Instituto de MATEMÁTICA E ESTATÍSTICA UFRGS

DEPARTAMENTO DE ESTATÍSTICA
E-mail: dest@mat.ufrgs.br

Trabalho de Conclusão de Curso

Uma aplicação do modelo de Processamento de Linguagem Natural BERT para classificação de notícias falsas

Gabriel Holmer Saul

21 de setembro de 2023

Gabriel Holmer Saul

Uma aplicação do modelo de Processamento de Linguagem Natural BERT para classificação de notícias falsas

Trabalho de Conclusão apresentado à comissão de Graduação do Departamento de Estatística da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do título de Bacharel em Estatística.

Orientadora: Profa. Dra. Márcia Helena Barbian

Porto Alegre
9 de Setembro de 2023

Gabriel Holmer Saul

Uma aplicação do modelo de Processamento de Linguagem Natural BERT para classificação de notícias falsas

Este Trabalho foi julgado adequado para obtenção dos créditos da disciplina Trabalho de Conclusão de Curso em Estatística e aprovado em sua forma final pela Orientadora e pela Banca Examinadora.

Orientadora:_____
Profa. Dra. Márcia Helena Barbian, UFMG
Doutora pela Universidade Federal de Minas Gerais, Belo Horizonte, MG

Banca Examinadora:

Prof. Dr. João Henrique Ferreira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul, Porto Alegre, RS

Profa. Dra. Viviane Pereira Moreira, MDX
Doutora pela Middlesex University London, Inglaterra

Porto Alegre
9 de Setembro de 2023

Agradecimentos

Agradeço à minha mãe Cláudia, à minha avó Neuza e ao meu pai Pedro por todo o apoio, carinho e amor incondicional.

Aos meus amigos e colegas da faculdade, em especial Rafael e Tainá, que compartilharam comigo momentos de alegria e dificuldade ao longo do curso.

Aos meus amigos Bruno, Frederico, Guilherme e Gustavo por alegrarem meus dias mesmo quando estive longe de casa.

Ao Júnior pela amizade desde o Ensino Médio e por sempre me carregar no Dota.

Aos meus amigos Felipe, Leonardo e Stéfano pelos ensaios da banca e por me fazerem redescobrir o mundo da música.

À minha orientadora Márcia, não somente pelo apoio durante o TCC, como também pelos ensinamentos que tive em todas as oportunidades em que colaboramos.

Aos professores João e Viviane, por aceitarem participar da banca deste trabalho.

Aos demais professores da UFRGS, por contribuírem na minha formação profissional e acadêmica.

Por fim, agradeço a todos os familiares e amigos que me apoiaram de alguma forma nesta jornada.

Resumo

O aumento contínuo da geração e armazenamento de dados de texto, somado ao expressivo aumento do poder computacional, trouxe a necessidade de se ajustar modelos estatísticos mais complexos, provocando uma demanda substancial por métodos da área de Processamento de Linguagem Natural (NLP). As técnicas de NLP podem ser aplicadas em um vasto número de tarefas, como tradução de máquina, análise de sentimento e classificação de texto. Dentre estas tarefas, destaca-se a detecção automatizada de notícias falsas, também chamadas de *fake news*, que são cada vez mais presentes na sociedade com o advento de programas de computadores capazes de gerar e disseminar notícias falsas entre diversos meios de comunicação. Para combater a propagação de *fake news*, é possível utilizar o modelo de NLP BERT, que é capaz de representar o significado semântico e sintático de uma palavra a partir do seu contexto. Neste trabalho, é feita uma aplicação do modelo BERT para determinar se uma notícia específica de língua inglesa é falsa ou não, analisando a descrição textual de seu título. O banco de dados utilizado contém 72.134 notícias, já categorizadas entre verdadeiras e falsas, que foram extraídas de diferentes portais. Os resultados apontam para um bom desempenho do modelo ajustado, o que pode contribuir no combate à desinformação e auxiliar estudos futuros na área.

Palavras-Chave: Processamento de Linguagem Natural, *Deep Learning*, *Word Embeddings*, *Transformers*, BERT, *Fake News*.

Abstract

The continuous increase in the generation and storage of text data, added to the significant increase in computational power, brought the need to fit more complex statistical models, causing a substantial demand for methods in the field of Natural Language Processing (NLP). NLP techniques can be applied to a vast number of tasks, such as machine translation, sentiment analysis and text classification. Among these tasks, the automated detection of fake news stands out, which are increasingly present in society with the advent of computer programs capable of generating and disseminating fake news among various means of communication. To combat the spread of fake news, it is possible to use the BERT NLP model, which is capable of representing the semantic and syntactic meaning of a word from its context. In this work, an application of the BERT model is made to determine whether a certain news article in English is fake or not, analyzing the textual description of its title. The dataset used contains 72,134 news, already categorized between true and false, which were extracted from different portals. The results point to a good performance of the adjusted model, which can contribute to the fight against misinformation and help future studies in the area.

Keywords: Natural Language Processing, Deep Learning, Word Embeddings, Transformers, BERT, Fake News.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Iniciada na metade do século XX, a era da informação trouxe uma série de inovações à sociedade, principalmente na área da tecnologia. Com o aumento exponencial da geração e armazenamento de dados e o expressivo aumento do poder computacional, se tornou possível ajustar modelos estatísticos mais complexos. Segundo Breiman (2001), os modelos paramétricos não eram mais capazes de descrever a complexidade dos dados gerados neste contexto, mesmo sendo preferência da maioria dos estatísticos da época. Isso trouxe a necessidade de desenvolver e aplicar métodos capazes de extrair informações a partir de bases de dados mais complexas, de grande volume e diversidade. Dentre os diversos problemas que podem surgir com a análise de conjuntos de dados complexos, podemos destacar aqueles relacionados a variáveis do tipo texto. Tem-se como exemplos comentários em redes sociais, descrição de produtos, opiniões de clientes, textos de notícias e artigos científicos (Chowdhary e Chowdhary, 2020).

Um problema de grande relevância que envolve análise de texto é a propagação de notícias falsas, também chamadas de *fake news* (Vosoughi et al., 2018). Conter a propagação de notícias falsas é de suma importância para que se possa preservar a integridade da informação, mitigar a polarização social e proteger as eleições e demais processos democráticos. Este fenômeno não é novo, apesar de *fake news* ser um termo recente. No passado, informações falsas eram frequentemente disseminadas através de materiais impressos, como panfletos e jornais. Entretanto, com o advento de programas de computadores capazes de gerar e disseminar notícias falsas massivamente, somado à ampliação do alcance destas notícias em relação ao alcance de materiais impressos, se faz necessário a criação de um sistema automatizado de detecção de *fake news* (Monteiro et al., 2018).

Para auxiliar neste problema, é possível utilizar métodos capazes de analisar dados do tipo texto por meio de uma área dentro da inteligência artificial chamada de Processamento de Linguagem Natural (NLP) (Martin, 2009). Algumas aplicações de técnicas de NLP incluem o trabalho de Xu et al. (2019), que utilizam *word embeddings* para analisar textos de filmes, com o intuito de expor o reforço social dos estereótipos de gênero. Ou ainda, Swartz et al. (2017), que desenvolveram e implementaram uma ferramenta que utiliza NLP para classificar laudos radiológicos com alta acurácia. Também, temos Gao et al. (2019), que implementaram variações do modelo de NLP BERT para classificação de sentimento em nível de aspecto, melhorando o estado da arte de modelos desta área.

O ramo de NLP apresenta alguns desafios inerentes à natureza dos dados do tipo texto. Além de desenvolver modelos para resolver diferentes problemas deste

ramo, também é necessário encapsular o significado de palavras ou sentenças em uma representação computacional. Alguns modelos de inteligência artificial, como *Naive Bayes* (Rish et al., 2001) e *Boosting* (Schapire, 1990), usam um método de representação de texto chamado de *Bag of Words*. Este método, embora muito utilizado, não é capaz de explicar a relação semântica entre palavras. Já no *word embedding*, as palavras são representadas como um vetor numérico. Espera-se que o significado das palavras neste método possa ser representado em um espaço vetorial, em que palavras com sentido linguístico parecido possuam vetores próximos entre si.

Em 2013, Mikolov et al. (2013) publicaram o *word2vec*, que utiliza redes neurais artificiais para aprender semelhanças entre palavras com base na distribuição delas. Este artigo foi responsável, em grande parte, pela popularização de representações textuais de *word embeddings*. Os anos de 2014 e 2015 foram marcados pela crescente popularidade de modelos como as Redes Neurais Recorrentes (RNN) (Rumelhart et al., 1986) e o *Long Short-Term Memory* (LSTM) (Hochreiter e Schmidhuber, 1997). Estes modelos apresentaram uma arquitetura eficaz para modelar dados sequenciais, como dados do tipo texto. Todavia, o RNN e o LSTM não são boas escolhas para modelar longas sequências de texto, pois possuem problemas como longo tempo de processamento e dissipação do gradiente (Hochreiter et al., 2001). Em 2016, Bahdanau et al. (2014) introduziram o mecanismo de atenção, para resolver estes problemas do RNN e do LSTM. Este mecanismo informa ao modelo quais partes de uma sequência de texto devem receber mais importância em detrimento de outras. O artigo *"Attention is all you need"*, publicado por Vaswani et al. (2017), foi um ponto de inflexão na área de NLP, com a introdução do modelo de *transformer*. Este modelo utiliza somente o mecanismo de atenção para modelar a relação entre as palavras. Por conta disso, o *transformer* é capaz de processar os elementos da sequência em paralelo, aumentando a velocidade de treinamento do modelo em comparação à arquiteturas como o RNN e o LSTM. No ano seguinte à publicação de Vaswani et al. (2017), Devlin et al. (2018) desenvolveram o modelo BERT (Representações de *Encoders* Bidirecionais de *Transformers*), que faz uma adaptação bidirecional na arquitetura de *transformer*. A característica bidirecional do BERT permite que ele extraia informações de uma sequência textual em ambas as direções simultaneamente, o que resulta em um melhor desempenho do modelo. Assim como o *word2vec*, o BERT também representa o sentido de uma palavra a partir de outras ao seu redor, porém o BERT é sensível ao contexto. Isto significa que um *word embedding* diferente é calculado para uma mesma palavra caso ela esteja em contextos diferentes.

Dentre os trabalhos anteriores que utilizam métodos de inteligência artificial para classificação de *fake news*, é possível citar Lai et al. (2022), que aplicam os métodos de *Term Frequency-Inverse Document Frequency* (TF-IDF) e *K-Nearest Neighbors* (KNN) para classificar notícias paquistanesas entre verdadeira ou falsa. Também, Giordani et al. (2022) desenvolveram uma ferramenta capaz de auxiliar na classificação de notícias de língua portuguesa potencialmente falsas, por meio do método *word2vec*. Oshikawa et al. (2018) fazem uma análise de notícias falsas com múltiplos métodos de NLP, como RNN e LSTM, mas nenhum destes métodos inclui arquiteturas de redes neurais baseadas em *transformers*.

O objetivo deste trabalho é ajustar um modelo BERT para classificar notícias de língua inglesa entre verdadeira ou falsa, tomando como base a descrição textual

de seu título. Para isso, serão utilizados métodos e estratégias de aprendizado profundo, também chamado de *deep learning*, a fim de treinar, testar e avaliar o modelo ajustado. Para a leitura e manipulação do banco de dados, modelagem e avaliação dos resultados, foi utilizada a linguagem de programação **Python** (versão 3.11.3).

Este trabalho está estruturado da seguinte forma: no Capítulo 2 é feita uma breve apresentação das particularidades da análise de texto e, em seguida, são descritos alguns dos métodos de representação de texto mais populares na área de NLP, como o *one-hot encoding*, o *Bag of Words* e o *word embedding*. No Capítulo 3 são abordadas diferentes arquiteturas de *deep learning*, como os métodos de RNN, LSTM, *Embeddings from Language Models* (ELMo), *transformer* e, principalmente, BERT. No Capítulo 4 são apresentados os resultados da aplicação do modelo BERT em um conjunto de dados de notícias falsas denominado *WELFake* (Verma et al., 2021). No Capítulo 5 são descritas as conclusões da análise e considerações de trabalhos futuros.

2 Análise de texto

A Estatística é uma área do conhecimento que tem por objetivo obter, organizar e analisar dados para explicação de eventos, estudos e experimentos (Weber, 2006). Tal objetivo pode ser alcançado por meio do uso de métodos estatísticos, cuja escolha dependerá, dentre outros fatores, do tipo de dado coletado. Os dados podem ser definidos como uma sequência de símbolos quantificados ou quantificáveis e que podem ser armazenados em um computador e processados por ele (Setzer, 1999). Normalmente, os tipos de dados são classificados em quatro categorias principais: contínuo, discreto, ordinal e nominal (Bussab e Morettin, 2010). Números reais, números inteiros, escalas e atributos podem ser facilmente classificados em uma destas categorias. Já dados do tipo texto, por não serem estruturados, não possuem uma representação numérica ou categórica intrínseca, tornando a sua análise estatística mais complexa em comparação a outros tipos de dados.

Dentre os desafios de analisar dados de texto, é possível citar a dificuldade em lidar com o tamanho do vocabulário de uma língua. A cada ano, são criadas novas palavras enquanto outras caem em desuso, ao contrário dos números reais ou inteiros, que possuem um conjunto conhecido fixo de valores, embora infinitos. Além disso, o número de palavras do vocabulário de línguas aglutinantes, como turco e húngaro, é teoricamente infinito, já que não existe limite sintático para o número de sufixos derivacionais que uma palavra pode receber (Hakkani-Tür et al., 2002). Este processo chamado de morfologia derivacional, juntamente com transformações socioculturais, garante que o vocabulário linguístico estará sempre em constante mudança.

Outro desafio na análise de texto é que há somente abordagens qualitativas para aprender relações entre palavras, como sinonímia ou antonímia, mas não quantitativas. Neste sentido, seria interessante ter à disposição algo similar aos operadores matemáticos, em que pode-se quantificar a relação entre números com operações como subtração ou divisão.

Por fim, para que seja possível realizar uma análise estatística de texto em um computador, é necessário representar o significado de uma palavra numericamente. Essa tarefa é complexa, entretanto, pois não existe um consenso sobre a definição de significado de uma palavra entre linguistas e estudiosos da linguagem (Ferreira, 2010). Dentre as definições existentes, Kroeger (2019) disserta sobre a semântica denotacional, que é a ligação entre expressões linguísticas e o mundo, e a semântica cognitiva, que é a ligação entre expressões linguísticas e representações mentais. Outra definição possível é a semântica distributiva (Lenci et al., 2008), cuja ideia é representar o significado de uma palavra a partir do contexto em que ela aparece na

frase. Esta ideia é uma das mais populares em NLP por oferecer uma ótima maneira de aprender sobre o significado das palavras.

Com isso, é possível observar que muitos dos métodos estatísticos tradicionais podem não ser capazes de extrair informações relevantes de conjuntos de dados de texto, sendo necessário utilizar uma representação textual numérica que equivalha ao seu significado linguístico bem como de métodos que utilizem tal representação para executar as análises.

2.1 Métodos de representação textual

Uma forma natural de representar palavras computacionalmente é através de algo semelhante a um dicionário. Em Miller (1985), foi desenvolvido o *WordNet*, um banco de dados léxico que estabelece relações semânticas entre palavras, incluindo sinônimos, hipônimos e merônimos. Essas relações ajudam a capturar as nuances dos significados das palavras e facilitam a análise semântica em computadores a nível qualitativo. Entretanto, existem certos problemas em sistemas como o *WordNet*: a incapacidade de representar o significado de novas palavras sem trabalho manual contínuo, a definição subjetiva de palavras, a limitação em computar a similaridade entre elas, entre outros.

Neste capítulo, serão descritos alguns métodos de representação textual frequentemente utilizados em NLP. Eles transformam o texto em uma representação numérica, permitindo que algoritmos de NLP sejam capazes de quantificar relações semânticas entre palavras. Algumas definições importantes que serão utilizadas neste trabalho incluem:

- O *corpus* C é um conjunto de todos os M documentos de interesse da pesquisa, tal que $C = \{D_1, D_2, \ldots, D_j, \ldots, D_M\}$.
- Um documento D_j é uma sequência de N_j caracteres, tal que $D_j = \{w_1, w_2, \ldots, w_{N_j}\}$, podendo D_j ser uma sentença, um parágrafo, um livro ou qualquer outro segmento de conteúdo escrito.
- *Tokenização* é um processo no qual o documento D_j é dividido em palavras, subpalavras ou caracteres. Por exemplo, a sentença 'Eu amo NLP' pode ser *tokenizada* como ['Eu', 'amo', 'NLP'] ou ['E', 'u', ' ', 'a', 'm', 'o', ' ', 'N', 'L', 'P'] dependendo do método de *tokenização* escolhido. Cada elemento x_i resultante do processo é chamado de *token* e D_j pode ser representado desta forma como $D_j = \{x_1, x_2, \ldots, x_i, \ldots, x_{N_j}\}$.
- O vocabulário V é o conjunto de *tokens* distintos do *corpus* C, sendo N o tamanho do conjunto V.

2.1.1 One-hot encoding

O *one-hot encoding* é um método que possui um princípio similar à variáveis indicadoras, também chamadas de variáveis *dummy*. Ele é popular para lidar com tarefas de classificação multiclasse, devido à sua eficácia e simplicidade (Rodríguez et al., 2018). No *one-hot encoding*, cada *token* do vocabulário V é transformado em um vetor de tamanho N, além de receber um código identificador k, onde $k \in$

$\{1, 2, \ldots, N\}$. O vetor correspondente de cada *token* receberá o valor 0 em todas as posições, exceto na posição k, onde receberá o valor 1 (Vajjala et al., 2020). Desta forma, cada *token* distinto será representado por um vetor binário único, enquanto que *tokens* idênticos serão representados pelo mesmo vetor. Este método pode ser útil em tarefas de classificação, uma vez que transforma variáveis categóricas em vetores numéricos.

Como exemplo de aplicação do *one-hot encoding*, pode-se considerar um *corpus* C_1 contendo os seguintes documentos:

$$D_1 = \{\text{“Eu”}, \text{“leio”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}\}$$
$$D_2 = \{\text{“Eu”}, \text{“escrevo”}, \text{“um”}, \text{“artigo”}, \text{“de”}, \text{“estatística”}\}$$
$$D_3 = \{\text{“Eu”}, \text{“guardo”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}, \text{“e”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“estatística”}\}$$

Assim, o vocabulário de C_1 é dado por:

$$V = \{\text{“Eu”}, \text{“leio”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}, \text{“escrevo”}, \text{“artigo”}, \text{“estatística”}, \text{“guardo”}, \text{“e”}\}$$

Então, a representação *one-hot encoding* de D_1 e D_2 é dada por:

$$D_1 = \{ \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} \}; D_2 = \{ \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix} \}$$

Apesar de ser um método simples e de fácil implementação, fica claro o problema de armazenamento de dados ao se trabalhar com grandes conjuntos de texto, pois a dimensão dos vetores aumenta em função do número de palavras distintas do vocabulário (Vajjala et al., 2020). Além disso, como os vetores gerados pelo *one-hot encoding* são sempre ortogonais entre si, o método não é capaz de representar a relação entre palavras, tal como o *WordNet*.

2.1.2 Bag of words

O *Bag of Words* (BoW) é um método utilizado desde Harris (1954), em que o documento D_j é denotado pela quantidade de ocorrências de cada um de seus *tokens* em relação ao vocabulário V. Para contar o número de ocorrências de um elemento, o BoW realiza a correspondência exata de *tokens*, que pode ser considerada como um mapeamento rígido das palavras para o termo base (Zhao e Mao, 2017). Para cada documento, é criado um vetor numérico de tamanho N e cada posição do vetor

corresponde à frequência de um *token* do vocabulário naquele documento. Desta forma, é possível representar os *tokens* de modo simples e de fácil interpretação (Vajjala et al., 2020).

Utilizando o mesmo *corpus* C_1 do exemplo anterior, com os documentos e o vetor de vocabulário correspondentes:

$$
\begin{aligned}
D_1 &= \{\text{“Eu”}, \text{“leio”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}\} \\
D_2 &= \{\text{“Eu”}, \text{“escrevo”}, \text{“um”}, \text{“artigo”}, \text{“de”}, \text{“estatística”}\} \\
D_3 &= \{\text{“Eu”}, \text{“guardo”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}, \text{“e”}, \text{“um”}, \text{“livro”}, \\
&\quad \text{“de”}, \text{“estatística”}\} \\
V &= \{\text{“Eu”}, \text{“leio”}, \text{“um”}, \text{“livro”}, \text{“de”}, \text{“matemática”}, \text{“escrevo”}, \text{“artigo”}, \\
&\quad \text{“estatística”}, \text{“guardo”}, \text{“e”}\}
\end{aligned}
$$

Então a representação BoW deste conjunto de texto é dada por:

$$
\begin{aligned}
D_1 &= \{1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0\} \\
D_2 &= \{1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0\} \\
D_3 &= \{1, 0, 2, 2, 2, 1, 0, 0, 1, 1, 1\}
\end{aligned}
$$

Assim como o *one-hot encoding*, a representação do BoW sofre por ser muito esparsa, ter alta dimensionalidade e ser incapaz de capturar significados semânticos de alto nível por trás dos dados de texto (Zhao e Mao, 2017).

2.1.3 Word embedding

Com o objetivo de abordar as limitações de modelos tradicionais de representação de texto, Bengio et al. (2000) propuseram um método de linguagem probabilística utilizando redes neurais chamado *word embedding*, ou vetor de palavras, que captura relacionamentos entre palavras em uma representação numérica contínua. Neste método, cada palavra ou *token* é mapeado em um vetor de tamanho I, onde I é um hiper-parâmetro, de forma que as palavras do *corpus* estarão dispostas em um espaço vetorial $\mathbb{R}^I$.

Assumindo que a representação vetorial das palavras é fiel ao seu significado linguístico, os *word embeddings* devem codificar a informação semântica e sintática das palavras, onde a informação semântica se correlaciona principalmente com o significado das palavras, enquanto que a informação sintática se refere aos seus papéis estruturais (Li e Yang, 2017). Uma suposição básica do método é que palavras em contextos semelhantes devem ter significados semelhantes (Harris, 1954), demonstrando que os *word embeddings* tem a capacidade de representar numericamente o significado de palavras pela definição da semântica distributiva (Lenci et al., 2008).

Existem diversos modelos que geram *word embeddings* a partir de um *corpus*. Um dos mais famoso entre eles é o *word2vec*, proposto por Mikolov et al. (2013), que pode ser treinado para prever uma palavra-alvo a partir de palavras de contexto, utilizando a arquitetura *Skip-gram*, ou para prever palavras de contexto a partir de uma palavra-alvo, utilizando a arquitetura *Continuous Bag of Words* (CBOW). É possível dizer que Mikolov et al. (2013) teve um papel fundamental para a popularização da representação de palavras por *word embeddings*. A maioria dos métodos

contemporâneos de NLP como o ELMo (Peters et al., 2018), o *Generative Pre-trained Transformer* (GPT) (Radford et al., 2018) e o BERT (Devlin et al., 2018) utilizam tais representações em suas metodologias.

Para exemplificar visualmente os vetores de palavras, considere a figura abaixo, que foi elaborada utilizado um modelo *Word2Vec* pré-treinado em português de dimensão 50 desenvolvido por Hartmann et al. (2017). Neste modelo, cada palavra é representada por um vetor numérico de tamanho 50 e, para que seja possível visualizá-las em um gráfico bidimensional, foi reduzida a dimensão dos vetores de 50 para 2 utilizando Análise de Componentes Principais.

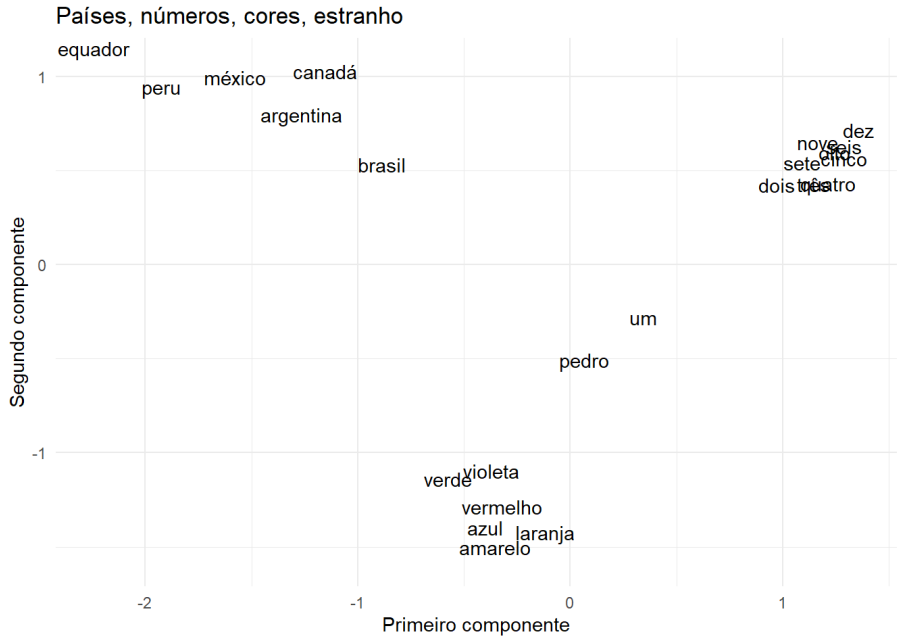

Figura 2.1: Visualização gráfica de vetores de palavras

As palavras foram escolhidas com a intenção de agrupá-las em 4 categorias: países, como "Brasil", "Canadá" e "México"; números, como "nove", "sete" e "quatro"; cores, como "verde", "azul" e "amarelo"; e nome próprio, constituído somente por "pedro". É possível notar que as palavras das categorias de países, cores e números ficaram próximas entre si no espaço vetorial, enquanto que os agrupamentos das categorias ficaram distantes uns dos outros. O nome próprio "pedro", por ser a única palavra de seu grupo, ficou equidistante de todas as categorias. Isso mostra a capacidade dos *word embeddings* de capturar a nuances entre as palavras e alocá-las em *clusters* adequados no espaço vetorial.

Uma exceção notável do comportamento esperado dos vetores acima é a palavra "um", que ficou afastada do grupo de números. Isso aconteceu porque, além de ser um numeral, "um" também pode ser usado como um artigo indefinido na língua portuguesa, diferentemente das outras palavras da categoria de números.

3 Deep learning

Um neurônio biológico recebe múltiplos sinais e os processa a fim de enviar, ou não, um único fluxo de energia para outros neurônios. A conversão de um padrão complexo de entradas em uma decisão simples, ativar ou não ativar, sugeriu aos primeiros pesquisadores da área que cada neurônio biológico executa uma função cognitiva elementar: reduzir a complexidade ao categorizar seus padrões de entrada (Kriegeskorte e Golan, 2019). Inspirados por esta ideia, as redes neurais artificiais são um ramo da inteligência artificial que é capaz de aproximar funções e dinâmicas ao aprender com exemplos (Kriegeskorte e Golan, 2019), (Gardner e Dorling, 1998).

Os modelos de redes neurais são compostos por unidades chamadas de neurônios artificiais, que combinam valores de entrada (*inputs*) e produzem pelo menos uma saída (*output*). A forma como os neurônios são organizados depende da arquitetura utilizada, que, por sua vez, depende do objetivo de pesquisa. Uma das arquiteturas mais simples é o *Multi-Layer Perceptron* de uma camada oculta, representado na Figura (3.1).

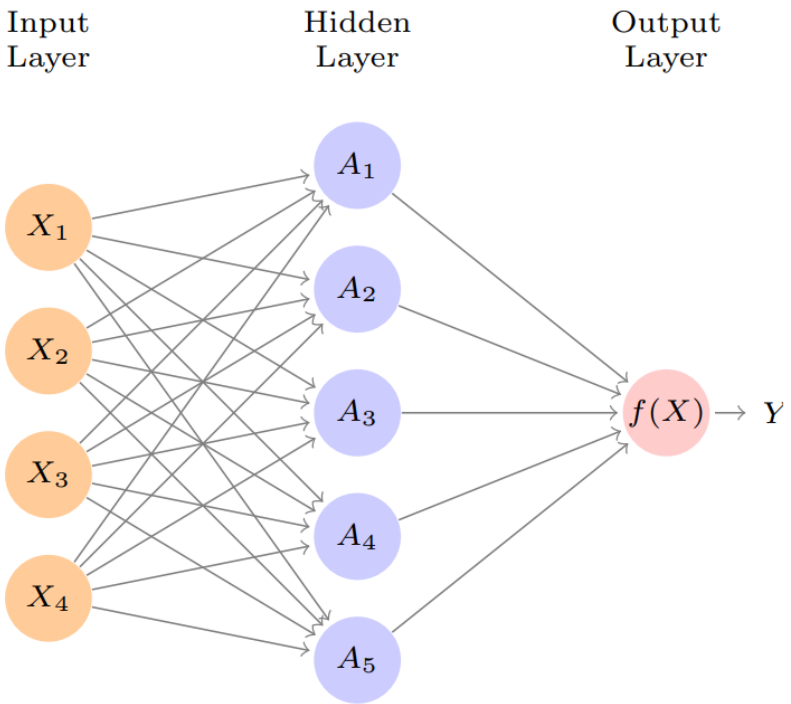

Figura 3.1: Exemplo de uma rede neural com uma camada oculta
Fonte: (James et al., 2013)

Esta arquitetura organiza os neurônios em três camadas, denominadas camada de entrada, oculta e de saída. Todos os neurônios de uma camada se conectam com todos os neurônios da camada seguinte por meio de ligações e, por isso, esta arquitetura é dita ser totalmente conectada. Cada ligação possui um peso, que reflete a importância da informação transmitida de um neurônio anterior para um posterior. Se um modelo de redes neurais utilizar esta arquitetura, é possível dizer que os únicos parâmetros que o modelo precisa estimar são os pesos das ligações. Se os pesos forem estimados corretamente e os neurônios forem ativados por uma função não linear, o modelo pode ser considerado um aproximador universal para qualquer função contínua (Camuñas-Mesa et al., 2019).

Para ajustar os pesos da rede, inicia-se com a camada de entrada, que contém as p covariáveis numéricas $X = (X_1, X_2, \ldots, X_p)$ que serão utilizadas para estimar a resposta Y. Cada covariável é representada por um neurônio na camada de entrada, que se conecta com todos os K neurônios $A = (A_1.A_2, \ldots, A_K)$ da camada oculta L seguinte. Assim, os valores de $\mathbf{A}$ são calculados como:

$$A_k = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j). \tag{3.1}$$

Na Equação 3.1, w_{kj} refere-se ao peso da ligação entre os neurônios X_j e A_k. Para cada neurônio A_k, é feita a multiplicação de todos os neurônios de entrada X_j com seus respectivos pesos w_{kj}. Esses valores são somados e, em seguida, também é somado um valor de viés, ou intercepto, w_{k0}. Este resultado é passado para uma função não linear $g(.)$ chamada de função de ativação, que é responsável por introduzir não linearidade ao modelo (Goodfellow et al., 2016). Se $g(.)$ fosse linear, a rede neural inteira permaneceria como uma função linear de $\mathbf{X}$ e o modelo não seria capaz de capturar relações complexas entre os dados. Dentre as funções de ativação mais utilizadas, temos a função sigmoide, a *Rectified Linear Unit* (ReLU) e a tangente hiperbólica (Patterson e Gibson, 2017), respectivamente descritas nas Equações em 3.2. É importante notar que a inserção do viés w_{k0} na Equação 3.1 permite o deslocamento da função de ativação horizontalmente (Kriegeskorte e Golan, 2019).

$$g(x) = \frac{1}{1 + e^{-x}}; \qquad g(x) = max(0, x); \qquad g(x) = tanh(x). \tag{3.2}$$

Uma vez estimados os valores de $\mathbf{A}$, é possível estimar os valores dos neurônios da camada de saída. Esta camada costuma ser construída com base no objetivo de pesquisa, podendo conter um único neurônio, caso o interesse seja predizer uma variável quantitativa ou qualitativa com um número fixo de classes, ou múltiplos neurônios, caso o interesse seja classificar as observações em um conjunto de categorias, em que cada categoria seria representada por um dos neurônios da camada de saída. No exemplo da Figura (3.1), podemos obter o neurônio $f(\mathbf{X})$ como:

$$f(X) = g(\beta_0 + \sum_{k=1}^{K} \beta_k A_k). \tag{3.3}$$

Similar ao primeiro passo, β_0 representa o viés, $g(.)$ a função de ativação e β_k o peso da ligação entre o neurônio A_k e $f(\mathbf{X})$. É possível reescrever a Equação 3.3 utilizando notação matricial, em que $\mathbf{X} \in \mathbb{R}^{p \times 1}$ é o vetor dos neurônios de entrada, $\mathbf{W}_1 \in \mathbb{R}^{K \times p}$ é a matriz de pesos entre a primeira e a segunda camada do modelo,

$\mathbf{b}_1 \in \mathbb{R}^{K\times 1}$ é o vetor de vieses associado à $\mathbf{W}_1$, $\mathbf{W}_2 \in \mathbb{R}^{1\times K}$ é a matriz de pesos entre a segunda e a terceira camada do modelo, $\mathbf{b}_2 \in \mathbb{R}^{1\times 1}$ é o vetor de vieses associado à $\mathbf{W}_2$. Neste exemplo, $\mathbf{W}_2$ e $\mathbf{b}_2$ são um vetor e um escalar, respectivamente, pois só há um neurônio na camada de saída. Fazendo esta reescrita e unindo os termos das Equações 3.1 e 3.3, temos:

$$f(\mathbf{X}) = g(\mathbf{W}_2 g(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2). \tag{3.4}$$

A arquitetura *Multi-Layer Perceptron* de uma camada oculta pode ser considerada do tipo *Feed-Forward*, uma vez que a informação flui somente no sentido da camada de *inputs* para a camada de *output*. Esta transmissão de informação descrita na Equação 3.4 também é chamada de *forward pass*. Quando um modelo de rede neural possui mais de uma camada oculta, pode-se considerar que este é um modelo de *deep learning* (Kriegeskorte e Golan, 2019).

Deep learning é uma subárea de *machine learning* e Inteligência Artificial (Patterson e Gibson, 2017) que tem adquirido muita popularidade nas últimas décadas entre pesquisadores de NLP. Em grande parte, isso ocorre devido aos métodos de *deep learning* terem melhorado drasticamente o estado da arte em campos como reconhecimento de fala, reconhecimento visual e detecção de objetos (LeCun et al., 2015). Um exemplo clássico de *deep learning* é o modelo *Multi-Layer Perceptron* de duas camadas ocultas (Goodfellow et al., 2016) ilustrado na Figura (3.2).

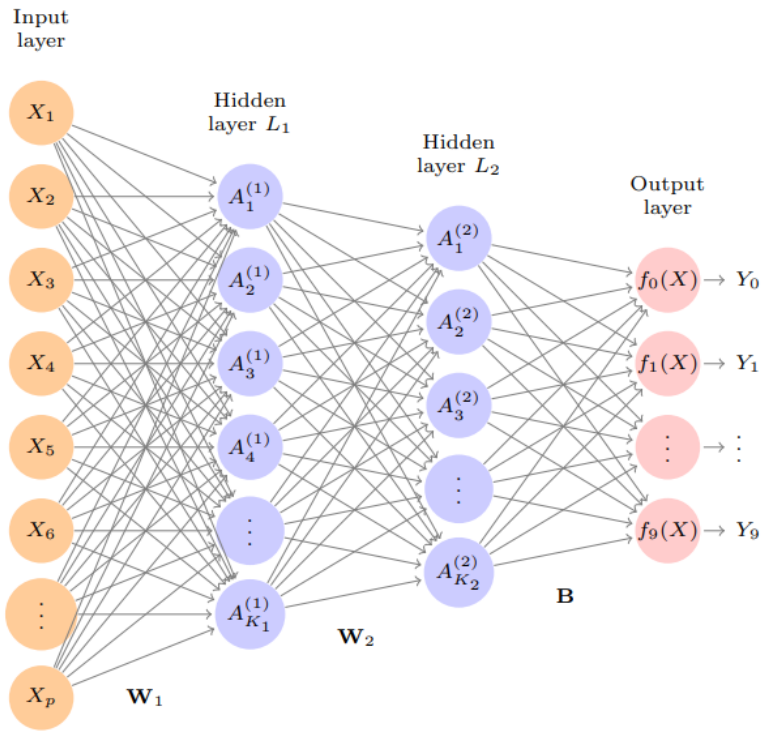

Figura 3.2: Exemplo de uma rede neural com duas camadas ocultas.
Fonte: (James et al., 2013)

Para que um modelo de *deep learning*, como o da Figura (3.2), seja capaz de obter um bom desempenho em uma determinada tarefa T, é necessário coletar um conjunto de exemplos e armazená-los em um banco de dados. Cada exemplo é

uma observação das p covariáveis que serão utilizadas na camada de entrada da rede neural. É esperado que o modelo aprenda com as observações do banco de dados e ajuste os pesos de cada neurônio para performar a tarefa T, o que ocorre em uma fase chamada de treinamento do modelo. Existem diversos métodos para treinar um modelo de *deep learning* a depender da escolha da tarefa T, da medida de desempenho e do tipo de experiência que o modelo pode adquirir durante o treinamento.

Em tarefas de classificação, o modelo precisa especificar a qual categoria a observação de covariáveis $\mathbf{X}$ pertence (Goodfellow et al., 2016). Nestas tarefas, cada neurônio da camada de saída representa uma das categorias possíveis e é desejável que o maior valor do vetor retornado por $f(\mathbf{X})$ seja o do neurônio correspondente à verdadeira categoria da observação $\mathbf{X}$. Em tarefas de aprendizagem supervisionada, o modelo adquire experiência por meio de exemplos do banco de dados, que possuem não somente as observações das p covariáveis, como também de uma variável alvo ou *target* que se deseja estimar (Goodfellow et al., 2016).

Desta forma, tarefas de classificação supervisionada são aquelas em que é possível avaliar a estimativa $f(\mathbf{X})$ com base na variável alvo $\mathbf{Y}$ correspondente à $\mathbf{X}$, que pode ser interpretada como a verdadeira categoria de $\mathbf{X}$. Utilizando $\mathbf{Y}$ e $f(\mathbf{X})$, é possível construir uma função custo $c(\mathbf{Y}, f(\mathbf{X}))$ como uma medida de desempenho que o modelo obteve na classificação da observação $\mathbf{X}$ em que, quanto menor for $c(\mathbf{Y}, f(\mathbf{X}))$, menor será o erro de classificação. Uma abordagem comum para treinar um modelo de *deep learning* é utilizar métodos que minimizem a função custo, ao indicar a mudança necessária a ser feita no valor dos parâmetros da rede.

Dentre os métodos para minimizar o custo total da rede, que pode ser definido como, por exemplo, a média de $c(\mathbf{Y}, f(\mathbf{X}))$ entre todas as observações do banco de dados, tem-se o *back-propagation*. Este algoritmo computa o gradiente da função custo em relação aos pesos da rede, começando pelos pesos que ligam a última camada oculta e a camada de saída, e os atualiza seguindo a ideia descrita pela Equação abaixo.

$$\theta^{'} = \theta - \gamma \nabla_{\theta} c(\theta). \tag{3.5}$$

Utilizando a Figura (3.2) como exemplo, o θ da Equação 3.5 representa um parâmetro qualquer da matriz de pesos $\mathbf{B}$ e $-\nabla_{\theta} c(\theta)$ é o gradiente computado pelo *back-propagation*. Logo, $\nabla_{\theta} c(\theta)$ indica o valor ao qual θ precisa ser modificado para que $c(\theta)$ ande na direção de seu mínimo. Desta forma, $\theta^{'}$ é o valor atualizado do parâmetro θ. A constante γ, chamada de taxa de aprendizado, determina a magnitude do passo no qual os parâmetros do modelo são atualizados pelo gradiente. Quanto maior a taxa de aprendizado, mais rápida é a convergência dos parâmetros e maior é o risco do valor ótimo ser ultrapassado. Em contrapartida, quanto menor a taxa de aprendizado, mais lenta é a convergência dos parâmetros e menor é o risco do valor ótimo ser ultrapassado.

Uma vez concluída a atualização dos pesos $\mathbf{B}$, este processo é iterado para atualizar os pesos $\mathbf{W}_2$ e, em seguida, $\mathbf{W}_1$, utilizando o gradiente calculado sobre uma camada para atualizar os pesos que ligam esta camada com a anterior. Esta atualização é o que motiva o nome do método *back-propagation*, pois os pesos da camada de saída são atualizados em direção à camada de entrada. Neste trabalho, o método *back-propagation* será utilizado para se referir a todo o algoritmo de aprendizagem.

Quando todas as observações do banco de dados são utilizadas pelo processo de

aprendizagem durante o treinamento, é dito que foi completada uma época (Nielsen, 2015). A quantidade de épocas utilizadas no treinamento precisa ser grande o suficiente para aproximar o custo total da rede a zero.

Uma vez que treinar uma rede neural com todo o conjunto de dados em cada época pode ser caro computacionalmente, é possível utilizar um método alternativo que envolve calcular o gradiente somente em subconjuntos do banco de dados chamados de mini-lotes. O número de observações em cada mini-lote, também chamado de tamanho do mini-lote, é uma constante que precisa ser definida *a priori*. Quanto maior o tamanho do mini-lote, mais precisa será a estimativa do gradiente e maior será o poder computacional necessário. Então, o gradiente é calculado n_{ml} vezes em cada época com base, por exemplo, no custo médio de cada mini-lote, utilizando o método *back-propagation*, em que n_{ml} se refere ao número de mini-lotes. Esta técnica é conhecida como gradiente descendente por mini-lote (Goodfellow et al., 2016). A convergência por este método possui maior variabilidade do que a convergência pelo gradiente descendente e nem sempre vai na direção do ponto ótimo, mas ela é capaz de convergir ao mínimo da função custo mais rápido, o que propicia um aumento da velocidade computacional do gradiente. Além disso, é importante notar que o desempenho de um modelo de *deep learning* também depende do número de observações do banco de dados. Quanto maior o número de pesos a serem ajustados, maior deve ser o número de observações para obter um ajuste adequado.

Uma vez encerrada a fase de treinamento, espera-se que o modelo obtenha um bom desempenho na tarefa T ao lidar com novas observações, não somente naquelas em que o modelo foi treinado. Ou seja, é desejável que o modelo atinja a generalização. Uma maneira de verificar a generalização do modelo é computando certas medidas de avaliação do modelo em um banco de dados de teste, cujas observações são coletadas separadamente do banco utilizado no treinamento. Se forem feitas as suposições de que as observações deste conjuntos de dados são independentes e identicamente distribuídas, é possível construir ambos os bancos de treinamento e de teste a partir do mesmo banco de dados, coletado inicialmente para o treinamento da rede neural. Uma proporção das observações do banco original é amostrada para ser alocada ao banco de teste, enquanto que o restante das observações é destinada ao banco de treinamento. Normalmente, o banco de treinamento contém 70% das observações do banco original, enquanto que o banco de teste contém os 30% restantes. Assim, ao invés de treinar o modelo com todas as observações, é utilizado somente os dados do banco de treinamento, sendo o banco de teste reservado para avaliar a generalização do modelo. Caso o modelo obtenha um desempenho insuficiente na fase de treinamento, é dito estar ocorrendo o *underfitting*, em que o modelo não foi capaz de aprender com as observações do banco de treinamento. Caso o modelo obtenha um desempenho adequado na fase de treinamento, mas não na fase de teste, é dito estar ocorrendo o *overfitting*, em que o modelo não foi capaz de generalizar a informação aprendida no treinamento. Portanto, é desejável treinar um modelo de *deep learning* para que o desempenho no banco de treinamento e de teste seja adequado.

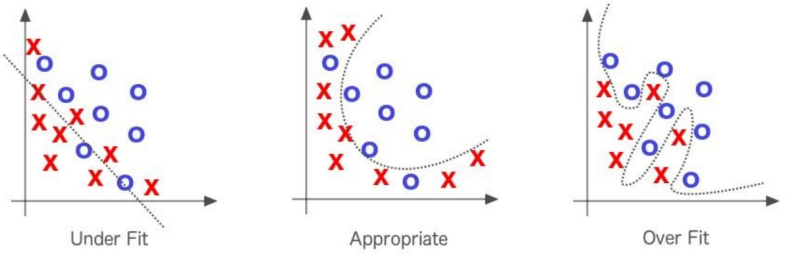

Figura 3.3: *Underfitting* e *overfitting* em *machine learning*
Fonte: (Patterson e Gibson, 2017)

Durante a fase de treinamento, são utilizadas algumas configurações que controlam o comportamento do algoritmo de aprendizagem. Tais configurações, chamadas de hiperparâmetros, não são ajustadas no decorrer do treinamento e precisam ser escolhidas antes do início deste processo Goodfellow et al. (2016). Dentre os exemplos de hiperparâmetros, temos o número de camadas ocultas em uma arquitetura *Multi-Layer Perceptron*, o número de neurônios em cada camada oculta, o tamanho do mini-lote, a taxa de aprendizado e o número de épocas. Ainda assumindo que as observações do banco de dados são independentes e identicamente distribuídas, é possível amostrar uma proporção das observações do banco de treinamento para ser alocada a um banco de validação. Este conjunto é usado para estimar o desempenho do modelo durante ou após o treinamento, permitindo que os hiperparâmetros sejam atualizados adequadamente. Desta forma, é possível escolher o melhor conjunto de hiperparâmetros baseando-se no desempenho do modelo treinado com este conjunto no banco de validação. Finalizada a otimização dos hiperparâmetros, o modelo treinado pode ser avaliado usando o banco de teste. É necesário empregar uma terceira partição do banco de dados, na forma de um conjunto de validação, pois os dados de teste não podem utilizados para fazer ajustes no modelo, incluindo seus hiperparâmetros Goodfellow et al. (2016). As proporções de observações do banco de dados que são alocadas para os bancos de treinamento, validação e teste normalmente são 70%, 15% e 15%, respectivamente. Estes valores, apesar de frequentemente utilizados, são apenas uma sugestão e podem mudar de acordo com o problema de pesquisa.

3.1 Modelos sequenciais

As arquiteturas de redes neurais são projetadas para solucionar categorias específicas de problemas e lidar com tipos específicos de dados. A arquitetura de *Multi-Layer Perceptron*, ainda que muito popular, pode não ser adequada para resolver certas tarefas. Por exemplo, dados sequenciais como faixas de áudio, notas musicais, palavras em um texto e séries temporais possuem uma estrutura de dependência entre os elementos da sequência que precisa ser levada em consideração pelo modelo. Tais tarefas podem ser melhor tratadas por uma classe de modelos de *deep learning* chamada de modelos sequenciais (Sutskever et al., 2014). Nesta classe, são utilizadas arquiteturas de redes neurais em que o processamento de um neurônio da camada de entrada é influenciado não apenas pelo próprio neurônio, mas também pelos outros neurônios da mesma camada. A forma como este processamento é feito depende do modelo escolhido. Uma vez que o foco deste trabalho é analisar dados de texto, os modelos sequenciais serão abordados no contexto de NLP e o termo

sequência estará se referindo especificamente à sequências textuais, ou seja, *tokens* em um documento.

Dentre os modelos sequenciais de *deep learning*, podemos citar a arquitetura de Redes Neurais Recorrentes (RNN) Rumelhart et al. (1986). Neste modelo, cada *token* da sequência, representado por um neurônio X_i da camada de entrada, possui uma ligação com um neurônio A_i da camada oculta seguinte. A representação textual dos *tokens* $\mathbf{X}$ utilizada pelo RNN é o *word embedding* e o peso da ligação com cada A_i é uma matriz de pesos $\mathbf{W}$. Desta forma, diferentemente da arquitetura *Multi-Layer Perceptron*, cada neurônio da camada oculta é um vetor e não mais um escalar. Esta representação dos *tokens* por *word embeddings* é comum a todos os modelos de NLP que serão vistos daqui em diante.

O RNN utiliza ligações entre os neurônios da camada oculta, denominadas de estados ocultos, que permitem à informação aprendida por um neurônio A_i ser transferida para um neurônio A_{i+1} (Sherstinsky, 2018), processo este visualmente exemplificado pela Figura (3.4). É importante notar que são usadas as mesmas matrizes de pesos $\mathbf{W}$, $\mathbf{U}$ e $\mathbf{B}$ conforme cada elemento da sequência é processado, o que é uma forma de compartilhamento de pesos utilizado por modelos de RNN. Esta arquitetura, porém, possui algumas desvantagens conhecidas. Dentre elas, é possível citar o problema de memória de curto prazo, que torna a contribuição da informação de *tokens* iniciais irrelevante à *tokens* mais distantes em sequências longas. Isso acontece devido aos problemas de gradiente explosivo e dissipação do gradiente, que ocorre na fase de treinamento do modelo (Hochreiter et al., 2001). Além disso, o tempo de treinamento do RNN é considerado lento, pois os *tokens* são processados sequencialmente, de modo que um *token* X_i somente é processado após todos os *tokens* anteriores serem processados.

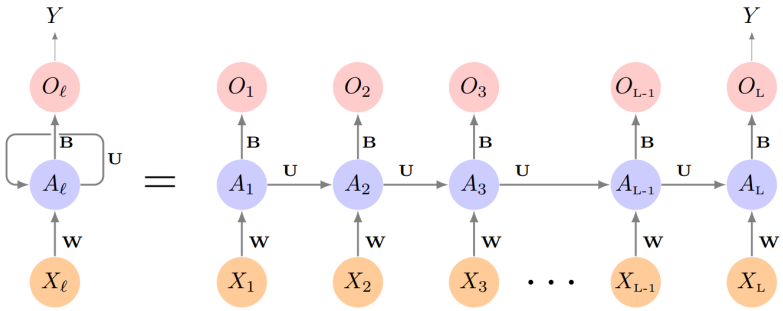

Figura 3.4: Exemplo de um RNN simples
Fonte: (James et al., 2013)

O modelo *Long Short-Term Memory* (LSTM) foi proposto por Hochreiter e Schmidhuber (1997) para corrigir o problema de memória de curto prazo do RNN. No LSTM, cada neurônio A é substituído por um conjunto de mecanismos chamado de célula. As células são compostas por uma série de portões que regulam o fluxo de informação entre os *tokens* da sequência. Assim, o modelo aprende quais informações da sequência são mais relevantes para manter na camada oculta e quais podem ser "esquecidas". Isso garante que, mesmo para sequências longas, a informação aprendida nos *tokens* iniciais não será perdida. Apesar desta inovação, o tempo de treinamento do LSTM é considerado lento pois, além dos *tokens* serem processados sequencialmente como ocorre no RNN, esta arquitetura contém mais parâmetros a serem ajustados devido a introdução do mecanismo de portões e células.

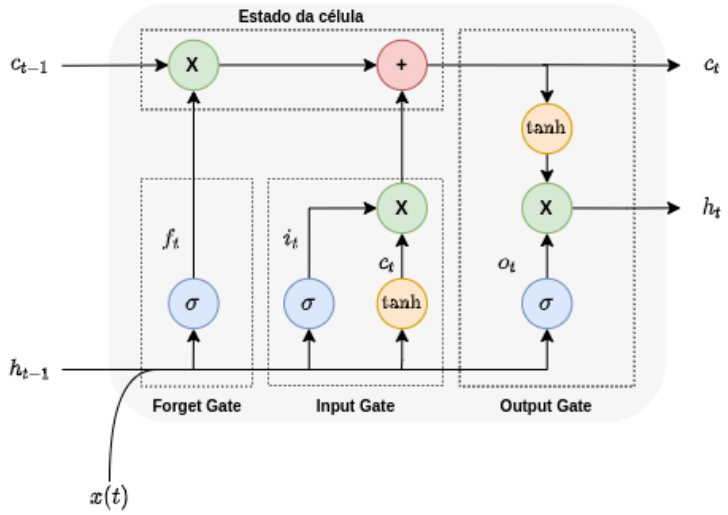

Figura 3.5: Célula básica de uma rede neural LSTM
Fonte: (Barbosa et al., 2021)

No início de 2018, foi proposta uma representação profunda contextualizada de palavras chamada de *Embeddings from Language Models* (ELMo) por Peters et al. (2018). Com o objetivo de treinar um modelo que capture o significado contextual dos *tokens*, o ELMo gera *word embeddings* como funções da sequência de entrada. Desta forma, um *token* X_i pode ser representado por diferentes vetores de palavras dependendo do conjunto de *tokens* $\mathbf{X}$ em que ele está inserido. Diferente de outros métodos como *word2vec* que atribuem um *word embedding* estático a cada palavra, este tipo de vetor pode ser chamado de *word embedding* contextual.

Modelos como RNN e LSTM transferem informação entre os *tokens* sequencialmente, do *token* inicial ao final ou vice-versa. O ELMo, por sua vez, utiliza uma arquitetura de rede neural bidirecional, que considera ambas as direções, representada na Figura (3.6). Nela, os *tokens* da sequência são calculados sobre modelos de linguagem bidirecional profundos (biLM) de duas camadas com convoluções de caracteres, para que cada *token* tenha a oportunidade de receber informação tanto de *tokens* anteriores da sequência, como de posteriores (Peters et al., 2018). Os vetores de palavras resultantes do treinamento de cada direção são concatenados e espera-se que o vetor final se aproxime mais do significado semântico do *token* do que *word embeddings* estáticos. A arquitetura utilizada pelo ELMo, entretanto, é subótima para um treinamento bidirecional, pois os *word embeddings* são gerados com concatenações de dois resultados independentes: da esquerda para a direita e da direita para a esquerda. Mesmo assim, o ELMo foi capaz de alcançar resultados em estado da arte para múltiplas tarefas de NLP, como resposta a perguntas, implicação textual e análise de sentimento, demonstrando o potencial de modelos capazes de gerar *word embeddings* contextuais.

3.2 Transformer

Originalmente desenvolvido para solucionar tarefas de NLP, o *transformer* é uma arquitetura de rede neural para dados sequenciais proposta por Vaswani et al. (2017) que melhorou o estado da arte em tarefas como tradução de máquina (Vaswani et al.,

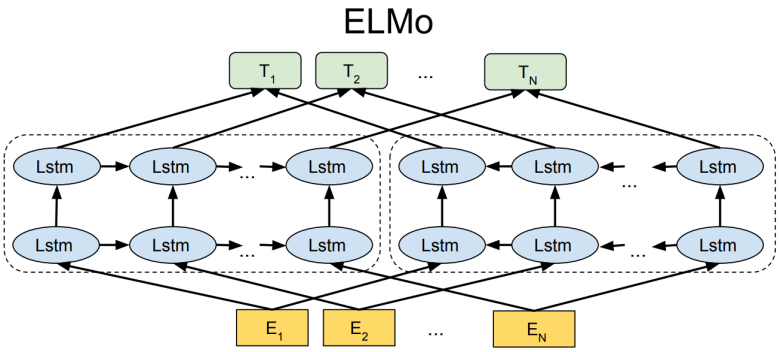

Figura 3.6: Arquitetura do modelo ELMo
Fonte: (Devlin et al., 2018)

2017), geração de documento (Liu et al., 2018) e análise sintática (Kitaev e Klein, 2018). Esta arquitetura utiliza um tipo de mecanismo de atenção chamado de autoatenção, que também foi proposto por Vaswani et al. (2017). Este mecanismo permite que os elementos das sequências de entrada sejam processados em paralelo, o que aumenta a velocidade de treinamento do modelo em relação à outras arquiteturas como o RNN ou o LSTM, em que os elementos da sequência são processados um após o outro. A arquitetura de *transformer* revolucionou a área de NLP (Zhao et al., 2021), sendo o primeiro modelo de conversão de sequência baseado inteiramente no mecanismo de autoatenção, sem fazer uso de recorrência ou convolução Niu et al. (2021). Para detalhar o algoritmo do modelo de *transformer*, considere a tarefa de tradução de máquina em que se deseja traduzir o documento da língua portuguesa para seu documento equivalente na língua inglesa:

$$\begin{aligned} D_{input1} &= \text{“o menino comeu o bolo porque ele estava morrendo de fome”}, \\ D_{target1} &= \text{“}\textit{the boy ate the cake because he was starving}\text{”}. \end{aligned}$$

O mecanismo de atenção foi introduzido por Bahdanau et al. (2014) e é inspirado nos sistemas biológicos dos seres humanos que, ao processar uma grande quantidade de informação, tendem a se focar seletivamente em determinadas partes e, quando necessário, ignorar outras (Niu et al., 2021). Este mecanismo calcula escores e pesos de atenção para cada elemento da sequência de entrada, refletindo o nível de importância que deve ser atribuída a estes elementos. Os escores são normalmente calculados com base no relacionamento entre o elemento de entrada e os elementos da sequência que está sendo atendida. Por exemplo, espera-se que a palavra “*starving*” do documento $D_{target1}$ possua um alto escore de atenção com as palavras “morrendo”, “de” e “fome” do documento D_{input1}, uma vez que a expressão “morrendo de fome” pode ser traduzida para o inglês como “*starving*”. Já o escore de atenção entre a palavra “porque” e a segunda palavra “*the*” é esperado ser baixo, uma vez que a relação semântica entre essas palavras é fraca.

O mecanismo de autoatenção é um tipo específico de atenção. Nele, os escores e pesos de atenção são calculados entre os elementos da mesma sequência, permitindo que o modelo capture relações semânticas complexas dentro da sequência. Por exemplo, espera-se que a palavra “menino” do documento D_{input1} possua um alto escore de atenção com a palavra “ele”, dado que ambas as palavras estão se referindo

ao mesmo sujeito. É importante notar que a autoatenção é calculada independentemente em cada *token* da sequência e não sequencialmente, permitindo que cada *token* seja processado em paralelo.

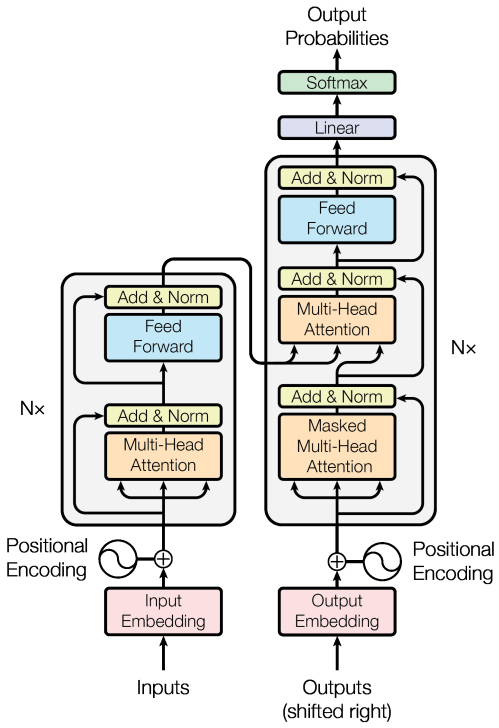

Figura 3.7: Arquitetura do modelo Transformer
Fonte: (Vaswani et al., 2017)

A arquitetura de *transformer* pode ser dividida em duas partes: o *encoder* e o *decoder*. O *encoder* recebe uma sequência de *word embeddings* de entrada e é responsável por produzir uma sequência de estados ocultos que encapsulam as informações sobre o conteúdo e o contexto de cada *token* da sequência. O *decoder*, por sua vez, recebe as representações geradas pelo *encoder* e gera uma sequência de *tokens* preditos, um elemento por vez. A cada passo, o modelo é auto-regressivo, consumindo os *tokens* gerados anteriormente como entrada adicional ao gerar o próximo *token* (Vaswani et al., 2017). O processo é repetido até que o *token* especial que indica o final da sequência, "EOF", seja predito. Em ambas as partes da arquitetura, existe uma etapa anterior de pré-processamento em que são aplicadas transformações nas sequências de *input* e *output* para que possam ser processadas pelo *encoder* e *decoder*, respectivamente.

3.2.1 Pré-processamento

A primeira etapa da fase de pré-processamento consiste em *tokenizar* ambos os documentos utilizando algum método de *tokenização*, como considerar um *token* sendo uma palavra. Na primeira posição do $D_{target1}$, é inserido um *token* especial que indica o inicio de sequência, "$<START>$", o que desloca a posição dos outros *tokens* uma unidade para a direita.

Considerando novamente o exemplo de tradução de máquina proposto anteriormente, D_{input1} é uma sequência que contém $N_{input1} = 11$ *tokens* e $D_{target1}$ é uma

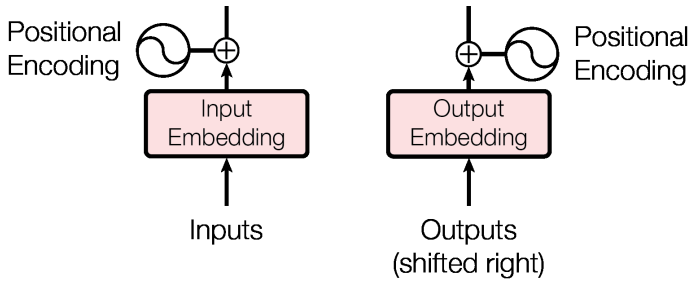

Figura 3.8: Etapa relativa ao pré-processamento do modelo Transformer
Fonte: (Vaswani et al., 2017)

sequência que contém $N_{target1} = 10$ *tokens*. Então, cada *token* dos documentos é convertido em sua representação de *word embedding* com tamanho d_{model} de, por exemplo, 512. Por serem conjuntos de vetores, é possível definir D_{input1} como uma matriz $N_{input1} \times d_{model}$, denominada matriz de entrada, e $D_{target1}$ como uma matriz $N_{target1} \times d_{model}$, denominada matriz de saída, em que cada linha das matrizes é um *token* e cada coluna é um elemento do *word embedding* desses *tokens*. O *transformer* pode utilizar os elementos da matriz de saída para ajustar seus parâmetros durante o processo de treinamento, mas não nos processos de validação e teste. Para seguir com o exemplo, será considerado o processo de treinamento, em que o modelo pode usar os elementos de $D_{target1}$.

Diferente de modelos como o RNN e o LSTM, em que a posição dos *tokens* é bem definida pelo uso de recorrência na arquitetura dos modelos, a arquitetura de *transformer*, por processar a sequência como um todo e não *token* por *token*, não modela a ordem dos *tokens*. A ordenação é essencial para a compreensão linguística da sentença, uma vez que, se alterada a ordem, o significado da sentença também poderá mudar. Para solucionar este problema, é somado às matrizes de entrada e saída uma matriz chamada de *positional encoding* (PE), que contém informações sobre a posição dos *tokens* na sequência. A dimensão de PE muda conforme a matriz à qual ela é somada, sendo $N_{input1} \times d_{model}$, quando somada à matriz de entrada, e $N_{output1} \times d_{model}$, quando somada à matriz de saída. Os elementos de PE podem ser calculados seguindo a seguinte equação proposta por Vaswani et al. (2017).

$$PE(pos, i) = \begin{cases} sin\left(\dfrac{pos}{10000^{2i/d_{model}}}\right) & \text{se } i \text{ é ímpar} \\ cos\left(\dfrac{pos}{10000^{2i/d_{model}}}\right) & \text{se } i \text{ é par} \end{cases} \tag{3.6}$$

Considerando a matriz de entrada como exemplo, na Equação 3.6, *pos* é o índice da posição do *token* na sequência, variando de 0 a $N_{input1} - 1$ e i é o índice de cada elemento dos *word embeddings*, variando de 0 a $d_{model} - 1$. É importante notar que o cálculo de PE depende somente de d_{model} e do número de *tokens* de cada documento, mas não do *token* específico da sequência. Isto implica que, se todos os *tokens* do *corpus* forem representados por *word embeddings* de mesmo tamanho, cada linha da matriz PE conterá sempre o mesmo valor, variando somente o número total de linhas da matriz. Inclusive, se todos os documentos do *corpus* possuírem o mesmo número de *tokens*, então a matriz PE será igual para todas as matrizes de entrada.

A soma de PE com as matrizes de entrada e saída é feita para que os *word embeddings* dos *tokens* que estão na mesma posição em documentos diferentes se-

jam deslocados para uma região específica do espaço vetorial. Espera-se que este deslocamento crie *clusters* posicionais no espaço vetorial, que representam a posição dos *tokens* em suas sequências, de maneira similar ao que ocorre com agrupamentos semânticos, ilustrados na Figura (2.1). Após somadas à seus respectivos *positional encoding*, estas novas matrizes de entrada e saída são processadas, respectivamente, pelo *encoder* e *decoder* do modelo.

3.2.2 Encoder

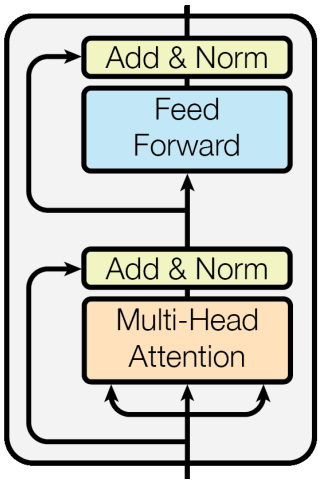

Figura 3.9: Encoder do modelo Transformer
Fonte: (Vaswani et al., 2017)

Considere o estado atual da matriz de entrada como "Estado 1". Dentro da estrutura do *encoder*, a matriz D_{input1} é processada por uma subcamada denominada *multi-head attention*, que é o cerne da arquitetura de *transformer*. Sua compreensão é de fundamental importância, pois não somente é nesta subcamada que o mecanismo de autoatenção é aplicado, como também esta subcamada é utilizada múltiplas vezes ao longo do modelo de *transformer*, seja em sua forma original ou com pequenas modificações.

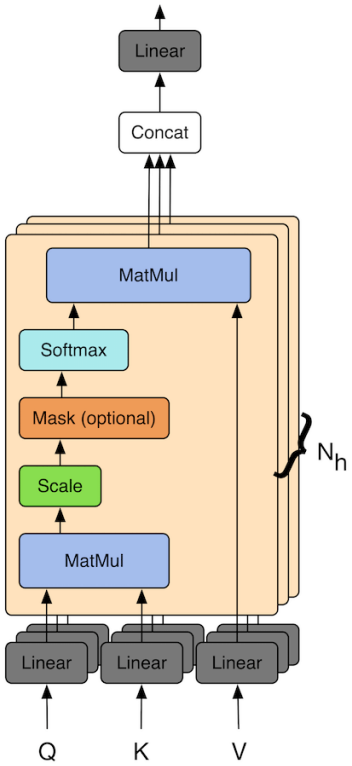

Figura 3.10: Detalhamento da subcamada Multi-Head Attention
Fonte: (Ghader, 2021)

No *multi-head attention*, existem três matrizes de pesos, que serão ajustados como parâmetros do modelo, denominadas $W^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W^K \in \mathbb{R}^{d_{model} \times d_k}$ e $W^V \in \mathbb{R}^{d_{model} \times d_v}$. As dimensões d_k e d_v são obtidas por meio da fórmula $d_k = d_v = d_{model}/h$, sendo h um hiperparâmetro do modelo que será explicado em breve. Estas três matrizes são utilizadas para calcular outras três matrizes, denominadas $\mathbf{Q} \in \mathbb{R}^{N_{input1} \times d_k}$ $(query)$, $\mathbf{K} \in \mathbb{R}^{N_{input1} \times d_k}$ (key) e $\mathbf{V} \in \mathbb{R}^{N_{input1} \times d_v}$ $(value)$. $\mathbf{Q}$, $\mathbf{K}$ e $\mathbf{V}$ são obtidas resolvendo as equações $\mathbf{Q} = D_{input1} \cdot W^Q$, $\mathbf{K} = D_{input1} \cdot W^K$ e $\mathbf{V} = D_{input1} \cdot W^V$. Ou seja, $\mathbf{Q}$, $\mathbf{K}$ e $\mathbf{V}$ extraem informações dos *tokens* de entrada que representam diferentes aspectos destes dados e que são utilizados no cálculo dos escores de atenção. Intuitivamente, a *query* representa aquilo que se deseja encontrar ou focar na sequência de entrada, *value* contém informações contextuais associadas a cada elemento na sequência de entrada e *key* auxilia na identificação de partes relevantes de *value* para uma determinada *query*.

Assim, a saída da subcamada de *multi-head attention*, pode ser calculada por meio da fórmula abaixo.

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}. \tag{3.7}$$

Na Equação 3.7, ilustrada na parte central da Figura (3.10), $\mathbf{Q} \cdot \mathbf{K}^T$ produz uma matriz de dimensão $N_{input1} \times N_{input1}$, em que cada elemento corresponde ao escore de atenção entre os *tokens* da sequência. Em seguida, os escores são escalonados ao serem divididos por $\sqrt{d_k}$. Isso impede um comportamento explosivo dos escores, uma vez que estes tendem a ser maiores conforme o tamanho dos *word embeddings*

de entrada d_{model} também aumenta. A matriz dos escores em escala passa pela função $softmax$, definida abaixo, para obter os pesos de atenção,

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}. \tag{3.8}$$

Esta função normaliza os escores para valores entre 0 e 1, de forma que o somatório dos elementos de cada linha da matriz dos pesos de atenção resulta em 1. Por fim, estes pesos são multiplicados pela matriz *value* para gerar uma matriz final com as mesmas dimensões que D_{input1}. Os pesos de atenção mais elevados manterão os valores dos *word embeddings* dos *tokens* quase inalterados, indicando ao modelo que estes são mais importante que os outros para o aprendizado. Já *tokens* com pesos de atenção menor são considerados pouco informativos ao modelo.

Espera-se que a representação computada pelo *multi-head attention* seja capaz de capturar relações contextuais dos *tokens* da sequência, ajudando o modelo a entender relações semânticas e sintáticas da língua. Para ser capaz de capturar diferentes padrões linguísticos e aumentar a velocidade do cálculo da Equação 3.7, o *transformer* separa as matrizes $\mathbf{Q}$, $\mathbf{K}$ e $\mathbf{V}$ em h conjuntos para serem processados em paralelo.

Considerando $h = 8$, as três matrizes são divididas em oito conjuntos, chamados de *head*. Cada matriz destes conjuntos tem dimensão 11×64 e cada *head* calcula a Equação 3.7 em paralelo, como ilustrado na Figura (3.10), daí o nome da subcamada ser *multi-head attention*. As matrizes finais de cada *head* são concatenadas e este resultado passa para a próxima etapa do *encoder*.

Ao final da subcamada de *multi-head attention* e, inclusive, ao final de todas as subcamadas do *encoder* e do *decoder*, é aplicada uma etapa de conexão residual (He et al., 2016) seguida pela normalização de camada (Ba et al., 2016), intitulada "*Add & Norm*" na Figura (3.9). Nesta etapa, é aplicada uma normalização de camada na soma entre a matriz resultante do *multi-head attention* com a matriz de entrada no "Estado 1". O resultado desta operação é uma matriz com as mesmas dimensões de D_{input1}. A conexão residual permite que os gradientes fluam diretamente pelas redes, enquanto que a normalização de camada auxilia na estabilização do modelo durante o treinamento, resultando em uma redução substancial do tempo de treinamento necessário.

A representação dos *tokens* passa então para a ultima subcamada do *encoder*, as redes neurais *feed-forward*. Nela, cada linha da matriz $N_{input1} \times d_{model}$ é processada por um modelo *Multi-Layer Perceptron* de uma camada oculta em paralelo, como descrito pela Equação 3.4. Transpondo os elementos desta equação para o contexto do *encoder*, $\mathbf{X}$ é o *word embedding* de um dos *tokens* da matriz, sendo um vetor de tamanho d_{model}. $\mathbf{W}_1$ tem dimensão $d_{ff} \times d_{model}$ e $\mathbf{b}_1 \in \mathbb{R}^{d_{ff} \times 1}$. d_{ff} é outro hiperparâmetro do modelo de *transformer* e os autores utilizaram um valor de $d_{ff} = 2048$ em Vaswani et al. (2017). Além disso, a função de ativação $g(.)$ utilizada é a ReLU (3.2), $\mathbf{W}_2 \in \mathbb{R}^{d_{model} \times d_{ff}}$ e $\mathbf{b}_2 \in \mathbb{R}^{d_{model} \times 1}$, o que resulta em um vetor de neurônios de saída $f(\mathbf{X})$ de tamanho d_{model}. Uma vez que os *word embeddings* de todos os *tokens* foram processados pelos modelos *feed-forward*, os vetores $f(\mathbf{X})$ de cada *token* são concatenados para formar novamente uma matriz $N_{input1} \times d_{model}$, sendo esta uma nova representação de D_{input1}. Esta subcamada é incluída no *transformer* para incorporar funções mais diversas ao modelo, como transformações não lineares. Sem as redes *feed-forward*, o *encoder* produziria somente uma média ponderada entre

os *word embeddings* da sequência de entrada. Após esta subcamada, é novamente aplicada a etapa de "*Add & Norm*" entre a representação atual e a representação imediatamente anterior à subcamada *feed-forward* e esta é a representação final de saída do *encoder* do modelo.

A matriz resultante do *encoder* codifica a sequência de entrada em uma representação contextual com informações de atenção, o que ajudará o *decoder* a focar nos *tokens* apropriados. No *transformer*, é possível utilizar uma pilha de N_x *encoders*, em que N_x é um hiperparâmetro e Vaswani et al. (2017) utilizaram $N_x = 6$ no modelo original. Nesta estrutura, a matriz resultante de um *encoder* é usada como entrada no próximo *encoder*. A matriz resultante do topo da pilha é considerada a representação final desta etapa da arquitetura. Para qualquer que seja a escolha de N_x, a fase de *decoder* também precisa utilizar uma pilha de N_x *decoders*. No exemplo de $N_x = 6$, a conexão entre a matriz resultante da pilha de *encoders* com a pilha de *decoders*, adaptando a Figura (3.7), pode ser ilustrada pela Figura (3.11).

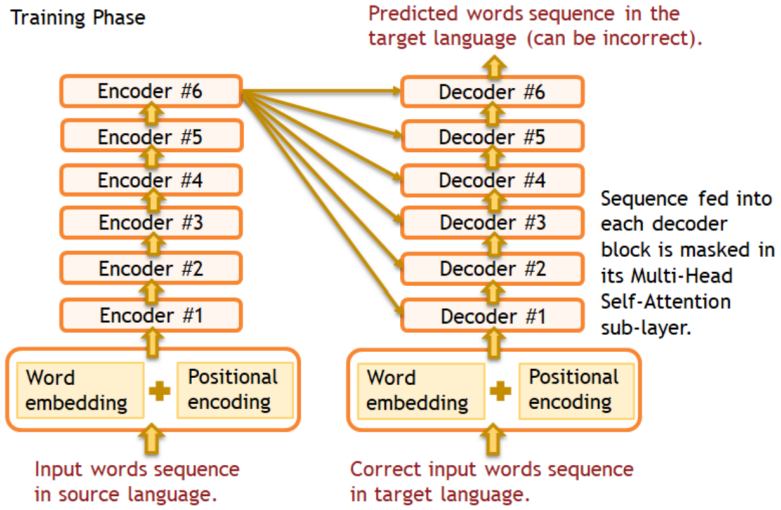

Figura 3.11: Ilustração da conexão entre a representação final dos encoders e os decoders

Fonte: (Ahmed et al., 2022)

A motivação de empilhar os *encoders* é permitir que o modelo capture propriedades brutas da sequência de entrada nos *encoders* inferiores e propriedades mais elaboradas nos *encoders* superiores. Por exemplo, Tenney et al. (2019) mostraram que as primeiras camadas se concentram na identificação de rotulagem gramatical, seguidas por constituintes, dependências, papéis semânticos e correferência.

3.2.3 Decoder

Uma vez que o modelo principal de estudo deste trabalho não utiliza o componente de *decoder* em sua arquitetura, não será feita uma explicação em profundidade deste componente, se limitando somente em sua ideia geral e nas etapas mais importantes para a sua compreensão.

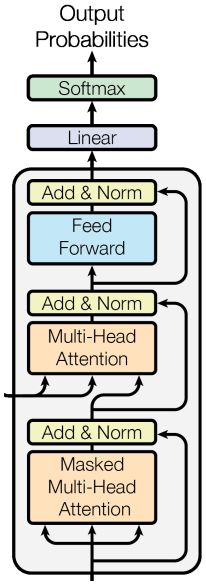

Figura 3.12: Decoder do modelo Transformer
Fonte: (Vaswani et al., 2017)

No *decoder*, existem algumas subcamadas similares às do *encoder*, como as conexões residuais e normalização de camada, as redes neurais *feed-forward* e o *multi-head attention*. Apesar das semelhanças com o *encoder*, o propósito do *decoder* e alguns detalhes internos de algumas destas subcamadas são diferentes. O objetivo do *decoder* é predizer qual é o próximo *token* da sequência de saída até que o *token* especial de fim de sequência, "EOF", seja predito ou até ser atingido um limite máximo de predições predefinido.

O *decoder* faz isso estimando a probabilidade de cada *token* do vocabulário ser o próximo da sequência e escolhendo aquele com a maior probabilidade. Para estimar estas probabilidades, o *decoder* recebe a matriz resultante do último *encoder* da pilha e do *decoder* anterior, ou da matriz de saída $D_{target1}$, caso o *decoder* em questão seja o primeiro da pilha. Durante o treinamento do *transformer* e seguindo o exemplo proposto, o *decoder* prevê o primeiro *token* da sequência de saída utilizando somente o primeiro *token* da sentença "*<START> the boy ate the cake because he was starving*", com o *token* "*<START>*" incluído na fase de pré-processamento. Supondo que o *token* predito seja "*one*", é calculado a diferença entre o *token* esperado, "*the*", e o *token* predito por meio da similaridade de cosseno (Faruqui et al., 2016) entre seus respectivos *word embeddings*. Utilizando esta métrica, é aplicado o algoritmo *back-propagation* para atualizar todos os parâmetros do modelo, como os pesos das redes neurais *feed-forward* e os pesos das matrizes W^Q, W^K e W^V. Para predizer o segundo *token* da sequência, o *transformer* utiliza os *tokens* "*<START>*" e "*the*" durante a fase de treinamento. Entretanto, na fase de validação e teste, são utilizados os *tokens* "*<START>*" e "*one*", ou seja, os *tokens* estimados anteriormente são utilizados na matriz de entrada para estimar o próximo *token*, o que atribui o comportamento auto-regressivo ao modelo.

Durante o treinamento, o *decoder* segue tentando estimar o *token* correto até o final da matriz de entrada, com o objetivo de ajustar os parâmetros do modelo. Se estes parâmetros forem estimados corretamente, espera-se que a sentença predita pelo modelo, a partir da sentença de entrada "o menino comeu o bolo porque ele estava morrendo de fome", seja "*the boy ate the cake because he was starving EOF*".

3.3 BERT

No ano seguinte à publicação da arquitetura de *transformer*, Devlin et al. (2018) propuseram o modelo BERT (Representações de *Encoders* Bidirecionais de *Transformers*). Este modelo foi projetado para pré-treinar representações bidirecionais, utilizando ambas as direções de uma sequência textual simultaneamente para ajustar os *word embeddings* dos *tokens*. Isto contrasta com o ELMo, que produz uma concatenação superficial de vetores da esquerda para a direita e da direita para a esquerda, os quais são treinados independentemente. Entretanto, assim como o ELMo, o BERT também gera *word embeddings* contextuais. Por meio do BERT, Devlin et al. (2018) melhoram o estado da arte em onze tarefas de NLP, como resposta a perguntas e inferência linguística.

A arquitetura do modelo BERT é formada por uma pilha de N_e *encoders* baseados na implementação original do *transformer* de Vaswani et al. (2017), como ilustrado na Figura (3.9). A entrada do BERT é uma sentença textual e, após a fase de pré-processamento, os *word embeddings* dos *tokens* da sequência são processados pela pilha de *encoders* na forma de uma matriz de entrada. A saída do último *encoder* da pilha é uma matriz de mesma dimensão que a matriz de entrada, em que cada linha desta matriz é um *word embedding* contendo informações contextuais dos *tokens* obtidos pelo modelo.

Devlin et al. (2018) projetaram o modelo BERT para possuir dois estágios: pré-treinamento e ajuste fino. Durante o pré-treinamento, o modelo é treinado em diferentes tarefas utilizando um *corpus* genérico na língua inglesa com o objetivo de tornar o modelo capaz de compreender componentes linguísticos básicos. Este estágio utiliza o conceito de transferência de aprendizagem (Ruder et al., 2019), em que se deseja treinar um modelo com melhores propriedades de generalização para que se possa reduzir o tempo total de treinamento de outras tarefas distintas. Para o ajuste fino, é feita uma modificação somente na última camada do BERT, que é então inicializado com os parâmetros do modelo pré-treinado. Em seguida, na fase de treinamento, todos os parâmetros do modelo são ajustados, podendo ser aplicado para diferentes tarefas a depender do interesse de pesquisa. Cada tarefa possui modelos separados, embora haja uma diferença mínima nas arquiteturas durante o ajuste fino tanto entre si quanto entre elas e a arquitetura pré-treinada. Esta unificação das arquiteturas do BERT em diferentes tarefas torna o modelo notavelmente flexível.

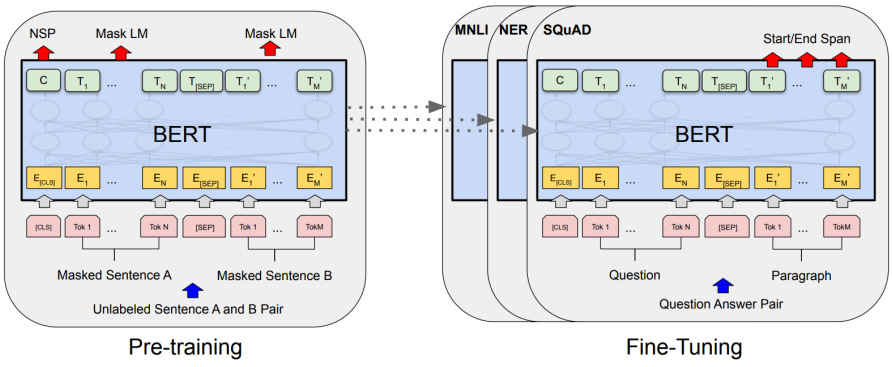

Figura 3.13: Procedimentos gerais de pré-treinamento e ajuste fino do BERT
Fonte: (Devlin et al., 2018)

3.3.1 Pré-treinamento

Devlin et al. (2018) pré-treinaram o BERT utilizando duas tarefas: *Masked Language Model* (MLM) e *Next Sentence Prediction* (NSP).

O MLM, inspirado pela tarefa *cloze* (Taylor, 1953), é a principal inovação do BERT, uma vez que é nesta tarefa que a representação bidirecional profunda é obtida. O algoritmo do MLM consiste em selecionar aleatoriamente 15% dos *tokens* de cada sequência de entrada para serem mascarados por um *token* denominado [MASK]. O objetivo desta tarefa é prever qual é a palavra mascarada e, intuitivamente, espera-se que o modelo utilize a informação dos *tokens* ao redor de [MASK] para alcançar tal objetivo.

Uma desvantagem deste método é que pode ocorrer uma discrepância entre o modelo pré-treinado e um modelo com ajuste fino. Porque no pré-treinamento é utilizada a tarefa MLM, mas no ajuste fino o modelo pode ser treinado para realizar uma tarefa completamente diferente, é possível que os *word embeddings* gerados pelo modelo pré-treinado não sejam representativos o suficiente para os *tokens* do estágio de ajuste fino. Para mitigar este problema, quando um *token* for escolhido para ser mascarado, o que ocorre em 15% dos *tokens* de cada sequência, ele será: de fato mascarado pelo *token* [MASK] em 80% das vezes, substituído por um *token* aleatório em 10% das vezes e mantido inalterado em 10% das vezes. A vantagem deste procedimento é que o *encoder* é forçado a manter uma representação contextual de todos os *tokens* de entrada, já que o *encoder* não sabe quais palavras ele será solicitado a prever ou quais foram substituídas por palavras aleatórias (Devlin et al., 2018).

Existem muitas tarefas relevantes de NLP que envolvem compreender a relação entre duas sequências de texto, como resposta a pergunta e inferência de linguagem natural. Tendo em vista que o objetivo do modelo pré-treinado é ser generalizável para outras tarefas, é fundamental que ele possa representar a relação entre duas sentenças. Na tarefa de NSP, a sequência de entrada do modelo é a concatenação de duas sentenças textuais distintas, denominadas A e B. Em 50% das vezes, B é a próxima sentença real que segue de A e em 50% das vezes B é uma sentença aleatória do *corpus*. O objetivo desta tarefa é prever se B é o segmento de A ou não.

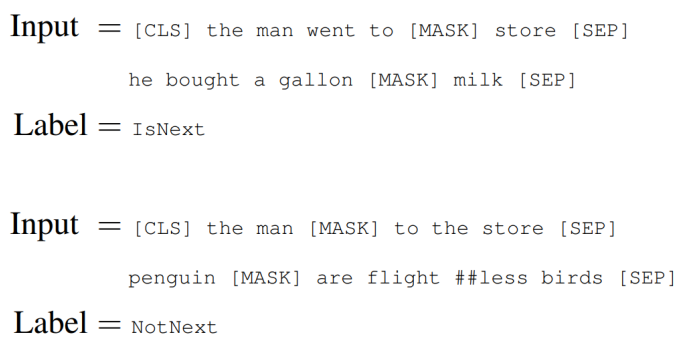

Figura 3.14: Exemplos da tarefa de NSP na fase de pré-treinamento
Fonte: (Devlin et al., 2018)

Assim como ocorre no treinamento do *transformer*, as sentenças textuais de entrada precisam passar por uma fase de pré-processamento antes de serem pré-

treinadas pela pilha de *encoders* do BERT. Considere as sentenças:

$$D_{input2} = \text{“}my\ dog\ is\ cute\text{”}$$
$$D_{input3} = \text{“}he\ likes\ playing\text{”}.$$

A primeira etapa desta fase consiste em *tokenizar* ambas as sentenças por meio de algum método de *tokenização*. O BERT utiliza um método específico chamado de *WordPiece* (Wu et al., 2016). Neste método, os elementos das sentenças são divididos em subpalavras para capturar variações morfológicas, otimizar o armazenamento do vocabulário e lidar com palavras que ficariam fora do vocabulário do modelo. Por exemplo, a sentença D_{input3} pode ser *tokenizada* como {“*he*”, “*likes*”, “*play*”, “*##ing*”}. O prefixo “*##*” indica que aquela subpalavra forma uma palavra completa se concatenada com a anterior. Após *tokenizar* as sentenças, é necessário inserir um *token* especial, [SEP], ao final de cada uma delas, indicando ao modelo que aquela sequência foi encerrada. Então, as sequências *tokenizadas* D_{input2} e D_{input3} são concatenadas em uma só, denominada D_{input} e é inserido outro *token* especial, [CLS], no início de D_{input}. Por fim, cada *token* desta sequência é convertido em sua representação de *word embedding* com tamanho d_{model}. Esta sequência está ilustrada na Figura (3.15), ao lado da expressão *Token Embeddings*.

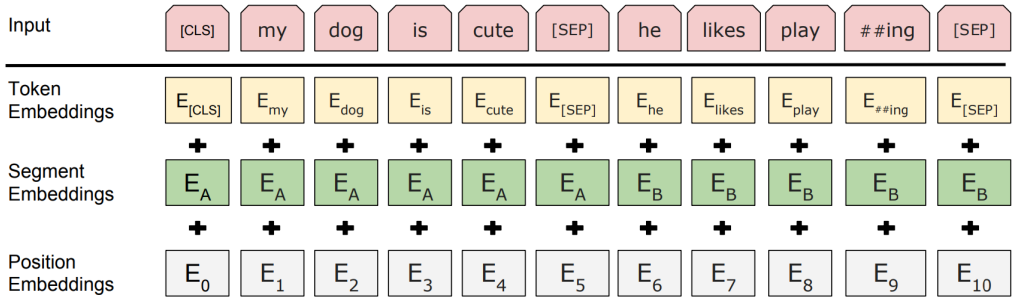

Figura 3.15: Representação de entrada do BERT
Fonte: (Devlin et al., 2018)

O *token* [CLS], também chamado de *token* de classificação, é responsável por representar o sentido da sequência de entrada como um todo em seu *word embedding*, sendo muito utilizado para tarefas de classificação. Para obter a representação de entrada final do modelo BERT, os elementos de D_{input} são somados a dois conjuntos de vetores: *position embedding* e *segment embedding*. O *position embedding* segue o mesmo princípio da arquitetura de *transformer* ao somar a matriz de entrada à uma matriz de *positional encoding*, para incluir a representação de ordem entre os *token*. Cada vetor de D_{input} é somado a um vetor correspondente de *positional embedding* e espera-se que este deslocamento no espaço vetorial seja o suficiente para representar a ordenação dos elementos da sequência ao modelo.

O *segment embedding*, por sua vez, é responsável por diferenciar as sentenças dentro da sequência de entrada. No contexto de pré-treinamento, são utilizadas as sentenças A e B para a tarefa de NSP. Portanto, cada vetor de D_{input}, ao ser somado com um vetor correspondente de *segment embedding*, será deslocado para uma entre duas direções no espaço vetorial, de acordo com a sentença a qual pertencem. Por último, é aplicado o mascaramento a alguns dos elementos da sequência, de acordo com o algoritmo de MLM. Este processo final da fase de pré-processamento está ilustrado na Figura (3.15), com exceção da etapa de mascaramento.

Finalizada a etapa de pré-processamento, D_{input} é convertida em uma matriz de entrada $N_{input} \times d_{model}$, em que $N_{input} = 11$ é o número de *tokens* da sequência de entrada e d_{model} é o tamanho dos *word embeddings* destes *tokens*. Em seguida, D_{input} é processada pela pilha de N_e *encoders*, conforme explicado na Seção 3.2.2, para resolver as tarefas de MLM e NSP. O objeto de saída do último *encoder* da pilha é chamado de matriz de saída, ou D_{output}, e possui as mesmas dimensões de D_{input}. Cada linha de D_{output} contém *word embeddings* contextualizados dos *tokens* de entrada, podendo ser utilizada para um estágio posterior de ajuste fino. Além de descreverem o algoritmo do modelo BERT, Devlin et al. (2018) desenvolveram dois modelos pré-treinados, denominados $\text{BERT}_{\text{BASE}}$ e $\text{BERT}_{\text{LARGE}}$, ilustrados na Figura (3.16). O $\text{BERT}_{\text{BASE}}$ possui 110 milhões de parâmetros e foi treinado em 4 TPUs por 4 dias. Já o $\text{BERT}_{\text{LARGE}}$ possui 340 milhões de parâmetros e foi treinado em 16 TPUs por 4 dias. Estes modelos estão disponíveis para uso no repositório https://github.com/google-research/bert, sob a licença *Apache* 2.0.

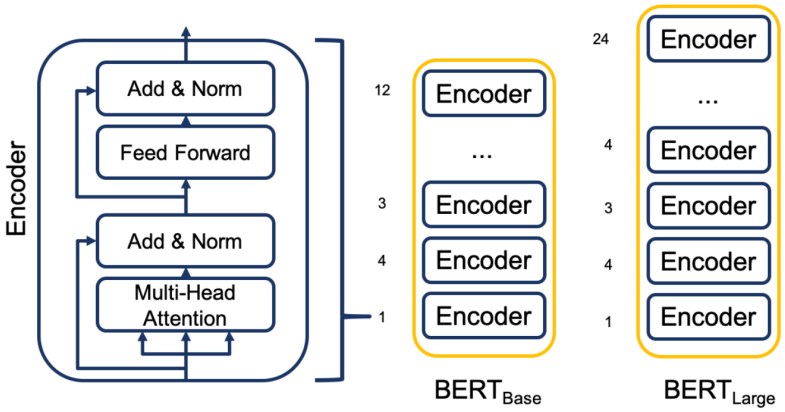

Figura 3.16: Pilha de encoders do BERT
Fonte: (Evtimov et al., 2020)

3.3.2 Ajuste fino

Para ilustrar o estágio de ajuste fino, considere uma tarefa de classificação binária, em que se deseja determinar se uma crítica de filme é positiva ou negativa com base no texto da crítica em inglês. Considere ainda, uma sentença de entrada:

$$D_{input} = \text{“}\textit{This movie is awesome}\text{”}.$$

É possível ajustar um modelo BERT, denominado M_1, para categorizar as sentenças textuais entre positiva ou negativa. Primeiramente, é necessário considerar algumas modificações na fase de pré-processamento em relação ao modelo pré-treinado. Uma vez que somente uma sentença é processada pelo modelo por vez, os vetores de *segment embedding* serão todos iguais para qualquer sentença. Assim, D_{input} pode ser *tokenizada* como:

$$\{\text{“[CLS]”}, \text{“}\textit{this}\text{”}, \text{“}\textit{movie}\text{”}, \text{“}\textit{is}\text{”}, \text{“}\textit{awesome}\text{”}, \text{“[SEP]”}\}.$$

A matriz de entrada é obtida seguindo os mesmos passos descritos na Seção 3.3.1. Em seguida, para tomar proveito de modelos pré-treinados publicamente disponíveis, é possível inicializar os parâmetros de M_1 com os mesmos valores dos parâmetros de

algum modelo como, por exemplo, o $\text{BERT}_{\text{LARGE}}$. É importante notar que alguns hiperparâmetros, como o tamanho da pilha de *encoders*, serão os mesmos utilizados no $\text{BERT}_{\text{LARGE}}$, não podendo ser modificados em M_1. Para classificar D_{input}, será necessário utilizar o *word embedding* relativo ao *token* [CLS] na matriz de saída do modelo, uma vez que ele encapsula o sentido contextualizado de toda a sequência de entrada. [CLS] é um vetor com dimensões $\mathbb{R}^{d_{model} \times 1}$ e, a partir dele, é possível treinar uma matriz de pesos $W_{cls} \in \mathbb{R}^{2 \times d_{model}}$ com o intuito de obter um vetor $L_{cls} \in \mathbb{R}^{2 \times 1}$, seguindo a fórmula abaixo:

$$L_{cls} = W_{cls} \cdot [\text{CLS}] \tag{3.9}$$

Considerando $d_{model} = 512$, W_{cls} possui 1024 pesos a serem ajustados durante a fase de treinamento. Os outros parâmetros, importados do $\text{BERT}_{\text{LARGE}}$, também serão ajustados durante o treinamento, mas espera-se que estes valores não sejam alterados substancialmente, visto que eles já foram pré-treinados. Os elementos de L_{cls} são chamados de *logits*, que são os resultados brutos do modelo antes de serem transformados em probabilidades. Pode-se considerar, por exemplo, o primeiro elemento de L_{cls} como classificação negativa e o segundo elemento como classificação positiva. A partir dos *logits*, é escolhido o elemento com o maior valor e a posição deste elemento será a classe predita. Como ilustração, caso $L_{cls} = \begin{bmatrix} 4.85 & -5.11 \end{bmatrix}^T$, a sentença de entrada será categorizada como negativa e, caso $L_{cls} = \begin{bmatrix} -3.99 & 4.68 \end{bmatrix}^T$ a sentença será categorizada como positiva. Se os pesos de W_{cls} forem bem ajustados, espera-se que M_1 obtenha um bom desempenho ao classificar uma nova crítica textual em inglês entre positiva ou negativa.

Para além de classificação binária, a Figura (3.13) ilustra a modificação necessária na arquitetura do BERT para a tarefa de resposta a perguntas. A flexibilidade da etapa de ajuste fino auxiliou diversos autores a desenvolverem novos modelos de NLP a partir da arquitetura do BERT, como o *Robustly optimized* BERT *approach* (RoBERTa) (Liu et al., 2019), o *XLNet* (Yang et al., 2019) e o *Vision-and-Language* BERT (ViLBERT) (Lu et al., 2019).

4 Resultados

Neste capítulo, descrevemos o banco de dados utilizado no ajuste de um modelo BERT, assim como as etapas de pré-processamento aplicadas aos dados. Em seguida, apresentamos o detalhamento das etapas de treinamento, validação e teste do modelo ajustado. Para o desenvolvimento desta análise, foram utilizadas as linguagens de programação **Python** (versão 3.11.3), em conjunto com o editor de código-fonte *Visual Studio Code*, e **R** (versão 4.2.1), em conjunto com o ambiente de desenvolvimento *RStudio*. Ainda, foram utilizados pacotes em Python como *numpy* (Harris et al., 2020) (versão 1.24.4), *pandas* (Wes McKinney, 2010) (versão 2.0.3), *transformers* (Wolf et al., 2019) (versão 4.30.2) e *torch* (Paszke et al., 2019) (versão 2.0.1), bem como o pacote em *R ggplot2* (Wickham, 2016) (versão 3.4.0).

4.1 Banco de dados

O banco de dados original (Verma et al., 2021) foi obtido pelo portal *Kaggle* e contém informações acerca de 72.134 artigos de notícias na língua inglesa, dos quais 35.028 são notícias verdadeiras e 37.106 são falsas. Para compor o banco, Verma et al. (2021) extraíram artigos dos portais *Kaggle*, *McIntire*, *Reuters* e *BuzzFeed Political*. O conjunto de dados possui 4 colunas: "identificação", que é um número de identificação único da notícia; "título", que apresenta o título da notícia no formato de texto; "texto", que apresenta o conteúdo da notícia no formato de texto; e *"label"*, que é 0 para notícias verdadeiras e 1 para falsas. O objetivo neste capítulo é utilizar as colunas "título" e *"label"* do banco de dados apresentado para ajustar um modelo BERT para classificação binária supervisionada de texto, capaz de detectar quando o título de uma notícia em inglês provém de um artigo falso ou não.

4.1.1 Pré-processamento

Na fase de pré-processamento foram mantidas somente as colunas de interesse "título" e *"label"* e, em seguida, foram removidas 558 observações que continham dados faltantes em pelo menos uma das colunas de interesse. A seguir, foram retiradas 9.233 observações por conterem valores duplicados na coluna "título". Além disso, foi observado que algumas notícias do banco não estavam na língua inglesa. Para corrigir este problema, foi utilizado o pacote *langdetect* (versão 1.0.9) do Python para detectar a língua de cada título e 209 observações foram removidas do banco por estarem em outras línguas como russo, árabe ou chinês. Durante o pré-processamento, não foram removidos sinais de pontuação ou *stop words*, que é um conjunto de pala-

vras que são normalmente desconsideradas da análise, como preposições ou conjunções. Esta escolha foi tomada porque a remoção de tais *tokens* implicaria na remoção de contexto que o modelo poderia ter usado para obter melhores resultados. Além disso, pesquisadores como Schofield et al. (2017) têm debatido se a remoção de *stop words* de fato impacta positivamente no desempenho de modelos de NLP.

Finalizada a fase de pré-processamento, o banco de dados final possui 62.134 observações, das quais 34.404 são notícias verdadeiras e 27.730 são falsas. É possível notar que os dados são levemente desbalanceados, pois aproximadamente 55,37% das notícias são verdadeiras e 44,63% são falsas.

Como exemplo de um título de uma notícia verdadeira, temos: *"Bobby Jindal, raised Hindu, uses story of Christian conversion to woo evangelicals for potential 2016 bid"*.

Já como exemplo de um título de uma notícia falsa, temos: *"UNBELIEVABLE! OBAMA'S ATTORNEY GENERAL SAYS MOST CHARLOTTE RIOTERS WERE PEACEFUL PROTESTERS... In Her Home State Of North Carolina [VIDEO]"*.

4.2 Ajuste do modelo

Utilizando o conceito de transferência de aprendizagem para aumentar a velocidade de treinamento, os parâmetros do modelo foram inicializados a partir do modelo pré-treinado $\text{BERT}_{\text{BASE}}$ *uncased*, que não faz distinção entre caracteres maiúsculos e minúsculos. O pacote *transformers* fornece uma classe denominada *BertForSequenceClassification*, que carrega os parâmetros do modelo pré-treinado e fornece ferramentas que auxiliam no estágio de ajuste fino em tarefas de classificação de sequência. Como estamos interessados em fazer uma classificação binária, utilizamos um classificador com somente duas categorias.

Este modelo possui um objeto chamado de *tokenizador*, que lida com o processo de *tokenização* das sequências de texto. Utilizando o *tokenizador* do BERT, verificamos que os números mínimo, máximo e médio de *tokens* dos títulos do banco são de, respectivamente, 3, 96 e 18,1779. Além disso, caracteres especiais como "ã", "é" e "ç" já são automaticamente convertidos pelo *tokenizador* para "a", "e" e "c", respectivamente.

Foi definido um limite máximo de 64 *tokens* para cada sequência de títulos, truncando a sequência caso ela ultrapasse este limite. Somente 3 observações do banco foram truncadas como consequência desta definição. Após *tokenizarmos* todas as sentenças da coluna "título", vamos dividir o banco entre treinamento, validação e teste, utilizando a proporção de separação 80%, 10% e 10% para cada partição, respectivamente. Desta forma, foram alocadas aleatoriamente 49.707 observações para o banco de treinamento, 6.213 para o banco de validação e 6.214 para o banco de teste. A proporção de sentenças verdadeiras e falsas nestes três bancos e no banco pré-processado total é aproximadamente a mesma.

Na fase de validação, buscou-se escolher o melhor conjunto de hiperparâmetros, tendo como base o desempenho do modelo no banco de validação. Para tarefas de ajuste fino de um modelo BERT pré-treinado, Devlin et al. (2018) recomendam utilizar um tamanho do mini-lote de 16 ou 32, uma taxa de aprendizado de 2e-05, 3e-05 ou 5e-05 e um número de épocas de 2, 3 ou 4. Foram ajustados 18 modelos com as primeiras 2000 observações do banco de dados e, para cada ajuste, foi

utilizada uma combinação diferente dos valores de hiperparâmetros recomendados por Devlin et al. (2018). Observou-se os resultados e, baseado na análise de métricas como acurácia, perda e tempo de ajuste dos modelos no banco de validação, foram escolhidos o tamanho do mini-lote de 32, a taxa de aprendizado de 3e-05 e 3 épocas. Os resultados da fase de validação podem ser consultados na Tabela 4.1. Então, estes hiperparâmetros foram utilizados na fase de treinamento, que levou aproximadamente 12 horas para finalizar o ajuste do modelo.

Tabela 4.1: Medidas de avaliação do modelo com diferentes combinações de hiperparâmetros. Foram utilizadas 1600 observações para o banco de treinamento e 200 para o banco de validação.

Época	Tamanho do mini-lote	Taxa de aprendizado	Acurácia de validação	Perda de validação	Tempo de validação	Acurácia de treinamento	Perda de treinamento	Tempo de treinamento
4	32	3e-05	0.9375	0.1940	00:00:17	0.9769	0.0775	00:06:10
4	32	5e-05	0.9375	0.2229	00:00:18	0.9900	0.0331	00:07:03
3	16	5e-05	0.9231	0.2479	00:00:17	0.9669	0.0987	00:07:23
4	16	3e-05	0.9231	0.3033	00:00:17	0.9856	0.0508	00:07:26
4	16	5e-05	0.9231	0.3370	00:00:17	0.9906	0.0290	00:07:19
3	16	3e-05	0.9183	0.2595	00:00:17	0.9669	0.1025	00:07:22
3	16	2e-05	0.9183	0.2680	00:00:17	0.9556	0.1335	00:07:19
4	16	2e-05	0.9183	0.2787	00:00:17	0.9769	0.0839	00:07:25
3	32	3e-05	0.9152	0.1983	00:00:17	0.9525	0.1435	00:06:35
2	32	5e-05	0.9152	0.2433	00:00:17	0.9313	0.1894	00:06:35
2	16	5e-05	0.9135	0.2292	00:00:18	0.9375	0.1823	00:06:58
3	32	5e-05	0.9107	0.2036	00:00:16	0.9581	0.1145	00:06:37
2	16	3e-05	0.8990	0.2880	00:00:17	0.9456	0.1796	00:07:22
2	16	2e-05	0.8942	0.2646	00:00:13	0.9256	0.2158	00:14:18
4	32	2e-05	0.8929	0.2791	00:00:17	0.9681	0.1112	00:06:37
3	32	2e-05	0.8884	0.2705	00:00:16	0.9413	0.1802	00:06:37
2	32	3e-05	0.8795	0.2838	00:00:16	0.9206	0.2265	00:06:04
2	32	2e-05	0.8750	0.3195	00:00:17	0.9113	0.2691	00:06:26

4.3 Avaliação dos resultados

A seguir, estão descritas as definições das medidas utilizadas para avaliar os resultados do modelo ajustado. Os verdadeiros positivos (TP) representam o número de observações que foram corretamente preditas como positivas pelo modelo. Uma vez que *"label"* é 1 quando a notícia é falsa e 0 quando ela é verdadeira, uma predição positiva indica a classificação da notícia como falsa e, de maneira análoga, uma predição negativa indica a classificação da notícia como verdadeira. Os verdadeiros negativos (TN) representam o número de observações que foram corretamente preditas como negativas pelo modelo. Já os falsos positivos (FP) indicam o número de observações preditas incorretamente como positivas pelo modelo. Por fim, os falsos negativos (FN) indicam a quantidade de observações preditas incorretamente como negativas pelo modelo. As estatísticas TP, TN, FP e FN podem ser expressadas em uma matriz de confusão, ilustrada na Tabela 4.2.

Tabela 4.2: Definição dos elementos de uma matriz de confusão

	Predito positivo	Predito negativo	Somatório
Verdadeiro positivo	TP	FN	TP + FN
Verdadeiro negativo	FP	TN	FP + TN
Somatório	TP + FP	FN + TN	TP + TN + FP + FN

A acurácia (Hossin e Sulaiman, 2015) é a proporção entre o número de observações corretamente preditas sobre o número total de observações. Quando o conjunto de dados é desbalanceado, analisar somente a acurácia pode nos levar a conclusões errôneas. A precisão mede a proporção de previsões positivas verdadeiras entre todas as previsões positivas, ou ainda, $TP/(TP + FP)$. Esta medida pode ser útil nos casos em que os falsos positivos prejudicam mais os resultados do que os falsos negativos. A sensibilidade corresponde à proporção de casos positivos reais corretamente previstos pelo modelo, ou seja, $TP/(TP + FN)$. Quando os falsos negativos são mais preocupantes do que os falsos positivos esta métrica pode ser adequada. A especificidade, por sua vez, mede a proporção de predições negativas verdadeiras entre todas as observações negativas reais, ou ainda, $TN/(TN + FP)$. Além disso, o escore F1 é definido como a média harmônica entre a precisão e a sensibilidade, sendo útil ao lidar com conjuntos de dados desbalanceados. O cálculo do escore F1 é dado por $2 \times \frac{\text{precisão} * \text{sensibilidade}}{\text{precisão} + \text{sensibilidade}}$. Finalmente, a perda de treinamento, ou validação, é obtida por meio do cálculo da função custo para um mini-lote específico dos dados de treinamento, ou validação. Todas as proporções apresentadas estão no formato de porcentagem para facilitar a visualização.

Tabela 4.3: Medidas de avaliação do modelo nos bancos de treinamento e validação em cada época

Época	1	2	3
Perda média de treinamento	0,1943	0,0943	0,0425
Perda média de validação	0,1529	0,1825	0,2424
Acurácia de treinamento	92,1976%	96,7002%	98,7934%
Acurácia de validação	94,4872%	94,4551%	94,9679%
Tempo de treinamento	3:55:57	3:45:13	3:44:42
Tempo de validação	0:10:01	0:09:05	0:09:19

A Tabela 4.3 apresenta algumas medidas extraídas dos bancos de treinamento e validação ao longo da fase de treinamento. Nota-se que a perda média entre os mini-lotes e a acurácia de treinamento diminuiu e aumentou, respectivamente, durante a fase de treinamento. Isso parece indicar que o modelo foi capaz de aprender com os dados, obtendo uma acurácia final de 98,79% no banco de treinamento. Entretanto, é possível observar que a perda média de validação aumentou ao longo das épocas, o que pode ser um indício de *overfitting* do modelo. Como a acurácia de validação se manteve aproximadamente constante ao longo das épocas, o indício de *overfitting* observado na perda média de validação pode não ser tão expressivo.

A Tabela 4.4, apresenta os valores de TP, TN, FP e FN observados no banco de teste. Analisando estes dados, há um número maior de falsos negativos, 201, em comparação com os falsos positivos, 132. Isso parece sinalizar que o modelo classifica incorretamente mais as *fake news* como verdadeiras, do que as notícias

Tabela 4.4: Matriz de confusão do banco de teste com valores puros

	Predito positivo	Predito negativo	Somatório
Verdadeiro positivo	2565	201	2766
Verdadeiro negativo	132	3316	3448
Somatório	2697	3517	6214

verdadeiras como falsas. Intuitivamente, consideramos que o pior erro do modelo é o falso negativo, pois, se uma notícia provém de uma fonte idônea, não deve ser difícil perceber um erro de classificação do modelo caso ele categorize uma notícia desta fonte como falsa. Contudo, para desenvolver um processo automatizado de detecção de *fake news*, é de suma importância que o modelo não deixe de identificar corretamente uma notícia falsa.

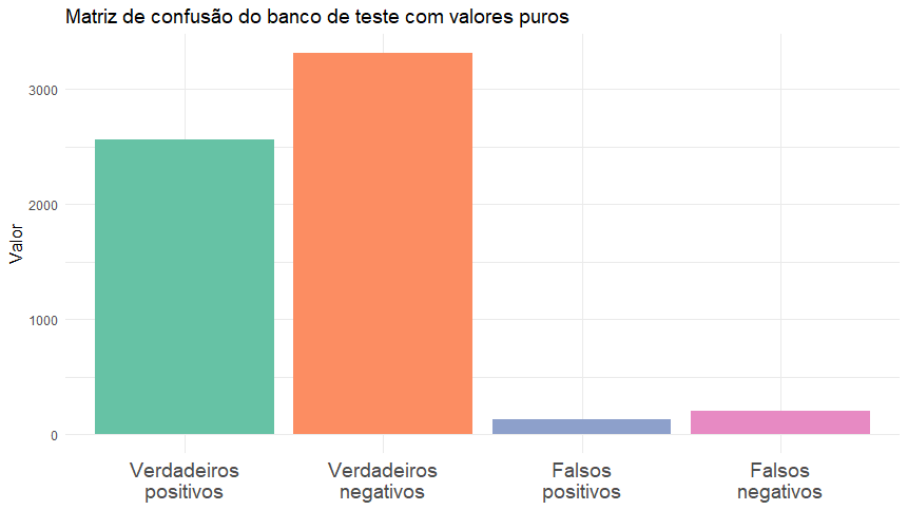

Figura 4.1: Matriz de confusão ilustrada em um gráfico de barras

Na Figura (4.1), temos a ilustração gráfica dos valores analisados na Tabela 4.4. Comparando o tamanho das barras, a diferença entre os falsos positivos e negativos não parece ser tão substancial, o que aponta para um ajuste razoável do modelo.

Tabela 4.5: Matriz de confusão do banco de teste com valores em porcentagem

	Predito positivo	Predito negativo	Somatório
Verdadeiro positivo	41,2778%	3,2346%	44,5124%
Verdadeiro negativo	2,1242%	53,3634%	55,4876%
Somatório	43,4020%	56,5980%	100%

Na Tabela 4.5, temos os mesmos dados apresentados na Tabela 4.4, mas em valores proporcionais, obtidos através da divisão de cada elemento da matriz de confusão pela soma de todos os seus elementos. Em porcentagem, a diferença entre os falsos positivos e negativos não parece tão grande, confirmando o que foi observado na Figura (4.1). Além disso, considerando que aproximadamente 55,37% das notícias do banco de teste são verdadeiras e 44,63% são falsas, a tabela acima parece indicar valores aceitáveis de verdadeiros positivos e negativos.

Tabela 4.6: Medidas de avaliação do modelo ajustado no banco de teste

Medida de avaliação	Valor
Acurácia	94,6411%
Precisão	95,1057%
Sensibilidade	92,7332%
Especificidade	96,1717%
Escore F1	93,9044%

A Tabela 4.6 contém as principais medidas para avaliar a qualidade do ajuste do modelo treinado, obtidas no banco de teste. A sensibilidade foi a medida com o pior desempenho, de 92,73%. Visto que ela mede a capacidade do modelo de prever corretamente quando uma notícia é *fake news*, um valor baixo desta medida indica uma alta taxa de falsos negativos, considerado o pior tipo de erro para esta análise. Todavia, 92,73% pode ser interpretado como um valor alto, principalmente se considerarmos que este foi a menor das medidas observadas. A medida com o melhor desempenho foi a especificidade, com 96,17%. Esta medida possui um comportamento inverso ao da sensibilidade, sendo a taxa de notícias verdadeiras corretamente preditas. Isso significa que o modelo é particularmente bom para detectar quando uma notícia é verdadeira. Analisando o escore F1, observamos que ele também atingiu um valor elevado, de 93,90%. Considerando que o escore F1 é uma medida que penaliza a presença de dados desbalanceados, podemos inferir que o desequilíbrio presente na coluna *"label"* não afetou o desempenho do modelo. Por último, a acurácia e precisão no banco de teste foram de 94,64% e 95,11%, respectivamente. Por serem valores calculados no banco de teste, eles parecem indicar a ausência de *overfitting* do modelo. Além disso, estes pode ser considerados bons resultados, sugerindo que o modelo obteve um bom desempenho de maneira geral.

Como exemplo de um título de uma notícia verdadeira, mas que o modelo classificou como falsa, temos: *"Melania Trump Is First Catholic to Live in White House Since JFK"*.

Já como exemplo de um título de uma notícia falsa, mas que o modelo classificou como verdadeira, temos: *"In UNESCO, Palestinians claim ownership of Dead Sea Scrolls"*.

Os códigos utilizados para ajustar e avaliar o modelo BERT estão disponíveis para consulta no repositório `https://github.com/ghsaul/tcc_project`, sob a licença *Creative Commons Zero v1.0 Universal*.

5 Conclusão

Neste trabalho, foi apresentada uma aplicação do modelo BERT para identificação de *fake news* em títulos de notícias de diferentes meios de comunicação americanos. Como o banco de dados possui uma variável indicando a veracidade da notícia, o problema abordado nesse trabalho envolve métodos de aprendizagem supervisionado, especificamente de classificação binária. O uso do modelo pré-treinado $\text{BERT}_{\text{BASE}}$ *uncased* e de ferramentas de programação, forneceu uma abordagem unificada para o treinamento do modelo, economizando tempo e recursos computacionais. Esta estrutura de pré-treinamento e ajuste fino que o modelo oferece, é uma das razões pelas quais o BERT se tornou uma escolha popular e eficaz na área de Processamento de Linguagem Natural (Zhao et al., 2022).

Baseando-se em métricas como acurácia, precisão e sensibilidade, concluímos que o modelo ajustado obteve um bom desempenho na tarefa de classificar notícias falsas em inglês. Isto parece evidenciar que modelos capazes de representar o significado de uma palavra a partir do seu contexto podem alcançar um bom desempenho em tarefas de NLP. Os resultados deste trabalho podem contribuir no combate à desinformação e auxiliar aplicações na área como Giordani et al. (2022). Apesar do bom desempenho, o método pode ser aperfeiçoado ainda mais. Caso fosse utilizado o texto completo da notícia em vez de apenas o seu título, o modelo possivelmente ganharia um contexto mais abrangente. No entanto, isto não foi feito devido à limitações de recursos computacionais.

Em trabalhos futuros, é possível utilizar alguma modificação do modelo GPT (Radford et al., 2018) para produzir avanços no estudo de detecção de *fake news*. Além disso, é interessante aplicar o método de gradientes integrados (Sundararajan et al., 2017), implementado no pacote em Python *captum* (Kokhlikyan et al., 2020). Esta técnica permite visualizar a importância de cada *token* à predição do modelo, contribuindo com a interpretabilidade do BERT. Por fim, é possível usar variações do BERT que utilizam texto e imagem simultaneamente como dado de entrada do modelo, para analisar *fake news* não somente em textos, mas também em imagens alteradas digitalmente. Dentre os modelos disponíveis deste tipo, temos o *Visual-Linguistic* BERT (VL-BERT) (Su et al., 2019), o *UNiversal Image-TExt Representation* (UNITER) (Chen et al., 2020) e o VisualBERT (Li et al., 2019).

Referências Bibliográficas

Ahmed, S., Nielsen, I., Tripathi, A., Siddiqui, S., Rasool, G., e Ramachandran, R. (2022). Transformers in time-series analysis: A tutorial. arxiv 2022. *arXiv preprint arXiv:2205.01138*.

Ba, J. L., Kiros, J. R., e Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., e Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Barbosa, G. N., Bezerra, G. M. G., de Medeiros, D. S., Lopez, M. A., e Mattos, D. M. (2021). Segurança em redes 5g: Oportunidades e desafios em detecção de anomalias e predição de tráfego baseadas em aprendizado de máquina. *Sociedade Brasileira de Computação*.

Bengio, Y., Ducharme, R., e Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231.

Bussab, W. d. O. e Morettin, P. A. (2010). Estatística básica. In *Estatística básica*, pages xvi–540.

Camuñas-Mesa, L. A., Linares-Barranco, B., e Serrano-Gotarredona, T. (2019). Neuromorphic spiking neural networks and their memristor-cmos hardware implementations. *Materials*, 12(17):2745.

Chen, Y.-C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., Cheng, Y., e Liu, J. (2020). Uniter: Universal image-text representation learning. In *European conference on computer vision*, pages 104–120. Springer.

Chowdhary, K. e Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649.

Devlin, J., Chang, M., Lee, K., e Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Evtimov, R., Falli, M., e Maiwald, A. (2020). Anti social online behaviour detection with bert. https://humboldt-wi.github.io/blog/research/information_systems_1920/bert_blog_post/. [Online; accessed 6-September-2023].

Faruqui, M., Tsvetkov, Y., Rastogi, P., e Dyer, C. (2016). Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*.

Ferreira, M. d. S. (2010). *Um estudo wittgensteiniano sobre a identidade das unidades linguísticas*. PhD thesis, PUC-Rio.

Gao, Z., Feng, A., Song, X., e Wu, X. (2019). Target-dependent sentiment classification with bert. *Ieee Access*, 7:154290–154299.

Gardner, M. W. e Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636.

Ghader, H. (2021). An empirical analysis of phrase-based and neural machine translation. *arXiv preprint arXiv:2103.03108*.

Giordani, L., Darú, G., Queiroz, R., Buzinaro, V., Neiva, D. K., Guzman, D. C. F., Henriques, M. J., e Louzada Neto, F. (2022). Automatic identification of fake news in portuguese. *Anais*.

Goodfellow, I., Bengio, Y., e Courville, A. (2016). *Deep learning*. MIT press.

Hakkani-Tür, D. Z., Oflazer, K., e Tür, G. (2002). Statistical morphological disambiguation for agglutinative languages. *Computers and the Humanities*, 36:381–410.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., e Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hartmann, N., Fonseca, E., Shulby, C., Treviso, M., Rodrigues, J., e Aluisio, S. (2017). Portuguese word embeddings: Evaluating on word analogies and natural language tasks.

He, K., Zhang, X., Ren, S., e Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hochreiter, S. e Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Hossin, M. e Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1.

James, G., Witten, D., Hastie, T., Tibshirani, R., et al. (2013). *An introduction to statistical learning*, volume 112. Springer.

Kitaev, N. e Klein, D. (2018). Constituency parsing with a self-attentive encoder. *arXiv preprint arXiv:1805.01052*.

Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S., e Reblitz-Richardson, O. (2020). Captum: A unified and generic model interpretability library for pytorch.

Kriegeskorte, N. e Golan, T. (2019). Neural network models and deep learning. *Current Biology*, 29(7):R231–R236.

Kroeger, P. R. (2019). *Analyzing meaning*. Number 5 in Textbooks in Language Sciences. Language Science Press, Berlin.

Lai, C.-M., Chen, M.-H., Kristiani, E., Verma, V. K., e Yang, C.-T. (2022). Fake news classification based on content level features. *Applied Sciences*, 12(3):1116.

LeCun, Y., Bengio, Y., e Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lenci, A. et al. (2008). Distributional semantics in linguistic and cognitive research. *Italian journal of linguistics*, 20(1):1–31.

Li, L. H., Yatskar, M., Yin, D., Hsieh, C.-J., e Chang, K.-W. (2019). Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*.

Li, Y. e Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, volume 26.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., e Shazeer, N. (2018). Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., e Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lu, J., Batra, D., Parikh, D., e Lee, S. (2019). Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32.

Martin, J. H. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall.

Mikolov, T., Chen, K., Corrado, G., e Dean, J. (2013). Efficient estimation of word representations in vector space.

Miller, G. A. (1985). Wordnet: Dictionary browser'in information in data. In *Proceedings of the First Conference of the UW Centre fot the New Oxford Dictionary. Waterloo, Canada: University of Waterloo.*

Monteiro, R. A., Santos, R. L., Pardo, T. A., De Almeida, T. A., Ruiz, E. E., e Vale, O. A. (2018). Contributions to the study of fake news in portuguese: New corpus and automatic detection results. In *Computational Processing of the Portuguese Language: 13th International Conference, PROPOR 2018, Canela, Brazil, September 24–26, 2018, Proceedings 13*, pages 324–334. Springer.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA.

Niu, Z., Zhong, G., e Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62.

Oshikawa, R., Qian, J., e Wang, W. Y. (2018). A survey on natural language processing for fake news detection. *arXiv preprint arXiv:1811.00770*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., e Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703.

Patterson, J. e Gibson, A. (2017). *Deep learning: A practitioner's approach*. "O'Reilly Media, Inc.".

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., e Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365.

Radford, A., Narasimhan, K., Salimans, T., e Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.

Rodríguez, P., Bautista, M. A., Gonzalez, J., e Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31.

Ruder, S., Peters, M. E., Swayamdipta, S., e Wolf, T. (2019). Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18.

Rumelhart, D. E., Hinton, G. E., e Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5:197–227.

Schofield, A., Magnusson, M., e Mimno, D. (2017). Pulling out the stops: Rethinking stopword removal for topic models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, short papers*, pages 432–436.

Setzer, V. W. (1999). Dado, informação, conhecimento e competência. *DataGrama-Zero Revista de Ciência da Informação, n. 0*, 28.

Sherstinsky, A. (2018). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314.

Su, W., Zhu, X., Cao, Y., Li, B., Lu, L., Wei, F., e Dai, J. (2019). Vl-bert: Pre-training of generic visual-linguistic representations. *arXiv preprint arXiv:1908.08530*.

Sundararajan, M., Taly, A., e Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Sutskever, I., Vinyals, O., e Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Swartz, J., Koziatek, C., Theobald, J., Smith, S., e Iturrate, E. (2017). Creation of a simple natural language processing tool to support an imaging utilization quality dashboard. *International journal of medical informatics*, 101:93–99.

Taylor, W. L. (1953). cloze procedure: A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433.

Tenney, I., Das, D., e Pavlick, E. (2019). Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Vajjala, S., Majumder, B., Gupta, A., e Surana, H. (2020). *Practical natural language processing: A comprehensive guide to building real-world NLP systems*. O'Reilly Media.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., e Polosukhin, I. (2017). Attention is all you need.

Verma, P. K., Agrawal, P., e Prodan, R. (2021). WELFake dataset for fake news detection in text data.

Vosoughi, S., Roy, D., e Aral, S. (2018). The spread of true and false news online. *science*, 359(6380):1146–1151.

Weber, S. H. (2006). Desenvolvimento de nova função densidade de probabilidade para avaliação de regeneração natural. *Universidade Federal do Paraná, Curitiba*, page 87.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt e Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., e Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, H., Zhang, Z., Wu, L., e Wang, C.-J. (2019). The cinderella complex: Word embeddings reveal gender stereotypes in movies and books. *PloS one*, 14(11):e0225385.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., e Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., e Koltun, V. (2021). Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268.

Zhao, R. e Mao, K. (2017). Fuzzy bag-of-words model for document representation. *IEEE transactions on fuzzy systems*, 26(2):794–804.

Zhao, Z., Wang, Y., e Wang, Y. (2022). Multi-level fusion of wav2vec 2.0 and bert for multimodal emotion recognition. *arXiv preprint arXiv:2207.04697*.