Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Avaliação Automática de Redações no Modelo do ENEM por meio do *fine-tuning* do BERTimbau

José Lucas Silva Mayer

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Denis Deratani Mauá
Cossupervisor: Igor Cataneo Silveira

São Paulo
2023

In memoriam de Luiz Augusto Paulino Jorge,
que se aventurou comigo nas primeiras jor-
nadas da minha vida pela computação.

Agradecimentos

Love is the one thing we're capable of perceiving that transcends dimensions of time and space
— Dr. Amelia Brand, *Interstellar*

Agradeço, de ínicio, à minha família, por ser minha base em toda a minha jornada acadêmica. Em especial, agradeço à minha mãe, Alexandra D. Silva Mayer, ao meu pai, Alaor Mayer, e ao meu irmão, Luan Silva Mayer, que sempre acreditaram no meu potencial e me deram todo o apoio e suporte necessário até aqui. Agradeço, também, às minhas avós Cícera Donizete da Silva e Elfrides Gasparini Mayer, por serem meu amparo emocional desde cedo.

Aos meus amigos do ensino fundamental e da minha cidade natal, em especial ao Jadeir Kelvin, ao Pedro Zauchin e ao Ulisses Borges, agradeço imensamente o apoio durante os anos da graduação. Foram incontáveis os dias em que vocês me ouviram desabafar sobre matérias difíceis ou questões emocionais da fase adulta. Sou muito grato pela presença de vocês em minha vida, e sei que o nosso amor não tem distância.

Aos amigos que tive o privilégio de conhecer na graduação: vocês tornaram o processo muito mais leve. Agradeço imensamente pelos sofrimentos compartilhados, pelos almoços e jantas na universidade, pelas festas, pelos jogos, pelos papos furados no saguão do CCSL, entre outros. Em especial, agradeço ao Max Cabrajac, um dos amigos mais legais que tive o privilégio de encontrar na vida, com quem sei que posso contar para qualquer coisa. Desbravar a Ciência da Computação e a Matemática com vocês foi muito mais legal.

Agradeço, também, aos meus professores, que sempre foram muito solícitos e dedicados em ensinar. Vocês foram muito importantes para o meu processo de maturação acadêmica e com certeza têm um papel crucial no avanço da ciência no país. Agradeço, em especial, ao meu orientador Denis Deratani Mauá, e ao meu coorientador Igor Cataneo Silveira, que estiveram disponíveis ao longo de todo o ano não só para tirar minhas dúvidas a respeito deste projeto, mas também para oferecer suporte e ferramentas para a execução

do trabalho.

Por fim, agradeço à universidade dos meus sonhos, por me permitir estudar o que amo de forma gratuita e com qualidade. Sem dúvidas, a experiência da graduação na USP será, no futuro, uma das memórias mais bonitas que poderei ter!

Resumo

José Lucas Silva Mayer. **Avaliação Automática de Redações no Modelo do ENEM por meio do *fine-tuning* do BERTimbau**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A avaliação automatizada de redações, que integra as ferramentas de inteligência artificial à educação, tem obtido um notável crescimento nos últimos anos, impulsionada pelo uso de técnicas avançadas de processamento de linguagem natural. Esta abordagem, caracterizada pela atribuição de notas a produções textuais de estudantes, ganha destaque no âmbito do Exame Nacional do Ensino Médio (ENEM), principal meio de admissão ao ensino superior no Brasil. Nesse contexto, a tecnologia demonstra grande potencial para agilizar a cadeia logística de avaliações, oferecendo meios eficazes de qualificar as competências de escrita dos alunos em escala nacional. Este trabalho explora o desenvolvimento de sistemas automáticos de correção de redações em língua portuguesa, alinhados ao formato do ENEM. A partir de coletâneas abertas de textos avaliados por especialistas, elaboraram-se estruturas que se utilizam do BERTimbau, um modelo de linguagem fundamentado em *transformers* e adaptado ao português, para codificar redações com riqueza semântica, empregando-as no treinamento de redes neurais capazes de atribuir notas correspondentes. Em particular, cinco sistemas independentes foram construídos com o intuito de avaliar os textos a partir de uma matriz de competências, como exigido pelo ENEM. Além disso, ao longo do projeto, aprimorou-se a eficácia dos modelos por meio da utilização de técnicas especializadas de aprendizado. Isso incluiu a investigação automática de configurações ideais de hiperparâmetros para as redes, buscando aprimorar a precisão dos resultados obtidos. Apesar da constatada ineficácia no processo de otimização de hiperparâmetros, a opção por arquiteturas fixas revelou-se promissora, aproximando os resultados da avaliação automática da qualidade das correções humanas.

Palavras-chave: ENEM. Inteligência Artificial. Processamento de Linguagem Natural. Avaliação Automática de Redações. Transformadores. BERTimbau.

Abstract

José Lucas Silva Mayer. **Automated Essay Scoring of ENEM Essays through BERTimbau fine-tuning**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

The automated essay scoring, integrating artificial intelligence tools into education, has experienced remarkable growth in recent years, driven by the use of advanced natural language processing techniques. This approach, characterized by grading students' textual productions, stands out within the scope of the *Exame Nacional do Ensino Médio* (ENEM), the primary means of admission to higher education in Brazil. In this context, technology demonstrates great potential to speed up the assessment logistics chain, offering effective means of qualifying students' writing skills on a national scale. This work explores the development of automatic essay correction systems in the Portuguese language, aligned with the ENEM format. The structures were built from open collections of texts evaluated by experts, utilizing BERTimbau, a large language model based on transformers and adapted to Portuguese, to generate essays embeddings with semantic richness. These structures are employed in the training of neural networks capable of assigning corresponding grades. Specifically, five independent systems were constructed to evaluate texts based on a competency matrix, as required by the ENEM. Throughout the project, the effectiveness of the models was enhanced through the use of specialized learning techniques. This included the automated hypertuning procedure for the networks, aiming to improve the accuracy of the results obtained. Despite the observed inefficiency in the hyperparameter optimization, the choice of fixed architectures proved promising, bringing the results of automatic assessment closer to the quality of human evaluations.

Keywords: ENEM. Artificial Intelligence. Natural Language Processing. Automated Essay Scoring. Transformers. BERTimbau.

Lista de abreviaturas

ENEM	Exame Nacional do Ensino Médio
INEP	Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira
SiSU	Sistema de Seleção Unificada
TOEFL	*Test of English as a Foreign Language*
NLP	Processamento de Linguagem Natural (*Natural Language Processing*)
AES	Avaliação Automática de Redações (*Automated Essay Scoring*)
PEG	*Project Essay Grader*
SVM	Máquina de Vetores de Suporte (*Support Vector Machine*)
RNN	Rede Neural Recorrente (*Recurrent Neural Network*)
BRNN	*Bidirectional Recurrent Neural Network*
BPTT	*Backpropagation Through Time*
LSTM	*Long Short-Term Memory*
BiLSTM	*Bidirectional Long Short-Term Memory*
GRU	*Gated Recurrent Unit*
BERT	*Bidirectional Encoder Representations from Transformers*
MLM	Modelagem de Linguagem Mascarada *Masked Language Model*
NSP	Previsão de Próxima Frase *Next Sentence Prediction*
SGD	Gradiente Descendente Estocástico (*Stochastic Gradient Descent*)
MSE	Erro Quadrático Médio (*Mean Squared Error*)
PCE	Proporção de Correspondência Exata
QWK	*Quadratic Weighted Kappa*
DCC	Departamento de Ciência da Computação
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de figuras

Lista de tabelas

Lista de programas

Sumário

Capítulo 1

Introdução

Tradicionalmente, técnicas de classificação de texto têm desempenhado um papel fundamental na avaliação e atribuição de notas a redações com base em vários parâmetros, como conteúdo, estrutura, coerência e proficiência linguística (Rudner *et al.*, 2006; Persing *et al.*, 2010; Somasundaran *et al.*, 2014). Algoritmos clássicos de aprendizado de máquina têm sido utilizados para identificar e avaliar características-chave dentro das redações, com o intuito de mimetizar o processo humano de correção manual dos textos (Larkey, 1998). Essas técnicas muitas vezes dependem de padrões avaliados sob medida e de conhecimento específico do domínio, o que pode ser trabalhoso e menos adaptável à natureza dinâmica da linguagem.

Além disso, os modelos tradicionais de avaliação textual automática enfrentam desafios na compreensão da semântica e do contexto das redações (Lim *et al.*, 2021). Por exemplo, podem ter dificuldade em reconhecer nuances externas, perífrases ou referências culturais, que são aspectos críticos de uma correção eficaz (Brasil, 2019a). Tal limitação abriu caminho para modelos mais sofisticados, capazes de capturar a essência da linguagem humana de maneira mais abrangente.

Em particular, o surgimento recente de modelos de linguagem, como as arquiteturas de *transformers* (Vaswani *et al.*, 2017) e o *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin *et al.*, 2018), auxiliou na mitigação desse problema. Os transformadores se destacaram em várias tarefas de Processamento de Linguagem Natural (NLP) ao considerar o contexto e as relações entre palavras, tornando-os excepcionalmente adequados para a avaliação de redações. O BERT, que, por sua vez, é inspirado na lógica dos *transformers*, revolucionou a área de NLP ao introduzir os *embeddings*[1] contextualizados de palavras.

Modelos como o BERT operam com base nos princípios de pré-treinamento e de ajuste fino. Durante a fase do pré-treinamento, aprendem as complexidades da linguagem ao prever palavras ausentes em frases e compreender o contexto em que são usadas. Posteriormente, o ajuste fino permite que os modelos se adaptem a tarefas específicas, como a correção automática, a partir do treinamento com dados já rotulados. A capacidade

[1] *Embeddings*, no âmbito do NLP, são representações vetoriais numéricas de palavras, cuja codificação tem o intuito de garantir que o computador entenda sua semântica (Zhang *et al.*, 2023).

de considerar a redação como uma unidade contextual, em oposição a palavras ou frases isoladas, capacita as arquiteturas baseadas em *transformers* a capturar as complexidades da linguagem, incluindo semântica, coerência e até mesmo significados subjacentes ao texto (Yang *et al.*, 2020).

Tais qualidades desempenham um papel fundamental na correção automática de testes de conhecimento e produção textual, como no caso do Exame Nacional do Ensino Médio (ENEM), um dos principais instrumentos de avaliação educacional do Brasil (Oliveira, 2016). Na prova de redação desse exame, são estabelecidos critérios objetivos de pontuação que abordam competências como coesão textual, argumentação e domínio temático (Brasil, 2023), características que podem ser assimiladas na etapa de pré-treinamento, por exemplo.

Utilizando-se do BERT e dos modelos de linguagem baseados na arquitetura de *transformers*, os sistemas de avaliação automática de redações têm a possibilidade de apresentar um avanço significativo na capacidade de imitar os avaliadores humanos. Esses modelos podem medir a qualidade completa do texto, reconhecer nuances da língua portuguesa e fornecer uma nota coerente de acordo com critérios objetivos de correção. Ao longo desta monografia, exploraremos os princípios do ajuste fino do BERT na correção automática de redações com enfoque no modelo do ENEM, os desafios abarcados por essa tarefa e, por fim, o potencial dessa abordagem em relação aos métodos tradicionais e humanos de avaliação.

1.1 Contextualização

A avaliação de redações em contextos educacionais é uma tarefa de grande importância, pois permite mensurar a capacidade dos estudantes de comunicar suas ideias de maneira clara, coerente e persuasiva. No entanto, no cenário de acesso a esses textos em língua portuguesa, especialmente no âmbito do ENEM no Brasil, a disponibilidade de conjuntos de dados rotulados tem sido historicamente limitada, como pontuado por Marinho *et al.* (2021). Apesar do desenvolvimento de diversos sistemas de avaliação automática de redações em língua inglesa, adaptados para exames como o *Test of English as a Foreign Language* (TOEFL) (Rupp *et al.*, 2019), poucos surgiram para atender a textos em português ou ao modelo específico exigido pelo ENEM (Amorim e Veloso, 2017).

Dentre os poucos estudos na área, desafios adicionais têm sido a utilização histórica de mecanismos mais primordiais de aprendizado de máquina e a falta de transparência sobre os dados. Trabalhos anteriores nesse campo, como o de Amorim e Veloso (2017), utilizaram técnicas de extração de características dos textos e métodos mais simples para pontuar as redações. Além disso, as bases de textos utilizadas muitas vezes não são divulgadas publicamente, como no estudo de Fonseca *et al.* (2018), dificultando a verificação e replicação dos resultados e limitando a aplicação de abordagens mais sofisticadas.

Com o intuito de enfrentar essas barreiras, em Marinho *et al.*, 2021, os autores desenvolveram uma base de redações composta por 4570 textos dissertativo-argumentativos escritos por estudantes do ensino médio no Brasil e avaliados por especialistas na área de correção desses textos, nomeada Essay-BR. Tal conjunto foi projetado especificamente

para atender às necessidades da avaliação automática de redações no modelo do ENEM, dado que as notas foram atribuídas de acordo com a matriz de referência do exame.

Para criar o conjunto das redações, os autores utilizaram técnicas de raspagem de dados por meio de ferramentas de extração e de filtragem, que permitiram a coleta dos textos e avaliações, organizados entre dezembro de 2015 e abril de 2020, com base em dois sites públicos: Vestibular UOL e Educação UOL. Ambos os portais educacionais são fontes confiáveis, uma vez que os ensaios foram escritos por estudantes do ensino médio e avaliados por especialistas, seguindo os critérios do ENEM.

A diversidade de temas presentes na Essay-BR também é uma característica notável. Com um total de 86 tópicos-base, o conjunto de dados reflete a ampla gama de proposições abordadas no ENEM (Stein, 2023), abrangendo questões relacionadas a direitos humanos, desafios políticos, movimentos populares, dentre outros. Isso torna a base de textos uma representação fiel das complexidades e diversidades temáticas enfrentadas pelos estudantes no exame, o que é fundamental para avaliações mais precisas.

O conjunto de dados possui, também, uma versão estendida contendo 6479 textos e 151 temas. Essa versão foi coletada entre o período de dezembro de 2015 a agosto de 2021, com a mesma técnica utilizada para a coleta do Essay-BR (Marinho *et al.*, 2022). As redações da base estendida também têm notas atribuídas de acordo com a matriz de referência do ENEM, variando somente na formatação da estrutura de colunas.

A disponibilidade pública desses dados promove, por fim, a transparência e a replicabilidade da pesquisa nesse campo, possibilitando que pesquisadores apliquem abordagens mais sofisticadas para avanços na correção automática de redações. Essas bases desempenham um papel crucial no desenvolvimento de sistemas de avaliação automática em língua portuguesa e são as principais referências utilizadas por este trabalho.

1.2 Motivação

O Exame Nacional do Ensino Médio (ENEM) é uma avaliação educacional de grande importância no Brasil, realizada anualmente pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP). Desde sua criação em 1998, o exame tem evoluído para se tornar uma referência no processo de acesso ao ensino superior no país. A pontuação obtida pelos estudantes no ENEM é utilizada para diversos fins, sendo um dos mais notáveis o ingresso em universidades públicas por meio do Sistema de Seleção Unificada (SiSU) (Oliveira, 2016).

A redação é uma das principais provas do exame, cujo intuito é avaliar a defesa de um ponto de vista com respeito a um tema definido *a priori*. Ela é capaz de mensurar a capacidade dos estudantes de comunicar ideias de forma clara, coerente e persuasiva, respeitando o gênero dissertativo-argumentativo e a norma padrão da língua portuguesa, conforme detalhado em Brasil (2023). Além disso, habilidades críticas, como a análise de informações e a construção de argumentos sólidos, também são levadas em conta no processo de correção da prova (Brasil, 2019a; Brasil, 2019b).

De acordo com a Cartilha do Participante de 2023 (Brasil, 2023), a avaliação das redações é baseada em cinco competências, cujas notas podem variar de 0 a 200 em

intervalos de 40 pontos. A pontuação final é calculada somando-se todos esses valores, de modo que o resultado final esteja entre 0 e 1000. O processo de correção de cada redação é feito por dois avaliadores independentes e, caso as notas divirjam em mais de 100 pontos no total ou em mais de 80 pontos em cada competência, o texto é corrigido por um terceiro profissional. Por fim, se a pontuação ainda for divergente com ambas as avaliações, a redação é avaliada por uma banca de três professores.

Entretanto, a correção manual de uma grande quantidade de redações representa um desafio logístico significativo, levando algumas organizações que atuam nessa área a, inclusive, adotar soluções mais ágeis, como apontado por Taghipour e Ng, 2016. Ademais, a disparidade entre a quantidade de profissionais e o número de estudantes que participam do ENEM a cada ano leva a uma sobrecarga que, além de aumentar o tempo de espera dos resultados, exige um esforço excessivo por parte dos avaliadores (Lesme, 2021).

Nessa perspectiva, a automação da correção de redações utilizando modelos de linguagem pode configurar uma solução promissora para o problema. O BERT e outras arquiteturas baseadas em *transformers* têm o potencial de oferecer uma avaliação objetiva das redações dos estudantes, dado que eles podem ser refinados com uma gama de exemplos de redações já avaliadas por profissionais especializados e, assim, aprender a reconhecer critérios de qualidade para cada competência com base em dados reais.

Além disso, a avaliação automática tem potencial para promover uma resposta mais rápida e eficiente, uma vez que as notas podem ser geradas em um período mais curto. Isso facilitaria o processo logístico de divulgação dos resultados, reduzindo significativamente o tempo necessário para que os estudantes tenham acesso às suas pontuações.

1.3 Objetivos

Os principais objetivos deste trabalho são:

- Desenvolver um modelo de correção automática de redações baseado no BERT, utilizando a arquitetura de *transformers* para avaliar e pontuar redações em língua portuguesa.
- Comparar o desempenho do modelo desenvolvido com métodos tradicionais de extração de características de texto e pontuação de redações.
- Avaliar a acurácia e as métricas de avaliação do modelo desenvolvido em relação às notas atribuídas pelos avaliadores humanos nas redações, considerando as cinco competências estabelecidas no ENEM.
- Analisar os resultados obtidos e discutir as vantagens e desvantagens da abordagem proposta em relação aos métodos tradicionais e à correção manual realizada por avaliadores humanos.
- Documentar o processo de escolha da arquitetura do modelo, o treinamento com *tuning* de hiperparâmetros e os resultados alcançados.

Ao longo da monografia, cada tópico será melhor desenvolvido e detalhado, de modo a fornecer uma visão mais concreta sobre o escopo do trabalho e os resultados obtidos.

Capítulo 2

Fundamentação Teórica

2.1 A Redação no ENEM

A prova de redação no ENEM desempenha um papel fundamental na qualificação das habilidades dos estudantes, medindo a capacidade de expressarem ideias de maneira clara, coerente e persuasiva. Para avaliar as redações, são estabelecidos critérios objetivos que consideram a estrutura do texto e métricas específicas do exame, baseadas em uma matriz de referência.

2.1.1 Estrutura do Texto

A estrutura da redação no ENEM segue o formato dissertativo-argumentativo, exigindo que os participantes defendam um ponto de vista sobre um tema previamente estabelecido, conforme abordado em Brasil, 2023. A composição textual deve respeitar a norma padrão da língua portuguesa e o gênero determinado, mantendo uma introdução, desenvolvimento e conclusão. A clareza na exposição de ideias, a adequação ao tema e a coesão textual são elementos essenciais para uma boa pontuação.

Para os autores Fiorin e Savioli (2006), a dissertação deve explicitar uma visão concreta da realidade, recorrendo a referências para ilustrar afirmações ou para sustentar argumentos. No contexto do ENEM, esse é o papel exercido pela coesão e articulação da redação, já que o texto deve manter uma progressão lógica e um encadeamento de ideias com o fim de defender o ponto de vista do aluno. Além disso, os autores defendem que a dissertação sempre deve seguir um eixo temático à medida que aborda conceitos do mundo de modo abstrato, sem relações temporais ou espaciais. Nesse sentido, na prova, é imperativo que o texto apresente uma coerência temática, alinhando-se ao tópico proposto.

Ainda ao longo do texto, os estudantes devem articular informações externas para embasar seus discursos, o que é definido, em Fiorin e Savioli, 2006, como argumentos de autoridade e argumentos baseados em provas concretas. Assim, aliado à seleção de elementos coesivos adequados, o propósito da redação deve ser o de convencer o interlocutor em torno do tema abordado.

Por fim, é exigido que os alunos apresentem uma proposta de intervenção na conclusão

do texto, que deve conter as ações necessárias para mitigar o problema levantado, os agentes responsáveis, os meios de solução, os possíveis desdobramentos da aplicação da proposta e, por fim, um detalhamento mais refinado da mediação (Brasil, 2023, p. 20).

2.1.2 Métricas de Avaliação

A avaliação das redações no ENEM adota uma abordagem multifacetada, incorporando diversas métricas para assegurar uma análise abrangente, incluindo desde a coesão textual até a capacidade de argumentação e a adequação ao padrão culto da língua. A seleção e organização eficazes de informações também são critérios fundamentais. A combinação desses parâmetros contribui significativamente para a construção de uma avaliação equitativa e holística das habilidades de escrita dos participantes.

Cada redação no ENEM é avaliada com base na matriz de referência do exame, composta por cinco competências, delineadas em suas respectivas habilidades, conforme detalhado em Brasil, 2023. A primeira competência concentra-se na avaliação do emprego correto da língua portuguesa, abrangendo aspectos como ortografia, pontuação, concordância, regência verbal, etc. A segunda competência analisa a capacidade de interpretação do participante em relação ao tema proposto. Na terceira competência, são avaliados critérios como seleção e organização de informações, articulação de ideias e construção de argumentos. A quarta competência incide sobre o encadeamento dos argumentos e a progressão temática do texto. Por fim, a quinta competência avalia a habilidade do participante em elaborar uma proposta de intervenção para o problema abordado, considerando a adequação ao tema, a viabilidade prática de aplicação e o respeito aos direitos humanos. As cinco competências podem ser sintetizadas, de acordo com a Cartilha de Redação do ENEM (Brasil, 2023), em:

I. Demonstrar domínio da modalidade escrita formal da língua portuguesa.

II. Compreender a proposta de redação e aplicar conceitos das várias áreas de conhecimento para desenvolver o tema, dentro dos limites estruturais do texto dissertativo-argumentativo em prosa.

III. Selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista.

IV. Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação.

V. Elaborar proposta de intervenção para o problema abordado, respeitando os direitos humanos.

2.1.3 Atribuição de Notas

A atribuição de notas na redação do ENEM segue um processo rigoroso. Cada uma das cinco competências é avaliada em uma escala de 0 a 200 pontos, variando, nesse intervalo, em valores múltiplos de 40. As redações são corrigidas, inicialmente, por dois especialistas independentes e todo o processo é meticulosamente estruturado para prevenir possíveis divergências nas pontuações finais.

Duas avaliações são consideradas divergentes caso a nota atribuida a qualquer uma das competências difira em mais de 80 pontos ou se a diferença total entre as notas seja superior a 100 pontos. Nesses casos, uma terceira correção é realizada, e a nota final é a média aritmética das duas avaliações mais próximas. Caso todas as avaliações ainda sejam discrepantes entre si, a redação é submetida a uma banca independente, responsável por atribuir a nota final ao texto.

As notas levam em consideração as métricas da matriz de referência do exame, conforme estabelecido em Brasil, 2023, p. 9-22. A pontuação final varia de 0 a 1000 pontos, sendo calculada a partir da soma dos pontos atribuídos a cada competência. Este sistema de avaliação visa garantir uma análise abrangente e justa das redações dos participantes.

A tabela 2.1 abaixo, extraída da Cartilha de Redação do ENEM, ilustra a associação entre as pontuações atribuídas à competência I e os níveis de desempenho esperados dos alunos. As demais competências possuem uma associação similar de acordo com seus respectivos campos de avaliação.

Pontuação	**Níveis de desempenho**
200 pontos	Demonstra excelente domínio da modalidade escrita formal da língua portuguesa e de escolha de registro. Desvios gramaticais ou de convenções da escrita serão aceitos somente como excepcionalidade e quando não caracterizarem reincidência.
160 pontos	Demonstra bom domínio da modalidade escrita formal da língua portuguesa e de escolha de registro, com poucos desvios gramaticais e de convenções da escrita.
120 pontos	Demonstra domínio mediano da modalidade escrita formal da língua portuguesa e de escolha de registro, com alguns desvios gramaticais e de convenções da escrita.
80 pontos	Demonstra domínio insuficiente da modalidade escrita formal da língua portuguesa, com muitos desvios gramaticais, de escolha de registro e de convenções da escrita.
40 pontos	Demonstra domínio precário da modalidade escrita formal da língua portuguesa, de forma sistemática, com diversificados e frequentes desvios gramaticais, de escolha de registro e de convenções da escrita.
0 pontos	Demonstra desconhecimento da modalidade escrita formal da língua portuguesa.

Tabela 2.1: *Níveis de desempenho esperados para a competência I e notas associadas. Tabela extraída de Brasil, 2023, p. 10*

2.2 Avaliação Automática de Redações

A avaliação automática de redações (AES) tem se destacado significativamente nos últimos anos devido à disparidade entre a disponibilidade de profissionais capacitados para correção de textos e o número expressivo de estudantes. O avanço das técnicas

de modelagem de linguagem e processamento de linguagem natural (NLP) impulsionou inúmeros trabalhos na tentativa de simplificar o processo de correção.

Este campo de pesquisa é intrinsecamente multidisciplinar, demandando conhecimentos em linguística, PLN, aprendizado de máquina e estatística. A meta central consiste em desenvolver modelos capazes de atribuir notas automaticamente a redações, seguindo critérios similares aos adotados por especialistas.

No cenário brasileiro, embora as pesquisas tenham crescido, a AES ainda é subexplorada devido às dificuldades em acessar conjuntos de dados e recursos computacionais. Muitos dos trabalhos nacionais foram desenvolvidos com foco no modelo de redações do ENEM, valendo-se de *corpus* abertos de textos de portais de correção, como o UOL Redação.

2.2.1 Definição e Importância

A avaliação automática de redações é o processo no qual algoritmos computacionais são empregados para analisar e pontuar ensaios escritos em linguagem natural. Seu ciclo operacional engloba principalmente duas fases: a codificação dos textos, transformando-os em representações compreensíveis para os computadores, e a extração de informações, cujo propósito é identificar elementos relevantes no conteúdo textual para a atribuição da pontuação.

O cerne da AES reside na capacidade de desenvolver modelos capazes de atribuir notas automaticamente, em conformidade com critérios previamente estabelecidos. Tais critérios podem ser fundamentados em modelos estatísticos ou em técnicas de aprendizado de máquina supervisionado, em que os algoritmos são treinados com conjuntos de textos já avaliados por especialistas. É possível, dessa forma, desenvolver sistemas que aprendam os padrões e nuances que guiam a correção das redações.

A importância da área encontra-se na necessidade de enfrentar desafios logísticos e temporais associados à correção manual de redações, especialmente em contextos educacionais massivos, como ocorre no ENEM. A utilização de sistemas automáticos de avaliação não apenas agiliza o processo de correção, reduzindo o tempo necessário para a atribuição de notas, mas também alivia o esforço demandado por profissionais, como levantado em Costa *et al.*, 2020.

A objetividade, característica-chave de muitos critérios de avaliação de redações, também é um recurso importante dos sistemas automáticos. A AES contribui para uma correção mais uniforme e padronizada, minimizando possíveis variações subjetivas entre diferentes avaliadores e facilitando o processo de generalização exigido dos especialistas (Myers, 2003). Além disso, a capacidade de processar grandes volumes de redações de forma eficiente a torna uma ferramenta valiosa em contextos educacionais de larga escala, onde a demanda por correção é significativa.

Em suma, a avaliação automática de redações é um campo de pesquisa e aplicação promissor, integrando conhecimentos de linguística, processamento de linguagem natural, aprendizado de máquina e estatística para endereçar desafios educacionais contemporâneos.

2.2.2 Breve Histórico

O surgimento da AES remonta a pesquisas pioneiras da década de 1960, destacando-se o *Project Essay Grader* (PEG), desenvolvido para aprimorar a avaliação em larga escala de redações (Page, 1966). O PEG utiliza medidas como comprimento médio de palavras e extensão dos textos para prever a qualidade das produções. Embora tenha sido elogiado, à época, por sua comparabilidade com avaliações humanas e eficiência computacional, a primeira versão do PEG recebeu críticas por negligenciar aspectos semânticos e por sua vulnerabilidade a práticas fraudulentas. Posteriormente, na década de 1990, esses problemas foram mitigados com a incorporação de dicionários e esquemas especiais de classificação ao sistema (Barrera e Shermis, 2002).

Um marco significativo no desenvolvimento da avaliação automática de redações no contexto nacional foi a pesquisa conduzida por Amorim e Bazelato (2013), com o desenvolvimento de um classificador bayesiano com cerca de 400 redações extraídas da base de dados do UOL Redações.

Subsequentemente, Amorim e Veloso (2017) propuseram um modelo baseado em regressão, treinado com 1840 textos. O trabalho foi realizado incorporando-se, para cada produção, duas classes de características: as que dizem respeito ao ENEM, chamadas de específicas de domínio, e as gerais, inspiradas no trabalho de Attali e Burstein (2006).

Paralelamente, Júnior *et al.* (2017) utilizaram técnicas de Máquinas de Vetores de Suporte (SVM) com cerca de 4000 redações extraídas do portal UOL, empregando o corretor gramatical CoGrOO (Silva, 2013) para avaliar aspectos ortográficos do texto. Esse ensaio focou apenas na primeira competência do ENEM, que qualifica a adequação à modalidade formal da língua portuguesa.

Em outro estudo relevante, conduzido por Fonseca *et al.* (2018), foram adotadas duas abordagens distintas. Em uma delas, os autores implementaram uma arquitetura de rede neural profunda com camadas bidirecionais de Long Short-Term Memory (BiLSTM). Na outra, criaram 681 características para alimentar um regressor na avaliação de redações, atingindo um resultado promissor e de melhor desempenho. Este trabalho utilizou uma base de cerca de 56000 redações para treinar o sistema.

No contexto do Brasil, apesar do avanço recente da área de AES, muitos dos estudos anteriores não disponibilizaram publicamente os conjuntos de textos utilizados, o que gera desafios para comparações robustas entre os trabalhos. Além disso, torna-se um desafio utilizar novas técnicas de avaliacão automática devido à escassez de dados representativos em língua portuguesa.

2.3 Processamento de Linguagem Natural no Computador

Ao lidarmos com textos no âmbito computacional, é crucial convertê-los em formatos adequados às máquinas. Algumas abordagens empregadas em NLP são os *tokens*, os *n*-gramas, os *one-hot encodings* e os *word embeddings*.

2.3.1 *Tokens* e a Representação de Palavras

Tokens são unidades fundamentais de um texto, que podem englobar tanto palavras completas quanto partes delas. Formalmente, considerando um texto T composto por n elementos $t_1, t_2, \cdots, t_n$ que possam ser separados de forma discreta, os *tokens* podem ser definidos como o conjunto $\{t_1, t_2, ..., t_n\}$ (Manning e Schütze, 1999). De modo mais simples, *tokens* são os elementos individuais de um texto, que podem ser divididos de forma a isolar unidades relevantes para um processamento.

O conceito de *tokens* é crucial na representação de palavras, possibilitando que o computador processe e compreenda o conteúdo textual de maneira estruturada. Sua identificação e segmentação adequadas são vitais para tarefas de processamento de linguagem natural, como contagem de frequência, análise sintática e desenvolvimento de modelos.

Ao lidar com *tokens*, é possível aplicar técnicas como a tokenização, que consiste na subdivisão de um texto em unidades individuais. Essa abordagem facilita a manipulação e análise, permitindo que algoritmos processem informações linguísticas de maneira mais eficaz.

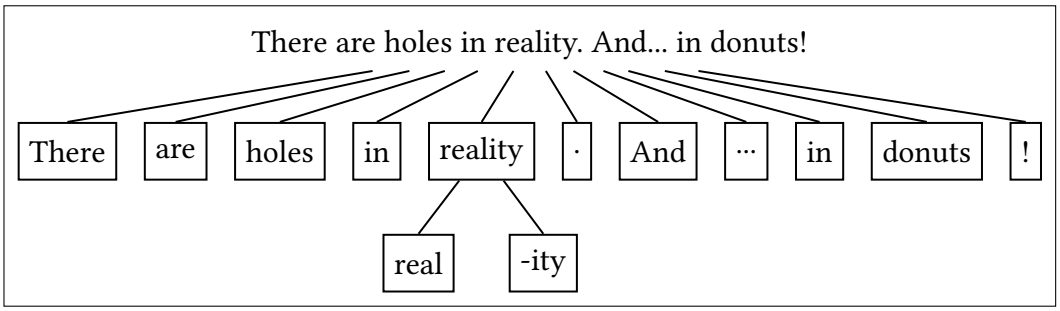

Figura 2.1: *Exemplo de tokenização de uma frase em palavras, subpalavras e pontuações.*

2.3.2 *N*-gramas e Modelos de Predição de Palavras

Os *N*-gramas, em NLP, são estruturas de agrupamento de palavras ou *tokens* que desempenham um papel crucial na tarefa de previsão em textos. Tratam-se de uma junção de todos os elementos subjacentes possíveis, formando conjuntos de n unidades. Essas representações são utilizadas, aliadas aos conceitos da teoria de Markov e das probabilidades, para treinar modelos simples de predição de palavras.

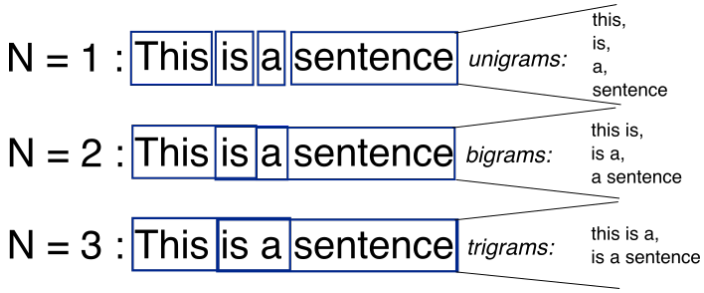

Figura 2.2: *Exemplos de n-gramas para $n = 1, 2, 3$. Imagem extraída de Bhattacharjee, 2018.*

O objetivo, nesse caso, é estimar a função de probabilidade $P(W_t|W_{t-(n-1)}, \cdots, W_{t-1})$, onde W_t é a palavra atual e $W_{t-(n-1)}, \cdots, W_{t-1}$ representa o histórico de palavras. Sob a perspectiva estocástica, a classificação da história anterior $(W_{t-(n-1)}, \cdots, W_{t-1})$ é essencial para predizer as novas ocorrências, já que, com uma quantia suficientemente grande de textos, é possível estimar quais palavras tendem a aparecer em sequência.

No entanto, lidar com cada história textual separadamente é impraticável, já que é possível receber como entrada textos totalmente novos, sem construções de referência. A teoria de Markov, assim, surge como uma solução plausível, considerando que apenas o contexto local anterior, representado pelo conjunto das últimas palavras $(W_{t-(n-1)}, ..., W_{t-1})$, influencia a escolha da próxima ocorrência. Essa suposição leva em conta que a ordem de palavras em um texto é relevante e que geralmente existe maior correlação estatística entre unidades vizinhas, conforme pontuado em Bengio *et al.*, 2003.

Agrupando todas as histórias que compartilham as mesmas $n-1$ palavras em uma mesma classe de equivalência, cria-se um modelo de Markov de ordem $n-1$, conhecido como modelo de linguagem N-grama (Manning e Schütze, 1999). Ao nomear tais modelos, a terminologia comumente utilizada refere-se a valores específicos de n, como bigrama para $n = 2$ e trigrama para $n = 3$. A abordagem de agrupar contextos semelhantes com base na teoria de Markov oferece uma maneira eficaz de prever palavras subsequentes em textos, com ampla aplicação no processamento de linguagem natural e análise de sentimentos.

2.3.3 *One-Hot Encodings*

A técnica de *one-hot encoding* é uma abordagem de representação palavras ou *tokens* no âmbito do NLP. Nesse mecanismo, cada elemento do vocabulário é mapeado para um vetor binário distinto, onde todos os valores são zero, exceto aquele correspondente à posição da palavra de interesse, para a qual atribui-se o valor 1. Esse método cria representações vetoriais esparsas, cuja dimensionalidade é equivalente ao tamanho do vocabulário.

A principal característica do *one-hot encoding* é a independência entre as representações de palavras. Cada uma delas é tratada como uma entidade única, sem consideração pela semelhança semântica ou relações contextuais com outras unidades do vocabulário. A representação binária resultante destaca a presença ou ausência de palavras específicas, mas não incorpora informações sobre o significado relativo delas ou suas interações semânticas.

Quando usado para codificar características ou classes, o *one-hot encoding* pode ser considerado mais interpretável, dada sua natureza de independência explícita. A representação binária atribui um valor de 1 à categoria de interesse, com sua devida especificação, e 0 para todas as outras categorias, individualizando a presença ou ausência de uma característica e aumentando a interpretação de modelos, como mostrado por Manai *et al.* (2023).

Apesar de sua simplicidade e interpretabilidade, o *one-hot encoding* apresenta desvantagens significativas em termos de eficiência computacional e capacidade de generalização. A representação esparsa resulta em um alto consumo de memória, especialmente em vocabulários extensos, e a falta de captura de relações semânticas limita sua utilidade em tarefas mais complexas de NLP.

2.3.4 *Word Embeddings*

Word embeddings referem-se a representações vetoriais de palavras que capturam relações semânticas e contextuais. Distintas dos N-gramas, que se restringem a contextos locais, essas representações incorporam informações abrangentes de todo o texto.

A origem dessas codificações remonta aos estágios iniciais dos modelos neurais probabilísticos de linguagem. Inspirados nas redes neurais de compressão de texto propostas por Schmidhuber e Heil (1996), Bengio *et al.* (2003) conceberam um modelo de predição de palavras com uma estrutura análoga. Nesse trabalho, uma camada de projeção foi introduzida na rede neural, desempenhando o papel de mapear palavras para vetores de baixa dimensão.

Essa abordagem serviu como fundamento para o desenvolvimento de técnicas mais refinadas de *word embedding*, que se disseminaram amplamente em aplicações de NLP. A representação matemática de palavras, expressa como vetores, proporciona não apenas manipulações computacionais eficientes, mas também a modelagem de aspectos semânticos pertinentes aos textos.

Lidar com representações vetoriais reais de palavras viabiliza a compreensão e exploração de relações semânticas complexas. Tais codificações permitem expressar analogias e operações sobre os elementos, como no exemplo da soma das palavras "rei" e "mulher", que resulta em vetores próximos da representação de "rainha" (Mikolov *et al.*, 2013).

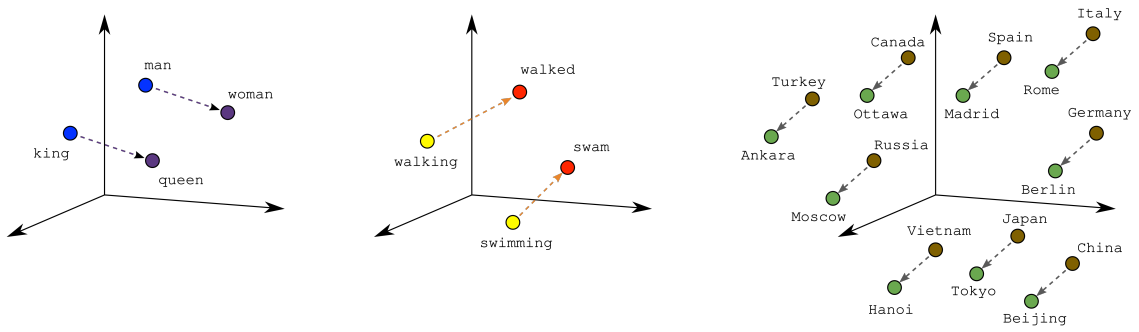

Figura 2.3: *Representação vetorial de palavras sob alguns eixos semânticos. Imagem extraída de https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space*

O emprego de *word embeddings* é, atualmente, mais frequente, já que o método confere vantagens significativas em termos de manipulação e compreensão semântica. Representações semelhantes a essa estrutura são empregadas em modelos de linguagem mais avançados, como o BERT. Três métodos notáveis de *word embedding* são o Word2Vec, GloVe e FastText.

2.3.4.1 Word2Vec

O Word2Vec é uma técnica de *word embedding* que realiza uma tradução de palavras para vetores em espaços contínuos. Este método, fundamentado em redes neurais, destaca-se por sua habilidade em preservar relações semânticas através de representações distribuídas (Mikolov *et al.*, 2013). O cerne do processo consiste na aprendizagem de representações que não apenas mapeiam palavras individuais, mas também capturam a proximidade semântica entre elas.

Essa codificação opera de modo a otimizar a representação vetorial de palavras com base em seu contexto geral, possibilitando a criação de *embeddings* que capturam a semelhança lexical e aspectos relacionados ao significado. O modelo se destaca principalmente na incorporação de analogias e de consequências lógicas, tais quais "**falar** está para **falado** como **escrever** está para **escrito**".

2.3.4.2 GloVe

O GloVe (*Global Vectors for Word Representation*) apresenta uma perspectiva distinta no panorama dos *word embeddings*. Ele se destaca na geração de representações vetoriais fundamentadas em estatísticas globais de coocorrência de palavras em um *corpus*. Essa abordagem permite uma visão abrangente das relações semânticas que transcendem contextos imediatos.

A metodologia do GloVe se baseia na construção de uma matriz de coocorrência, refletindo a probabilidade de duas palavras aparecerem juntas em um dado contexto (Pennington *et al.*, 2014). Por meio de técnicas algébricas, o modelo é usado para derivar *embeddings* que melhor representem as relações semânticas capturadas por essa matriz. Dessa forma, transcende-se a limitação de representar palavras isoladamente, proporcionando uma visão holística das conexões semânticas de um texto.

2.3.4.3 FastText

O FastText, uma extensão evolutiva do Word2Vec, é um tipo de codificação que atua em nível de subpalavras. Cada palavra, nesse método, é representada por *n*-gramas de suas composições. Essa inovação confere à técnica uma notável flexibilidade na manipulação de palavras desconhecidas, capturando informações semânticas em camadas mais inferiores de um texto, e permite uma maior agilidade no processo de treinamento.

Considerar subpalavras no processo de geração de *embeddings* capacita o FastText a representar os vocábulos como combinações de suas partes constituintes. Para a palavra *where*, por exemplo, esse modelo pode considerar os trigramas `<wh, whe, her, ere, re>`, conforme delineado por Bojanowski *et al.* (2016). Isso se revela particularmente valioso em línguas com estrutura morfológica complexa, como a língua portuguesa, onde as palavras são formadas por múltiplos elementos significativos. Alguns vocábulos não conhecidos, nesse caso, podem ser construídos pelos *n*-gramas, dando poder de generalização ao modelo.

Além disso, o FastText possui uma eficácia singular no processo de treinamento, permitindo a criação de *embeddings* em ambientes computacionais mais simples. Conforme

evidenciado por Joulin *et al.* (2016), métodos baseados em convoluções, tais quais os utilizados para treinar representações de Word2Vec, demonstram ser muito mais lentos que o FastText, que alcança tempos de treinamento da ordem de minutos em CPUs padrões de 20 *threads*. Essa notável agilidade contribui significativamente para a adoção do FastText em tarefas de NLP que exijam rápidas codificações.

2.4 Modelos de Linguagem Neurais

Os modelos de linguagem neurais são uma classe de entidades estatísticas que atribuem probabilidades a cadeias de palavras. Em geral, dado uma sequência de t *tokens* $\boldsymbol{W} = w_1, \cdots, w_t$ e um vocabulário de tamanho V, atuam de modo a aproximar a função de distribuição de probabilidades de um novo *token* w_{t+1}, indicado por $P(w_{t+1}|W)$ (S. Sun e Iyyer, 2021). Esses modelos utilizam uma função de composição, denotada por h, que é aplicada sobre a sequência W, produzindo uma saída $\boldsymbol{s} = h(\boldsymbol{W})$.

Como o intuito é obter uma aproximação da distribuição de probabilidades sobre o vocabulário, uma restrição fundamental é a de que a resposta $\boldsymbol{r}$ atribuída pelos modelos de linguagem, com $\boldsymbol{r} = (r_1, \cdots, r_V) \in \mathbb{R}^V$, tenha valores cuja soma total seja igual a 1. Para tal, utiliza-se, comumente, a função *softmax* (Bridle, 1989), cujo objetivo é normalizar um vetor $\boldsymbol{z} = (z_1, \cdots, z_n) \in \mathbb{R}^n$:

$$g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \text{ , para todo } i \in \{1, \cdots, n\} \tag{2.1}$$

A saída $\boldsymbol{s}$ é, então, utilizada para estimar a probabilidade de ocorrência de um novo *token* w_{t+1}, conforme a equação 2.1, a partir do mapeamento realizado por g. Esse processo gera a resposta $r = \widehat{P}(w_{t+1}|W)$, um vetor de distribuição de probabilidades aproximada para o *token* w_{t+1} sobre o vocabulário.

2.4.1 Redes Neurais Tradicionais

As redes neurais tradicionais foram as pioneiras no desenvolvimento de modelos de linguagem. Os autores Bengio *et al.* (2003) foram os primeiros a lidar com essa estrutura, propondo uma arquitetura mais simples com duas camadas principais. A primeira delas realiza uma tarefa de projeção, mapeando palavras para vetores de baixa dimensão. Já a segunda atua como uma camada oculta, que recebe uma sequência de vetores e produz uma saída com base em uma função de ativação.

A rede recebe como entrada representações em *one-hot encoding* de palavras, que são multiplicadas por uma matriz de projeção $C \in \mathbb{R}^{d \times V}$, onde d é a dimensão dos vetores de projeção e V é o tamanho do vocabulário. Essa matriz é treinada para realizar o mapeamento linear de palavras para vetores de representação distribuída, como os *word embeddings*. A saída da camada de projeção é uma matriz $X \in \mathbb{R}^{d \times t}$, onde t é o tamanho da sequência de entrada.

A matriz X é, então, utilizada como entrada para a camada oculta, composta por uma quantia fixa de unidades de processamento, convencionalmente chamadas de neurônios.

Cada neurônio recebe a sequência de vetores $x_i \in \mathbb{R}^d$ e produz uma saída $h_i \in \mathbb{R}^d$, calculada por:

$$h_i(x_i) = \tanh(b + Wx_i) \tag{2.2}$$

onde $W \in \mathbb{R}^{d \times d}$ é uma matriz de pesos e $b \in \mathbb{R}^d$ é um vetor de viés. A saída dessa etapa, representada pela matriz $H \in \mathbb{R}^{d \times t}$, passa por uma camada de tratamento com a função *softmax* (2.1), gerando, finalmente, um vetor que representa a aproximação da função de probabilidades.

A Figura 2.4 ilustra a estrutura dessa rede neural:

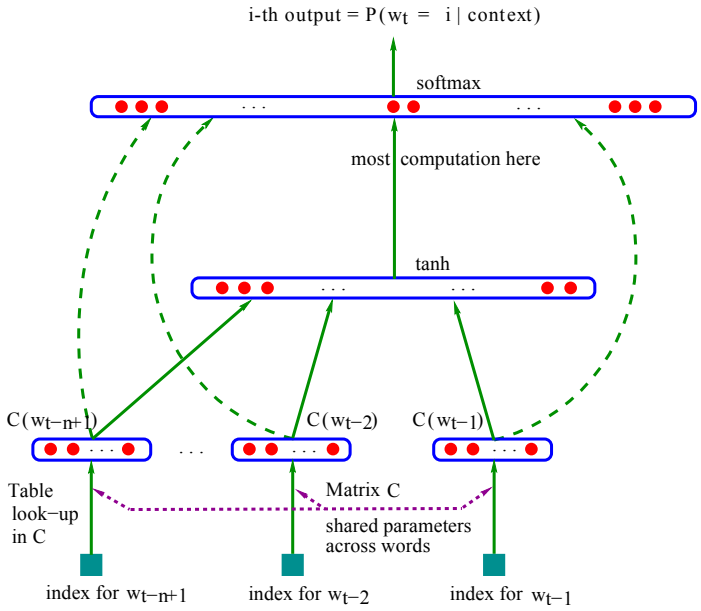

Figura 2.4: *Arquitetura de uma rede neural tradicional para modelagem de linguagem. Imagem extraída de* Bengio *et al., 2003.*

O treinamento dessa arquitetura de redes neurais tradicionais, apresentada por Bengio *et al.* (2003), é realizado por meio da busca iterativa do valor θ que maximize a verossimilhança em relação às probabilidades logarítmiticas de um *corpus* de tamanho T, representada pela equação 2.3. O algoritmo utilizado para essa busca é o de subida estocástica do gradiente (Harrington, 2012).

$$\mathcal{L}(\theta) = \frac{1}{T} \sum_{t=1}^{T} \log \widehat{P}(w_t | w_{t-1}, \cdots, w_1; \theta) \tag{2.3}$$

À época, esses modelos representaram um marco importante no âmbito do NLP. Além da abordagem pioneira na representação de palavras, futuramente estabelecidos como os *embeddings*, sua capacidade de aprender funções que estimam probabilidades oferece uma base sólida para a compreensão e geração de linguagem.

Todavia, a utilização dessa abordagem para lidar com textos suscitou muitas limitações na área. Uma delas é a incapacidade de capturar a dependência temporal entre as palavras em uma sequência, já que essas redes processam cada entrada de forma independente, sem considerar a ordem ou a relação cronológica entre vocábulos. Isso dificulta o uso do modelo para análise de textos cuja estrutura sequencial é essencial para o entendimento, como na avaliação automática de redações.

Outro desafio enfrentado pelas redes neurais tradicionais é a dificuldade em lidar com *corpus* de diferentes comprimentos. Como essas estruturas têm uma arquitetura fixa com um número predefinido de camadas e unidades, torna-se custoso processar sequências de diferentes tamanhos de forma eficiente, já que o modelo possui uma complexidade computacional exponencial em relação ao tamanho da entrada.

2.4.2 Redes Neurais Recorrentes (RNN)

Para superar as limitações das redes neurais tradicionais em representar relações temporais e de ordem, surgiram as Redes Neurais Recorrentes (RNNs), baseadas na arquitetura proposta por Rumelhart *et al.* (1986). As RNNs têm a habilidade de aprender padrões de qualquer sequência de dados, como genomas, séries temporais, entre outros. Tal capacidade as torna particularmente adequadas para lidar, também, com tarefas de compreensão de textos, tendo em vista o caráter sequencial dessas estruturas (Zhang *et al.*, 2023).

A arquitetura se diferencia principalmente pela introdução de um componente de memória, que é atualizado a cada novo elemento da sequência. A cada passo, a saída gerada pela rede realimenta a entrada, fazendo com que as RNNs levem em conta não só o novo dado, mas também as informações já processadas anteriormente. Assim, para um passo de tempo t e uma função de ativação f, a saída da rede h_t é calculada pela equação 2.4:

$$h_t = f(\boldsymbol{W}_{ih}x_t + \boldsymbol{b}_{ih} + \boldsymbol{W}_{hh}h_{t-1} + \boldsymbol{b}_{hh}) \tag{2.4}$$

em que x_t é a entrada no tempo t, $\boldsymbol{W}_{ih}$ os pesos da camada oculta, $\boldsymbol{W}_{hh}$ os pesos da ciclagem de informação (de camada oculta para camada oculta), $\boldsymbol{b}_{ih}$ os vieses referentes à entrada atual e $\boldsymbol{b}_{hh}$ os vieses relacionados com o histórico temporal. O termo h_{t-1} representa o estado oculto anterior, incorporando informações das etapas de tempo passadas.

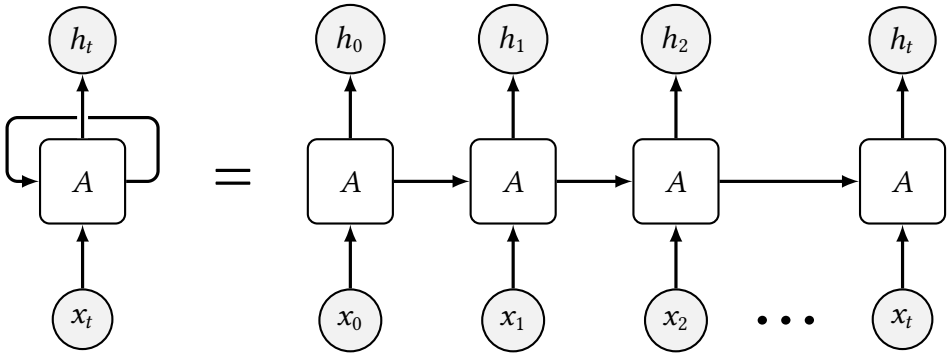

Figura 2.5: *Arquitetura de uma RNN em formato simplificado e em formato sequencial aberto. Imagem adaptada de Zhang et al., 2023.*

Para o treinamento, as RNNs utilizam o algoritmo *backpropagation through time* (BPTT), que é uma variação do algoritmo de retropropagação de redes tradicionais (Schmidt, 2019). O BPTT, de modo geral, atua criando uma versão sequencial aberta das RNNs ao longo do tempo, conforme mostrado na figura 2.5. Essa representação é, então, tratada como uma estrutura tradicional, permitindo a aplicação do algoritmo de *backpropagation* comum.

Algumas variações de RNNs, como as *Bidirectional Recurrent Neural Networks* (BRNN), também surgiram para atender outras demandas específicas da área de NLP, como a previsão de palavras em meio de sequências. Esses modelos incorporam a capacidade de processar a informação tanto no sentido direto quanto no inverso, permitindo que a previsão de uma palavra considere não apenas o contexto precedente, mas também o subsequente. A estrutura esquemática das BRNNs pode ser vista na figura 2.6.

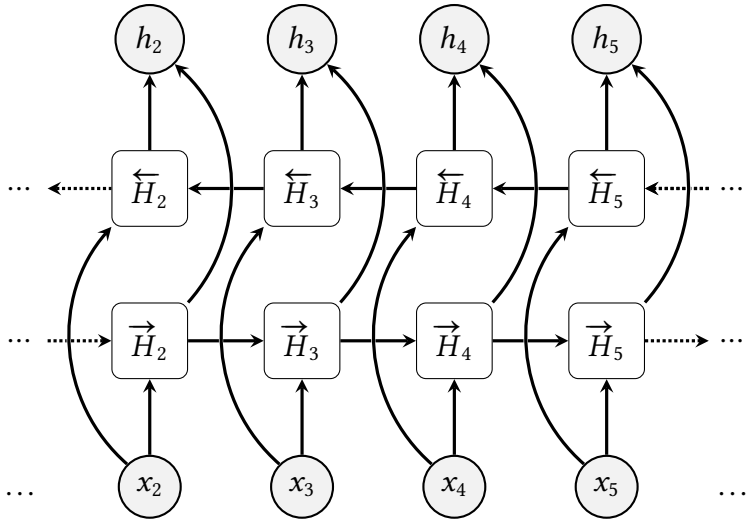

Figura 2.6: *Arquitetura de uma Bidirectional Recurrent Neural Network (BRNN). Imagem adaptada de Zhang et al., 2023.*

A possibilidade de manter estados internos ao longo das sequências permite que as RNNs capturem dependências de longo alcance em textos, um desafio enfrentado pelas redes neurais tradicionais. Além disso, a abordagem temporal habilita a utilização delas em *corpus* de tamanhos variáveis, garantindo que se adaptem a diferentes contextos.

No entanto, a arquitetura recorrente também apresenta desafios, sendo o mais proeminente o problema de desvanecimento ou explosão do gradiente. Durante o treinamento, os vetores responsáveis por minimizar a função de perda podem se tornar extremamente pequenos ou grandes, dificultando a atualização eficiente dos parâmetros da rede. Por estarem associadas aos pesos da camada oculta, essas variações resultam em dificuldades para aprender dependências temporais de longo prazo, já que o histórico pode ser pouco relevante ou muito influente, a depender dos gradientes. Isso compromete principalmente a capacidade da RNN de reter informações pertinentes em sequências extensas, como já pontuado por Schmidt (2019).

Ademais, as redes neurais recorrentes podem ser consideradas ineficientes e pouco escaláveis em relação a outras abordagens. Devido ao seu caráter sequencial, elas não

conseguem processar entradas paralelamente, o que aumenta o tempo de treino de forma substancial.

Para superar o problema do desvanecimento do gradiente, foram propostas variantes de RNNs, como as redes *Long Short-Term Memory* (LSTM) e as *Gated Recurrent Units* (GRU).

2.4.2.1 *Long Short-Term Memory* (LSTM)

As LSTMs são arquiteturas de RNNs que introduzem unidades de memória atualizadas e controladas por meio de portas (Hochreiter e Schmidhuber, 1997). Isso permite que elas capturem relações de longo prazo em sequências sem uma propagação de erros que cresça descomedidamente, tornando-as especialmente eficazes em tarefas de NLP que usam a abordagem recorrente.

A estrutura de uma LSTM é formada por células de memória, que são envolvidas por três portas: de entrada ($\boldsymbol{I}_t$), esquecimento ($\boldsymbol{F}_t$) e saída ($\boldsymbol{O}_t$). A porta de entrada controla a atualização da célula de memória, a de esquecimento regula a retenção de informações em relação às computações anteriores e a de saída determina o valor final da unidade.

$$\boldsymbol{I}_t = f\left(\boldsymbol{W}_{xi}\boldsymbol{X}_t + \boldsymbol{W}_{hi}\boldsymbol{H}_{t-1} + \boldsymbol{b}_i\right) \tag{2.5}$$

$$\boldsymbol{F}_t = f\left(\boldsymbol{W}_{xf}\boldsymbol{X}_t + \boldsymbol{W}_{hf}\boldsymbol{H}_{t-1} + \boldsymbol{b}_f\right) \tag{2.6}$$

$$\boldsymbol{O}_t = f\left(\boldsymbol{W}_{xo}\boldsymbol{X}_t + \boldsymbol{W}_{ho}\boldsymbol{H}_{t-1} + \boldsymbol{b}_o\right) \tag{2.7}$$

As portas são calculadas pelas equações 2.5, 2.6 e 2.7, respectivamente, em que $\boldsymbol{W}_{xi}$, $\boldsymbol{W}_{hi}$, $\boldsymbol{W}_{xf}$, $\boldsymbol{W}_{hf}$, $\boldsymbol{W}_{xo}$ e $\boldsymbol{W}_{ho}$ são matrizes de pesos e $\boldsymbol{b}_i$, $\boldsymbol{b}_f$ e $\boldsymbol{b}_o$ são vetores de viés. A função de ativação f utilizada nas portas é a sigmoide, dada pela equação 2.8, que retorna valores entre 0 e 1. Ela é utilizada para controlar a quantidade de informação que deve ser passada para a célula de memória.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

Conforme pontuado em Schmidt, 2019, as LSTMs têm, ainda, um componente de memória $\tilde{\boldsymbol{C}}_t$, cuja atualização é controlada pela combinação com as portas da célula. O principal objetivo desse elemento é regular a quantia de informação que deve ser passada para um novo estado de memória $\boldsymbol{C}_t$ com base no estado anterior $\boldsymbol{C}_{t-1}$. O cálculo de $\tilde{\boldsymbol{C}}_t$ é dado pela equação 2.9:

$$\tilde{\boldsymbol{C}}_t = \tanh\left(\boldsymbol{W}_{xc}\boldsymbol{X}_t + \boldsymbol{W}_{hc}\boldsymbol{H}_{t-1} + \boldsymbol{b}_c\right) \tag{2.9}$$

em que $\boldsymbol{W}_{xc}$ e $\boldsymbol{W}_{hc}$ são matrizes de pesos e $\boldsymbol{b}_c$ é um vetor de viés. A função de ativação de tangente hiperbólica é utilizada para normalizar os valores da célula de memória, retornando valores entre -1 e 1. Por fim, o estado interno da célula de memória $\boldsymbol{C}_t$ é calculado pela expressão 2.10 e a saída da unidade $\boldsymbol{H}_t$ é dada pela equação 2.11, em que $\odot$ é a multiplicação elemento a elemento, chamada de produto de Hadamard.

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t \tag{2.10}$$

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t) \tag{2.11}$$

A Figura 2.7 ilustra a arquitetura geral das LSTMs.

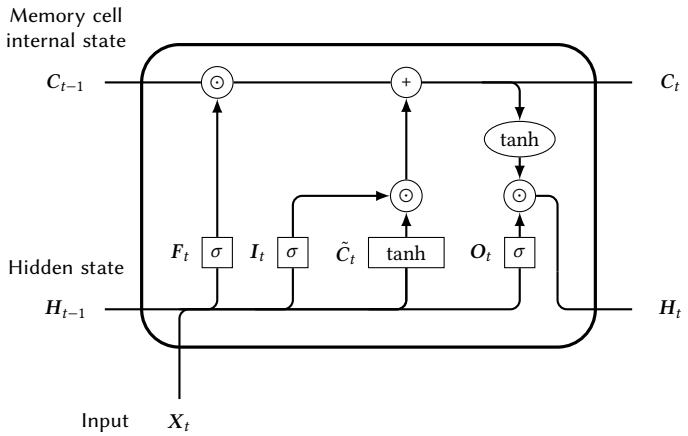

Figura 2.7: *Arquitetura de uma célula de memória das redes Long Short-Term Memory (LSTM) em um dado tempo. Imagem adaptada de* Zhang *et al., 2023.*

A capacidade de aprender dependências de longo prazo em sequências destaca as LSTMs das redes neurais recorrentes convencionais. A estrutura de memória permite que elas capturem relações entre elementos distantes em uma sequência sem que o gradiente desvaneça ou cresça excessivamente. Isso é, em particular, importante para lidar com textos extensos, como os de uma redação.

Entretanto, as LSTMs não são isentas de desvantagens. Uma delas é a complexidade computacional, já que a estrutura de memória e as portas adicionam uma quantidade considerável de parâmetros ao modelo. Somado a isso, assim como no caso das RNNs convencionais, a arquitetura sequencial dessas redes não permite um processamento paralelo, agravando o problema de eficiência no treino.

2.4.2.2 *Gated Recurrent Unit* (GRU)

Outra variação de RNNs é a *Gated Recurrent Unit* (GRU), que visa simplificar a arquitetura das LSTMs, mantendo uma desempenho de avaliação semelhante. A GRU possui menos parâmetros e apenas duas portas: uma de atualização e uma de reinício (Cho *et al.*, 2014). Isso a torna mais eficiente em termos computacionais e, em muitos casos, tão eficaz quanto as LSTMs.

A estrutura de uma GRU possui unidades de memória controladas por duas portas principais: a de atualização $\boldsymbol{Z}_t$ e a de reinício $\boldsymbol{R}_t$. Elas regulam o fluxo de informação de maneira eficiente à medida que atuam com menos parâmetros, sem perder a propriedade das LSTMs de assimilação de relações a longo prazo.

Em geral, a porta de atualização ajuda a capturar as dependências de longo prazo, enquanto a de reinício auxilia na modelagem das dependências de curto prazo. A primeira é calculada pela equação 2.12 e a segunda pela equação 2.13:

$$\boldsymbol{Z}_t = f\left(\boldsymbol{W}_{xr}\boldsymbol{X}_t + \boldsymbol{W}_{hr}\boldsymbol{H}_{t-1} + \boldsymbol{b}_r\right) \tag{2.12}$$

$$\boldsymbol{R}_t = f\left(\boldsymbol{W}_{xz}\boldsymbol{X}_t + \boldsymbol{W}_{hz}\boldsymbol{H}_{t-1} + \boldsymbol{b}_z\right) \tag{2.13}$$

em que $\boldsymbol{W}_{xr}$, $\boldsymbol{W}_{hr}$, $\boldsymbol{W}_{xz}$ e $\boldsymbol{W}_{hz}$ são matrizes de pesos, $\boldsymbol{b}_r$ e $\boldsymbol{b}_z$ são vetores de viés e f é a função de ativação sigmoide (2.8). Ambas as expressões são utilizadas para calcular um estado oculto intermediário $\tilde{\boldsymbol{H}}_t$, que é uma versão atualizada do estado anterior $\boldsymbol{H}_{t-1}$, conforme a equação 2.14.

$$\tilde{\boldsymbol{H}}_t = \tanh\left(\boldsymbol{W}_{xh}\boldsymbol{X}_t + \boldsymbol{W}_{hh}(\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}) + \boldsymbol{b}_h\right) \tag{2.14}$$

Na expressão, $\boldsymbol{W}_{xh}$ e $\boldsymbol{W}_{hh}$ são matrizes de pesos e $\boldsymbol{b}_h$ é um vetor de viés. A atualização da unidade de memória $\boldsymbol{H}_t$ é, então, calculada pela equação 2.15, conforme Zhang *et al.* (2023).

$$\boldsymbol{H}_t = (1 - \boldsymbol{Z}_t) \odot \tilde{\boldsymbol{H}}_t + \boldsymbol{Z}_t \odot \boldsymbol{H}_{t-1} \tag{2.15}$$

A Figura 2.8 apresenta a arquitetura da GRU, evidenciando as portas de atualização e reinício que controlam o fluxo de informação na unidade de memória.

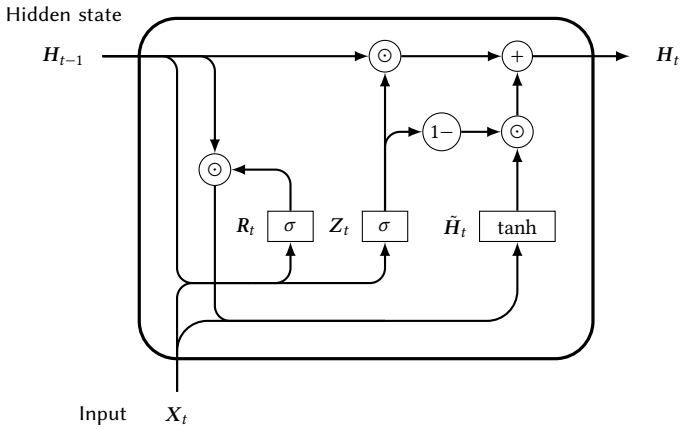

Figura 2.8: *Arquitetura de uma unidade de memória de uma Gated Recurrent Unit (GRU) em um dado tempo. Imagem adaptada de Zhang et al., 2023.*

As GRUs oferecem uma alternativa competitiva às LSTMs em muitas aplicações, principalmente quando a eficiência no treinamento é um requisito chave, já que ambas têm um desempenho prático semelhante. Além disso, esse modelo possui uma implementação mais simples, dada a menor complexidade de parâmetros a serem operados.

No entanto, as GRUs podem ser mais limitadas em tarefas de NLP que envolvam dependências de longo prazo, já que as LSTMs têm uma estrutura de memória mais robusta

capaz de assimilar adequadamente relações distantes em textos. Isso dificulta o uso desse modelo na área de avaliação automática de redações.

2.5 Modelos de Linguagem Baseados em Transformadores

Os modelos de linguagem baseados em transformadores representam uma evolução significativa no campo de NLP. Introduzidos por Vaswani *et al.* (2017), foram motivados principalmente pela tarefa de tradução automática de textos, a fim de superar as limitações apresentadas pelas estruturas *encoder-decoder* recorrentes e convolucionais.

A arquitetura dos transformadores é baseada no mecanismo de atenção, que permite ao modelo atribuir diferentes pesos a partes da entrada, codificando a importância de elementos da sequência. Essa abordagem, em especial, possibilita a modelagem de dependências entre a entrada e a saída, flexibilizando a necessidade de recorrência.

Os chamados *transformers* têm se destacado devido a sua eficácia em lidar com tarefas complexas de NLP, como tradução automática, resumo de textos e geração de respostas a perguntas. Além disso, eles são particularmente adequados para lidar com textos extensos, como os de uma redação, ao passo que permitem um processamento paralelo eficiente.

2.5.1 Atenção

A atenção é o cerne dos modelos de transformadores. Esse mecanismo permite assinalar diferentes importâncias aos *tokens* de entrada ao realizar uma tarefa, modelando a dependência entre eles. Seu cálculo considera uma pontuação para cada par de elementos em uma sequência, normalizada pela função *softmax* para obter os pesos.

O mecanismo pode ser representado por meio de uma matriz de atenção, em que cada linha e coluna representam um elemento da sequência. Para o caso dos transformadores, que buscam modelar a relação interna entre os próprios *tokens* de entrada, essa estrutura é chamada matriz de autoatenção. A figura 2.9 ilustra esse conceito.

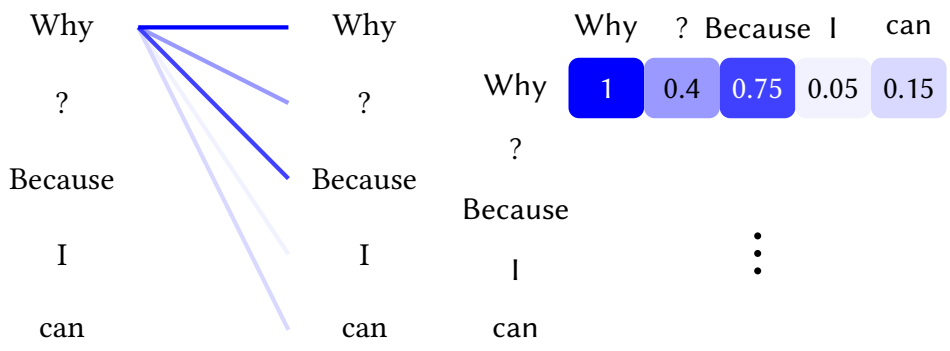

Figura 2.9: *Representação de uma matriz de autoatenção com os pesos do primeiro token em relação a sua própria frase. Imagem inspirada em Vaswani et al., 2017.*

Matematicamente, dada uma sequência de entrada $\boldsymbol{X} = (x_1, x_2, ..., x_n)$ e uma função de pontuação a que mede a relevância entre dois elementos, a atenção para x_i em relação a um termo k da sequência ($\boldsymbol{A}(x_i, x_k)$) é calculada por 2.16:

$$\boldsymbol{A}(x_i, x_k) = \frac{e^{a(x_i, x_k)}}{\sum_{j=1}^{n} e^{a(x_i, x_j)}} \tag{2.16}$$

em que a pode ser uma função que compara os elementos, como o produto escalar, por exemplo. Esse processo é executado para todos os elementos da sequência, gerando pesos de atenção que destacam a importância relativa de cada elemento para a tarefa em questão. Os *transformers* utilizam a função *scaled dot-product* para modelar as relações, que será abordada e detalhada mais à frente.

2.5.2 Arquitetura

A arquitetura dos modelos de linguagem baseados em transformadores, conforme proposta por Vaswani *et al.* (2017), é caracterizada pela presença de dois módulos principais: o de codificação e o de decodificação. Ambos os componentes são organizados em camadas empilhadas, compreendendo uma estrutura que tem se destacado em tarefas variadas no âmbito do NLP.

No nível de cada camada, a arquitetura incorpora módulos de atenção *multihead*, os quais desempenham um papel fundamental na modelagem de relações de dependência em diferentes partes do contexto. Esses módulos possibilitam a consideração simultânea de informações distantes, promovendo a captura eficiente de padrões sequenciais complexos. Adicionalmente, em cada camada são empregadas redes tradicionais (*feed-forward*) totalmente conectadas, enriquecendo a representação ao permitir a expressão de relações não lineares entre os elementos da sequência.

Além desses elementos, as conexões residuais são introduzidas em cada camada para abordar questões relacionadas à propagação do gradiente. Essas conexões, ao fornecer atalhos na passagem do gradiente, facilitam a otimização em profundidade, promovendo, assim, a eficácia do treinamento em arquiteturas mais extensas.

Após as conexões residuais, são feitas normalizações em cada camada, cujo objetivo é a manutenção da estabilidade e a facilitação da convergência durante o treinamento (Zhang *et al.*, 2023). Tal prática contribui para a eficácia do aprendizado ao mitigar possíveis desafios associados à variação na escala dos dados em arquiteturas profundas, como no caso dos *transformers*.

Como os transformadores não possuem uma estrutura sequencial inerente, o modelo incorpora uma etapa de codificação posicional, essencial para fornecer informações sobre a ordem das palavras em uma sequência de entrada. Essa codificação pode ser, ainda, pré-determinada ou aprendida pela rede. Ao adicionar dados de posição às representações vetoriais, a técnica capacita o mecanismo de atenção a antecipar a ordem temporal das palavras, permitindo um melhor desempenho dos *transformers* em contextos linguísticos complexos.

Em suma, os modelos baseados em transformadores apresentam uma configuração estratificada que integra diversos componentes para realizar operações de atenção, conectividade total e normalização, culminando em uma abordagem poderosa para lidar, em especial, com linguagens. A figura 2.10 ilustra o diagrama de sua arquitetura, em que N é um número arbitrário pré-definido de camadas empilhadas.

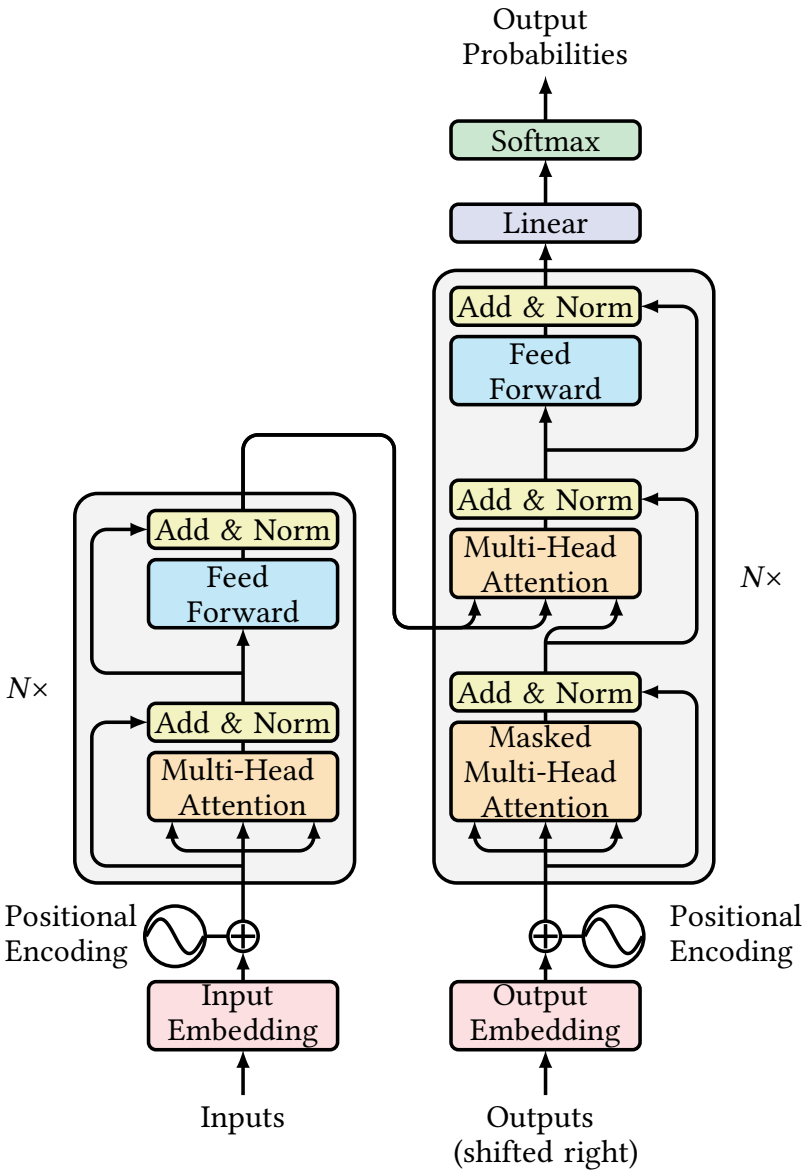

Figura 2.10: *Arquitetura de um modelo de linguagem baseado em transformadores. Imagem extraída de* Vaswani *et al., 2017.*

Na estrutura da figura 2.10, à esquerda, estão representados os módulos de codificação, que geralmente são utilizados para gerar representações matemáticas da entrada que capturem relações semânticas. Já à direita, estão os módulos de decodificação, mais focados em tarefas de previsão de *tokens* em frases, geração de textos e elaboração de respostas a perguntas.

O foco deste trabalho é a tarefa de avaliação automática de redações, que envolve, principalmente, a etapa de codificação das produções. A seguir, serão detalhados os mecanismos de atenção multicabeça e as subestruturas dos transformadores.

2.5.2.1 *Multihead Attention*

Em geral, o mecanismo de atenção, conforme Zhang *et al.* (2023), pode ser representado por meio de pares de chave-valor (k, v) e de consultadores q, em que podemos utilizar q para obter informações de uma base de dados de diferentes (k, v). Seu cálculo, nesse caso, é dado de acordo com a equação 2.17, em que $\mathcal{D}$ é o conjunto de pares chave-valor $(\mathcal{D} = \{(k_1, v_1), \cdots, (k_m, v_m)\})$.

$$\boldsymbol{A}(\mathcal{D}, q) = \sum_{i=1}^{m} \boldsymbol{\alpha}(q, k_i) v_i \tag{2.17}$$

A atenção *multihead* é uma extensão desse mecanismo, que permite que o modelo aprenda diferentes tipos de relações entre os elementos da sequência, ao mesmo tempo em que reduz a complexidade computacional ao considerar uma abordagem paralelizável com matrizes de consultadores (Q), chaves (K) e valores (V) (Vaswani *et al.*, 2017).

Os pesos, nessa técnica, são calculados por meio de uma projeção linear da entrada, seguida da função de atenção *scaled dot-product* abaixo e de uma projeção linear da saída. O cálculo dessa função utiliza uma escala inversamente proporcional à dimensão dos consultadores e chaves d_k (2.18):

$$\boldsymbol{\alpha}(Q, K) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{2.18}$$

O valor da atenção resultante é, então, dado pela equação 2.19.

$$\boldsymbol{A}(Q, K, V) = \boldsymbol{\alpha}(Q, K)V \tag{2.19}$$

Ao fim, o resultado de cada cabeça é concatenado e novamente projetado linearmente para formar a representação final $\boldsymbol{M}_a$, cujo cálculo é dado pela equação 2.20:

$$\boldsymbol{M}_a = \text{concat}(\boldsymbol{H}_1, \cdots, \boldsymbol{H}_h) W^O \tag{2.20}$$

em que $\boldsymbol{H}_i$, para $i \in \{1, \cdots, h\}$, é a saída da i-ésima cabeça de atenção e W^O é uma matriz de pesos. $\boldsymbol{H}_i$ é dado, em função de $\boldsymbol{A}$, pela expressão 2.21.

$$\boldsymbol{H}_i = \boldsymbol{A}(\boldsymbol{Q}_i W_i^Q, \boldsymbol{K}_i W_i^K, \boldsymbol{V}_i W_i^V) \tag{2.21}$$

Em que W_i^Q, W_i^K e W_i^V são matrizes de pesos e $\boldsymbol{Q}_i$, $\boldsymbol{K}_i$ e $\boldsymbol{V}_i$ são vetores de consultadores, chaves e valores da i-ésima cabeça, respectivamente. A figura 2.11 ilustra o cálculo da atenção *multihead*.

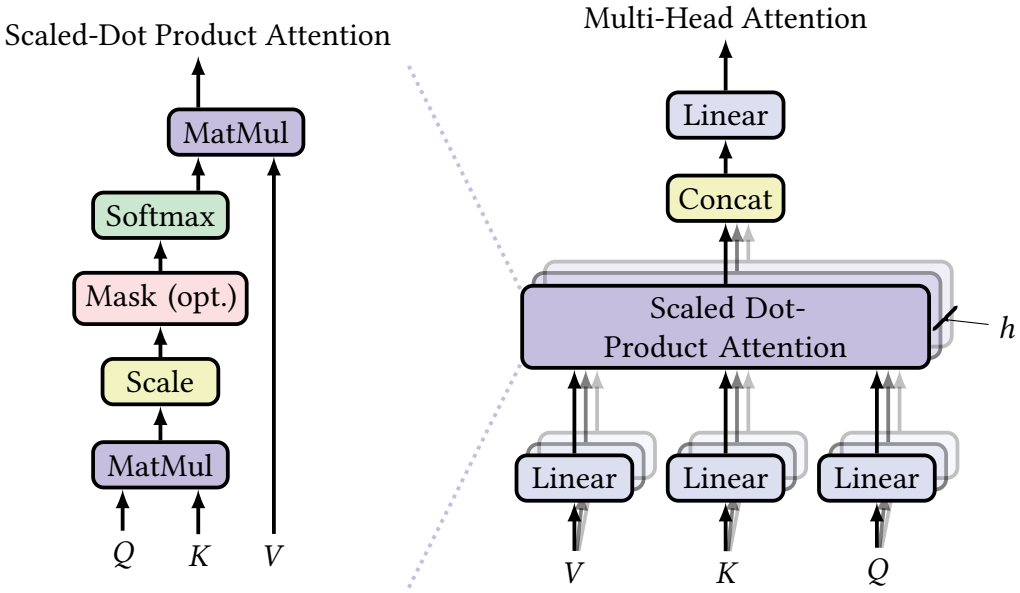

Figura 2.11: *À direita, o cálculo da atenção multihead. À esquerda, a função de atenção scaled dot-product. Imagem adaptada de* Vaswani et al., 2017.

2.5.2.2 Codificador

O codificador processa a entrada sequencial e produz uma representação contextualizada para cada elemento. Cada camada desse módulo contém duas subcamadas subsequentes: a de atenção *multihead* e a de redes neurais totalmente conectadas.

A subcamada de atenção *multihead* permite que o modelo aprenda diferentes tipos de relações entre os elementos da sequência já codificados. Cada cabeça opera em uma projeção linear da entrada, e as saídas são concatenadas e novamente projetadas linearmente para formar a representação final $\boldsymbol{X}$, após a normalização e conexão residual de $\boldsymbol{M}_a$, conforme mostrado na seção anterior.

Em seguida, a saída é submetida a uma rede neural totalmente conectada, que consiste em duas camadas lineares com uma função de ativação ReLU (indicada pela equação 2.22) entre elas. Essa subcamada permite que o modelo aprenda representações não lineares dos dados, enriquecendo a representação final.

$$f(x) = \max(0, x) \tag{2.22}$$

A saída de uma camada do codificador, então, é dada pela normalização e conexão residual da equação 2.23:

$$\boldsymbol{S} = f(\boldsymbol{W}_1\boldsymbol{X} + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{2.23}$$

em que $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_1$ e $\boldsymbol{b}_2$ são parâmetros da rede neural totalmente conectada (matrizes de pesos e vetores de vieses, nessa ordem) e f é a função de ativação ReLU (2.22).

2.5.2.3 Decodificador

O decodificador gera uma sequência de saída contextualizada com base na representação do codificador. Assim como no módulo de codificação, cada camada dessa etapa também possui subcamadas de atenção *multihead* e redes neurais totalmente conectadas. O que difere o módulo decodificador, entretanto, é a inclusão de uma nova camada de atenção com máscara antes da entrada das representações codificadas.

A máscara de atenção é utilizada para evitar que o modelo tenha acesso a informações futuras durante a geração da sequência de saída. Ela faz com que o mecanismo de atenção atue nas codificações dos elementos de modo a suprimir antecipações temporais da sequência pelos *transformers*.

O cálculo final de uma camada do codificador é dado de maneira similar ao da seção anterior, com a inclusão das codificações de entrada e da nova etapa de atenção com máscara.

2.6 BERT: *Bidirectional Encoder Representations from Transformers*

Os modelos baseados em transformadores representaram um marco significativo no avanço da área do NLP, redefinindo as fronteiras da compreensão de linguagem por máquinas. Entre as inovações motivadas pelo surgimento dos *transformers*, destaca-se o BERT (*Bidirectional Encoder Representations from Transformers*), um modelo introduzido por Devlin *et al.* (2018) que revolucionou o processamento de sequências textuais pelo computador.

O BERT se diferencia ao capturar, de maneira única, o contexto bidirecional das palavras em uma sequência, superando as limitações dos modelos unidirecionais. Sua arquitetura, baseada em múltiplas camadas de transformadores empilhadas, permite uma codificação contextual robusta dos elementos, capturando relações não só com as palavras anteriores, mas também subsequentes.

O processo de pré-treinamento do BERT permite ao modelo compreender contextos de forma sofisticada, promovendo uma representação rica de palavras ou *tokens* em textos. Além disso, a arquitetura pode ser utilizada como base para o treinamento de sistemas especialistas por meio, geralmente, do aprendizado supervisionado. Neste trabalho, por exemplo, o BERT é refinado para atuar como um classificador de textos, com o objetivo de avaliar redações no modelo do ENEM.

A fim de se adequar à diversidade linguística, foram desenvolvidas algumas variações do modelo a partir de seu pré-treinamento em conjuntos de textos específicos de uma dada linguagem. Dentre as versões, destaca-se o BERTimbau, uma variante do BERT com foco no português brasileiro (Souza *et al.*, 2020). As diferentes alternativas existentes demonstram a capacidade de generalização do modelo, que é capaz de aprender particularidades semânticas e sintáticas de diversas línguas.

O BERT representa não apenas um avanço técnico, mas também uma ferramenta essencial para diversas aplicações em NLP, incluindo a tarefa de AES. A seguir, abordaremos

brevemente os aspectos de sua arquitetura, o processo de treinamento e refinamento, a variação do modelo para a língua portuguesa e, por fim, como o BERT pode ser usado para atribuir notas às redações do ENEM de forma automática.

2.6.1 Arquitetura

A arquitetura do BERT é meticulosamente projetada para capturar de forma eficaz as relações contextuais em uma sequência de palavras, incorporando transformadores bidirecionais em múltiplas camadas. Cada camada desempenha um papel crucial na codificação contextual, permitindo que o modelo compreenda as dependências tanto à esquerda quanto à direita de cada unidade. A Figura 2.12 esquematiza a arquitetura geral do BERT.

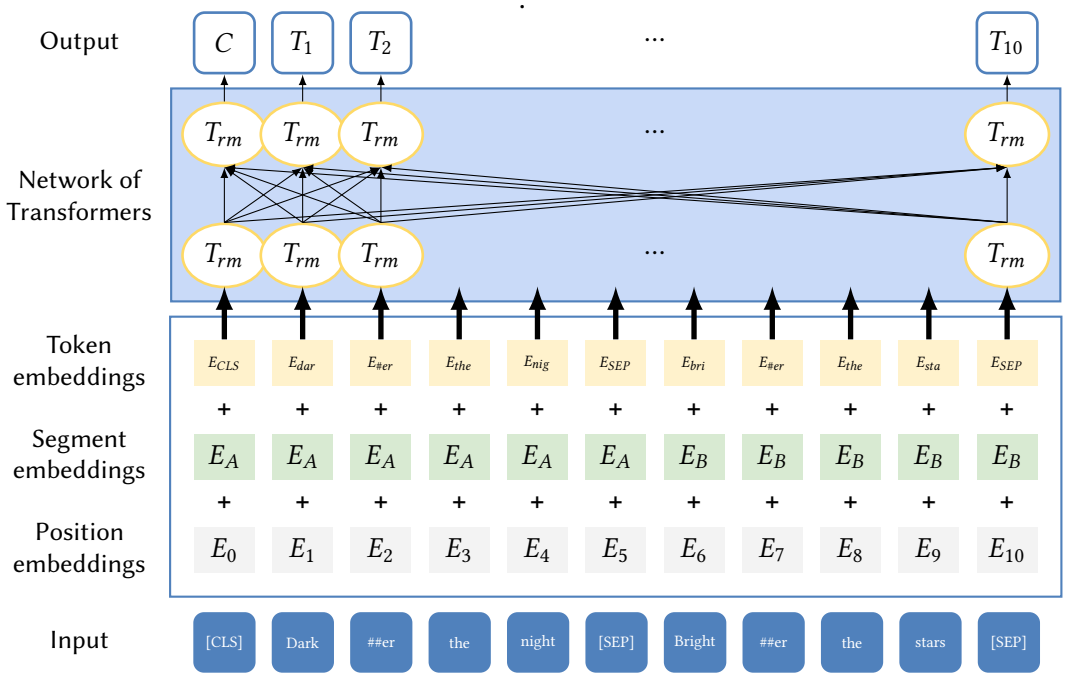

Figura 2.12: *Arquitetura geral do BERT, considerando a codificação dos tokens e as camadas de transformadores. Imagem adaptada de J. Sun et al., 2022*

Antes de serem processadas pelas camadas de transformadores, as sequências de palavras de entrada são codificadas em *tokens*. A tokenização divide as palavras em unidades menores, permitindo que o modelo lide de maneira mais flexível com diferentes formas e variações linguísticas. A tokenização em subpalavras também ajuda no desafio de lidar com o vocabulário diversificado e a variabilidade morfológica presente em diferentes idiomas.

Alguns *tokens* especiais também são levados em conta na modelagem. O *token* `[CLS]`, por exemplo, é adicionado ao início de cada sequência e representa a tarefa de classificação da sentença. Já o *token* `[SEP]` é adicionado ao final de cada sentença, indicando seu fim. Esses elementos especiais são importantes, em particular, na etapa de treinamento, já que tanto pares de pergunta e resposta quanto textos completos a serem classificados podem ser modelados sem mudar o formato da sequência de entrada (Devlin *et al.*, 2018).

O BERT mantém, além disso, vetores que codificam a posição e o segmento de cada *token* na sequência, permitindo que o modelo entenda a ordem de palavras e o mecanismo de separação de sentenças. Essas codificações são essenciais para a compreensão de textos, já que a posição das palavras é relevante para o significado.

As camadas de transformadores, então, recebem como entrada os vetores de *tokens*, posição e segmento, codificando relações contextuais entre as palavras. Cada camada, por meio das conexões bidirecionais entre todos os *transformers* anteriores, gera uma representação dos vocábulos de entrada. Isso possibilita ao BERT capturar interações complexas entre os elementos, fornecendo uma compreensão aprimorada de uma sequência. As saídas de todas as camadas são combinadas para formar uma representação final enriquecida.

Os autores Devlin *et al.* (2018) propõem duas arquiteturas do BERT: uma basilar (**BERT$_{\text{BASE}}$**) de 12 camadas de transformadores empilhadas, com 12 cabeças de atenção em cada uma, e outra estendida (**BERT$_{\text{LARGE}}$**) de 24 camadas de transformadores, com 16 cabeças de atenção em cada uma.

2.6.2 Pré-treinamento e *Fine-tuning*

O pré-treinamento é a etapa em que um modelo de linguagem é capaz de aprender aspectos semânticos e contextuais das sequências, a partir de um grande conjunto de dados. Para cumprir esse objetivo, o BERT é submetido a duas tarefas: a predição de palavras mascaradas e a identificação de relação entre duas sentenças (Zhang *et al.*, 2023).

A primeira tarefa, chamada *Masked Language Model* (MLM), consiste em mascarar aleatoriamente 15% dos *tokens* de entrada e treinar o modelo para prever as palavras mascaradas com base no contexto global. Essa abordagem promove a compreensão bidirecional de uma sequência, já que as relações de dependência interna de um texto são assimiladas à medida que o BERT otimiza seus parâmetros para cumprir tal função.

Já a segunda tarefa, chamada *Next Sentence Prediction* (NSP), envolve a apresentação de pares de sentenças ao modelo, que deve prever se uma ocorre após a outra no texto original. Essa atribuição aprimora a capacidade do BERT de entender relações semânticas, sendo especialmente útil em contextos de resposta a perguntas ou realização de inferências lógicas.

O pré-treinamento do BERT, em Devlin *et al.*, 2018, foi realizado a partir de um conjunto de dados de 3,3 bilhões de palavras, das quais 2,5 bilhões são de artigos do Wikipédia em inglês e 800 milhões são do BookCorpus, um compilado de cerca de 7000 livros autopublicados (Zhu *et al.*, 2015). Além disso, para a tokenização das palavras, foram utilizadas as codificações do algoritmo WordPiece (Wu *et al.*, 2016), com um vocabulário de 30000 *tokens*.

Após o pré-treinamento, o BERT pode ser ajustado para tarefas específicas por meio de um processo chamado *fine-tuning*. Durante essa etapa, o modelo pode ser acoplado a outras arquiteturas, que são treinadas em conjuntos de dados rotulados de acordo com a tarefa desejada. Uma mesma versão pré-treinada do BERT pode ser aplicada a diferentes tarefas de NLP, uma vez que o modelo pré-treinado já possui uma compreensão sofisticada da

linguagem, o que caracteriza essa abordagem, também, como menos computacionalmente custosa.

A figura 2.13 ilustra o processo de pré-treinamento e *fine-tuning* do BERT.

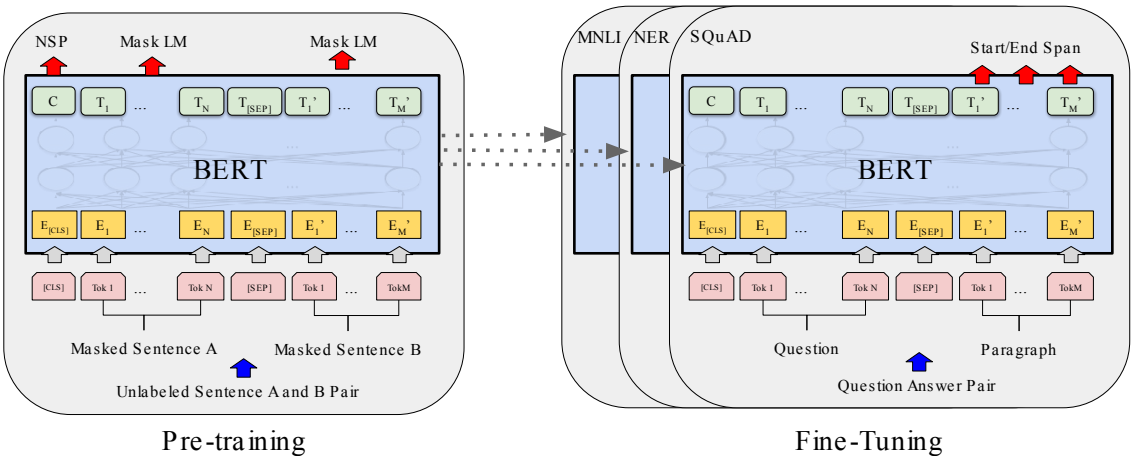

Figura 2.13: *Processo de pré-treinamento do BERT, à esquerda, e uso de modelos pré-treinados no fine-tuning, à direita. Imagem extraída de* Devlin *et al., 2018.*

2.6.3 BERTimbau: BERT para o Português Brasileiro

A demanda por capturar nuances complexas da língua portuguesa impulsionou o desenvolvimento do BERTimbau, uma adaptação do BERT para o português brasileiro (Souza *et al.*, 2020). Sem alterar a arquitetura ou o processo de pré-treinamento, essa versão usa uma base de textos distinta para ajustar os pesos do modelo, aprendendo a lidar com o vocabulário nacional.

O BERTimbau foi treinado utilizando o conjunto de dados brWaC (Wagner Filho *et al.*, 2018), uma extensa coleção de textos em português extraídos e filtrados de páginas da web. A base é formada por 2,7 bilhões de *tokens*, que foram reprocessados por Souza *et al.* (2020) para a retirada de *mojibakes*[1] e *tags* de marcação reminescentes.

A tokenização do conjunto de dados, crucial para lidar com a morfologia variada do português, foi realizada pelo algoritmo SentencePiece (Kudo e Richardson, 2018), com posterior conversão para o formato do WordPiece a fim de manter consistência com os *tokens* do BERT original.

O processo de treinamento do BERTimbau foi realizado com as mesmas duas tarefas de Devlin *et al.* (2018), a partir da predição de palavras ocultas em sentenças e da identificação de relação entre duas frases. Essas etapas contribuem, em especial, para a capacidade do modelo em compreender complexidades semânticas do português, aprimorando sua eficácia ao lidar com textos nessa língua. O BERTimbau também está disponível nas variantes

[1] Os *mojibakes* são anomalias que acometem textos da internet devido a erros de interpretação de caracteres em diferentes codificações. A exemplo, a palavra "redação", na codificação UTF-8, será mostrada como "redaÃ§Ã£o", na codificação ISO-8859-1. Esses problemas são, em particular, comuns na língua portuguesa devido à riqueza em acentuação.

basilar e extensa, ambas com a mesma quantidade de parâmetros das versões do BERT original, o que permite adaptações de acordo com a necessidade da tarefa final a ser realizada.

A disponibilidade de um modelo pré-treinado em língua portuguesa proporciona uma base sólida para uma gama de aplicações em contexto nacional, incluindo a avaliação automática de redações no formato do ENEM, explorada nesta monografia.

2.6.4 O BERTimbau na Avaliação Automática de Redações do ENEM

Devido a sua capacidade de capturar relações complexas em textos da língua portuguesa, o BERTimbau pode ser utilizado na tarefa de AES no âmbito do ENEM. Com o modelo, que é responsável por codificar as redações em *embeddings*, é possível obter representações matemáticas ricas em significado, permitindo a utilização de técnicas de aprendizado de máquina capazes de se especializar em atribuir notas aos textos.

Destaca-se, também, que modelos de linguagem como o BERTimbau, que se beneficiam da paralelização, podem ser utilizados para avaliar redações em larga escala de forma relativamente barata, evitando os gargalos logísticos decorrentes de uma correção exclusivamente humana.

Algumas das técnicas de treinamento de sistemas especializados são os algoritmos supervisionados, que podem ser otimizados para seguir um padrão de um conjunto de dados rotulados. O BERTimbau, por exemplo, pode ser acoplado a uma rede neural, treinada para classificar redações de acordo com notas atribuídas por corretores humanos. Tal é a abordagem deste trabalho, que será melhor explorada no capítulo 3.

Capítulo 3

Metodologia

Com a fundamentação teórica fornecida na seção 2, exploramos as potencialidades do aprendizado profundo e do processamento de linguagem natural para desenvolver um modelo especializado na avaliação automática de redações, seguindo o formato do ENEM.

A base de dados Essay-BR, disponível em suas versões simples (Marinho *et al.*, 2021) e estendida (Marinho *et al.*, 2022), desempenhou um papel crucial no treinamento dos modelos avaliadores. Cada modelo foi treinado para atribuir notas a competências específicas do exame, oferecendo uma abordagem granular e detalhada para a avaliação automatizada de redações.

A arquitetura de cada sistema de avaliação foi concebida com base no processo de *fine-tuning* do BERT. A estrutura envolve uma camada de entrada para a codificação dos textos e uma camada de redes neurais especializadas na atribuição de notas. O modelo escolhido para o refinamento foi o BERTimbau, pré-treinado em uma ampla gama de textos em língua portuguesa, com o objetivo principal de capturar nuances linguísticas específicas — essenciais para a correta avaliação de redações no contexto do ENEM.

Após a fase de treinamento, submetemos cada modelo a uma avaliação rigorosa, empregando métricas de desempenho padrão, como proporção de correspondência exata e perda, além de métricas específicas do ENEM, como a avaliação consistente entre competências. Este processo visa assegurar não só a precisão geral dos sistemas avaliadores, mas também a coerência e alinhamento com os critérios estabelecidos pelo exame.

3.1 Bases de dados

Para treinar modelos de avaliação automática, partimos de uma abordagem de aprendizado supervisionado, utilizando conjuntos de redações já avaliadas por especialistas humanos. A base de dados que orientou a elaboração desse trabalho foi a Essay-BR, uma coletânea de textos no modelo do ENEM com avaliações que seguem a a matriz de referência do exame (Marinho *et al.*, 2021; Marinho *et al.*, 2022).

As redações foram extraídas de dois portais educacionais abertos do Brasil, Vestibular

UOL[1] e Educação UOL[2], que disponibilizam a correção dos textos por especialistas da área publicamente. Em ambos os *websites*, existem diferentes propostas de temas semelhantes às do ENEM, de modo que alunos podem submeter suas escritas para a avaliação de acordo com sua preferência.

Para popular a base, os autores da Essay-BR utilizaram um *web crawler*, sistema encarregado de raspar de dados de sites, que coletou o conteúdo das páginas de avaliação individual em diferentes eixos temáticos. Os resultados obtidos, então, foram filtrados, com o intuito de extrair o texto e outras informações relevantes das redações, como as avaliações por competências e a pontuação geral.

Neste trabalho, visando lidar com a base de textos, utilizamos a biblioteca Pandas[3], que fornece estruturas e ferramentas de análise de dados de alto desempenho, e a biblioteca Matplotlib[4], que permite a visualização das informações em diferentes formatos de gráficos. A Essay-BR possui duas variantes de tamanho, as quais detalharemos nas seções 3.1.1 e 3.1.2 a seguir.

3.1.1 Essay-BR Básica

A primeira versão da Essay-BR, a que vamos referenciar como básica, contém uma quantidade de 4570 textos coletados de dezembro de 2015 a abril de 2020, contemplando uma variedade de 86 temas distintos. Desse total, 798 redações foram coletadas do Educação UOL, enquanto que 3772 redações foram obtidas do Vestibular UOL, produzindo uma coletânea rica em diferentes conteúdos e estilos de escrita.

Cada instância na base de dados é caracterizada por cinco informações relacionadas às redações: o identificador do tema (`prompt`), o título (`title`), o texto propriamente dito (`essay`), as notas atribuídas às competências (`competence`), e a pontuação final (`score`). O título, por ser opcional, pode ser vazio. Além disso, a redação é representada como uma lista de parágrafos isolados.

A avaliação das redações é expressa por meio de notas atribuídas a diferentes competências (`competence`), representadas por uma lista de 5 números inteiros que variam de 0 a 200, com incrementos de 40. Tais notas, essenciais para uma correção multifacetada, são as responsáveis por formar a pontuação final (`score`). Embora o valor de `score` seja, em essência, uma informação redundante, já que pode ser derivado da soma das notas de cada competência, sua inclusão na base de dados revela-se valiosa para a análise exploratória dos dados.

A Figura 3.1 ilustra, de forma visual, a distribuição das pontuações finais e das notas individuais das competências na Essay-BR básica. Essa representação gráfica revela padrões iniciais sobre a variabilidade e a dispersão dos resultados obtidos pelos alunos, proporcionando uma visão panorâmica das características gerais de avaliação.

[1] https://vestibular.brasilescola.uol.com.br

[2] https://educacao.uol.com.br/bancoderedacoes

[3] https://pandas.pydata.org

[4] https://matplotlib.org

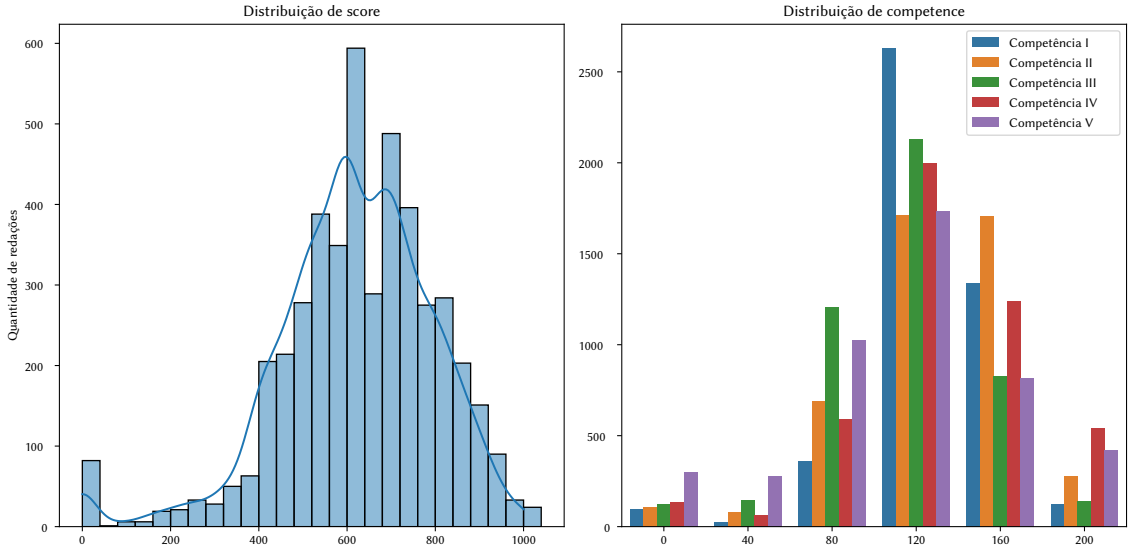

Figura 3.1: *Gráficos com a distribuição da pontuação final (à esquerda) e a distribuição das notas das competências (à direita) da Essay-BR básica.*

Pela figura 3.1, nota-se que o conjunto de dados apresenta uma distribuição com características próximas a de uma normal para as pontuações finais e notas de competências, contendo médias ligeiramente superiores às medianas. Destaca-se, ainda, a variabilidade nas notas de competências, sendo algumas mais concentradas — como a competência I — e outras mais dispersas — como a competência V.

Para a Essay-BR básica, os autores Marinho *et al.* (2021) disponibilizaram uma biblioteca em Python que visa facilitar a manipulação dos dados das redações, permitindo desde a criação de *dataframes*[5] até a divisão das instâncias em conjuntos de treino, teste e validação. O código-fonte e o conteúdo correspondente estão disponíveis no GitHub[6].

3.1.2 Essay-BR Estendida

A versão estendida da Essay-BR contém uma quantidade de 6579 textos em 151 temas distintos, coletados de dezembro de 2015 a agosto de 2021. Além das instâncias já mencionadas em 3.1.1, ela contém mais 1160 redações reunidas do portal Vestibular UOL, em um período de tempo mais atualizado, e 849 redações extraídas do trabalho de Amorim e Veloso (2017).

A base de dados contém instâncias com 9 informações diferentes a respeito das redações: o identificador do tema (`prompt`), o título (`title`), o texto (`essay`), as notas das competências (`c1`, `c2`, `c3`, `c4` e `c5`) e a pontuação final (`score`). Ao contrário da organização anterior, as notas das competências são separadas em 5 colunas distintas na versão estendida.

Para a análise exploratória dos dados, utilizamos a mesma abordagem da seção 3.1.1.

[5] Os *dataframes* são as estruturas de dados de tabela do Pandas, que permitem a visualização, manipulação e extração de informações de uma base.

[6] https://github.com/rafaelanchieta/essay

A figura 3.2 mostra a distribuição de pontuações finais e das notas das competências da Essay-BR estendida.

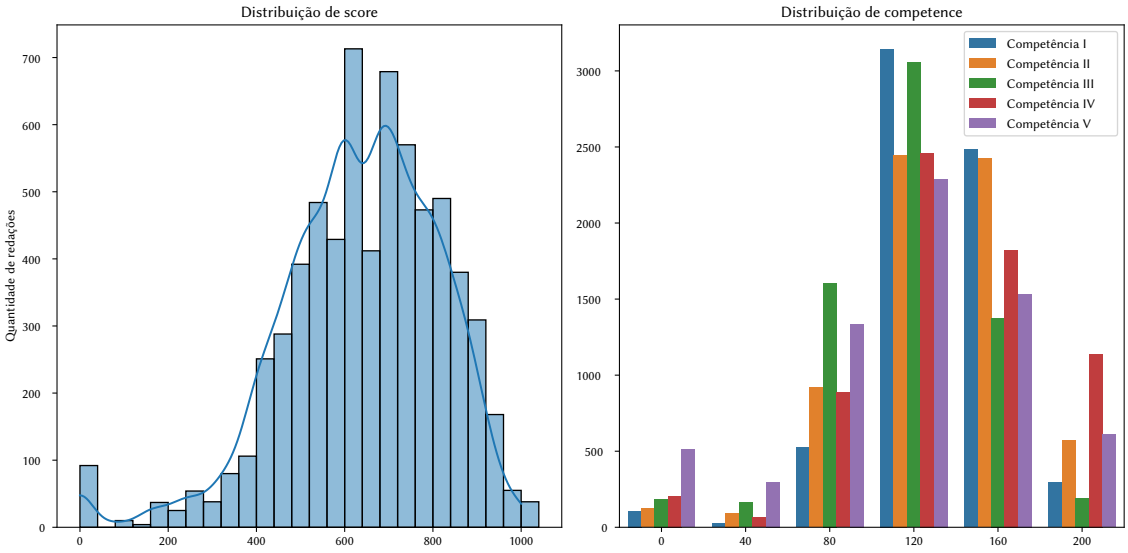

Figura 3.2: *Gráficos com a distribuição da pontuação final (à esquerda) e a distribuição das notas das competências (à direita) da Essay-BR estendida.*

Em geral, a partir da figura 3.2, é possível notar o mesmo padrão da versão anterior, mas com um pequeno acréscimo na frequência de notas finais maiores. Observa-se que esse aumento ocorreu, principalmente, devido ao incremento de notas 160 para a primeira competência, dado o aumento excepcional nessa categoria.

Os autores Marinho *et al.* (2022) também disponibilizaram uma biblioteca de manipulação de dados para a Essay-BR estendida, cujo código-fonte pode ser acessado pelo GitHub[7]. Entretanto, devido a pequenas diferenças na formatação e leitura dos dados, utilizamos, neste trabalho, um *fork*[8] adaptado da dependência.

A biblioteca atualizada tem o principal intuito de transformar o *dataframe* estendido no mesmo formato do básico, contraindo as competências a uma única coluna. O principal objetivo é manter uma consistência na *pipeline* de preparação dos dados, abordada na seção 3.2. Escolhemos a mesma convenção do Essay-BR básico devido a sua utilização mais frequente para experimentos iniciais do projeto.

3.2 Pré-processamento

Antes de iniciar o treinamento dos modelos, é crucial realizar o pré-processamento dos dados brutos para usá-los de maneira adequada. Essa etapa envolve transformações em sua formatação e estrutura, visando facilitar a extração eficiente de padrões. No contexto deste

[7] https://github.com/lplnufpi/essay-br

[8] https://github.com/josemayer/essay-br

trabalho, diversos tratamentos foram implementados, com foco principal na remodelação das redações e das avaliações para a otimização do processo de aprendizado.

Como passo inicial, realizamos a junção dos textos em uma única sequência, adotando uma formatação adequada para a entrada do modelo. Para isso, concatenamos todos os elementos da lista de parágrafos da redação, separando-os por caracteres marcadores de nova linha. A Figura 3.3 ilustra um exemplo de redação antes e depois do processo, destacando a simplificação da estrutura textual.

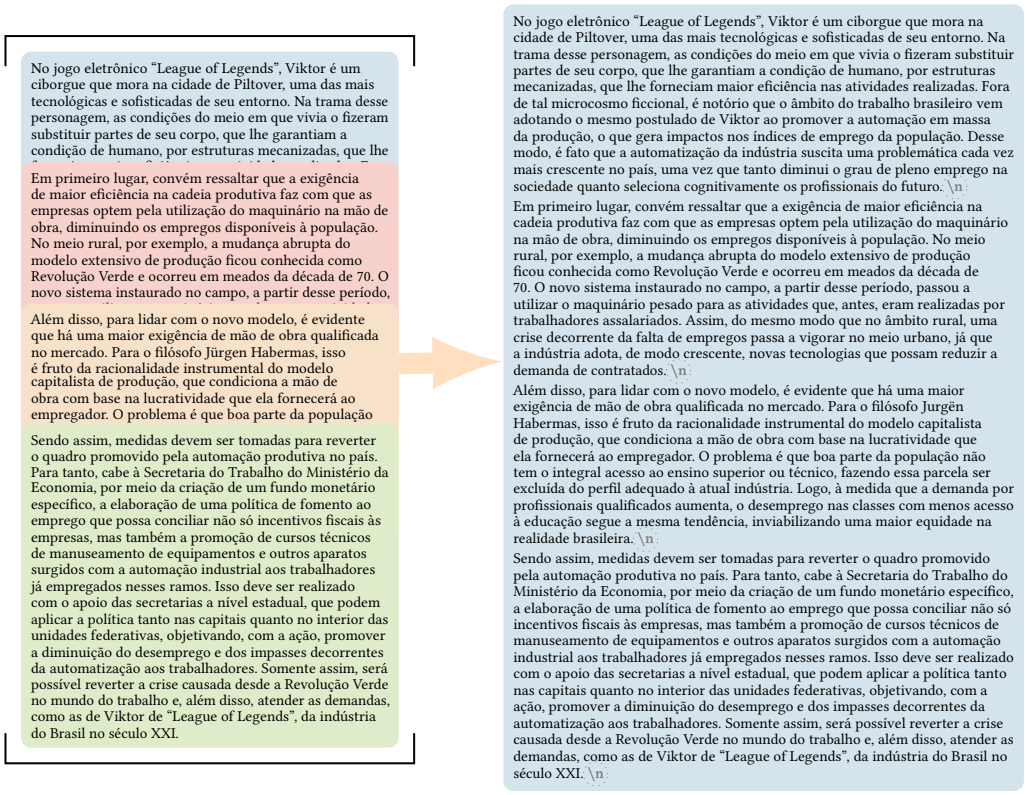

Figura 3.3: *Exemplo de redação antes e depois da concatenação dos parágrafos.*

Em seguida, normalizamos as notas atribuídas às competências, de modo que elas estejam no intervalo de 0 a 5. Para isso, dividimos cada pontuação por 40, diferença máxima de duas categorias de avaliação. Assim, cada aluno pode ter uma nota final inteira de 0 a 25, considerando a soma dos valores padronizados. É importante ressaltar que não normalizamos o campo `score`, dado que ele não é utilizado no treinamento.

Por fim, separamos as notas das competências em 5 colunas distintas, de `compI`, a `compV`, de modo que cada uma corresponda a uma pontuação. O objetivo principal é facilitar a construção dos cinco sistemas avaliadores isolados, extraindo apenas as colunas do texto e da nota da competência correspondente a cada treinamento.

3.3 Modelagem

A concretização deste trabalho envolveu a aplicação de técnicas avançadas de aprendizado profundo e NLP, abordagem crucial para a construção de sistemas avaliadores capazes de capturar nuances complexas da língua portuguesa. Além disso, a execução do projeto foi pautada em escolhas de ferramentas adequadas para o desenvolvimento das redes especialistas, levando em consideração o repertório de recursos voltados ao aprendizado de máquina.

Os sistemas de avaliação foram desenvolvidos por meio do processo de *fine-tuning* do BERT, utilizando a biblioteca de *transformers* do HuggingFace[9], uma plataforma aberta de *machine learning* e ciência de dados que oferece uma variedade de repositórios contendo, especialmente, modelos de linguagem abertos. A biblioteca disponibiliza uma interface de rápido acesso a arquiteturas pré-treinadas, como o BERTimbau.

Além disso, para desenvolver a estrutura do modelo especialista, utilizamos o Keras[10], que fornece uma abordagem de alto nível para projetar redes neurais. A biblioteca permite a especificação de diversos hiperparâmetros relevantes para o processo de treinamento, como a função de otimização e o tamanho dos lotes utilizados no processo, por exemplo.

Dentre as técnicas para a atribuição de nota, escolhemos a regressão, que é um método de aprendizado que mapeia uma entrada para um valor numérico contínuo. A decisão foi guiada pelo fato de que a distribuição das notas das competências é relativamente normal e unimodal, como mostrado na análise exploratória realizada anteriormente.

Por fim, escolhemos o erro médio quadrático (MSE) como métrica de desempenho prinicipal, utilizando-a para avaliar e orientar o treinamento dos modelos. Algumas outras técnicas de verificação de concordância, como a taxa de correspondência exata e a visualização das matrizes de confusão, também foram utilizadas para um julgamento empírico da eficácia dos sistemas avaliadores na prática.

3.3.1 Arquitetura

A arquitetura do modelo foi projetada concentrando-se em duas etapas fundamentais: a de codificação dos textos e a de avaliação propriamente dita. Inicialmente, estruturamos uma camada de entrada baseada no BERTimbau, cuja função primordial é receber os textos de redações e, por meio de sua capacidade de processamento contextual, gerar *embeddings* que capturam informações semânticas e sintáticas. Essa camada representa uma fase crucial da estrutura, permitindo ao modelo uma compreensão rica e contextualizada dos textos submetidos.

Posteriormente, projetamos uma camada de atribuição de notas, usando uma configuração de redes neurais com a saída associada a uma função de regressão linear. Essa camada é responsável por transformar a representação do primeiro elemento da sequência de saída do BERT, que corresponde ao *token* de classificação, em notas numéricas, refletindo a pontuação atribuída em relação à competência em foco.

[9] https://huggingface.co

[10] https://keras.io/

É essencial notar que, independentemente da competência, a arquitetura do sistema é generalizável o suficiente para aprender a atribuir as notas. Dessa forma, definimos a estrutura dos cinco modelos avaliadores como a mesma, com diferença apenas na variável de interesse no processo de treinamento. A Figura 3.4 abaixo permite a visualização da arquitetura de maneira esquemática, evidenciando as duas camadas de alto nível e a conexão entre elas.

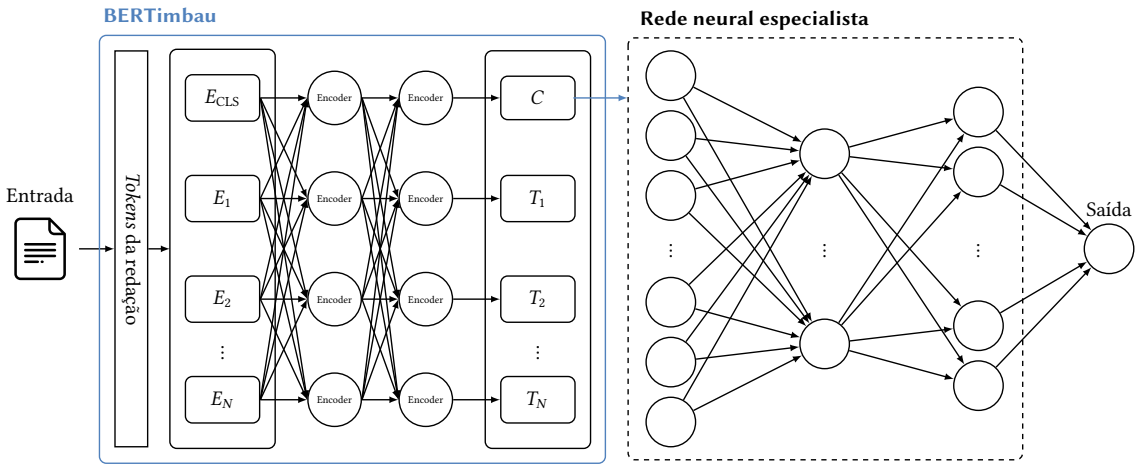

Figura 3.4: *Arquitetura geral de um sistema avaliador.*

Quanto à configuração da rede neural especialista, optamos por uma estrutura composta por três camadas ocultas, com dimensões definidas como 3000, 2000 e 2500 neurônios, respectivamente. A escolha dos tamanhos foi orientada pela necessidade de criar uma arquitetura suficientemente complexa para capturar as relações presentes nos textos das redações. Entretanto, uma quantidade grande de parâmetros pode ocasionar em problemas de sobreajustes do modelo.

Introduzimos, para mitigar esse risco, a função de regularização de *dropout* entre todas as camadas da rede neural especialista, com fator de 10%. Essa técnica tem o intuito de agir aleatoriamente, ignorando uma quantidade de pesos proporcional ao fator nas transições da etapa de treinamento. Dessa forma, ela garante uma maior generalização do modelo, contribuindo para melhor desempenho em dados não vistos.

Quanto aos cálculos realizados em cada camada, escolhemos duas funções de ativação com caráter não linear para a efetiva extração de características das redações. A primeira delas, SeLU, é eficaz em lidar com problemas de gradiente desvanecente e em proporcionar uma convergência mais estável durante o treinamento. A outra, sigmoide (2.8), é uma escolha comum para modelagem de problemas de classificação binária e pode ser efetiva em lidar com esse tipo de características dos textos. Por fim, para a camada de saída, utilizamos a função linear com o intuito de obter o resultado da regressão final. A SeLU é representada pela equação 3.1, em que $\alpha = 1,67326324$ e $\lambda = 1,05070098$.

$$\mathrm{f}(x) = \lambda \begin{cases} x, & \text{se } x > 0 \\ \alpha(e^x - 1), & \text{se } x \leq 0 \end{cases} \tag{3.1}$$

3.3.2 Implementação

A implementação da arquitetura é feita a partir do Keras, com uma classe de hipermodelos. Essa classe é útil, em especial, para a otimização de parâmetros que devem ser escolhidos de forma antecipada pelos projetistas, como a dimensão das camadas ocultas, por exemplo. Nela, adicionamos um atributo especial para armazenar o modelo do BERTimbau e redefinimos os métodos de construção da rede e de treino. O trecho de código em Python 3.1 mostra, na prática, a implementação.

Programa 3.1 Implementação em Python da classe `EssayHyperModel`

```
1    import tensorflow as tf
2    import keras_tuner as kt
3    from tf.keras.optimizers import Adam
4    from tf.keras.layers import Dense, Dropout, Input
5    from tf.keras.models import Model
6
7    class EssayHyperModel(kt.HyperModel):
8       def __init__(self, bert):
9         self.bert = bert
10
11      def build(self, hp):
12         input_ids = Input(shape=(None,), dtype=tf.int32, name="input_ids")
13         embedding = self.bert({'input_ids': input_ids})['pooler_output']
14
15         x = Dense(3000, activation=hp.Choice('a_l1', values=['selu', 'sigmoid'
               ]))(embedding)
16         x = Dropout(0.1)(x)
17         x = Dense(2000, activation=hp.Choice('a_l2', values=['selu', 'sigmoid'
               ]))(x)
18         x = Dropout(0.1)(x)
19         x = Dense(2500, activation=hp.Choice('a_l3', values=['selu', 'sigmoid'
               ]))(x)
20         x = Dropout(0.1)(x)
21
22         output = Dense(1, activation='linear')(x)
23
24         model = Model(inputs=input_ids, outputs=output)
25
26         optimizer = Adam(learning_rate=hp.Choice('lr', values=[2e-3, 2e-5]))
27         model.compile(optimizer=optimizer, loss='mean_squared_error', metrics
               =['mse'])
28
29         return model
30
31      def fit(self, hp, model, *args, **kwargs):
32         return model.fit(
33            *args,
34            batch_size=hp.Choice("bs", [2, 3, 4]),
35            **kwargs,
36         )
```

A primeira camada da rede consiste em uma entrada que recebe sequências de identificadores de *tokens* dos textos, provenientes do processo de tokenização. Essa camada

permite o processamento de sequências com comprimentos variáveis, sendo útil para lidar com redações de diferentes tamanhos. Os *tokens* de entrada são então processados pelo BERTimbau, gerando *embeddings* correspondentes à saída do *pooler* do modelo.

Em seguida, na estruturação da rede especialista, define-se uma arquitetura de três camadas densas, cada uma seguida por uma etapa de *dropout* para regularização. A escolha da função de ativação de cada nível é realizada de forma dinâmica, permitindo a personalização durante o processo de treinamento do modelo. Nas linhas 15, 17 e 19 são também definidas as dimensões das camadas densas: 3000, 2000 e 2500, respectivamente. A camada de saída, definida na linha 22, possui uma única unidade, utilizando ativação linear.

O otimizador escolhido para treinamento é o Adam, conforme a linha 26, e sua taxa de aprendizado é determinada como um dos parâmetros do espaço de busca hiperparamétrica. A função de perda adotada, explicitada na linha 27, é o erro quadrático médio (MSE), com o intuito de penalizar desvios significativos entre as predições do modelo e os valores de validação. O cálculo dessa métrica é dado pela equação 3.2, em que y_i são os valores esperados e $\hat{y}_i$ são os valores obtidos pelo sistema avaliador, para $i \in \{1, \cdots, n\}$.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.2}$$

O processo de treinamento do modelo é configurado para ajustar-se aos parâmetros definidos dinamicamente pelo otimizador, técnica chamada de *hypertuning*. O tamanho do lote, conforme a linha 34, também é tratado como um hiperparâmetro. Na seção 3.3.3, explicaremos o processo de *hypertuning* sobre um espaço de possibilidades limitado.

3.3.3 Otimização de Hiperparâmetros

A otimização de hiperparâmetros é uma técnica utilizada para encontrar a melhor configuração de parâmetros pré-definidos de um modelo de aprendizado de máquina. Essa abordagem é essencial para a obtenção de resultados satisfatórios, já que alguns fatores definidos a nível de arquitetura são muito influentes no processo de treinamento.

Para realizar a otimização, utilizamos a biblioteca Keras Tuner, que fornece uma interface de alto nível para o processo de *tuning*. Com ela, podemos definir um espaço de busca, que contém os parâmetros a serem otimizados, e um algoritmo de exploração, que define a estratégia de pesquisa nesse espaço. No caso do sistema especialista, optamos por realizar uma varredura completa nas possibilidades de hiperparâmetros, utilizando o algoritmo de busca em grade que analisa todas as combinações possíveis de variáveis.

O espaço de busca é definido ainda na criação da arquitetura dos hipermodelos, utilizando um método de escolha para definir as possibilidades desejadas. Na etapa de treino, cada iteração de configuração distinta é definida como uma *trial*, em que a rede neural é otimizada normalmente em um dado número de épocas e, depois, avaliada em um conjunto de validação. Essa avaliação é a responsável por indicar se os hiperparâmetros da *trial* melhoraram ou pioraram o resultado obtido. Ao fim de todo o processo, a melhor configuração é recuperada para a construção de uma nova rede.

No caso do sistema avaliador, os hiperparâmetros escolhidos foram a taxa de aprendizado (`lr`), o tamanho do lote (`bs`) e as funções de ativação das camadas ocultas (`a_l1`, `a_l2` e `a_l3`). A taxa de aprendizado é o fator que determina o tamanho do passo dado pelo otimizador de treino em cada iteração, enquanto que o tamanho do lote é a quantidade de instâncias utilizadas para o ajuste dos pesos do modelo a cada época. A tabela 3.1 lista cada um dos hiperparâmetros da classe `EssayHyperModel` (Programa 3.1) e suas possibilidades de escolha.

Hiperparâmetro	**Valores**
Taxa de aprendizado (`lr`)	2×10^{-3} e 2×10^{-5}
Tamanho do lote (`bs`)	2, 3 e 4
Função de ativação da camada 1 (`a_l1`)	SeLU e Sigmoide
Função de ativação da camada 2 (`a_l2`)	SeLU e Sigmoide
Função de ativação da camada 3 (`a_l3`)	SeLU e Sigmoide

Tabela 3.1: *Hiperparâmetros da classe `EssayHyperModel` escolhidos para otimização, com seus respectivos espaços de escolha.*

3.3.4 Avaliação

No contexto do aprendizado de máquina, a avaliação de modelos é uma etapa crucial para a verificação de sua eficácia. Neste trabalho, utilizamos uma abordagem de validação cruzada, que consiste em dividir a base de dados em conjuntos de treino e teste, de modo que o modelo seja treinado em uma parcela dos dados e avaliado em outra independente. Essa técnica é fundamental para a verificação da capacidade de generalização dos sistemas avaliadores, já que permite a análise de seu desempenho em dados não vistos.

A divisão das entradas foi realizada a partir da biblioteca de manipulação disponibilizada pelos autores da Essay-BR (Marinho *et al.*, 2021; Marinho *et al.*, 2022), de modo que o conjunto de teste representa 15% das instâncias totais, totalmente isoladas daquelas usadas no processo de treino. Além disso, como a saída da rede especialista retorna um valor contínuo, realizamos o tratamento das notas inferidas com uma função de arredondamento para um número inteiro mais próximo, a fim de adequá-lo ao formato da parcela de testes.

A avaliação dos modelos foi realizada com base nas métricas de MSE (Equação 3.2), proporção de correspondência exata, divergência e *quadratic weighted kappa* (QWK). A seguir, detalharemos cada uma delas.

3.3.4.1 Erro Quadrático Médio (MSE)

O erro quadrático médio (MSE) é uma métrica de desempenho comumente utilizada para avaliar modelos de regressão. Ela é definida como a média dos erros quadráticos entre as predições e os valores esperados, conforme a equação 3.2.

3.3.4.2 Proporção de Correspondência Exata

A proporção de correspondência exata é uma métrica de desempenho utilizada para avaliar modelos de classificação com múltiplas categorias. Ela é definida como a taxa de predições corretas em relação ao total de predições. Considerando $\hat{y}_i$ como o i-ésimo valor inferido e y_i como o i-ésimo valor real, definimos a proporção de correspondência exata pela equação 3.3, em que $I(\hat{y}_i = y_i)$ vale 1 caso os dois valores sejam iguais e 0 em caso contrário.

$$\text{MR} = \frac{1}{n} \sum_{i=0}^{n} I(\hat{y}_i = y_i) \tag{3.3}$$

3.3.4.3 Divergência

A métrica de divergência avalia a consistência interna da nota de uma competência, inspirada no processo de correção das redações do ENEM. Ela é calculada como a proporção de avaliações que diferiram em mais de 2 pontos para a saída do sistema avaliador. Essa convenção é feita com base na definição de Brasil (2023), que classifica como divergente duas correções cujas notas de competência difiram em mais de 100 pontos (2 pontos, na abordagem normalizada). Considerando a função I:

$$I(\hat{y}_i \lesseqgtr y_i) = \begin{cases} 1, & \text{se } |\hat{y}_i - y_i| > 2 \\ 0, & \text{caso contrário} \end{cases} \tag{3.4}$$

em que $\hat{y}_i$ é o i-ésimo valor inferido e y_i é o i-ésimo valor real, a divergência pode ser calculada pela equação 3.5.

$$\text{D} = \frac{1}{n} \sum_{i=0}^{n} I(\hat{y}_i \lesseqgtr y_i) \tag{3.5}$$

3.3.4.4 *Quadratic Weighted Kappa* (QWK)

O *quadratic weighted kappa* (QWK) é uma métrica de desempenho usada para avaliar modelos de classificação multicategóricos, introduzida por Cohen (1968). Essa métrica é definida através da comparação entre a concordância observada O entre as classificações e a concordância esperada ao acaso E, levando em consideração os pesos w_{ij} que refletem a importância das discordâncias entre diferentes categorias.

A fórmula do QWK é expressa pela equação 3.6, em que O_{ij} é o número normalizado de pares de instâncias classificadas como i por um avaliador e j por outro, E_{ij} é o número normalizado esperado de pares de instâncias que seriam classificadas como i por um avaliador e como j por outro, assumindo que não houvesse correspondência entre as avaliações, e w_{ij} são os pesos atribuídos aos pares de classes i e j, considerando a discordância quadrática relativa entre as categorias.

$$QWK = 1 - \frac{\sum_{i,j} w_{ij} O_{ij}}{\sum_{i,j} w_{ij} E_{ij}} \tag{3.6}$$

Na implementação do algoritmo de cálculo do QWK, utiliza-se matrizes O e E, de dimensões $i \times j$, para armazenar as informações de O_{ij} e E_{ij}. A matriz O é calculada a partir da matriz de confusão, enquanto que a matriz E é calculada pela distribuição marginal dos valores inferidos em relação aos valores reais.

Essa métrica varia no intervalo de -1 a 1, onde 1 indica concordância perfeita, 0 indica concordância ao acaso e valores negativos indicam discordância pior do que a esperada aleatoriamente. O QWK é particularmente útil em problemas onde as categorias são uma sequência, como no caso das notas das competências, considerando não apenas a correspondência direta entre as previsões e os rótulos reais, mas também a conformidade em termos de ordem.

De modo geral, podemos definir intervalos de interpretação para o QWK, conforme a tabela 3.2.

QWK	Interpretação
-1	Discordância completa
0	Concordância ao acaso
$0 - 0,2$	Concordância precária
$0,2 - 0,4$	Concordância moderada
$0,4 - 0,6$	Concordância boa
$0,6 - 0,8$	Concordância muito boa
$0,8 - 1$	Concordância quase perfeita
1	Concordância perfeita

Tabela 3.2: *Intervalos de interpretação para valores do quadratic weighted kappa (QWK). Tabela extraída de https://www.kaggle.com/code/prashant111/simple-explanation-of-quadratic-weighted-kappa.*

3.4 Treinamento

O treinamento é a etapa principal de ajuste da arquitetura, em que os sistemas aprendem os padrões de avaliação a partir da base de redações anotada. A biblioteca TensorFlow[11] foi a escolha principal para esse estágio, dado seu suporte eficiente para operações de álgebra linear e sua facilidade de utilização. Detalharemos, a seguir, o processo de otimização dos sistemas avaliadores, definindo melhor as configurações utilizadas e o ambiente de execução.

[11] https://www.tensorflow.org

3.4.1 Configurações

No processo de treinamento, foi utilizado um conjunto de dados complementar à divisão de testes da Essay-BR, com 85% do total de instâncias. Desse valor, 70% da base original foi alocada para a fase de otimização do modelo, enquanto que os 15% restantes foram reservados para a validação e ajuste da estrutura. No caso das redes especialistas, tal ajuste ocorreu por meio do uso do MSE como função de perda.

Alguns dos parâmetros de treinamento foram escolhidos de maneira automática, com o intuito de otimizar o desempenho geral dos avaliadores, como o tamanho do lote (*batch size*) e a taxa de aprendizado. Além disso, o Adam (*Adaptative Moment Estimation*), proposto por Kingma e Ba (2017), foi definido como otimizador da rede neural, devido à sua eficácia em combinar as vantagens do método de gradiente estocástico (SGD) com a adaptação da taxa de aprendizado ao longo do treino.

Durante a busca dos hiperparâmetros, foram utilizados dois *callbacks* personalizados para monitoramento e limpeza. O primeiro, `LogCallback`, tem o intuito de registrar as avaliações de cada *trial* e o segundo, `DeleteCallback`, visa remover salvamentos parciais de redes que possam sobrecarregar o disco. Cada iteração treina um modelo distinto por 5 épocas, atualizando o conjunto ótimo de hiperparâmetros ao longo do processo.

Ao fim da busca, os melhores valores são usados para a construção de um novo modelo, que é treinado em 50 épocas. Nessa etapa, utilizamos somente o *callback* `ModelCheckpoint` do Keras, a fim de salvar a rede com a melhor validação dentre todo o processo de otimização. Isso é feito para prevenir a ocorrência de sobreajustes, já que um número de épocas muito alto pode levar a tal risco. O trecho de código 3.2 abaixo demonstra a implementação das configurações abordadas, suprimindo algumas informações com "`...`" para fins de concisão.

Programa 3.2 Algoritmo do treinamento e busca de hiperparâmetros

```
hypermodel = EssayHyperModel(bert)

tuner = kt.GridSearch(hypermodel, objective='val_loss', executions_per_trial
    =1, ...)
tuner.search(..., epochs=5, callbacks=[LogCallback(...), DeleteCallback()])

best_model_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_model = tuner.hypermodel.build(best_model_hps)

history = best_model.fit(
   ...,
   epochs=50,
   batch_size=best_model_hps.get('batch_size'),
   callbacks=[ModelCheckpoint(filepath=(...), monitor='val_loss',
       save_best_only=True)]
)
```

3.4.2 Ambiente

O treinamento dos modelos foi realizado em quatro ambientes controlados, todos com *hardwares* adequados para lidar com a tarefa de processamento massivo. O primeiro deles, o Google Colab[12], foi utilizado no início do projeto para a realização de experimentos iniciais e para a construção de redes mais simples em *notebooks*, documentos interativos que permitem execução de código. Já o segundo, a Rede GNU/Linux do IME-USP[13], participou das etapas primitivas de *hypertuning* das redes especialistas e foi crucial para a transição do projeto para uma versão puramente em Python.

Na versão mais madura do trabalho, utilizamos unidades com maior poder computacional para a realização dos treinamentos e otimização de hiperparâmetros definivos. Uma delas, chamada PGM, é uma máquina física da seção de pesquisas do Departamento de Ciência da Computação (DCC) do IME-USP. A outra é uma máquina virtual hospedada no serviço Google Cloud[14].

Os quatro ambientes dispunham de GPUs que aceleraram o processo de treinamento, reduzindo o tempo de execução das iterações de *hypertuning*. Abaixo, são listadas as especificações de cada máquina usada.

- **Google Colab**: GPU Tesla T4, 16GB de GPU RAM.
- **Rede GNU/Linux**: GPU NVIDIA GeForce RTX 3060, 12GB de GPU RAM.
- **PGM**: GPU NVIDIA GeForce GTX 1080 Ti, 12GB de GPU RAM.
- **Google Cloud**: GPU NVIDIA Tesla P100, 16GB de GPU RAM.

[12] https://colab.research.google.com/

[13] https://www.linux.ime.usp.br/

[14] https://cloud.google.com/

Capítulo 4

Experimentos

Com base na metodologia detalhada no capítulo anterior, uma série de experimentos foi conduzida com o propósito de avaliar o desempenho dos sistemas de correção automática desenvolvidos. Durante essa etapa, exploramos diversas configurações treinando modelos não só com a versão estendida, mas também com a versão básica da Essay-BR — dada a variação dos conjuntos de dados.

Além disso, para garantir uma comparação adequada, foram incluídos experimentos de controle que utilizam hiperparâmetros fixados e conhecidos para o treinamento dos modelos, provenientes de explorações iniciais do projeto. Essa abordagem permite metrificar o impacto do *hypertuning* no desenvolvimento de redes especialistas de avaliação.

Por fim, visando explorar a capacidade de aprendizado dos modelos, foram utilizados gráficos que denotam a evolução de cada treinamento realizado. Isso proporciona uma compreensão mais abrangente da influência de escolhas estruturais no processo de otimização, além de indicar caminhos possíveis para contornar o subdesempenho dos sistemas de correção automática.

4.1 Treinamento

Nos experimentos de treinamento, optamos por manter o registro do histórico de desempenho apenas do conjunto de dados estendido, para fins de concisão. No entanto, ressaltamos que a versão básica da Essay-BR também foi utilizada para a criação de outras de redes, cujos resultados são usados para comparações gerais na seção 4.2.

A versão do BERTimbau escolhida como base para a implementação das redes especialistas foi a `bert-base-portuguese-cased` do HuggingFace, que possui o mesmo número de parâmetros do $\text{BERT}_{\text{BASE}}$, abordado na seção 2.6.1, e é sensitiva a letras maíusculas e minúsculas nos textos de entrada.

Em geral, foram realizados experimentos que levaram em conta hiperparâmetros otimizados e fixos. A seguir, detalharemos os resultados obtidos no processo de *hypertuning*, seguido da análise comparativa da função de perda para ambos os casos de treino.

4.1.1 Otimização de Hiperparâmetros

Na otimização dos hiperparâmetros, todas as combinações possíveis de valores foram testadas, em um total de 48 *trials* ($2 \cdot 2 \cdot 2 \cdot 2 \cdot 3$). A melhor configuração das redes é representada pelo conjunto de variáveis que fazem a função de perda ter o menor valor possível entre as iterações.

Nas seções 4.1.1.1, 4.1.1.2, 4.1.1.3, 4.1.1.4 e 4.1.1.5 exploraremos melhor o processo de otimização para os sistemas avaliadores de cada competência.

4.1.1.1 Competência I

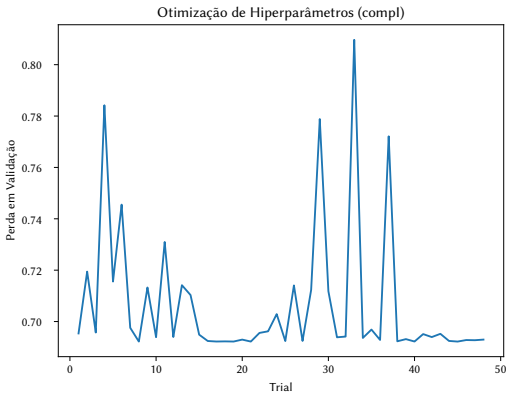

Figura 4.1: *Evolução da função de perda no hypertuning do modelo da competência I.*

A figura 4.1 mostra a evolução da função de perda ao longo das iterações da busca em grade para o modelo da competência I. A melhor configuração de hiperparâmetros obtida correspondeu a uma taxa de aprendizado de $2 \cdot 10^{-3}$, a um tamanho de lote de 4 e a funções de ativação de SeLU, Sigmoide e Sigmoide, respectivamente, nas três camadas ocultas. A perda de validação para esse conjunto foi de aproximadamente 0,692203.

4.1.1.2 Competência II

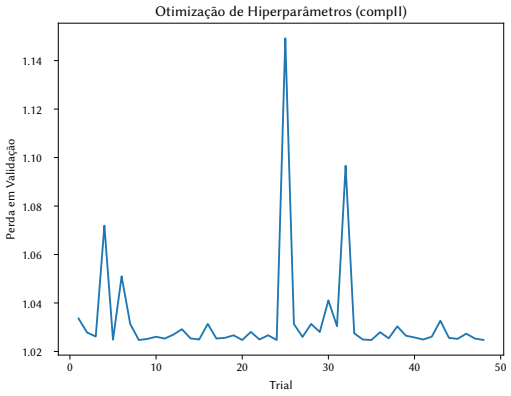

Figura 4.2: *Evolução da função de perda no hypertuning do modelo da competência II.*

A figura 4.2 evidencia a evolução da função de perda para a competência II. A melhor configuração de hiperparâmetros obtida correspondeu a uma taxa de aprendizado de $2 \cdot 10^{-3}$, a um tamanho de lote de 3 e a funções de ativação de SeLU, SeLU e Sigmoide, respectivamente. A melhor perda de validação obtida foi de aproximadamente 1,024713.

4.1.1.3 Competência III

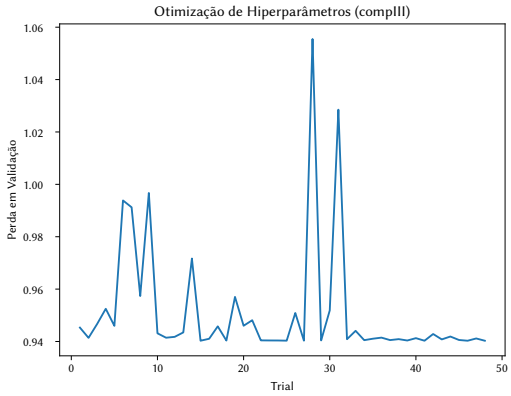

Figura 4.3: *Evolução da função de perda no hypertuning do modelo da competência III.*

A figura 4.3 demonstra a evolução da função de perda para a competência III. A melhor configuração de hiperparâmetros obtida correspondeu a uma taxa de aprendizado de $2 \cdot 10^{-5}$, a um tamanho de lote de 4 e a funções de ativação de Sigmoide, Sigmoide e Sigmoide, respectivamente. A melhor perda de validação registrada foi de aproximadamente 0,940287.

4.1.1.4 Competência IV

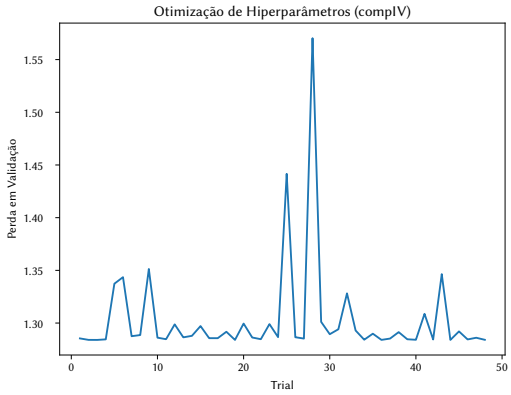

Figura 4.4: *Evolução da função de perda no hypertuning do modelo da competência IV.*

A figura 4.4 ilustra a evolução da função de perda para a competência IV. A melhor configuração de hiperparâmetros obtida correspondeu a uma taxa de aprendizado de

$2 \cdot 10^{-3}$, a um tamanho de lote de 2 e a funções de ativação de SeLU, Sigmoide e Sigmoide, respectivamente. A melhor perda de validação obtida foi de aproximadamente 1,283995.

4.1.1.5 Competência V

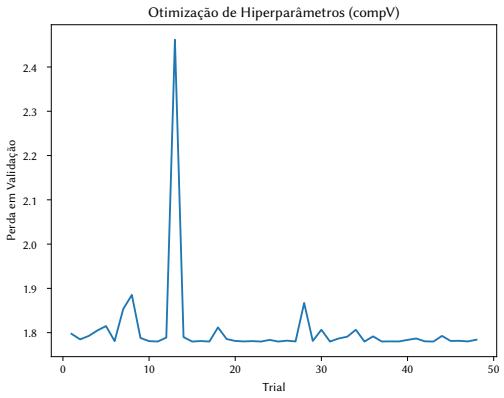

Figura 4.5: *Evolução da função de perda no hypertuning do modelo da competência V.*

A figura 4.5 mostra a evolução da função de perda para a competência V. A melhor configuração de hiperparâmetros obtida correspondeu a uma taxa de aprendizado de $2 \cdot 10^{-3}$, a um tamanho de lote de 2 e a funções de ativação de Sigmoide, Sigmoide e Sigmoide, respectivamente. A melhor perda de validação obtida foi de aproximadamente 1,779886.

4.1.2 Análise Comparativa

Além dos experimentos de otimização, realizamos experimentos de controle com hiperparâmetros fixados e conhecidos. Essa abordagem possibilita que se avalie a influência do processo de *hypertuning* no desenvolvimento de redes especialistas de correção. O treinamento de controle foi feito nas duas bases de dados, mas a análise comparativa concentra-se nos resultados da base estendida.

Os mesmos hiperparâmetros foram escolhidos para os cinco modelos de avaliação: taxa de aprendizado de $2{\cdot}10^{-5}$, tamanho de lote de 4 e funções de ativação SeLU, Sigmoide e SeLU, respectivamente. Essa configuração está dentro do espaço de possibilidades (Subseção 3.3.3) e foi selecionada experimentalmente no início do projeto por apresentar um desempenho razoável.

Os modelos de controle também foram treinados em 50 épocas, de modo que apenas a rede com melhor função de perda na base de validação foi salva. Isso é feito por meio do *callback* `ModelCheckpoint`, apresentado anteriormente (Subseção 3.4.1). Nas seções 4.1.2.1, 4.1.2.2, 4.1.2.3, 4.1.2.4 e 4.1.2.5 confrontaremos, visualmente, a evolução dos treinamentos com *hypertuning* e com hiperparâmetros fixados.

4.1.2.1 Competência I

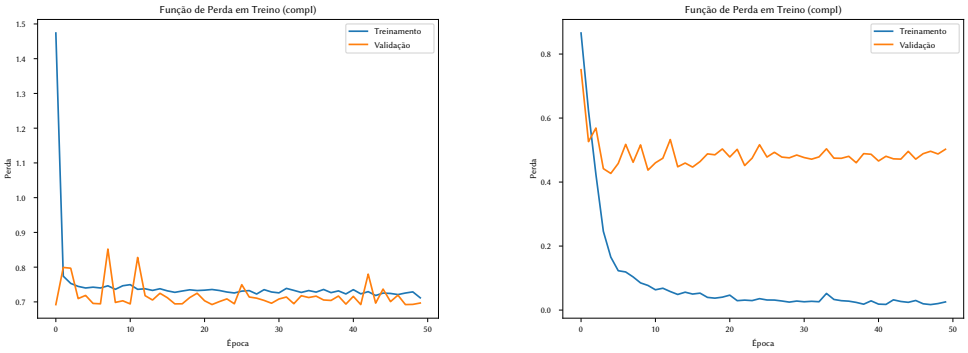

Figura 4.6: *Evolução do treinamento com hypertuning (à esquerda) e do treinamento com hiperparâmetros fixados (à direita) do sistema especialista da competência I.*

Pela figura 4.6, é possível notar que o treinamento com hiperparâmetros fixados apresentou uma evolução mais suave da função de perda, de modo que, em torno da 20ª época, o valor para a base de validação estagnou em cerca de 0,5 e o valor para a base de treino oscilou entre 0,05. No caso do *hypertuning*, a função de perda registrou números mais altos tanto para a base de treino quanto de validação, convergindo para um valor de aproximadamente 0,7 já nas primeiras épocas. Constata-se, assim, que o processo de otimização não surtiu efeitos para a competência I.

4.1.2.2 Competência II

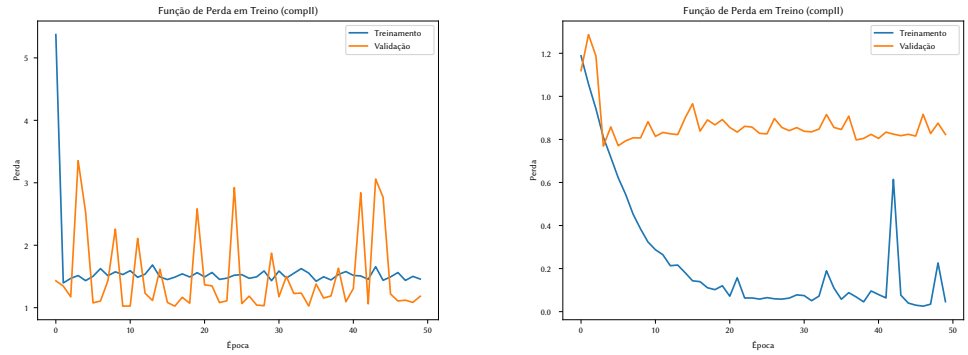

Figura 4.7: *Evolução do treinamento com hypertuning (à esquerda) e do treinamento com hiperparâmetros fixados (à direita) do sistema especialista da competência II.*

Pela figura 4.7, é possível notar que o treinamento com hiperparâmetros fixados também apresentou uma evolução mais suave da função de perda, de modo que, em torno da 10ª época, o valor para a base de validação estagnou em cerca de 0,8. Além disso, a partir da 20ª época, a perda para a base de treino convergiu para aproximadamente 0,08, com alguns picos próximos das 40 iterações. No caso do *hypertuning*, a função também registrou números mais altos para a base de treino e de validação. Na primeira situação, o valor

estagnou em 1,5 ainda no início, enquanto que, na segunda, a perda oscilou entre 1 e 3 ao longo de todo o processo. Nota-se, assim, que o processo de otimização não foi efetivo para a competência II.

4.1.2.3 Competência III

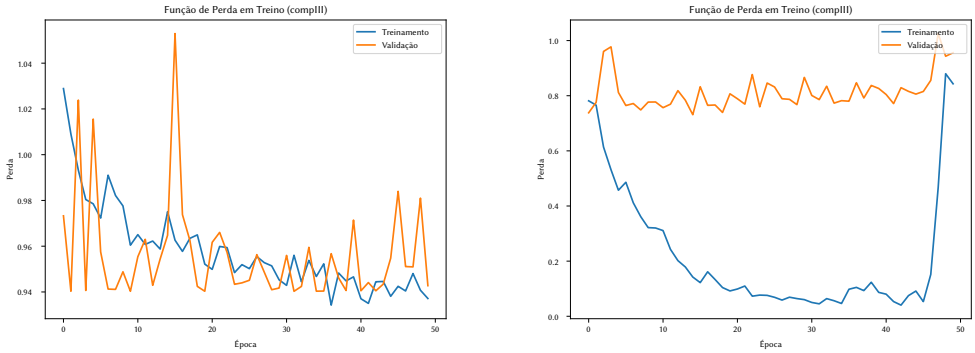

Figura 4.8: *Evolução do treinamento com hypertuning (à esquerda) e do treinamento com hiperparâmetros fixados (à direita) do sistema especialista da competência III.*

Pela figura 4.8, é possível notar que o treinamento com hiperparâmetros fixados apresentou uma evolução levemente melhor da função de perda, de modo que, em torno das primeiras épocas, o valor para a base de validação oscilou entre 0,8 e, entre a 30ª e a 45ª época, o valor para a base de treino ficou em torno de 0,1. No final das iterações, houve, ainda, um aumento significativo da perda tanto para a base de treino quanto para a base de validação, que é irrelevante devido ao *callback* `ModelCheckpoint`. No caso do *hypertuning*, a função aplicada aos dados de treino apresentou um decaimento ao longo das épocas, mas a perda para os dados de validação teve uma oscilação entre 1,04 e 0,94 entre todo o processo. Desse modo, observa-se que os parâmetros fixados apresentam uma leve vantagem em relação ao processo de otimização para a competência III.

4.1.2.4 Competência IV

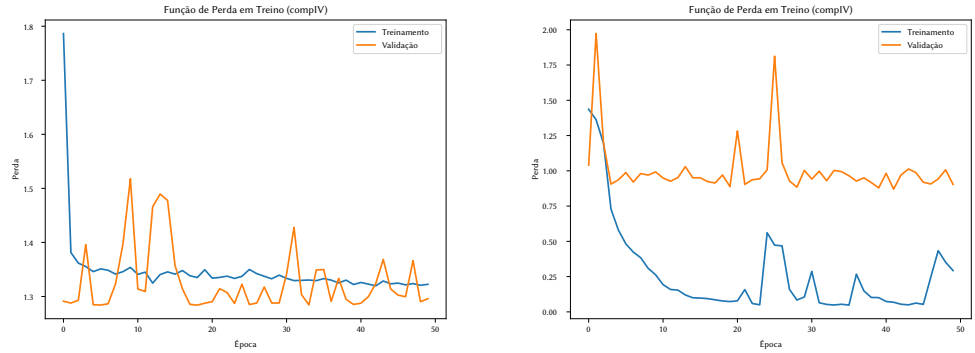

Figura 4.9: *Evolução do treinamento com hypertuning (à esquerda) e do treinamento com hiperparâmetros fixados (à direita) do sistema especialista da competência IV.*

Pela figura 4.9, é possível notar que o treinamento com hiperparâmetros fixados apresentou uma evolução levemente melhor da função de perda, de modo que, já no início, o valor para a base de validação oscilou em torno de 0,9, com alguns picos no meio do processo. Além disso, a partir da 15ª época, a perda para a base de treino convergiu para aproximadamente 0,1, apresentando 4 aumentos substanciais até o final. No caso do *hypertuning*, a função seguiu registrando números mais altos para a base de treino e de validação. Na primeira circunstância, o valor estagnou em 1,35 ainda nas primeiras iterações, enquanto que, na segunda, a perda oscilou entre 1,3 e 1,5 ao longo do início e do fim do processo. Assim, infere-se que os parâmetros fixados possuem vantagens em relação ao *hypertuning* para a competência IV.

4.1.2.5 Competência V

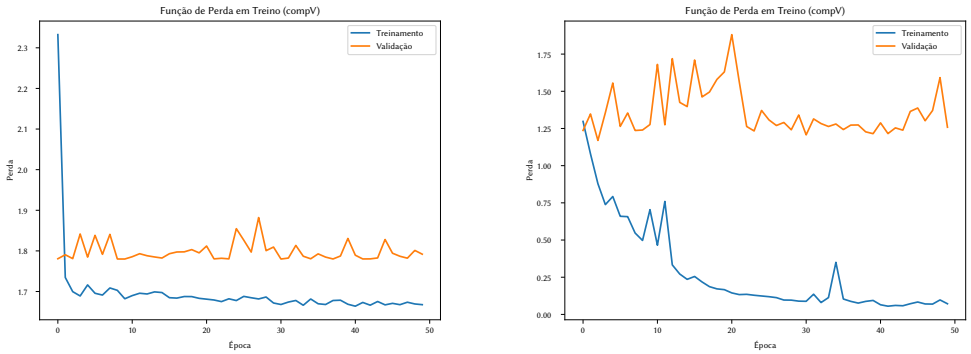

Figura 4.10: *Evolução do treinamento com hypertuning (à esquerda) e do treinamento com hiperparâmetros fixados (à direita) do sistema especialista da competência V.*

Pela figura 4.10, é possível notar que o treinamento com hiperparâmetros fixados apresentou uma evolução mais suave da função de perda, de modo que, a partir da 25ª época, o valor para a base de validação oscilou em torno de 1,3, com alguns pequenos picos no fim do processo, e, a partir da 20ª época, a perda para a base de treino convergiu para aproximadamente 0,1, apresentando apenas um aumento substancial posteriormente. No caso do *hypertuning*, a função registrou números mais altos para a base de treino e de validação. Na primeira circunstância, o valor oscilou por volta de 1,8 em todas as iterações, enquanto que, na segunda, a perda desceu muito rápido para 1,7 nas primeiras épocas e assim permaneceu até o fim. Assim, nota-se que os parâmetros fixados possuem vantagens em relação ao *hypertuning* para a competência V.

4.2 Avaliação

Após o treinamento das redes especialistas, experimentos de avaliação dos modelos criados foram conduzidos visando comparar o desempenho dos sistemas de correção automática. Para isso, utilizamos as bases de dados estendida e simples da Essay-BR, além das arquiteturas com hiperparâmetros otimizados e fixos, de modo que os resultados obtidos pudessem ser comparados.

Analisaremos o desempenho dos modelos utilizando as métricas apresentadas na seção 3.3.4 — *Quadratic Weighted Kappa* (**QWK**), Divergência (**DIV**), Proporção de Correspondência Exata (**PCE**) e *Mean Squared Error* (**MSE**). Quanto maior o QWK e o PCE, melhor a qualidade da rede. De modo análogo, quanto menor a DIV e o MSE, melhor o desempenho do modelo.

Compararemos as versões desenvolvidas na tabela 4.1, em que **RTBE** e **RTBS** representam as redes com *hypertuning* nas bases estendida e simples, respectivamente, e **RFBE** e **RFBS** representam as redes com hiperparâmetros fixos nas bases estendida e simples, também nessa ordem.

	Comp. I				Comp. II				Comp. III				Comp. IV				Comp. V			
	DIV	**MSE**	**PCE**	**QWK**	**DIV**	**MSE**	**PCE**	**QWK**	**DIV**	**MSE**	**PCE**	**QWK**	**DIV**	**MSE**	**PCE**	**QWK**	**DIV**	**MSE**	**PCE**	**QWK**
RTBE	0,012	0,653	0,468	0,0	0,017	0,978	0,367	0,0	0,029	0,903	0,465	0,0	0,032	1,294	0,381	0,0	0,079	1,729	0,338	0,0
RTBS	0,018	0,651	0,561	0,0	0,021	0,998	0,349	0,0	0,023	0,846	0,467	0,0	0,027	1,125	0,427	0,0	0,061	1,580	0,381	0,0
RFBE	**0,008**	**0,358**	0,683	**0,595**	0,017	0,702	0,554	0,468	0,013	0,623	0,546	0,501	0,022	0,805	**0,513**	0,576	0,036	1,156	0,421	0,483
RFBS	0,013	0,417	**0,720**	0,542	**0,016**	**0,640**	**0,572**	**0,562**	**0,010**	**0,554**	**0,593**	**0,539**	**0,010**	**0,701**	0,504	**0,621**	**0,030**	**0,932**	**0,494**	**0,548**

Tabela 4.1: *Métricas de avaliação dos modelos de correção automática por competência.*

Nas seções 4.2.1, 4.2.2, 4.2.3, 4.2.4 e 4.2.5 avaliaremos os modelos construídos para cada competência, com base nas métricas apresentadas na tabela 4.1, e datalharemos o processo de inferência das notas por parte das melhores redes especialistas.

4.2.1 Competência I

A melhor rede especialista para a competência I foi a **RFBE**, que obteve um QWK de 0,595, uma divergência de 0,008, um MSE de 0,358 e uma PCE de 0,683. A única métrica em que esse modelo ficou atrás foi a PCE, que foi maior para a **RFBS**. A figura 4.11 mostra a matriz de confusão para a rede.

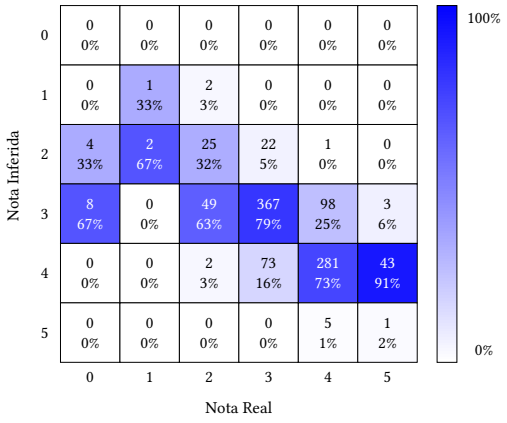

Figura 4.11: *Matriz de confusão do modelo RFBE da competência I.*

Em geral, a partir da figura 4.11, é possível observar que a rede especialista apresentou um desempenho razoável, de modo que as notas inferidas, na maioria das vezes, estão próximas ou são iguais às notas reais. Nota-se, entretanto, uma dificuldade em atribuir notas extremas: o modelo geralmente erra ao inferir pontuações 0, 1 e 5.

4.2.2 Competência II

A melhor rede especialista para a competência II foi a **RFBS**, que obteve um QWK de 0,562, uma divergência de 0,016, um MSE de 0,640 e uma PCE de 0,572. A figura 4.12 mostra a matriz de confusão para a rede.

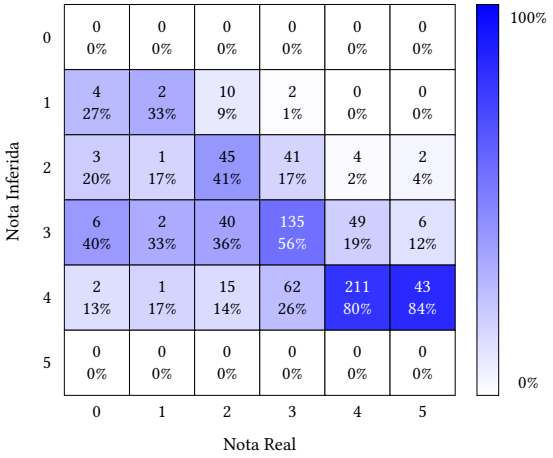

Figura 4.12: *Matriz de confusão do modelo RFBS da competência II.*

Em geral, a partir da figura 4.12, é possível observar que a rede especialista apresentou um desempenho razoável, de modo que notas inferidas, também na maior parte das vezes, estão próximas ou são iguais às notas reais. Para pontuações 0, 1 e 2, no entanto, notamos que a distribuição das notas inferidas tem um comportamente análogo ao aleatório. O melhor modelo da competência II também tem dificuldades de generalização, já que não atribuiu nenhuma nota 0 ou 5 para a divisão de testes.

4.2.3 Competência III

A melhor rede especialista para a competência III foi a **RFBS**, que obteve um QWK de 0,539, uma divergência de 0,010, um MSE de 0,554 e uma PCE de 0,593. A figura 4.13 mostra a matriz de confusão para a rede.

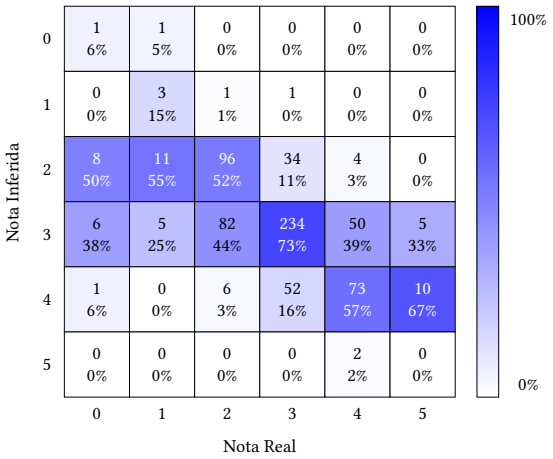

Figura 4.13: *Matriz de confusão do modelo RFBS da competência III.*

Em geral, a partir da figura 4.13, é possível observar que a rede especialista apresentou um desempenho razoável, de modo que as notas inferidas, também na maioria das vezes, estão próximas ou são iguais às notas reais. O melhor modelo da competência III teve dificuldades em lidar com notas extremas, já que errou em 100% dos casos que atribuiu nota 5 para uma competência e acertou apenas 6% das notas 0.

4.2.4 Competência IV

A melhor rede especialista para a competência IV foi a **RFBS**, que obteve um QWK de 0,621, uma divergência de 0,010, um MSE de 0,701 e uma PCE de 0,504. O modelo só registrou subdesempenho na métrica de PCE, em relação à versão **RFBE**. A figura 4.14 mostra a matriz de confusão para a rede.

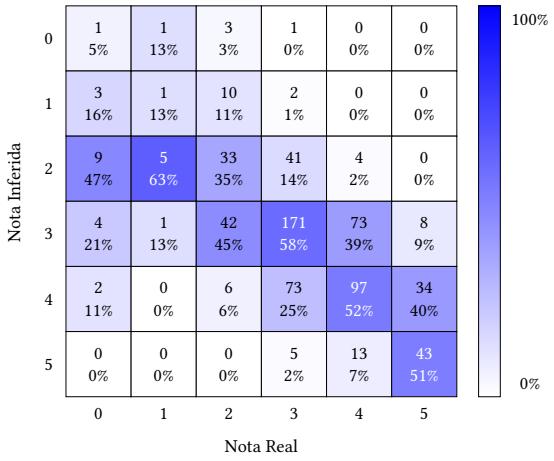

Figura 4.14: *Matriz de confusão do modelo RFBS da competência IV.*

Em geral, a partir da figura 4.14, é possível observar que a rede especialista apresentou um desempenho razoável, de modo que as notas inferidas, em boa parte das vezes, estão próximas ou são iguais às notas reais. O melhor modelo para a competência IV, no entanto, tem dificuldades em reconhecer notas baixas como 0 e 1, atribuindo-as inadequadamente em relação às pontuações reais.

4.2.5 Competência V

Por fim, a melhor rede especialista para a competência V foi a **RFBS**, que obteve um QWK de 0,548, uma divergência de 0,030, um MSE de 0,932 e uma PCE de 0,494. A figura 4.15 mostra a matriz de confusão para a rede.

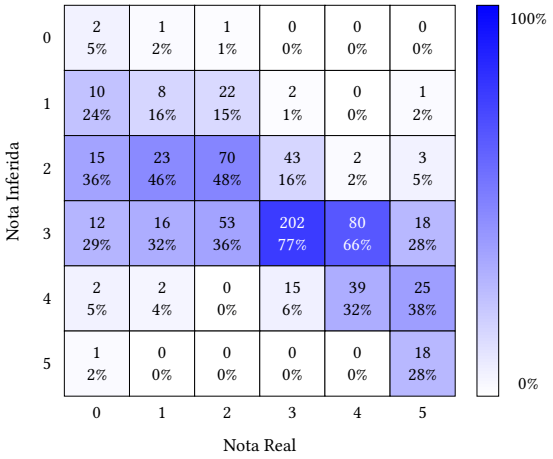

Figura 4.15: *Matriz de confusão do modelo RFBS da competência V.*

Em geral, a partir da figura 4.15, é possível observar que a rede especialista apresentou um desempenho razoável, de modo que as notas inferidas, para pontuações reais acima de 2, também estão próximas ou são iguais às notas reais. Nota-se que o melhor modelo para a competência V, no entanto, tem dificuldades em reconhecer notas baixas como 0 e 1, atribuindo quantidades significativas de pontuações 2 e 3.

Capítulo 5

Conclusões

Neste trabalho, foram explorados o desenvolvimento e a avaliação de modelos de correção automática de redações em língua portuguesa, com foco no contexto do ENEM no Brasil. Para isso, foi utilizada a base de dados Essay-BR, que contém diversos textos avaliados por profissionais, na construção de sistemas especializados em atribuir notas a cada competência exigida pelo exame.

A abordagem adotada utilizou o BERT, uma arquitetura baseada em *transformers*, para extrair informações relevantes dos textos dissertativo-argumentativos produzidos pelos alunos, com o propósito de treinar modelos generalistas o suficiente para pontuar novas produções.

A avaliação dos sistemas produzidos também foi realizada com base no conjunto de dados da Essay-BR, considerando instâncias isoladas de teste. Nesse caso, diversas técnicas de treino, como o *hypertuning*, foram empregadas, com o intuito de avaliar a eficácia real da construção desses modelos e responder se, de fato, são viáveis de aplicação frente à correção humana.

Durante o trabalho, um dos desafios enfrentados foi o antagonismo entre o tempo de treino e a convergência das redes no processo de *hypertuning*. Devido ao grande espaço de exploração de hiperparâmetros, tornou-se necessário diminuir o número de épocas no treinamento de cada configuração, o que levou a escolhas genéricas de valores que não necessariamente refletiam o cenário de convergência do processo, mas eram mais sensíveis às inicializações aleatórias das redes. A forma plausível de se solucionar tal problema seria pelo aumento do número de épocas, que geraria, no entanto, períodos de treino mais longos.

Os resultados obtidos, contemplados no capítulo 4, corroboraram essas dificuldades. Eles mostram que a técnica de *hypertuning* não produz efeitos satisfatórios em relação ao aprendizado dos sistemas, já que, a partir dela, são escolhidos como melhores hiperparâmetros os valores que só otimizam as redes na média, mas não as generalizam. Além disso, dadas as frequentes oscilações da função de perda ao longo dos treinos com as melhores configurações, evidenciadas pelos gráficos 4.1, 4.2, 4.3, 4.4 e 4.5, nota-se que a quantidade de dados pode não ser suficiente para que essas redes aprendam adequadamente a tarefa de avaliação.

Por outro lado, a técnica de *fine-tuning* com hiperparâmetros fixados, que foi utilizada para treinar os melhores modelos obtidos, mostrou-se mais eficaz, já que é possível realizar experimentos que contemplem um número maior de épocas utilizando-se de configurações com maiores chances de convergência. Apesar de ser igualmente difícil encontrar valores razoáveis iniciais, a formulação de possíveis heurísticas no método manual pode tornar o processo de escolha de novos hiperparâmetros mais rápido.

Outra constatação importante foi a da eficácia da Essay-BR básica em relação a sua versão estendida. Apesar de conter mais dados, notou-se, pela tabela 4.1, que a base com mais instâncias apresentou subdesempenho em relação às métricas do conjunto simplificado. Isso pode ser consequência da introdução das novas redações de uma mesma fonte à Essay-BR, que podem não ter agregado valores de generalização suficientes ao treinamento.

De modo geral, os resultados obtidos com os modelos treinados foram satisfatórios, já que os melhores sistemas foram capazes de pontuar as redações com notas próximas às atribuídas pelos avaliadores humanos. No entanto, a análise dos erros cometidos pelos modelos mostrou que ainda há muito espaço para melhorias, principalmente no que diz respeito à pontuação das competências II e V, que avaliam a adequação ao tema e a capacidade de elaborar uma proposta de intervenção para o problema, respectivamente.

Deduz-se que um dos impasses da avaliação dessas duas competências é a correlação intrínseca que ambas possuem com o tema das redações, informação não contemplada nos textos submetidos a treino. Uma das possíveis soluções seria adicionar esse dado no início da entrada de cada rede especialista, utilizando o *token* especial `[SEP]` para realizar a divisão entre o tema e a redação.

Como abordagens futuras, sugere-se a construção de um mecanismo de explicabilidade dos modelos, que permita a análise de quais características dos textos são mais relevantes para a atribuição de notas. Além disso, a utilização de técnicas de *data augmentation* pode ser uma alternativa para a melhoria dos resultados, já que a base de dados utilizada é relativamente pequena. Por fim, utilizar novos modelos de linguagem pode ser uma escolha viável para obter resultados melhores, permitindo uma comparação que não se limite ao nível da arquitetura das redes especialistas, mas também explore suas estruturas subjacentes.

Referências

[Amorim e Bazelato 2013] Evelin Amorim e Bruno Bazelato. "A bayesian classifier to automatic correction of portuguese essays". In: *Nuevas Ideas en Informática Educativa*. Porto Alegre, RS, Brasil, 2013, pp. 779–782 (citado na pg. 9).

[Amorim e Veloso 2017] Evelin Amorim e Adriano Veloso. "A multi-aspect analysis of automatic essay scoring for Brazilian Portuguese". In: *Proceedings of the Student Research Workshop at the 15th Conference of the European Chapter of the Association for Computational Linguistics*. Valencia, Spain: Association for Computational Linguistics, abr. de 2017, pp. 94–102. URL: https://aclanthology.org/E17-4010 (citado nas pgs. 2, 9, 33).

[Attali e Burstein 2006] Yigal Attali e Jill Burstein. "Automated essay scoring with e-rater® v.2". *The Journal of Technology, Learning and Assessment* 4.3 (fev. de 2006). URL: https://ejournals.bc.edu/index.php/jtla/article/view/1650 (citado na pg. 9).

[Barrera e Shermis 2002] Felicia D. Barrera e Mark D. Shermis. "Exit assessments: evaluating writing ability through automated essay scoring". In: Annual Meeting of the American Educational Research Association. New Orleans, LA, 2002 (citado na pg. 9).

[Bengio *et al.* 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent e Christian Janvin. "A neural probabilistic language model". *J. Mach. Learn. Res.* 3 (mar. de 2003), pp. 1137–1155. ISSN: 1532-4435 (citado nas pgs. 11, 12, 14, 15).

[Bhattacharjee 2018] J. Bhattacharjee. *FastText Quick Start Guide: Get Started with Facebook's Library for Text Representation and Classification*. Packt Publishing, 2018. ISBN: 9781789130997 (citado na pg. 10).

[Bojanowski *et al.* 2016] Piotr Bojanowski, Edouard Grave, Armand Joulin e Tomás Mikolov. "Enriching word vectors with subword information". *CoRR* abs/1607.04606 (2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606 (citado na pg. 13).

[Brasil 2019a] Brasil. *ENEM Redações 2019: Material de Leitura (Competência III)*. Brasília: Insituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira, 2019. URL: https://download.inep.gov.br/educacao_basica/enem/downloads/2020/Competencia_3.pdf (acesso em 01/12/2023) (citado nas pgs. 1, 3).

[Brasil 2019b] Brasil. *ENEM Redações 2019: Material de Leitura (Competência IV)*. Brasília: Insituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira, 2019. url: https://download.inep.gov.br/educacao_basica/enem/downloads/2020/Competencia_4.pdf (acesso em 01/12/2023) (citado na pg. 3).

[Brasil 2023] Brasil. *A Redação do Enem 2023: cartilha do participante*. Brasília: Insituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira, 2023. url: https://download.inep.gov.br/publicacoes/institucionais/avaliacoes_e_exames_da_educacao_basica/a_redacao_no_enem_2023_cartilha_do_participante.pdf (acesso em 20/10/2023) (citado nas pgs. 2, 3, 5–7, 41).

[Bridle 1989] John Bridle. "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters". In: *Advances in Neural Information Processing Systems*. Ed. por D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. url: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf (citado na pg. 14).

[Cho *et al.* 2014] Kyunghyun Cho *et al.* "Learning phrase representations using rnn encoder-decoder for statistical machine translation" (2014). doi: 10.48550/arXiv.1406.1078. arXiv: 1406.1078 [cs.CL] (citado na pg. 19).

[Cohen 1968] Jacob Cohen. "Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit." *Psychological Bulletin* 70 (1968), pp. 213–220. url: https://api.semanticscholar.org/CorpusID:29694079 (citado na pg. 41).

[Costa *et al.* 2020] Luciana Costa, Elaine Oliveira e Alberto Castro Júnior. "Corretor automático de redações em língua portuguesa: um mapeamento sistemático de literatura". In: *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. Online: SBC, 2020, pp. 1403–1412. doi: 10.5753/cbie.sbie.2020.1403. url: https://sol.sbc.org.br/index.php/sbie/article/view/12896 (citado na pg. 8).

[Devlin *et al.* 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee e Kristina Toutanova. "BERT: pre-training of deep bidirectional transformers for language understanding". *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. url: http://arxiv.org/abs/1810.04805 (citado nas pgs. 1, 26–29).

[Fiorin e Savioli 2006] José Luiz Fiorin e Francisco Platão Savioli. *Lições de texto: leitura e redação*. Ática, 2006 (citado na pg. 5).

[Fonseca *et al.* 2018] Erick Fonseca, Ivo Medeiros, Dayse Kamikawachi e Alessandro Bokan. "Automatically grading brazilian student essays: 13th international conference, propor 2018, canela, brazil, september 24–26, 2018, proceedings". In: jan. de 2018, pp. 170–179. isbn: 978-3-319-99721-6. doi: 10.1007/978-3-319-99722-3_18 (citado nas pgs. 2, 9).

[Harrington 2012] Peter Harrington. *Machine Learning in Action*. USA: Manning Publications Co., 2012. isbn: 1617290181 (citado na pg. 15).

[Hochreiter e Schmidhuber 1997] Sepp Hochreiter e Jürgen Schmidhuber. "Long Short-Term Memory". *Neural computation* 9 (dez. de 1997), pp. 1735–80. doi: 10.1162/neco.1997.9.8.1735 (citado na pg. 18).

[Joulin *et al.* 2016] Armand Joulin, Edouard Grave, Piotr Bojanowski e Tomás Mikolov. "Bag of tricks for efficient text classification". *CoRR* abs/1607.01759 (2016). arXiv: 1607.01759. url: http://arxiv.org/abs/1607.01759 (citado na pg. 14).

[Júnior *et al.* 2017] Celso Júnior, Marcos Spalenza e Elias de Oliveira. "Proposta de um sistema de avaliação automática de redações do enem utilizando técnicas de aprendizagem de máquina e processamento de linguagem natural". In: *Anais do Computer on the Beach 2017*. Florianópolis, SC, Brasil, 2017 (citado na pg. 9).

[Kingma e Ba 2017] Diederik P. Kingma e Jimmy Ba. "Adam: a method for stochastic optimization" (2017). arXiv: 1412.6980 [cs.LG] (citado na pg. 43).

[Kudo e Richardson 2018] Taku Kudo e John Richardson. "Sentencepiece: a simple and language independent subword tokenizer and detokenizer for neural text processing" (2018). arXiv: 1808.06226 [cs.CL] (citado na pg. 29).

[Larkey 1998] Leah S. Larkey. "Automatic essay grading using text categorization techniques". In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '98. Melbourne, Australia: Association for Computing Machinery, 1998, pp. 90–95. isbn: 1581130155. doi: 10.1145/290941.290965. url: https://doi.org/10.1145/290941.290965 (citado na pg. 1).

[Lesme 2021] Adriano Lesme. "Enem 2021: corretores podem corrigir até 200 redações por dia". *Brasil Escola* (1 de dez. de 2021). url: https://vestibular.brasilescola.uol.com.br/enem/enem-2021-corretores-podem-corrigir-ate-200-redacoes-por-dia/351641.html (acesso em 20/10/2023) (citado na pg. 4).

[Lim *et al.* 2021] Chun Lim, Chih How Bong, Wee Sian Wong e Nung Kion Lee. "A comprehensive review of automated essay scoring (aes) research and development". *Pertanika Journal of Science and Technology* 29 (jul. de 2021). doi: 10.47836/pjst.29.3.27 (citado na pg. 1).

[Manai *et al.* 2023] Elyes Manai, Mohamed Mejri e Jaouhar Fattahi. "Impact of feature encoding on malware classification explainability" (2023). arXiv: 2307.05614 [cs.LG] (citado na pg. 11).

[Manning e Schütze 1999] Christopher D. Manning e Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: The MIT Press, 1999 (citado nas pgs. 10, 11).

[Marinho *et al.* 2021] Jeziel Marinho, Rafael Anchiêta e Raimundo Moura. "Essay-br: a brazilian corpus of essays" (2021), pp. 53–64. DOI: 10.5753/dsw.2021.17414. URL: https://sol.sbc.org.br/index.php/dsw/article/view/17414 (citado nas pgs. 2, 31, 33, 40).

[Marinho *et al.* 2022] Jeziel Marinho, Rafael Anchiêta e Raimundo Moura. "Essay-br: a brazilian corpus to automatic essay scoring task". *Journal of Information and Data Management* 13.1 (2022), pp. 65–76. DOI: 10.5753/jidm.2022.2340. URL: https://sol.sbc.org.br/journals/index.php/jidm/article/view/2340 (citado nas pgs. 3, 31, 34, 40).

[Mikolov *et al.* 2013] Tomas Mikolov, Wen-tau Yih e Geoffrey Zweig. "Linguistic regularities in continuous space word representations". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, jun. de 2013, pp. 746–751. URL: https://aclanthology.org/N13-1090 (citado nas pgs. 12, 13).

[Myers 2003] Miles Myers. "What can computers and aes contribute to a k-12 writing program?" In: *Automated essay scoring: A cross-disciplinary perspective*. New York, NY: Lawrence Erlbaum Associates, 2003. ISBN: 9781410606860. DOI: 10.4324/9781410606860 (citado na pg. 8).

[Oliveira 2016] Andrea Oliveira. "O enem como processo seletivo para o ensino superior: algumas considerações sobre a democratização do acesso e sobre o construto do exame". *Jornal de Políticas Educacionais* 9 (mai. de 2016), pp. 156–167. DOI: 10.5380/jpe.v9i17/18.40721 (citado nas pgs. 2, 3).

[Page 1966] Ellis B. Page. "The imminence of... grading essays by computer". In: *The Phi Delta Kappan*. Vol. 47. 5. 1966, pp. 238–243 (citado na pg. 9).

[Pennington *et al.* 2014] Jeffrey Pennington, Richard Socher e Christopher Manning. "GloVe: global vectors for word representation"". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, out. de 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (citado na pg. 13).

[Persing *et al.* 2010] Isaac Persing, Alan Davis e Vincent Ng. "Modeling organization in student essays". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, out. de 2010, pp. 229–239. URL: https://aclanthology.org/D10-1023 (citado na pg. 1).

[Rudner *et al.* 2006] Lawrence M. Rudner, Veronica Garcia e Catherine Welch. "An evaluation of intellimetric™ essay scoring system". *The Journal of Technology, Learning and Assessment* 4.4 (mar. de 2006). URL: https://ejournals.bc.edu/index.php/jtla/article/view/1651 (citado na pg. 1).

[Rumelhart *et al.* 1986] David E. Rumelhart, Geoffrey E. Hinton e Ronald J. Williams. "Learning representations by back-propagating errors". *Nature* 323 (1986), pp. 533–536. doi: 10.1038/323533a0 (citado na pg. 16).

[Rupp *et al.* 2019] André A. Rupp, Jodi M. Casabianca, Maleika Krüger, Stefan Keller e Olaf Köller. "Automated essay scoring at scale: a case study in switzerland and germany". *ETS Research Report Series* 2019.1 (2019), pp. 1–23. doi: https://doi.org/10.1002/ets2.12249. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ets2.12249. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/ets2.12249 (citado na pg. 2).

[Schmidhuber e Heil 1996] Jürgen Schmidhuber e Stefan Heil. "Sequential neural text compression". *IEEE Transactions on Neural Networks* 7.1 (1996), pp. 142–146. doi: 10.1109/72.478398 (citado na pg. 12).

[Schmidt 2019] Robin M. Schmidt. "Recurrent neural networks (rnns): a gentle introduction and overview" (2019). arXiv: 1912.05911 [cs.LG] (citado nas pgs. 17, 18).

[Silva 2013] Willian Silva. *Aprimorando o corretor gramatical CoGrOO*. São Paulo, SP, Brasil, 2013. doi: 10.11606/D.45.2013.tde-02052013-135414 (citado na pg. 9).

[Somasundaran *et al.* 2014] Swapna Somasundaran, Jill Burstein e Martin Chodorow. "Lexical chaining for measuring discourse coherence quality in test-taker essays". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University e Association for Computational Linguistics, ago. de 2014, pp. 950–961. url: https://aclanthology.org/C14-1090 (citado na pg. 1).

[Souza *et al.* 2020] Fábio Souza, Rodrigo Nogueira e Roberto Lotufo. "Bertimbau: pretrained bert models for brazilian portuguese". In: *Intelligent Systems: 9th Brazilian Conference, BRACIS 2020, Rio Grande, Brazil, October 20–23, 2020, Proceedings, Part I*. Rio Grande, Brazil: Springer-Verlag, 2020, pp. 403–417. isbn: 978-3-030-61376-1. doi: 10.1007/978-3-030-61377-8_28. url: https://doi.org/10.1007/978-3-030-61377-8_28 (citado nas pgs. 26, 29).

[Stein 2023] Allan Stein. "Veja quais eixos temáticos têm mais chance de cair na redação do enem". *Redação para o ENEM* (10 de jul. de 2023). url: https://redacaoparaoenem.com.br/veja-quais-eixos-tematicos-tem-mais-chance-de-cair-na-redacao-do-enem/ (acesso em 01/12/2023) (citado na pg. 3).

[J. Sun *et al.* 2022] Junlin Sun, Yanrong Liu, Jing Cui e Handong He. "Deep learning-based methods for natural hazard named entity recognition". *Scientific Reports* 12 (mar. de 2022), p. 4598. doi: 10.1038/s41598-022-08667-2 (citado na pg. 27).

[S. Sun e Iyyer 2021] Simeng Sun e Mohit Iyyer. "Revisiting simple neural probabilistic language models". *CoRR* abs/2104.03474 (2021). arXiv: 2104.03474. url: https://arxiv.org/abs/2104.03474 (citado na pg. 14).

[Taghipour e Ng 2016] Kaveh Taghipour e Hwee Tou Ng. "A neural approach to automated essay scoring" (nov. de 2016), pp. 1882–1891. DOI: 10.18653/v1/D16-1193. URL: https://aclanthology.org/D16-1193 (citado na pg. 4).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (citado nas pgs. 1, 21–25).

[Wagner Filho *et al.* 2018] Jorge A. Wagner Filho, Rodrigo Wilkens, Marco Idiart e 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: European Language Resources Association (ELRA), mai. de 2018. URL: https://aclanthology.org/L18-1686 (citado na pg. 29).

[Wu *et al.* 2016] Yonghui Wu *et al.* "Google's neural machine translation system: bridging the gap between human and machine translation" (2016). arXiv: 1609.08144 [cs.CL] (citado na pg. 28).

[Yang *et al.* 2020] Ruosong Yang, Jiannong Cao, Zhiyuan Wen, Youzheng Wu e Xiaodong He. "Enhancing automated essay scoring performance via fine-tuning pre-trained language models with combination of regression and ranking". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, nov. de 2020, pp. 1560–1569. DOI: 10.18653/v1/2020.findings-emnlp.141. URL: https://aclanthology.org/2020.findings-emnlp.141 (citado na pg. 2).

[Zhang *et al.* 2023] Aston Zhang, Zachary C. Lipton, Mu Li e Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. URL: https://d2l.ai (citado nas pgs. 1, 16, 17, 19, 20, 22, 24, 28).

[Zhu *et al.* 2015] Yukun Zhu *et al.* "Aligning books and movies: towards story-like visual explanations by watching movies and reading books" (2015). arXiv: 1506.06724 [cs.CV] (citado na pg. 28).