CBPF – Centro Brasileiro de Pesquisas Físicas

Dissertação de Mestrado

Detecção Automatizada de Sintomas de COVID-19 em Texto usando Redes Neurais do tipo *Transformer*

Vítor Machado Marques

Orientador
Dr. Clécio Roque De Bom

Rio de Janeiro, RJ
2022

“DETECÇÃO AUTOMATIZADA DE SINTOMAS DE COVID-19 EM TEXTO USANDO REDES NEURAIS DO TIPO TRANSFORMER”

VITOR MACHADO MARQUES

Dissertação de Mestrado Profissional em Física com ênfase em Instrumentação Científica, apresentada no Centro Brasileiro de Pesquisas Físicas do Ministério da Ciência Tecnologia e Inovação. Fazendo parte da banca examinadora os seguintes professores:

Clécio Roque De Bom – Presidente/Orientador/CBPF

Kary Ann Del Carmen Ocaña Guautherot – Coorientadora/LNCC

Eduardo Bezerra da Silva – CEFET-RJ

Marcelo Portes de Albuquerque - CBPF

Rio de Janeiro, 13 de julho de 2022.

Vítor Machado Marques

Detecção Automatizada de Sintomas de COVID-19 em Texto usando Redes Neurais do tipo Transformer

Trabalho apresentado ao Programa de Pós-Graduação no Centro Brasileiro de Pesquisas Físicas como requisito parcial para obtenção do grau de Mestre em Física com Ênfase em Instrumentação Científica.

CBPF – Centro Brasileiro de Pesquisas Físicas

Orientador: Dr. Clécio Roque De Bom
Coorientador: Dra. Kary Ann del Carmen Ocaña Gautherot

Rio de Janeiro, RJ
2022

Machado, Vitor
Detecção Automatizada de Sintomas de COVID-19 em Texto usando Redes Neurais do tipo *Transformer* / Vítor Machado Marques. - 2022
74 f. : il.

Dissertação de Mestrado – CBPF – Centro Brasileiro de Pesquisas Físicas , Rio de Janeiro, RJ, 2022.
Orientador: Dr. Clécio Roque De Bom

1. Classificação de Texto. 2. Redes *Transformer*. 3. COVID-19. 4. BERT. 5. Inteligência Artificial. I. Título
CDU 02:141:005.7

Agradecimentos

Agradeço primeiramente ao meu orientador, Clécio, sem o qual nada disso teria acontecido. Foi graças a ele que eu aprendi o que é fazer pesquisa de verdade e foi ele quem me inspirou a decidir fazer uma carreira acadêmica. Apesar de nem sempre sermos cativados pelo mesmos temas, espero sinceramente ter a oportunidade de trabalhar novamente com ele no futuro, pois trabalhar ao seu lado foi uma experiência ímpar em relação a tudo que eu já vivi na minha vida acadêmica até hoje. Agradeço pela sua transparência ao longo de todo o processo e, principalmente, pelo seu olhar humano para com as pessoas com quem ele trabalha.

Agradeço às pessoas que trabalharam diretamente comigo ao longo do mestrado, especialmente à Kary, Miriam e Gabriel, por toda a ajuda e suporte. Aqui cabe um agradecimento reforçado à Kary, que por diversas vezes olhou para meu trabalho com mais atenção e detalhe do que eu mesmo fui capaz. Tenho certeza que, sem ela, esse trabalho seria muito mais difícil de ser lido e compreendido do que o necessário.

Agradeço especialmente aos meus amigos, que me deram suporte e ajuda nos momentos fáceis e difíceis. Que assistiram todas as minhas apresentações que conseguiram e sempre me impulsionaram para frente, mesmo quando eu mesmo não tinha tanta certeza. Uma lista incompleta, mas bastante significativa, das pessoas que a Física trouxe para a minha vida inclui a Deborah Andrade, Juliana Rosa, João Valeriano, Mariana Casement, Henrique Borotto, Luísa Parucker, Pedro Cintra, João Augusto, Larissa Born, Vitor Dantas, Henrique Lettieri, Felipe Fontenele, Ludmila Lima, Igor Reis, Caio Oliveira, Tabata Luiza, Ranier Menote, Paulo Moises e Lorena Lima. Quanto a amigos de fora da Física, agradeço especialmente ao Mateus Ponciano, Yan Lima, Isabella Lima e Simone Cunha. Cada uma dessas pessoa deu sua importante contribuição para mim nessa jornada, em maior ou menor grau. Mas gostaria de destacar o papel da Juliana, cujos cafés com ela sempre foram momentos de calmaria no meio do turbilhão da vida acadêmica. Momentos em que eu podia apenas respirar e ouvir sobre seus mil projetos e ideias, enquanto meus ombros se tornavam leves sem o peso do dia a dia.

Mas é claro que o maior destaque não poderia ficar com outra pessoa que não a Deborah, cuja companhia e, mais importante, a parceria, me acompanham há anos. Agradeço

por sempre apoiar minhas ideias e projetos, por mais improváveis que pudessem soar. Agradeço por ter compartilhado comigo os momentos bons e por ter me apoiado nos difíceis. Agradeço por sempre ouvir minhas divagações sobre meu trabalho e estudos paralelos, se esforçando para entender onde estava a beleza oculta que eu via. Não tenha dúvida que meu maior agradecimento aqui é a você.

Apesar da lista já estar extensa, cabe ainda um agradecimento a um amigo que a Física não me trouxe, pois já estava comigo muitos anos antes, João Pedro Rabelo. Que nossa amizade, que já dura quase 2 décadas, dure por muitas mais que virão.

Por fim, agradeço ao CBPF por ter me acolhido durante os difíceis anos da pandemia, além de todo o suporte para que este trabalho fosse possível. Agradeço também ao Sérgio e à Prontlife, sem os quais eu não teria tido a oportunidade de viver essa jornada.

Resumo

A presente dissertação apresenta uma metodologia para detecção automatizada de sintomas de COVID-19 em texto, através de técnicas de Aprendizado Profundo, nomeadamente, redes do tipo *Transformer*. Tal metodologia foi desenvolvida inicialmente para identificar, em publicações no *Twitter*, sintomas predefinidos de COVID-19. A metodologia proposta foi avaliada em uma base de dados (doravante, *dataset*) de *tweets* pública, em português brasileiro, de relatos de sintomas de COVID-19 feitos pelos usuários. Ao final, esta foi encapsulada na forma de um *chatbot* no aplicativo de mensagens *Telegram*. Ao longo do trabalho, mineramos o *dataset* para extrair frases com sintomas e construímos redes de Aprendizagem Profunda com alta acurácia para identificação da presença de sintomas. Em paralelo, usamos um processo de Validação Cruzada para avaliar a performance dos resultado e, ao final, interpretamos os resultados usando o algoritmo de *Local Interpretable Model-Agnostic Explanations* (LIME). Montamos um *dataset* final com 4321 *tweets*, dos quais 907 contém ao menos um dos 14 sintomas predefinidos de COVID-19. A rede BERTimbau identificou todos os 14 sintomas reportados em *tweets* em português, identificando cada sintomas com mais de 97% de acurácia e mais de 0.95 de AUC-ROC nos dados de teste. Com a rede treinada em mãos, montamos o *chatbot*, o qual funciona tanto como ferramenta de validação da rede treinada quanto como fonte de novos dados para criação de um novo *dataset*.

Este projeto é um passo no caminho do desenvolvimento de uma futura ferramenta clínica automatizada auxiliar para anamnese, e.g., *chatbots* médicos, para auxiliar avaliações clínicas mais céleres em português.

Palavras Chaves: Redes *Transformer*, Aprendizado Profundo, BERT, Classificação de Texto, COVID-19

Abstract

This dissertation presents a methodology for the automatic detection in written text of COVID-19 symptoms using Machine Learning techniques, more specifically, Transformer networks. The present methodology was initially developed to identify in Twitter's publications previously-defined COVID-19 symptoms. The proposed methodology was evaluated in a public dataset composed by tweets in Brazilian Portuguese reporting COVID-19 symptoms. At the end, it was encapsulated in a chatbot constructed in the messaging app Telegram. During our work, the dataset was mined to extract sentences describing symptoms, contributing to the development of Machine Learning networks with high accuracy for detecting the presence of symptoms. Simultaneously, a Cross-Validation framework was used to evaluate the results' performance and, at the end, the results was interpreted using the Local Interpretable Model-Agnostic Explanations (LIME) algorithm. A final dataset composed of 4321 tweets was built, of which 907 contain at least one of the 14 previously-defined COVID-19 symptoms. The BERTimbau network identified all the 14 symptoms reported in Portuguese tweets, identifying each symptom with an accuracy of more than 97% and AUC-ROC higher than 0.95 in the test dataset. Once in possession of the trained network, the chatbot was built, working as a tool for validating the trained network as well as a source of fresh data to construct a new dataset. This project is a step towards the future development of an automatic subsidiary clinical anamnesis tool, e. g., a medical chatbot for faster clinical evaluation.

Key-Words: Transformer Networks, Deep Learning, BERT, Text Classification, COVID-19

Lista de ilustrações

Figura 2.1 – Esquema representativo do funcionamento de um neurônio artificial. . . 6
Figura 2.2 – Esquema representativo de um exemplo de rede neural artificial. 7
Figura 2.3 – Destaque de parte de uma rede neural artificial para exemplificar o processo de retropropagação. 9
Figura 2.4 – Sobreajuste no treinamento de uma rede neural. 15
Figura 2.5 – Esquema representativo de um neurônio convolucional de tamanho 4x4x3. 17
Figura 2.6 – Esquema representativo de uma camada recorrente. 18
Figura 2.7 – Esquema representativo do funcionamento de uma validação cruzada do tipo *K-Fold* 21
Figura 2.8 – Esquema representativo do significado de cada uma das métricas fundamentais em uma classificação binária. 22
Figura 2.9 – Exemplo de Curvas ROC e seus respectivos valores de AUC-ROC. . . . 24
Figura 3.1 – Esquema representativo do funcionamento do mecanismo de atenção de uma rede do tipo *Transformer* 30
Figura 3.2 – Esquema representativo da arquitetura de uma rede BERT como utilizada no presente trabalho. 32
Figura 3.3 – Exemplo do funcionamento de um tokenizador do tipo *WorkPiece* . . . 33
Figura 3.4 – Esquema representativo de como os *embeddings* de uma rede do tipo BERT são construídos 34
Figura 3.5 – Esquema representativo do funcionamento do mecanismo de atenção unidirecional de uma rede do tipo *GPT* 38
Figura 4.1 – Workflow utilizado para a montagem do *dataset* final 40
Figura 4.2 – Interface da ferramenta *TweetSearcher* 42
Figura 5.1 – Esquema descrevendo os passos a que toda frase do *dataset* final foi submetida para se identificar os sintomas presentes. 46
Figura 5.2 – Gráficos da evolução da perda utilizando-se pesos durante o treinamento. 49
Figura 5.3 – Gráficos da evolução da acurácia utilizando-se pesos durante o treinamento. 50

Figura 5.4 – Gráficos das ROCs para cada sintoma quando o treinamento utilizou pesos no cálculo da perda. 51
Figura 5.5 – Gráficos das Curvas de Precisão-Revocação para cada sintoma quando o treinamento utilizou pesos no cálculo da perda. 52
Figura 5.6 – Gráficos da evolução da perda sem a utilização de pesos durante o treinamento. 53
Figura 5.7 – Gráficos da evolução da acurácia sem a utilização de pesos durante o treinamento. 54
Figura 5.8 – Gráficos das ROCs para cada sintoma quando o treinamento não utilizou pesos no cálculo da perda. 55
Figura 5.9 – Gráficos das Curvas de Precisão-Revocação para cada sintoma quando o treinamento não utilizou pesos no cálculo da perda. 56
Figura 5.10–Evolução da perda durante o treinamento da rede BERTimbau escolhida. 57
Figura 5.11–Evolução da acurácia durante o treinamento da rede BERTimbau escolhida. 58
Figura 5.12–Curvas ROC e de Precisão-Recall para cada sintoma na rede BERTimbau. 59
Figura 5.13–Modelo LIME aplicado à exemplos de erros feitos pela rede. 61
Figura 5.14–Fluxograma do funcionamento do *chatbot*. 64
Figura 5.15–Exemplo da inicialização do *chatbot*. 65
Figura 5.16–Exemplo de resposta do *chatbot* com sintomas identificados. 66
Figura 5.17–Exemplo de como adicionar um sintoma não identificado pelo *chatbot*. . 66
Figura 5.18–Exemplo de como retirar um sintoma identificado erroneamente pelo *chatbot*. 67
Figura 5.19–Exemplo do final de um interação com o *chatbot*. 68

Lista de tabelas

Tabela 1 – Número de sintomas presentes nos *Datasets* inicial e final. 41
Tabela 2 – Comparação entre a quantidade de dados dos *datasets* inicial e final. . 43
Tabela 3 – Meadiana e desvio-padrão da AUC-ROC e da Precisão Média por sintoma. 60

Lista de abreviaturas e siglas

CBPF	Centro Brasileiro de Pesquisas Físicas
IA	Inteligência Artificial
ML	*Machine Learning*
NLP	*Natural Language Processing*
DL	*Deep Learning*
RNA	Rede Neural Artificial
CNN	*Convolutional Neural Network*
ROC	*Receiver Operating Curve*
AUC-ROC	*Area Under ROC Curve*
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
ReLU	*Rectified Linear Unit*
Tanh	Tangente Hiperbólica
AdaGrad	*Adaptative Gradient*
RMSProp	*Root Mean Square Propagation*
Adam	*Adaptative Moment Estimation*
AdamW	Adam *with decoupled weight decay*
TE	*Token Embedding*

PE	*Positional Embedding*
MLM	Modelagem de Linguagem Mascarada
NSP	*Next Sentence Prediction*
BERT	*Bidirectional Encoder Representations from Transformers*
PWFFN	*Position-Wise Feed Forward Network*
GPT	*Generative Pre-trained Transformer*
PBE	*Byte-Pair-Encoding*
CLM	*Causal Language Model*
LIME	*Local Interpretable Model-agnostic Explanations*

Sumário

Lista de ilustrações . . . . . **xi**

Lista de tabelas . . . . . **xiii**

Sumário . . . . . **xvii**

1 INTRODUÇÃO **1**
1.1 Trabalhos Relacionados **2**

2 BREVE INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL **5**
2.1 Redes Neurais **5**
2.2 Função Perda **7**
2.3 Treinamento de uma rede neural **8**
2.4 Otimizadores **10**
2.5 Funções de Ativação **13**
2.6 Problemas no Treinamento **14**
2.6.1 Dissipação do Gradiente 14
2.6.2 Explosão do Gradiente 14
2.6.3 Sobreajuste 15
2.6.4 Subajuste 16
2.7 Principais Tipos de Camada **16**
2.7.1 Camada Densa ou Linear 16
2.7.2 Camada Convolucional 16
2.7.3 Camada Recorrente 16
2.7.4 Normalização de Camada 18
2.7.5 Camada de Retirada 18
2.8 Tipos de Aprendizagem **19**
2.9 Avaliação de performance **20**
2.9.1 Validação Cruzada *K-Fold* 20
2.9.2 Métricas de performance 20
2.9.3 Área sob a Curva Característica de Operação do Receptor (AUC-ROC) . . . 22
2.9.4 Curva de Precisão-Revocação 23
2.10 Problemas multi-rótulo **25**

3 PROCESSAMENTO DE LINGUAGEM NATURAL 27
3.1 Técnicas Inicias 27
3.1.1 Tokenização e Vocabulário 27
3.1.2 *Embedding* 28
3.1.3 Bolsa de Palavras 28
3.1.4 Palavras de Parada 28
3.1.5 *Stemming* 29
3.2 Transformers 29
3.3 Bidirectional Encoder Representations from Transformers (BERT) . 31
3.3.1 Dicionário e Tokenizador 31
3.3.2 *Embeddings* 33
3.3.3 Pré-Treinamento 34
3.3.3.1 Modelagem de Linguagem Mascarada (MLM) 34
3.3.3.2 Predição de Próxima Frase (NSP) 35
3.3.4 Corpo e Cabeça 35
3.4 BERT-Multilingual 36
3.5 BERTimbau 36
3.5.1 Vocabulário e Tokenizador 36
3.5.2 Treinamento 37
3.6 GPT-2 37
3.6.1 Pré-Treinamento 37

4 OBTENÇÃO DOS DADOS 39
4.1 Seleção dos Dados: Palavras-Chave e Sintomas de COVID-19 . . . 40
4.2 Aquisição dos tweets: TweetSearcher 41
4.3 Primeira Análise de Dados: Verificação dos Sintomas de COVID-19 42
4.4 Segunda Análise de Dados: Veriricação de Sintomas em Tweets . . 43
4.5 Terceira Análise de Dados: Balanceando a Frequência dos Sintomas nos Tweets 43

5 PROCESSAMENTO DOS DADOS E RESULTADOS 45
5.1 Processamento dos Dados 45
5.2 Testes de Baseline 47
5.2.1 Resultados com balanceamento na função perda 48
5.2.2 Resultados sem balanceamento na função perda 53
5.3 Resultados do Modelo BERTimbau e Discussão 56
5.4 Local Interpretable Model-Agnostic Explanations (LIME) 61
5.5 Aplicação em Chatbot 62
5.5.1 Funcionamento do *chatbot* 63

6 CONSIDERAÇÕES FINAIS 69

REFERÊNCIAS 71

Capítulo 1

Introdução

A pandemia causada pelo coronavírus (COVID-19) deixou milhões de infectados e mortos no mundo inteiro[1]. A infecção por COVID-19 possui um período de hospitalização longo, de 10 a 13 dias[2], e devido à sua transmissão por via aérea, hospitais se tornaram potenciais focos de risco de novas infecções. Apesar disso, muitas pessoas buscaram hospitais por medo de estarem infectadas, expondo-se ao vírus da COVID-19 no próprio hospital.

Fornecer acesso à informações específicas sobre saúde diretamente para as pessoas usando *chatbots* é uma forma de reduzir a exposição em hospitais e o contato com infectados por COVID-19 e outras doenças infecciosas, reduzindo o fluxo nas unidades de saúde e melhorando as condições de trabalho dos profissionais da saúde [1, 2]. Além disso, o uso de *chatbots* para a realização de auto avaliações automatizadas permite que os profissionais de saúde processem os dados antes de ter contato direto com pacientes.

Estudos iniciais foram direcionados para identificar sintomas em pacientes infectados pelo vírus, principalmente aqueles hospitalizados ou que receberam atendimento de saúde [3]. Entretanto, a maioria dos infectados teve sintomas leves ou foram assintomáticos e não buscaram por atendimento de saúde, apesar da real proporção de assintomáticos não possuir um valor preciso [4, 5].

Metodologias baseadas em redes do tipo *Transformer* [6, 7, 8] são utilizadas em diversas aplicações para processamento de linguagem natural, como classificação em texto [9], tradução [6] e sumarização de textos [10]. Além disso, vários estudos adaptando Aprendizado Profundo para analisar relatos de COVID-19 foram relatados em [11, 12, 13].

Baseado nisso, exploramos o uso das mídias sociais, especificamente do *Twitter*, para estudar sintomas relatados por pessoas que testaram positivo para COVID-19 e por pessoas com sintomas, mas sem resultado de teste.

[1] https://www.who.int/publications/m/item/weekly-operational-update-on-covid-19—8-november-2021

[2] https://www.cdc.gov/coronavirus/2019-ncov/hcp/clinical-guidance-management-patients.html

Nossos objetivos primários foram (i) desenvolver uma rede de aprendizado profundo capaz de identificar, de forma automatizada, sintomas de COVID-19 em texto para ser usada em contexto de autoavaliações; (ii) montar um *chatbot* de autoavaliação capaz de receber informações do paciente e usar a rede neural em tempo real para identificar sintomas relacionados a COVID-19.

1.1 Trabalhos Relacionados

Existem diversos trabalhos que usam modelos computacionais para auxiliar na área médica, inclusive no combate da pandemia de COVID-19. Entre esses trabalhos, destacamos os que usam de dados do *Twitter* para treinar modelos de classificação de texto relacionados à COVID-19. Dentre esses trabalhos, podemos citar Sarker *et al.*, 2021 [14], que analisa o espectro de sintomas de COVID-19 auto-relatados por usuários do *Twitter*, com o objetivo de complementar a lista de sintomas identificados em ambiente clínico. Esse foi o primeiro trabalho, pelo nosso conhecimento, que focou na extração de sintomas de COVID-19 de mídias sociais públicas.

Seguindo a linha de problemas de classificação com dados do *Twitter*, Kumar *et al.*, 2021 [15], descreve abordagens para classificar *tweets* contendo sintomas de COVID-19 em três classes (auto-relatados, relatos não-presenciais e menções na literatura/mídia) nas tarefas compartilhadas do *Social Media Mining for Health Applications* (SMM4H) em 2021. Foram implementados diferentes modelos para essa tarefa de classificação de texto, obtendo vários resultados com altos índices de acerto, o que sugere ser um problema que possui diversas soluções com alta performance.

Valdes *et al.*, 2021 [16], decreve a classificação automática de publicações no *Twitter* relacionados à COVID-19. Seu trabalho foi focado em resolver um problema de classificação binário: identificar *tweets* com auto-relatos de potenciais casos de COVID-19, e classificar *tweets* contendo sintomas de COVID-19. Baseado nos resultados obtidos, os autores concluíram que um modelo treinado com dados específicos do domínio, no caso, *tweets* relatando casos ou sintomas de COVID-19, poderia superar um modelo treinado com uma quantidade muito mais significatica de dados de domínio geral, não relacionados a essa questão específica.

Em [17], Luo *et al.* apresentam um sistema que busca distinguir automaticamente *tweets* entre auto-relatos de potenciais casos de COVID-19 daqueles que não o são. Foi proposto um conjunto de diferentes modelos computacionais que balanceia o resultado de cada modelo para chegar a uma resposta final.

Por fim, o estudo em [18] objetivou detectar e caracterizar conversas (no *Twitter*) geradas por usuários que poderiam estar associadas a sintomas relacionados à COVID-19, experiência com acesso à testagem, e menções de recuperações de doenças usando uma abordagem de Aprendizado de Máquina não-supervisionado.

Tendo em vista nosso objetivo de usar um *chatbot* como ferramenta de automatização para autoavaliações, existe uma preocupação sobre a disposição das pessoas em interagir com o *chatbot*. De acordo com [19], a maioria dos usuários de internet seriam receptivos em usar *chatbots* de saúde, apesar da hesitação em relação a essa tecnologia poder comprometer o engajamento dos usuários. Além disso, as perspectivas, motivações e capacidades dos pacientes devem ser levadas em conta e ouvidas durante o desenvolvimento de *chatbots* de saúde para aumentar sua frequência de uso e qualidade.

No presente trabalho, usamos redes *Transformer* treinadas de forma supervisionada para caracterizar sintomas auto-relatados e menções a recuperações relacionados à COVID-19 no *Twitter* brasileiro. Muitos usuários reportaram sintomas relacionadas à COVID-19, mas não foram testados para confirmar suas suspeitas. Futuros estudos podem continuar a explorar a utilidade de abordagens de vigilância da informação para estimar a severidade da COVID-19, extendendo a pesquisa para mais bases de dados em texto e áudio, além da possibilidade de utilizar modelos de Inteligência Artificial mais refinados.

Capítulo 2

Breve introdução à Inteligência Artificial

O termo Inteligência Artificial (doravante IA) se refere, em linhas gerais, à criação de agentes racionais artificiais. Em outras palavras, busca-se dotar um programa de computador da capacidade de tomar decisões e/ou realizar ações baseadas em uma busca pelo melhor resultado possível [20]. Uma das formas mais bem sucedidas da atualidade em uma vasta variedade de tarefas é o chamado Aprendizado de Máquina (doravante ML, de *Machine Learning*), que usa modelos que são construídos a partir dos dados [21].

Aprendizagem Profunda (doravante DL, de *Deep Learning*) é um subcampo de Aprendizado de Máquina que utiliza um algoritmo de aprendizagem inspirado no cérebro humano para processar padrões, as chamadas redes neurais. O algoritmo é baseado em estruturas chamadas de neurônios artificiais, em clara analogia à sua inspiração, que são ordenados em camadas até formar uma rede neural. Uma rede neural como um todo possui uma profundidade, dada pelo número de camadas de neurônios, e cada camada possui sua largura, referente à quantidade de neurônios presentes nela. Redes neurais usam uma combinação de transformações lineares e não-lineares para processar dados e aprender seus padrões [22, 23].

2.1 Redes Neurais

O conceito fundamental de uma rede neural é a proposta de representação matemática do neurônio. O funcionamento de um único neurônio artificial, que pode ser visualizado na Figura 2.1, consiste em receber os dados de entrada $(x_1, x_2, ..., x_n)$, ponderá-los por seus respectivos pesos $(w_1, w_2, ..., w_n)$, somar o resultado a um viés (b), aplicar uma função de ativação (f) e, por fim, passar o resultado para a próxima camada. Os pesos, como o próprio nome sugere, servem para dar uma importância relativa diferente para cada

entrada. O grande sucesso das redes neurais dentro do campo da inteligência artificial se dá pela sua capacidade de aprender qual o melhor peso usar para cada entrada, processo que veremos em futuras seções. O viés também é um parâmetro aprendível que visa dar maior liberdade ao sistema. Até então, todo o processo descrito é uma simples função linear tendo os dados de entrada como variáveis. Mesmo em uma rede neural com diversos neurônios, continuaríamos com uma função linear, incapaz de realizar aprendizados que exigissem algum caráter não-linear. Acrescentar um caráter não-linear aos neurônios é a tarefa da função de ativação, que é aplicada sobre o somatório feito e gera o sinal de saída do neurônio artificial. Existem uma infinidade de funções de ativação diferentes, cada uma melhor adaptada a diferentes tipos de problemas.

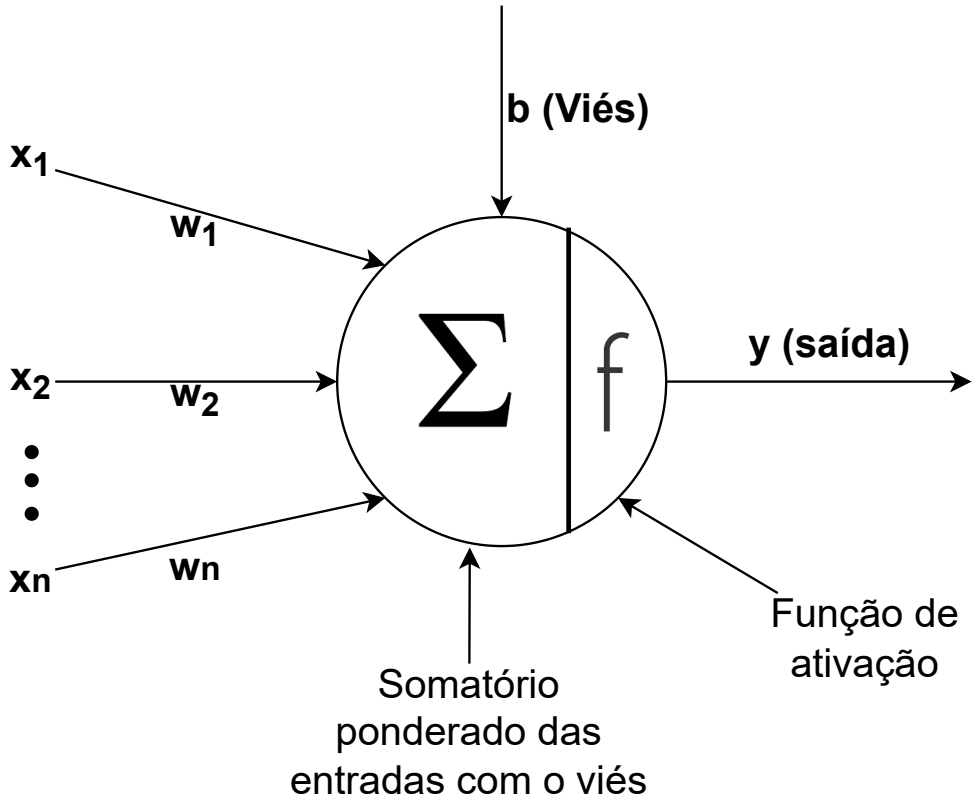


Figura 2.1 – Esquema representativo do funcionamento de um neurônio artificial. As entradas $(x_1, x_2, ..., x_n)$ são ponderadas por seus respectivos pesos $(w_1, w_2, ..., w_n)$ e somadas ao valor de um viés (b). Por fim, uma função de ativação é aplicada ao resultado do somatório, gerando a saída (y) do neurônio.

Os neurônios artificiais são divididos em camadas, que determinam a ordem de processamento das informações, como pode ser visualizado na Figura 2.2. Nela, temos um exemplo de rede neural que recebe duas entradas (x_1, x_2) e devolve duas saídas (y_1, y_2) após passar por quatro camadas de neurônios. Entre cada camada, as saídas da camada anterior entram no próximo neurônio após serem ponderadas pelos seus respectivos pesos. Em seguida, é adicionado o viés do neurônio, e computada a função de ativação, cuja saída servirá como entrada para a próxima camada. O processo se repete até alcançarmos a

última camada, que retorna a saída da rede neural.

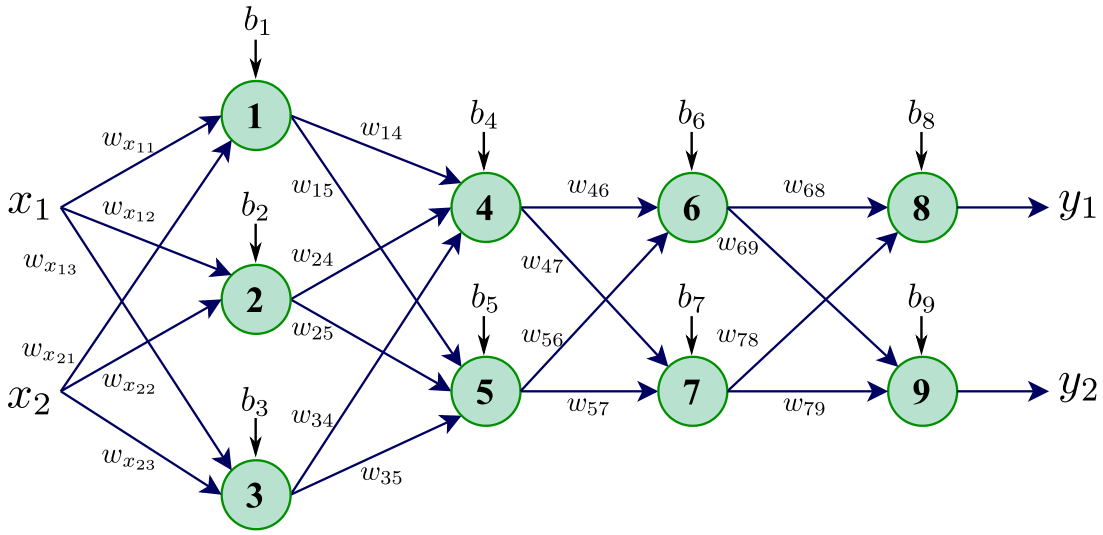


Figura 2.2 – Esquema representativo de um exemplo de rede neural artificial. As entradas (x_1, x_2) são processadas por nove neurônios distribuídos em quatro camadas, gerando duas saídas (y_1, y_2).

2.2 Função Perda

Até então analisamos como uma rede neural produz uma saída, mas falta sabermos qual a qualidade da saída gerada. Existem diversas formas de comparar o resultado obtido em uma rede neural com o valor esperado, e cabe a alguém definir como essa comparação será feita. O processo de comparação é feito pela chamada função perda (do inglês, *loss function*), que é uma função que recebe os resultados e o valor esperado e retorna um número, denominado perda (do inglês, *loss*).

O processo de treinamento de uma rede neural se resume em buscar reduzir o valor da perda, como será mostrado nas próximas seções. Isso demonstra qual a importânica de se escolher uma função perda alinhada com os objetivos da rede neural, porque toda a otimização será feita a partir dela. Caso a função perda e o objetivo da construção da rede não estejam alinhados, é possível que a rede aprenda muito bem (ou seja, reduza bastante o valor da perda), mas tenha péssimos resultados em sua aplicação prática.

Um ótimo exemplo é treinar uma rede neural para detectar operações fraudulentas em cartão de crédito sem usar nenhum tipo de técnica de balanceamento de dados no processo. Para isso, pode-se usar uma função perda que simplesmente tira o módulo da diferença entre a saída da rede (entre 0 e 1) e o valor real (0 ou 1). Como as operações fraudulentas são raras se comparadas com as operações rotineiras, uma forma simples da rede reduzir enormemente o valor de sua perda é simplesmente classificar todas as operações como rotineiras, e nunca tentar apontar uma fraude. O valor da perda será minímo, mas a rede será completamente inútil para atingir o objetivo buscado.

Uma das funções perda mais utilizadas para problemas de classificação é a chamada entropia cruzada, que faz uma comparação entre a distribuição de probabilidades verdadeira e a prevista pelo modelo. Nesse contexto, estamos realizando uma classificação do tipo multi-classe, quando há várias possíveis classes, mas cada dado pertence à apenas uma delas. Nesse caso, a distribuição de probabilidades verdadeira será 1 para a classe correta e 0 para as demais.

Em uma rede neural de classificação multi-classe, a saída da rede deve ser um vetor ($\mathbf{z}$) com uma entrada para cada possível classe. Para se obter a distribuição de probabilidades a partir da saída do modelo, podemos aplicar a chamada função *softmax*:

$$\sigma(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=0}^{C} e^{z_j}} \tag{2.1}$$

onde $\sigma(\mathbf{z}_i)$ é a saída da função *softmax* na entrada i do vetor $\mathbf{z}$. Podemos observar que o somatório da saída da função *softmax* em todas as entrada de $\mathbf{z}$ nos retornará 1, logo podemos usar suas saídas como uma distribuição de probabilidades para o resultado da nossa rede neural.

Usando y_j como o valor verdadeiro da função distribuição de probabilidade para a classe j (0 ou 1) e $\hat{y}_j$ como o valor da previsão da rede sobre a classe j (entre 0 e 1), a função perda entropia cruzada se escreve como

$$EC_{perda} = -\sum_{j=1}^{C} y_j \cdot log(\hat{y}_j) \tag{2.2}$$

Para o caso em que queremos fazer uma classificação binária, ou seja, quando há apenas duas classes, a entropia cruzada se reduz à chamada entropia cruzada binária:

$$ECB_{perda} = -(y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})) \tag{2.3}$$

onde y é o valor real do dado (0 ou 1) e $\hat{y}$ é o valor previsto pelo modelo (entre 0 e 1). Como o valor de y é sempre 1 ou 0, apenas parte da equação é realmente computada. Uma classificação sem erros geraria um valor igual a 0 na função perda.

2.3 Treinamento de uma rede neural

O treinamento de uma rede neural ocorre em duas etapas: (i), um lote (do inglês, *batch*) de dados é inserido na rede, que o processa e entrega uma saída, e (ii), o resultado obtido é comparado com o valor esperado através da função perda e usado para alterar os pesos e viéses da rede em um processo chamado de retropropagação (do inglês, *backpropagation*).

Como mostrado na Figura 2.2, os dados (x_1, x_2) entram pelo lado esquerdo da rede neural, são processados pelos neurônios e geram as saídas (y_1, y_2). Esse processo é cha-

mado de propagação direta e ocorre em lotes de dados. As saídas de cada dado de entrada são comparadas com seus valores esperados através da função perda previamente escolhida. O tamanho do lote de dados é um hiperparâmetro que pode ser otimizado para melhorar o treinamento de uma rede.

Usando o resultado da função perda, é iniciado o processo de retropropagação, que é uma forma de levar a informação final sobre a função perda para a rede. Esse processo é feito pelo algoritmo de minimização, como por exemplo o método do gradiente descendente, que calcula o gradiente da função perda com respeito a todos os pesos e viéses da rede neural. Por fim, o oposto do gradiente obtido (gradiente com o sinal negativo) é multiplicado por uma taxa de aprendizado e o resultado é usado para atualizar todos os pesos e viéses.

Para ilustrar esse processo, digamos que desejamos atualizar o valor do peso w_{68} da Figura 2.2. Para isso, façamos o destaque apresentado na Figura 2.3. Para facilitar o entendimento dos cálculos, representaremos a saída de cada neurônio i como y'_i. Assim, w'_{68}, que é w_{68} atualizado pelo gradiente descendente, será dado por:

$$w'_{68} = w_{68} - \varepsilon \cdot \frac{\partial L(y_1, y_2)}{\partial w_{68}} = w_{68} - \varepsilon \cdot \frac{\partial L(y_1, y_2)}{\partial y_1} \frac{\partial y_1}{\partial w_{68}} \tag{2.4}$$

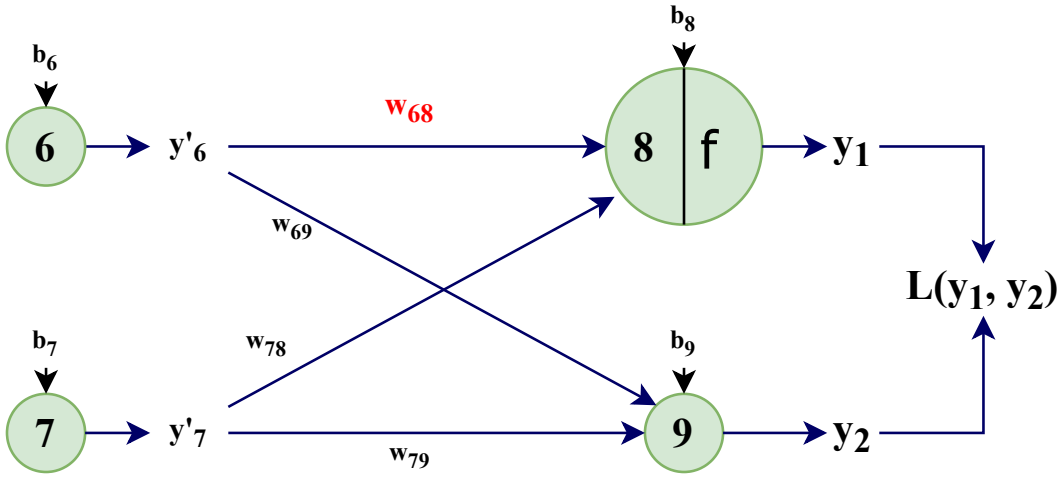


Figura 2.3 – Destaque da figura anterior para facilitar o entendimento do processo de retropropagação. Apenas as duas últimas camadas de neurônios e as saídas da rede neural artificial são mostradas. Ao final, as saídas são usadas para o cálculo da função perda $L(y_1, y_2)$.

onde ε é a chamada taxa de aprendizado, definida previamente. Para fazermos essa derivada parcial, cabe representarmos y_1 em termos das entradas do neurônio 8 e de sua função de ativação. Assim, temos que

$$y_1 = f(w_{68} \cdot y'_6 + w_{78} \cdot y'_7 + b_8). \tag{2.5}$$

Substituindo essa representação de y_1 na equação 2.4 e aplicando a regra da cadeia, obtemos

$$w'_{68} = w_{68} - \varepsilon \cdot \frac{\partial L(y_1, y_2)}{\partial y_1} f'(w_{68} \cdot y'_6 + w_{78} \cdot y'_7 + b_8) \cdot y'_6 \tag{2.6}$$

onde $f'()$ é a derivada da função de ativação. Logo, o processo de retropropagação depende diretamente das variáveis:

1. Taxa de Aprendizado;
2. Derivada da Função Perda;
3. Derivada das Funções de Ativação;
4. Saídas dos Neurônios Artificiais.

Esse processo de propagação direta e retropropagação ocorre diversas vezes com o objetivo de fazer a rede chegar a um mínimo local da função perda que seja aceitável para a utilização desejada. Cada vez que esse processo ocorre é chamado de passo. Quando todos os dados do conjunto de treinamento passam por esse processo, dizemos que foi computada uma época. A quantidade de passos ou épocas utilizados no treinamento de uma rede é também um hiperparâmetro de otimização no treinamento.

Ao contrário do que pode parecer intuitivo, treinar com mais épocas nem sempre gera melhores resultados, podendo inclusive gerar resultados piores. Essa discussão será feita com mais detalhes na seção 2.6.

2.4 Otimizadores

Podem ser utilizados otimizadores como uma forma de complementar o algoritmo de gradiente descente padrão no momento de treinamento da rede. Seu objetivo é conseguir um treinamento mais ágil, mais estável ou mesmo evitar mínimos locais não ótimos.

O principal foco dos otimizadores é no hiperparâmetro taxa de aprendizado. Esse é um hiperparâmetro difícil de se escolher, pois valores muito baixos farão com que o treinamento seja demasiado lento ou que o modelo fique preso em mínimos locais não ótimos no espaço de parâmetros, enquanto que valores muito grandes podem impedir que o modelo alcance mínimos no espaço de parâmetros.

O otimizador gradiente adaptativo (doravante AdaGrad, de *adaptative gradient*) objetiva reescalar a taxa de aprendizagem para cada dimensão do espaço de parâmetros em relação ao somatório dos gradientes em iterações anteriores. Assim, dimensões cujo gradiente é muito alto tendem a ter sua taxa de aprendizagem diminuída com o tempo, enquanto que dimensões com gradiente baixo não são tão afetadas [24].

Para efeitos de comparação, vejamos a forma geral da taxa de aprendizagem original da seguinte forma:

$$w_i^{t+1} = w_i^t - \varepsilon \cdot g_t \tag{2.7}$$

onde $g_t = \nabla_i L(w_i^t)$, ou seja, a dimensão do gradiente da função perda associada ao peso w_i no passo t. A maior limitação dessa equação é o uso da mesma taxa de aprendizagem para todos os pesos a serem atualizados. O otimizador AdaGrad busca resolver isso reescalando a taxa de aprendizagem em cada dimensão através da equação

$$w_i^{t+1} = w_i^t - \frac{\varepsilon}{\sqrt{\eta + \sum_{\tau=1}^{t} g_\tau^2}} \cdot g_t \tag{2.8}$$

onde η é um pequeno valor usado para impedir divisões por zero. No otimizador AdaGrad, todas os valores anteriores do gradiente são levados em conta no momento de atualizar os pesos. Como esses valores estão sendo sempre somado a cada passo, a taxa de aprendizagem real diminui com o passar das épocas.

Porém, essa diminuição na taxa de aprendizado pode acabar sendo exagerada e culminar em um congelamento do aprendizado, com a taxa de aprendizagem se aproximando do zero.

A propagação de raiz quadrática média (doravante RMSProp, de *Root Mean Square Propagation*) é outro otimizador que adapta a taxa de aprendizado e surgiu como uma forma de melhorar o AdaGrad. Ao invés de usar a soma acumulada dos gradientes ao quadrado, ele usa a média móvel exponencial do gradiente com dois termos. Assim, um maior peso é dado para o termo mais recente, ou seja, para o gradiente do último passo dado [1]. A equação que representa a média móvel exponencial do gradiente ($E[g^2]_t$) comumente usada é

$$E[g^2]_t = v_t = \beta \cdot v_{t-1} + (1 - \beta) g_t^2 \tag{2.9}$$

onde β é um parâmetro entre 0 e 1, cujo valor padrão nesse tipo de aplicação é 0.9. A inicialização de v_t costuma ser no zero. Assim, os pesos são atualizados pelo RMSProp de acordo com a equação

$$w_i^{t+1} = w_i^t - \frac{\varepsilon}{\sqrt{\eta + v_t}} \cdot g_t \tag{2.10}$$

Ao invés de usar simplesmente o gradiente descendente, podemos utilizar a técnica conhecida como gradiente descendente com momento, que substitui o gradiente por uma ponderação de gradientes, chamada de momento [25]. Essa ponderação é, na verdade, uma média móvel exponencial do gradiente atual e dos anteriores

$$m_t = \alpha \cdot m_{t-1} + (1 - \alpha) g_t \tag{2.11}$$

[1] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

onde α é um parâmetro entre 0 e 1, cujo valor padrão nesse tipo de aplicação é 0.9. A inicialização de m_t costuma ser no zero. Assim, os pesos são atualizados pelo momento de acordo com a equação

$$w_i^{t+1} = w_i^t - \varepsilon \cdot m_t \tag{2.12}$$

Como uma forma de unir os benefícios do Gradiente Descendente com Momento e do RMSProp, foi criado o otimizador de estimativa de momento adaptativa (doravante Adam, de *Adaptative Moment Estimation*), sendo um otimizador que usa taxas de aprendizagem reescaladas para cada parâmetro ao mesmo tempo que conta com o uso de momentos [26].

$$w_i^{t+1} = w_i^t - \frac{\varepsilon}{\hat{v}_t} \cdot \hat{m}_t \tag{2.13}$$

onde

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.14}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.15}$$

são correções para m_t e v_t, respectivamente, e

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) g_t \tag{2.16}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.17}$$

Os valores padrões sugeridos pelo artigo original são de $\beta_1 = 0.9$, $\beta_2 = 0.999$ e $\varepsilon = 0.002$. As inicializações de m_t e de v_t são, por padrão, zero.

Por fim, o otimizador Adam com decaimento de peso desacoplado (doravante AdamW, de Adam *with decoupled weight decay*) surgiu como uma forma de superar problemas de generalização enfrentados pelo otimizador Adam. Generalização no contexto de ML se refere à capacidade de um modelo em gerar bons resultados em dados nunca antes vistos. Ao adicionar explicitamente o valor do próprio peso na expressão que o atualiza, o valor do peso entra explicitamente no cálculo da função perda, fazendo com que seja interessante reduzir o valor dos pesos. Pesos menores tem menor chance de gerarem sobreajuste, o que por si só é uma vantagem.

$$w_i^{t+1} = w_i^t - \varepsilon \cdot \left(\frac{\hat{m}_t}{\hat{v}_t} + \gamma \cdot w_i^t \right) \tag{2.18}$$

onde γ é o hiperparâmetro de decaimento de peso e possui valor padrão igual a 0.01.

Os autores mostraram experimentalmente que o otimizador AdamW é capaz de produzir resultados melhores para a perda e modelos com maior capacidade de generalização do que o otimizador Adam [27].

2.5 Funções de Ativação

Como vimos anteriormente, as funções de ativação são as responsáveis pela capacidade das redes neurais de aprender padrões não-lineares. Uma característica desejada para que uma função seja utilizada como função de ativação é que ela e sua derivada estejam bem definidas e sejam suaves em todo o domínio dos reais. Existem exceções à essa regra, que costumam ser resolvidas usando os valores à esquerda ou à direita nos pontos problemáticos.

A função linear é a mais simples das funções, sendo utilizada na última camada principalmente em regressões, quando o resultado final é um número real sem restrições. Nesse caso, a saída do neurônio é simplesmente uma combinação linear das suas entradas, podendo se dizer também que trata-se simplesmente de um neurônio sem função de ativação.

Uma das funções de ativação há mais tempo usada é a chamada Unidade Linear Retificada, comumente chamada por ReLU (de *Rectified Linear Unit*). A função de ativação ReLU é definida como:

$$f(x) = max(0,\ x) \tag{2.19}$$

Seu efeito, na prática, é selecionar neurônios para desativar, ou seja, para não participarem do cálculo da rede. Todos os neurônios que produzem saída negativa têm seus resultados mapeados para zero, que seria o equivalente a desativar sua participação na rede.

A função sigmóide é comumente utilizada como função de ativação em diversas arquiteturas de redes neurais. Além de seu caráter não-linear, essencial para a capacidade de generalização das redes neurais, outra importante característica dessa função é sua imagem, que se restringe ao intervalo aberto (0, 1), como pode ser observado a partir de sua definição:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.20}$$

Essa restrição em sua imagem e o formato de sua derivada contribuem para impedir a explosão do gradiente durante o treinamento, assunto que será tratado na próxima seção. Outra vantagem do uso da função sigmóide é que sua derivada pode ser expressa em termos da própria função, facilitando os cálculos realizados durante o treinamento.

$$f'(x) = f(x) \cdot (1 - f(x)) \tag{2.21}$$

A função sigmóide pode ser usada como a última função de ativação em problemas de classificação, já que uma rede densa terminada em uma função sigmóide é equivalente a um regressor logístico, resultando em uma probabilidade para a classificação de uma classe. A forma de sua derivada contribui para esse processo, já que possui seu máximo

centrado no zero, o que gera uma tendência a empurrar os resultados para os extremos do intervalo.

Apesar de todas as vantagens listadas, também existem desvantagens em seu uso. A principal desvantagem é que ela têm a tendência de gerar a dissipação do gradiente para valores muito altos ou baixos, fazendo com que o processo de aprendizado se torne lento ou praticamente seja interrompido nas primeiras camadas.

Por fim, a tangente hiperbólica como função de ativação possui a maior parte das vantagens da função sigmóide, mas seu uso ao final de uma rede densa não possui a mesma equivalência com um classificador logístico, impedindo seu uso como fonte de probabilidades. Apesar disso, seu gráfico é centrado no zero, oferecendo uma simetria extra quando comparada à sigmóide. Por outro lado, sua derivada converge para zero mais rapidamente nos extremos, fazendo com que a dissipação do gradiente seja um problema mais sério.

2.6 Problemas no Treinamento

2.6.1 Dissipação do Gradiente

Como foi visto na seção 2.3, o processo de treinamento de uma rede neural depende diretamente do cálculo de gradientes. O problema da dissipação do gradiente ocorre quando o gradiente para alguns pesos (normalmente, os pesos das primeiras camadas) fica próximo à zero, fazendo com que a atualização de seus valores seja demasiadamente lenta. Esse processo ocorre devido à forma que a equação 2.6 toma quando analizada para pesos das primeiras camadas. A forma geral ficaria sendo:

$$w' = w - \varepsilon \cdot \frac{\partial L}{\partial y_i} f'(z_1) \cdot w_2 f'(z_2) \cdot w_3 f'(z_3) \cdot ... \cdot w_n f'(z_n) \tag{2.22}$$

sendo os w_i pesos da rede neural, os z_i saídas de neurônios e os f'() a derivada da função de ativação. Quando a derivada da função de ativação possui valores baixos, como é o caso da função sigmóide, o produto de vários valores baixos tendem a fazer o gradiente se aproximar do zero. Um gradiente próximo de zero nas primeiras camadas faz com que seu processo de aprendizado seja muito mais lento do que o das últimas camadas, podendo fazer com que o processo de treinamento como um todo seja muito lento e mais épocas sejam necessárias, aumentando o tempo gasto e o custo computacional do processo.

2.6.2 Explosão do Gradiente

A explosão do gradiente é o problema contrário do apresentado na dissipação do gradiente. O problema ocorre quando a derivada da função de ativação tende a assumir valores maiores do que 1, fazendo com que o produto desses valores fique muito alto para

as primeiras camadas da rede. Assim, cada passo no processo de aprendizado fica grande demais e as primeiras camadas têm dificuldade de convergir para um mínimo.

Uma forma de mitigar tanto esse problema quanto o anterior é o uso de diferentes funções de ativação nas camadas intermediárias, de forma que os produtos de suas derivadas se equilibrem e não gerem nem uma dissipação e nem uma explosão do gradiente nas primeiras camadas.

2.6.3 Sobreajuste

O sobreajuste é um dos principais problemas no momento de treinar redes neurais. Sobreajuste consiste em treinar excessivamente uma rede, de forma que ela fique específica demais para os dados de treino, perdendo a capacidade de generalizar para outros dados, mesmo que originados do mesmo conjunto de dados.

Uma forma de identificar o sobreajuste é através do gráfico da função perda nos dados de treino e de validação durante o processo de treinamento. Caso o gráfico da perda do treino continue decaindo enquanto a perda da validação se mantém estagnada (ou mesmo sobe), é um forte indicativo de sobreajuste, como pode ser observado na Figura 2.4.

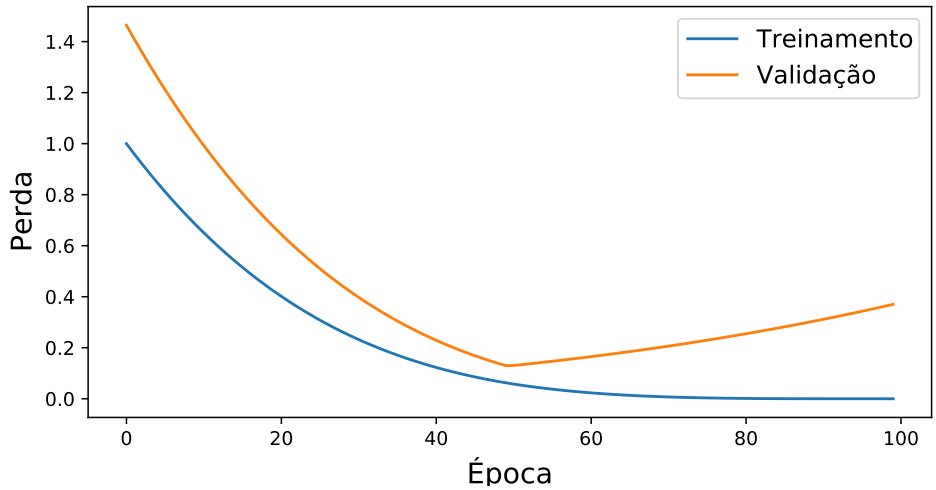


Figura 2.4 – Exemplo de sobreajuste no treinamento de uma rede neural artificial.

As técnicas mais comuns para reduzir esse efeito são:

- Uso de uma "parada antecipada", que consiste em utilizar a época em que a perda dos dados de validação é mínima ao invés de usar a última época do treinamento;
- Redução da complexidade ou profundidade da rede neural utilizada;
- Fazer uma melhor separação dos dados para que os conjuntos de treino e validação possuam distribuições mais parecidas.

2.6.4 Subajuste

Subajuste é o problema contrário do sobreajuste, e consiste na rede neural ser incapaz de aprender o suficiente nos dados de treino a ponto de ter bons resultados. As técnicas mais comuns para reduzir esse efeito são aumentar as épocas de treinamento, aumentar a complexidade da rede neural, e melhorar o pré-processamento dos dados.

2.7 Principais Tipos de Camada

2.7.1 Camada Densa ou Linear

As camadas densas (ou lineares) são o tipo mais comum e podem ser visualizadas na Figura 2.2, onde todas as camadas são densas. Camadas densas são as camadas cujos neurônios recebem como entrada todas as saídas da camada anterior. São também chamadas de lineares porque, caso não se use uma função de ativação, elas retornam apenas uma combinação linear de suas entradas.

2.7.2 Camada Convolucional

O surgimento das camadas convolucionais foi muito importante para o tratamento de imagens em DL [28]. A ideia por trás desse tipo de camada é reconhecer padrões em imagens. Podem ser linhas, curvas, bordas, etc.

Um neurônio em uma camada convolucional funciona como um filtro, onde cada número é um peso aprendível do neurônio. Um típico neurônio convolucional de tamanho 4x4 e com 3 canais (profundidade) pode ser vista na Figura 2.5. Esse neurônio poderia processar uma imagem com 3 canais de cor como, por exemplo, o padrão RGB. O neurônio passa ao longo de toda a imagem, enxergando uma parte de cada vez e buscando o padrão associado aos seus pesos. Para cada posição do filtro, multiplicamos um a um as entradas da imagem e os pesos do neurônio, somamos os resultados e chegamos a um número associado àquela parte da imagem. Ao passar por toda a imagem, o filtro formará uma nova imagem com os resultados desse processo, muitas vezes menor do que a imagem original. Essa é a saída da camada convolucional que será entregue à camada seguinte, que pode ser outra camada convolucional ou uma camada de outro tipo.

Assim como nas camadas densas, não é incomum se ter vários neurônios em uma mesma camada convolucional, cada um aprendendo um padrão diferente e enviando uma saída diferente para a camada seguinte, desde que todos tenham a mesma dimensão.

2.7.3 Camada Recorrente

Redes neurais com camadas recorrentes já foram o estado da arte em problemas de processamento de linguagem natural. A principal característica desse tipo de camada é

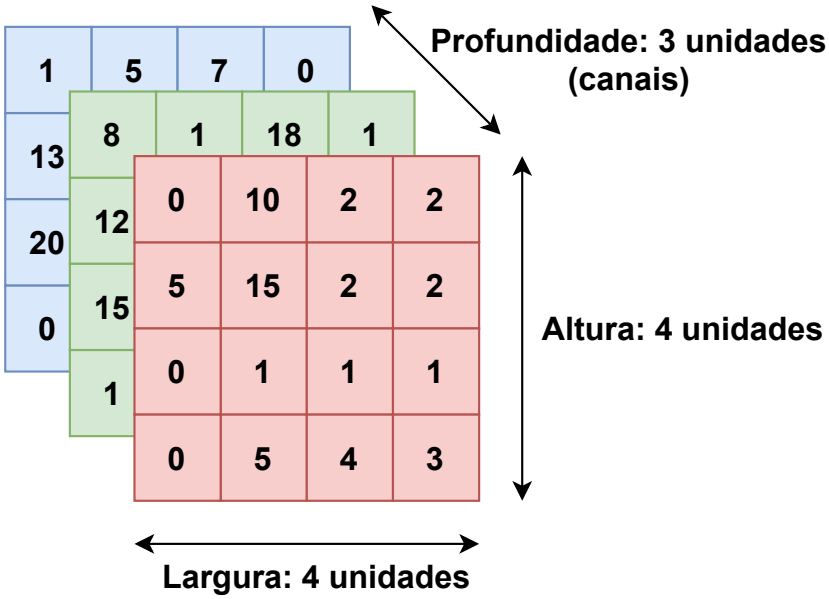


Figura 2.5 – Esquema representativo das entradas de um neurônio convolucional de tamanho 4x4x3. Todos os números que aparecem na imagem são entradas aprendíveis do neurônio.

que as entradas não são processadas em paralelo, mas são processadas uma após a outra, em sequência. Isso ocorre porque a camada recorrente usa a saída de uma entrada no processamento da entrada seguinte. Infelizmente, essa sequência linear de informação cria um gargalo quando se deseja acelerar os cálculos da rede neural, pois a dependência do processamento de uma entrada do processamento da entrada anterior dificulta a paralelização dos cálculos [22].

Uma típica camada recorrente pode ser vista na Figura 2.6. Do lado esquerdo, temos uma representação resumida de seu funcionamento, enquanto o lado direito mostra ela em uma forma mais explicativa. Nesse camada, a entrada x_0 é multiplicada pelos pesos U, passa pelo neurônio A e sua função de ativação, é multiplicado pelos pesos V e se torna a saída h_0. Ou seja, podemos escrever:

$$h_0 = V \cdot A(U \cdot x_0) \tag{2.23}$$

Enquanto isso, o valor $A(U \cdot x_0)$ é copiado e multiplicado pelos pesos W, ficando o valor final guardado. Quando a próxima entrada (x_1) for processada, o valor guardado é adicionado antes da aplicação da função de ativação. Em outras palavras, a entrada recebida pelo neurônio não será apenas $U \cdot x_1$, mas $U \cdot x_1 + W \cdot A(U \cdot x_0)$. Dessa forma, informação sobre a entrada anterior é usada para processar também a entrada atual.

Essa forma de processar informação, alimentando a camada com uma entrada de cada vez e usando informação anterior para avaliar a próxima entrada faz muito sentido quando estamos processando texto. Por exemplo, podemos inserir uma palavra de cada vez na

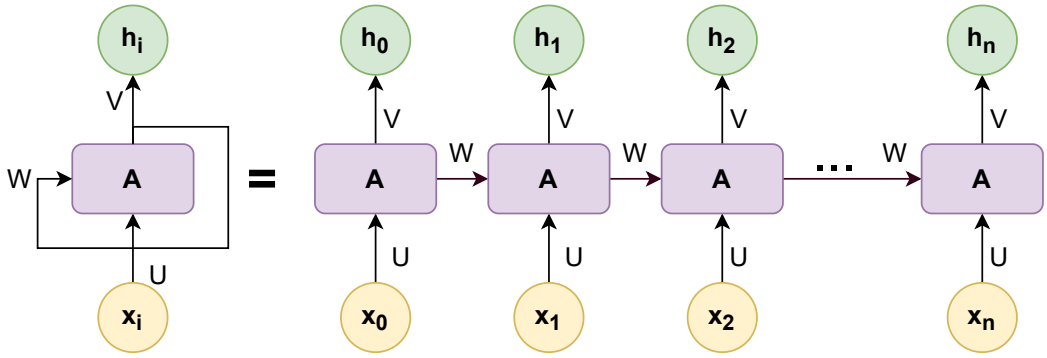


Figura 2.6 – Esquema representativo do funcionamento de uma camada recorrente em uma rede neural artificial. U, V, W são conjuntos de pesos aprendíveis da camada.

camada e usar informação sobre as palavras que aparecem antes na frase para processar as palavras que aparecem depois.

2.7.4 Normalização de Camada

Um dos desafios no processo de aprendizado de redes neurais é que o gradiente da função perda em relação aos pesos de uma camada é altamente dependente dos resultados dos neurônios da camada anterior, podendo gerar fortes correlações indesejadas. A Normalização de Camada é uma técnica que visa reduzir esse problema ao normalizar as entradas em uma camada, diminuindo a correlação com cada saída específica e, consequentemente, reduzindo o tempo de treinamento e o gasto computacional [29].

Logo, cada saída da camada l (a_i^l) é regularizada da seguinte forma:

$$\bar{a}_i^l = \frac{g_i^l}{\sigma^l} \cdot (a_i^l - \mu^l) \tag{2.24}$$

onde g_i^l é simplesmente um parâmetro de escala aprendível e μ^l e σ^l são a média e o desvio-padrão da média das saídas da camada anterior, definidos como mostrado abaixo.

$$\mu^l = \frac{1}{H} \sum_{i=1}^{H} a_i^l \tag{2.25}$$

$$\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^l - \mu^l)^2} \tag{2.26}$$

onde H é o número de neurônios na camada anterior (que coincide com o número de saídas).

2.7.5 Camada de Retirada

Uma das maiores preocupações no momento de se treinar uma rede neural é o sobreajuste, como já foi discutido anteriormente. Uma forma de reduzir essa mazela é através

do uso de camadas de retirada (do inglês, *dropout*), que ajudam a gerar redundância dentro da rede. O hiperparâmetro que uma camada de retirada recebe é uma probabilidade entre 0 e 1, que consiste na probabilidade de desativação de saídas da camada anterior, alterando seu valor para 0 e fazendo com que eles não sejam usados no processo de treinamento. Na prática, é equivalente a desativar os neurônios da camada anterior. Quais neurônios são de fato desativados muda em cada passo do treinamento, ou seja, sempre que um novo lote de dados é computado. Assim, diferentes rotas são treinadas dentro da rede e a rede como um todo tem mais dificuldade em ficar específica demais para os dados de treinamento. Em outras palavras, a rede tem maior dificuldade de fazer um sobreajuste dos dados de treino.

Quando a rede neural é usada fora do modo de treinamento, as camadas de retirada permanecem inativas, liberando todos os demais neurônios para participar dos cálculos. Este é o caso, por exemplo, de quando a rede computa os dados de validação e de teste.

2.8 Tipos de Aprendizagem

Descreveremos os dois tipos de aprendizagem que contém relação com o presente trabalho, que são também os modelos mais comuns. Apesar de existirem outros modelos de aprendizagem, eles fogem ao escopo do presente trabalho.

Aprendizagem supervisionada ocorre quando o modelo precisa receber os dados e seus gabaritos para aprender a relacioná-los. Os exemplos mais comuns são a classificação e a regressão. Por exemplo, se desejarmos treinar um modelo que, a partir de uma foto de um animal, decidisse se o animal é um gato ou um cachorro, precisaríamos alimentá-lo com várias fotos de gatos e cachorros junto com a resposta esperada. Só depois desse processo de aprendizagem supervisionada por humanos (que geraram os gabaritos) o modelo conseguiria de fato prever corretamente se uma nova foto é de um gato ou de um cachorro.

Aprendizagem não-supervisionada ocorre quando o modelo tem acesso apenas aos dados, sem gabaritos separados, e tem que retirar conhecimento de forma independente. Em geral, modelos que utilizam esse tipo de aprendizagem possuem algum parâmetro que eles buscam aprender como minimizar. Esses modelos podem ser usados para classificação, para identificação dos principais tópicos de um texto, para identificação das características mais importantes de dados (escolha de *features*), dentre diversas outras aplicações. Por exemplo, podemos oferecer ao modelo dados sobre os filmes favoritos de diversas pessoas e pedir para que ele separe essas pessoas em grupos. Dessa forma, teremos grupos de pessoas com preferências cinematográficas parecidas e podemos usar o modelo como um recomendador de filmes para novas pessoas a partir dos filmes que elas gostam.

2.9 Avaliação de performance

2.9.1 Validação Cruzada *K-Fold*

Validação Cruzada é um conjunto de técnicas comumente usada para gerar estabilidade nos resultados e precisão na avaliação de erros. Antes do uso da validação cruzada, é importante separar uma parte do conjunto de dados para servir como conjunto de teste ao final. No método específico *K-Fold*, os dados de treino são divididos em diversos conjuntos e são criados o mesmo número de modelos independentes com a mesma arquitetura e pesos iniciais. Em cada versão da separação dos dados (ou *fold*), um modelo usa um conjunto diferente de dados para validação, usando todos os demais conjuntos para treinamento. Um exemplo de Validação Cruzada com 5 (cinco) *folds* é ilustrado na Figura 2.7.

Ao final, é usado o conjunto de dados de teste para avaliar os modelos finais, por ser um conjunto de dados nunca processado pelas redes treinadas. Cada modelo faz suas previsões no conjunto de testes, gerando métricas e gráficos próprios. Usando a mediana e os percentis das métricas, sobrepõe-se os gráficos e obtém-se informações sobre a estabilidade dos resultados da arquitetura utilizada e uma boa estimativa das faixas de erro esperadas.

O uso dessa técnica busca reduzir erros na avaliação de um modelo causados por dependência na ordem dos dados utilizados para aprender. Um modelo especialmente sensível à escolha do grupo de treino dos dados que recebe pode ter resultados inconsistentes. Utilizando diferentes conjuntos de dados para treinar o mesmo modelo, é possível observar o quão estáveis são seus resultados.

2.9.2 Métricas de performance

Em modelos de classificação binária, temos 4 (quatro) métricas cujas interações entre si nos trazem importantes entendimentos. São elas: o número de falsos positivos, de falsos negativos, de verdadeiros positivos e de verdadeiros negativos, como mostrado na Figura 2.8.

Essas métricas são todas dependentes do hiperparâmetro chamado limiar de escolha utilizado em uma classificação, que é o valor mínimo de probabilidade aceitável para se classificar um objeto como pertencente à uma classe. Quanto mais próximo o limiar de escolha for de 0, mais objetos serão classificados como positivos, enquanto que quanto mais próximo ele for de 1, menos objetos serão classificados como positivos. Tendo isso em mente, as métricas básicas são:

- Verdadeiro Positivo (doravante TP, de *true positive*): caso em que a rede classificou como positivo e a resposta era positivo;

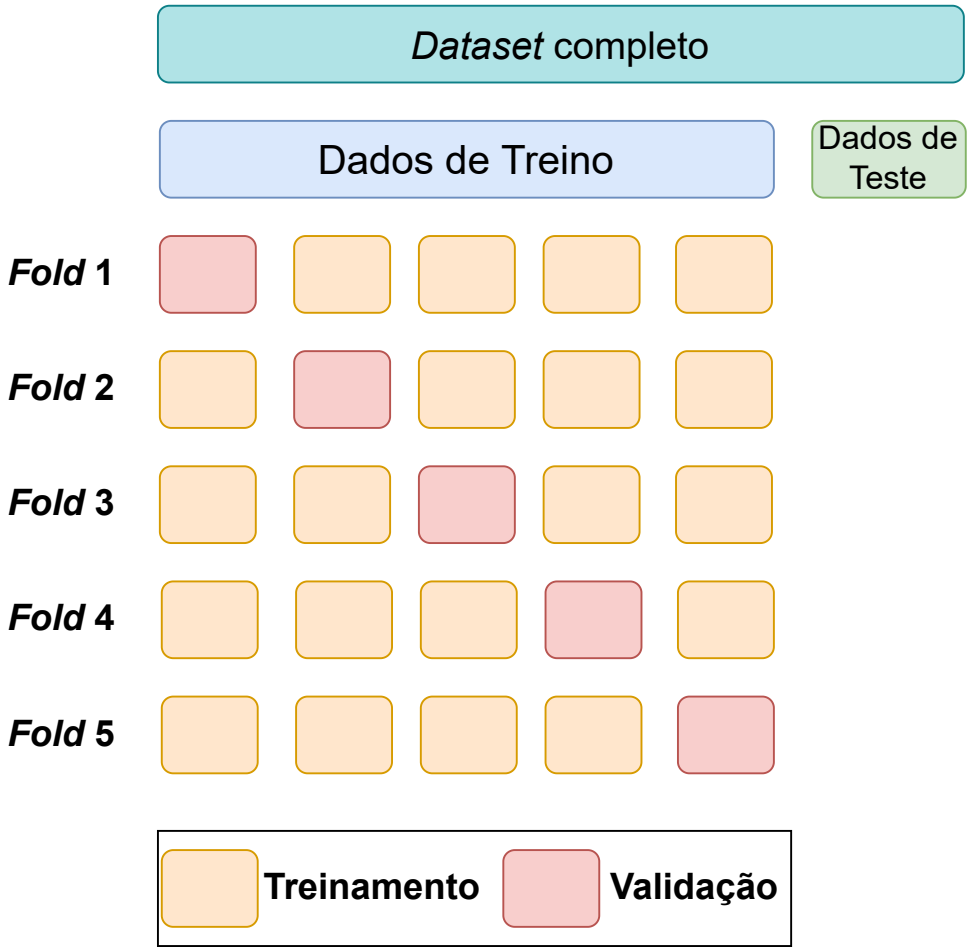


Figura 2.7 – Inicialmente, o conjunto de dados completo é dividido entre dados de treino e de teste. Importante ressaltar que os dados de teste não serão utilizados neste momento. Os dados de treino são novamente divididos em 5 (cinco) conjuntos e são criados o mesmo número de modelos independentes com a mesma arquitetura e pesos iniciais. Em cada versão da separação dos dados (ou *fold*), um modelo usa um conjunto diferente de dados para validação, usando todos os demais conjuntos para treinamento. O conjunto de dados de teste é utilizado para avaliar os modelos finais, por ser um conjunto de dados nunca processado por nenhuma das 5 (cinco) redes treinadas.

- Falso Positivo (doravante FP, de *false positive*): caso em que a rede classificou como positivo, mas a resposta era negativo;
- Verdadeiro Negativo (doravante TN, de *true negative*): caso em que a rede classificou como negativo e a resposta era negativo;
- Falso Negativo (doravante FN, de *false negative*): caso em que a rede classificou como negativo, mas a resposta era positivo.

As demais métricas apresentadas são interações entre as 4 (quatro) métricas básicas aqui descritas.

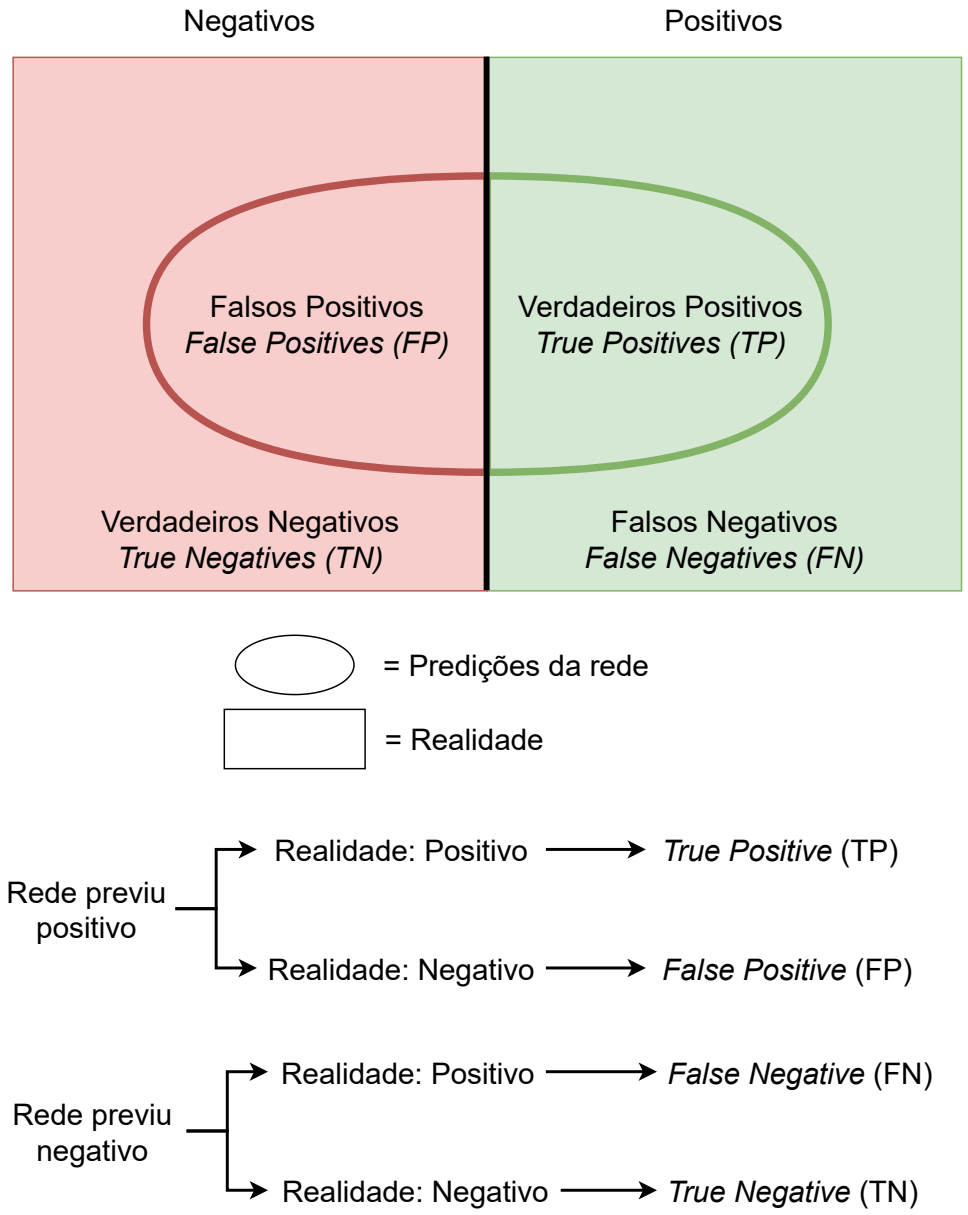


Figura 2.8 – Esquema representativo do significado de cada uma das métricas fundamentais em uma classificação binária. Os dados dentro da elipse são os que a rede considerou como sendo positivos.

2.9.3 Área sob a Curva Característica de Operação do Receptor (AUC-ROC)

Duas métricas muito importantes para avaliarmos a qualidade de um modelo de inteligência artificial são a taxa de verdadeiros positivos (doravante TPR, sigla para *true positive rate*) e a taxa de falsos positivos (doravante FPR, sigla para *false positive rate*). Em termos de equações, temos:

$$TPR = \frac{TP}{TP + FN} \tag{2.27}$$

$$FPR = \frac{FP}{FP + TN} \tag{2.28}$$

Como a soma entre TP e FN representa exatamente todos os dados cuja resposta é positivo, a TPR é uma proporção entre quantos dados a rede acertou que eram positivos e a quantidade real de positivos no conjunto de dados. Limiares de escolha mais baixos aumentam a valor de TPR, pois mais dados serão julgados como positivos (mesmo que erroneamente). Assim, TP aumenta enquanto FN diminui.

Por outro lado, a soma entre FP e TN representa todos os dados cuja resposta é negativa. Logo, a FPR é uma proporção entre a quantidade de dados negativos que a rede julgou corretamente e a verdadeira quantidade de dados negativos. Limiares de escolha mais altos aumentam a valor de FPR, pois mais dados serão julgados como negativos (mesmo que erroneamente). Assim, TN aumenta enquanto FP diminui.

A Curva Característica de Operação do Receptor (doravante Curva ROC, de *Receiver Operating Characteristic*) relaciona essas duas variáveis. A curva é, graficamente, TPR *vs.* FPR variando o limiar de escolha. Em outras palavras, cada ponto da curva representa um possível valor de TPR e FPR, a depender do limiar de escolha definido.

A Área Sob a Curva ROC (doravante AUC-ROC, sigla para *area under the ROC curve*) é uma métrica que resume a informação gráfica da Curva ROC. A AUC-ROC varia de 0 a 1, sendo 0 um classificador que erra todas previsões e 1, um classificador que acerta todas as previsões. Um classificador binário aleatório possui uma AUC-ROC de 0.5, acertando e errando previsões na mesma proporção. Assim, um valor de AUC-ROC maior de 0.5 é um ótimo parâmetro inicial para um modelo. Além disso, é uma das métricas que serve como um parâmetro de escolha entre modelos. A Figura 2.9 mostra um exempo de curva ROC de um classificador perfeito (AUC-ROC = 1), de um classificador aleatório (AUC-ROC = 0.5) e de um classificador com resultados intermediários (AUC-ROC = 0.8).

2.9.4 Curva de Precisão-Revocação

Duas outras métricas muito importantes para avaliarmos a qualidade de um modelo de inteligência artificial são a precisão e o revocação. Em termos de equações, temos:

$$Precisão = \frac{TP}{TP + FP} \tag{2.29}$$

$$Revocação = TPR = \frac{TP}{TP + FN} \tag{2.30}$$

Como a soma entre TP e FP representa todos os dados que a rede classificou como positivos, a precisão é a proporção de quantos dados que a rede julgou como positivos eram de fato positivos. A interação entre o limiar de escolha e a precisão não é tão simples, pois o aumento do limiar de escolha reduz quantos dados são julgados como positivos, reduzindo tanto a quantidade de TP quanto de FP.

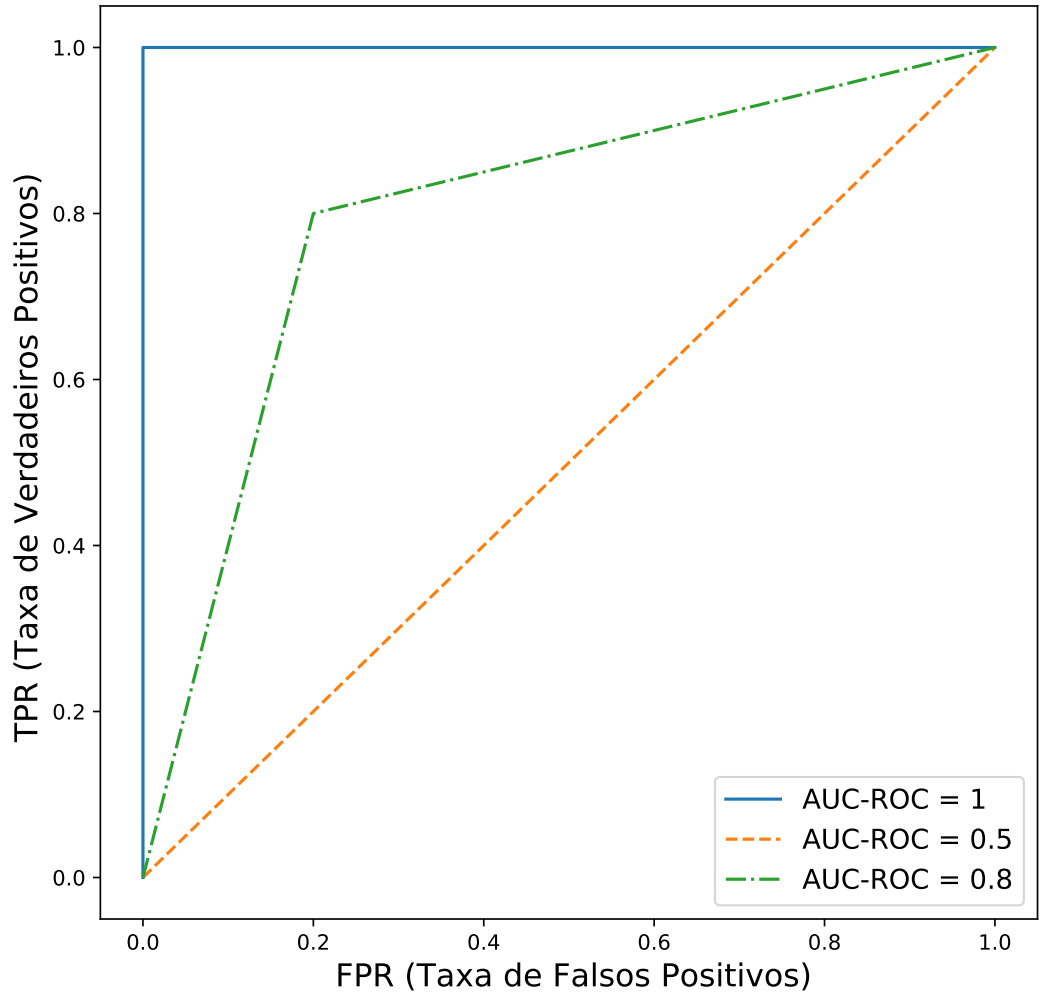


Figura 2.9 – Exemplos de Curvas ROC, sendo uma perfeita (AUC-ROC = 1), uma com predições aleatórias (AUC-ROC = 0.5) e uma com resultado intermediário em relação às outras (AUC-ROC = 0.8).

Revocação é um nome alternativo para TPR, comumente usado no contexto da Curva de Precisão-Revocação. Como já descrito anteriormente, limiares de escolha mais baixos aumentam o valor da revocação, pois mais dados serão julgados como positivos (mesmo que erroneamente). Assim, TP aumenta enquanto FN diminui.

A Curva de Precisão-Revocação relaciona essas duas variáveis. A curva é, graficamente, Precisão *vs.* Revocação variando o limiar de escolha. A informação que ela nos traz é um balanço entre pureza e completeza de nossos resultados, ou seja, para cada valor de limiar de escolha, quantos dos elementos verdadeiramente positivos conseguimos selecionar (revocação) e qual foi a proporção entre seleções corretas e incorretas (precisão). Idealmente, queremos que ambos os valores sejam o mais alto possível.

Podemos sintetizar essas informações na métrica chamada Precisão Média, que representa a área sob a Curva de Precisão-Revocação. Para cada valor k do hiperparâmetro limiar de escolha, temos um valor respectivo $Revocação_k$ e $Precisão_k$ para a revocação e a precisão do modelo. Assim, a Precisão Média pode ser calculada como sendo

$$Precisão\ Média = \sum_{k=0}^{n-1}(Revocação_k - Revocação_{k-1}) \cdot Precisão_k \tag{2.31}$$

onde a soma é feita sobre todos os n valores adotados pro limiar de escolha e fixamos $Revocação_n = 0$ e $Precisão_n = 1$. Idealmente, a área sob uma Curva de Precisão-Revocação possui valor igual a 1, logo o mesmo ocorre com a Precisão Média. Assim como a AUC-ROC, o valor da Precisão Média é usado como parâmetro de escolha entre modelos.

2.10 Problemas multi-rótulo

Dentro da classe de problemas de classificação em aprendizado supervisionado, temos uma categoria comumente chamada de classificação multi-rótulo, que consiste em classificar objetos que podem pertencer a mais de uma categoria ao mesmo tempo. Por exemplo, se desejarmos classificar pessoas de acordo com os tipos de livros que elas gostam de ler, teremos um exemplo de classificação multi-rótulo. A pessoa A pode gostar de ler livros de romance e mistério, a pessoa B, de mistério apenas, a pessoa C, de terror e ficção científica, enquanto a pessoa D não gosta de ler nenhum gênero literário.

As técnicas comumente usadas nesse tipo de classificação se assemelham às usadas em classificações binárias. A função erro comumente adotada consiste em somar as entropias cruzadas binárias (ECB) de todas as classes.

$$ECB_{perda} = \sum_{j=1}^{C} -(y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)) \tag{2.32}$$

onde C é o número de classes, y_j é o valor real da classe j (0 ou 1) e $\hat{y}_j$ é o valor previsto pelo modelo para a classe j (entre 0 e 1). Como pode ser observado na equação acima, soma-se a entropia cruzada binária associada à cada classe (cada j). Como o valor de cada y_j é sempre 1 ou 0, apenas parte da equação é realmente computada. Um valor perfeito dessa função perda seria 0. Quando o processo é feito em um lotes de dados, tira-se a média (ou faz-se a soma) das perdas de cada dado.

Uma versão mais estável numericamente é a chamada entropia cruzada binária com regressão logística (do inglês, *binary cross-entropy with logits*), que usa uma função sigmóide dentro dos logaritmos.

$$ECB_{reg.log.} = \sum_{j=1}^{C} -(y_j \cdot log(f(\hat{y}_j)) + (1 - y_j) \cdot log(f(1 - \hat{y}_j))) \tag{2.33}$$

Lembrando que a função sigmóide é dada por:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.34}$$

Capítulo 3

Processamento de Linguagem Natural

Processamento de Linguagem Natural (doravante NLP, de *Natural Language Processing*) é uma área dentro de IA voltada ao processamento da linguagem humana por computadores, estando o processamento de informações em texto dentro de seu escopo.

3.1 Técnicas Inicias

3.1.1 Tokenização e Vocabulário

Técnicas de NLP exigem que seus dados de entrada (texto) sejam representados de forma que seja possível realizar operações. Essa representação normalmente se dá por meio de *tokens*, que nada mais são do que representar um trecho de texto através de um número, normalmente um dado do tipo int.

Existem diversas técnicas de tokenização, que podem separar o texto por palavras, caracteres ou trechos de palavras, que chamaremos de sub-palavras. A consideração ou não de letras maiúsculas, pontuação ou caracteres especiais (emojis, por exemplo) variam de acordo com a técnica de tokenização utilizada e com o objetivo do trabalho. Por exemplo, um trabalho que objetivasse construir uma rede neural capaz de identificar nomes de pessoas em um texto poderia ser beneficiada por um processo de tokenização que identificasse palavras que iniciam com letra maiúscula de forma diferente das que iniciam com letra minúscula.

Associado ao processo de tokenização, há a criação de um dicionário, que funciona como um conjunto de pares do tipo {trecho de texto: token associado}. O objetivo do dicionário é servir como um bijeção entre os trechos de texto e seus respectivos tokens.

Importante ressaltar que tokenizadores precisam ser pré-treinados, ou seja, precisam

ser alimentados com dados textuais para construírem seus dicionários e serem posteriormente utilizados.

3.1.2 *Embedding*

Embedding é o processo de representar vetorialmente objetos de forma que os objetos com significado semelhante tenham representações semelhantes. Esse processo pode ser feito com *tokens*, posição de cada *token* em uma frase e outras informações, a depender do *embedding* escolhido.

3.1.3 Bolsa de Palavras

Uma das formas mais simples de representar dados em texto é usando a chamada Bolsa de Palavras (*Bag of Words*). Nesse modelo, toda a estrutura e contexto são perdidos e é feita uma simples contagem das palavras presentes em cada texto do nosso *dataset*.

O modelo funciona da seguinte forma:

1. Tokenização;
2. Construção do vocabulário;
3. Codificação.

Cada texto do dataset é tokenizado (1) de forma que cada palavra se torna um *token* único. Assim, nosso dado deixa de ser uma *string* de texto e passa a ser uma lista de números, que são os *tokens*. Em seguida, o vocabulário é construído (2) coletando e ordenando cada *token* único existente no conjunto de dados. A forma em que os *tokens* são ordenados não é importante nesse tipo de modelo, podendo ser ordem alfabética, ordem de aparecimento ou qualquer outra forma desejada. Por fim, os dados são codificados (3) a partir do vocabulário construído. Cada dado passa de uma lista de *tokens* para um vetor em que cada entrada é o número de vezes que determinada palavra do vocabulário aparece no texto original do dado.

Com os dados estruturado dessa forma, qualquer algoritmo que receba um vetor como entrada pode ser utilizado para processar o conjunto de dados, ficando a escolha dependente, principalmente, da tarefa que se deseja realizar.

3.1.4 Palavras de Parada

Palavras de Parada (*Stopwords*) é um tipo comum de pré-processamento feito para reduzir a complexidade de *datasets* de texto sem perder informações relevantes. A técnica consiste em se fazer uma lista de palavras que fazem parte de construções gramaticais da língua, mas que não acrescentam significado, e retirá-las do conjunto de dados. Em

português, algumas palavras que podem ser retiradas sem alterar o significado da grande maioria das frases são “que”, “a”, “o”, “e” e “então”.

Dessa forma, menos palavras precisarão ser processadas e podemos construir vocabulários mais enxutos, apenas com palavras que carregam mais significado.

3.1.5 *Stemming*

Stemming é mais uma técnica de pré-processamento cujo objetivo é reduzir a complexidade do conjunto de dados e o tamanho do vocabulário. A ideia aqui é reduzir variações que uma palavra pode ter devido às flexões de gênero, número, tempo, etc. . A partir de uma série de regras gramaticais que variam para cada língua, deseja-se usar apenas o radical de cada palavra, não ela inteira. Importante frisar que o radical da palavra nesse contexto nem sempre coincide com o radical gramatical da palavra, porque o nosso será feito a partir de um conjunto de regras que muitas vezes falham em coincidir com as regras gramaticais quando tratamos de flexões irregulares.

Alguns exemplos: “eles”, “ele”, “elas” e “ela” se tornariam “el”, enquanto “corrida”, “correr”, “correram”, “correrão” se tornariam “corr”. Existem diferentes algoritmos de *stemming*, cada um com suas regras próprias e que visam reduzir mais ou menos as palavras.

3.2 Transformers

A metodologia do tipo *Transformer* é uma técnica de aprendizagem profunda introduzida em 2017 para tarefas de processamento de linguagem natural. A principal ideia da metodologia *Transformer* é o uso de um mecanismo de atenção ensinável para escolher quais partes do texto são mais relevantes para o significado de cada palavra. [6].

Um texto precisa ser tokenizado antes de ser processado pelo mecanismo de atenção. Em outras palavras, o texto é dividido em palavras e sub-palavras, então cada palavra (ou sub-palavra) é substituída pelo seu *token* correspondente (um número de identificação). Cada um dos *tokens* possui uma representação *embedding* diferente, um vetor com entradas reais e dimensão fixada que representa o *token*.

O processo de atenção do *Transformers* acontece na chamada camada de atenção *multi-head*, formada por várias subcamadas de auto-atenção. O objetivo dessa camada é transformar cada representação de *embedding* em um novo *embedding* que é uma combinação linear de todos os *embeddings* da frase.

Em cada camada de auto-atenção, também chamada de *head*, o *embedding* de entrada é multiplicado por 3 (três) matrizes com pesos aprendíveis chamados W_Q, W_K, e W_V, resultando em 3 (três vetores) q_i, k_i, e v_i, respectivamente, para cada token de entrada. Esses vetores são conhecidos como “*consulta*”, “*chave*” e “*valor*”, respectivamente. Por

exemplo, o mecanismo de atenção pro Token 1 envolve multiplicar q_1, o vetor consulta do Token 1, por cada um dos vetores k_i, os vetores chave de cada token, resultando em uma pontuação para cada token. Os *scores* são regularizados, divididos por 8 no nosso exemplo, e alimentam uma função do tipo *softmax*, resultando em um número entre 0 e 1 para cada um deles. Em seguida, cada um desses números é multiplicado por seu vetor valor v_i. A soma desses vetores valor ponderados é o vetor z_1, a saída do Token 1. Em seguida, o vetor z_1 de cada *head* são concatenados e multiplicados por W_O, dando a saída do Token 1, a novo *embedding* para ele. Os cálculos dentro de uma camada de auto-atenção podem ser visualizados na Figura 3.1. A ideia por trás desse processo é simples: quanto mais similares forem as representações da chave e da consulta, maior a pontuação e mais atenção é dada ao token.

Particularmente, redes *Transformer* são indicados para nossa tarefa porque esse método é baseado em atenção sobre todo o contexto da frase de uma vez, e alcançou resultados estado-da-arte em tarefas de classificação [30, 31] .

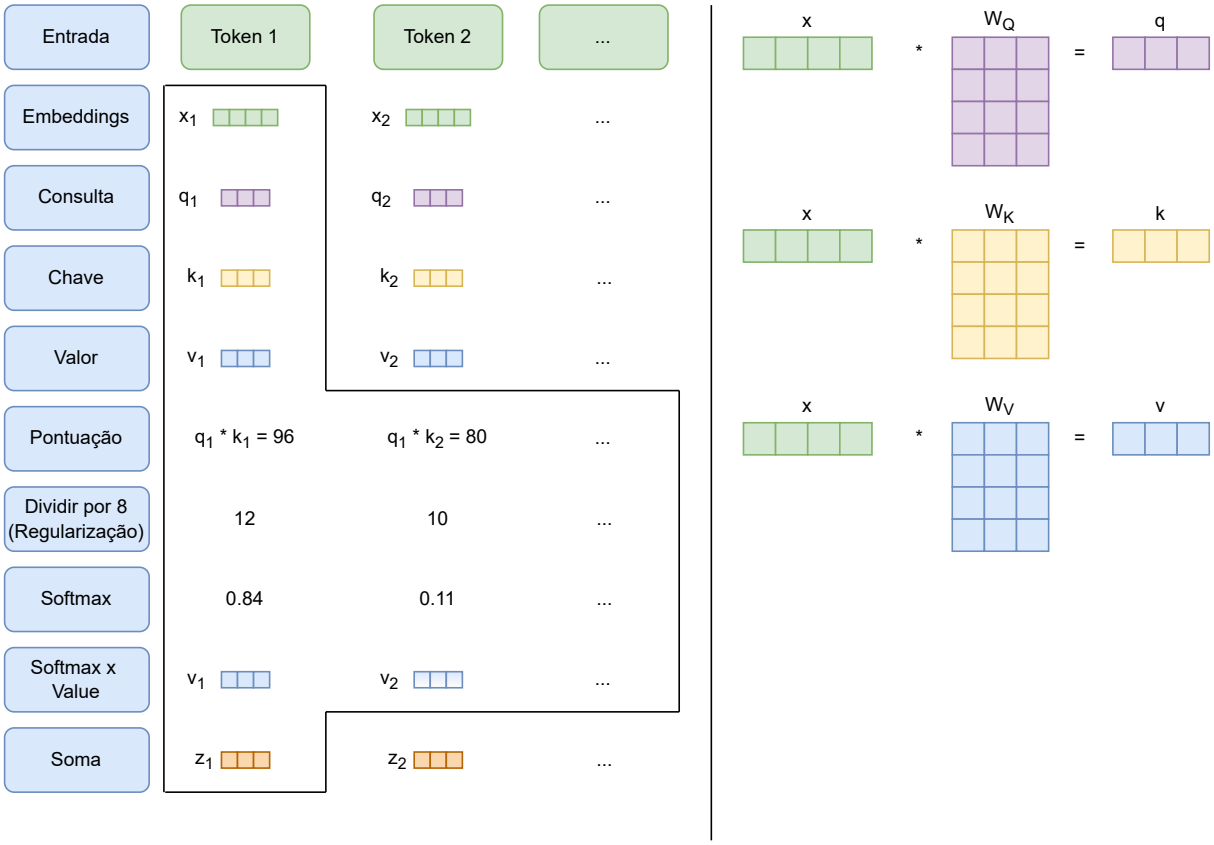


Figura 3.1 – Na esquerda, esquema exemplificando os cálculos feitos em uma *head* de atenção. Na direita, representação gráfica de como obter os vetores consulta, chave e valor a partir do *embedding* original de um *token*.

3.3 *Bidirectional Encoder Representations from Transformers* (BERT)

BERT é uma rede *Transformer* pré-treinada usando a Wikpédia em inglês e o conjunto de dados *BooksCorpus* [7]. Modelos baseados na arquitetura BERT têm atingido resultados de estado-da-arte em múltiplas tarefas de classificação, justificando assim sua escolha para a tarefa pretendida [9, 32, 33].

BERT é uma rede do tipo *encoder*, ou seja, é baseada em um empilhamento de camadas de *encoder*. Cada *encoder* possui, em seu interior, uma camada de atenção *multi-head*, do tipo descrito na seção anterior, e uma rede densa com atenção à posição (doravante PWFFN, de *Position-Wise Feed Forward Network*). Além disso, o *encoder* também possui em seu interior saltos de posições, camadas de retirada (com $p = 0.1$) e camadas de normalização, práticas padrões para um treinamento mais efetivo de redes neurais.

A PWFFN é uma rede densa que processa cada *embedding* de forma independente, ao invés de processar a sequêcia de *embeddings* como um vetor. Em seu interior há uma subcamada densa com dimensionalidade de saída de 3072 e outra com dimensionalidade de saída de 768 (o tamanho padrão de um *embedding*). Tudo descrito até aqui é o chamado corpo da rede. Uma BERT de tamanho base possui 12 camadas de *encoders*, somando por volta de 110 milhões de parâmetros. Uma rede BERT de tamanho base pode ser vista na Figura 3.2.

No modelo utilizado no presente trabalho, depois de todas as camadas de *encoders*, a rede termina com uma camada densa com dimensionalidade de saída de 768 e uma tangente hiperbólica, *tanh*, como função de ativação, seguida por uma camada de retirada (com $p = 0.1$) e uma camada densa final de dimensionalidade de saída de 14 (*i.e.*, o número de sintomas a serem identificados no nosso estudo de caso). Na camada de classificação, há uma função de ativação do tipo sigmóide para cada uma das saídas. Essa parte final, responsável pela classificação em nosso trabalho, é chamada de cabeça do modelo.

3.3.1 Dicionário e Tokenizador

O processo de formação do dicionário e pré-treinamento do tokenizador de uma rede BERT se dá na forma de um processo regido por regras pré-estabelecidas. Apesar de existirem diversas técnicas de tokenização, redes BERT utilizam, por padrão, uma técnica de tokenização chamada de *wordpiece*, que está descrita abaixo. Para o pré-treinamento, é necessário definir o tamanho máximo do dicionário e alimentar o tokenizador com dezenas ou centenas de milhares de textos para se obter bons resultados.

Primeiro, todos os textos são pré-tokenizados em coleções de palavras e então cada palavra em uma coleção de caracteres. Primeiramente, os caracteres únicos são identificados e adicionados ao dicionário de *tokens*. Em seguida, o tokenizador calcula, para cada par de *tokens* adjacentes presentes nas palavras, qual o par cuja probabilidade dividida pelo produto entre as probabilidades individuais dos *tokens* dá o resultado mais alto. Esse par

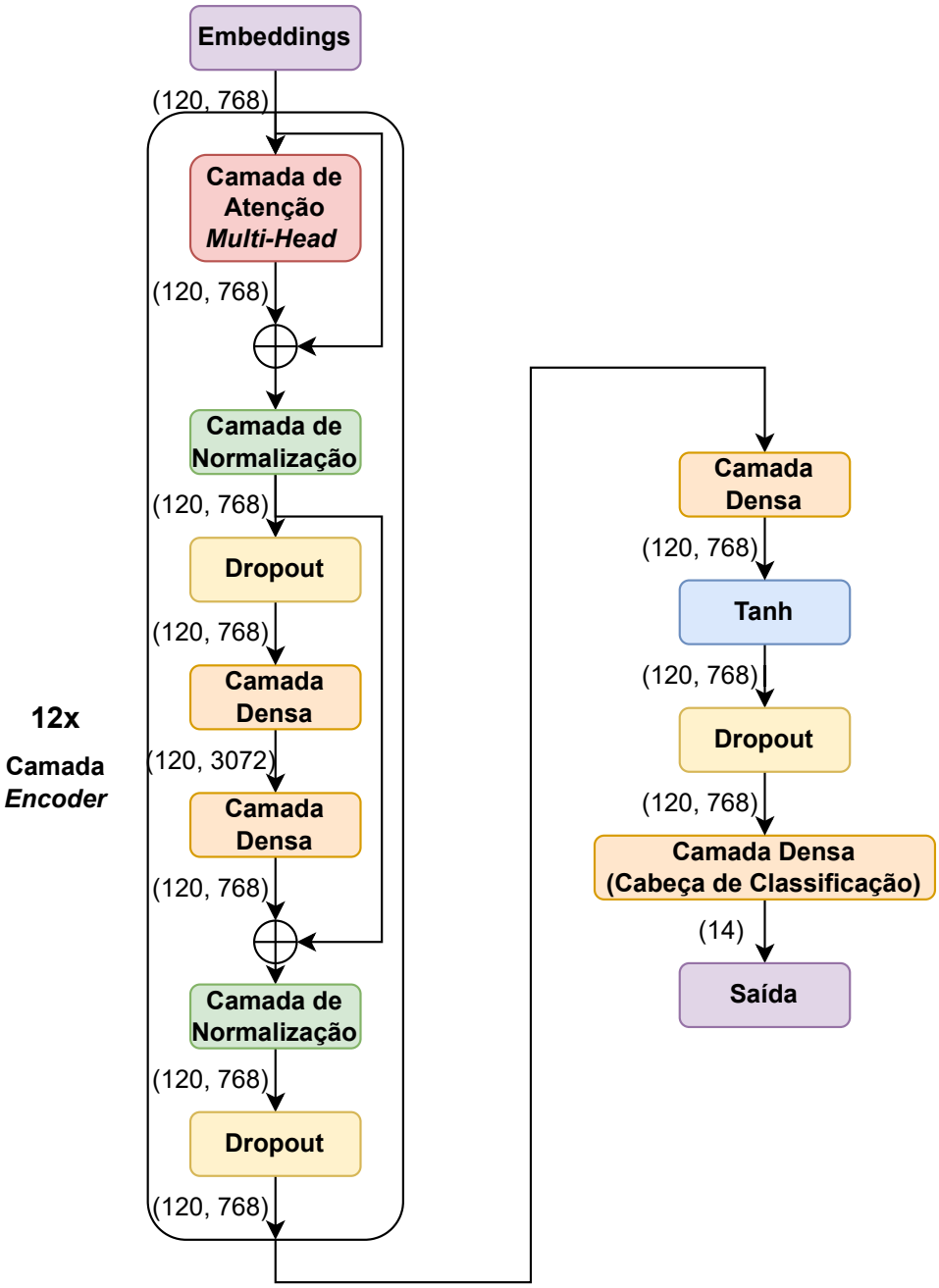


Figura 3.2 – Esquema representativo da arquitetura de uma rede BERT como utilizada no presente trabalho, com uma cabeça de classificação de 14 sintomas. Os números que aparecem após cada camada indicam o tamanho da saída da respectiva camada.

será a nova entrada do dicionário. Esse processo se repete até que o tamanho do dicionário alcance o valor desejado ou até que as probabilidades fiquem abaixo de um valor mínimo pré-estabelecido [7, 34]. Um exemplo de tokenização do tipo *wordpiece* pode ser visto na Figura 3.3.

A diferenciação entre letras maiúsculas e minúsculas depende do tokenizador utilizado. Neste trabalho iremos utilizar tokenizadores que não diferenciam entre maiúsculas e minúsculas, tendo em vista que trabalhamos com textos não revisados e retirados da

internet, onde regras gramaticais nem sempre são seguidas. Os dicionários de redes BERT costumam ter por volta de 30 mil tokens.

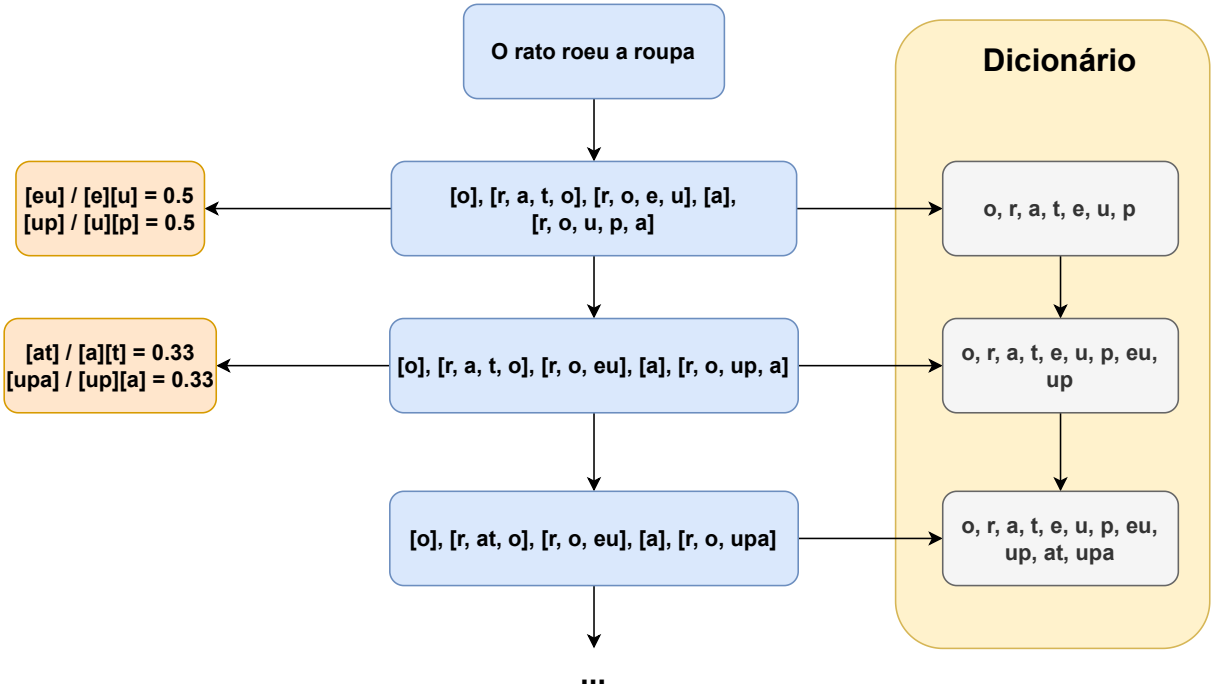


Figura 3.3 – Exemplo do funcionamento de um tokenizador do tipo *WorkPiece* usando a frase "O rato roeu a roupa".

3.3.2 *Embeddings*

O *token embedding* (doravante TE) da rede BERT associa cada token a um vetor com 768 dimensões. Essas representações são geradas durante a fase de pré-treinamento da rede BERT.

O *positional embedding* (doravante PE) tem a tarefa de representar a informação sobre a posição dos tokens na frase. Sem ele, o processo se reduziria a uma bolsa de palavras.

No caso específico de redes BERT, é usado, por padrão, um PE absoluto, ou seja, que carrega a informação da posição absoluta de cada token na frase, em contraste com um PE relativo, que carregaria a informação da distância relativa de cada *token* em relação aos demais. Além disso, o PE utilizado é aprendível, sendo que seu aprendizado ocorre durante o pré-treinamento. [7, 35].

Cada posição na frase, que tem capacidade para até 512 tokens, possui uma representação de PE como um vetor de 768 dimensões.

Além disso, temos o *Segment Embedding*, um *embedding* aprendível para diferenciar se o token está antes ou depois do token [SEP], ou seja, se pertence à primeira ou segunda frase, sendo que seu aprendizado ocorre durante o pré-treinamento.

Por fim, o *Final Embedding* é a simples soma dos demais *embeddings* descritos até aqui e é o *embedding* que de fato é processado pela rede BERT. Uma representação de sua construção é feita na Figura 3.4.

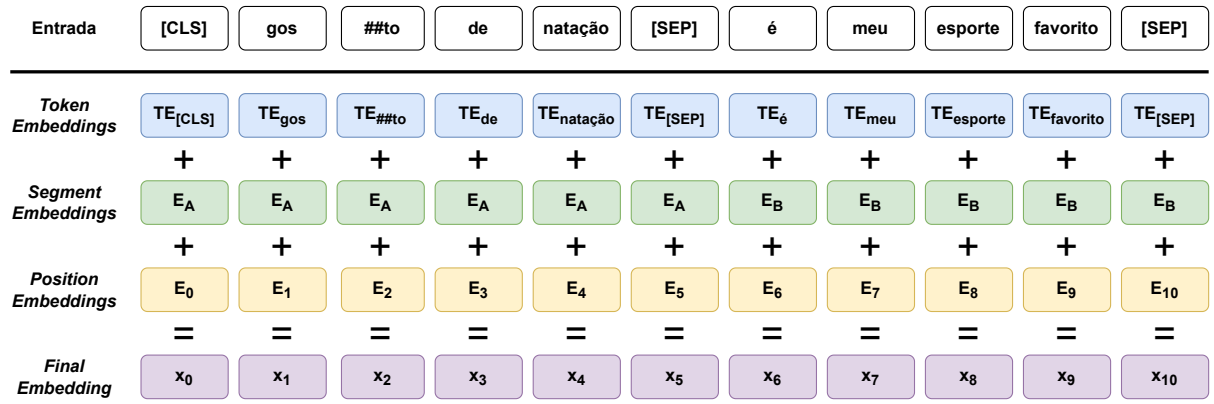


Figura 3.4 – Esquema representativo de como os *embeddings* de uma rede do tipo BERT são construídos ao somar-se os *embeddings* dos tipos *token*, *segment* e *position*.

3.3.3 Pré-Treinamento

O sucesso de redes BERT vem de seu pré-treinamento que utiliza duas técnicas únicas: a modelagem de linguagem mascarada e a predição de próxima frase. Durante o pré-treinamento, a perda utilizada é um somatório da perda obtida em cada uma dessas tarefas. O que diferencia duas redes BERT de mesma arquitetura é seu processo de pré-treinamento, incluindo em qual *dataset* (e língua) o pré-treinamento foi feito.

3.3.3.1 Modelagem de Linguagem Mascarada (MLM)

No MLM, a ideia é que a rede BERT receba um dado, processe-o e devolva como saída o dado original. Para que esse processo não seja fácil demais, parte dos *tokens* do dado original são substituídos pelo *token* especial [MASK].

O processo como um todo ocorre da seguinte forma: utilizando um tokenizador do tipo *WordPiece*, os dados são tokenizados e, em seguida, truncados ou completados para que sejam representados por uma lista com 512 *tokens*. Os dados, a partir deste ponto, serão chamados de token_ids. Em seguida, são criadas 3 (três) variáveis:

1. Ids de entrada: uma cópia da variável token_ids, mas com cerca de 15% dos seus tokens sendo mascarados usando o token [MASK];

2. Máscara de atenção: um tensor de 1s e 0s marcando as posições de tokens reais e de tokens usados para completar a lista até 512, caso existam. Importante observar que essa máscara de atenção não têm nada a ver com os tokens [MASK];

3. Rótulos: os token_ids originais, sem máscaras.

O modelo usará a máscara de atenção para definir para quais *tokens* olhar, desconsiderando os *tokens* utilizados para completar a lista. A tarefa do modelo é receber os ids de entrada, processá-los como descrito anteriormente e entregar, para cada *token* [MASK], uma *softmax* de todo o dicionário da rede. O resultado com maior probabilidade será a resposta da rede para aquele *token* [MASK]. Seu objetivo é descobrir qual a palavra original naquela posição, usando o rótulo como gabarito para computar a função perda, utilizando o AdamW como otimizador.

Durante o treinamento para MLM, o modelo tokeniza toda a frase (que possui o *token* [MASK] em 15% da frase), a processa usando a rede BERT e a passa por uma rede densa que devolve um resultado (*output logits*) para cada *token*. Cada um desses resultados possui a dimensão do dicionário (por volta de 30k *tokens*) e será passado por uma *softmax*, devolvendo uma distribuição de probabilidades. Passando a distribuição por um *argmax*, se obtém o token_id de maior probabilidade. O objetivo é que a rede devolva a frase original, mas só são avaliados os resultados para os tokens [MASK].

Nem todos dos 15% dos tokens aleatoriamente escolhidos são de fato substituídos por *tokens* [MASK], apenas 80% deles o são. Outros 10% são substituídos por palavras aleatórias do vocabulário e nos últimos 10%, o token não é substituído, permitindo que a rede faça a predição do próprio token. Essa tarefa é treinada usando entropia cruzada como função perda.

3.3.3.2 Predição de Próxima Frase (NSP)

A Predição de Próxima Frase (doravante NSP, de *Next Sentence Prediction*) é outra tarefa a qual na rede BERT é pré-treinada. Durante o pré-treinamento, o modelo foi treinado sempre recebendo duas frases separadas por um token [SEP]. Em 50% do treinamento, a segunda frase era uma continuação direta da primeira, e nos demais 50% foram usadas frases pegas de forma aleatória. A classificação se as frases eram continuação uma da outra ou não, foi feita usando o token especial [CLS], que é sempre o primeiro. A função perda utilizada é a entropia cruzada binária.

3.3.4 Corpo e Cabeça

Uma rede BERT, assim como várias outras redes, pode ser dividida em duas partes: a cabeça e o corpo. O corpo é formado por toda a arquitetura da rede, com exceção da camada final que dá a saída da rede. A cabeça, por outro lado, é exatamente a camada de saída da rede. Essa separação entre corpo e cabeça existe porque, uma vez que o corpo é pré-treinado (pelos criadores originais da BERT, nesse caso), tudo que precisamos fazer para usar a rede em um tipo de problema diferente é mudar a cabeça e treinar o modelo na nossa tarefa específica. Esse processo é chamado de *fine tuning*.

A saída do corpo da rede BERT é uma lista de novos *embeddings* para os *tokens* originais que entraram na rede. No caso em que queremos usar uma cabeça de classificação, o padrão é utilizar apenas o *embedding* final do primeiro token da lista, ou seja, do token especial <CLS>, como entrada para a cabeça de classificação.

Comparado ao pré-treinamento, o *fine tunning* é muito barato computacionalmente, além de ter se mostrado altamente eficiente em diversas aplicações [7].

3.4 BERT-Multilingual

BERT-Multilingual é uma versão da BERT-Base liberada pelos próprios autores da original [7]. Diferentemente da BERT padrão, essa versão foi pré-treinada nas 104 línguas com maiores Wikipédia e possui um vocabulário de 119,547 *tokens*, cerca de 4 (quatro) vezes o tamanho do vocabulário da versão original.

3.5 BERTimbau

BERTimbau é uma versão da rede BERT pré-treinada no corpo dos textos do *corpus* brWaC, que soma 2.68 trilhões de *tokens* em 3.53 milhões de documentos [36, 37].

3.5.1 Vocabulário e Tokenizador

Diferentemente da BERT original, os criadores da rede BERTimbau usaram a técnica *Byte-Pair-Encoding* (doravante PBE) para tokenizar seus dados para o pré-treinamento. PBE é uma técnica de tokenização muito similar à *wordpiece*, introduzida em [38]. O único parâmetro necessário nessa técnica é o tamanho máximo do dicionário desejado.

Similarmente à técnica *wordpiece*, os textos são pré-tokenizados em coleções de palavras e então cada palavra em uma coleção de caracteres. Em seguida, cada caracter individual é adicionado ao vocabulário. Em seguida, o tokenizador calcula, para cada par de tokens adjacentes presentes nas palavras, qual o par que aparece a maior quantidade de vezes. Esse par será a nova entrada do dicionário, sendo essa a única diferença entre as ténicas PBE e *wordpiece*. Esse processo se repete até que o tamanho do dicionário alcance o valor desejado.

Ao final, é feita uma adaptação do dicionário resultante para o padrão *wordpiece*, para manter a compatibilidade com a rede BERT. O pré-treinamento foi realizado em 2.000.000 de frases aleatórias de artigos da Wikipedia em português, até que se chegou a 30.000 tokens [36].

3.5.2 Treinamento

A rede BERTimbau foi pré-treinada em MLM e em NSP, similarmente à BERT original. O modelo foi treinado em 1 milhão de passos, similarmente à BERT original, com taxa de aprendizagem de 10^{-4}, utilizando uma estratégia de aquecimento da taxa de aprendizado durante os primeiros 10.000 passos seguido por um decaimento linear da mesma durante os demais passos. Essa é um técnica utilizada em treinamentos de redes neurais que ajuda a minimizar a perda durante o treinamento. Durante os primeiros 10.000 passos, a taxa de aprendizado cresce linearmente, começando do zero, até o valor definido acima de 10^{-4}. Durante o resto do treinamento, esse valor cai linearmente até voltar ao zero [7]. Dessa forma, os primeiros passos servem para a rede buscar uma região no espaço de parâmetros que possua um minímo local com valor de perda menor do que a região original em que os parâmetros foram inicializados. A segunda fase do processo, quando a taxa de aprendizado está se reduzindo, objetiva chegar ao mínimo dentro dessa região. O uso de passos menores busca previnir que os pesos saiam da região com mínimo encontrada pela fase anterior.

Para a BERTimbau Base, os pesos inicias são pegos da BERT-Base-Multilingual, versão da rede BERT apresentada acima. O tokenizador foi descartado, já que o vocabulário é diferente. O tamanho do lote utilizado durante o pré-treinamento foi de 128 e o treinamento durou por volta de 8 épocas nos dados de treino.

3.6 GPT-2

GPT-2 faz parte de outra família de redes *Transformer*, a família das GPT. Diferentemente da família BERT, que é uma família de modelos do tipo *encoder*, as GPTs são modelos do tipo *decoder* [8]. A diferença de um *encoder* para um *decoder* está no mecanismo de atenção: *encoders* utilizam mecanismos de atenção bidirecionais, enquanto *decoders* utilizam mecanismos de atenção unidirecionais [6]. Os cálculos dentro de uma camada de auto-atenção unidirecional podem ser visualizados na Figura 3.5.

3.6.1 Pré-Treinamento

Assim como no caso da arquitetura BERTimbau, a GPT-2 usa um tokenizador do tipo BPE com um vocabulário de 50.257 *tokens*, podendo receber dados com tamanho de até 1024 *tokens*, o dobro de uma BERT padrão.

Diferentemente da família BERT, a arquitetura GPT-2 é um Modelo de Linguagem Causal (doravante CLM, de *Causal Language Model*), o que significa que seu treinamento se dá através da tentativa de prever a próxima palavra de uma frase usando apenas as palavras anteriores como entrada. Ou seja, estamos diante de um modelo unidirecional,

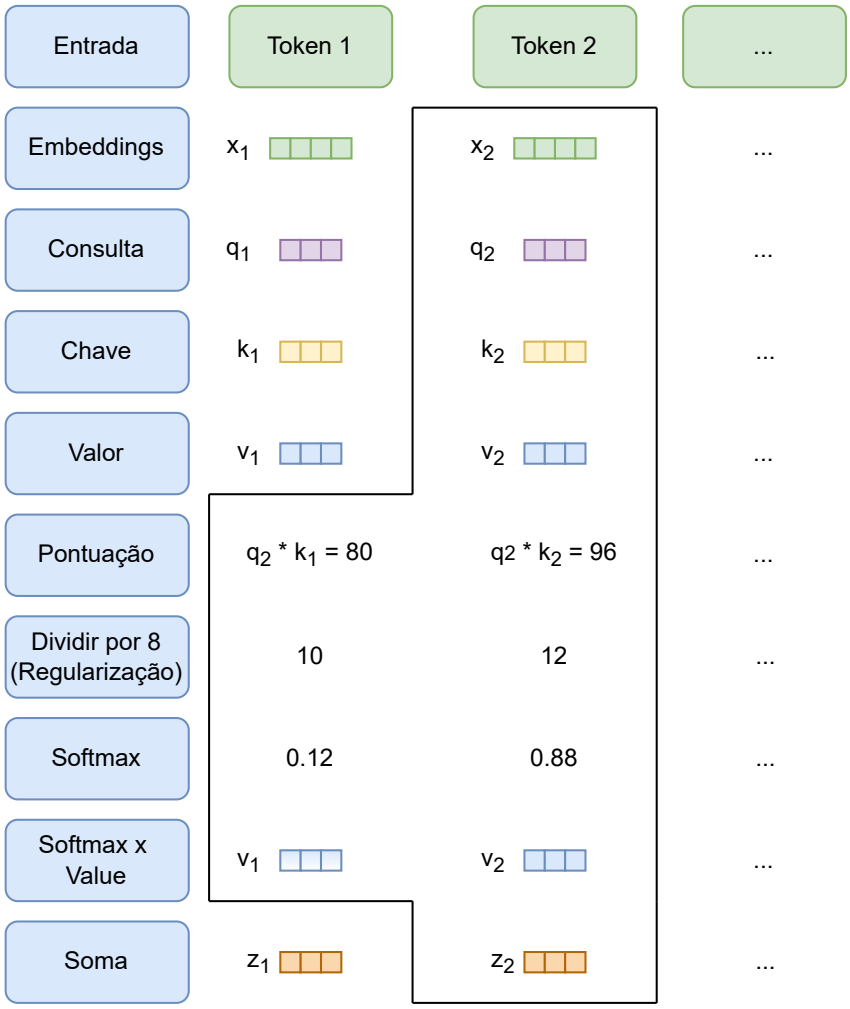


Figura 3.5 – Esquema exemplificando os cálculos feitos em uma *head* de atenção unidirecional.

que não olha todo o contexto da frase para fazer suas predições. Baseado na predição feita, uma função perda do tipo entropia cruzada é utilizada no treinamento.

Uma arquitetura CLM é usada na GPT-2 pois seu maior objetivo está em obter resultados de alta performance para geração de texto. Assim, dando uma semente para a rede na forma de uma frase inicial, ela é capaz de gerar novas frases ou mesmo parágrafos sobre o tema. Uma observação importante sobre esses textos gerados é que eles não possuem nenhum compromisso com a coerência lógica, são simplesmente construções gramaticais aprendidas com textos da internet.

Capítulo 4

Obtenção dos Dados

Durante o projeto, foi realizada uma análise de dados em *tweets* sobre COVID-19 no Brasil publicados entre 1º de fevereiro e 31 de abril de 2021. Um fluxo de trabalho foi desenhado para extrair e processar *tweets* para serem analisados por técnicas de Aprendizado Profundo. O objetivo de tal fluxo é prover um conjunto de dados adequado à nossa metodologia sobre sintomas de COVID-19 em *tweets*.

O fluxo de trabalho, representado na Figura 4.1, é formado por 8 (oito) atividades principais, cada uma sendo representada por um algoritmo, software ou análise de dados. A atividade 1 consiste na escolha inicial dos sintomas de COVID a serem analisados, realizada por profissionais da área médica. A atividade 2 consiste no levantamento de palavras-chave relacionadas à COVID e aos sintomas escolhidos, inculindo *hashtags* relacionadas. A atividade 3 executa o *script* *TweetSearcher* para extrair dados da base do *Twitter* a partir da lista de sintomas e palavras-chave construída nos passos anteriores. As atividades 4, 6 e 8 validam os resultados obtidos usando especialistas em análises de dados. As atividades 5 e 7 executam ferramentas de Inteligência Artificial usando Aprendizado Profundo, Processamento de Linguagem Natural e/ou Redes Neurais nos dados recebidos das atividades 4 e 6, gerando recomendações de como melhorar os conjuntos de dados. Por fim, a atividade 8 valida os resultados e gera um conjunto de dados final no formato CSV.

A execução desse fluxo de trabalho é conduzida por uma interação humano-máquina envolvendo especialistas em análise de dados, saúde e inteligência artificial. Além disso, o fluxo de trabalho pode ser repetido quantas vezes forem necessárias até se alcançar um consenso sobre a qualidade dos dados. Mais detalhes sobre as atividades descritas acima são apresentados abaixo.

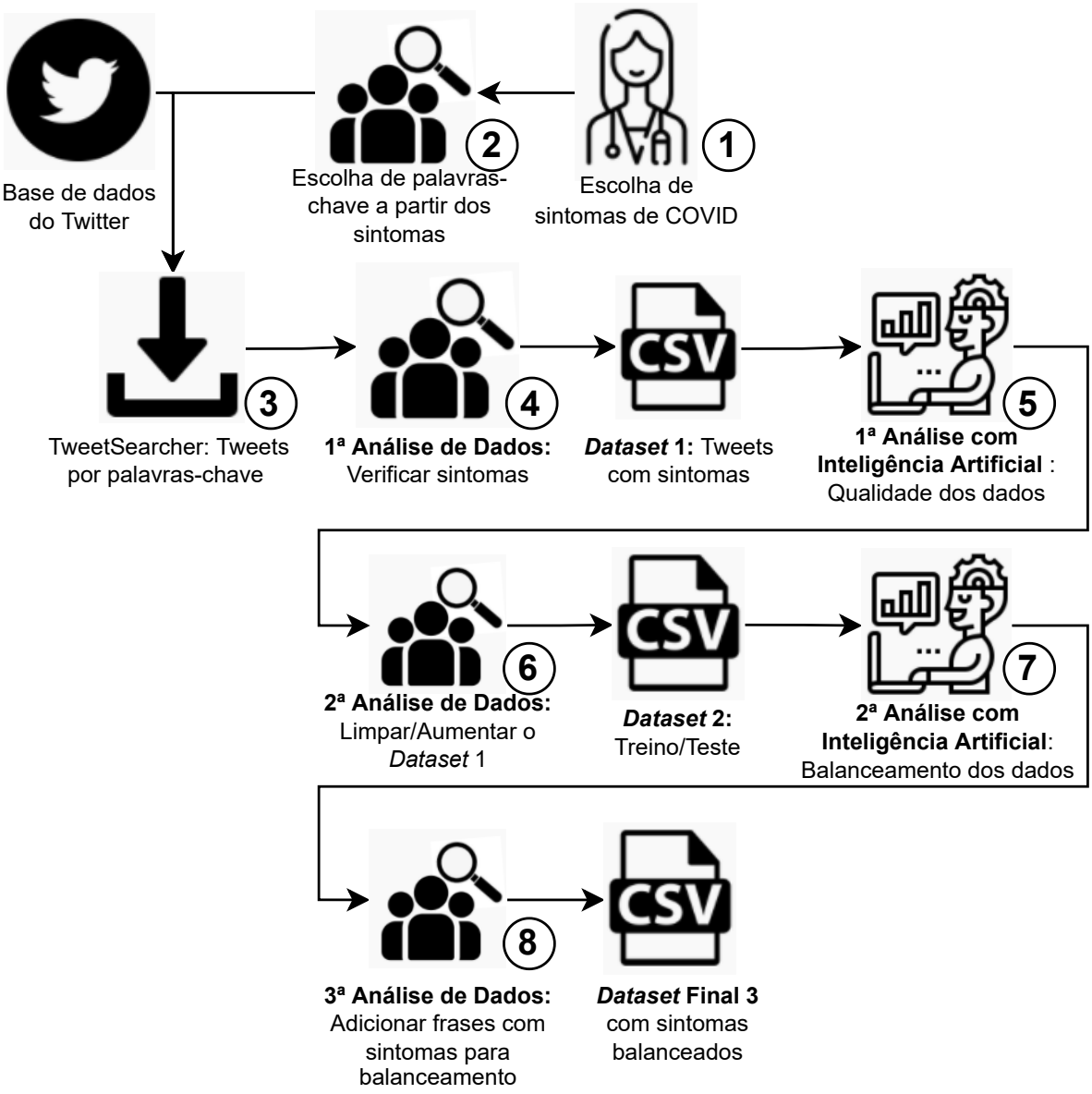


Figura 4.1 – Workflow utilizado para a montagem do *dataset* final utilizado para o treinamento das redes neurais artificias.

4.1 Seleção dos Dados: Palavras-Chave e Sintomas de COVID-19

O conjunto utilizado de palavras-chave relacionadas ao coronavírus foi: *covid*, *corona* e *c19*, incluindo *hashtags* relacionadas, como *#covid19*. Não foram utilizadas variações na escrita tais como *cov1d*. Como os resultados gerados com essas palavra-chave foram muito numerosos, um filtro foi utilizado para que os *tweets* possuíssem ao menos um sintoma de uma lista pré-montada. Adicionalmente, um segundo filtro foi utilizado para remover *retweets* e imagens.

O conjunto inicial de sintomas em português escolhido é apresentado na Tabela 1. Eles são: calafrio, congestão nasal, coriza, diarreia, dor de cabeça, dor muscular, dor de garganta, febre, mal estar, perda de olfato, perda de paladar, sonolência, tosse, enjoo e

cansaço, totalizando 15 sintomas.

Sintomas	**Qtd. de Sintomas no *Dataset* 1**	**Qtd. de Sintomas no *Dataset* 3**
Calafrio	13	102
Cansaço	17	116
Congestão Nasal	8	139
Coriza	57	143
Diarreia	35	105
Dor de Cabeça	56	154
Dor de Garganta	15	136
Dor Muscular	27	116
Enjoo	59	106
Febre	26	124
Perda de Olfato	13	114
Perda de Paladar	18	105
Sonolência	7	101
Tosse	366	392
Mal estar	1	-
Total	**735 (em 556 *tweets*)**	**1953 (em 907 *tweets*)**

Tabela 1 – Número de sintomas presentes nos *Datasets* 1 (inicial) e 3 (final).

4.2 Aquisição dos *tweets*: *TweetSearcher*

A terceira atividade executa o *TweetSearcher* para extrair *tweets* da base de dados do *Twitter*, armazenando-os em nossa base de dados local. O *TweetSearcher* é um *script* em Python que se comunica com a API do *Twitter* através da biblioteca Tweepy[1]. O *script* realiza uma pesquisa cruzada entre os sintomas e palavras-chave relacionados à COVID-19, como mostrado na Figura 4.2. A API do *Twitter* pode filtrar palavras não relacionadas usando um lista de bloqueio (*blocklist*) ou até mesmo parâmetros de busca (*query parameters*). Filtramos *retweets* e *tweets* com imagens. Por fim, o *script* cria um arquivo no formato *JavaScript Object Notation Lines* (JSONL) para armazenar os *tweets* obtidos em um formato *Comma Separated Values* (CSV). A ferramenta *TweetSearcher* está disponível no repositório do GitHub[2].

[1] https://www.tweepy.org/
[2] https://github.com/rafaelstjf/tweets_searcher

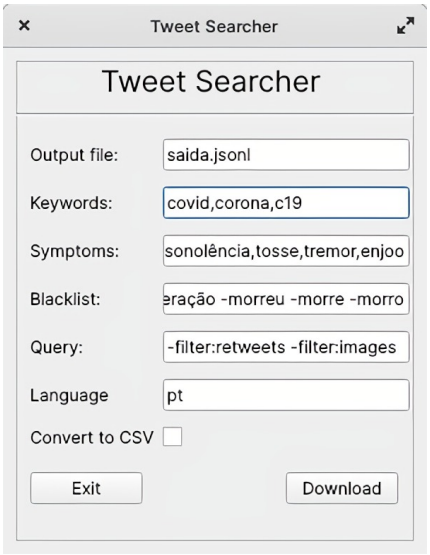


Figura 4.2 – Interface da ferramenta *TweetSearcher*

4.3 Primeira Análise de Dados: Verificação dos Sintomas de COVID-19

Ao menos três analistas de dados revisaram manualmente os *tweets* para identificar auto-relatos verdadeiros. Múltiplas camadas de filtro (atividades 4, 6 e 8) nos deram um conjunto de possíveis relatos de sintomas em casos positivos de COVID-19 (algumas centenas) reportados pelos usuários.

A atividade 4 verificou a presença de sintomas de COVID-19 nos *tweets*. Posteriormente foram removidos usuários da nossa base de dados nos casos em que:

1. Seus relatos em *tweets* foram julgados como sendo falsos;
2. Os relatos eram *retweets* de outros usuários;
3. Usuário afirmou que seus testes foram negativos, apesar de sua crença inicial, baseada nos sintomas, que o levou a crer que havia contraído o vírus (em um mesmo *tweet*).

Por exemplo, *tweets* como "**COVID-19: Brasil chega a 25,4 milhões de vacinados**" foram descartados por considerarmos que não há nenhum sintoma de COVID-19 em seu texto. Por fim, para descobrir usuários que auto-reportaram testes positivos para COVID-19, categorizamos os *tweets* em quatro conjuntos baseado na presença de sintomas:

1. *Tweets* sem sintomas;
2. *Tweets* com sintomas e teste positivo para a COVID-19 (esse é o nosso "Conjunto COVID-19 Positivo");

3. *Tweets* com sintomas e teste negativo;

4. *Tweets* com sintomas, mas sem teste.

4.4 Segunda Análise de Dados: Veriricação de Sintomas em *Tweets*

A atividade 6 identificou e excluiu *tweets* (identificados por técnicas de ML na atividade 5) que apresentaram palavras atípicas derivadas dos sintomas. Por exemplo, a palavra "sonoridade" foi excluída, pois não possui relação semântica com a palavra "sonolência". Importante observar que ambas palavras podem gerar a mesma subpalavra "sono", apesar de possuírem significados bastante diversos, o que gerou casos de confusão nas arquiteturas de Rede Neural utilizadas. Apesar disso, nem todas palavras com esse mesmo problema foram removidas, como a análise final dos resultados mostrará.

4.5 Terceira Análise de Dados: Balanceando a Frequência dos Sintomas nos *Tweets*

A partir da dificuldade das arquiteturas de Rede Neural em generalizar a identificação de sintomas com poucos casos durante a atividade 7, foram adicionados mais *tweets*, consequentemente mais aparições de sintomas, retirados do *Twitter* durante a atividade 8. A adição de novos dados se fez presente até que a quantidade mínima estabelecida de 100 aparições de cada sintoma fosse alcançada. Ao final, nosso *dataset* inicial (1) continha 556 *tweets* contando 735 sintomas, que foram aumentados no *dataset* final (3), composto por 907 *tweets* contendo 1.953 sintomas, como mostrado na Tabela 2.

	***Dataset* Inicial**	***Dataset* Final**
***Tweets* com sintomas**	556	907
***Tweets* sem sintomas**	3419	3414
Total	3975	4321

Tabela 2 – Comparação entre a quantidade de dados dos *datasets* inicial e final.

Como havia apenas um exemplo do sintoma "mal estar" , ele foi excluído de nossas análises. As atividades 6 e 8 foram repetidamente executadas até que a frequência de todos os sintomas estivesse balanceada, com ao menos 100 exemplos de cada.

Capítulo 5

Processamento dos Dados e Resultados

5.1 Processamento dos Dados

Diferentes redes BERT e uma GPT-2 foram utilizadas para identificar sintomas nas frases do conjunto de dados 3 obtido na seção anterior. A Figura 5.1 mostra um esquema descrevendo os passos para se executar a rede BERT, similares à todos os demais casos. Os passos são descritos em seguida.

Como as frases coletadas pela API do *Twitter* possuem um contexto diverso do pretendido pela rede neural (relatos de sintomas), o passo 1 consiste em uma limpeza dos dados usando expressões regulares (RegEx). Todos os *links*, nomes de usuário (iniciados com "@"), emojis, *hashtags* e "*RT*" foram removidos por serem próprios do contexto de *tweets*, mas não de um relato. Abreviações como "vc" no lugar de "você" não foram tratadas ou removidas devido à capacidade de redes *Transformer* de processá-las, mantendo assim a coerência e concordância originais das frases. Erros ortográficos também não foram tratados, pois são esperados em um contexto autêntico de relato escrito.

O passo 2 consiste em usar o tokenizador da rede para tokenizar cada dado, ou seja, cada frase. Esse processo separa cada palavra e confere se ela está presente no dicionário do tokenizador. Se a palavra estiver no dicionário, ela é substituída por seu *token* correspondente (um número identificador). Caso contrário, a palavra é separada em sub-palavras, que são substituídas por seus respectivos *tokens*. Existem também *tokens* especiais que indicam o início e o fim da frase tokenizada. A lista final de *tokens* deve ter 120 *tokens*. Caso esse número não seja alcançado, ao final da lista são adicionados *zeros* até que a lista que representa a frase tenha 120 *tokens*. Em outras palavras, a lista é truncada em 120 *tokens*. Apesar das redes BERT e GPT-2 serem capazes de computar até 512 e 1024 *tokens* por dado, respectivamente, limitamos esse valor para 120 devido às restrições no

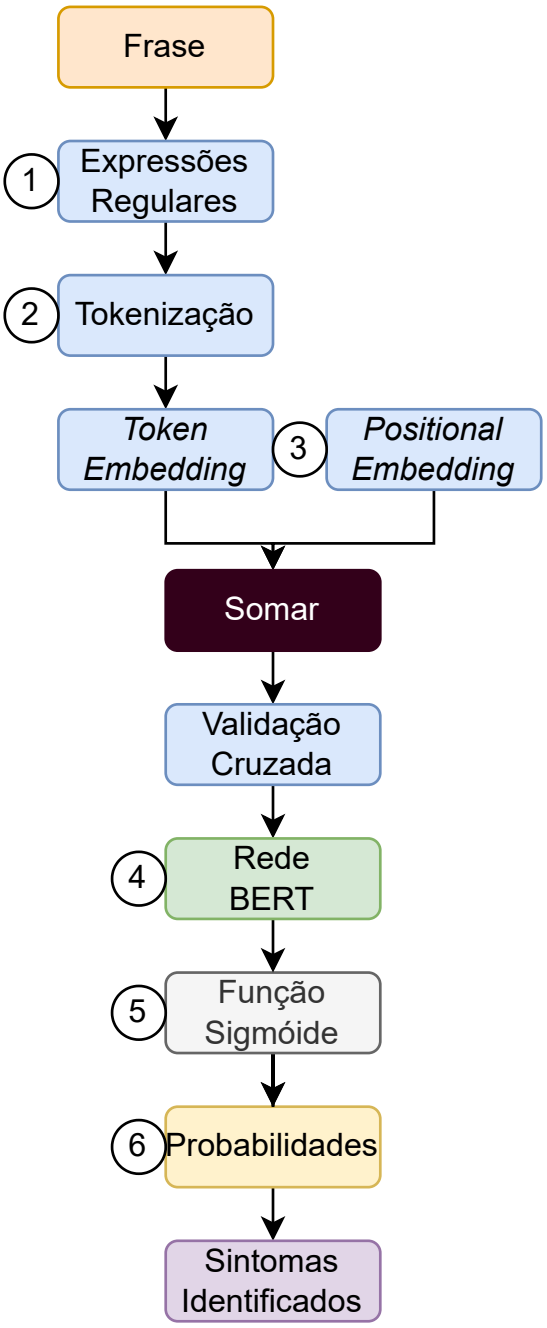


Figura 5.1 – Esquema descrevendo os passos a que toda frase do *dataset* final foi submetida para se identificar os sintomas presentes. Em 1, cada frase foi pré-processada usando Expressões Regulares (RegEx) para remover links, "*RT*", emojis, hashtgs e nomes de usuário. Em 2, a frase foi tokenizada pelo tokenizador da rede BERTimbau. Em 3, os tokens são mapeados pelos *Token Embedding* (TE) e *Positional Embedding* (PE) e as representações são somadas. A Validação Cruzada cria 5 (cinco) redes BERT independentes (4), cujos resultados são passados por uma função sigmóide (5), gerando probabilidade (6) que serão convertidas em uma lista de sintomas detectados no *tweet*.

tamanho da memória das nossas GPUs. Apesar disso, como *tweets* possuem um limite de 280 caracteres, os 120 *tokens* utilizados são suficientes para representar toda a informação

de um *tweet*.

No passo 3, temos o *Token Embedding* e o *Positional Embedding*. *Embedding* é o processo de representar uma informação usando um vetor com entradas numéricas. Se a informação é um *token*, chamamos de *Token Embedding* (TE); se estamos codificando uma posição, chamamos de *Positional Embedding* (PE). Cada *token* possui sua própria representação em *embedding* dada pela rede, *i.e.*, cada *token* é mapeado em um vetor de 512 dimensões pré-treinado para representar o "$significado$" do *token*. Ao TE adicionamos o PE para indicar o posição de cada *token* dentro da frase. O PE representa a informação sobre a posição de cada *token*, permitindo que a rede diferencie a distância entre os *tokens* dentro da frase.

Após o processo de *embedding*, o conjunto de dados é dividido pela Validação Cruzada e usado para alimentar a rede BERT, no exemplo da imagem (passo 4). A saída é um vetor com a dimensionalidade igual ao número de sintomas. Em seguida, cada entrada do vetor de saída passa por uma função sigmóide (passo 5), retornando um número entre 0 e 1, que pode ser interpretado como a probabilidade da presença de determinado sintoma na frase (passo 6). É necessário escolher um valor mínimo de probabilidade para que consideremos como indicativo da presença do sintoma; esse valor é chamado de limiar de escolha, já discutido anteriormente no presente trabalho. O limiar de escolha usado foi probabilidade ≥ 0.5. Por fim, o processo retorna uma lista de sintomas identificados em cada frase.

Durante o treinamento, foi utilizado o otimizador AdamW, o mesmo implementado originalmente pela BERT, com $\beta_1 = 0.9$, $\beta_2 = 0.999$ e decaimento de pesos do tipo L2 com valor de 0.01. Todos modelos foram treinado por 50 (cinquenta) épocas e foi usada uma parada antecipada e selecionado o melhor modelo no conjunto de validação em cada tarefa.

5.2 Testes de *Baseline*

Inicialmente foram treinadas 4 redes do tipo *Transformer* para avaliação do conjunto de dados, sendo elas:

1. BERT *Base Uncased*: versão tamanho base da rede BERT, pré-treinada em inglês;
2. BERTimbau: versão tamanho base da rede BERT pré-treinada em textos em português;
3. GPT-2: versão padrão da rede GPT-2, pré-treinada em inglês;
4. BERT *Base Multilingual Cased*: versão tamanho base da rede BERT pré-treinada nas 104 línguas com as maiores Wikipédias.

Em todos os casos foi utilizado um subconjunto dos dados de forma que o número de dados com sintomas fosse o mesmo dos dados sem sintomas. A escolha era feita pegando-se todos os dados com sintomas e buscando um número idêntico de dados sem sintomas no restante do conjunto de dados. A divisão dos dados em conjuntos de treino e teste ocorre após essa pré-seleção dos dados.

Todas as redes foram testadas em duas versões de treinamento: (1) treinamento normal; e (2) treinamento utilizando pesos para balancear os dados. O treinamento utilizando pesos altera levemente o cálculo da função perda da seguinte forma:

$$ECB_{perda} = \sum_{j=1}^{C} -k_j(y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)) \tag{5.1}$$

em que cada peso k_j é definido da seguinte forma:

$$k_j = \frac{1}{\sum_i^N y_j} \cdot \frac{N}{N_{sintomas}} \tag{5.2}$$

onde N é o número total de dados do conjunto de treinamento e $N_{sintomas}$ é o número de dados do conjunto de treinamento que possui algum sintoma. Dessa forma, os sintomas com menos aparições terão um k_j maior, incentivando a rede a aprender melhor a reconhecer tais sintomas.

Foi utilizado um valor padrão para a taxa de aprendizagem em todos os modelos $\varepsilon = 2 \cdot 10^{-5}$. Além disso, o tamanho do lote de dados para treinamento foi padronizado em 32 (trinta e dois), e o número de modelos da validação cruzada padronizado foi 5 (cinco).

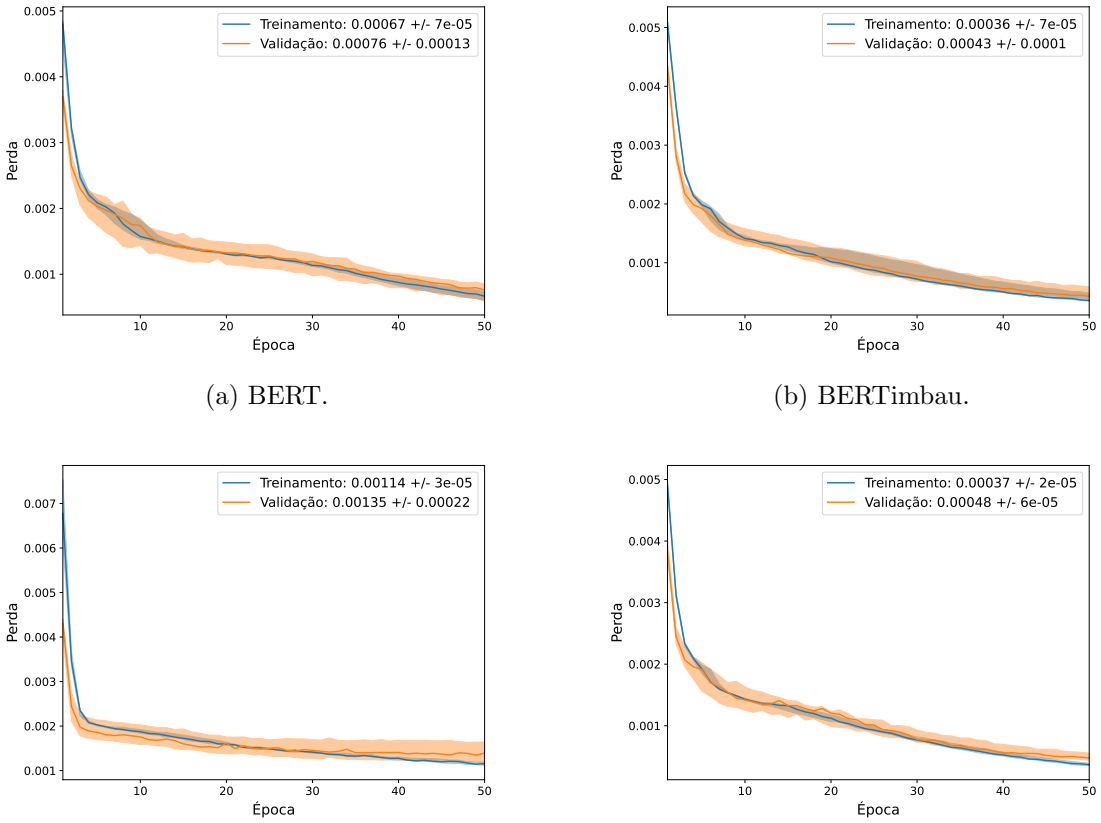
5.2.1 Resultados com balanceamento na função perda

Como foi mostrado na Tabela 1, existem sintomas que aparecem até quase 4 (quatro) vezes mais do que outros. Essa diferença na distribuição dos sintomas motivou o uso dos pesos na função perda para que a rede fosse capaz de aprender mesmo os sintomas com menor quantidade de aparições.

Como pode ser observado na Figura 5.2, todas as redes passaram por um processo de real aprendizado, pois tanto a perda no treinamento quanto na validação seguiram em queda durante todo o treinamento. Em outras palavras, não se observou sobreajuste ou subajuste, sendo um ótimo indicativo sobre a escolha das arquiteturas.

O mesmo não se observa quando analisamos a Figura 5.3, onde a evolução da acurácia nos dados de validação de todas as redes ficou abaixo de 90%. Aqui já é possível ver diferenças de performance entre as arquiteturas, sendo que as redes BERTimbau e BERT Multilingual foram as que apresentaram melhores resultados para a acurácia.

Outras métricas além da acurácia devem ser analisadas, sob risco de cairmos em uma visão distorcida sobre a qualidade das arquiteturas. Por exemplo, uma rede com alta



(a) BERT.

(b) BERTimbau.

(c) GPT-2

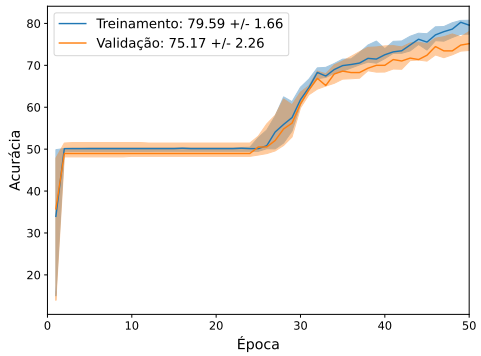
(d) BERT *Multilingual*

Figura 5.2 – Gráficos da evolução da perda durante o treinamento das redes neurais com a utilização de pesos.

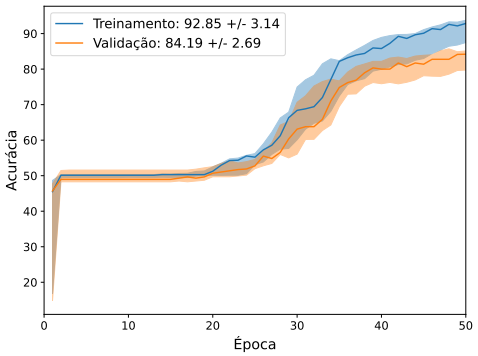
acurácia pode ter descartado todas as aparições de um sintomas com menos aparições. Assim, a rede obtém boa acurácia geral, mas péssimos resultados específicos para aquele sintoma.

A Figura 5.4 nos mostra que, com exceção da arquitetura GPT-2, todas as redes obtiveram AUC-ROC acima de 0.9 em todos os sintomas individualmente, levando-se em conta 3 casas decimais. As redes BERTimbau e BERT Multilingual continuam com os melhores resultados até então.

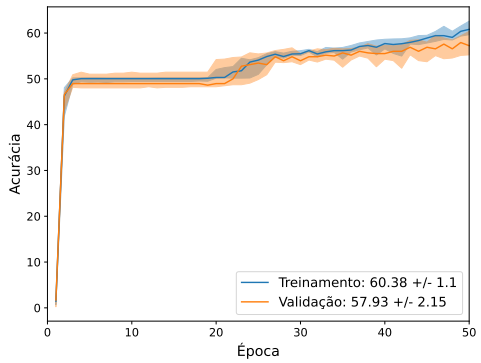
Apesar dos bons resultados até então, a Figura 5.5 nos mostra que os resultados obtidos não são tão confiáveis. Mesmo analisando apenas as melhores arquiteturas candidatas, há sintomas cuja precisão fica abaixo 70% em ambas, sugerindo que há espaço para aperfeiçoamento. Isso nos motiva a analisar os resultados sem a utilização de pesos durante o treinamento, para avaliarmos com mais propriedade se o uso dos pesos de fato ajudou o processo de treinamento ou não.



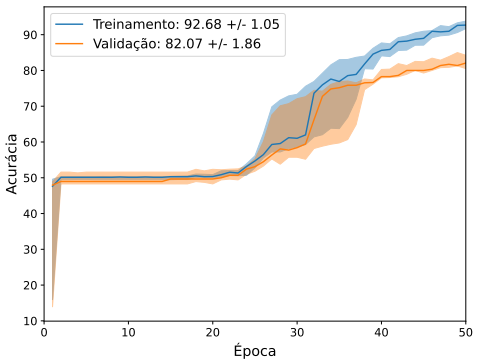
(a) BERT



(b) BERTimbau

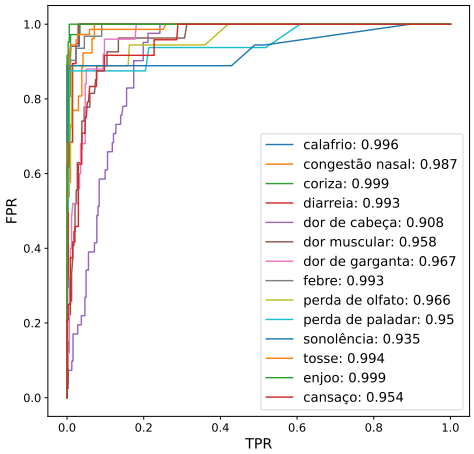


(c) GPT-2

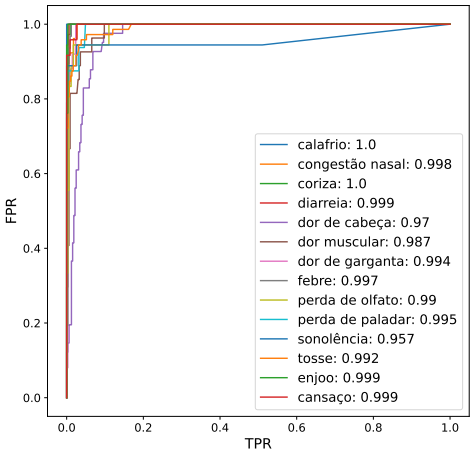


(d) BERT *Multilingual*

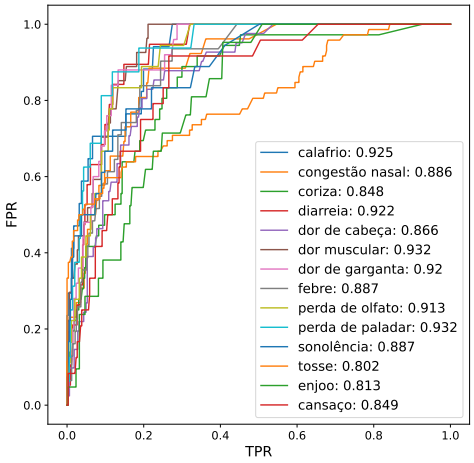
Figura 5.3 – Gráficos da evolução da acurácia durante o treinamento das redes neurais com a utilização de pesos.



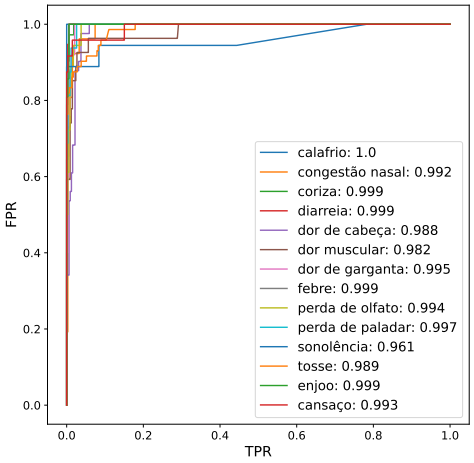
(a) BERT.



(b) BERTimbau.

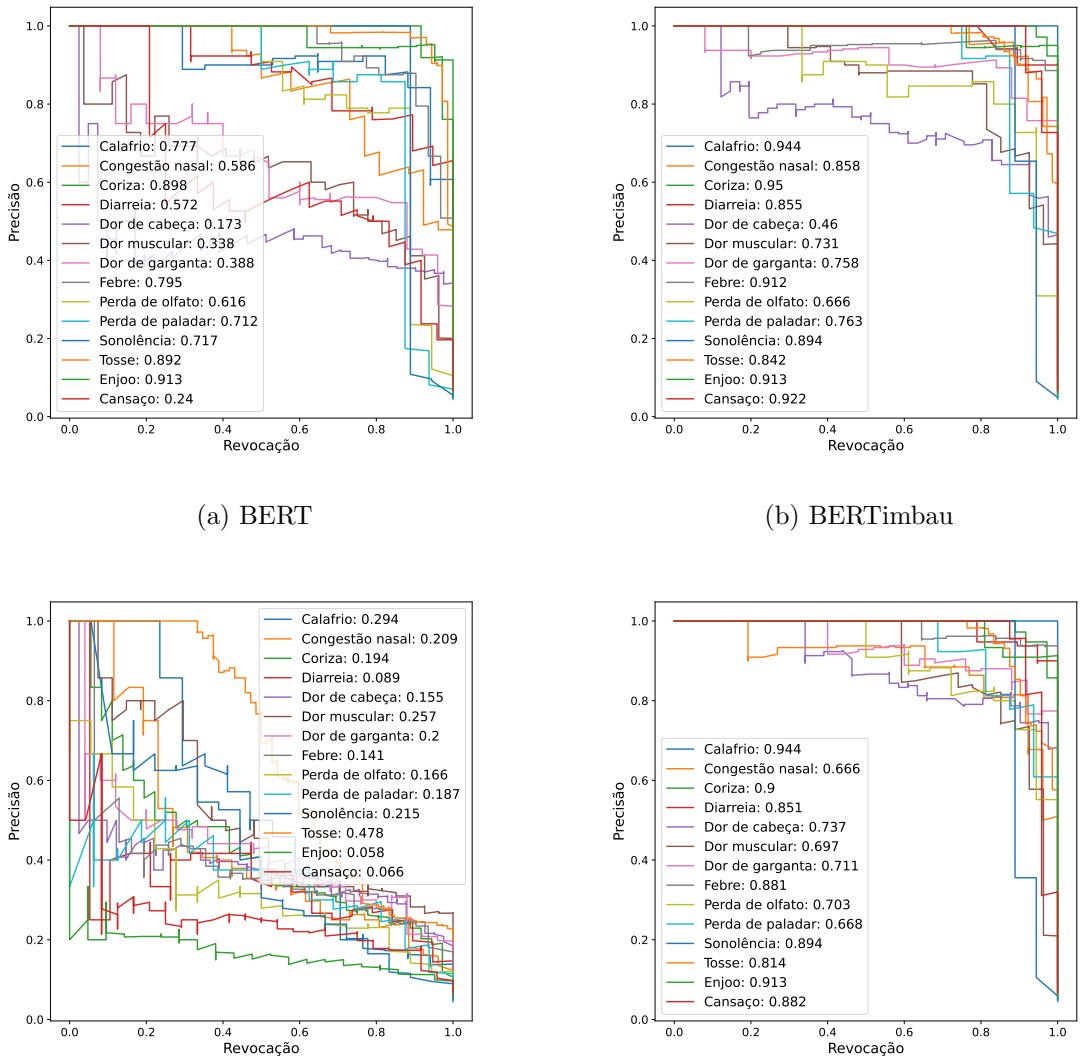


(c) GPT-2



(d) BERT *Multilingual*

Figura 5.4 – Gráficos das ROCs para cada sintoma e seus respectivos valores de AUC-ROC quando o treinamento utilizou pesos no cálculo da perda.



(a) BERT

(b) BERTimbau

(c) GPT-2

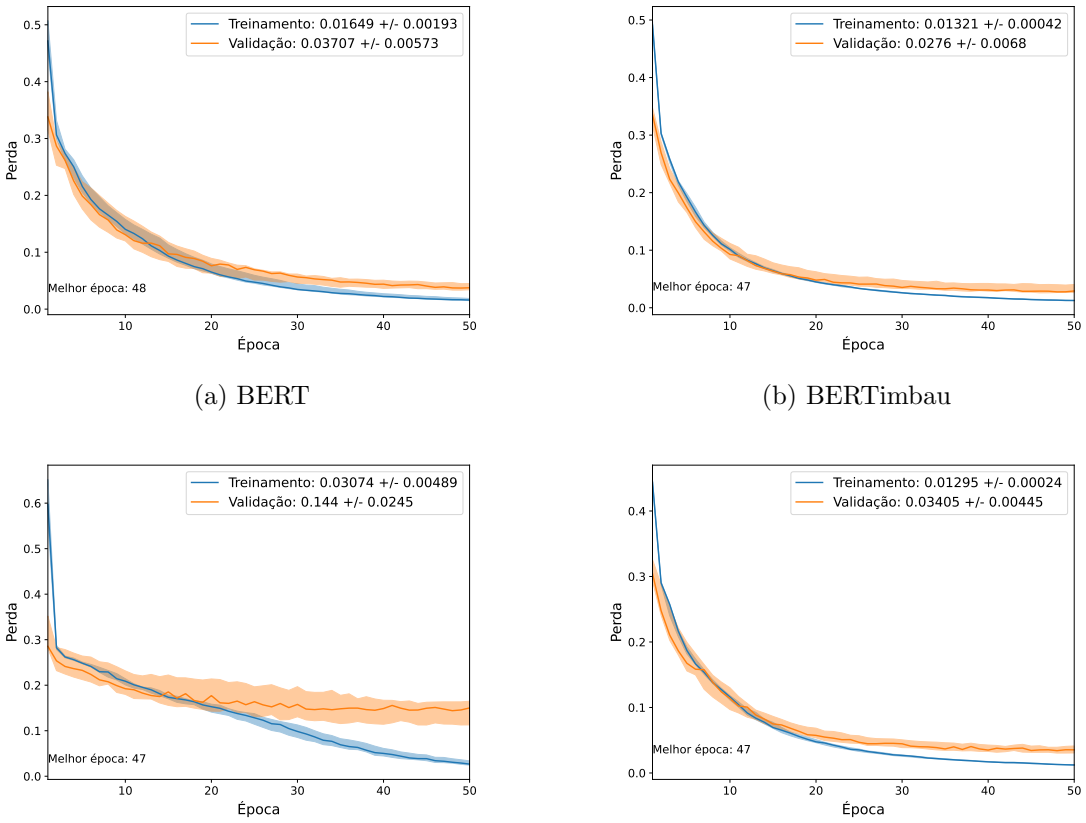
(d) BERT *Multilingual*

Figura 5.5 – Gráficos das Curvas de Precisão-Revocação para cada sintoma e seus respectivos valores de Precisão Média quando o treinamento utilizou pesos no cálculo da perda.

5.2.2 Resultados sem balanceamento na função perda

Todas as redes foram retreinadas do zero, dessa vez utilizando uma entropia cruzada binária sem balanceamento como função perda durante o processo de treinamento.

Similarmente ao caso utilizando balanceamento, a Figura 5.6 demonstra que o treinamento ocorreu sem grandes problemas em todas as arquiteturas, com exceção da GPT-2, que apresenta fortes sinais de sobreajuste aos dados de treino. Isso pode ser notado em seu gráfico que, por volta da época 10, passou a reduzir a perda apenas nos dados de treinamento, enquanto os de validação mantiveram sua perda praticamente constante.



(a) BERT

(b) BERTimbau

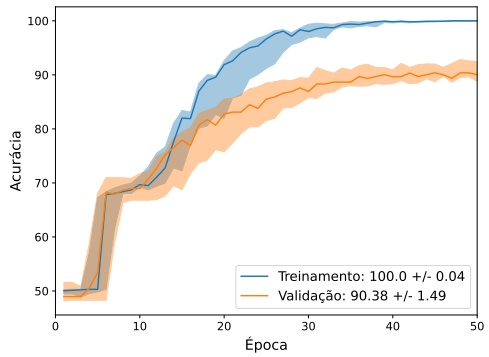
(c) GPT-2

(d) BERT *Multilingual*

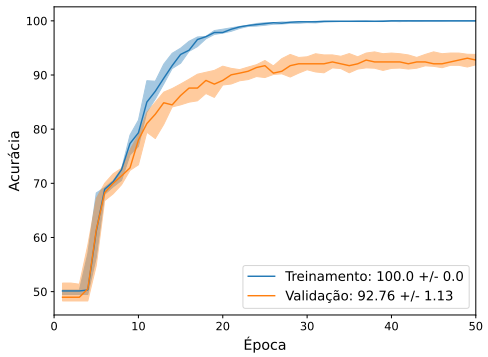
Figura 5.6 – Gráficos da evolução da perda durante o treinamento das redes neurais sem a utilização de pesos.

A Figura 5.3 reforça o sobreajuste da GPT-2, pois o gráfico da acurácia dos dados de treinamento segue em subida até o final do treinamento, enquanto o da validação fica estagnado abaixo dos 70% de acurácia. Todas as demais redes alcançaram acurácia nos dados de validação acima de 90%, indicando resultados melhores do que os obtidos quando balanceamento foi utilizado no treinamento.

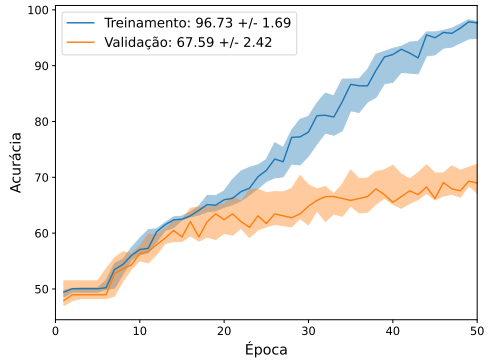
As curvas ROC da Figura 5.8 reforçam o mesmo entendimento de que todas as redes da família BERT foram bem-sucedidas na tarefa de classificação. Enquanto isso, a rede GPT-2 continua com um rendimento muito abaixo das demais.



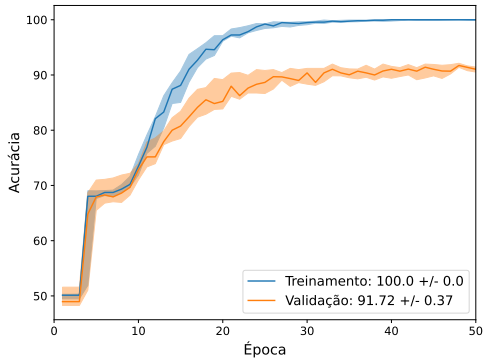
(a) BERT Base



(b) BERTimbau



(c) GPT-2



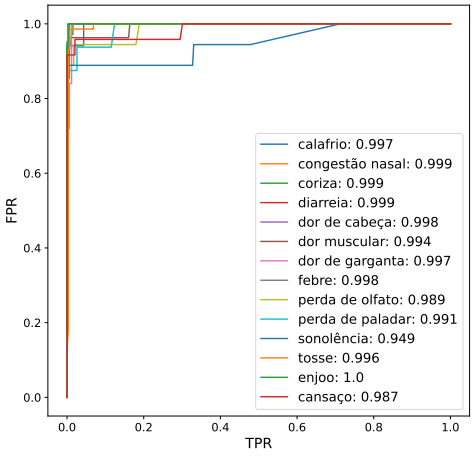
(d) BERT *Multilingual*

Figura 5.7 – Gráficos da evolução da acurácia durante o treinamento das redes neurais sem a utilização de pesos.

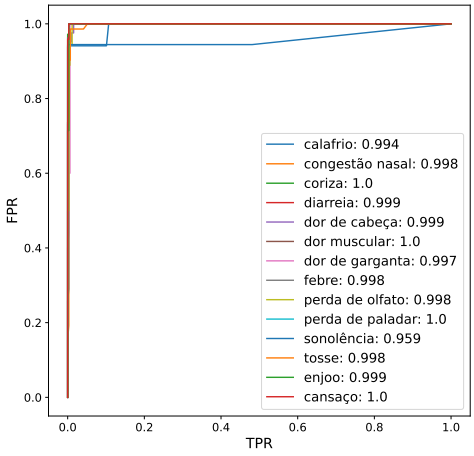
Por fim, as curvas de Precisão-Recall apresentadas na Figura 5.9 confirmam os resultados anteriores obtidos pelas demais métricas. Todas as métricas obtidas para o treinamento sem uso de pesos indicam que esse processo se mostrou mais eficaz do que o uso de pesos, contrariando a expectativas iniciais.

Resta agora escolher entre um dos modelos analisados para dar continuidade às análises e aplicações pretendidas, como o *chatbot*. A arquitetura GPT-2 é a primeira a ser descartada devido a seus resultados muito aquém das demais. Entre os 3 (três) modelos restantes, o BERT Multilingual tem a desvantagem de possuir o maior dos vocabulários, sendo cerca de 4 (quatro) vezes o tamanho dos vocabulários dos demais modelos. Como a maior parte desse vocabulário não é referente ao português (já que o modelo foi treinado em 104 línguas), muitos dos *tokens* presentes nele nunca serão usados em nosso contexto, como os *tokens* referentes ao mandarim, por exemplo, que são retirados do texto durante a limpeza por RegEx. Esse vocabulário maior faz com que o treinamento e aplicação da rede sejam mais custosos computacionalmete do que as alternativas, motivando sua preterição.

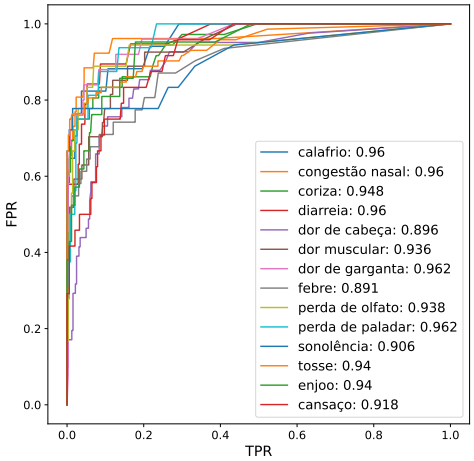
Por fim, nos resta escolher entre a BERT Base e a BERTimbau. Pensando em futuras expansões do conjunto de dados e sintomas a serem detectados, a BERTimbau possui



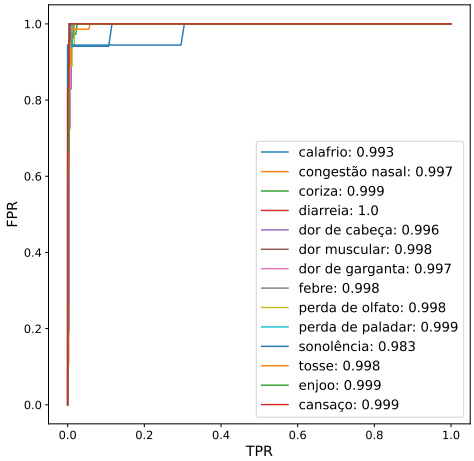
(a) BERT Base



(b) BERTimbau



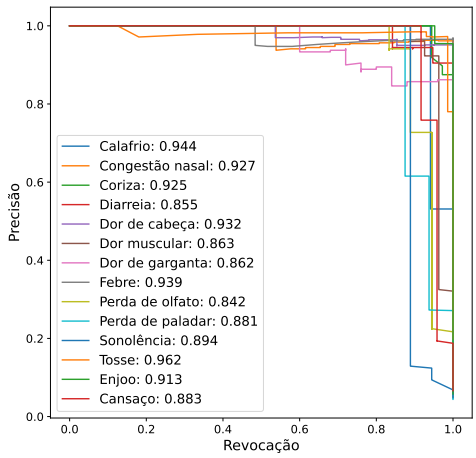
(c) GPT-2



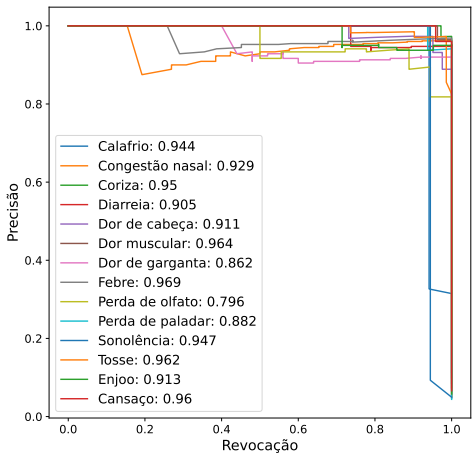
(d) BERT *Multilingual*

Figura 5.8 – Gráficos das ROCs para cada sintoma e seus respectivos valores de AUC-ROC quando o treinamento não utilizou pesos no cálculo da perda.

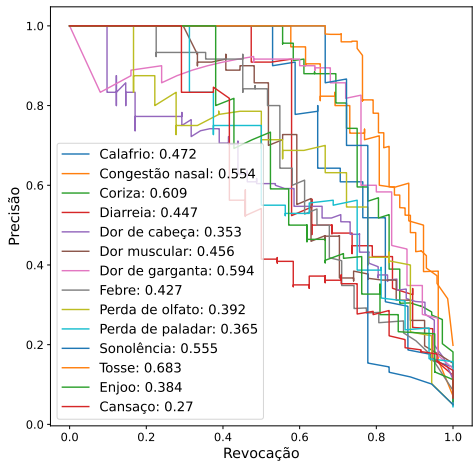
a vantagem de ter seu dicionário treinado em português, o que pode ser um indicativo de maior capacidade de generalização. Por esses motivos, as análises mais detalhadas e testes utilizando o LIME serão feitos com a BERTimbau.



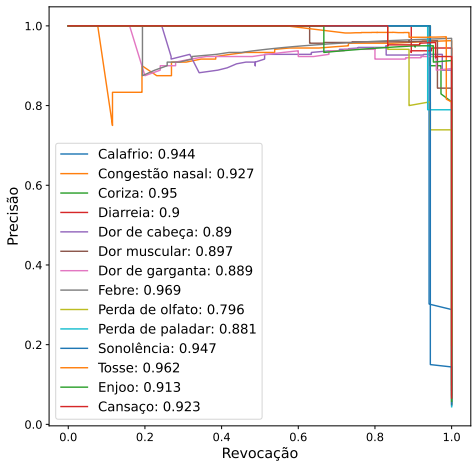
(a) BERT Base



(b) BERTimbau



(c) GPT-2



(d) BERT Multilingual

Figura 5.9 – Gráficos das Curvas de Precisão-Revocação para cada sintoma e seus respectivos valores de Precisão Média quando o treinamento não utilizou pesos no cálculo da perda.

5.3 Resultados do Modelo BERTimbau e Discussão

Recapitulando, o treinamento do modelo convergiu, pois o valor da perda dos dados de treino e de validação decaiu até estacionar em um platô, como demonstrado na Figura 5.10. A mediana da perda em relação aos *folds* foi usada para selecionar a melhor época para ser usada nos dados de teste, ou seja, a época em que o valor da perda nos dados de validação é mínima.

Além de um valor de acurácia geral de 93.11 ± 0.10 (Figura 5.11), o modelo alcançou altos valores de AUC-ROC e precisão média em todos os sintomas nos dados de teste, como observado na Figura 5.12. Tais resultados indicam que o modelo é altamente eficaz em diferenciar a presença de sintomas em *tweets* e gera poucos falsos positivos. Além

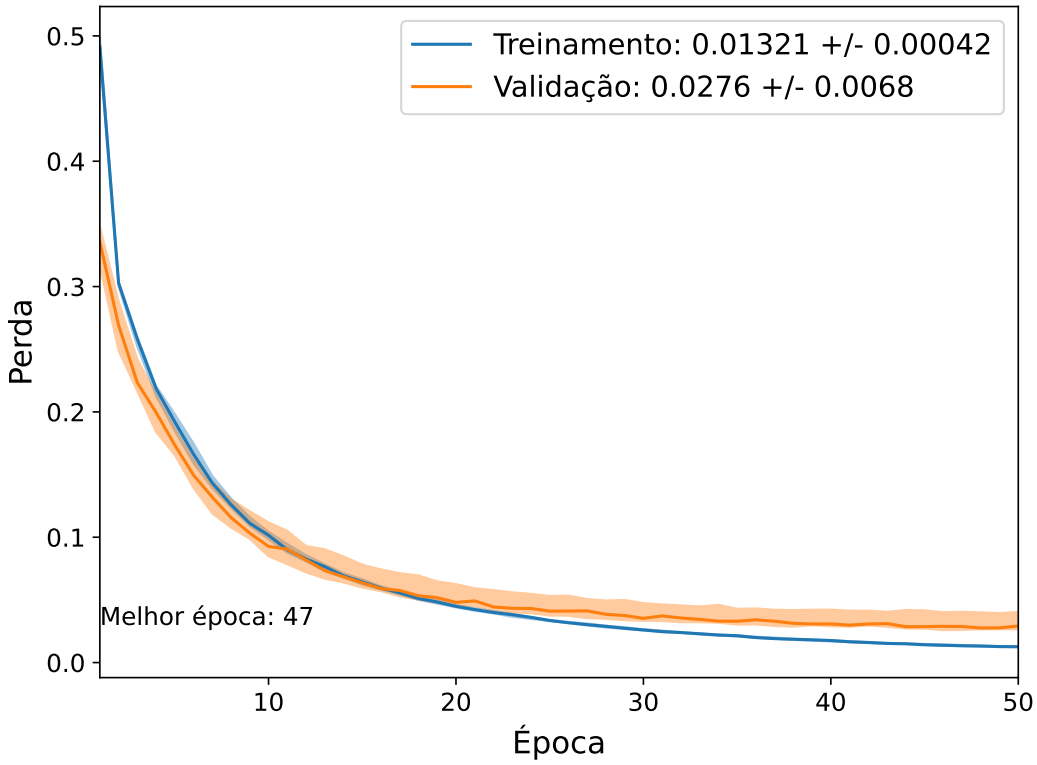


Figura 5.10 – Evolução da perda durante o treinamento da rede BERTimbau escolhida.

disso, a Tabela 3 mostra que os resultados através dos *folds* da Validação Cruzada são altamente estáveis em todos os sintomas.

Como alguns sintomas podem ser descritos pelos usuários usando diferentes sinônimos, resultados levemente menores de precisão e acurácia nos resultados são esperados, *e.g.*, em "congestão nasal", "dor de cabeça"e "dor de garganta". O modelo aprendeu as formas mais comuns de descrevê-los, mas alguns sinônimos mais raros não foram generalizados, levando a uma precisão levemente mais baixa. O maior problema se refere aos usuários usando diferentes palavras para expressar o que estão sentindo. Essas diferenças podem ser regionais ou mesmo por idade, reforçando a necessidade de coletar o mais diverso e completo *dataset* possível para se realizar o treinamento.

Um problema diferente aconteceu com "perda de olfato" e "perda de paladar": poucos tweets tinham um dos sintomas presentes sem a presença do outro. Consequentemente, o modelo criou uma correlação entre os dois sintomas, levando a um decréscimo em ambas precisões.

O sintomas *sonolência* teve a menor AUC-ROC, provavelmente devido à presença, no conjunto de dados, de palavras muito similares na escrita, mas cujo significado é bastante diverso, tais como *sonoro* e *sonolência*. Como suas precisões podem demonstrar, o modelo foi capaz de identificar casos reais de *sonolência*, mas a AUC-ROC indica que a certeza no

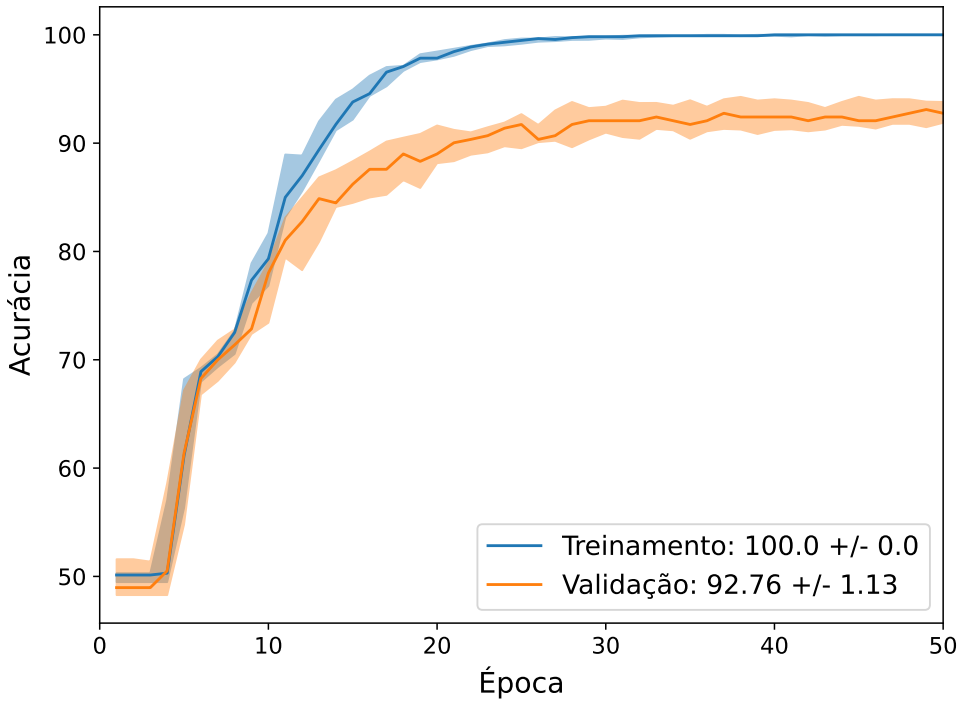


Figura 5.11 – Evolução da acuráica durante o treinamento da rede BERTimbau escolhida.

resultado final poderia ser maior. Apesar deste problema poder ser minimizado criando um dicionário mais especializado para a tarefa, o problema provavelmente é consequência do fato do conjunto de dados usado para treinamento ser bem diferente do contexto de aplicação imaginado. De fato, em um contexto de relato, a presença de palavras que são tão distantes do contexto clínico é menos esperado.

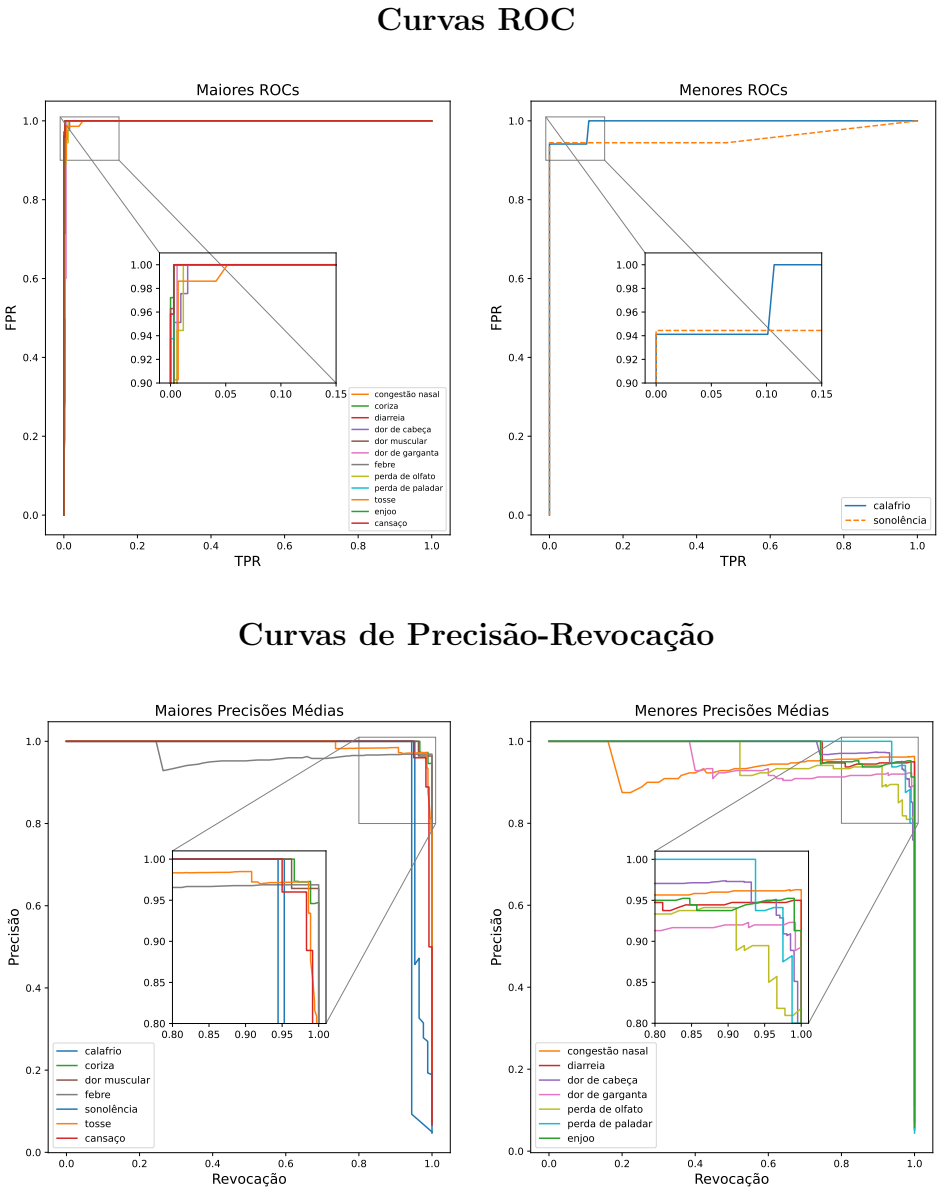


Figura 5.12 – Curvas ROC e de Precisão-Recall para cada sintoma na rede BERTimbau.

	ROC-AUC	Precisão Média
Calafrio	0.99388 ±0.00304	0.944 ±0.010
Cansaço	0.99988 ±0.00017	0.960 ±0.001
Congestão Nasal	0.99755 ±0.00084	0.927 ±0.007
Coriza	0.99992 ±0.00003	0.950 ±0.005
Diarreia	0.99924 ±0.00065	0.905 ±.008
Dor de Cabeça	0.99871 ±0.00040	0.910 ±0.014
Dor de Garganta	0.99710 ±0.00029	0.862 ±0.011
Dor Muscular	0.99989 ±0.00003	0.964 ±0.024
Enjoo	0.99916 ±0.00034	0.913 ±0.007
Febre	0.99781 ±0.00049	0.969 ±0.001
Perda de Olfato	0.99791 ±0.00032	0.842 ±0.020
Perda de Paladar	0.99982 ±0.00003	0.882 ±0.030
Sonolência	0.95886 ±0.00019	0.947 ±0.010
Tosse	0.99821 ±0.00012	0.951 ±0.002

Tabela 3 – Meadiana e desvio-padrão da AUC-ROC e da Precisão Média por sintoma. Todas as métricas foram medidas no *dataset* de teste.

5.4 *Local Interpretable Model-Agnostic Explanations* (LIME)

LIME é um método usado para interpretar decisões individuais de um modelo ao invés de buscar uma explicação global. LIME faz isso executando o modelo diversas vezes, mostrando diferentes partes da mesma frase (no caso de dados de texto) em cada rodada. Dessa forma, LIME consegue identificar as palavras mais importantes usadas pelo modelo para classificação e retorna uma estimativa da confiança sobre as explicações dadas.

LIME auxilia a encontrar viéses na classificação a partir das palavras marcadas e confiança estimada, permitindo melhorias na adequação do modelo às informações presentes no texto (no nosso caso, os tweets que descrevem sintomas de COVID).

Usar o LIME para analisar predições incorretas pode ser uma fonte de ideias para futuras melhorias, como observado na Figura 5.13. Os exemplos apresentados na figura sugerem que o modelo possui viéses inerentes aos dados. Em (a), apesar da presença da palavra "entupido", que o LIME identifica como altamente importante, o modelo falha em retornar a presença do sintoma "congestão nasal". Isso é provavelmente uma consequência da falta de mais exemplos conectando a palavra ao sintoma em nosso *dataset*.

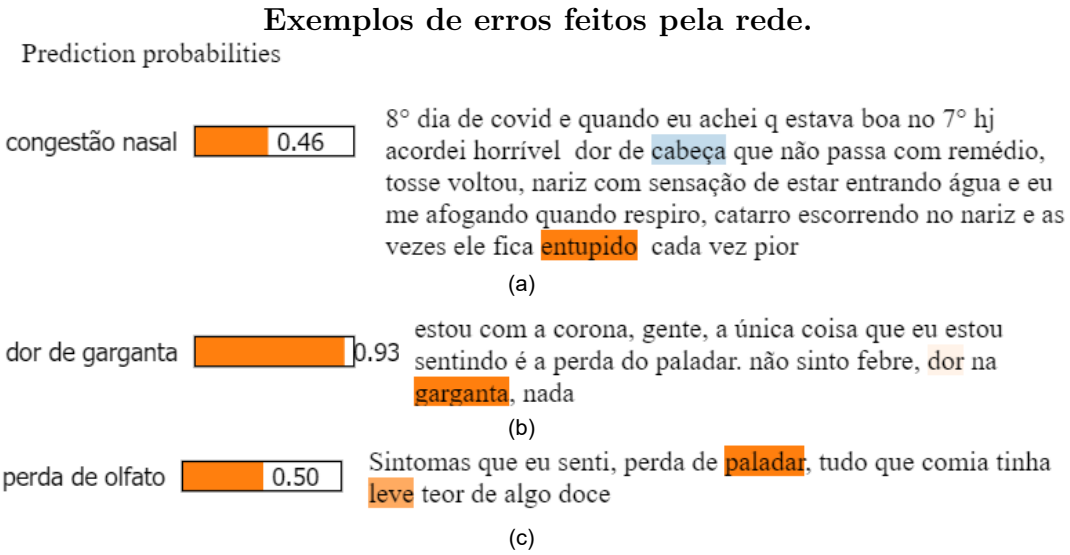


Figura 5.13 – Modelo LIME aplicado à exemplos de erros cometidos pela rede no *dataset* de teste. (a): Caso de falso negativo; (b): Caso de falso positivo; (c): Caso de falso positivo.

Em (b), o problema é diferente, pois o modelo não foi treinado com muitos exemplos que descreveram a ausência de sintomas. Portanto, todas as palavras são usadas para indicar a presença da condição clínica, ignorando situações que indiquem sua ausência. Considerando o objetivo deste modelo, essa é uma limitação fundamental. Uma possível forma de minimizar tal problema em uma aplicação na vida real seria um alerta para que os usuários nunca descrevam um sintoma que eles não estejam sentindo, restringindo-se aos sintomas que eles identificaram em si mesmos. Futuros trabalhos também podem

explorar a capacidade de modelos do tipo *Transformer* de generalizar seu entendimento de quando uma frase descreve a ausência de um sintoma.

Em (c), temos um problema de correlação, porque o modelo usa a descrição de "perda de paladar" para indicar "perda de olfato". Uma exploração no conjunto de dados é benéfico para identificar que, em quase todos os casos, esse dois sintomas aparecem juntos, confundindo o modelo quando ele tiver que analisar um caso em que apenas um deles aparece.

O LIME provê uma ferramenta para entender como um modelo identifica sintomas e aponta situações onde faltam dados de treinamento para o modelo. Um modelo preditivo pode falhar em casos sub-representados nos dados de treino. O LIME nos deu dois grandes entendimentos usando apenas os três exemplos ilustrados e outros similares a eles. Primeiramente, a necessidade de mais dados para que o modelo possa generalizar sua habilidade de identificar sintomas. Mais exemplos de formas alternativas de descrever o mesmo sintomas e novas frases que descrevam "perda de paladar" e "perda de olfato" separadamente seriam necessárias para tal, como indicado na Figura 5.13 (a) e (c). Em segundo lugar, é essencial assegurar que o modelo não seja usado em descrições negativas de sintomas. Apesar deste limitação, o modelo segue relevante em auxiliar a identificar pacientes com risco de COVID-19, já que a limitação envolve falsos positivos, e não falsos negativos.

5.5 Aplicação em *Chatbot*

Seguindo a motivação inicial do trabalho, a criação de uma ferramenta de detecção automatizada de sintomas, foi construído um protótipo de *chatbot*. A rede BERTimbau treinada sem balanceamento foi integrada ao seu funcionamento. Ao buscar por uma interface simples e que não gerasse dificuldades para seus usuários, optamos pelo uso do aplicativo de mensagens *Telegram*. O *chatbot* foi construído usando-se a biblioteca *python-telegram-bot*[1] e uma API oficial do *Telegram*[2].

Nosso *chatbot* funciona também como uma forma de validação do modelo BERTimbau treinado e ferramenta de construção de um novo *dataset* para treinamento. Em sua versão atual, o *chatbot* salva 5 (cinco) informações em uma nova linha de um arquivo CSV ao final da interação com um humano. Essas informações são:

- Dado 1: Dia e hora do final da interação do usuário com o *chatbot*.
- Dado 2: Mensagem do usuário descrevendo seus sintomas;
- Dado 3: Lista de sintomas que a rede BERTimbau treinada identificou;

[1] https://github.com/python-telegram-bot/python-telegram-bot
[2] https://core.telegram.org/bots/api

- Dado 4: Lista de sintomas corrigida pelo usuário;
- Dado 5: Texto livre enviado pelo usuário descrevendo outras condições de saúde.

Com esses dados, é possível avaliar a capacidade de generalização da rede BERTimbau com dados gerados diretamente pelos usuários. Uma comparação entre os dados 3 e 4 seria o suficiente para tal, pois seria a comparação entre o resultado da rede BERTimbau com a lista de sintomas corrigida pelo próprio usuário. Casos que mostrassem erros da rede neural nessa comparação podem ser posteriormente avaliados indvidualmente com o auxílio de técnicas como o LIME.

As mensagens dos usuário descrevendo seus sintomas (dado 2) em conjunto com a lista de sintomas corrigida pelo usuário (dado 4) servem como um novo *dataset*. Se este for alimentado até que cada sintoma possua, ao menos, 100 (cem) aparições, poderia ser usado para treinar novas redes neurais. Por fim, a interação com o *chatbot* gera dados textuais (dados 2 e 5) que fornecem informações importantes sobre outros sintomas de COVID-19 que podem estar relacionados a diferentes fases da doença ou diferentes variantes, que poderiam ser analisados por profissionais da saúde. Esperamos que nosso *chatbot* e a tecnologia por trás dele possa servir de inspiração para o acompanhamento de outras doenças.

5.5.1 Funcionamento do *chatbot*

O fluxograma do funcionamento do *chatbot* pode ser visto na Figura 5.14. Para iniciar a conversa, basta que o usuário mande uma mensagem escrita "*/start*". Ao receber esse comando, o *chatbot* requisita que a pessoa descreva quais sintomas está sentindo em apenas uma mensagem, e aguarda pela resposta, como mostrado na Figura 5.15. Ao receber uma resposta, uma cópia do texto da resposta é salvo em uma variável separada chamada "mensagem"(dado 2), que não será mais atualizada durante toda a interação.

O texto da mensagem passa pelos seguintes processamentos no *backend* do *chatbot*:

1. Tokenização do texto;
2. Processos de *Token Embedding* e *Positional Embedding*, e sua soma;
3. Processamento pela rede BERTimbau treinada;
4. Função sigmóide final para o cálculo das probabilidade;
5. Uso do limiar de escolha padrão (probabilidade do sintoma ≥ 0.5) para gerar a lista de sintomas identificados.

O resultado final desse processamento é uma lista de sintomas que é salva em duas variáveis diferentes: a lista de "sintomas identificados" (dado 3) e a lista de "sintomas

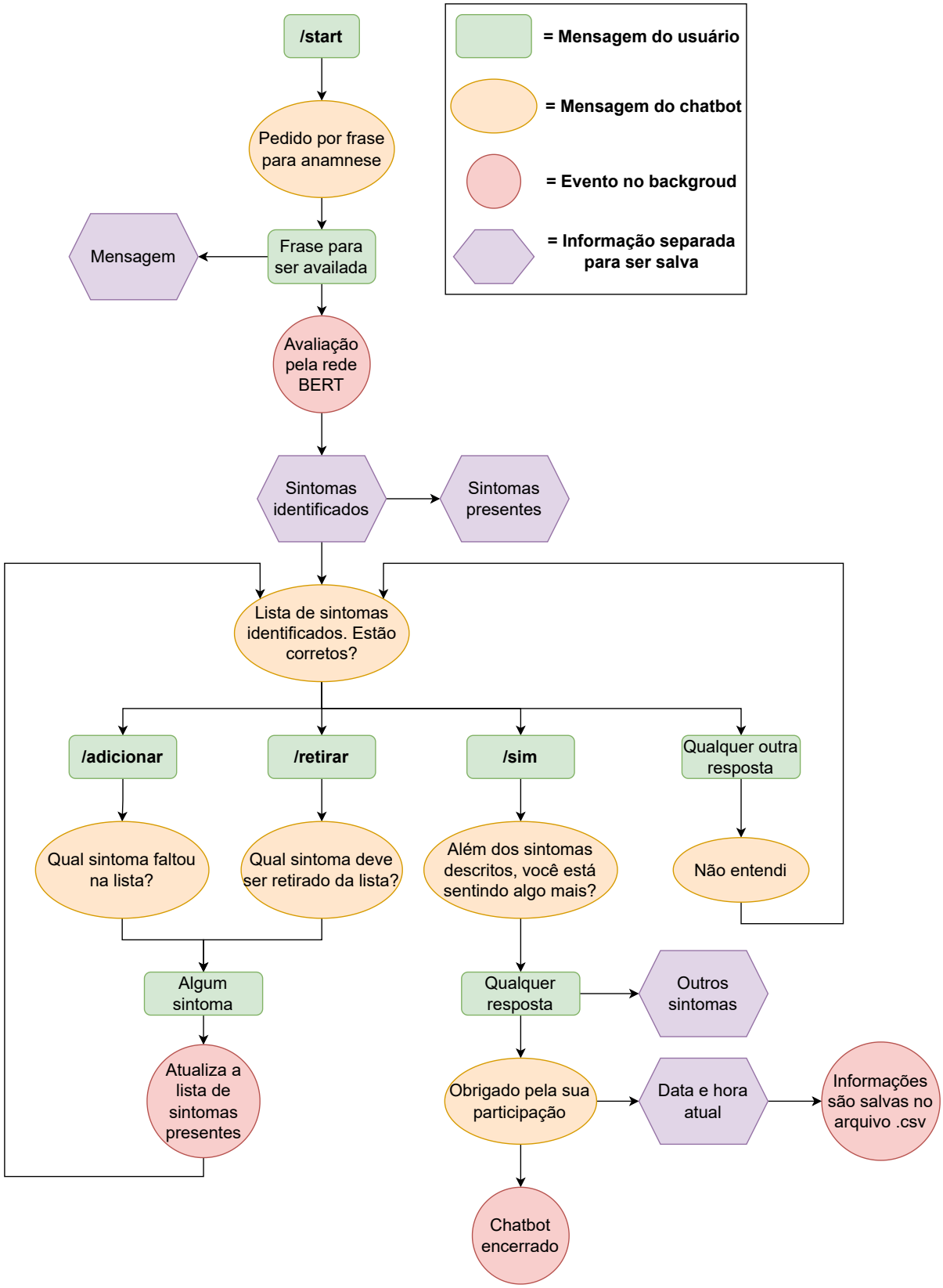


Figura 5.14 – Fluxograma do funcionamento do *chatbot*.

presentes" (dado 4). Importante ressaltar que nesta fase do fluxo do *chatbot* essas variáveis ainda não foram salvas no novo *dataset* que está sendo criado. Elas serão salvas apenas ao final, quando a interação do usuário com o *chatbot* terminar. A partir deste ponto, a

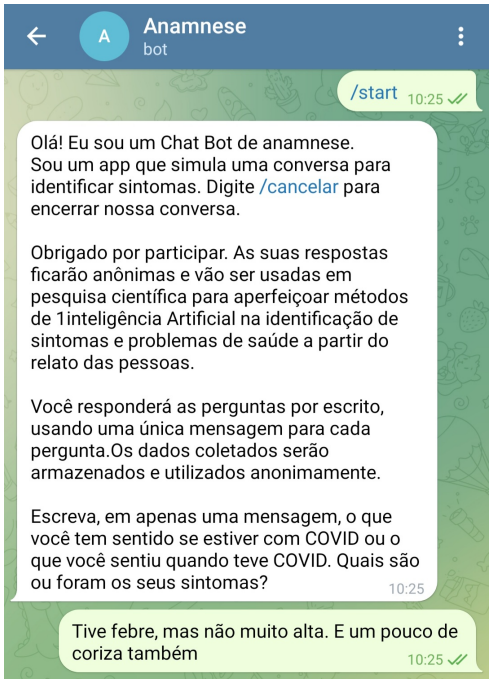


Figura 5.15 – Exemplo do início de uma interação com o *chatbot*.

variável "sintomas identificados"(dado 3) não será mais atualizada.

A lista de "sintomas presentes"(dado 4) é então apresentada para o usuário, como mostrado na Figura 5.16, e pede-se para ele optar entre:

- Adicionar um sintoma que foi descrito em sua mensagem, mas que não está presente na lista de sintomas, usando o comando "*/adicionar*";
- Retirar um sintoma que não foi descrito em sua mensagem, mas que está presente na lista de sintomas, usando o comando "*/retirar*";
- Confirmar que a lista de sintomas apresentada está de acordo com o descrito em sua mensagem, usando o comando "*/sim*";

Caso o usuário responda algo que não seja um dos 3 (três) comandos descritos acima, o *chatbot* simplesmente responde "não entendi" e repete ao usuário o pedido anterior de optar entre os comandos.

Caso o usuário responda com o comando "*/adicionar*", ele é apresentado à lista dos 14 (catorze) sintomas que a rede neural é capaz de identificar e pede-se para que ele identifique qual destes sintomas a rede neural não foi capaz de identificar, como mostrado na Figura 5.17. Ao receber a resposta do usuário com um sintoma válido, a lista de "sintomas presentes" é atualizada. Se o usuário escrever qualquer coisa que não seja um sintoma válido, a lista mantem-se inalterada. Repetir um sintoma que já estava presente na lista também não a altera de nenhuma forma. Após a atualização, o usuário recebe a lista de "sintomas presentes" (dado 4) atualizada e pede-se para ele optar novamente entre "*/adicionar*", "*/retirar*" ou "*/sim*".

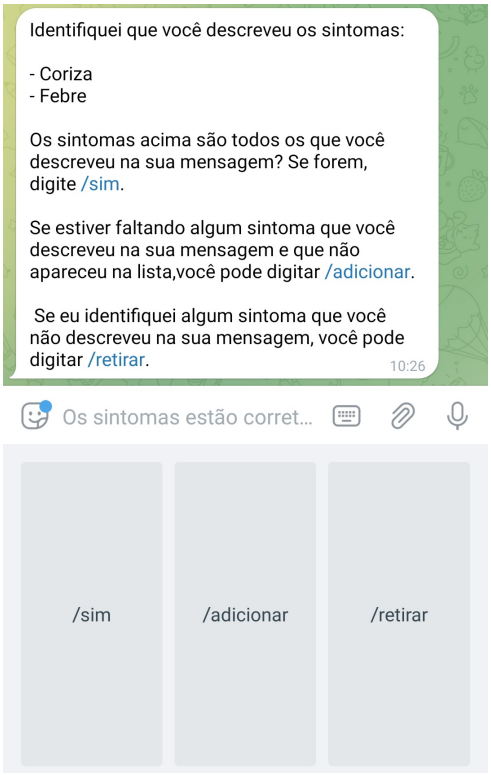


Figura 5.16 – Exemplo de resposta do *chatbot* com sintomas identificados.

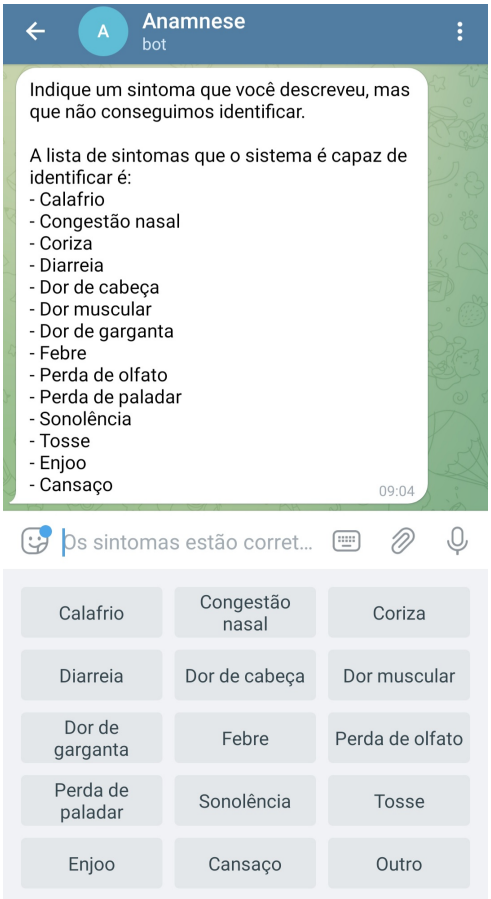


Figura 5.17 – Exemplo de como adicionar um sintoma não identificado pelo *chatbot*.

O caso em que o usuário responde com "*/retirar*" é similar ao caso anterior, com a diferença que o *chatbot* pede que seja identificado um sintoma erroneamente colocado na lista de sintomas presentes, como pode ser observado na Figura 5.18. Nesse caso, se o usuário pedir para retirar um sintoma que não estava na lista, nada ocorre à lista. O processo de adicionar ou retirar um sintoma à lista de sintomas presentes pode ser feito quantas vezes o usuário desejar, e a única variável sendo atualizada é a lista de "sintomas presentes"(dado 4).

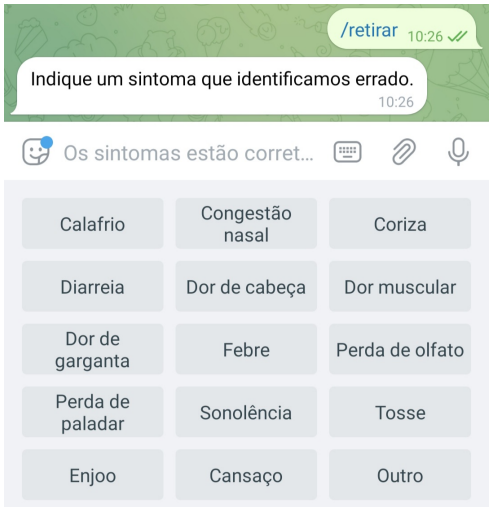


Figura 5.18 – Exemplo de como retirar um sintoma identificado erroneamente pelo *chatbot*.

Finalmente, ao digitar o comando "*/sim*", o *chatbot* pergunta se o usuário está sentindo mais alguma coisa além do descrito, podendo incluir sintomas que a rede neural não foi treinada para identificar, como demonstrado na Figura 5.19. A resposta para esse pergunta é salva na variável "outros sintomas" (dado 5). Por fim, o *chatbot* agradece pela participação do usuário e encerra a conversa. Nesse momento, é salvo o dia e horário do fim da conversa em uma variável "dia e horário atual" (dado 1).

Com a conversa encerrada, as 5 (cinco) variáveis separadas pelo sistema são salvas em uma nova linha de um arquivo CSV previamente criado. As variáveis, na ordem em que são salvas, são:

1. Dia e horário do final da conversa;
2. Mensagem;
3. Sintomas presentes;
4. Sintomas identificados;
5. Outros sintomas.

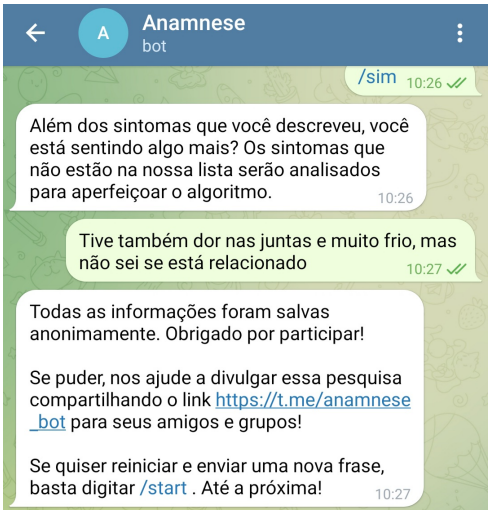


Figura 5.19 – Exemplo do final de um interação com o *chatbot*.

Com essas informações, é possível criar um novo dataset para retreinar a rede neural BERTimbau usando as colunas “mensagem” e “sintomas presentes”. Por outro lado, pode-se também utilizar as colunas “sintomas presentes” e “sintomas identificados” para entender quais as situações em que a rede neural está errando, possibilitando assim identificar viéses da aplicação.

Capítulo 6

Considerações Finais

Com a chegada da pandemia de COVID-19, a necessidade de oferecer tratamento médico minimizando o risco de novas infecções criou uma nova motivação para pesquisas. Iniciativas envolvendo identificação de sintomas da COVID-19 utilizando inteligência artificial foram feitas principalmente para a língua inglesa. O objetivo do presente trabalho foi construir uma rede neural do tipo *Transformer* para texto capaz de detectar a presença de 14 sintomas de COVID-19 escolhidos por profissionais da saúde. Para esse objetivo, foi utilizado um *dataset* minerado diretamente do *Twitter* para treinar a rede neural e os resultados foram analisados usando-se o método LIME.

Como deseja-se que a rede neural seja capaz de processar textos reais, o uso de um *dataset* advindo do *Twitter* possui a vantagem de ter seus textos produzidos por pessoas escrevendo de forma informal, ou seja, sem uso de jargões ou estruturas de escrita demasiadamente formais. Dessa forma, regionalismos, gírias, abreviaturas e erros de gramática ou escrita são preservados, permitindo que a rede neural seja treinada em um contexto próximo ao da aplicação pretendida, uma anamnese automatizada.

O uso de redes *Transformer*, especialmente as do tipo BERT, se mostrou altamente eficiente para a detecção de sintomas de COVID-19 em texto. Entre as arquiteturas testadas, os melhores resultados foram os da rede BERTimbau treinada com uma função perda do tipo Entropia Binária Cruzada sem balanceamento. A rede BERTimbau apresentou acurácia acima de 97%, AUC-ROC acima de 0.95 e Precisão Média acima de 0.84 em cada um dos 14 sintomas. Os casos com previsões equivocadas da rede neural foram interpretados utilizando-se o método LIME, que permitiu identificar que a expansão do *dataset* é essencial para elevar as métricas obtidas. Por fim, integramos a rede com sucesso a um *chatbot* montado no aplicativo de mensagens *Telegram*. O *chatbot* desenvolvido possui uso interativo e amigável, sendo capaz de entregar respostas processadas pela rede BERTimbau em tempo real. O *chatbot* está funcional e disponível para uso[1], servindo

[1] https://t.me/anamnese_bot

para validação da presente dissertação e para a criação de novos *datasets* relacionados à COVID-19.

O enriquecimento do *dataset* parece ser o melhor caminho para a melhoria das métricas de qualidade, sendo o *chatbot* uma importante ferramenta para tal enriquecimento. Novos dados que possuam mais exemplos descrevendo os sintomas utilizando outras palavras, maior diversidade de combinação de sintomas na mesma frase e exemplos de negação da presença de um sintoma foram os principais enriquecimentos identificados para o aumento da capacidade de generalização da rede neural.

Existem vários caminhos abertos para a expansão do presente trabalho. Por exemplo, pode-se aumentar o número de sintomas analisados pela rede neural, ou mesmo analisar outras condições prévias de saúde da pessoa, como diabetes e câncer. Outra expansão natural do trabalho seria criar um modelo similar para dados de áudio, aumentando as alternativas possíveis para interação com os futuros usuários dessas tecnologias.

A empresa Lemobs foi responsável pela apresentação inicial do problema tratado nesse projeto e por seu financiamento. A empresa continua investindo no desenvolvimento da solução usando as estratégias mencionadas acima. A empresa demonstrou interesse em integrar uma versão do *chatbot* desenvolvido em seus produtos para uso em avaliações de pessoas com suspeita de COVID-19 e em outros protocolos e *scores* de risco em áreas como saúde mental, risco cardíaco, entre outras.

Os resultados obtidos pela rede BERTimbau e as análises feitas utilizando-se o LIME foram publicados na Revista de Notas Técnicas do CBPF sob o código DOI 10.7437/nt2236-7640/2022.01.001.

Referências

[1] Andrew Reyner, Wibowo Tjiptomongsoguno, Audrey Chen, Hubert Sanyoto, Edy Irwansyah, and Bayu Kanigoro. *Medical Chatbot Techniques: A Review*, pages 1–11. 11 2020.

[2] Lekha Athota, Vinod Kumar Shukla, Nitin Pandey, and Ajay Rana. Chatbot for Healthcare System Using Artificial Intelligence, 2020.

[3] Wei-jie Guan, Zheng-yi Ni, Yu Hu, Wen-hua Liang, Chun-quan Ou, Jian-xing He, Lei Liu, Hong Shan, Chun-liang Lei, David S.C. Hui, Bin Du, Lan-juan Li, Guang Zeng, Kwok-Yung Yuen, Ru-chong Chen, Chun-li Tang, Tao Wang, Ping-yan Chen, Jie Xiang, Shi-yue Li, Jin-lin Wang, Zi-jing Liang, Yi-xiang Peng, Li Wei, Yong Liu, Ya-hua Hu, Peng Peng, Jian-ming Wang, Ji-yang Liu, Zhong Chen, Gang Li, Zhi-jian Zheng, Shao-qin Qiu, Jie Luo, Chang-jiang Ye, Shao-yong Zhu, and Nan-shan Zhong. Clinical characteristics of coronavirus disease 2019 in china. *New England Journal of Medicine*, 382(18):1708–1720, 2020.

[4] Yan Bai, Lingsheng Yao, Tao Wei, Fei Tian, Dong-Yan Jin, Lijuan Chen, and Meiyun Wang. Presumed Asymptomatic Carrier Transmission of COVID-19. *JAMA*, 323(14):1406–1407, 04 2020.

[5] Qun Li, Xuhua Guan, Peng Wu, Xiaoye Wang, Lei Zhou, Yeqing Tong, Ruiqi Ren, Kathy S.M. Leung, Eric H.Y. Lau, Jessica Y. Wong, Xuesen Xing, Nijuan Xiang, Yang Wu, Chao Li, Qi Chen, Dan Li, Tian Liu, Jing Zhao, Man Liu, Wenxiao Tu, Chuding Chen, Lianmei Jin, Rui Yang, Qi Wang, Suhua Zhou, Rui Wang, Hui Liu, Yinbo Luo, Yuan Liu, Ge Shao, Huan Li, Zhongfa Tao, Yang Yang, Zhiqiang Deng, Boxi Liu, Zhitao Ma, Yanping Zhang, Guoqing Shi, Tommy T.Y. Lam, Joseph T. Wu, George F. Gao, Benjamin J. Cowling, Bo Yang, Gabriel M. Leung, and Zijian Feng. Early transmission dynamics in wuhan, china, of novel coronavirus–infected pneumonia. *New England Journal of Medicine*, 382(13):1199–1207, 2020. PMID: 31995857.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[8] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.

[9] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: BERT for document classification. *CoRR*, abs/1904.08398, 2019.

[10] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *CoRR*, abs/1908.08345, 2019.

[11] Muhammad Attique Khan, Nazar Hussain, Abdul Majid, Majed Alhaisoni, Bukhari Syed Ahmad Chan, Seifedine Kadry, Yunyoung Nam, and Zhang Yu-Dong. Classification of positive covid-19 ct scans using deep learning. *Computers, Materials, & Continua*, pages 2923–2938, 2021.

[12] Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. Deep learning applications for covid-19. *Journal of big Data*, 8(1):1–54, 2021.

[13] Talha Burak Alakus and Ibrahim Turkoglu. Comparison of deep learning approaches to predict covid-19 infection. *Chaos, Solitons & Fractals*, 140:110120, 2020.

[14] Abeed Sarker, Sahithi Lakamana, Whitney Hogg-Bremer, Angel Xie, Mohammed Ali Al-Garadi, and Yuan-Chi Yang. Self-reported COVID-19 symptoms on Twitter: an analysis and a research resource. *Journal of the American Medical Informatics Association*, 27(8):1310–1315, 07 2020.

[15] Deepak Kumar, Nalin Kumar, and Subhankar Mishra. NLP@NISER: Classification of COVID19 tweets containing
symptoms, June 2021.

[16] Alberto Valdes, Jesus Lopez, and Manuel Montes. UACH-INAOE at SMM4H: a BERT based approach for classification of COVID-19 Twitter posts. In *Proceedings of the Sixth Social Media Mining for Health
(#SMM4H) Workshop and Shared Task*, pages 65–68, Mexico City, Mexico, June 2021. Association for Computational Linguistics.

[17] Ying Luo, Lis Pereira, and Kobayashi Ichiro. OCHADAI at SMM4H-2021 task 5: Classifying self-reporting tweets on potential cases of COVID-19 by ensembling pre-trained language models. In *Proceedings of the Sixth Social Media Mining for Health*

(#SMM4H) Workshop and Shared Task, pages 123–125, Mexico City, Mexico, June 2021. Association for Computational Linguistics.

[18] Tim Mackey, Vidya Purushothaman, Jiawei Li, Neal Shah, Matthew Nali, Cortni Bardier, Bryan Liang, Mingxiang Cai, and Raphael Cuomo. Machine learning to detect self-reporting of symptoms, testing access, and recovery associated with covid-19 on twitter: Retrospective big data infoveillance study. *JMIR Public Health Surveill*, 6(2):e19509, Jun 2020.

[19] Tom; et. al Nadarzynski1. Acceptability of artificial intelligence (ai)-led chatbot services in healthcare: A mixed-methods study. *Digital Health*, 5:1–12, 2019.

[20] P. Norvig and S. Russell. *Inteligência artificial: Tradução da 3a Edição*. Elsevier Brasil, 2014.

[21] Andreas C Müller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. "O'Reilly Media, Inc.", 2016.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] John A Hertz. *Introduction to the theory of neural computation*. CRC Press, 2018.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[25] Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[27] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[30] Santiago González-Carvajal and Eduardo C. Garrido-Merchán. Comparing BERT against traditional machine learning text classification. *CoRR*, abs/2005.13012, 2020.

[31] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. A modular deep learning approach for extreme multi-label text classification. *CoRR*, abs/1905.02331, 2019.

[32] Zhengjie Gao, Ao Feng, Xinyu Song, and Xi Wu. Target-dependent sentiment classification with bert. *IEEE Access*, 7:154290–154299, 2019.

[33] Harish Tayyar Madabushi, Elena Kochkina, and Michael Castelle. Cost-sensitive BERT for generalisable sentence classification with imbalanced data. *CoRR*, abs/2003.11563, 2020.

[34] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[35] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. On position embeddings in {bert}. In *International Conference on Learning Representations*, 2021.

[36] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Bertimbau: Pretrained bert models for brazilian portuguese. In Ricardo Cerri and Ronaldo C. Prati, editors, *Intelligent Systems*, pages 403–417, Cham, 2020. Springer International Publishing.

[37] Jorge A. Wagner Filho, Rodrigo Wilkens, Marco Idiart, and Aline Villavicencio. The brWaC corpus: A new open resource for Brazilian Portuguese. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[38] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.