2015), a partir de informações das etapas anteriores, as RNNs são capazes de compreender a etapa atual. Por exemplo, a frase “As nuvens estão no”, prever que a próxima palavra será “céu” não é difícil, devido ao fato da diferença entre as informações importantes e o local necessário ser pequeno, neste caso a rede é capaz de aprender a partir das informações passadas.

No entanto, utilizando a frase “Eu cresci na França, no dia 14 de maio, tenho 15 anos e falo”, provavelmente a próxima palavra será o nome de um o idioma, o “francês”, para

descobrir qual o idioma é necessário possuir o contexto “França” no final da frase para descobrir a palavra, a lacuna entre as informações relevantes e o ponto em que é necessário tornar-se muito grande, surgindo o problema das dependências de longo prazo (OLAH, 2015).

Na teoria, as RNNs são capazes de lidar com as dependências de longo prazo, assim como um ser humano é capaz de escolher cuidadosamente parâmetros para resolver problemas. Na prática, as RNNs não parecem capazes de aprendê-las, entretanto as redes *Long Short-Term Memory* conseguem lidar com esse problema e são apresentadas no próximo capítulo (OLAH, 2015).

2.3.2.1 Long Short-Term Memory

Segundo Hochreiter e Schmidhuber (1997), redes neurais recorrentes por meio de suas conexões de *feedback* podem armazenar representações de eventos de entrada recentes em forma de ativações, chamadas de memória de curto prazo (*short-term memory*). O que permite a sua utilização em diversas aplicações, como o processamento de fala e composição musical.

Um grande problema relacionado a redes neurais recorrentes são os algoritmos utilizados para aprender o que colocar na memória de curto prazo, no qual levam muito tempo ou não funcionam muito bem (HOCHREITER; SCHMIDHUBER, 1997).

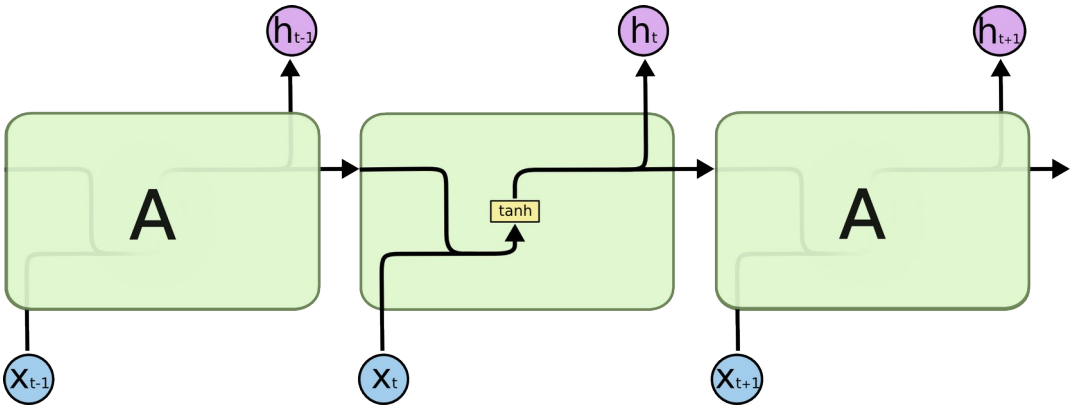
Utilizando aprendizagem recorrente em tempo real, sinais de erro que retrocedem no tempo tendem a explodir ou desaparecer, a evolução temporal do erro da retropropagação depende exponencialmente do tamanho dos pesos, o que pode levar, por exemplo, a pesos oscilantes ou a tempos que inválidos (HOCHREITER; SCHMIDHUBER, 1997).

A arquitetura recorrente chamada Memória de Longo Prazo (*Long Short-Term Memory* – LSTM), proposta por Hochreiter e Schmidhuber (1997), foi desenvolvida para superar esses problemas de retorno de erros. As LSTMs são projetadas explicitamente para evitar o problema de dependência a longo prazo, lembrar informações por longos períodos de tempo é a principal característica dessa arquitetura (OLAH, 2015).

As LSTMs aprendem a diminuir os intervalos de tempo em excesso de 1000 etapas, mesmo no caso de sequências de entrada barulhentas e incompressíveis. Além disso, é um algoritmo eficiente e baseado em gradiente para uma arquitetura que aplica erros constantes, impedindo que explodam e desapareçam (HOCHREITER; SCHMIDHUBER, 1997).

Segundo o autor Olah (2015), redes neurais recorrentes possuem o formato de uma cadeia de módulos repetidos de rede neural, como exemplificado na imagem abaixo, no qual o X é a entrada da rede, h é a resposta e t representa em qual iteração a rede se encontra. Além disso, nas RNNs padrões o módulo de repetição possui uma estrutura muito simples, como uma única camada *tanh*.

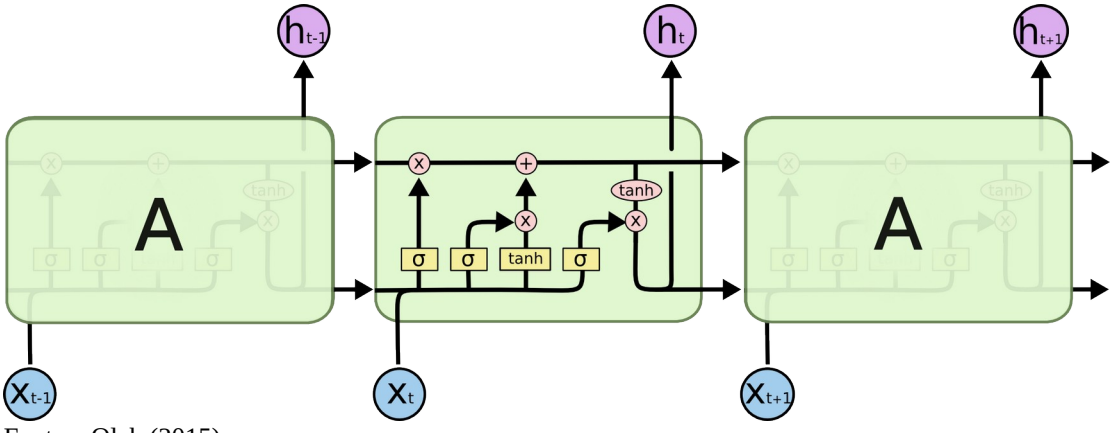
Figura 7 – Módulo de repetição em uma RNN padrão



Fonte – Olah (2015)

Diferentemente das uma RNNs padrões, as LSTMs possuem quatro camadas no módulo de repetição, como apresentado na figura abaixo, por meio dos retângulos amarelos. Além disso, cada linha carrega um vetor inteiro, os círculos na cor rosa representam operações pontuais, as linhas mescladas representam concatenação e as linhas com bifurcações representam que o valor é encaminhado para mais de um local (OLAH, 2015).

Figura 8 – Módulo de repetição em uma LSTM



Fonte – Olah (2015)

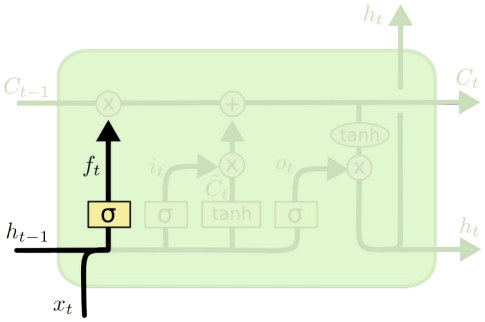
De acordo com Olah (2015), uma rede LSTM possui a capacidade de remover ou adicionar informações ao estado da célula, representado pela linha horizontal na parte superior

da figura apresentada acima. Além disso, o estado da célula é regulado a partir de estruturas chamadas portões.

A camada *sigmoide* gera números entre zero e um, que representam a probabilidade de cada componente ser liberado. A rede LSTM possui três portões com a camada *sigmoide*, para proteger e controlar o estado da célula (OLAH, 2015).

O primeiro passo realizado na rede LSTM é exercido pela camada *sigmoide* chamada de "portão do esquecimento" (*forget gate*), no qual são determinados quais informações serão removidas do estado da célula. Como pode ser visualizado na figura abaixo, é obtido a saída da última unidade LSTM ($h_{(t-1)}$) no tempo $t-1$ e a entrada atual (x_t) no tempo t . A função *sigmoide* determina qual parte da saída antiga deve ser eliminada e, com isso, f_t é representado a partir de um vetor com valores entre 0 e 1 para cada número no estado da célula $C_{(t-1)}$, no qual o valor um mantém a informação e zero realiza a exclusão da informação (LE et al., 2019).

Figura 9 – Camada *sigmoide* (Portão do esquecimento)



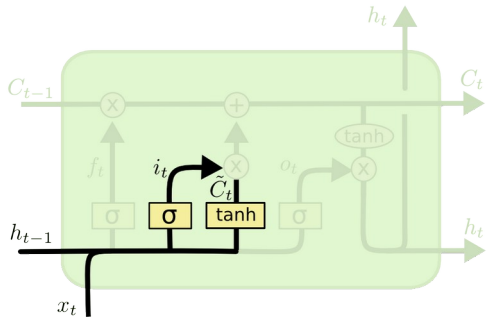
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fonte – Olah (2015)

De acordo com a equação da camada do portão do esquecimento da figura acima, σ é a função *sigmoide*, W_f são as matrizes de peso e b_f é o *bias*.

De acordo com Olah (2015), o segundo passo é realizado pela camada *sigmoide* chamada de "portão de entrada" (*Input Gate*), no qual são determinados quais valores serão atualizados a partir do valor um ou zero. Posteriormente, a camada *tanh* cria um vetor de novos valores candidatos, chamado $\widetilde{C}_t$, atribui pesos aos valores que passaram e define um nível de importância, variando entre um e menos um, como apresentado na figura abaixo.

Figura 10 – Camada *sigmoide* (Portão de entrada) e camada *tanh*



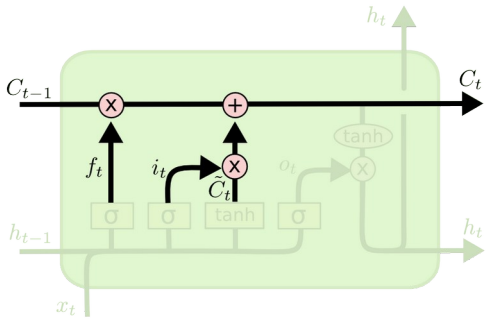
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fonte – Olah (2015)

De acordo com as equações acima, o $\widetilde{C}_t$ é o vetor de valores candidatos, W são as matrizes de peso e b é o *bias*.

A próxima etapa é responsável por atualizar o antigo estado da célula $C_{(t-1)}$ para o novo estado da célula C_t . Primeiro, é realizado a multiplicação do antigo estado $C_{(t-1)}$ pelo f_t , com o objetivo de retirar o que deve-se esquecer. Em seguida, é somado com $i_t * \widetilde{C}_t$ gerando os novos valores dimensionados o quanto foi decidido atualizar cada valor de estado (OLAH, 2015).

Figura 11 – Atualização do antigo estado da célula



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

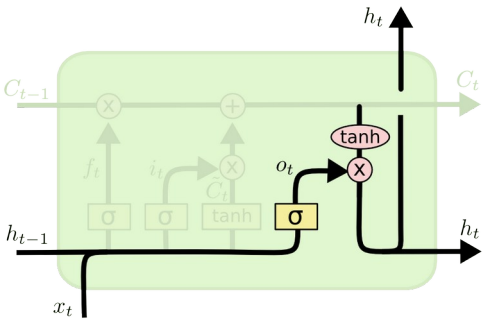
Fonte – Olah (2015)

De acordo com as equações acima, C_t é estado da célula no tempo t , $C_{(t-1)}$ é estado da célula no tempo $t-1$, W são as matrizes de peso e b é o *bias*.

Na última etapa, os valores de saída (h_t) são baseados no estado da célula de saída (o_t), entretanto não estão filtrados. Com isso, é executada a camada *sigmoide* chamada de "portão de saída" (*Output gate*) que decide quais partes do estado da célula chegam à saída. Em seguida, a saída da porta *sigmoide* (o_t) é multiplicada pelos novos

valores criados pela camada *tanh* a partir do estado da célula (C_t), com um valor que varia entre -1 e 1 , como visualizado na figura abaixo (LE et al., 2019).

Figura 12 – Camada *sigmoide* (Portão de saída)



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fonte – Olah (2015)

De acordo com as equações acima, W_o e b_o são as matrizes de peso e o *bias*, ambas do portão de saída.

3 TRABALHOS CORRELATOS

No decorrer deste trabalho realizou-se uma revisão bibliográfica da literatura com o objetivo de identificar o estado da arte de modelos e métodos utilizados para a detecção de *fake news*. Deste modo, foi possível identificar as principais características utilizadas nas detecções, tais como: Variáveis utilizadas nos modelos; Arquiteturas de Redes Neurais utilizadas; Métricas de erro para análise de qualidade da detecção.

Nas próximas seções, são elucidados os trabalhos selecionados e comparados às suas principais características que respondem as perguntas de pesquisa da revisão sistemática.

3.1 AUTOMATIC IDENTIFICATION OF FAKE NEWS USING DEEP LEARNING

Adbullah, Qawasmeh e Tawalbeh (2019) desenvolveram dois modelos com base no conjunto de dados fornecido pelo *Fake News Challenge* (FNC), sendo um o conjunto correspondente a um total de 49.972 pares de títulos e textos (na língua inglesa), separados em 60% para treinamento, 20% para validação e 20% para teste.

Segundo os autores, as entradas foram transformados em espaço vetorial usando o modelo vetorial de quatro palavras do Google *News* pré treinado com 300 dimensões, retirado de 3 bilhões de palavras em execução da Wikipédia, além disso também foi utilizado a biblioteca Gensim para o mapeamento das palavras em vetores de números reais como pesos iniciais.

Após diversos testes em diferentes arquiteturas de redes neurais artificiais, os dois modelos que apresentaram resultados satisfatórios são o LSTM bidirecional (*Bidirectional LSTM*) e o LSTM de várias cabeças (*Multi-head* LSTM). O modelo LSTM bidirecional apresentou a maior *accuracy*, com 85% e precisão de 77%, o segundo modelo é o LSTM de várias cabeças gerou uma *accuracy* de cerca de 83% e precisão de 84,5%. Em termos de precisão, o modelo LSTM apresentou o maior valor, com 88% (ADBULLAH; QAWASMEH; TAWALBEH, 2019).

3.2 FIND: FAKE INFORMATION AND NEWS DETECTIONS USING DEEP LEARNING

Verma, Mittal e Dawn (2019) comentam que para o conjunto de dados de notícias possam ser aproveitados nos algoritmos, foi necessário o uso da biblioteca NLTK (*Natural Language Toolkit*). Os dados textuais são compostos por muitas palavras irrelevantes,

chamadas de *Stopwords*, que não influenciam o contexto da frase e nem ajudam na classificação do contexto, como “isto”, “de”, “ou”, “e”. Essas palavras podem ser removidas para melhorar o desempenho do algoritmo.

Além da biblioteca NLTK, também foi utilizada a biblioteca *FastText* para incorporação de palavras, divisão das palavras em n-gramas, permitindo que as palavras sejam representadas corretamente a partir de n-gramas de outras palavras. Foi utilizado a tokenização de texto de pré-processamento do *keras*, no qual cada texto é transformado em um vetor utilizado para criar uma matriz de palavras incorporadas para servirem de pesos padrão para a rede neural (VERMA; MITTAL; DAWN, 2019).

Segundo os autores, o conjunto de dados (na língua inglesa) em sua grande parte possui apenas notícias indianas e foi buscado a partir dos sites: *Faking News*, *Rising Kashmir e Germany investigating unprecedented spread of fake news online*.

De acordo com Verma, Mittal e Dawn (2019), para a avaliação dos resultados foi utilizado a matriz de confusão e um conjunto de dados de 20.000 artigos de notícias, o modelo LSTM mostrou uma precisão de 94,3% e perda de 0,209. O modelo GRU (Unidades Recorrentes Raladas – *Grated Recurrent Units*) gerou uma precisão de 91,9% e perda de 0,011.

Após o primeiro teste, o modelo que apresentou o melhor resultado, neste caso o modelo LSTM, foi utilizado em um segundo teste com um conjunto de dados de 72.000 notícias, no qual obteve uma precisão de 99,04% e perda de 0,11165. 88% (VERMA; MITTAL; DAWN, 2019).

3.3 DEEP LEARNING PARA CLASSIFICAÇÃO DE FAKE NEWS POR SUMARIZAÇÃO DE TEXTO

Segundo Marumo (2018), para a construção do conjunto de dados foi buscado notícias verdadeiras a partir do site de notícias da Universidade Estadual de Londrina, a Agência UEL e o site do G1. Já as falsas notícias foram buscadas a partir de sites como o Diário de Pernambuco e o Sensacionalista.

De acordo com Marumo (2018), para o desenvolvimento do trabalho primeiro foi necessário realizar a mineração do texto, com o objetivo de identificar padrões e extrair informações úteis e implícitas para facilitar o entendimento dos documentos textuais, esse processo pode ser dividido em: Obtenção dos dados; pré-processamento / limpeza dos dados;

transformação dos dados a partir técnicas de conversão para representação numérica; mineração dos dados / aplicação dos algoritmos; Avaliação.

Foi realizado o pré-processamento dos dados, uma etapa de limpeza no texto para garantir a qualidade dos dados e diminuir o tempo de processamento. Todas letras foram convertidas para letras minúsculas, foram removidos dígitos de números, utilizando *Gensim* e *stopwords* utilizando NLTK. Além disso, também foi aplicado a técnica de sumarização, criação de resumos, não se limitando apenas a análise de algumas frases, a partir de abordagens como *Extract* e *Abstract* (MARUMO, 2018).

Segundo o autor, para a transformação dos dados foi utilizado a técnica *Word2Vec* e um dos seus respectivos métodos de *Word Embedding*, representação vetorial das palavras que permite que palavras de significados parecidos tenham representações similares.

O conjunto de dados foi separado em 80% para treinamento e 20% para teste e a avaliação dos resultados é realizada a partir das métricas acurácia, precisão e revocação (*recall*). A tabela abaixo apresenta os resultados obtidos a partir do conjunto de dados com textos pré-processados, sendo um total de 799 notícias, no qual 286 são *fake news* e 513 são reais (MARUMO, 2018).

Tabela 1 – Resultados das RNA em textos pré-processados

	RF	SVM	LSTM	RNN
Análise e texto são FN (%)	74	69	77	24
Análise e texto não são FN (%)	82	66	81	79
Análise não é FN e texto é FN (%)	18	34	19	21
Análise é FN e texto não é FN (%)	26	31	23	74
Acurácia (%)	78.2	67.2	79.3	52.5
Precisão	0.807	0.667	0.805	0.555
Revocação	0.743	0.689	0.772	0.258

Fonte: Adaptado de Marumo (2018)

Segundo Marumo (2018), a tabela abaixo apresenta os resultados obtidos a partir do conjunto de dados com textos sumarizados, sendo um total de 609 notícias, no qual 187 são *fake news* e 422 são reais.

Tabela 2 – Resultados das RNA em textos sumarizados

	RF	SVM	LSTM	RNN
Análise e texto são FN (%)	56	67	59	39
Análise e texto não são FN (%)	69	48	52	55
Análise não é FN e texto é FN (%)	31	52	48	61
Análise é FN e texto não é FN (%)	44	33	41	45
Acurácia (%)	62.8	57.5	55.9	47.3
Precisão	0.648	0.562	0.555	0.468
Revocação	0.563	0.671	0.594	0.394

Fonte: Adaptado de Marumo (2018)

De acordo com as tabelas acima, em textos sumarizados, a RF obteve a melhor acurácia, de 62.8% seguida da SVM (*Support Vector Machine*) com 57.5%, LSTM com 55.9% e RNN (*Recurrent Neural Network*) com 47.3%. Contudo, em textos pré-processados a acurácia geral foi maior, a LSTM obteve a melhor acurácia, de 79.3% seguida da RF com 78.2%, SVM com 67.2% e, por fim, a RNN com 52.5% (MARUMO, 2018).

3.4 FAKE NEWS DETECTION USING A DEEP NEURAL NETWORK

Segundo Kaliyar (2018), para construção do algoritmo de detecção de *fake news* foram utilizadas técnicas de processamento de linguagem natural, aprendizado de máquina e aprendizado profundo, após a etapa de implementação foi realizada a comparação de quais modelos fornecerão mais precisão.

Para a construção do conjunto de dados, os artigos de notícias falsas e forjadas foram obtidos a partir da plataforma Kaggle[1]. Já os artigos de notícias verdadeiras foram obtidos a partir da companhia *Signal*[2] (KALIYAR, 2018).

De acordo com Kaliyar (2018), para preparação do conjunto de dados foram utilizadas diversas ferramentas, como a extração de recursos, a partir do n-grama e TF-IDF. Além disso, foi utilizado a incorporação de palavras e o word2vec nas redes e o uso do *select best* e chi2 para extração de recursos no modelo de aprendizado de máquina.

1https://www.kaggle.com/mrisdal/fake-news

2http://research.signalmedia.co/newsir16/signaldataset.html

Utilizando o modelo CNN, a precisão do modelo após duas épocas foi de 91,3%, após aumentar um pouco as camadas intermediárias o modelo teve sua precisão aumentada para 98,3%. Além disso, utilizando uma combinação dos modelos CNN e LSTM, resultou uma precisão de 97,3% (KALIYAR, 2018).

A tabela abaixo apresenta os resultados obtidos a partir da utilização de outros modelos para a detecção de *fake news* e seus respectivos valores (KALIYAR, 2018).

Tabela 3 – Resultados de outros modelos utilizados

	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Naïve Bayes model	89	90	90	90
Decision Tree	73	75	74	73
Random Forest	71	72	71	71
K Nearest Neighbors	53	54	53	50

Fonte: Adaptado de Kaliyar (2018)

Segundo o autor, o modelo com a melhor precisão foi o modelo CNN, com 98,3%. Entretanto, quando comparado os outros modelos, o que apresentou o melhor desempenho geral é o modelo *Naïve Bayes*, com acurácia de 89% e precisão de 90%.

3.5 EARLY DETECTION OF FAKE NEWS “BEFORE IT FLIES HIGH”

Segundo Gereme e Zhu (2019), o modelo para detecção de notícias falsas foi construído a partir do aprendizado profundo e conteúdo de notícias. Além disso, os experimentos foram realizados com base em conjuntos de dados do mundo real.

A tabela abaixo apresenta a composição do *dataset*, no qual foram utilizados dois conjuntos de dados conhecidos, Kaggle e George Mclntire. O terceiro *dataset* é uma concatenação dos dois conjuntos de dados, no qual foram embaralhados aleatoriamente para criar o conjunto de dados combinado (GEREME; ZHU, 2019).

Tabela 4 – Construção do dataset

Dataset	Origem	Fake news	Real news	Total
Kaggle	Busca a partir de 244 sites pelo BSDetecter	10,349	10,369	20,718
George Mclntire	The New York Times, WSJ, Bloomberg,	3,151	3,160	6,311

	NPR, the Guardian e Kaggle			
KaggleMclntire	União das notícias do Kaggle e do Mcnltire	13,500	13,529	27,029

Fonte: Adaptado de Gereme e Zhu (2019)

Segundo os autores, o código foi escrito em Python 3.7.0 com o auxílio das bibliotecas TensorFlow, NumPy e Keras. Além disso, para o pré-processamento dos dados, foram necessárias algumas etapas, descritas abaixo:

1. Notícias não identificadas foram deletadas;
2. As *stopwords* foram removidas;
3. Os dados de texto foram divididos com espaço, filtrado sinais de pontuação e alterado todo o texto para minúsculas a partir do método "*text_to_sequences()*" da classe Tokenizer de Keras;
4. As sequências foram geradas, preenchidas e truncadas a partir do método "*pad_sequence()*" com o parâmetro *maxlen* igual a 5000 para CNN e 500 para LSTM.

Gereme e Zhu (2019) destacam que após todo o pré-processamento, cada conjunto de dados foi dividido em 80% dos dados para treinamento e os 20% para validação. Desta forma, os resultados obtidos nos três conjuntos de dados podem ser visualizados na tabela abaixo, a partir do uso dos modelos Classificador *Naive Bayes*, *Long Short-Term Memory* (LSTM) e *Very Deep Convolutional Networks* (VDCNN).

Tabela 5 – Desempenho dos modelos

Dataset	Parâmetros	Modelos		
		Naive Bayes	LSTM	VDCNN
Kaggle	Accuracy (%)	89,64	95,67	99,04
	Recall (%)	-	94,82	98,76
	Precision (%)	-	96,57	99,33
	F1 Score (%)	-	95,67	99,05
George Mclntire	Accuracy (%)	89,82	90,89	97,32
	Recall (%)	-	90,07	95,44
	Precision (%)	-	91,74	99,18
	F1 Score (%)	-	90,90	97,28

KaggleMclntire	Accuracy (%)	89,64	90,41	94,22
	Recall (%)	-	91,40	95,55
	Precision (%)	-	89,55	93,13
	F1 Score (%)	-	90,46	94,32

Fonte: Adaptado de Gereme e Zhu (2019)

Todos os três modelos apresentaram ótimos resultados em seus testes, no entanto a melhor acurácia obtida foi no *dataset* Kaggle a partir do modelo VDCNN, com 99,04 %, enquanto o LSTM obteve 95,67 % e Naive Bayes 89,64 % (GEREME; ZHU, 2019).

De acordo com Gereme e Zhu (2019), considerando o conjunto de dados combinado e mais diversificado, KaggleMclntire, a acurácia obtida a partir do modelo VDCNN foi de 94,22 %, enquanto o LSTM obteve 90,41 % e Naive Bayes 89,64 %.

4 DESENVOLVIMENTO

Neste capítulo são apresentados e detalhados os dados disponíveis, sua forma de obtenção, os tratamentos realizados e a conversão do texto para representação numérica antes de realizar os treinamentos, validações e testes das redes neurais.

Este capítulo também descreve os recursos utilizados, envolvendo uma breve explicação das bibliotecas utilizadas na conversão do texto para representação numérica, as funções de ativação e os algoritmos de treinamento.

São apresentados os modelos de RNAs empregados, incluindo os parâmetros alterados durante o treinamento. Por último, são evidenciados as métricas de desempenho utilizadas para avaliar o desempenho dos modelos em questão e também como foi realizada a definição da melhor configuração de RNA.

4.1 DADOS DISPONÍVEIS

O conjunto de dados escolhido para esse trabalho, foi o *dataset* “Fake.Br Corpus”[3], composto por notícias 7.200 notícias, com exatamente 3.600 verdadeiras e 3.600 falsas. Além disso, todas as notícias estão escritas na língua portuguesa do Brasil (MONTEIRO et al., 2018).

Todos os textos estão sem formatação, possuem um tamanho semelhante entre si, cada um se encontra em um arquivo diferente e, além disso, todas as notícias foram buscadas em um intervalo de 2 anos, iniciado em janeiro de 2016 até janeiro de 2018 (MONTEIRO et al., 2018).

De acordo Monteiro et al. (2018), para garantir a qualidade e confiabilidade do *dataset*, as notícias falsas coletadas foram verificadas manualmente (Incluindo seus títulos), garantindo que são realmente falsas e, em seguida, foi realizado a busca de forma quase automática das notícias verdadeiras correspondentes para cada uma delas.

As notícias falsas foram retiradas de 4 sites, que são: Diario do Brasil (3.338 notícias), A Folha do Brasil (190 notícias), The Jornal Brasil (65 notícias) e Top Five TV (7 notícias). Entretanto, foram mantidas apenas as que eram inteiramente falsas e não representavam meias verdades (MONTEIRO et al., 2018).

Segundo Monteiro et al. (2018), as notícias verdadeiras foram buscadas a partir de um rastreador da web em 4 sites, que são: A saber, G1, Folha de São Paulo e Estação. A busca

3 https://github.com/roneysco/Fake.br-Corpus

foi realizada a partir de palavras-chaves e as palavras mais frequentes nos textos das notícias falsas, para que cada falsa notícia possua um correspondente verdadeiro.

Monteiro et al. (2018) destaca que as notícias do *dataset* foram rotuladas manualmente em seis grandes categorias, a tabela abaixo apresenta a distribuição dos textos por categoria.

Tabela 6 – Distribuição das notícias do dataset por categoria

Categoria	Quantidade	Quantidade (%)
Política	4.180	58,0
TV & Celebridades	1.544	21,4
Sociedade & Notícias diárias	1.276	17,7
Ciência & Tecnologia	112	1,5
Economia	44	0,7
Religião	44	0,7

Fonte: Adaptado de Monteiro et al. (2018)

É interessante destacar do *dataset*, por exemplo, que os erros de ortografia foram mais frequentes nas notícias falsas, o percentual de notícias falsas com erros é de 36,0%, enquanto nas notícias verdadeiras esse valor cai para 3,0% (MONTEIRO et al., 2018).

Monteiro et al. (2018) evidencia que cada notícia do *dataset*, além do texto e a informação se é uma falsa ou verdadeira notícia, ainda possui os seguintes metadados (Quando encontrado): “*author*”, “*link*”, “*category*”, “*date of publication*”, “*number of tokens*”, “*number of words without punctuation*”, “*number of types*”, “*number of links inside the news*”, “*number of words in uppercase*”, “*number of verbs*”, “*number of subjunctive and imperative verbs*”, “*number of nouns*”, “*number of adjectives*”, “*number of adverbs*”, “*number of modal verbs (mainly auxiliary verbs)*”, “*number of singular first and second personal pronouns*”, “*number of plural first personal pronouns*”, “*number of pronouns*”, “*pausality*”, “*number of characters*”, “*average sentence length*”, “*average word length*”, “*percentage of news with spelling errors*”, “*emotiveness*” e “*diversity*”.

4.2 TRATAMENTO DOS DADOS

Os dados obtidos foram dispostos em uma planilha e organizados em dez colunas, explicadas a seguir: ‘*ID*’, auto incremento de identificação do texto; ‘*fake_news*’,

identificação se o texto é uma falsa (1) ou verdadeira (0) notícia; ‘*text*’, o texto da notícia; ‘*author*’, nome do autor da notícia; ‘*average_word_length*’, média de caracteres das palavras do texto; ‘*category*’, categoria da notícia; ‘*date_publication*’, data de publicação; ‘link’, link da notícia; ‘*percent_speeling_errors*’, percentual de erros da notícia e ‘*number_words*’, número de palavras da notícia com o pré-processamento já realizado.

A figura abaixo apresenta o gráfico de nuvem de palavras dos textos, no qual conta a frequência com que cada palavra aparece nos textos e seta o tamanho das palavras proporcionais à frequência.

Figura 13 – Gráfico de nuvem de palavras

Fonte – Acervo do autor

Entretanto, os dados desta planilha ainda precisam ser convertidos para representação numérica para que possam ser utilizados como entrada no treinamento dos modelos.

Com a disposição dos dados já efetuada em uma única tabela, é necessário realizar um pré-processamento de cada texto, que consiste em realizar as etapas a seguir e na ordem que são citadas:

- Realizado a substituição das letras maiúsculas por minúsculas utilizando o Python;

- Atualizado os caracteres para um espaço em branco utilizando o Python, exceto os caracteres: Letras, algumas letras com acentos e o espaço (tratado mais à frente);
- Removido palavras com apenas um carácter por meio da biblioteca Pandas;
- Removido *stopwords* por meio da biblioteca Gensim, utilizando o NLTK;
- Removido múltiplos espaços por um único espaço utilizando o Python;

Após realizar as etapas de tratamento dos textos, todos os textos são armazenados em um CSV e cada texto possui o formato apresentado na tabela abaixo.

Tabela 7 – Estrutura de uma notícia com tratamento dos dados

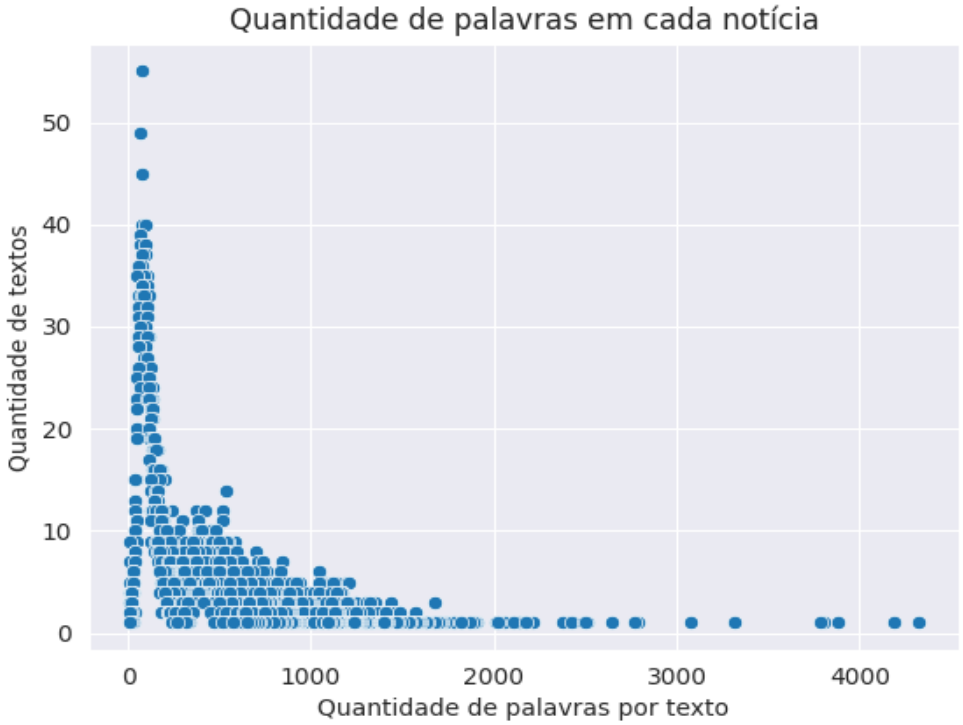
Atributos	Valores
ID	17
fake_news	1
text	ostentou facebook juiz aumentou pensão ex mulher juiz determina aumento pensão comerciante após mulher comprovar … lei juiz faz acordo possibilidade pai pagar necessidade filho receber chamamos binômio diz
author	Josias Oliveira
average_word_length	4.25282
category	sociedade_cotidiano
date_publication	2017-09-14
link	https://afolhabrasil.com.br/politica/ostentou-no-facebook-e-o-juiz-aumentou-pensao-para-ex-mulher/
percent_speeling_errors	0.00564972
number_words	371

Fonte: Acervo do autor

Para o conjunto de dados ser utilizado como base para a detecção de *fake news* é necessário realizar uma atualização dos textos para o mesmo número de palavras, com o objetivo de achar um número de palavras ideal para os todos os textos, para que possam ser utilizados nos modelos de RNAs.

A figura abaixo apresenta o tamanho dos textos, por meio dela é possível visualizar que existem poucos textos com mais de 2000 palavras, dois textos com mais de 4000 palavras e grande parte dos textos possuem poucas palavras, por exemplo existem mais de 50 textos entre, mais ou menos, 50 palavras.

Figura 14 – Número de palavras de cada notícia



Fonte – Acervo do autor

O conjunto de dados sem realizar nenhuma remoção de palavras nos textos possui um total de 7200 textos, entretanto o menor número de palavras encontrado em um texto foi 4 e o maior número de palavras encontrado em um texto foi 4327, como pode ser visualizado na figura acima. Dessa forma, se utilizar todos os textos sem limitar o tamanho, todos os textos devem ter 4 palavras.

Dessa forma, a partir do conjunto de dados base foram criados quatro novos conjuntos de dados que são apresentados abaixo junto com o conjunto de dados original.

Tabela 8 – Conjuntos de dados criados e suas características

Dataset	Quant. Notícias	Quant. Notícias True	Quant. Notícias False	Média Palavras	Média Palavras True	Média Palavras False
original	7200	3600	3600	372,04	634,94	109,15
50 palavras	6853	3599	3254	389,11	635,11	117,02
100 palavras	5166	3576	1590	491,53	638,67	160,59

150 palavras	4170	3520	650	580,03	646,81	218,39
200 palavras	3699	3432	267	631,98	658,82	286,86

Fonte: Acervo do autor

A partir da tabela acima percebe-se que utilizar o conjunto de dados de 150 palavras se torna inviável, devido ao fato dele possuir um baixo número de notícias falsas, apenas 650 notícias. O mesmo ocorre com o conjunto de dados de 200 palavras, no qual possui apenas 267 notícias falsas.

Como relatado anteriormente o conjunto de dados original, sem remover textos com poucas palavras, também se torna inviável pois o menor texto possui apenas 4 palavras o que é muito pouco para a realização de uma boa detecção.

Dessa forma, os conjuntos de dados viáveis para detecção de *fake news* são os de 50 e 100 palavras, devido ao fato de possuírem um bom número de textos e uma divisão entre notícias falsas e verdadeiras aceitável.

Após realizar o tratamento e pré-processamento, falta realizar a conversão para representação numérica, que será explicada no capítulo 4.4 REPRESENTAÇÃO NUMÉRICA DOS DADOS.

4.3 RECURSOS UTILIZADOS

Nesta seção são apresentados os recursos utilizados para construção da aplicação e as funções de ativação.

4.3.1 Bibliotecas

Neste capítulo é apresentado as bibliotecas que são utilizadas para construção dos modelos de detecção de *Fake News*.

4.3.1.1 Numpy

O NumPy é uma biblioteca para o Python que fornece um objeto de matriz multidimensional e vários objetos derivados, como matrizes mascaradas. Além disso, apresenta diversas operações com as matrizes, como manipulação de forma, classificação, seleção, álgebra linear básica, operações estatísticas básicas, geradores de números aleatórios, transformada de *Fourier,* entre outras operações (NumPy, 2020).

De acordo com NumPy (2020) é distribuído a partir da licença BSD, sendo desenvolvido e mantido publicamente no *GitHub* por uma comunidade vibrante, ágil e diversa. O núcleo do NumPy é um código C bem otimizado, no qual aproveita a flexibilidade do Python com a velocidade do código compilado. Desta forma, se torna rápido e versátil, utilizando conceitos de vetorização, indexação e transmissão (NumPy, 2020).

Diferente das listas em Python, os *arrays* NumPy possuem um tamanho fixo na criação, com isso ao mudar o tamanho de um *array,* o NumPy criará um novo *array* e excluirá o original. Os elementos em uma matriz NumPy devem ser todos do mesmo tipo de dado, dessa forma terão o mesmo tamanho na memória. Entretanto é possível a construção de *arrays* de objetos, no qual permitem *arrays* de elementos de tamanhos diferentes (NumPy, 2020).

Dessa forma, a matriz multidimensional disponibilizada pela biblioteca NumPy pode ser utilizada neste trabalho para organizar os dados durante a execução e servir como base para realizar as operações com os modelos de RNAs.

4.3.1.2 Pandas

O pandas é uma biblioteca desenvolvida no ano de 2008. Em 2009, teve seu código aberto e foi ativamente apoiado hoje por uma comunidade de indivíduos com ideias semelhantes em todo o mundo. Possui como objetivo se tornar a ferramenta de análise / manipulação de dados (fictícios e do mundo real) de código aberto mais poderosa e flexível disponível em qualquer idioma (Pandas, 2020).

Segundo Pandas (2020) a biblioteca fornece um objeto *DataFrame* rápido e eficiente para manipulação de dados com indexação integrada, ferramentas para ler e gravar dados entre estruturas de dados na memória e diferentes formatos, por exemplo arquivos CSV e de texto, Microsoft Excel, entre outros.

Com o Pandas também é possível realizar a remodelagem e rotação flexível de conjuntos de dados, fornece uma maneira intuitiva de trabalhar com dados de alta dimensão em uma estrutura de dados de dimensão inferior. Tudo isso é realizado de forma altamente otimizada e com um ótimo desempenho, com partes críticas do código escritas em *Cython* ou C (Pandas, 2020).

O Python com pandas se encontra presente em diversas áreas, como: ensino, finanças, neurociência, economia, estatística, publicidade, análise da web, entre outras (Pandas, 2020). Dessa forma, o objeto *DataFrame* disponibilizado pela biblioteca Pandas

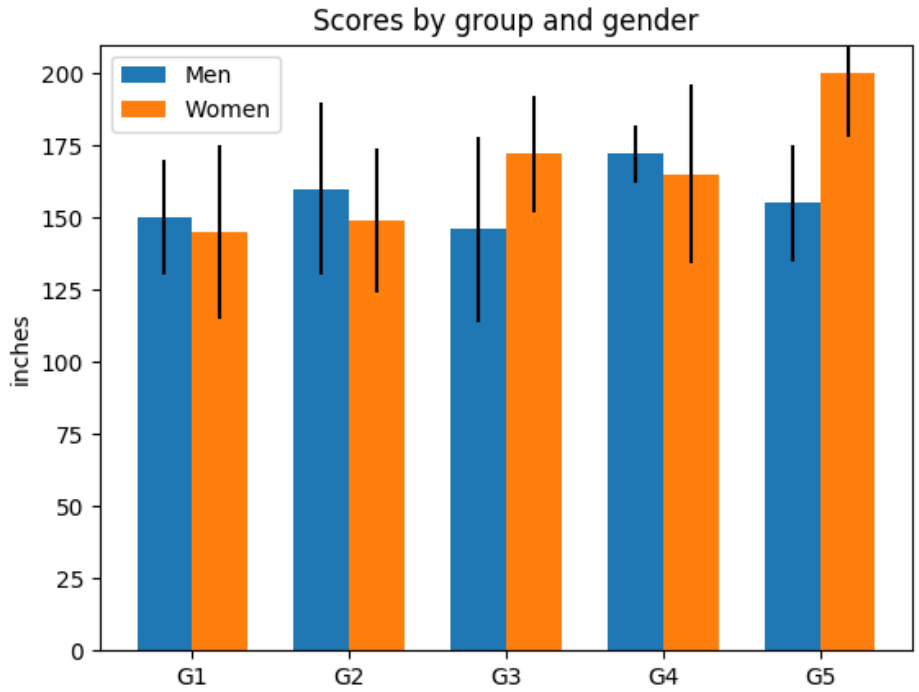
pode ser utilizado neste trabalho e manipulação dos dados, auxiliar no armazenamento e leitura dos arquivos.

4.3.1.3 Matplotlib

O Matplotlib é uma biblioteca utilizada para a criação de gráficos 2D em Python. Teve seu surgimento a partir da emulação dos comandos gráficos do MATLAB, contudo é independente do MATLAB e pode ser utilizado de forma orientada a objetos. O Matplotlib é escrito em Python puro, mas também utiliza outras bibliotecas e extensões, como a biblioteca NumPy (HUNTER, 2007).

Segundo Hunter (2007) o Matplotlib possui como objetivo permitir a criação plotagens simples com apenas um ou alguns comandos, sem a necessidade de grandes quantidades de códigos, instâncias e chamadas de métodos. A figura abaixo apresenta um gráfico de barras criado a partir da biblioteca.

Figura 15 – Exemplo de gráfico de barras do Matplotlib



Fonte – Hunter (2007)

O Matplotlib pode ser utilizado em muitos contextos diferentes, por exemplo para a geração de arquivos PostScript automaticamente para enviar a uma impressora ou editores,

para a geração de PNGs em um servidor de aplicativos da web para inclusão em páginas da web geradas dinamicamente, entre outros casos (HUNTER, 2007).

Dessa forma, os gráficos 2D fornecidos pela biblioteca Matplotlib podem ser utilizados neste trabalho para visualizar o *dataset* e os resultados obtidos a partir dos modelos de uma forma clara e simples.

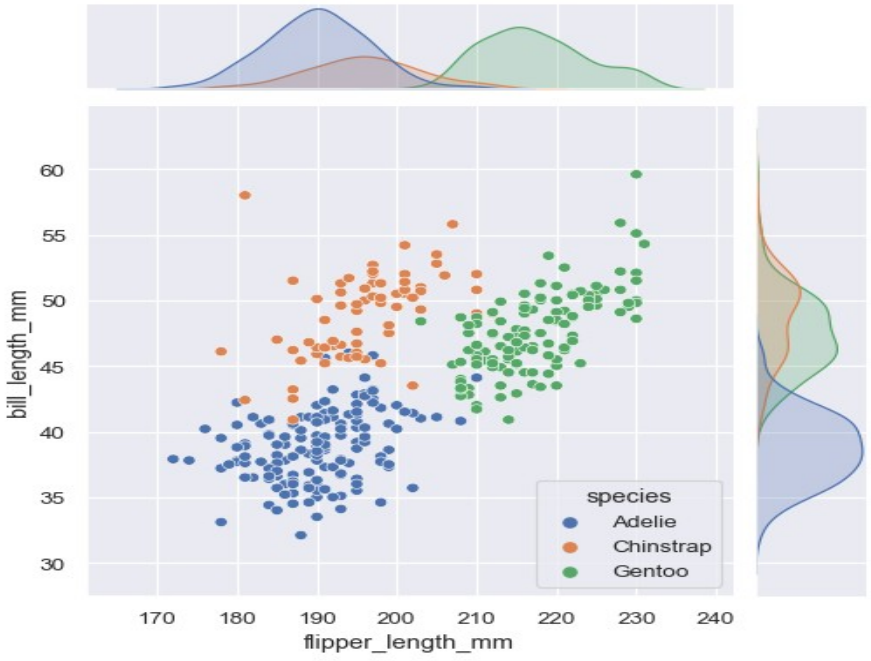
4.3.1.4 Seaborn

O Seaborn é uma biblioteca de visualização de dados em Python baseada no Matplotlib, no qual fornece uma interface de alto nível para desenhar gráficos estatísticos com visuais elegantes, informativos e possui suporte às estruturas de dados do pandas (WASKOM, 2020).

Segundo Waskom (2020) as plotagens do Seaborn operam em *dataframes* e matrizes contendo conjuntos de dados inteiros e executam internamente o mapeamento semântico e agregação estatística necessários para produzir plotagens informativas.

A imagem abaixo apresenta um exemplo de gráfico construído com a biblioteca Seaborn, no qual apresenta a distribuição conjunta entre duas variáveis junto com a distribuição marginal de cada variável (WASKOM, 2020).

Figura 16 – Exemplo de gráfico de distribuição do Seaborn



Fonte – Waskom (2020)

Dessa forma, os gráficos fornecidos pela biblioteca Seaborn podem ser utilizados neste trabalho como um complemento e auxílio junto com a biblioteca Matplotlib para visualizar o *dataset*, entender os dados e os resultados obtidos a partir dos modelos de uma forma clara e simples.

4.3.1.5 NLTK

O *Natural Language Toolkit* (NLTK) é uma biblioteca utilizada para trabalhar com dados de linguagem humana, fornecendo interfaces para corpora e recursos lexicais, além disso também fornece um conjunto de bibliotecas de processamento de texto para classificação, *tokenização*, lematização, marcação, análise e raciocínio semântico e *wrappers* para bibliotecas de Processamento de Linguagem Natural (PNL) (NLTK, 2020).

É uma biblioteca gratuita, de código aberto e voltada para a comunidade, uma ferramenta maravilhosa para ensinar e trabalhar em linguística computacional com o Python, pode ser utilizada por engenheiros, estudantes, educadores, pesquisadores, entre outros (NLTK, 2020).

Dessa forma, as ferramentas fornecidas pela biblioteca NLTK podem ser utilizadas neste trabalho para auxiliar no tratamento dos textos antes de serem convertidos para a representação numérica, removendo por exemplo as *stopwords*.

Segundo Microsoft (2017) as *stopwords* (palavras irrelevantes) são cadeias de caracteres que ocorrem com frequência em um texto e possuem um baixo valor informativo, com isso acabam sendo descartadas. Por exemplo, no inglês, palavras como ‘*a*’, ‘*and*’, ‘*is*’ e ‘*the*’ são retiradas do texto pois são consideradas inúteis.

4.3.1.6 Gensim

De acordo com Řehůřek e Sojka (2010) a biblioteca Gensim teve seu surgimento no ano de 2008, no qual possuía como objetivo gerar uma pequena lista dos artigos mais semelhantes a um determinado artigo. É uma biblioteca de código aberto desenvolvida em Python com rotinas altamente otimizadas, licenciado sob a licença GNU LGPL v2.1 e hospedado, desde 2011, no GitHub.

O Gensim é uma biblioteca robusta, eficiente e descomplicada para realizar modelagem semântica não supervisionada de texto simples. Seu funcionamento e utilização é baseado na clareza, eficiência e escalabilidade, sendo essas características mantidas até hoje (ŘEHŮŘEK; SOJKA, 2010).

Dessa forma, a ferramenta Doc2Vec fornecida pela biblioteca Gensim pode ser utilizada neste trabalho para realizar a conversão dos textos para representação numérica, de forma que as palavras não percam suas semelhanças e diferenças de outras palavras dos textos.

O Word2Vec é um modelo para conversão de palavras isoladas para representação numérica, é um modelo que incorpora palavras em um espaço vetorial de dimensão inferior usando uma rede neural rasa. Dessa forma, o resultado é um conjunto de vetores, no qual vetores próximos no espaço vetorial possuem significados semelhantes e vetores distantes possuem significados diferentes (ŘEHŮŘEK; SOJKA, 2010).

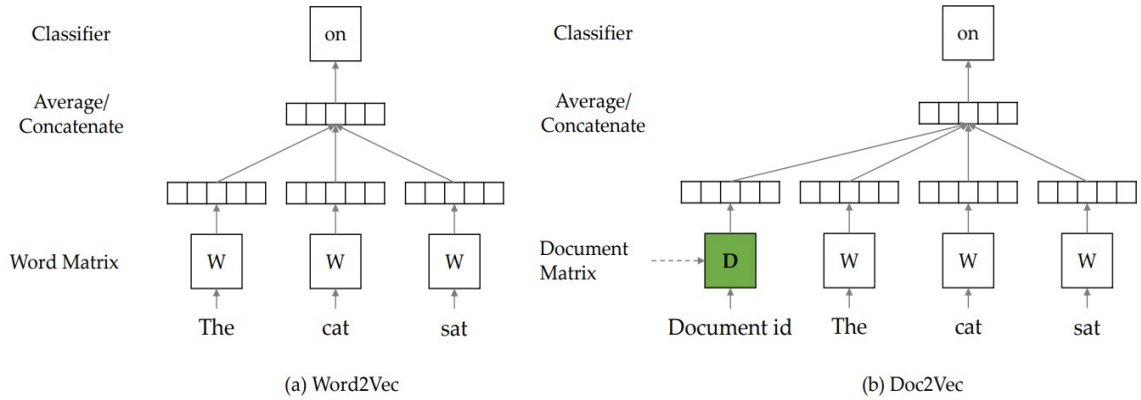
Segundo Řehůřek e Sojka (2010) como exemplo pode-se utilizar as palavras ‘forte’, ‘poderoso’ e ‘Paris’, quem possui significados semelhantes são as palavras ‘forte’ e ‘poderoso’, já ‘Paris’ estaria em um vetor distante, pois possui um significado diferente.

De acordo com Kim, Seo, Cho e Kang (2019) o Word2Vec utiliza redes neurais, no qual uma palavra é considerada como um único vetor e os valores dos elementos de uma palavra são afetados pelos valores de outras palavras ao redor da palavra alvo. O Doc2Vec é uma extensão do Word2Vec, no qual também utiliza redes neurais e cada documento é um vetor que possui seus próprios valores vetoriais no mesmo espaço das palavras.

O Doc2vec é inspirado na técnica de incorporação de palavras e é usado para extrair vetores de palavras que são específicos de um documento, tudo isso mantendo a relação semântica entre vários documentos. O Doc2vec provou ser eficaz principalmente na classificação de sentimentos binários, por exemplo sentimentos positivos e negativos (LEE; YOON, 2017).

A figura abaixo apresenta a estrutura e a diferença entre os modelos Word2Vec e Doc2Vec. No (a) Word2Vec, o contexto de três palavras ‘*The*’, ‘*cat*’ e ‘*sat*’ são usados para prever a quarta palavra ‘*on*’, já no (b) Doc2Vec o contexto do documento é usado para prever a quarta palavra (KIM; SEO; CHO; KANG, 2019).

Figura 17 – Diferença entre os modelos (a) Word2Vec e (b) Doc2Vec



Fonte – Kim, Seo, Cho e Kang (2019)

Segundo Kim, Seo, Cho e Kang (2019) como pode ser visualizado no (b) Doc2Vec na figura acima, cada documento é mapeado para um único vetor que é representado por uma coluna na matriz ‘D’ e cada palavra é mapeada para um único vetor que é representado por uma coluna na matriz ‘W’.

4.3.1.7 Keras

O Keras é uma API de aprendizado profundo desenvolvida em Python e construída com base no TensorFlow 2.0. É projetado seguindo as práticas recomendadas para reduzir a carga cognitiva, minimiza o número de ações do usuário em usos comuns, fornece mensagens de erro claras, uma grande documentação e tudo isso de uma forma simples (KERAS, 2020).

Segundo Keras (2020) é possível utilizar a API em diversos lugares, como: por meio do JavaScript para rodar diretamente no navegador; por meio do TF *Lite* para rodar em iOS, Android e dispositivos embutidos; por meio de uma API da web.

Com o Keras é possível implementar novas ideias de pesquisa e acelerar ciclos de experimentação. Pode ser utilizado com uma grande facilidade e foco na experiência do usuário, com isso acaba sendo utilizado como ferramenta de aprendizagem profunda e trabalho por muitos cursos universitários e organizações científicas, como CERN, NASA e NIH (KERAS, 2020).

De acordo com Keras (2020) o TensorFlow 2.0 é uma plataforma de aprendizado de máquina de código aberto de ponta a ponta. O Keras é a API de alto nível do TensorFlow 2.0, no qual fornece uma interface acessível e altamente produtiva para resolução de

problemas relacionados ao aprendizado de máquina por meio de abstrações e blocos de construção.

Dessa forma, as ferramentas fornecidas pela API Keras podem ser utilizadas neste trabalho para realizar a construção dos modelos de redes neurais que serão utilizados para a detecção de falsas notícias.

4.3.2 Funções de ativação

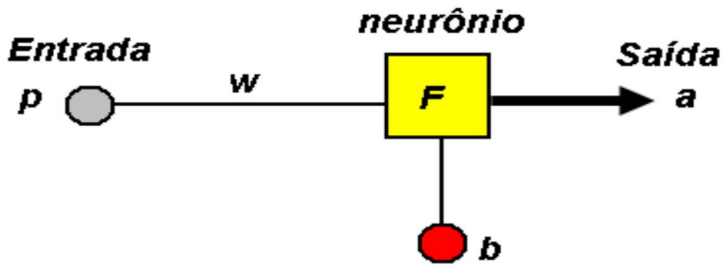
A qualidade e o desempenho de uma rede neural artificial é determinado por meio de diversos fatores, como número de camadas intermediárias, número de nodos intermediários, algoritmo de aprendizagem, função de ativação de cada nodo, entre outros fatores (GOMES, 2010).

Segundo Gomes (2010) como existem diversos fatores influenciadores no desempenho de uma rede, o foco muitas vezes são os algoritmos de aprendizagem e arquiteturas. Dessa forma, as funções de ativação acabam sendo ignoradas e escolhidas em um raso estudo.

As funções de ativação são capazes de influenciar fortemente a complexidade e o desempenho de um modelo. As funções de ativação frequentemente mais utilizadas são as funções ‘logit’ e ‘tangente hiperbólica’ (GOMES, 2010).

A estrutura de um neurônio artificial pode ser apresentada de forma simplificada de acordo com figura abaixo, uma versão completa se encontra no capítulo 2.2.2 Neurônio Artificial (FINOCCHIO, 2014).

Figura 18 – Neurônio artificial simplificado



Fonte – Finocchio (2014)

De acordo com Finocchio (2014) na figura 18 acima: p são os sinais de entradas, w são os pesos (é um parâmetro ajustável), b é o bias (é um parâmetro ajustável), F é uma função no qual as entradas geram as saídas e a é o sinal de saída. Essa figura em formato matemático pode ser visualizado na equação 3 abaixo.

$$a = F(w \times p + b) \tag{3}$$

Um neurônio, como o da figura 18, possui uma característica que é a sua função F, chamada de Função de Ativação, responsável por transformar os sinais de entrada no sinal de saída (FINOCCHIO, 2014). Existem diversas funções de ativação, entretanto as utilizadas neste trabalho são apresentadas nos tópicos a seguir.

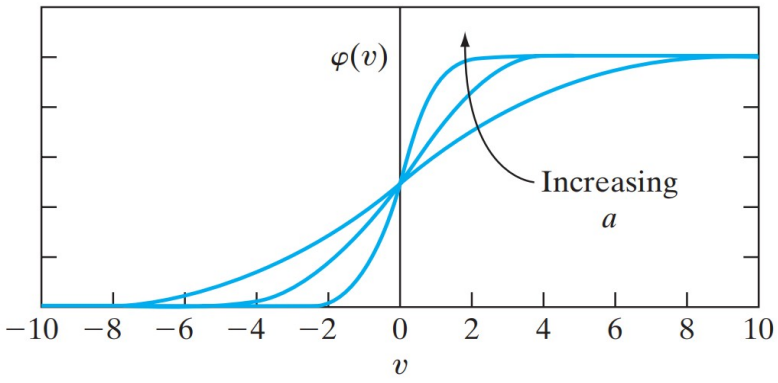
4.3.2.1 Sigmoide

Segundo Haykin (2009) a função sigmoide é a função de ativação mais usada na construção de redes neurais artificiais, sendo definida da seguinte forma:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{4}$$

No qual a é o parâmetro de inclinação da função sigmoide. Dessa forma, variando esse parâmetro obtemos funções sigmoides de diferentes inclinações, como pode ser visualizado na figura abaixo (HAYKIN, 2009).

Figura 19 – Função sigmoide com diferentes inclinações



Fonte – Haykin (2009)

De acordo com a equação 4, v é o campo local induzido, ou seja, a soma ponderada das entradas mais o bias, como pode ser visualizado na equação abaixo (HAYKIN, 2009).

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{5}$$

Segundo Sharma e Sharma (2020) a função sigmoide é uma função não linear e transforma os valores no intervalo de 0 a 1.

4.3.2.2 Tanh

De acordo com Facure (2017) a função de ativação Tanh (Tangente Hiperbólica) é semelhante à função sigmoide, no qual também possui um formato de “S”, entretanto varia de -1 a 1, em vez de 0 a 1 como na sigmoide. A função Tanh é definida da seguinte forma:

$$\tanh(v) = 2\varphi(2v) - 1 \tag{6}$$

Outra diferença entre a função de ativação Tanh e a Sigmoide, é que a Tanh é simétrica em torno da origem, dessa forma resulta em diferentes sinais de saídas de camadas anteriores que serão alimentadas como entrada para a próxima camada (SHARMA; SHARMA, 2020).

Segundo Sharma e Sharma (2020) a função Tanh é preferível em relação à função sigmoide, pois possui gradientes que não estão restritos a variar em uma certa direção.

4.3.2.3 ReLU

A função ReLU (Unidade Linear Retificada) e é uma função de ativação não linear amplamente utilizada em redes neurais. Sua vantagem em relação às outras funções é que todos os neurônios não são ativados ao mesmo tempo, dessa forma um neurônio será desativado apenas quando a saída da transformação linear for zero (SHARMA; SHARMA, 2020).

A função de ativação ReLU é definida da seguinte forma:

$$relu(v) = max(0, v) \tag{7}$$

De acordo com Sharma e Sharma (2020) a ReLU é mais eficiente em relação às outras funções devido ao fato de todos os neurônios não serem ativados ao mesmo tempo, apenas uma parcela dos neurônios são ativados por vez. Entretanto, em alguns casos o valor do gradiente é zero, devido ao fato dos pesos e vieses não serem atualizados durante a etapa de retropropagação da rede neural.

4.3.2.4 ELU

A função ELU (Unidade Linear Exponencial) proposta por Clevert, Unterthiner e Hochreiter é uma função de ativação com o objetivo de acelerar o treinamento das redes neurais profundas. Um dos seus benefícios é a redução do problema do gradiente de fuga utilizando identidade para valores positivos (CLEVERT; UNTERTHINER; HOCHREITER, 2016).

Segundo Clevert, Unterthiner e Hochreiter (2016) comparando com a função ReLU, a ELU possui valores negativos que permitem empurrar ativações da unidade para perto de zero, diminuindo a complexidade computacional. A função de ativação ELU com $0<\alpha$ é definida da seguinte forma:

$$elu(v)=\begin{matrix} v \Rightarrow se \rightarrow v>0 \\ \alpha(\exp(v)-1) \Rightarrow se \rightarrow v \leq 0 \end{matrix} \quad (8)$$

De acordo com a fórmula acima, o hiperparâmetro α da ELU é responsável por controlar o ponto de saturação para entradas negativas (CLEVERT; UNTERTHINER; HOCHREITER, 2016).

4.3.3 Validação Cruzada

Os algoritmos de aprendizado de máquina possuem diversas configurações que são utilizadas para controlar seu comportamento (BENGIO; GOODFELLOW; COURVILLE, 2015). Segundo Silva, Spatti e Flauzino (2016), a busca por um modelo que possui o melhor desempenho geralmente é realizada de forma empírica, devido ao fato de resultado esperado depender de diversos fatores, como: algoritmo de aprendizado utilizado, forma de montagem do algoritmo, forma de início das matrizes de pesos, complexidade de resolução do problema, entre outros fatores.

A qualidade do conjunto de dados utilizado, também um fator que pode prejudicar o desempenho do modelo, devido ao fato da possível existência de ruídos presentes nas amostras, erros estruturais, amostras espúrias (*outliers*), entre outros (SILVA; SPATTI; FLAUZINO, 2016).

De acordo com Silva, Spatti e Flauzino (2016), uma das principais técnicas estatísticas utilizadas para busca da melhor topologia a ser utilizada é a validação cruzada (em inglês: *cross-validation*). Ela utiliza como base diversas topologias, que são as características de modelo, por exemplo duas topologias no qual ambas possuem uma camadas intermediária,

a primeira possui 5 neurônios nessa camada intermediária e a segunda possui 10 neurônios nessa camada intermediária.

Dessa forma, a validação cruzada possui como objetivo avaliar a aptidão de cada topologia quando aplicada a um conjunto de dados que seja diferente daquele usado no ajuste de seus parâmetros internos (SILVA; SPATTI; FLAUZINO, 2016).

O método de validação cruzada que será utilizado nesse trabalho é o método por amostragem aleatória (*random subsampling cross-validation*). Esse método determina que o *dataset* seja aleatoriamente dividido em duas partes, o conjunto de treinamento e o conjunto de teste (validação) (SILVA; SPATTI; FLAUZINO, 2016).

O conjunto de treinamento, é utilizado para treinar todas as topologias candidatas e o conjunto de teste é somente aplicado para selecionar aquela que estará apresentando os melhores resultados de generalização. Dessa forma, o conjunto de teste não participa do processo de aprendizado, o que permite avaliar o desempenho da generalização proporcionada em cada uma das topologias candidatas, comparando os resultados produzidos em suas saídas com os respectivos valores desejados (SILVA; SPATTI; FLAUZINO, 2016).

Os dados de teste ainda podem ser divididos em dois subconjuntos, de teste e de validação. No qual, o subconjunto de validação é construído a partir dos dados de teste (BENGIO; GOODFELLOW; COURVILLE, 2015).

De acordo com Ripley (1996), o *dataset* pode ser divididos em:

- Conjunto de treinamento: Conjunto de amostras utilizados para aprendizagem, ou seja, para se adequar aos parâmetros do classificador;
- Conjunto de validação: Conjunto de amostras usados para ajustar os parâmetros de um classificador, por exemplo, para escolher o número de unidades ocultas em uma rede neural;
- Conjunto de testes: Conjunto de amostras usadas apenas para avaliar o desempenho de um classificador totalmente especificado.

O *dataset* para detecção de *fake news* é dividido em 70% para o conjunto de treinamento e 30% para o conjunto de testes (validação).

O conjunto de teste (validação) também é dividido em dois conjuntos, o conjunto de teste e o conjunto de validação, no qual o conjunto de teste possui 70% das amostras e o conjunto de validação possui 30% das amostras. Deste modo, o *dataset* resultou em três conjuntos, que são treino, teste e validação.

4.3.4 Algoritmos de treinamento

Segundo Vieira, Razente e Barioni (2017), um modelo com boa qualidade deve possuir boas predições, no qual é necessário que os parâmetros da rede (pesos W e bias b) estejam adequados para o sistema. Deste modo, é necessário saber quão bons esses parâmetros são.

Para avaliar a qualidade da previsão/detecção é necessário a utilização de uma função, chamada de função de custo (em inglês: *loss function* ou *cost function*). Essa função determina o quão longe se encontra a previsão/detecção atual da ideal, dessa forma quantifica o “custo” ou “perda” ao aceitarmos a predição gerada pelos parâmetros atuais do modelo (VIEIRA; RAZENTE; BARIONI, 2017).

Por exemplo, o modelo resultou da previsão/detecção os valores $\hat{Y}$, esses valores se comparados a classe verdadeira Y , resultam no custo e na perda que auxiliam na determinação da qualidade do modelo. Com isso, uma das funções de custo mais utilizadas em classificação é a Entropia Cruzada (em inglês: *cross-entropy*) (VIEIRA; RAZENTE; BARIONI, 2017).

De acordo com Vieira, Razente e Barioni (2017), com a função de custo definida, para realizar a redução de custo é necessário ajustar os parâmetros do modelo utilizando o algoritmo do Gradiente Descendente (GD) juntamente com o método de *backpropagation*, que permite obter o gradiente para a sequência de parâmetros presentes na rede usando a regra da cadeia.

Alguns modelos de redes neurais, como o modelo CNN possuem muitos parâmetros que precisam ser aprendidos fazendo com que seja necessário treiná-la usando milhares, ou até milhões, de dados do *dataset*. Com isso, a otimização com um modelo com muitos parâmetros torna a utilização de Gradiente Descendente inviável, devido ao fato do algoritmo calcular o gradiente para todas as instâncias individualmente (VIEIRA; RAZENTE; BARIONI, 2017).

Essa dificuldade pode ser eliminada utilizando algumas alternativas, a alternativa utilizada neste trabalho é o algoritmo Adam apresentado no tópico a seguir.

4.3.4.1 Adam

De acordo com os criadores Kingma e Ba (2015), o Adam (*Adaptive Moment Estimation*) é um método para otimização estocástica que requer apenas gradientes de

primeira ordem com pouca necessidade de memória. É um método voltado para problemas de aprendizado de máquina com grandes conjuntos de dados e/ou parâmetros de alta dimensão.

O Adam é baseado e construído com base em dois algoritmos, o Adagrad e o RMSProp (KINGMA; BA, 2015). O Adagrad é um algoritmo para otimização baseada em gradiente que adapta a taxa de aprendizagem aos parâmetros, utiliza baixa taxas de aprendizagem para parâmetros associados a recursos de ocorrência frequente e alta taxa de aprendizagem para parâmetros associados a recursos não frequentes (RUDER, 2017).

Segundo Ruder (2017), o Adadelta é uma extensão do Adagrad que visa reduzir sua taxa de aprendizagem agressiva e monotonicamente decrescente, dessa forma no lugar de acumular todos os gradientes quadrados anteriores, restringe o número de gradientes anteriores acumulados a algum tamanho fixo.

O RMSprop é um método de taxa de aprendizagem adaptativa proposto por Geoff Hinton, mas não publicado. O RMSprop foi desenvolvido na mesma época do Adadelta, também com o objetivo de resolver as taxas de aprendizagem radicalmente decrescentes do Adagrad (RUDER, 2017).

O Adam é um método que calcula as taxas de aprendizado adaptativo para cada parâmetro, além de armazenar uma média exponencialmente decadente de gradientes quadrados anteriores como Adadelta e RMSprop, Adam também mantém uma média exponencialmente decadente de gradientes anteriores (RUDER, 2017).

De acordo com Ruder (2017), o Adam adiciona correção de viés e momentum ao RMSprop. O RMSprop, Adadelta e Adam são algoritmos muito semelhantes que funcionam bem em circunstâncias semelhantes, no entanto a correção de viés ajuda o Adam a superar ligeiramente os resultados no final da otimização, conforme os gradientes se tornam mais esparsos.

4.4 REPRESENTAÇÃO NUMÉRICA DOS DADOS

Neste capítulo é apresentado a forma de conversão dos textos para representação numérica por meio da ferramenta Doc2Vec. Entretanto, a conversão é uma etapa complexa pois a ferramenta Doc2Vec utiliza Redes Neurais Artificiais e dependendo da configuração utilizada os resultados da detecção de *fake news* podem ser afetados.

A figura abaixo apresenta a disposição dos dados para treinamento (parâmetros de entrada) dos modelos de RNA e o parâmetro de saída.

Figura 20 – Parâmetros de entrada (em formato de texto) e saída

PARÂMETROS DE ENTRADA					PARÂMETRO DE SAÍDA	
palavra_01	palavra_02	palavra_03	...	palavra_300	0 / 1	(linha de exemplo)
ostentou	facebook	juiz	...	nove	1	
próximo	presidente	centro	...	durante	1	
ciro	gomes	ataca	...	destruindo	1	
após	derrubada	avião	...	síria	1	
alvo	lava	jato	...	bloqueio	0	
chefs	convidados	encontro	...	líquido	0	
lições	educação	moral	...	federais	0	
moraes	filiado	psdb	...	gestão	0	

Obs.: Antes de serem realmente utilizados é realizado a conversão das palavras para representação numérica

Fonte – Acervo do autor

Como visualizado na figura acima, existem 300 parâmetros de entrada, entretanto esse valor não será quantidade de parâmetros de entrada realmente utilizada nos modelos para detecção de *fake news*, visto que utilizando textos com mais de 200 palavras existem um número de falsas notícias muito pequeno para realizar a detecção, como explicado no capítulo 4.2 TRATAMENTO DOS DADOS.

A conversão dos textos para representação numérica é realizada por meio da ferramenta Doc2Vec, no qual também utiliza Redes Neurais Artificiais e é fornecida pela biblioteca Gensim, explicada no capítulo 4.3.1.6 Gensim (ŘEHŮŘEK; SOJKA, 2010).

Os parâmetros modificados utilizados na ferramenta Doc2Vec foram os seguintes:

- *min_count*=1
- *window*=5
- *vector_size*=Tamanhos dos textos do dataset
- *sample*=1e-4
- *negative*=5
- *workers*=7
- *epochs*=10
- *seed*=1

Após realizar a conversão dos textos para representação numérica, cada palavra corresponde a um parâmetro de entrada para cada texto em cada modelo. Entretanto, a Figura 20 – Parâmetros de entrada (em formato de texto) e saída é uma forma visual facilitada das entradas nos modelos, visto que as reais entradas nos modelos serão cada palavra do texto convertido para representação numérica, pois os modelos de RNA utilizados para detecção de *fake news* não aceitam palavras como parâmetros de entrada.

Uma exemplificação da conversão para representação pode ser visualizada na figura abaixo, no qual apresenta as palavras da Figura 20 – Parâmetros de entrada (em formato de texto) e saída já em formato numérico realizado pela ferramenta Doc2Vec.

Figura 21 – Parâmetros de entrada (em formato numérico) e saída

PARÂMETROS DE ENTRADA					PARÂMETRO DE SAÍDA	
palavra_01	palavra_02	palavra_03	...	palavra_300	0 / 1	(linha de exemplo)
0.1955745	-0.36848238	0.06341956	...	-0.2190714	1	
0.07045722	0.24210532	0.2753526	...	-0.18704985	1	
-0.08204109	0.18444747	0.049466193	...	0.03616112	1	
0.4345493	0.02589546	-0.23577702	...	-0.46048978	1	
-0.00641382	-0.4292729	0.28340545	...	-0.44986063	0	
-0.53358805	-0.10165366	0.009556446	...	-0.14985493	0	
-0.19424677	-0.14014886	-0.1365485	...	-0.15136641	0	
-0.13661788	0.001767755	0.25988016	...	-0.22669454	0	

Fonte – Acervo do autor

Além disso, o parâmetro de saída corresponde ao valor 0 ou 1, no qual o valor 1 significa que o modelo detectou a notícia como uma *fake news* e 0 significa que o modelo detectou a notícia como uma *real news*.

4.5 ÍNDICES PARA ANÁLISE DA QUALIDADE DA DETECÇÃO

Após a realização do tratamento do *dataset* e a conversão dos textos para representação numérica é necessário iniciar os testes com os modelos MLP e LSTM. Entretanto, os índices de análise são extremamente importantes para a etapa de teste, pois irão apresentar a qualidade da detecção de acordo com a perspectiva do índice.

Dessa forma, os índices utilizados neste trabalho são apresentados nos tópicos abaixo.

- ***Root Mean Square Error* - RMSE**

Segundo Amazon (2021) a Raiz quadrada do erro médio (RMSE) é uma medida de distância entre o destino numérico previsto e a resposta numérica real (verdade). Dessa forma, quanto menor o valor do RMSE, melhor será a precisão preditiva do modelo. Um modelo com previsões perfeitamente corretas teria um RMSE igual a 0.

De acordo com Otto (2019), o RMSE pode ser calculado da seguinte forma:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}} \tag{9}$$

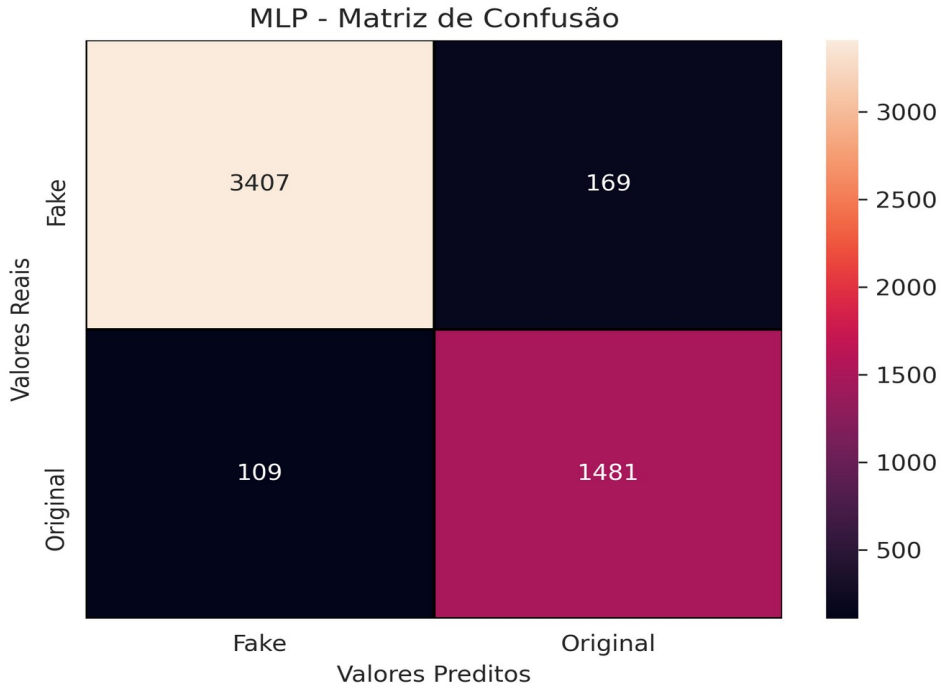
De acordo com a equação acima, y_i é a i-éssima observação de y e $\hat{y}$ é o valor previsto pelo modelo (OTTO, 2019).

- **Matriz de Confusão**

A matriz de confusão ilustra em uma tabela o número ou a porcentagem de previsões corretas e incorretas para cada classe comparando a classe prevista de uma observação com sua classe verdadeira (AMAZON, 2021), sendo muito útil para avaliação do modelo (SCHADE, 2019).

A imagem abaixo apresenta um exemplo da sua utilização na detecção de *Fake News*.

Figura 22 – Matriz de confusão



Fonte – Acervo do autor

De acordo com Schade (2019) como visualizado na figura acima, a matriz de confusão é composta por quatro valores apresentados abaixo.

– *True positive (TP) – Linha 01 Coluna 01*: Este valor representa a quantidade de falsas notícias que foram classificadas como falsas notícias.

– *False negative (FN) – Linha 01 Coluna 02*: Este valor representa a quantidade de verdadeiras notícias que foram classificadas como falsas notícias.

– *False positive (FP) – Linha 02 Coluna 01*: Este valor representa a quantidade de falsas notícias que foram classificadas como verdadeiras notícias.

– *True negative (TN) – Linha 02 Coluna 02*: Este valor representa a quantidade de verdadeiras notícias que foram classificadas como verdadeiras notícias.

- **F1 Score**

Segundo Rodrigues (2019) a métrica F1 Score é baseada na combinação entre outras duas métricas, que são o *recall* e a precisão, de modo que tragam um único número.

A precisão pode ser determinada da seguinte maneira: Das classificações que o modelo determinou como corretas, quais eram realmente corretas. A fórmula é determinada da seguinte maneira (RODRIGUES, 2019):

$$Precisão: \frac{TP}{TP+FP} \tag{10}$$

O *recall* pode ser determinado da seguinte maneira: Frequência no qual o modelo encontra exemplos de uma classe. A fórmula é determinada da seguinte maneira (RODRIGUES, 2019):

$$Recall: \frac{TP}{TP+NF} \tag{11}$$

Dessa forma, a métrica F1 Score é definida da seguinte forma (RODRIGUES, 2019):

$$F1\,Score: \frac{2*Precisão*Recall}{Precisão+Recall} \tag{12}$$

4.6 DEFINIÇÃO DA MELHOR CONFIGURAÇÃO DE RNA

Para a escolha da melhor configuração de RNA foram realizadas etapas separadas para cada modelo e avaliado qual é a melhor configuração. Dessa forma, para avaliar a qualidade de um modelo é utilizado a métrica RMSE, no qual quanto menor o valor melhor e a métrica F1-Score, no qual é uma porcentagem e quanto maior o valor melhor.

Para o modelo MLP, primeiramente foram realizados testes alterando as funções de ativação em cada uma das camadas, com o objetivo de encontrar as três melhores configurações. Em seguida, foi realizado testes com as três melhores configurações anteriores utilizando a quantidade de neurônios variada, dessa forma também foi buscado as três melhores configurações. Por último, foram realizados testes com as três melhores

configurações anteriores utilizando o *batch size* variado com o objetivo de encontrar a melhor configuração de acordo com todos os testes realizados.

Para o modelo LSTM, foi realizado todas as etapas realizadas com o modelo MLP e adicionado mais duas etapas. Dessa forma, a quarta etapa foi realizada aumentando o número de camadas intermediárias, entretanto essa etapa piorou os resultados, então os modelos não foram utilizados na próxima etapa. A quinta e última etapa utilizou os três melhores modelos anteriores (modelos da terceira etapa) e aumentou o número de épocas com o objetivo de aprimorar os resultados. Com isso, a melhor configuração foi escolhida.

Uma configuração de uma RNA se for executada mais de uma vez gerará resultados diferentes, em alguns casos melhores e em outros piores, dessa forma após encontrar a melhor configuração para cada modelo (nesse caso pode ser MLP ou LSTM) ela será executada diversas vezes até encontrar um resultado melhor (processo muito demorado para o modelo LSTM, devido ao fato desse modelo possuir uma execução lenta).

Após encontrar a melhor configuração para cada modelo, é comparado as métricas das duas configurações avaliando qual é realmente a melhor configuração encontrada.

5 RESULTADOS

Neste capítulo inicialmente é apresentado o treinamento do modelo MLP com diferentes configurações e os resultados obtidos para detecção de *fake news*. Em seguida, é apresentado o treinamento do modelo LSTM com diferentes configurações e os resultados obtidos para detecção de *fake news*. Dessa forma, com os resultados dos dois modelos obtidos são analisadas e comparadas as configurações que obtiveram os melhores desempenhos.

5.1 MODELO MLP

Esse capítulo apresenta o treinamento e resultado das diferentes configurações realizadas com o modelo MLP (*Multilayer Perceptron*).

Todos os testes apresentados abaixo com o modelo MLP foram realizados com algumas configurações fixas, que podem ser visualizadas na tabela abaixo.

Tabela 9 – Testes modelo MLP – Informações fixas

Informação	Valor
Número de Camadas	3 (1 camada de entrada, 1 camada intermediária, 1 camada de saída)
Quantidade de Épocas	50
Kernel Initializer	*Uniform*
Algoritmo de otimização	ADAM

Fonte: Acervo do autor

Os testes utilizando o modelo MLP foram separados em duas etapas, no qual cada uma das etapas possui três testes. Além disso, a primeira etapa é realizada utilizando o *dataset* de 50 palavras e a segunda etapa é realizada utilizando o *dataset* de 100 palavras.

5.1.1 Modelo MLP com o *dataset* de 50 palavras

As etapas abaixo para testes com o modelo MLP foram realizadas com base no *dataset* com 50 palavras. O primeiro teste realizado, foi utilizando funções de ativações variadas em todas as camadas, podendo variar entre *sigmoid*, *tanh*, *elu* e *relu*. Dessa forma, o *batch size* será igual a 10, como as configurações serão montadas com três camadas, a

quantidade de neurônios da primeira camada será 12, da segunda será 8 e da terceira será 1, além disso como mencionado anteriormente a quantidade de épocas será 50.

A tabela abaixo apresenta os resultados das 64 configurações obtidas. No qual um RMSE menor ou igual a 0.1 é destacado em verde e um F1 Score maior ou igual à 0.95 também é destacado em verde.

Tabela 10 – Testes modelo MLP – Funções de ativação variadas

Ativação L1	Ativação L2	Ativação L3	Loss	Acurácia Modelo (%)	RMSE	F1 Score
sigmoid	sigmoid	sigmoid	0.13	95.46	**0.08**	**0.95**
sigmoid	sigmoid	tanh	7.93	48.62	0.6	0.34
sigmoid	sigmoid	relu	0.16	95.3	**0.1**	0.94
sigmoid	sigmoid	elu	7.93	48.62	0.54	0.34
sigmoid	tanh	sigmoid	0.15	94.17	**0.09**	0.94
sigmoid	tanh	tanh	7.93	48.62	0.52	0.34
sigmoid	tanh	relu	7.93	48.62	0.51	0.34
sigmoid	tanh	elu	7.93	48.62	0.52	0.34
sigmoid	relu	sigmoid	0.14	94.81	**0.08**	0.94
sigmoid	relu	tanh	7.93	48.62	0.52	0.34
sigmoid	relu	relu	0.17	94.33	0.21	0.6
sigmoid	relu	elu	0.3	87.2	0.2	0.59
sigmoid	elu	sigmoid	0.14	94.81	**0.08**	0.94
sigmoid	elu	tanh	0.17	95.14	0.13	0.94
sigmoid	elu	relu	0.2	94.17	0.18	0.61
sigmoid	elu	elu	0.26	91.09	0.21	0.92
tanh	sigmoid	sigmoid	0.13	95.14	**0.08**	**0.95**
tanh	sigmoid	tanh	7.93	48.62	0.59	0.34
tanh	sigmoid	relu	7.93	48.62	0.51	0.34
tanh	sigmoid	elu	0.16	94.65	0.15	0.94
tanh	tanh	sigmoid	0.12	95.46	**0.07**	**0.95**
tanh	tanh	tanh	0.17	94.49	0.25	0.57
tanh	tanh	relu	0.2	95.3	0.12	0.94
tanh	tanh	elu	0.18	94.98	0.23	0.62
tanh	relu	sigmoid	0.14	95.14	**0.07**	**0.95**
tanh	relu	tanh	0.27	95.79	0.22	0.63
tanh	relu	relu	0.21	94.81	0.16	0.63
tanh	relu	elu	0.19	95.3	0.23	0.63
tanh	elu	sigmoid	0.12	95.46	**0.07**	**0.95**
tanh	elu	tanh	0.32	95.3	0.38	0.46
tanh	elu	relu	0.27	95.95	0.12	**0.95**
tanh	elu	elu	0.34	90.11	0.34	0.56
relu	sigmoid	sigmoid	0.14	95.62	**0.08**	**0.95**

relu	sigmoid	tanh	0.17	94.81	**0.1**	**0.95**
relu	sigmoid	relu	7.93	48.62	0.51	0.34
relu	sigmoid	elu	7.93	48.62	0.55	0.34
relu	tanh	sigmoid	0.13	95.62	**0.06**	**0.96**
relu	tanh	tanh	0.26	94.81	0.18	0.61
relu	tanh	relu	0.17	95.95	0.13	**0.96**
relu	tanh	elu	0.23	94.17	0.27	0.43
relu	relu	sigmoid	0.7	48.62	0.5	0.34
relu	relu	tanh	0.22	95.62	0.19	0.61
relu	relu	relu	0.14	95.14	0.23	0.58
relu	relu	elu	0.18	94.81	0.23	0.46
relu	elu	sigmoid	0.12	95.62	**0.06**	**0.96**
relu	elu	tanh	0.21	95.62	0.21	0.59
relu	elu	relu	0.2	95.14	0.17	0.62
relu	elu	elu	0.16	96.27	0.3	0.56
elu	sigmoid	sigmoid	0.14	94.98	**0.08**	**0.95**
elu	sigmoid	tanh	0.14	95.3	**0.08**	**0.96**
elu	sigmoid	relu	7.93	48.62	0.51	0.34
elu	sigmoid	elu	0.2	95.62	0.15	**0.95**
elu	tanh	sigmoid	0.14	95.46	**0.07**	**0.95**
elu	tanh	tanh	0.22	94.49	0.38	0.45
elu	tanh	relu	0.2	95.62	0.16	0.63
elu	tanh	elu	0.18	95.79	0.32	0.55
elu	relu	sigmoid	0.13	94.81	**0.08**	0.94
elu	relu	tanh	0.19	95.3	0.13	0.63
elu	relu	relu	0.59	90.28	0.15	0.91
elu	relu	elu	7.93	48.62	0.52	0.34
elu	elu	sigmoid	0.12	95.79	**0.07**	**0.95**
elu	elu	tanh	0.27	95.3	0.41	0.43
elu	elu	relu	7.93	48.62	0.51	0.34
elu	elu	elu	0.2	94.81	0.29	0.57

Fonte: Acervo do autor

Como observado na tabela acima, mesmo sendo o primeiro teste já existem ótimos resultados, por exemplo um resultado com RMSE de 0.06 e F1 Score de 96%. Os modelos apresentados abaixo possuem as melhores porcentagens para o F1 Score, e comparando as outras métricas com os outros modelos, esses três se destacam nos resultados.

- **Modelo 01**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 12;
  - Camada 02: Ativação tanh; Neurônios: 8;

- Camada 03: Ativação sigmoid; Neurônios: 1;
- *Loss*: 0.1339; Acurácia Modelo (%): 95.6240;
- RMSE: 0.0581; F1 Score: 0.9633;

- **Modelo 02**
  - *Batch size*: 10;
  - Camada 01: Ativação elu; Neurônios: 12;
  - Camada 02: Ativação elu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1234; Acurácia Modelo (%): 95.7861;
  - RMSE: 0.0693; F1 Score: 0.9531;
- **Modelo 03**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 12;
  - Camada 02: Ativação elu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1241; Acurácia Modelo (%): 95.6240;
  - RMSE: 0.0605; F1 Score: 0.9620;

O segundo teste é realizado pegando os três melhores modelos do primeiro teste e utilizado a quantidade de neurônios variada, no qual a primeira e a segunda camada variam a quantidade de neurônios entre 1, 2, 4, 8, 10, 25, 30, 50, 75, 90, 100, 128, 150, 200 e 300. A terceira camada sempre possuirá 1 neurônio.

Dessa forma, para cada um dos três modelos, foram obtidas 225 configurações de modelos, totalizando 675 configurações.

Para o modelo 01, uma das melhores configurações obtidas pode ser visualizada abaixo.

- **Modelo 04**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.4219; Acurácia Modelo (%): 94.9757;
  - RMSE: 0.0550; F1 Score: 0.9813;

Para o modelo 02 a melhor configuração obtida, no caso com o melhor RMSE e um dos melhores F1 Score, pode ser visualizada abaixo.

- **Modelo 05**
  - *Batch size*: 10;
  - Camada 01: Ativação elu; Neurônios: 200;
  - Camada 02: Ativação elu; Neurônios: 300;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1774; Acurácia Modelo (%): 94.9757;
  - RMSE: 0.0608; F1 Score: 0.9626;

Para o modelo 03, uma das melhores configurações obtidas (neste caso com o melhor RMSE) pode ser visualizada abaixo (nesse teste os resultados de *Loss* não foram muito bons).

- **Modelo 06**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 75;
  - Camada 02: Ativação elu; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.2686; Acurácia Modelo (%): 95.6240;
  - RMSE: 0.0476; F1 Score: 0.9800;

O terceiro e último teste é realizado pegando a melhor configuração para cada um dos três testes da etapa anterior e utilizado um *batch size* que varia entre os valores 1, 2, 5, 10, 25 e 50. Dessa forma, para cada um dos três modelos, foram obtidas 6 configurações de modelos, totalizando 18 configurações.

Para o modelo 04 a melhor configuração obtida (no qual possui o menor RMSE e o maior F1 Score) pode ser visualizada abaixo.

- **Modelo 07**
  - *Batch size*: 2;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.2595; Acurácia Modelo (%): 95.9481;
  - RMSE: 0.0429; F1 Score: 0.9821;

Para o modelo 05 a melhor configuração obtida (que possui o menor RMSE e o maior F1 Score) pode ser visualizada abaixo.

- **Modelo 08**
  - *Batch size*: 2;

- Camada 01: Ativação elu; Neurônios: 200;
- Camada 02: Ativação elu; Neurônios: 300;
- Camada 03: Ativação sigmoid; Neurônios: 1;
- *Loss*: 0.2410; Acurácia Modelo (%): 94.0032;
- RMSE: 0.0631; F1 Score: 0.9693;

Para o modelo 06 a melhor configuração obtida (neste caso com o melhor RMSE) pode ser visualizada abaixo.

- **Modelo 09**
  - *Batch size*: 2;
  - Camada 01: Ativação relu; Neurônios: 75;
  - Camada 02: Ativação elu; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.3366; Acurácia Modelo (%): 94.9757;
  - RMSE: 0.0528; F1 Score: 0.9779;

5.1.2 Modelo MLP com o *dataset* de 100 palavras

As etapas abaixo para testes com o modelo MLP foram realizadas com base no *dataset* com 100 palavras, além disso todos os testes abaixo são realizados da mesma forma que os testes com o *dataset* com 50 palavras e utilizando as mesmas configurações fixas que estão apresentados na tabela 9 – Testes modelo MLP – Informações fixas.

O primeiro teste realizado, foi utilizando funções de ativações variadas em todas as camadas, podendo variar entre *sigmoid*, *tanh*, *elu* e *relu*. Dessa forma, o *batch size* será igual a 10, como as configurações serão montadas com três camadas, a quantidade de neurônios da primeira camada será 12, da segunda será 8 e da terceira será 1, além disso como mencionado anteriormente a quantidade de épocas será 50.

Os modelos apresentados abaixo são os três melhores resultados obtidos, no qual possuem os três maiores F1 Score e os três menores RMSE.

- **Modelo 01**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 12;
  - Camada 02: Ativação tanh; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1483; Acurácia Modelo (%): 95.0538;

- RMSE: 0.0696; F1 Score: 0.9471;

- **Modelo 02**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 12;
  - Camada 02: Ativação elu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1444; Acurácia Modelo (%): 94.6237;
  - RMSE: 0.0701; F1 Score: 0.9473;
- **Modelo 03**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 12;
  - Camada 02: Ativação relu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1409; Acurácia Modelo (%): 94.4086;
  - RMSE: 0.0775; F1 Score: 0.9406;

O segundo teste é realizado pegando os três melhores modelos do primeiro teste e utilizado a quantidade de neurônios variada, como realizado no segundo teste com o *dataset* de 50 palavras. Para o modelo 01, uma das melhores configurações obtidas pode ser visualizada abaixo (No qual essa configuração possui o melhor RMSE e um dos melhores F1 Score).

- **Modelo 04**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 30;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.3711; Acurácia Modelo (%): 94.1935;
  - RMSE: 0.0646; F1 Score: 0.9764;

Para o modelo 02 a melhor configuração obtida, no caso com o segundo melhor RMSE e um dos melhores F1 Score, pode ser visualizada abaixo.

- **Modelo 05**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 150;
  - Camada 02: Ativação elu; Neurônios: 300;
  - Camada 03: Ativação sigmoid; Neurônios: 1;

- *Loss*: 0.5468; Acurácia Modelo (%): 93.9785;
- RMSE: 0.0635; F1 Score: 0.9730;

Para o modelo 03 a melhor configuração obtida, com o melhor RMSE e o melhor F1 Score pode ser visualizada abaixo.

- **Modelo 06**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 200;
  - Camada 02: Ativação relu; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.3015; Acurácia Modelo (%): 96.3441;
  - RMSE: 0.0436; F1 Score: 0.9805;

O terceiro e último teste é realizado pegando a melhor configuração para cada um dos três testes da etapa anterior e utilizado um valor variado para o *batch size*, como realizado no terceiro teste com o *dataset* de 50 palavras. Dessa forma, para o modelo 04 a melhor configuração obtida (no qual possui o menor RMSE e o melhor F1 Score) pode ser visualizada abaixo.

- **Modelo 07**
  - *Batch size*: 25;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 30;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.3001; Acurácia Modelo (%): 93.3333;
  - RMSE: 0.0715; F1 Score: 0.9728;

Para o modelo 05 a melhor configuração obtida, com o melhor RMSE pode ser visualizada abaixo.

- **Modelo 08**
  - *Batch size*: 5;
  - Camada 01: Ativação relu; Neurônios: 150;
  - Camada 02: Ativação elu; Neurônios: 300;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.4219; Acurácia Modelo (%): 93.9785;
  - RMSE: 0.0610; F1 Score: 0.9732;

Para o modelo 06 a melhor configuração obtida (no qual possui o menor RMSE) pode ser visualizada abaixo.

- **Modelo 09**
  - *Batch size*: 5;
  - Camada 01: Ativação relu; Neurônios: 200;
  - Camada 02: Ativação elu; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.5036; Acurácia Modelo (%): 92.4731;
  - RMSE: 0.0748; F1 Score: 0.9742;

5.1.3 Melhor modelo MLP

Avaliando as métricas de todos os resultados acima com o modelo MLP e o *dataset* com 50 palavras, o modelo 07 apresentado abaixo se destaca um pouco em relação aos outros modelos (visto que todos os resultados já foram muito bons).

- **Modelo 07 – *Dataset* 50 palavras**
  - *Batch size*: 2;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.2595; Acurácia Modelo (%): 95.9481;
  - RMSE: 0.0429; F1 Score: 0.9821;

Em relação ao modelo MLP e o *dataset* com 100 palavras, o modelo 06 apresentado abaixo se destaca em relação aos outros modelos.

- **Modelo 06 – *Dataset* 100 palavras**
  - *Batch size*: 10;
  - Camada 01: Ativação relu; Neurônios: 200;
  - Camada 02: Ativação relu; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.3015; Acurácia Modelo (%): 96.3441;
  - RMSE: 0.0436; F1 Score: 0.9805;

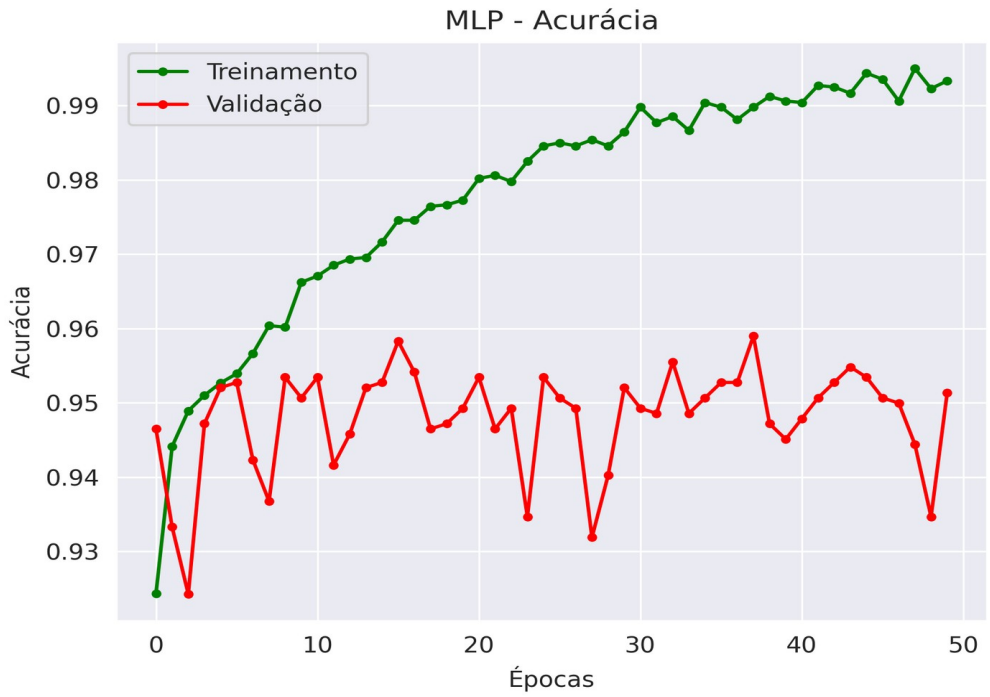
Observando os dois modelos acima, o modelo 07 do *dataset* 50 palavras possui as métricas melhores quando comparado ao modelo 06 do *dataset* de 100 palavras. Dessa forma, abaixo é apresentado alguns gráficos sobre o desempenho do modelo 07, entretanto rodando novamente a detecção com o modelo 07 os resultados serão diferentes, devido ao fato de uma rede neural artificial não gerar sempre os mesmo resultados, após executar diversas vezes o

modelo 07 o resultado abaixo foi obtido (Com resultados melhores que a primeira execução do modelo 07).

- **Melhor modelo utilizando a RNA MLP**
  - Número de camadas: 3 (1 camada de entrada, 1 camada intermediária, 1 camada de saída);
  - Quantidade de Épocas: 50;
  - *Kernel Initializer*: *Uniform;*
  - Algoritmo de otimização: ADAM;
  - *Batch size*: 2;
  - Camada 01: Ativação relu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 128;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1973; Acurácia Modelo (%): 96.2723;
  - RMSE: 0.0439; F1 Score: 0.9832;

A figura abaixo apresenta o histórico da acurácia do treinamento e da validação em todas as 50 épocas utilizadas para o melhor modelo MLP.

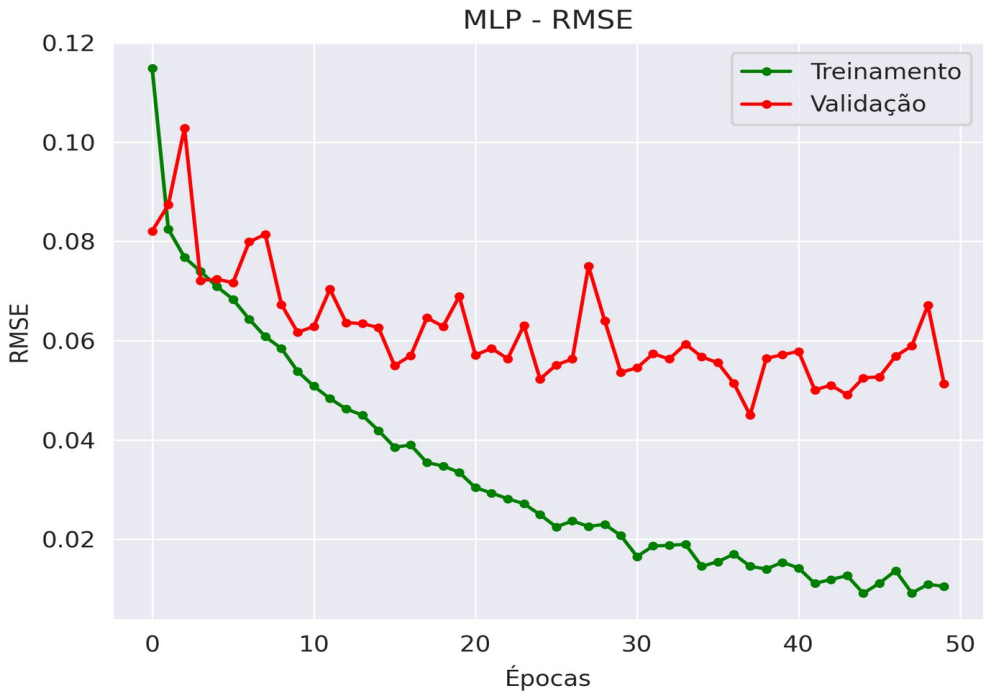
Figura 23 – MLP - Acurácia



Fonte – Acervo do autor

A figura abaixo apresenta o histórico da métrica RMSE do treinamento e da validação em todas as 50 épocas utilizadas para o melhor modelo MLP.

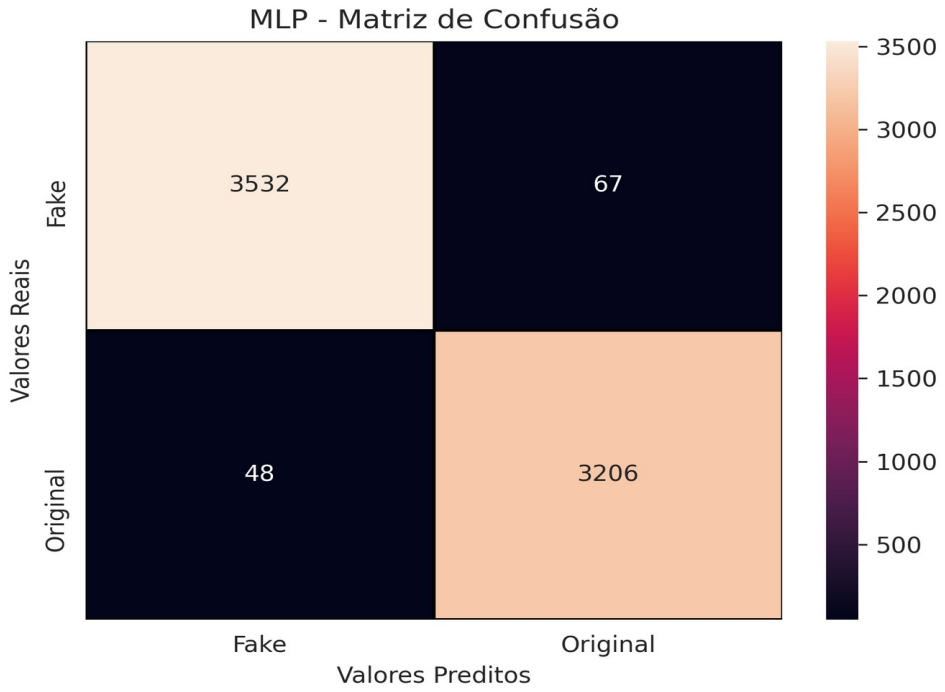
Figura 24 – MLP - RMSE



Fonte – Acervo do autor

A figura abaixo apresenta a Matriz de Confusão para o melhor modelo MLP, neste caso o modelo avaliou corretamente 3532 notícias como falsas, o modelo avaliou de forma errada 67 notícias como falsas, o modelo avaliou de forma errada 48 notícias como verdadeiras e o modelo avaliou corretamente 3206 notícias como verdadeiras.

Figura 25 – MLP - Matriz de confusão



Fonte – Acervo do autor

5.2 MODELO LSTM

Neste capítulo é apresentado o treinamento e resultado das diferentes configurações realizadas com o modelo LSTM (*Long Short Term Memory*). Nesse sentido, todos os testes apresentados abaixo foram realizados com algumas configurações fixas, que podem ser visualizadas na tabela abaixo.

Tabela 11 – Testes modelo LSTM – Informações fixas

Informação	Valor
Número de Camadas	3 (1 camada de entrada do tipo LSTM, 1 camada intermediária do tipo LSTM, 1 camada de saída do tipo MLP/Dense)
Quantidade de Épocas	20
Kernel Initializer	*Uniform*
Algoritmo de otimização	ADAM

Fonte: Acervo do autor

Os testes com o modelo LSTM apresentados abaixo são realizados com base no *dataset* de 50 palavras, devido ao fato de que o modelo MLP utilizando os *datasets* de 50 e 100 palavras não resultou uma diferença de desempenho muito grande. Além disso, utilizar textos com 50 palavras se torna mais eficiente e mais amplo para a utilização com pequenos textos ou apenas uma parte dos textos.

5.2.1 Modelo LSTM com o *dataset* de 50 palavras

O primeiro teste realizado, foi utilizando funções de ativações variadas em todas as camadas, podendo variar entre *sigmoid*, *tanh*, *elu* e *relu*. Além disso, o *batch size* será igual a 32 e como as configurações serão montadas com três camadas, a quantidade de neurônios da primeira camada será 12, da segunda será 8 e da terceira será 1.

Foram obtidas 64 configurações de modelos, dessa forma os três melhores modelos são apresentados abaixo (No qual possuem os melhores RMSE e F1 Score).

- **Modelo 01**
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 12;
  - Camada 02: Ativação relu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1720; Acurácia Modelo (%): 93.6791;
  - RMSE: 0.0895; F1 Score: 0.9242;
- **Modelo 02**
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 12;
  - Camada 02: Ativação tanh; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1764; Acurácia Modelo (%): 94.0032;
  - RMSE: 0.0945; F1 Score: 0.9303;
- **Modelo 03**
  - *Batch size*: 32;
  - Camada 01: Ativação tanh; Neurônios: 12;
  - Camada 02: Ativação elu; Neurônios: 8;
  - Camada 03: Ativação sigmoid; Neurônios: 1;

- *Loss*: 0.1636; Acurácia Modelo (%): 94.0032;
- RMSE: 0.0945; F1 Score: 0.9290;

O segundo teste é realizado pegando os três melhores modelos do primeiro teste e utilizado a quantidade de neurônios variada, assim como é realizado com o modelo MLP. Dessa forma, a primeira e a segunda camada variam a quantidade de neurônios entre 1, 2, 4, 8, 10, 25, 30, 50, 75, 90, 100, 128, 150, 200 e 300 e a terceira camada sempre possuirá 1 neurônio.

Para o modelo 01, uma das melhores configurações obtidas pode ser visualizada abaixo (Nesse caso possui um dos melhores resultados para o RMSE e o F1 Score).

- **Modelo 04**
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 150;
  - Camada 02: Ativação relu; Neurônios: 2;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1706; Acurácia Modelo (%): 94.8136;
  - RMSE: 0.0932; F1 Score: 0.9407;

Para o modelo 02, a melhor configuração encontrada, no caso com o melhor Loss, o segundo melhor RMSE e um dos melhores F1 Score, pode ser visualizada abaixo.

- **Modelo 05**
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1489; Acurácia Modelo (%): 95.1378;
  - RMSE: 0.0737; F1 Score: 0.9452;

A melhor configuração obtida para o modelo 03 pode ser visualizada abaixo (Nesse caso possui o maior percentual F1 Score e um dos menores valores para o RMSE).

- **Modelo 06**
  - *Batch size*: 32;
  - Camada 01: Ativação tanh; Neurônios: 300;
  - Camada 02: Ativação elu; Neurônios: 50;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1443; Acurácia Modelo (%): 94.3274;
  - RMSE: 0.0746; F1 Score: 0.9504;

O terceiro teste é realizado pegando o melhor resultado de cada um dos três modelos anteriores e utilizado valores variados para o *batch size*, dessa forma o *batch size* pode variar entre os seguintes valores: 1, 2, 5, 10, 25, 32, 50.

Para o modelo 04 a melhor configuração obtida pode ser visualizada abaixo (Esse modelo possui o melhor RMSE e o melhor F1 Score).

- **Modelo 07**
  - *Batch size*: 5;
  - Camada 01: Ativação elu; Neurônios: 150;
  - Camada 02: Ativação relu; Neurônios: 2;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1660; Acurácia Modelo (%): 94.6515;
  - RMSE: 0.0863; F1 Score: 0.9455;

A melhor configuração obtida (no qual possui o segundo menor valor para o RMSE e a melhor porcentagem F1 Score) para o modelo 05 pode ser visualizada abaixo.

- **Modelo 08**
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1611; Acurácia Modelo (%): 94.0032;
  - RMSE: 0.0929; F1 Score: 0.9475;

Para o modelo 06 a melhor configuração obtida pode ser visualizada abaixo (No qual possui o melhor RMSE e o melhor F1 Score).

- **Modelo 09**
  - *Batch size*: 1;
  - Camada 01: Ativação tanh; Neurônios: 300;
  - Camada 02: Ativação elu; Neurônios: 50;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1483; Acurácia Modelo (%): 94.4895;
  - RMSE: 0.0735; F1 Score: 0.9523;

O quarto teste foi realizado pegando o melhor resultado de cada um dos três modelos anteriores e adicionando mais uma camada intermediária para os modelos, essa camada é uma cópia da outra camada intermediária que existe. Entretanto, os resultados não foram melhores que os modelos anteriores, dessa forma esse teste não é apresentado.

O quinto teste é realizado pegando a melhor configuração de cada um dos três modelos anteriores (No caso os modelos 07, 08 e 09) e incrementado o número de épocas com o objetivo de melhorar os resultados.

- **Modelo 10**
  - Épocas: 50;
  - *Batch size*: 5;
  - Camada 01: Ativação elu; Neurônios: 150;
  - Camada 02: Ativação relu; Neurônios: 2;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1253; Acurácia Modelo (%): 96.2723;
  - RMSE: 0.0671; F1 Score: 0.9524;
- **Modelo 11**
  - Épocas: 50;
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1444; Acurácia Modelo (%): 95.2998;
  - RMSE: 0.0635; F1 Score: 0.9662;
- **Modelo 12**
  - Épocas: 50;
  - *Batch size*: 1;
  - Camada 01: Ativação tanh; Neurônios: 300;
  - Camada 02: Ativação elu; Neurônios: 50;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.4160; Acurácia Modelo (%): 87.0340;
  - RMSE: 0.1482; F1 Score: 0.8537;

5.2.2 Melhor modelo LSTM

Avaliando as métricas de todos os resultados acima com o modelo LSTM e o *dataset* com 50 palavras, o modelo 11 apresentado abaixo se destaca um pouco em relação aos outros modelos, devido ao fato de ter o melhor F1 Score e o menor RMSE de todos os modelos LSTM apresentados acima.

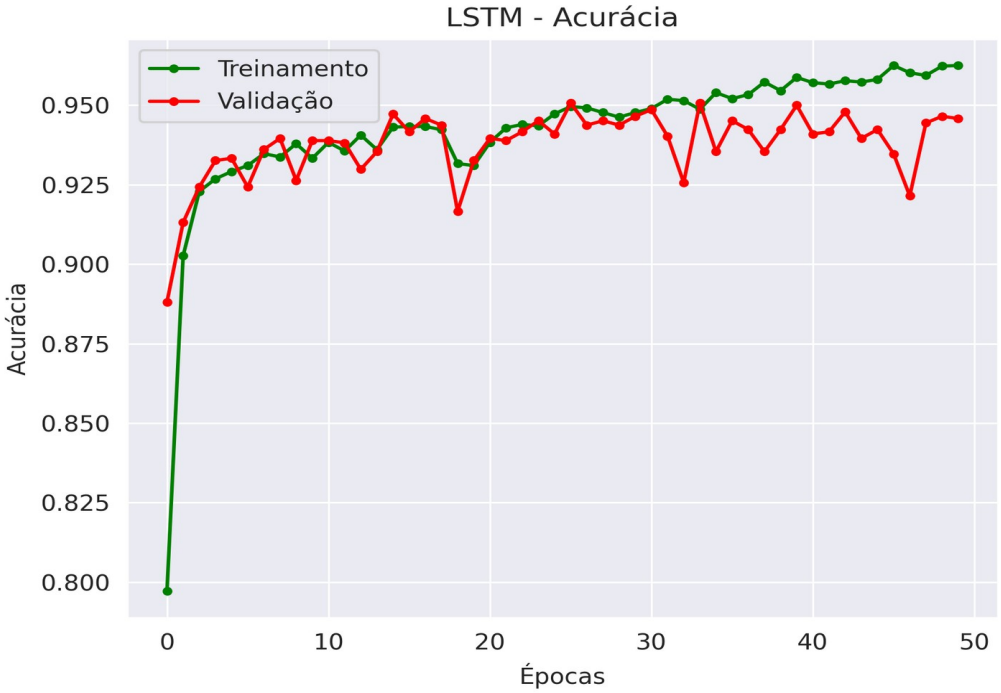
- **Modelo 11**
  - Épocas: 50;
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1444; Acurácia Modelo (%): 95.2998;
  - RMSE: 0.0635; F1 Score: 0.9662;

A seguir é apresentado alguns gráficos sobre o desempenho do modelo 11, entretanto rodando novamente a detecção com esse modelo os resultados serão diferentes, devido ao fato de uma rede neural artificial não gerar sempre os mesmo resultados. Dessa forma, após executar diversas vezes o modelo 11 o resultado abaixo foi obtido.

- **Melhor modelo utilizando a RNA LSTM**
  - Número de camadas: 3 (1 camada de entrada do tipo LSTM, 1 camada intermediária do tipo LSTM, 1 camada de saída do tipo MLP);
  - Quantidade de Épocas: 50;
  - *Kernel Initializer*: *Uniform;*
  - Algoritmo de otimização: ADAM;
  - *Batch size*: 32;
  - Camada 01: Ativação elu; Neurônios: 300;
  - Camada 02: Ativação tanh; Neurônios: 90;
  - Camada 03: Ativação sigmoid; Neurônios: 1;
  - *Loss*: 0.1107; Acurácia Modelo (%): 95.2998;
  - RMSE: 0.0551; F1 Score: 0.9627;

A figura abaixo apresenta o histórico da acurácia do treinamento e da validação em todas as 50 épocas utilizadas para o melhor modelo LSTM.

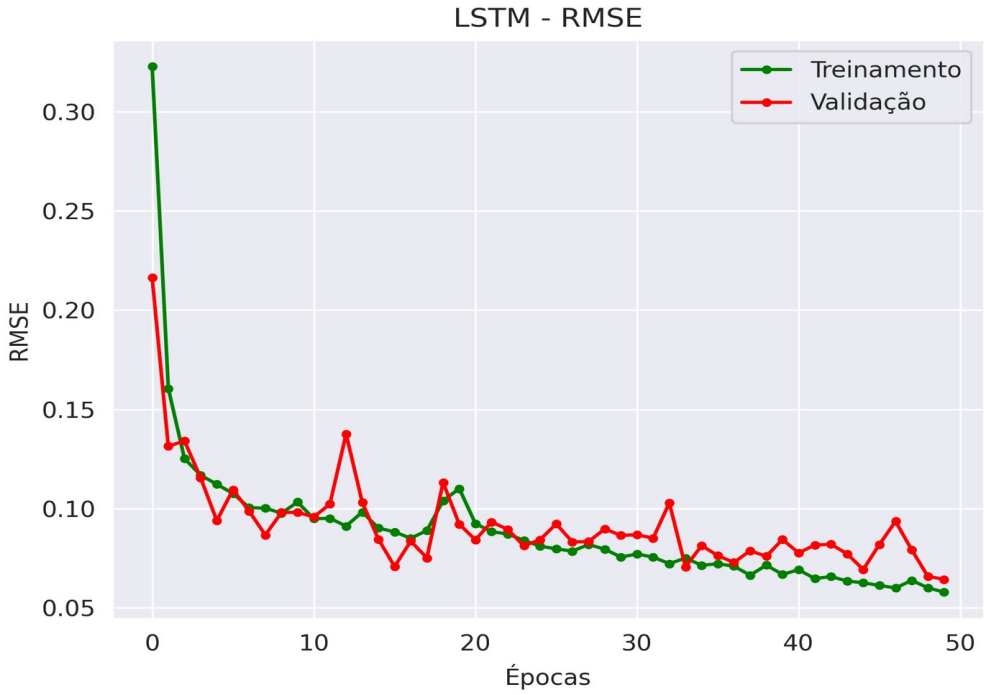
Figura 26 – LSTM - Acurácia



Fonte – Acervo do autor

A figura abaixo apresenta o histórico da métrica RMSE do treinamento e da validação em todas as 50 épocas utilizadas para o melhor modelo LSTM.

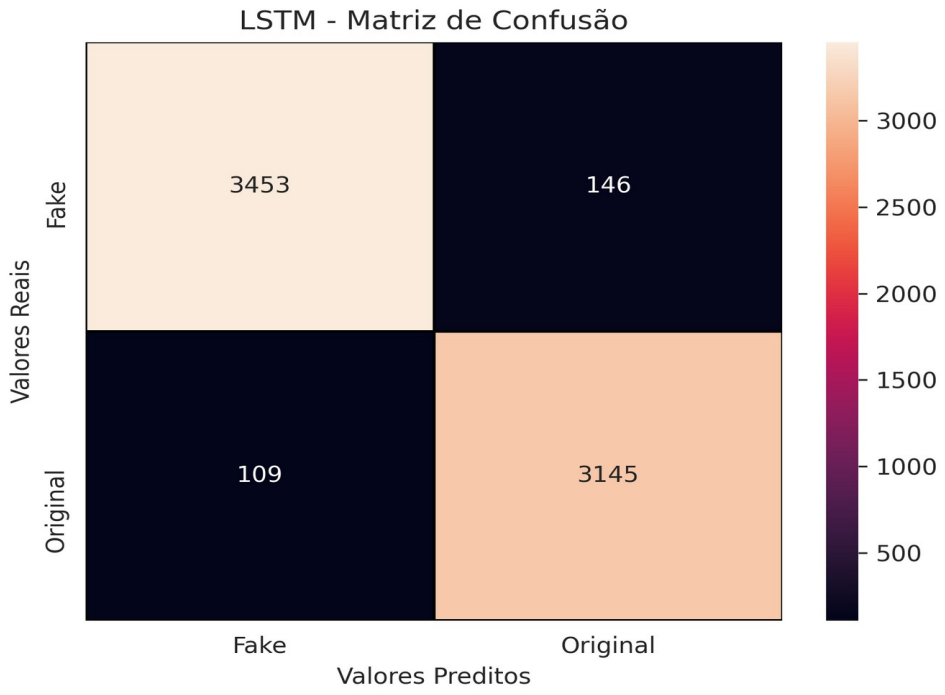
Figura 27 – LSTM - RMSE



Fonte – Acervo do autor

A figura abaixo apresenta a Matriz de Confusão para o melhor modelo LSTM, neste caso o modelo avaliou corretamente 3453 notícias como falsas, o modelo avaliou de forma errada 146 notícias como falsas, o modelo avaliou de forma errada 109 notícias como verdadeiras e o modelo avaliou corretamente 3154 notícias como verdadeiras.

Figura 28 – LSTM - Matriz de confusão



Fonte – Acervo do autor

6 CONCLUSÃO

As falsas notícias se encontram constantemente no nosso dia a dia e são um problema que deve ser combatido. Se encontram amplamente espalhadas pela internet e possuem como objetivo enganar seus leitores a partir de fatos que são intencionalmente falsos. Dessa forma, esse tipo de notícia quando compartilhada garante poder e vantagem, além disso pode causar julgamento precipitado, preconceitos, violência verbal e discriminação.

Existem diversos desafios para detectar uma *fake news*, muitas vezes elas se baseiam em histórias verdadeiras com poucos detalhes falsos, o que dificulta seu reconhecimento. Além disso, a carência por formas de detecção de *fake news* na língua portuguesa incentivou a elaboração deste documento.

O objetivo deste trabalho foi realizar a detecção de falsas notícias em sites de notícias brasileiros disponíveis nas mídias sociais, no qual gerou a pergunta de pesquisa se é possível obter um resultado favorável na detecção de *fake news* utilizando como base a língua portuguesa. Para a realização dessa detecção, foram utilizados modelos e técnicas de Redes Neurais Artificiais.

Durante o desenvolvimento do trabalho, foi realizada uma revisão bibliográfica da literatura com o objetivo de identificar o estado da arte de modelos e métodos utilizados para a detecção de *fake news*. No qual, foi possível identificar que diversos modelos são utilizados, alguns modelos mais comuns (Como o modelo MLP), utilizados em diversas necessidades diferentes e outros mais específicos (Modelos de aprendizagem profunda, como o modelo LSTM) para a utilização com textos.

A partir da busca por trabalhos correlatos, também foi possível observar as algumas características dos melhores modelos obtidos, as métricas utilizadas para avaliar o desempenho dos modelos e quais são as formas mais comuns e eficientes para realizar a conversão dos textos para representações numéricas.

Após a busca dos trabalhos correlatos, foi realizada a obtenção do conjunto de dados com falsas e verdadeiras notícias, o que não foi uma tarefa fácil, visto que a maioria dos *datasets* desse tipo se encontram na língua inglesa. Contudo, após diversas pesquisas foi encontrado o conjunto de dados “Fake.Br Corpus”[4].

4 https://github.com/roneysco/Fake.br-Corpus

Com o *dataset* obtido, foi necessário realizar alguns tratamentos nos textos e então foi realizado a conversão para representação numérica por meio da ferramenta Doc2Vec. Após essas etapas, as notícias foram organizadas em dois arquivos no formato “csv”, no qual um desses arquivos possui apenas as notícias com mais de 50 palavras e o outro arquivo possui apenas notícias com mais de 100 palavras, dessa forma os modelos serão treinados, validados e testados utilizando apenas as notícias maiores que o tamanho especificado.

Para a detecção de *fake news*, foram utilizados dois modelos, o MLP (*Multilayer Perceptron*) e o LSTM (*Long Short Term Memory*). Dessa forma, foram treinadas, validadas e testadas 2274 configurações, sendo 757 referentes ao modelo MLP com o *dataset* de 50 palavras, 757 referentes ao modelo MLP com o *dataset* de 100 palavras e 760 referentes ao modelo LSTM com o *dataset* de 50 palavras.

Para avaliar o desempenho das configurações criadas com os modelos, foram utilizadas as métricas RMSE e F1 Score. Dessa forma, avaliando as métricas foi selecionado a melhor configuração para cada um dos modelos, neste caso MLP e LSTM.

A configuração com o melhor desempenho obtida para a detecção de *fake news* a partir da rede MLP possui a quantidade de épocas igual a 50, o *kernel initializer* é o *uniform*, o algoritmo de otimização utilizado é o ADAM, o *batch size* é igual a 2 e o número de camadas é igual a 3 (sendo 1 camada de entrada, 1 camada intermediária e 1 camada de saída), dessa forma: A camada 01 possui a função de ativação *relu* e a quantidade de neurônios é 300, a camada 02 possui a função de ativação *tanh* e a quantidade de neurônios igual a 128 e a última camada possui a função de ativação *sigmoid* e a quantidade de neurônios é igual a 1. Com isso, os resultados obtidos foram: RMSE igual a 0.0439 e F1 Score igual a 98.32%.

Para a rede LSTM, a configuração com o melhor desempenho obtida para a detecção de *fake news* possui a quantidade de épocas igual a 50, o *kernel initializer* é o *uniform*, o algoritmo de otimização é o ADAM, o *batch size* é igual a 32 e o número de camadas é igual a 3 (sendo 1 camada de entrada do tipo LSTM, 1 camada intermediária do tipo LSTM e 1 camada de saída do tipo MLP), dessa forma a camada 01 possui a função de ativação *elu* e a quantidade de neurônios é igual a 300, a camada 02 possui a função de ativação *tanh* e a quantidade de neurônios é igual a 90 e a última camada possui a função de ativação *sigmoid* e quantidade de neurônios é igual a 1. Com isso, os resultados obtidos foram: RMSE: 0.0551 e F1 Score: 96.27%.

De acordo com os dois resultados acima, percebe-se que ambos os resultados foram satisfatórios e geraram ótimos resultados, além disso mesmo a rede MLP não sendo específica para a utilização com textos ela obteve, neste trabalho, melhores resultados quando comparada à rede LSTM.

Dessa forma, a pergunta de pesquisa foi respondida com sucesso e com resultados com baixo erro, que provam que a partir de um bom conjunto de dados, modelos e configurações corretas é possível detectar, com um ótimo desempenho, falsas notícias na língua portuguesa.

Deste modo, para a realização de trabalhos futuros, propõe-se realizar novas pesquisas com o objetivo de encontrar novas formas de conversão dos textos ou, até mesmo, aprimorar a forma de conversão utilizada neste trabalho. Outra ideia seria também aumentar o número de falsas e verdadeiras notícias do conjunto de dados. Além disso, uma futura atualização para este trabalho poderia ser realizada a partir de pesquisas e implementações de novos modelos com o objetivo de comparar com os antigos modelos e melhorar ainda mais os resultados.

REFERÊNCIAS

ADBULLAH, Malak; QAWASMEH, Ethar; TAWALBEH, Mais. Automatic Identification of Fake News Using Deep Learning. **2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)**. IEEE. 2019.

ALMEIDA, Diogo Medeiros de. **Modelos Híbridos de Séries Temporais Aplicados ao Sistema Automotivo On-Board Diagnostics**. 2018. 75 f. Monografia (Especialização) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2018.

AMAZON. **Amazon Machine Learning**: guia do desenvolvedor. 2021. Disponível em: https://docs.aws.amazon.com/pt_br/machine-learning/latest/dg/machinelearning-dg.pdf#regression-model-insights. Acesso em: 05 fev. 2021.

AOS FATOS. **Aos Fatos | Valorize o que é real**. Disponível em: https://aosfatos.org/. Acesso em: 08 abr. 2020.

BARBOSA, Maria Luciene Sampaio; SANTI, Vilso Junior. A intencionalidade nas notícias falsas: A nota de repúdio como estratégia de defesa do jornalismo na era das fake news. **Revista Pan-amazônica de Comunicação**, Palmas, v. 3, n. 3, p. 93-109, 2019.

BAUM, Matthew A. et al. The science of fake news. **American Association for the Advancement of Science**, 1200 New York Avenue NW, Washington, DC 20005. Vol. 359, mar. 2018.

BENGIO, Yoshua; GOODFELLOW, Ian; COURVILLE, Aaron. **Deep Learning**. The MIT Press, 2015.

CHATURVEDI, Anumeha. **2019 – The year of fake news**. Disponível em: https://economictimes.indiatimes.com/news/politics-and-nation/fake-news-still-a-menace-despite-government-crackdown-fact-checkers/articleshow/72895472.cms. Acesso em: 08 abr. 2020.

CLEVERT, Djork-Arne; UNTERTHINER, Thomas; HOCHREITER, Sepp. FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS (ELUS). **ICLR**. Linz, Austria, p. 1-1. jan. 2016.

DENG, Li; YU, Dong. Deep Learning: Methods and Applications. **Foundations and Trends® in Signal Processing**. Vol. 7. 2013.

DELMAZO, Caroline; VALENTE, Jonas C. L.. Fake news on online social media: propagation and reactions to misinformation in search of clicks: Propagation and reactions to misinformation in search of clicks. **Media & Jornalismo**, v. 18, n. 32, p. 155-169, 2018. Coimbra University Press.

FACURE, Matheus. **Funções de Ativação**: entendendo a importância da ativação correta nas redes neurais. 2017. Disponível em: https://matheusfacure.github.io/2017/07/12/activ-func/. Acesso em: 04 fev. 2021.

FAKE NEWS. In: DICIONÁRIO Cambridge Advanced Learner's Dictionary & Thesaurus. Cambridge University Press, 2020.

FURTADO, Maria Inês Vasconcellos. Redes Neurais Artificiais: Uma Abordagem Para Sala de Aula. **Ponta Grossa: Atena Editora**, 2019.

FINOCCHIO, Marco Antonio Ferreira. **Noções de Redes Neurais Artificiais**. 2014. Universidade Tecnológica Federal do Paraná.

GEREME, Fantahun Bogale; ZHU, William. Early Detection of Fake News "Before It Flies High". **Proceedings Of The 2nd International Conference On Big Data Technologies – Icbdt2019**, 2019. ACM Press.

GOMES, Gecynalda Soares da Silva. **Novas funções de ativação em redes neurais artificiais multilayer perceptron**. 2010. 137 f. Tese (Doutorado) - Curso de Ciência da Computação, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2010.

GUALDA, Isabella Peres. **Aplicação de Redes Neurais Artificiais na Ciência e Tecnologia de Alimentos: Estudo de Casos**. 2008. 152 f. Dissertação (Mestrado) - Curso de Ciência de Alimentos, Universidade Estadual de Londrina, Londrina, 2008.

HAYKIN, Simon S. Neural networks and learning machines. Third. Upper Saddle River, **NJ: Pearson Education**, 2009.

HIGINO, Jéssica. **Checagem contra fake news na campanha presidencial de 2018: O jornalismo do o Globo e a verificação da agência Lupa**. 2019. 31 f. TCC (Graduação) - Curso de Jornalismo, Universidade do Sul de Santa Catarina – Unisul, 2019.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, [s.l.], v. 9, n. 8, p. 1735-1780, nov. 1997. MIT Press - Journals. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

HUNTER, John D.. **Matplotlib**: a 2d graphics environment. Computing In Science & Engineering, [S.L.], v. 9, n. 3, p. 90-95, 2007. Institute of Electrical and Electronics Engineers (IEEE).

IFCN. **IFCN Code of Principles**. Disponível em: https://ifcncodeofprinciples.poynter.org/signatories. Acesso em: 08 abr. 2020.

JONES, M. Tim. **Um mergulho profundo nas redes neurais recorrentes**. 2017. Disponível em: https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes. Acesso em: 09 maio 2020.

KALIYAR, Rohit Kumar. Fake News Detection Using A Deep Neural Network. **2018 4th International Conference On Computing Communication And Automation (iccca)**, [s.l.], dez. 2018. IEEE.

KERAS, Keras. **Keras**: Simples. Flexível. Poderoso. Acesso em: 30 setembro 2020. Disponível em: https://keras.io/.

KIM, Donghwa; SEO, Deokseong; CHO, Suhyoun; KANG, Pilsung. Multi-co-training for document classification using various document representations: tf-idf, lda, and doc2vec. **Information Sciences**, [S.L.], v. 477, p. 15-29, mar. 2019. Elsevier BV. http://dx.doi.org/10.1016/j.ins.2018.10.006.

KINGMA, Diederik P.; BA, Jimmy Lei. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. In: ICLR, 2015.

LE, Xuan-hien et al. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. **Water**, [s.l.], v. 11, n. 7, p. 1387, 5 jul. 2019. MDPI AG. http://dx.doi.org/10.3390/w11071387.

LEE, Hana; YOON, Young. Engineering doc2vec for automatic classification of product descriptions on O2O applications. **Electronic Commerce Research**, [S.L.], v. 18, n. 3, p. 433-456, 28 ago. 2017. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s10660-017-9268-5.

LUPA, Equipe. **O que é a Agência Lupa?** 2015. Disponível em: https://piaui.folha.uol.com.br/lupa/2015/10/15/como-selecionamos-as-frases-que-serao-checadas/. Acesso em: 18 maio 2020.

MARUMO, Fabiano Shiiti. **Deep Learning para classificação de Fake News por sumarização de texto**. 2018. 54f. – Universidade Estadual de Londrina, Londrina, 2018.

MICROSOFT, Microsoft. **Configurar e gerenciar palavras irrelevantes e listas de palavras irrelevantes (stoplists) para pesquisa de texto completo**. Acesso em: 25 setembro 2020. Disponível em: https://docs.microsoft.com/pt-br/sql/relational-databases/search/configure-and-manage-stopwords-and-stoplists-for-full-text-search?view=sql-server-ver15.

MONTEIRO, Rafael A. et al. (2018) Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results. In: Villavicencio A. et al. (eds) Computational Processing of the Portuguese Language. **PROPOR 2018**. Lecture Notes in Computer Science, vol 11122. Springer, Cham

NELSON, David Michael Quirino. **Uso de redes neurais recorrentes para previsão de séries temporais financeiras**. 2017. 73 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, Belo Horizonte, 2017.

NLTK, Nltk. **Natural Language Toolkit**. Acesso em: 24 setembro 2020. Disponível em: https://www.nltk.org/.

NUMPY. NumPy. **NumPy v1.19.0**. 2020. Acesso em: 21 setembro 2020. Disponível em: https://numpy.org/.

OLAH, Christopher. **Understanding LSTM Networks**. 2015. Disponível em: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Acesso em: 13 abr. 2020.

OTTO, Saskia A.. **HOW TO NORMALIZE THE RMSE**. 2019. Disponível em: https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse/. Acesso em: 05 fev. 2021.

PANDAS, Pandas. **pandas:** pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Acesso em: 21 setembro 2020. Disponível em: https://pandas.pydata.org/.

ŘEHŮŘEK, Radim; SOJKA, Petr. Software Framework for Topic Modelling with Large Corpora. **ELRA**, p. 45-50, 22 mai. 2010. Valletta, Malta.

RIBEIRO, Márcio Moretto; ORTELLADO, Pablo. O que são e como lidar com as notícias falsas. **Revista Internacional de Direitos Humanos**, Sc, v. 5, p. 71-83, 2018.

RIPLEY, Brian. **Pattern Recognition and Neural Networks**. Cambridge: Cambridge University Press, 1996. 415 p.

RODRIGUES, Vilson. **Machine Learning: o que são Acurracy, Precision, Recall e F1 Score**. 2019. Disponível em: https://medium.com/@vilsonrodrigues/machine-learning-o-que-s%C3%A3o-acurracy-precision-recall-e-f1-score-f16762f165b0. Acesso em: 12 fev. 2021.

RUDER, Sebastian. **An overview of gradient descent optimization algorithms**. Dublin: 2017. Disponível em: https://arxiv.org/pdf/1609.04747.pdf. Acesso em: 30 nov. 2020.

SCHADE, Gabriel. **Machine Learning**: métricas para modelos de classificação. Métricas para Modelos de Classificação. 2019. Disponível em: https://gabrielschade.github.io/2019/03/12/ml-classificacao-metricas.html. Acesso em: 08 fev. 2021.

SHARMA, Siddharth; SHARMA, Simone. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. **International Journal Of Engineering Applied Sciences And Technology**, Jaipur, v. 4, n. 1, p. 310-316, abr. 2020.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. Redes Neurais Artificiais: para engenharia e ciências aplicadas. **Artliber Editora Ltda**, São Paulo. 2016.

SPINELLI, Egle Müller; SANTOS, Jéssica de Almeida. JORNALISMO NA ERA DA PÓS-VERDADE: fact-checking como ferramenta de combate às fake news. **Revista Observatório**, [s.l.], v. 4, n. 3, p. 759-782, 29 abr. 2018. Universidade Federal do Tocantins.

RODRIGUES, Welington Galvão. **Predição de Diâmetros e Cálculo de Volume de Clones de Eucalipto**: uma abordagem com redes multilayer perceptron e long-short term memory. 2019. 95 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Goiás, Goiânia, 2019.

TANDOC, Edson C.; LIM, Zheng Wei; LING, Richard. Defining “Fake News”. **Digital Journalism**, [s.l.], v. 6, n. 2, p. 137-153, 30 ago. 2017. Informa UK Limited.

TIN, Pham T. A Study on Deep Learning for Fake News Detection. **Japan Advanced Institute of Science and Technology**. Mar, 2018.

TRAUMANN, Thomas. **Como a indústria de notícias falsas dominou a eleição da França**: Os franceses escolhem seu novo presidente numa campanha dominada por notícias falsas e escândalos. O vale-tudo virou padrão mundial. 2017. ÉPOCA. Disponível em: https://epoca.globo.com/mundo/noticia/2017/04/como-industria-de-noticias-falsas-dominou-eleicao-da-franca.html. Acesso em: 08 abr. 2020.

VERMA, Abhishek; MITTAL, Vanshika; DAWN, Suma. FIND: Fake Information and News Detections using Deep Learning. **2019 Twelfth International Conference On Contemporary Computing (ic3)**, ago. 2019. IEEE.

VETRITTI, Fabiana Grieco Cabral de Mello. Fact-checking: a Importância da Checagem das Informações para o Exercício da Cidadania da População em Rede. In: PENSACOM BRASIL, 2019, São Paulo.

VIEIRA, Vaninha; RAZENTE, Humberto L.; BARIONI, Maria Camila N.. **TOPICOS EM GERENCIAMENTO DE DADOS E INFORMAÇÕES 2017**. Uberlândia - Mg: Sociedade Brasileira de Computação – Sbc, 2017.

WASKOM, Michael. **seaborn**: statistical data visualization. Acesso em: 22 setembro 2020. Disponível em: https://seaborn.pydata.org/.

APÊNDICE A – CONVERSÃO DOS TEXTOS PARA REPRESENTAÇÃO NUMÉRICA

```
import time
import pandas as pd
from gensim import utils
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

# lista de CSVs com os tamanhos correspondentes que serão convertidos
TEXT_LENGTH = [50, 100, 150, 200]
PATH_DATASETS_FORMATTED = 'datasets/formatted/'
PATH_DATASETS_CONVERTED = 'datasets/converted/'



def constructSentences(data):
    """
    Método que realiza a construção do array de sentenças que será utilizado no doc2vec
    """
    sentences = []
    for index, row in data.iteritems():
        # converte para um formato legível para o computador
        sentences.append(TaggedDocument(utils.to_unicode(row).split(), [str(index)]))
    return sentences



def dataProcessing(df, vector_dimension):
    """
    Método responsável por realizar o processamento dos textos convertendo o texto para o
formato numérico


    :param df: Dataframe com as notícias
    :type df: Dataframe
    :param vector_dimension: Quantidade de palavras que serão convertidas nos textos
    :type vector_dimension: int
    :return: Retorna o modelo Doc2Vec
    :rtype: Doc2Vec
    """
    # realiza a construção das sentenças
    x = constructSentences(df['text'])

    # monta o modelo Doc2Vec
    model = Doc2Vec(
        min_count=1,
```

```
        window=5,
        vector_size=vector_dimension,
        sample=1e-4,
        negative=5,
        workers=7,
        epochs=10,
        seed=1,
    )
    model.build_vocab(x)
    model.train(x, total_examples=model.corpus_count, epochs=model.epochs)

    return model


def generateCSV(df, model, vector_dimension):
    """
    Método responsável por realizar a geração do CSV com os textos em representação numérica

    :param df: Dataframe com as notícias
    :type df: Dataframe
    :param model: Modelo Doc2Vec
    :type model: Doc2Vec
    :param vector_dimension: Quantidade de palavras que serão convertidas nos textos
    :type vector_dimension: int
    :return: Retorna o CSV convertido
    :rtype: DataFrame
    """
    # realiza a criação das colunas do novo CSV
    columns = ['ID', 'fake_news', *[f'word_{i}' for i in range(vector_dimension)]]

    # cria o novo dataframe
    df_converted = pd.DataFrame(columns=columns)
    # itera em cada notícia e realiza a inserção no CSV
    for i in range(len(model.docvecs)):
        # converte o array de palavras do texto para um dicionário
        words = {f'word_{j}': model.docvecs.vectors_docs[i][j] for j in range(vector_dimension)}

        # adicionado o dicionário junto com as outras props
        df_converted = df_converted.append(
            {'ID': df['ID'][i], 'fake_news': df['fake_news'][i], **words},
            ignore_index=True,
        )

    # realiza as conversão de algumas colunas para valores inteiros
```

```
    df_converted[['ID', 'fake_news']] = df_converted[['ID', 'fake_news']].astype('int64')

    return df_converted


def main():
    """
    Método main do script
    """
    try:
        print('Iniciando a conversão dos datasets para representação numérica... ')
        inicio = time.time()

        # realiza a conversao dos CSVs listados em TEXT_LENGTH
        for dataset_atual in TEXT_LENGTH:
            dataset_nome = 'dataset_%i_palavras.csv' % dataset_atual
            print('Convertendo o CSV %s...' % dataset_nome)

            # realiza a leitura do CSV
            df = pd.read_csv(PATH_DATASETS_FORMATTED + dataset_nome, index_col=0)

            # realiza o conversão dos textos para representação numérica
            print('Realizando a conversão dos textos para representação numérica... ')
            model = dataProcessing(df, dataset_atual)

            # realiza a geração do CSV
            print('Realizando a geração do CSV... ')
            df_converted = generateCSV(df, model, dataset_atual)
            # realiza a criação do novo CSV
            df_converted.to_csv(PATH_DATASETS_CONVERTED + dataset_nome)

        fim = time.time()
        print('CSVs com os textos formatados criados com sucesso! ')
        print('Tempo de execução: %.2f minutos' % ((fim - inicio) / 60))
    except Exception as e:
        print('Falha ao gerar CSV: %s' % str(e))


if __name__ == '__main__':
    main()
```

Fonte: Acervo do autor

APÊNDICE B – CLASSE GENÉRICA MODEL

```
import json
import matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from pathlib import Path
from keras import backend
from sklearn.metrics import confusion_matrix, classification_report
matplotlib.use('Agg')
# setando um estilo padrão
sns.set_style('darkgrid')


def rmse(y_true, y_pred):
    """
    Método responsável por realizar o cálculo do RMSE
    """
    return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))


class Model:
    """
    Classe que representa um modelo genérico não funcional
    """

    # tipos de camadas
    LAYER_MLP = 'dense'
    LAYER_LSTM = 'lstm'
    LAYER_DROPOUT = 'dropout'

    # funções de ativação
    ATIVACAO_SIGMOID = 'sigmoid'
    ATIVACAO_TANH = 'tanh'
    ATIVACAO_RELU = 'relu'
    ATIVACAO_ELU = 'elu'

    def __init__(self, model_name, info, data, dataset_name, path_graphics):
        """
        Construtor da classe
```

```
        :param model_name: Nome do modelo que foi instanciado
        :type model_name: str
        :param info: Dict com as informações epochs, batch_size e layers para montar o modelo
        :type info: dict
        :param data: Dados utilizados na detecção
        :type data: dict
        :param dataset_name: Nome do dataset utilizado
        :type dataset_name: str
        :param path_graphics: Diretório para salvar os gráficos gerados
        :type path_graphics: str
        """
        self.model = None
        self.model_name = model_name
        self.info = info
        self.epochs = info['epochs']
        self.batch_size = info['batch_size']
        self.layers = info['layers']
        self.data = data
        self.dataset_name = dataset_name
        self.path_graphics = path_graphics

    def _updateData(self):
        """
        Método para sobreescrever e realizar atualizações com os dados
        """
        return

    def _create_model(self):
        """
        Método para sobreescrever e realizar a construção do modelo
        """
        return

    def __compile(self):
        """
        Método responsável por compilar o modelo com as métricas: Acurácia e RMSE
        """
        self.model.compile(
            loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy', rmse],
        )

    def __train(self):
```

```
        """
        Método responsável por realizar o treinamento e validação do modelo
        """
        history = self.model.fit(
            self.data['x_train'],
            self.data['y_train'],
            epochs=self.epochs,
            batch_size=self.batch_size,
            validation_data=(self.data['x_val'], self.data['y_val'])
        )
        return history

    def __test(self):
        """
        Método responsável por realizar o teste do modelo
        """
        # avalia o modelo com os dados de teste
        m_loss, m_accuracy_model, m_rmse = self.model.evaluate(self.data['x_test'],
self.data['y_test'])

        # gera as detecções se cada notícia é fake ou não
        predict = self.model.predict(self.data['x'])

        # valida a acurácia das detecções
        self.predictRounded = [round(x[0]) for x in predict]
        m_accuracy_detection = np.mean(self.predictRounded == self.data['y'])

        # monta um dict das métricas e retorna
        return {
            'loss': m_loss,
            'accuracy_model': m_accuracy_model,
            'accuracy_detection': m_accuracy_detection,
            'rmse': m_rmse,
        }

    def __result(self, history, metrics):
        """
        Método responsável por apresentar os resultados

        :param history: Histórico das métricas da detecção
        :type history: Tuple[Any]
        :param metrics: Resultado obtido pelas métricas
        :type metrics: dict
        """
```

```
        result = '\n# Teste ' + self.model_name + ' - ' + str(datetime.now()) + ' - ' +
self.dataset_name + '\n'
        result += json.dumps(self.info)

        result += '\nmétricas: '
        # quanto menor a perda, mais próximas nossas previsões são dos rótulos verdadeiros.
        result += 'loss: %.4f; ' % metrics['loss']
        result += 'accuracy_model(%%): %.4f; ' % (metrics['accuracy_model'] * 100)
        result += 'accuracy_detection(%%): %.4f; ' % (metrics['accuracy_detection'] * 100)
        result += 'rmse: %.4f; ' % metrics['rmse']

        report = classification_report(self.data['y'], self.predictRounded, output_dict=True)
        # macro avg dá a cada previsão um peso semelhante ao calcular a perda
        result += 'precision: %.4f; ' % report['macro avg']['precision']
        result += 'recall: %.4f; ' % report['macro avg']['recall']
        result += 'f1-score: %.4f; ' % report['macro avg']['f1-score']

        cm = confusion_matrix(self.data['y'], self.predictRounded)
        result += 'confusion_matrix: '
        result += (str(cm[0, 0]) + '-' + str(cm[0, 1]) + '-' + str(cm[1, 0]) + '-' + str(cm[1, 1]) + ';')
        result += '\n'

        # apresenta os resultados e também salva no arquivo results.txt
        print(result)
        f = open('results/results.txt', 'a')
        f.write(result)
        f.close()

        # gera os gráficos
        self.__graphics(history, cm)
        # salva o modelo
        self.__save_model()

    def __graphics(self, history, cm):
        """
        Método responsável por gerar os gráficos

        :param history: Histórico das métricas da detecção
        :type history: Tuple[Any]
        :param cm: Matriz de confusão
        :type cm: list
        """
        # Apresentação dos gráficos de treinamento e validação da rede
        Path(self.path_graphics).mkdir(parents=True, exist_ok=True)
```

```
    # cm
    cm = pd.DataFrame(cm, index=['Fake', 'Original'], columns=['Fake', 'Original'])
    sns_plot = sns.heatmap(
        cm, linecolor='black', linewidth=1, annot=True, fmt='',
        xticklabels=['Fake', 'Original'], yticklabels=['Fake', 'Original']
    )
    sns_plot.set_title(self.model_name + ' - Matriz de Confusão')
    sns_plot.set_xlabel('Valores Preditos')
    sns_plot.set_ylabel('Valores Reais')
    plt.savefig(self.path_graphics + 'cm.png', dpi=300)
    plt.close()

    # rmse
    plt.plot(history.history['rmse'], 'go-', markersize=3, label='Treinamento')
    plt.plot(history.history['val_rmse'], 'ro-', markersize=3, label='Validação')
    plt.title(self.model_name + ' - RMSE')
    plt.xlabel('Épocas')
    plt.ylabel('RMSE')
    plt.legend()
    plt.savefig(self.path_graphics + 'rmse.png', dpi=300)
    plt.close()

    # acurácia
    plt.plot(history.history['accuracy'], 'go-', markersize=3, label='Treinamento')
    plt.plot(history.history['val_accuracy'], 'ro-', markersize=3, label='Validação')
    plt.title(self.model_name + ' - Acurácia')
    plt.xlabel('Épocas')
    plt.ylabel('Acurácia')
    plt.legend()
    plt.savefig(self.path_graphics + 'acuracia.png', dpi=300)
    plt.close()

    # loss
    plt.plot(history.history['loss'], 'go-', markersize=3, label='Treinamento')
    plt.plot(history.history['val_loss'], 'ro-', markersize=3, label='Validação')
    plt.title(self.model_name + ' - Loss')
    plt.xlabel('Épocas')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig(self.path_graphics + 'loss.png', dpi=300)
    plt.close()

def __save_model(self):
```

```
        """
        Método responsável por salvar o modelo
        """
        self.model.save('results/modelo_%s.h5' % self.model_name)

    def predict(self):
        """
        Método que realiza todas as etapas para detecção de Fake News com o modelo
        """
        try:
            print('Criando o modelo %s... ' % self.model_name)
            self._updateData()
            self._create_model()
            self.__compile()

            print('Treinando e validando o modelo... ')
            history = self.__train()

            print('Testando o modelo... ')
            metrics = self.__test()

            print('Resultados: ')
            self.__result(history, metrics)
        except Exception as e:
            print('Erro! %s' % str(e))
```

Fonte: Acervo do autor

APÊNDICE C – CLASSE PARA CRIAÇÃO DO MODELO MLP

```
from keras.models import Sequential
from models.Model import Model
from keras.layers import Dense


class ModelMLP(Model):
    """
    Classe que representa o modelo MLP
    """

    def __init__(self, info, data, dataset_name):
        """
        Construtor da classe

        :param info: Dict com as informações epochs, batch_size e layers para montar o modelo
            As camadas para montar o modelo estão no formato:
            [{'qtd_neurons': qtd_neuronios, 'activation': 'funcao_ativacao',}]
        :type info: dict
        :param data: Dados utilizados na detecção
        :type data: dict
        :param dataset_name: Nome do dataset utilizado
        :type dataset_name: str
        """
        super().__init__('MLP', info, data, dataset_name, 'graphics/mlp/')

    def _create_model(self):
        """
        Método responsável por realizar a criação do modelo
        """
        # passa as camadas para uma nova variável sem ser por referência
        layers = self.layers[:]

        # valida se foram informados pelo menos 3 camadas: entrada, intermediária 01 e saída
        if len(layers) < 3:
            raise Exception('ERRO! ')

        # instacia o modelo
        self.model = Sequential()

        # insere a camada de entrada
        input_layer = layers.pop(0)
        self.model.add(Dense(
```

```
            input_layer['qtd_neurons'],
            kernel_initializer='uniform',
            activation=input_layer['activation'],
            input_dim=self.data['x_train'].shape[1],
        ))

        # insere as camadas intermediárias e de saída
        for layer in layers:
            self.model.add(Dense(
                layer['qtd_neurons'],
                kernel_initializer='uniform',
                activation=layer['activation'],
            ))
```

Fonte: Acervo do autor

APÊNDICE D - CLASSE PARA CRIAÇÃO DO MODELO LSTM

```
from keras.models import Sequential
from models.Model import Model
from keras.layers import Dense, LSTM, Dropout


class ModelLSTM(Model):
    """
    Classe que representa o modelo LSTM
    """

    def __init__(self, info, data, dataset_name):
        """
        Construtor da classe

        :param info: Dict com as informações epochs, batch_size e layers para montar o modelo
            As camadas para montar o modelo estão no formato:
            [{'qtd_neurons': qtd_neuronios, 'activation': 'funcao_ativacao',}]
        :type info: dict
        :param data: Dados utilizados na detecção
        :type data: dict
        :param dataset_name: Nome do dataset utilizado
        :type dataset_name: str
        """
        super().__init__('LSTM', info, data, dataset_name, 'graphics/lstm/')

    def _updateData(self):
        """
        Método sobreescrito para realizar atualizações com o dataset
        """
        # realiza o reshape dos dados para serem utilizados no modelo LSTM
        self.data['x'] = self.data['x'].reshape(
            self.data['x'].shape[0], self.data['x'].shape[1], 1
        )
        self.data['x_train'] = self.data['x_train'].reshape(
            self.data['x_train'].shape[0], self.data['x_train'].shape[1], 1
        )
        self.data['x_test'] = self.data['x_test'].reshape(
            self.data['x_test'].shape[0], self.data['x_test'].shape[1], 1
        )
        self.data['x_val'] = self.data['x_val'].reshape(
            self.data['x_val'].shape[0], self.data['x_val'].shape[1], 1
        )
```

```
def _create_model(self):
    """
    Método responsável por realizar a criação do modelo
    """
    # passa as camadas para uma nova variável sem ser por referência
    layers = self.layers[:]
    # instacia o modelo
    self.model = Sequential()

    # insere a camada de entrada
    input_layer = layers.pop(0)
    self.model.add(LSTM(
        input_layer['qtd_neurons'],
        kernel_initializer='uniform',
        activation=input_layer['activation'],
        # input_shape=(NUM_ENTRADAS, QTD_INFO_POR_ENTRADA)
        input_shape=(self.data['x_train'].shape[1], self.data['x_train'].shape[2]),
        return_sequences=True if input_layer.get('return_sequences') else False
    ))

    # insere as camadas intermediárias e de saída
    for layer in layers:
        # valida se é uma camada LSTM
        if layer['type'] == Model.LAYER_LSTM:
            self.model.add(LSTM(
                layer['qtd_neurons'],
                kernel_initializer='uniform',
                activation=layer['activation'],
                # input_shape=(NUM_ENTRADAS, QTD_INFO_POR_ENTRADA)
                return_sequences=True if layer.get('return_sequences') else False
            ))
            continue
        # valida se é uma camada MLP
        if layer['type'] == Model.LAYER_MLP:
            self.model.add(Dense(
                layer['qtd_neurons'],
                kernel_initializer='uniform',
                activation=layer['activation'],
            ))
            continue
        # valida se é uma camada de DROPOUT
        if layer['type'] == Model.LAYER_DROPOUT:
            self.model.add(Dropout(layer['value']))
```

```
            continue
        raise Exception('Camada inválida! ')
```

Fonte: Acervo do autor

APÊNDICE E – SCRIPT DE DETECÇÃO DE FAKE NEWS COM OS MODELOS MLP E LSTM

```
import time
import pandas as pd
from models.Model import Model
from models.ModelMLP import ModelMLP
from models.ModelLSTM import ModelLSTM
from sklearn.model_selection import train_test_split



def generateData(csv_name):
    """
    Busca os dados e organiza para utilização nos modelos

    :return: Retorna um dict com os dados
    :rtype: dict
    """
    # realiza a leitura do CSV
    df = pd.read_csv(csv_name, index_col=0)
    print('Dataset: ')
    print(df.head())

    # realiza a separação do dataset entre X e Y
    y = df['fake_news'].to_numpy()
    df = df.drop(columns=['ID', 'fake_news'])
    x = df.to_numpy()

    # divisão dos dados
    # treinamento => 70% | validação => 20% | teste => 10%
    x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.3)
    x_val, x_test, y_val, y_test = train_test_split(x_val, y_val, test_size=0.3)

    # agrupa os dados em um dicionário
    data = {
        'x': x,
        'x_train': x_train,
        'x_val': x_val,
        'x_test': x_test,
        'y': y,
        'y_train': y_train,
        'y_val': y_val,
        'y_test': y_test,
    }
```

```
    print('Quantidade de registros: %i ' % len(x))
    print('Quantidade de registros para treino: %i ' % len(x_train))
    print('Quantidade de registros para validação: %i ' % len(x_val))
    print('Quantidade de registros para teste: %i ' % len(x_test))
    return data


def generateMLP(data, dataset_name):
    """
    Realiza a detecção de fake news com o modelo MLP

    :param data: Dados utilizados no modelo
    :type data: dict
    :param dataset_name: Nome do dataset utilizado
    :type dataset_name: str
    """
    # MELHOR MODELO MLP NO DATASET DE 50 PALAVRAS
    model = {
        "epochs": 50,
        "batch_size": 2,
        "layers": [
            {"qtd_neurons": 300, "activation": "relu"},
            {"qtd_neurons": 128, "activation": "tanh"},
            {"qtd_neurons": 1, "activation": "sigmoid"}
        ]
    }
    model_mlp = ModelMLP(model, data, dataset_name)
    model_mlp.predict()


def generateLSTM(data, dataset_name):
    """
    Realiza a detecção de fake news com o modelo LSTM

    :param data: Dados utilizados no modelo
    :type data: dict
    :param dataset_name: Nome do dataset utilizado
    :type dataset_name: str
    """
    # MELHOR MODELO LSTM NO DATASET DE 50 PALAVRAS
    model = {
        "epochs": 50,
        "batch_size": 32,
        "layers": [
```

```
        {"qtd_neurons": 300, "activation": "elu", "return_sequences": True},
        {"type": "lstm", "qtd_neurons": 90, "activation": "tanh"},
        {"type": "dense", "qtd_neurons": 1, "activation": "sigmoid"}
    ]
  }
  model_lstm = ModelLSTM(model, data, dataset_name)
  model_lstm.predict()


PATH_DATASETS_FORMATTED = 'datasets/formatted/'
PATH_DATASETS_CONVERTED = 'datasets/converted/'


def main():
  """
  Método main do script
  """
  print('Iniciando a detecção de fake news')
  inicio = time.time()

  # testes com o dataset de 50 palavras no modelo MLP
  text_length = 50
  dataset_nome = PATH_DATASETS_CONVERTED + 'dataset_%i_palavras.csv' % text_length
  print('\n\n' + dataset_nome)
  data = generateData(dataset_nome)
  generateMLP(data, 'dataset_%i_palavras.csv' % text_length)

  # testes com o dataset de 50 palavras no modelo LSTM
  text_length = 50
  dataset_nome = PATH_DATASETS_CONVERTED + 'dataset_%i_palavras.csv' % text_length
  print('\n\n' + dataset_nome)
  data = generateData(dataset_nome)
  generateLSTM(data, 'dataset_%i_palavras.csv' % text_length)

  fim = time.time()
  print('Detecção de fake news realizada com sucesso! ')
  print('Tempo de execução: %.2f minutos' % ((fim - inicio) / 60))


if __name__ == '__main__':
  main()
```

Fonte: Acervo do autor