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

Estratégias para a avaliação automática da competência III da Redação do ENEM utilizando BERTimbau

Luís Davi Oliveira de Almeida Campos

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

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

São Paulo
2023

Agradecimentos

À minha família, que mesmo fisicamente distante durante meu período em São Paulo, sempre esteve junto comigo e me apoiou de todas as formas possíveis para que eu chegasse até aqui.

Ao Igor, pela paciência, pelo apoio e por tudo que pude aprender com ele durante todo o trabalho.

Aos meus amigos e colegas que me motivaram, me apioaram e estiveram ao meu lado durante a graduação.

Resumo

Luís Davi Oliveira de Almeida Campos. **Estratégias para a avaliação automática da competência III da Redação do ENEM utilizando BERTimbau**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A tarefa de Avaliação Automática de Redações já é estudada há algum tempo na área de Processamento de Linguagem Natural. Nos últimos anos, abordagens clássicas da área vem sendo amplamente superadas por modelos como o BERT, porém seu uso ainda foi pouco explorado em trabalhos em língua portuguesa. Esse trabalho propõe o uso do BERTimbau para avaliação da competência III Redação do ENEM. A abordagem utilizada foi de refinamento do BERTimbau a partir de um dataset contendo 6577 redações avaliadas. Três estratégias foram aplicadas para tentar melhorar o desempenho do modelo. Na primeira, o título do tema foi concatenado à redação antes da etapa de tokenização. Na segunda, além do tema, foram também concatenadas com a redação cinco características extraídas e transformadas em texto. Na terceira, as características foram colocadas em um tensor, o qual foi concatenado à representação gerada pelo BERT antes da entrada na camada de classificação. Os melhores resultados foram alcançados pela primeira estratégia, com um QWK superior à 0,57. O acréscimo de informações sobre as características nas estratégias seguintes não contribuiu para a melhora desses resultados. É proposto, ao final, o desenvolvimento de novos trabalhos que estudem formas eficientes de incrementar ou melhorar os dados passados para o BERTimbau.

Palavras-chave: Avaliação Automática de Redações. Processamento de Linguagem Natural. Redes Neurais. Modelos de Linguagem. Modelos baseados em Transformadores. Classificação de Textos. ENEM. BERT. BERTimbau.

Abstract

Luís Davi Oliveira de Almeida Campos. **Strategies for the Automatic Evaluation of Competence III from the ENEM Essay using BERTimbau**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

The task of Automated Essay Scoring has been studied for some time in the Natural Language Processing field. In recent years, classical approaches in the field have been widely surpassed by models such as BERT, but its use is still underexplored in portuguese-language studies. This work proposes applying BERTimbau for evaluating the competence III of the ENEM essay. The approach employed the fine-tuning of BERTimbau with a dataset containing 6577 evaluated essays. Three strategies were applied to try to enhance the model's performance. In the first strategy, the prompt title was concatenated to the essay before the tokenization step. In the second one, in addition to the prompt title, 5 extracted features were transformed into text and also concatenated to the essay. In the third, the features were placed in a tensor, which was concatenated with the representation generated from BERT, before feeding the classification layer. The best results were achieved by the first strategy, with a QWK exceeding 0.57. Including information about the features in the subsequent strategies did not improve these results. Finally, we advocate for the development of new studies examining efficient approaches to generate better data to feed BERTimbau.

Keywords: Automated Essay Scoring. Natural Language Processing. Neural Networks. Language Models. Transformer based models. Text Classification. ENEM. BERT. BERTimbau.

Lista de abreviaturas

ENEM	Exame Nacional do Ensino Médio
AAR	Avaliação Automática de Redações
PLN	Processamento de Linguagem Natural
IA	Inteligência Artificial
BERT	*Bidirectional Encoder Representation from Transformers*
API	*Aplication Programming Interface*
GELU	*Gaussian Error Linear Unit*
ReLU	*Rectified Linear Unit*
GPU	*Graphic Processing Unit*
TPU	*Tensor Processing Unit*
QWK	*Quadratic Weighted Kappa*
RMSE	*Root-Mean-Square Error*

Lista de figuras

Lista de tabelas

Lista de programas

Sumário

Capítulo 1

Introdução

1.1 Contextualização

O Processamento de Linguagem Natural (PLN), tradicionalmente tratado como uma subárea da Inteligência Artificial, é um campo da computação tão antigo quanto os próprios computadores. Seu principal objetivo é fazer os computadores realizarem tarefas envolvendo a linguagem humana, tais como o processamento de textos e a comunicação por meio da fala.

Nos seus primórdios, os sistemas de PLN eram essencialmente simbólicos, baseados em gramáticas formais e regras codificadas manualmente. Um exemplo clássico é ELIZA, um chat-bot apresentado em 1966, que usava expressões regulares e um pequeno conjunto de regras para emular respostas de um psicoterapeuta em conversas com usuários (Weizenbaum, 1966).

Com o passar dos anos a área foi evoluindo e, mesmo tempo, foi aumentando a também a capacidade dos computadores de armazenar e processar quantidades cada vez maiores de dados. Esse cenário alavancou o desenvolvimento de soluções baseadas em modelos mais estatísticos e conexionistas, capazes de lidar com esses enormes conjuntos de dados, como os modelos de aprendizado de máquina e aprendizado profundo.

Nesse contexto, modelos como o BERT (Devlin *et al.*, 2019), baseados na arquitetura *Transformers* (Vaswani *et al.*, 2017), surgem como uma evolução dos primeiros modelos de redes neurais, a exemplo das Redes Neurais Recorrentes (RNNs) e das Redes Neurais Convolucionais (CNNs) e, atualmente, se apresentam como a principal solução para a realização de uma série de tarefas de PLN, devido a sua elevada capacidade de aprendizado de relações contextuais entre as palavras.

No campo de avaliação automática de redações, por muitos anos, investiu-se em abordagens fortemente baseadas na extração de características dos textos seguida do uso de algoritmos de classificação e regressão como *Support Vector Machines* (SVMs) e Árvores de Decisão. Modelos de redes neurais profundas, a exemplo das RNNs e CNNs, também já foram explorados por alguns trabalhos, e apresentaram bom desempenho principalmente na avaliação de critérios que envolvem maior subjetividade, como os que

tratam de argumentação e organização das ideias no texto (Marinho, Cordeiro *et al.*, 2022).

Entretanto, ainda que o BERT nos últimos anos tenha se apresentado como a solução arquitetural estado-da-arte para diversas tarefas de PLN, incluindo a classificação de textos, seu uso para a correção automática de redações ainda foi pouco explorado em trabalhos em língua portuguesa. Nesse sentido, o trabalho de Souza *et al.* (2020), que introduziu o BERTimbau, uma versão do BERT treinada com textos em português, e o trabalho de Marinho, Anchiêta *et al.* (2021), que criou um *corpus* de redações modelo ENEM corrigidas por especialistas, proporcionam uma conjuntura extremamente favorável à exploração de abordagens que utilizem esses recursos.

1.2 Motivação

Com mais de 3,9 milhões de inscritos em 2023, a prova do ENEM tem, ao longo dos últimos anos, se consolidado como o principal meio de entrada de estudantes no ensino superior brasileiro. Em razão disso, a redação, que consiste em uma das cinco partes da prova, desempenha um papel fundamental no resultado final dos candidatos e, por conseguinte, nas chances de ingresso na instituição e no curso desejados.

Para que o aluno obtenha uma boa nota no dia da prova, entende-se que o treinamento constante e repetitivo da escrita é fundamental, o que demanda um acesso a meios de correção que promovam um *feedback* rápido e consistente. Além disso, também há uma demanda alta por professores habilitados para a avaliação das redações produzidas no dia da prova, devido ao enorme volume de textos, o que gera um alto custo para o INEP. Dessa forma, a pesquisa e o desenvolvimento de sistemas de correção automatizada, os quais possam auxiliar tanto estudantes quanto avaliadores, se faz extremamente relevante para mitigar as dificuldades envolvidas nesse processo.

1.3 Objetivo

O objetivo desse trabalho é, portanto, estudar e explorar o uso do BERTimbau para a correção automática de redações do ENEM. Mais especificamente, foi escolhido focar somente na competência III, a qual avalia a capacidade do candidato de "selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista".

A decisão de focar nessa competência parte do entendimento de que o BERT se destaca justamente na capacidade de capturar relações complexas entre partes do texto, o que é extremamente difícil de se conseguir somente por meio de métodos de engenharia de *features*. Ainda assim, esse trabalho também se propõe a investigar as possibilidades do uso de técnicas de extração de características em conjunto com o BERTimbau, de modo a tentar se beneficiar das vantagens inerentes a cada uma das duas abordagens.

Capítulo 2

Trabalhos Relacionados

A pesquisa na área de avaliação automática de redações começou com Page (1966), que apresentou um dos primeiros sistemas de correção para textos em língua inglesa. Desde então, muitos trabalhos outros trabalhos já foram introduzidos, porém focaremos nesse breve capítulo somente nos principais trabalhos desenvolvidos especificamente para a língua portuguesa.

Junior *et al.* (2017) propuserem a construção de um sistema de avaliação de redações focado na predição da nota da competência I do ENEM, que avalia o domínio da modalidade escrita formal em língua portuguesa. Foram utilizadas técnicas de engenharia de características e algoritmos de classificação como *Suport Vector Machine* e Gradient Boosting Forests. O desempenho dos modelos foi medido utilizando-se as métricas de precisão, recall e o erro médio absoluto.

O trabalho de Amorim e Veloso (2017) propôs o primeiro sistema completo de AAR para correção de redações do ENEM com a tarefa de predizer tanto a nota final como as notas de cada uma das cinco competências. A abordagem utilizada foi de treinamento de um modelo regressor alimentado com 19 *features*, que utilizou em seu treinamento um *dataset* de 1840 redações avaliadas por especialistas e extraídas do site Educação UOL. Os autores buscaram analisar a importância de cada uma das 19 características no desempenho do modelo e utilizaram o QWK como principal métrica de avaliação de performance.

Já em Fonseca *et al.* (2018), foram seguidas duas abordagens. Na primeira, foi utilizada uma rede neural profunda composta por duas camadas de RNNs do tipo LSTM bidirecionais (BiLSTM). Na segunda abordagem, foi implementado um sistema baseado em características, no qual foram utilizadas 681 *features* para treinar cinco regressores separadamente. As estratégias foram avaliadas utilizando um *corpus* de 56644 redações avaliadas e a principal métrica usada também foi o QWK.

Júnior (2020) apresentou um estudo sobre o uso de redes neurais para a avaliação da nota total da redação do ENEM e seu trabalho tem como diferencial a implementação de uma arquitetura baseada em aprendizado multitarefa, na qual cada um dos temas de redação foi considerado como uma tarefa diferente. Assim com nos trabalhos anteriormente citados, o *dataset* utilizado nesse trabalho não foi disponibilizado abertamente.

O trabalho de Marinho, Anchiêta *et al.* (2021) foi o primeiro a criar e disponibilizar um *corpus* completo de redações do ENEM corrigidas segundo os critérios mais atuais de avaliação das competências. Os autores buscaram implementar as estratégias utilizadas por Amorim e Veloso (2017) e Fonseca *et al.* (2018) para testá-las no *corpus* desenvolvido por eles. Posteriormente, os autores apresentaram em Marinho, Cordeiro *et al.* (2022) uma investigação de métodos baseados em *features*, *embeddings* e RNNs para a correção de redações do ENEM e conseguiram melhorar o desempenho dos modelos de AAR anteriores. Um dos diferenciais dessa pesquisa foi a definição de conjuntos de características próprios para cada competência, os quais foram usados para treinar cinco modelos diferentes.

Capítulo 3

Conceitos e Ferramentas

3.1 BERT

O BERT, acrônimo em inglês para *Bidirectional Encoder Representations from Transformers*, ou Representações de Codificadores Bidirecionais a partir de Transformadores, em tradução livre, é um modelo de aprendizado de máquina usado para processamento de linguagem natural. Ele foi desenvolvido por pesquisadores da Google AI Language e apresentado por Devlin *et al.* (2019) como o primeiro modelo de linguagem verdadeiramente bidirecional. Essa característica possibilita que o modelo aprenda o contexto das palavras baseado em todo o texto, ou seja, considerando as palavras à sua esquerda e à sua direita, ao contrário de modelos direcionais, que processam a entrada de texto sequencialmente e, consequentemente, limitam o contexto de aprendizado.

A inovação da aplicação do treinamento bidirecional, possibilitada pelo desenvolvimento da técnica de modelagem de linguagem mascarada, somada à facilidade de refinar o modelo pré-treinado para tarefas específicas, fez com que o BERT revolucionasse a área de processamento de linguagem nos últimos anos, atingindo resultados estado-da-arte em mais de 11 tarefas clássicas, que incluem, por exemplo, análise de sentimentos e reconhecimento de entendidades nomeadas. Como os próprios autores definem, o BERT é "conceitualmente simples e empiricamente poderoso" e veremos a seguir o porquê.

3.1.1 Como o BERT funciona?

Tokenização

Antes de entrar nos aspectos arquiteturais do modelo, é necessário primeiro falar da etapa de tokenização. Esse processo consiste em fracionar o texto em partes menores, chamadas de *tokens*, e em seguida converter essas partes para um formato numérico que possa ser compreendido pelo modelo. Isso pode ser realizado de diversas formas, seja a nível de palavras, de subpalavras ou até mesmo de caracteres (Figura 3.1). A ideia geral é conseguir encontrar a menor e mais significativa representação que faça sentido para o modelo.

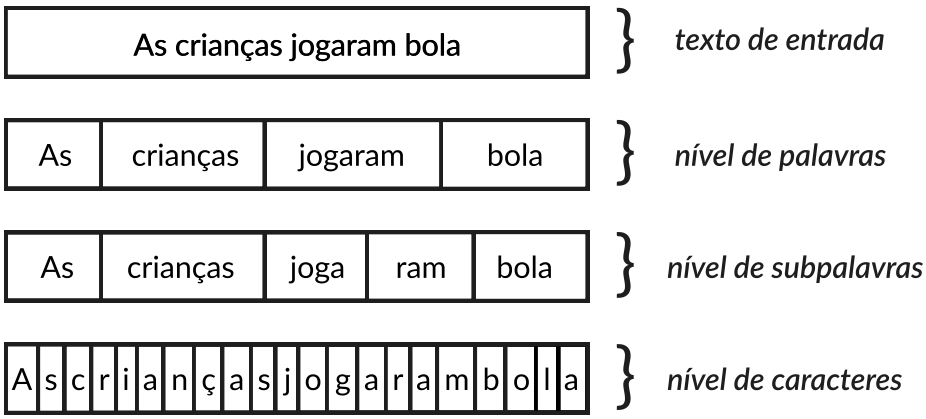


Figura 3.1: *Tipos de tokenização.*

O BERT utiliza um algoritmo chamado *WordPiece*[1], que realiza a tokenização a nível de subpalavras. A vantagem de se utilizar subpalavras está no equilíbrio que essa abordagem proporciona entre o tamanho do vocabulário e o tamanho da sequência de *tokens* resultante. Além disso, também se torna mais fácil lidar com palavras fora do vocabulário, reduzindo a necessidade de tratá-las como *tokens* desconhecidos.

A figura 3.2 ilustra as etapas do processo de tokenização. Nesse exemplo, o tokenizador recebe um lote (*batch*) de duas sequências de texto e devolve a codificação que será processada pelo BERT. Um lote é um conjunto de sequências de tamanho fixo que são passados de uma só vez para o modelo, o que permite um processamento mais rápido e eficiente dos dados a partir do uso de estratégias de paralelização.

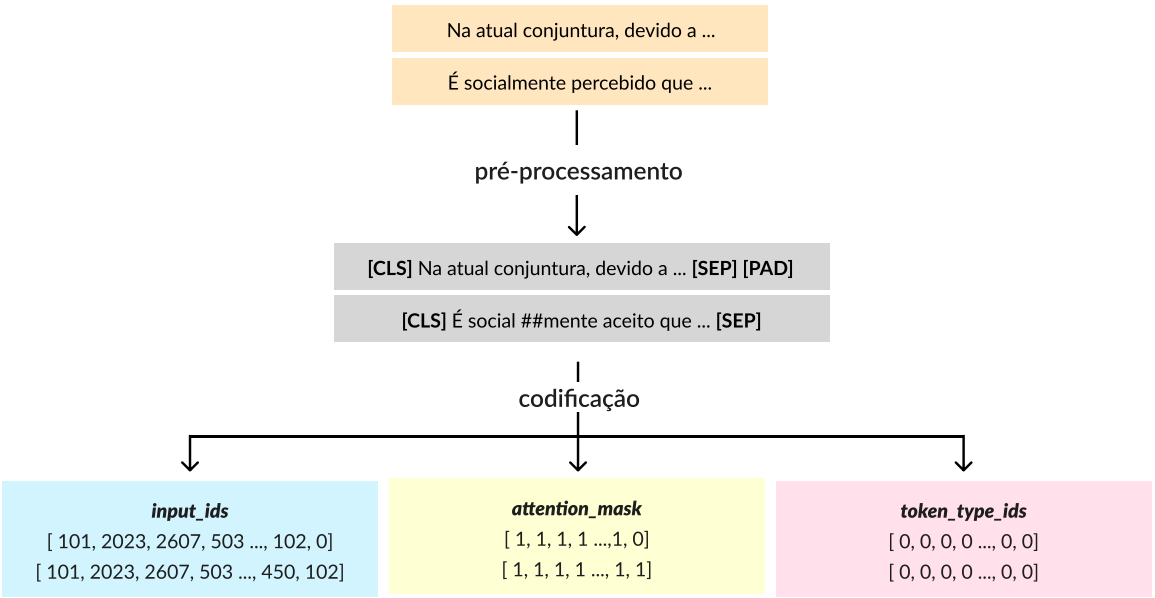


Figura 3.2: *Processo de tokenização.*

A primeira etapa da tokenização é o pré-processamento do texto, em que ele é par-

[1] O detalhamento do funcionamento do algoritmo de tokenização foge ao escopo desse trabalho. Para saber mais, o leitor pode consultar https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt.

ticionado em subpalavras. O prefixo ## é adicionado pelo tokenizador do BERT para identificar as subpalavras. Nessa etapa, também são adicionados ao texto alguns *tokens* especiais:

- **[CLS]:** Chamado de *token* de classificação, ele é acrescentado ao início de cada sequência e é usado para gerar uma representação agregada de todo texto, o que é útil para quando se deseja realizar tarefas de classificação.
- **[PAD]:** O *padding token*, ou *token* de preenchimento, é usado no final dos textos na quantidade necessária para que todas as sequências de texto do mesmo lote tenham igual tamanho. Isso é importante para a paralelização do processamento.
- **[SEP]:** O *token* de separação é acrescentado no final de cada texto em uma mesma sequência de entrada. Isso possibilita que o BERT diferencie e entenda os limites de cada trecho.
- **[UNK]:** *Token* utilizado para representar palavras desconhecidas do vocabulário, quando houverem.

Após esse preprocessamento, é então realizada a codificação dos tokens, a qual retorna os seguintes dados em vetores:

- **input_ids:** Cada token presente no vocabulário possui um inteiro único que o identifica. Os *input_ids* então mapeiam todos os *tokens* do texto com seus respectivos *ids*, cujos possíveis valores inteiros vão de 0 até o tamanho do vocabulário. No modelo BERT original, esse valor é igual a 30522.
- **attention_mask:** Sequências de texto menores que o tamanho máximo encontrado em um batch são preenchidas com os *padding tokens*. Todavia, esses tokens não agregam significado e, portanto, não devem influenciar o retorno o modelo. A *attention_mask* é utilizada para distinguir esses *tokens* dos demais. Todos os índices de elementos que sejam do tipo *padding tokens* recebem 0 e os demais recebem 1.
- **token_type_ids:** Quando se deseja realizar a tarefa de predição da próxima sentença, que é usada no treinamento do BERT como veremos mais à frente, a sequência de texto de entrada é composta de duas sentenças distintas. A ideia desse tipo de tarefa é predizer se a primeira sentença precede a segunda no texto original. Nesse contexto, os *token_type_ids* indicam quais *tokens* pertencem à primeira sentença (identificados com 0) e quais pertencem a segunda (identificados com 1). Para classificação de texto, entretanto, esses tokens não exercem influência no modelo e podem ser ignorados.

Embeddings

Chegamos agora de fato na arquitetura do BERT. Logo após a codificação dos tokens e o envio desses dados para a entrada do modelo, a primeira camada pela qual eles passam é o módulo dos *embeddings*. Mas o que são *embeddings*?

Embeddings são representações vetoriais de *tokens* as quais permitem que os organizemos no espaço. Eles podem ser entendidos como uma espécie de coordenadas dos *tokens*. Sendo assim, idealmente espera-se que *tokens* semanticamente próximos estejam mais próximos no espaço e *tokens* de sentido mais distintos estejam mais afastados entre si. O

tamanho desses vetores de *embeddings* é tão maior quanto a complexidade semântica e contextual que se deseja representar. Na versão base do BERT, os *embeddings* tem tamanho 768.

Para cada token, o BERT gera três *embeddings* (Figura 3.3):

- *Word Embeddings*: Esses são os *embeddings* que representam as palavras, mais especificamente os *tokens*. Cada *input_id* é portanto substituído pelo seu seu *word embedding* correspondente.
- *Positional Embeddings*: Esses são os *embeddings* que proveem ao modelo informação sobre a ordem dos *tokens* no texto. Eles representam posições de 0 a n-1, onde n é o tamanho máximo de uma sequência de texto, que é igual a 512 para o modelo base do BERT.
- *Token Type Embeddings*: Esses são os embeddings que representam os *token_type_ids*, de modo que só existem duas variações, uma correspondente aos *tokens* da primeira sentença e outra aos da segunda sentença. Contudo, como comentado anteriormente, essa informação só é relevante para o treinamento da tarefa de predição da próxima sentença.

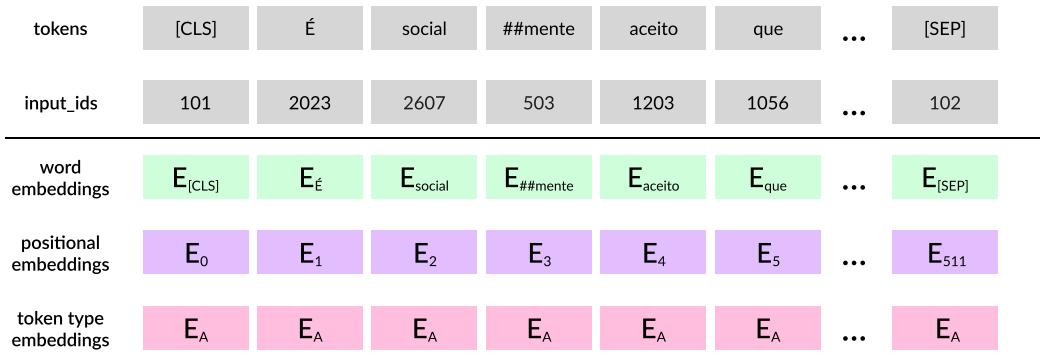


Figura 3.3: Embeddings *gerados pelo BERT.*

Os três *embeddings* são então somados de modo a produzir uma única representação vetorial que codifica informações semânticas e posicionais de cada *token* e identifica a sentença a que ele pertence (relevante somente para a tarefa de predição da próxima sentença). Todos esses *embeddings* são aprendidos durante a fase de pré-treino e podem ser posteriormente refinados para tarefas específicas - mais detalhes sobre esses processos serão explicados nas próximas seções.

Por fim, antes de serem enviados para o próximo módulo, esses *embeddings* ainda passam por uma camada de normalização e *dropout*, duas operações bastante comuns em várias partes do modelo. O procedimento de normalização tem como um de seus objetivos fazer com que a escala dos *embeddings* não apresente muita variação ao longo das camadas, garantindo maior estabilidade ao longo no treinamento. A camada de *dropout*, por sua vez, atua como uma etapa de regularização, zerando cada elemento do *embedding* com certa probabilidade (o BERT usa p=0.1). Isso faz como que o modelo não se torne muito dependente de nenhuma parte específica do *embedding*, tornando-o mais robusto e generalizável a novos dados.

Codificador

Com os *embeddings* gerados, chegamos agora ao módulo *Encoder*, ou Codificador. No modelo base do BERT, esse módulo é composto por 12 camadas *transformers* idênticas, em que a saída de uma camada alimenta a entrada da seguinte (Figura 3.4). Em geral, quanto mais camadas, mais padrões e relações complexas podem ser aprendidos pelo modelo. Podemos ter, por exemplo, relações sintáticas capturadas nas primeiras camadas, relações a nível de sentenças nas camadas intermediárias e relações a nível de documentos nas últimas camadas - tudo isso hipoteticamente, já que não é possível saber o que o modelo de fato aprende.

Cada uma dessas camadas, por sua vez, é composta por duas subcamadas principais: a camada de auto-atenção de múltiplas cabeças (*multi-head attention*) e a camada de rede neural *feed forward*. Vamos ver a seguir a função de cada uma.

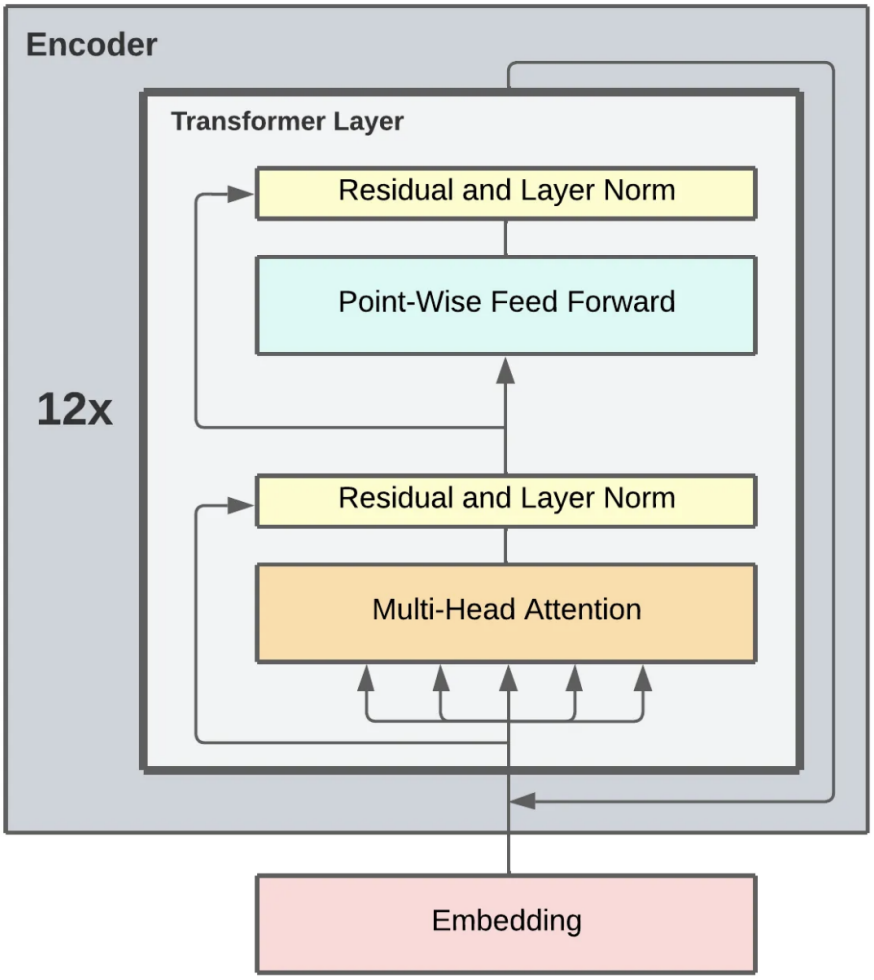


Figura 3.4: *Módulo codificador do BERT. Imagem extraída de https://medium.com/@alexmriggio/bert-for-sequence-classification-from-scratch-code-and-theory-fb88053800fa*

Auto-atenção

A chave para o sucesso da arquitetura *Transformers*, usada pelo BERT, está no uso da atenção - não por acaso, o nome do artigo que apresentou os *Transformers* se chama *Attention is all you need*, ou 'Atenção é tudo que você precisa' . A auto-atenção é o mecanismo que permite ao modelo definir a importância de cada palavra do contexto em relação à palavra sendo processada. Como resultado, a representação obtida para cada *token* é enriquecida com informações sobre como ele se relaciona com os demais *tokens* ao seu redor.

Note que antes da passagem dos *embeddings* pela camada de atenção todos os tokens com o mesmo *input_id* possuíam o mesmo *embedding*, diferindo somente em relação à sua posição na sequência. Porém, sabemos que palavras como rede e manga, por exemplo, podem ter diferentes significados em contextos distintos - rede pode se referir a rede de descanso, de computadores e manga pode se referir à fruta, a uma parte da roupa. É justamente na camada de atenção que o modelo se torna capaz de diferenciar os *embeddings* de cada um desses possíveis significados.

Por trás dos panos, o mecanismo de auto-atenção consiste na realização de uma série de operações matriciais envolvendo os *embeddings*. Essas operações calculam a similaridade entre todos os *embeddings* do texto e produzem uma pontuação para cada um que irá refletir o quanto cada *token* está "prestando atenção" nos demais *tokens* do texto, ou, em outras palavras, em que grau cada um dos *tokens* da sequência se relaciona com um *token* em particular.

A Figura 3.5 ilustra um exemplo de aplicação do mecanismo de auto-atenção. Considere as duas sentenças a seguir:

- O *gato* comeu o peixe porque **ele** estava faminto.
- O gato comeu o *peixe* porque **ele** estava fresco.

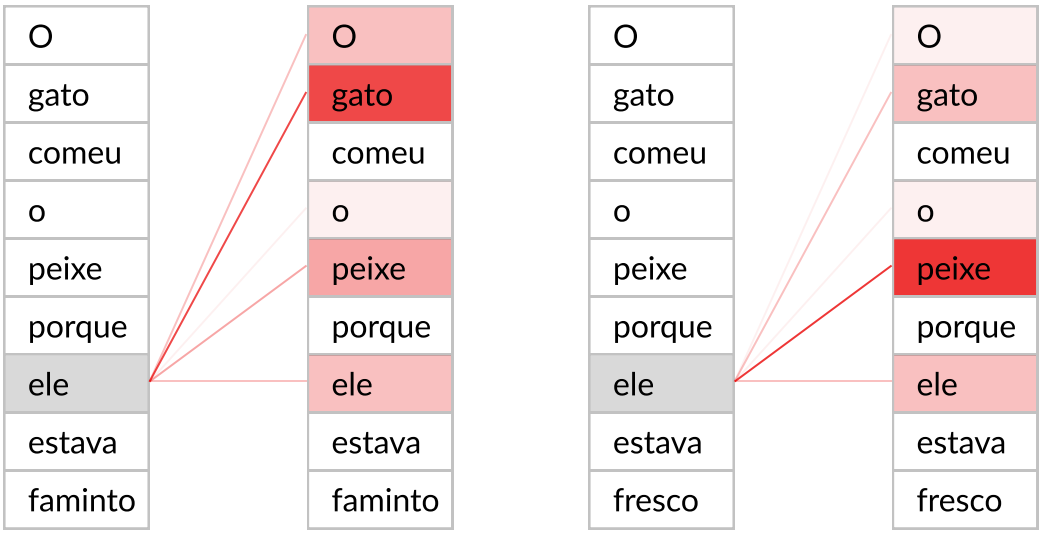


Figura 3.5: *Representação visual da auto-atenção. As palavras mais destacadas são as que a palavra em foco está prestando mais atenção. Imagem adaptada de https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452*

Na primeira sentença, está claro que a palavra “ele” se refere a “gato” e, na segunda, que se refere a “peixe”. Quando o modelo processa a palavra “ele”, a auto-atenção permite ao modelo associar corretamente corretamente o pronome“ele” à palavra correta, atribuindo-a uma maior pontuação.

Na arquitetura transformers, a unidade de processamento de atenção é chamada de cabeça de atenção (*attention head*) e, para cada palavra processada, são realizadas uma série de computações paralelas distintas em várias dessas unidades. Isso é o que se chama de atenção de múltiplas cabeças (*multi-head attention*). Cada uma dessas cabeças permitirá ao modelo aprender diferentes aspectos semânticos do texto que serão combinados na saída em uma única representação conjunta.

Podemos pensar então que são gerados na camada de atenção múltiplos subespaços de representações, os quais expandem a capacidade do modelo de focar em diferentes tipos de interações entre os *tokens* e com isso capturar nos *embeddings* interpretações mais ricas das nuances do texto. No exemplo da figura 3.6, vemos que, no processamento da palavra “ele”, a primeira cabeça está focando na palavra “gato”, enquanto a segunda está focando em “faminto”. Com efeito, a representação final da palavra “ele” gerada após a passagem pela camada de auto-atenção incorporará tanto aspectos da palavra “gato” quanto da palavra “faminto” [2].

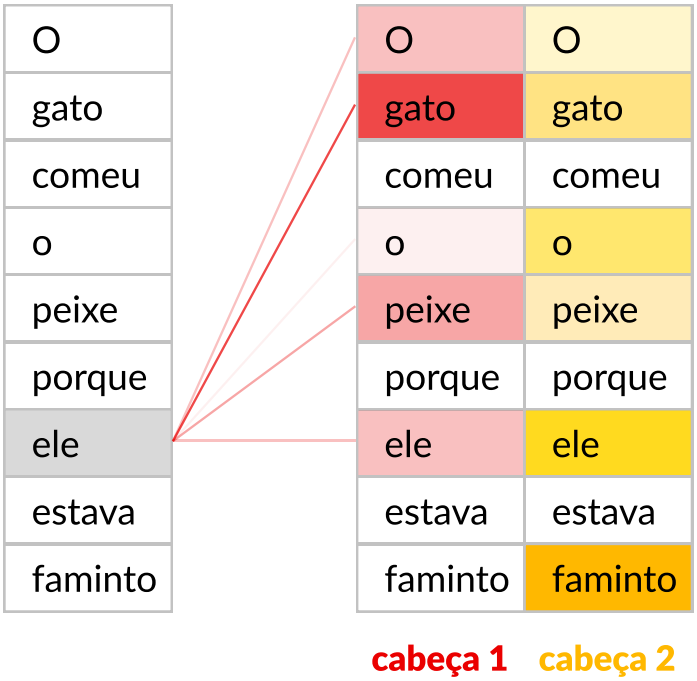


Figura 3.6: *Cálculos de atenção em múltiplas cabeças. Imagem adaptada de https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452*

Logo após o processamento pela camada de atenção de múltiplas cabeças, os *embeddings* retornados passam mais uma vez por uma camada de *dropout* e são somados aos *embeddings*

[2] Para uma explicação mais detalhada e bastante didática do funcionamento da camada de atenção de múltiplas cabeças, recomendamos ao leitor consultar https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

originais - os *embeddings* como eram antes de passarem pelo mecanismo de atenção. Esse processo de somar a entrada de uma camada com a sua saída é chamado de conexão residual. O objetivo é mitigar o problema do *vanishing gradient* (Hochreiter e Bengio, 2001) e permitir ao modelo identificar quais módulos não estão contribuindo para a melhora da performance. Ao final, os *embeddings* passam ainda por mais uma camada de normalização.

Rede neural *feed forward*

Chegamos então à subcamada de rede neural *feed forward*, o segundo componente principal da camada *transformers*. Seu objetivo é aumentar ainda mais a capacidade do modelo com parâmetros adicionais que podem ser aprendidos e facilitar o enriquecimento das representações camada a camada ao longo do módulo codificador. Nesse ponto, são aplicadas uma série de operações nos *embeddings*, que incluem passagem por uma função de ativação, aplicação de *dropout*, aplicação de conexões residuais e normalização. Essas últimas já foram vistas em etapas anteriores, então daremos um destaque à função de ativação, que aparece aqui pela primeira vez.

A função de ativação é uma função responsável por introduzir não-linearidade ao modelo. O objetivo é permitir a captura de padrões mais complexos dos dados que não apresentam um comportamento linear. A função de ativação utilizada na arquitetura *transformers* é a GELU, uma variação da ReLU, uma popular função de ativação utilizada em modelos de rede neural. A GELU dimensiona cada valor de entrada pela probabilidade cumulativa de todos os valores menores ou iguais ao valor de entrada em uma distribuição normal com média 0 e variância de 1. A principal vantagem da GELU em relação à ReLU é que ela é suave (diferenciável em todos os pontos), o que ajuda na otimização e convergência do modelo.

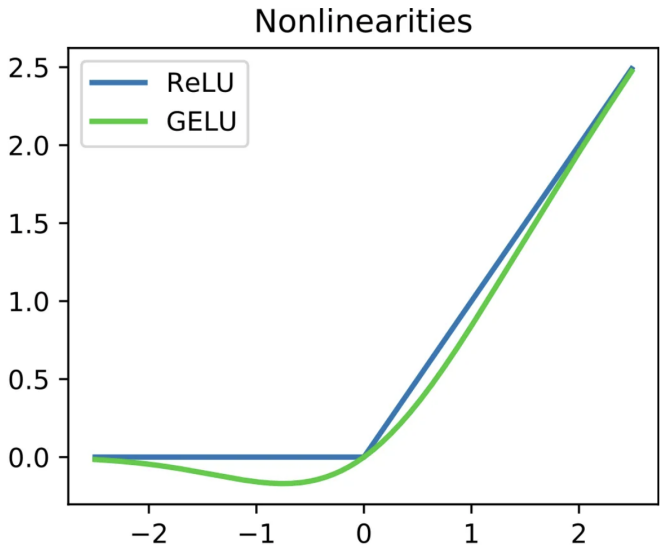


Figura 3.7: *Comparação das funções GELU e ReLU, destacando a suavidade apresentada pela GeLU. Imagem extraída de https://medium.com/@alexmriggio/bert-for-sequence-classification-from-scratch-code-and-theory-fb88053800fa*

Agregador

Chegamos finalmente ao último módulo do BERT, o módulo agregador (*pooler*). Ele recebe como entrada uma matriz devolvida pelo módulo codificador em que cada linha é uma representação contextualmente rica de um *token* da sequência de entrada (*embeddings*). Entretanto, para as tarefas de classificação de texto, precisamos de uma representação vetorial que agregue informações de todo o texto e não só de cada *token* individualmente.

Para isso, o agregador seleciona a representação do primeiro *token* da sequência de entrada, o *token* de classificação **[CLS]**. É esperado que, ao final do treinamento, ele contenha uma representação em alto nível de todo o texto e com isso possa ser usado para classificação. Antes de ser devolvido pelo modelo junto com a matriz de todas as representações, ele é processado ainda por uma camada linear e por uma função de ativação *tanh* (tangente hiperbólica), no intuito de aumentar ainda mais a capacidade do modelo de representar relações complexas.

3.1.2 Como o BERT aprende?

O trabalho original de Devlin *et al.* (2019) apresentou duas versões do BERT: $\text{BERT}_{\text{BASE}}$ e $\text{BERT}_{\text{LARGE}}$. A diferença entre ambos está essencialmente no tamanho. Considerando L como sendo o número de camadas *transformers* codificadoras, H o tamanho das representações e A o número de cabeças de atenção, cada modelo apresenta as seguintes dimensões:

- $\text{BERT}_{\text{BASE}}$: L = 12, H = 768, A = 12.
- $\text{BERT}_{\text{LARGE}}$: L = 24, H = 1024, A = 16.

Isso significa que o $\text{BERT}_{\text{BASE}}$ e o $\text{BERT}_{\text{LARGE}}$ possuem um total de 110 milhões e 340 milhões de parâmetros, respectivamente. Todos esses parâmetros são inicializados aleatoriamente e são esses eles que devem ser aprendidos pelo modelo durante o treinamento.

O processo de treinamento do BERT para uma tarefa específica é dividido em duas etapas: o pré-treino e o refinamento. Durante o pré-treino, o modelo é treinado do zero a partir de dados não-classificados (aprendizado auto-supervisionado) em duas tarefas distintas. Já no refinamento, primeiro o modelo é inicializado com os parâmetros aprendidos no pré-treino, em seguida uma nova camada é acoplada ao modelo de acordo com a tarefa que se deseja aprender e então todos os parâmetros são treinados usando dados rotulados (aprendizado supervisionado). Veremos a seguir mais detalhes sobre cada um desses processos.

Pré-treino

Ao treinar modelos de linguagem, há sempre o desafio de se definir o objetivo de predição. É comum escolher a "predição da próxima palavra" como a tarefa alvo, porém essa estratégia direcional inevitavelmente limita o contexto de aprendizado. Para superar esse desafio e viabilizar o treinamento de um modelo de linguagem bidirecional, o BERT utiliza duas estratégias:

A primeira estratégia é chamada modelagem de linguagem mascarada (*masked language*

modeling). Nessa tarefa, antes de passar a sequência de texto para o modelo, 15% das palavras são selecionadas aleatoriamente. Elas são então substituídas por uma de três opções: pelo *token* especial **[MASK]** com 80% de probabilidade, por um *token* aleatório do vocabulário com 10% de probabilidade, ou ainda, o *token* original é mantido com 10% de probabilidade.

O objetivo do modelo nessa estratégia deve ser predizer a palavra original que foi mascarada baseado no contexto provido pelas demais palavras não mascaradas do texto. Isso é feito adicionando-se uma cabeça de classificação (*classification head*) após a última camada do BERT, aplicando uma função softmax para obtenção das probabilidades de cada *token* e calculando a função de perda. Essa função de perda leva em consideração somente as predições para as palavras mascaradas e ignora a predição das demais, o que faz com que o modelo convirja mais lentamente que modelos direcionais, fator compensado pela maior compreensão contextual adquirida.

Já a segunda estratégia de treinamento é a predição de próxima sentença (*next sentence prediction*). Nessa tarefa, o modelo recebe pares de sentenças como entrada e aprende a predizer se a primeira sentença precede a segunda no texto original. Durante o treinamento, 50% das entradas são pares de sentenças consecutivas, enquanto nos outros 50% a segunda sentença é escolhida aleatoriamente do *corpus* de textos.

Para essa tarefa, os *tokens* especiais inseridos no pré-processamento são essenciais. O *token* de separação **[SEP]** é colocado no final de cada sentença para que o modelo consiga diferenciá-las, enquanto o *token* de classificação **[CLS]** é colocado no início da primeira sentença para armazenar a representação de toda a sequência de texto. Esse representação, após ser produzida e devolvida pelo modelo, passa ainda por outra cabeça de classificação, diferente da usada na tarefa de mascaramento, onde a probabilidade das sequências serem consecutivas é calculada com a aplicação da função softmax.

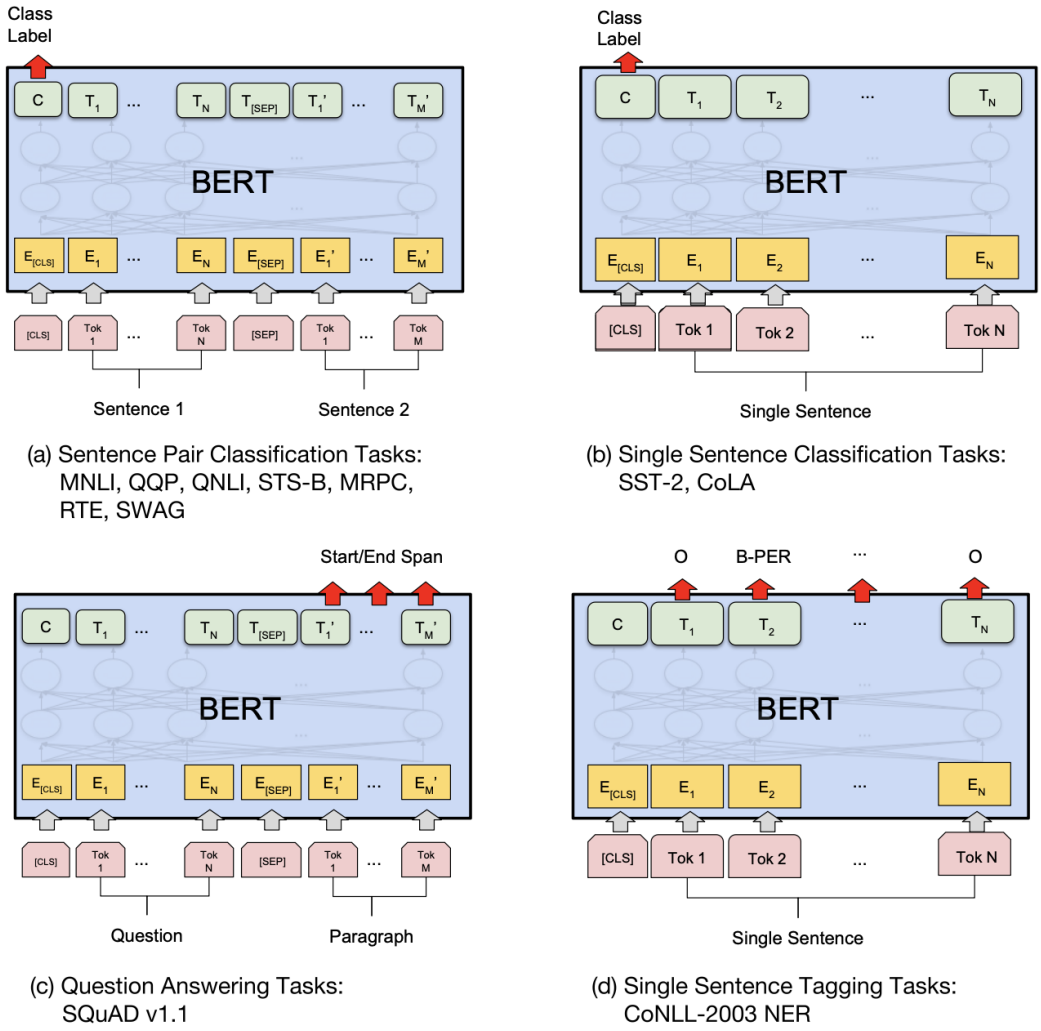
Um detalhe importante é que no treinamento do BERT as duas estratégias citadas são executadas ao mesmo tempo, o que é possível já que a primeira tem como objetivo aprender a representação de cada um dos *tokens* e a segunda de aprender a representação de todo o texto, armazenada no *token* **[CLS]**. Dessa forma, as duas cabeças acopladas ao modelo executam a sua tarefa de classificação e o modelo completo é otimizado por meio da minimização da função de perda conjunta das duas estratégias.

Refinamento

O refinamento é a etapa posterior ao pré-treino em que o modelo é ajustado para a tarefa específica que se deseja realizar. Nessa etapa, o modelo é inicializado com os parâmetros pré-treinados nas tarefas de modelagem de linguagem mascarada e predição de próxima sentença e as representações geradas por ele são usadas como entrada para uma simples camada de classificação linear. Todos os parâmetros são então refinados conjuntamente a partir de um conjunto de dados classificados específicos da tarefa. Note, porém, que somente um pequeno número de parâmetros - da camada linear - precisará ser aprendido do zero, já que os parâmetros do BERT já têm capturadas informações contextuais úteis que serão aproveitadas.

Comparado ao processo de pré-treino, o refinamento é significativamente mais simples

e exige menos recursos computacionais, menos tempo e menos dados de treinamento. Enquanto o pré-treino precisa de alguns dias de processamento em uma várias TPUs, o refinamento a partir de um modelo pré-treinado pode ser realizado em uma única GPU em apenas algumas horas. Por isso, a ideia é que o pré-treino seja realizado apenas uma única vez e o mesmo modelo possa ser refinado para diferentes tarefas. Esse procedimento é conhecido na área de PLN como *transferência de aprendizado*. Alguns exemplos de refinamento do BERT são ilustrados na figura 3.8.



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figura 3.8: *Refinamento do BERT em várias tarefas. Imagem extraída de* Devlin *et al., 2019*

3.2 BERTimbau

O modelo BERT original proposto por Devlin *et al.* (2019) usou na sua etapa de pré-treino um *corpus* composto pelo BooksCorpus (800 milhões de palavras) (Zhu *et al.*, 2015), e por textos da Wikipedia em inglês (2500 milhões de palavras). Sendo assim, ele se aplica apenas a tarefas de processamento de texto em inglês. Com o intuito de viabilizar a pesquisa e o desenvolvimento de aplicações em português em cenários com dados limitados e poucos

recursos para se treinar um modelo do zero, em Souza *et al.* (2020) foi apresentado pela primeira vez um conjunto de modelos BERT pré-treinados em um *corpus* de textos escritos em português brasileiro, denominado BERTimbau.

Seguindo o trabalho original do BERT, o BERTimbau foi treinado em dois tamanhos, *base* e *large*, com as mesmas configurações mencionadas em 3.1.2. Ambos os modelos foram disponibilizados abertamente para a comunidade de desenvolvedores e podem ser acessados em https://huggingface.co/neuralmind.

O *corpus* usado para o pré-treino dos modelos foi o brWaC (*Brazilian Web as Corpus*), produzido a partir de páginas web em português, que conta com 2,68 bilhões de *tokens* em 3,53 milhões de documentos - o maior e mais diverso *corpus* aberto em português disponível até então. Além disso, o vocabulário usado pelo tokenizador conta com 30.000 subpalavras e ambos os modelos são do tipo *cased*, ou seja, diferenciam caracteres minúsculos e maiúsculos.

3.3 A Competência III da Redação do ENEM

A prova de redação do ENEM exige do candidato a produção de um texto em prosa, do tipo dissertativo-argumentativo, sobre um tema de ordem social, científica, cultural ou política (Inep, 2023). A avaliação dos textos é realizada por dois corretores, de forma independente, de acordo com os critérios definidos por cinco competências, apresentadas na figura 3.9. Cada avaliador atribui uma nota entre 0 e 200 pontos, em intervalos de 40 pontos, para cada uma das cinco competências, que somadas irão compor a nota total de cada avaliador em até 1000 pontos. A nota final do participante é calculada pela média aritmética das notas totais atribuídas pelos dois avaliadores.

Competência I	Demonstrar domínio da modalidade escrita formal da língua portuguesa.
Competência 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.
Competência III	Selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista.
Competência IV	Demonstrar conhecimento dos mecanismos linguísticos necessários para a construção da argumentação.
Competência V	Elaborar proposta de intervenção para o problema abordado, respeitando os direitos humanos.

Figura 3.9: *Quadro de competências da Prova de Redação. Imagem extraída de Inep, 2023*

A competência III, selecionada para ser avaliada nesse trabalho, é a competência que avalia a capacidade do participante de "selecionar, relacionar, organizar e interpretar informações, fatos, opiniões e argumentos em defesa de um ponto de vista".

- Selecionar diz respeito ao processo de escolha, a partir do repertório dos textos motivadores e do seu próprio repertório, das informações e dos argumentos que serão trabalhados no texto.

- Relacionar se refere à capacidade de encadear as ideias de forma progressiva, deixando claro o caminho percorrido para se alcançar o seu ponto de vista sobre o tema.
- Organizar trata de como o participante realizou seu projeto de texto e se ele apresentou uma hierarquia produtiva de seus argumentos.
- Interpretar analisa se o candidato foi capaz de contextualizar as ideias apresentadas em relação ao tema e ao seu ponto de vista, garantindo que os fatos e opiniões selecionados se mostram pertinentes para a defesa de seu ponto de vista.

Os níveis de desempenho específicos que orientam a avaliação da competência III, são mostrados na figura 3.10 abaixo.

200 pontos	Apresenta informações, fatos e opiniões relacionados ao tema proposto, de forma consistente e organizada, configurando autoria, em defesa de um ponto de vista.
160 pontos	Apresenta informações, fatos e opiniões relacionados ao tema, de forma organizada, com indícios de autoria, em defesa de um ponto de vista.
120 pontos	Apresenta informações, fatos e opiniões relacionados ao tema, limitados aos argumentos dos textos motivadores e pouco organizados, em defesa de um ponto de vista.
80 pontos	Apresenta informações, fatos e opiniões relacionados ao tema, mas desorganizados ou contraditórios e limitados aos argumentos dos textos motivadores, em defesa de um ponto de vista.
40 pontos	Apresenta informações, fatos e opiniões pouco relacionados ao tema ou incoerentes e sem defesa de um ponto de vista.
0 ponto	Apresenta informações, fatos e opiniões não relacionados ao tema e sem defesa de um ponto de vista.

Figura 3.10: *Níveis de desempenho avaliados na Competência III. Imagem extraída de* Inep, 2023

A escolha dessa competência em específico para ser avaliada se deve ao fato de que ela essencialmente avalia a construção de sentido do texto. Sendo assim, como o BERTimbau se destaca justamente na capacidade de aprendizado de nuances semânticas e relações complexas entre as partes do texto, naturalmente, espera-se que seu desempenho na avaliação dessa competência em particular seja superior ao de outros modelos.

3.4 Corpus Essay-BR

Criado por Marinho, Anchiêta *et al.* (2021), com vistas a suprir a ausência de *corpus* de redações em português corrigidas manualmente, o Essay-Br é um *dataset* robusto que concentra uma coleção de redações escritas por estudantes brasileiros em uma plataforma on-line e avaliadas por especialistas de acordo com as cinco competências definidas pelo ENEM. Ao todo, são 4570 textos e 86 propostas de texto na versão básica e, na versão estendida, 6577 textos e 151 propostas de texto. Os temas incluem direitos humanos, questões políticas, atividades culturais, notícias falsas, movimentos populares, covid-19, entre

outros. Ambas as versões do *corpus* - padrão e estendida - estão disponíveis abertamente em https://github.com/rafaelanchieta/essay.

Para criar esse *corpus*, o primeiro desse tipo disponível publicamente que segue os critérios mais atualizados de avaliação da redação do ENEM, os autores extraíram os dados a partir de dois sites públicos e que permitem o uso das redações para fins de científicos: Vestibular UOL e Educação UOL. O *corpus* foram coletados e estruturados como mostra a tabela 3.1.

atributo	valor
prompt	86
title	Bem estar mental
essay	[É notório que as redes sociais estão cada vez...
c1	120
c2	120
c3	120
c4	160
c5	160
score	680

(a) *Redação*

atributo	valor
id	86
title	Impactos do uso das redes sociais na saúde mental
description	['Um impacto enorme, uma influência de grande ...
category	saúde

(b) *Proposta de texto*

Tabela 3.1: *Estrutura dos dados na versão estendida do* corpus *Essay-Br*

Cada redação possui o índice da sua proposta de texto correspondente (*prompt*), o título da redação (*title*)[3], o texto (*essay*), estruturado em uma lista de parágrafos, as notas das competências de 1 a 5 (*c1, c2, c3, c4, c5*) e a pontuação final obtida pela soma das notas de todas as competências. Já as propostas de texto, organizadas em um conjunto de dados separado, apresentam um índice (*id*), um título (*title*), um compilado de textos motivadores (*description*) e uma categoria (*category*).

No repositório contendo os dados, os autores disponibilizam ainda um *script python* para realizar a construção do *corpus* e a divisão entre os dados de treino, validação e teste. Para criar esses três conjuntos, os dados primeiro são embaralhados e em seguida repartidos de modo a preservar, em cada um dos conjuntos, a distribuição das notas finais e por competência verificada no *dataset* completo. Também são preservadas as proporções

[3] Algumas redações não apresentam um título, pois este é um elemento opcional na produção da redação. Embora seja considerado linha escrita, ele não é avaliado em qualquer aspecto relacionado às competências da matriz de referência (Inep, 2023) e por isso foi desconsiderado nos experimentos realizados.

de características como média de parágrafos, sentenças e *tokens* por redação, cujos valores verificados são 4, 2 e 300, respectivamente.

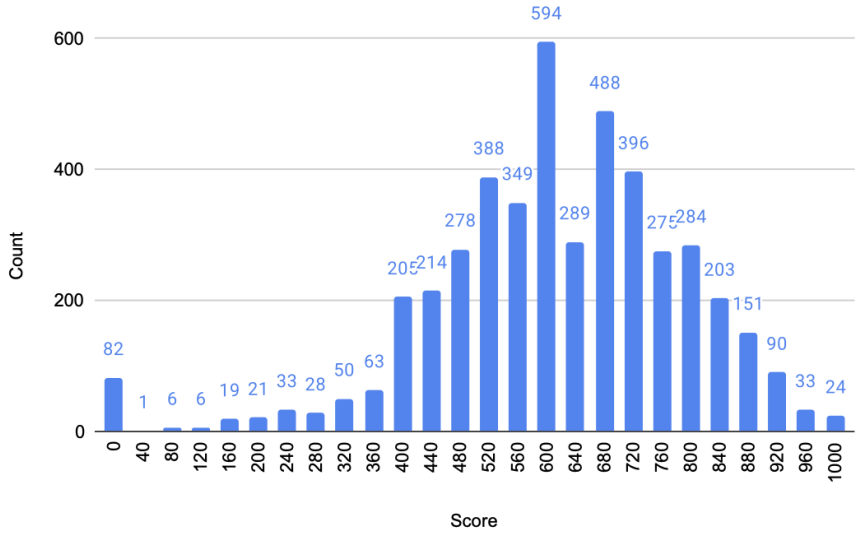


Figura 3.11: *Gráfico da distribuição das notas finais no dataset completo do Essay-Br*

Nota-se na figura 3.11 que a distribuição das notas verificada no *dataset* completo - e replicada em cada uma das partições - é desbalanceada. Das notas totais finais, que variam de 0 a 1000 pontos, as três notas mais frequentes são 600, 680, and 720, representando 13.00%, 10.68% e 8.67% do *corpus*. Já para as notas por competência, que variam de 0 a 200, a nota mais observada em todas as cinco competências é 120, indicando que, em geral, os estudantes possuam uma dominância média de todos os campos de avaliação.

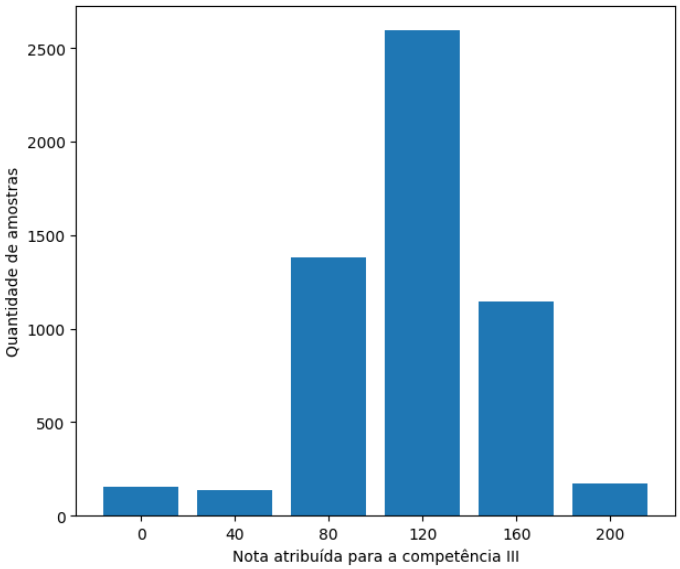


Figura 3.12: *Gráfico da distribuição das notas da competência III no dataset completo do Essay-Br*

3.5 Biblioteca Spacy

spaCy é uma biblioteca para processamento de linguagem natural em *python* que fornece modelos e *pipelines* pré-treinados para mais de 70 línguas (Honnibal *et al.*, 2020). Ela possui uma API simples e eficiente para a realização de tarefas de extração de informações de textos em larga escala e que permite construção de sistemas avançados de PLN. Dentre os recursos disponíveis estão componentes para a realização de reconhecimento de entidades nomeadas, análise de dependência, segmentação de frases, classificação de texto, lematização e análise morfológica.

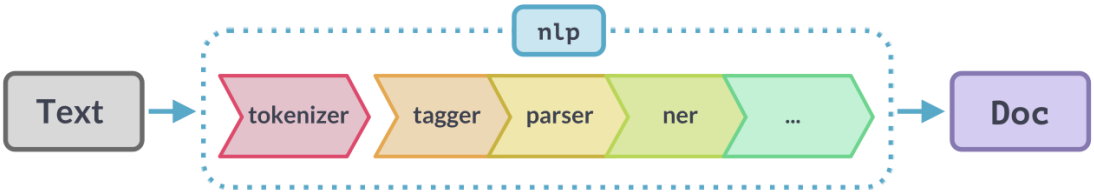


Figura 3.13: *Diagrama da pipeline nlp do spaCy. Imagem extraída de https://spacy.io/usage/processing-pipelines*

Os modelos pré-treinados podem ser baixados como pacotes e executados em poucas linhas. Ao chamar a *pipeline nlp* em um texto, o *spaCy* tokeniza o texto e produz um objeto próprio do tipo *doc*, que é então processado em várias etapas, as quais geralmente incluem um classificador, um lematizador, um analisador e um identificador de entidadades nomeadas (Figura 3.13). Após o processamento, basta então acessar os atributos do objeto *doc* para obter as informações extraídas do texto. O programa 3.1 mostra um exemplo de uso da *pipeline nlp* para análise e visualização de dependências em uma setença.

Programa 3.1 Exemplo de uso da *pipeline nlp* do *spaCy* para análise de dependências.

```
1  import spacy
2  from spacy import displacy
3
4  nlp = spacy.load("pt_core_news_lg")
5  doc = nlp("Carros autônomos transferem a responsabilidade de acidentes para
       os fabricantes")
6  displacy.render(doc, style="dep", options={"compact": True})
```

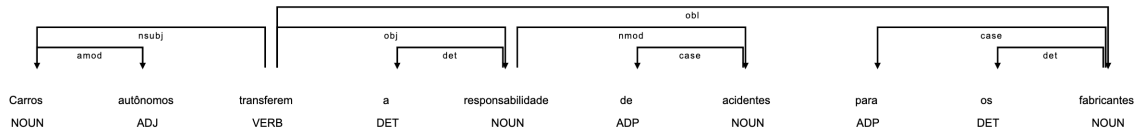


Figura 3.14: *Imagem gerada pelo programa 3.1*

A *pipeline* do spaCy para textos em língua portuguesa `pt_core_news_lg`, disponibilizada abertamente em https://spacy.io/models/pt, foi utilizada nesse trabalho para auxiliar na extração de características das redações utilizadas em algumas das estratégias de avaliação automática em conjunto com o BERTimbau.

Capítulo 4

Desenvolvimento

Neste capítulo, descrevemos o procedimento de refinamento do modelo pré-treinado $\text{BERTimbau}_{\text{BASE}}$. Para fins didáticos, detalhes muito específicos da implementação feita para esse trabalho foram omitidos ou simplificados, de modo a facilitar a descrição das principais etapas do processo. Contudo, o código completo do trabalho está disponível para consulta em https://github.com/ldcampos/essay_scoring_with_bertimbau. Convém ainda destacar, para além das ferramentas e bibliotecas apresentadas no capítulo 3, o uso auxiliar dos seguintes recursos:

- ***pytorch*:** Biblioteca *python* de código aberto otimizada para a implementação de modelos de aprendizado profundo, como modelos de visão computacional e processamento de linguagem natural. Sua grande vantagem é a capacidade de realizar cálculos usando tensores, estruturas similares a vetores e matrizes, mas que podem ser processadas tanto em CPUs quanto em GPUs e outros aceleradores de hardware.
- ***google colab*:** Serviço de *jupyter notebook* hospeadado pela *Google* que permite o desenvolvimento e a execução em nuvem de códigos em *python* por meio do navegador. A maior vantagem do uso de *notebooks* no *colab* é o acesso fácil a recursos computacionais como GPUs e TPUs, o que o torna especialmente adequado para tarefas de aprendizado de máquina.
- ***hugging face*:** Plataforma aberta e colaborativa voltada à comunidade de desenvolvedores de IA, que permite o desenvolvimento e compartilhamento de modelos e recursos de aprendizado de máquina. Ela inclui ainda uma API que auxilia, dentre outras coisas, no uso e treinamento dos modelos e na manipulação de *datasets*.
- ***scikit-learn*:** Biblioteca *python* aberta que fornece ferramentas simples e eficientes para análise de dados e aprendizado de máquina. Inclui algoritmos úteis para cálculo de métricas de avaliação e comparação de modelos (Pedregosa *et al.*, 2011).
- ***pandas*:** Biblioteca *python* aberta que oferece ferramentas flexíveis e eficientes para análise, manipulação e visualização de dados relacionais ou rotulados.
- ***numpy*:** Biblioteca de código aberto para desenvolvimento em *python* que dá suporte à criação e manipulação de poderosas estruturas vetoriais de múltiplas dimensões, as quais podem ser operadas por uma coleção de funções matemáticas de alto nível.

4.1 Refinamento do BERTimbau

4.1.1 Carregamento e Pré-processameto dos dados

O primeiro passo para refinamento de um modelo é o carregamento e pré-processamento do *dataset*. Como já citado na seção 3.4, o conjunto de dados escolhido para essa tarefa foi a versão extendida do *corpus Essay-BR* e os autores já disponibilizam um *script* que realiza a divisão do *dataset* em conjuntos de treino (70%), validação (15%) e teste (15%).

O conjunto de treino é usado para o processo de aprendizado de fato, em que o modelo ajusta os pesos internos da rede de modo a melhorar suas predições. O conjunto de validação, por sua vez, serve para verificar, durante o treinamento, se o modelo não está ficando muito ajustado às instâncias de treino e, com isso, perdendo sua capacidade de generalização para novos dados. Já o conjunto de teste serve como uma coleção de dados novos nunca visto pelo modelo e que podem, portanto, ser usados para calcular as métricas finais de avaliação de desempenho do modelo. Todos esses dados são armazenados em *dataframes* da biblioteca *pandas*.

Após o carregamento, a primeira modificação importante a ser realizada no pré-processamento é a remoção das colunas de dados desnecessárias para nosso ojetivo de treinamento. Dentre os atributos listados na seção 3.4, são mantidos somente o texto da redação, que alimentará a entrada do modelo e a nota da competência III, que é o rótulo (*label*) que o modelo deve aprender a predizer.

Outra modificação necessária está relacionada ao tipo de modelo que pretendemos treinar. Para a tarefa de AAR, há essencialmente dois caminhos possíveis. O primeiro é treinar um modelo regressor, em que o objetivo seria predizer diretamente a nota da competência dentro do intervalo de 0 a 200. O segundo é treinar um classificador, com o objetivo de predizer, dentre um pequeno conjunto de classes pré-determinadas, a qual delas a redação pertence.

Tradicionalmente na área de AAR, praticamente todas as pesquisas abordam essa tarefa como uma tarefa de regressão, como mostram Ke e Ng (2019). Entretanto, nesse mesmo trabalho, os autores também demonstram preocupação com o fato de que essa abordagem falha em prover um *feedback* adequado aos alunos, que não têm clareza em relação ao motivo da nota recebida.

A decisão final foi de implementar um modelo classificador nesse trabalho de conclusão de curso. Dois fatores foram preponderantes para essa escolha. O primeiro é o fato de que o objetivo do trabalho está restrito à avaliação de somente uma competência e não da nota final, como é feito na maioria dos trabalhos da área de AAR. Sendo assim, como as competências da redação do ENEM recebem notas baseadas em 6 níveis de desempenho bem determinados, como foi mostrado na figura 3.10, podemos facilmente usar esses níveis como classes. Além disso, os níveis de avaliação possuem descrições que justificam a nota atribuída, o que permite ao aluno ter um *feedback* da nota recebida.

Com isso, as notas de 0 a 200 atribuídas para a competência III foram então mapeadas para os níveis de desempenho correspondentes (Figura 4.1), de modo que o modelo possa

interpretá-los como índices que identificam as classes 6 classes do problema.

nota		*índice*
0	→	0
40	→	1
80	→	2
120	→	3
160	→	4
200	→	5

Figura 4.1: *Mapamento das notas em índices correspondentes aos níveis de avaliação das competências.*

4.1.2 Tokenização

Como vimos em 3.1.1, antes de passar os dados para o modelo é necessário realizar a tokenização. Para isso, precisamos inicializar o tokenizador a partir do *checkpoint* do modelo que iremos refinar. O *checkpoint* é essencialmente uma string que define o caminho até um modelo pré-treinado disponível no *Hugging Face* (no nosso caso o BERTimbau$_{\text{BASE}}$) e que pode ser baixado por meio da biblioteca *transformers*. O tokenizador deve ser definido a partir dele para garantir que estamos usando o mesmo vocabulário usado no pré-treino. Isso é feito usando a classe `BertTokenizer`.

A função `create_tokenized_dataset`, implementada no programa 4.1, é a responsável por realizar o processo de tokenização e codificação dos dados armazenados nos *dataframes*. Cada redação, armazenada como uma lista de parágrafos, é transformada em uma única string contínua, usando o método `join`, e é processada individualmente pela função `tokenizer.encode_plus`, da biblioteca *transformers*, que retorna um dicionário contendo os *input_ids* e a *attention_mask*, já no formato de tensores. Nessa função, definimos que cada sequência deve ter 512 *tokens*, tamanho máximo suportado pelo BERTimbau$_{\text{BASE}}$, e especificamos que a sequência deve ser truncada, ou seja, cortada, caso seja maior que esse valor, e preenchida com o *token* **[PAD]** caso seja menor. Por fim, a função retorna todos os *input_ids*, as *attention_masks* e também os rótulos (*labels*) das redações em um *dataset* de tensores gerado pela classe `TensorDataset` do *pytoch*.

4.1.3 Preparação dos Dados

Como os modelos baseados em *transformers* realizam o processamento de dados em lotes, após transformar os conjuntos de treino e validação em `TensorDatasets`, utilizamos a classe `DataLoader`, do *pytorch*, para criar um iterador sobre esses dados (Programa 4.2). Isso é fundamental, pois ajuda a reduzir significativamente o uso de memória durante o treinamento, já que, diferentemente de um *loop*, um iterador não requer que todo o dataset seja carregado na memória de uma só vez.

Para criar o iterador, é necessário definir o tamanho dos lotes que iremos passar. Em Devlin *et al.* (2019), os autores recomendam lotes de tamanho 16 ou 32. Por questões

Programa 4.1 Função de tokenização e codificação dos dataframes.

```
checkpoint = "neuralmind/bert-base-portuguese-cased"
tokenizer = BertTokenizer.from_pretrained(checkpoint)

max_len = 512

def create_tokenized_dataset(dataframe):
# Tokenize all the essays and map the tokens to their word IDs.
input_ids = []
attention_masks = []

print('Encoding all essays in the dataset...')

# For every essay...
for essay in dataframe['essay']:
 encoded_dict = tokenizer.encode_plus(
    " ".join(essay), # Essay to encode.
    add_special_tokens = True, # Add '[CLS]' and '[SEP]'
    max_length = max_len, # Pad & truncate all sentences.
    truncation = True,
    padding = 'max_length',
    return_attention_mask = True, # Construct attn. masks.
    return_tensors = 'pt', # Return pytorch tensors.
 )

 # Add the encoded essay to the list.
 input_ids.append(encoded_dict['input_ids'])

 # And its attention mask (simply differentiates padding from non-padding).
 attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(dataframe['label'].tolist())

return TensorDataset(input_ids, attention_masks, labels)
```

Programa 4.2 Criação dos iteradores sobre os dados de treinamento e validação.

```
batch_size = 16

train_dataloader = DataLoader(
        train_dataset, # The training samples.
        sampler = RandomSampler(train_dataset), # Select batches randomly
        batch_size = batch_size # Trains with this batch size.
    )

validation_dataloader = DataLoader(
        val_dataset, # The validation samples.
        sampler = SequentialSampler(val_dataset), # Pull out batches
            sequentially.
        batch_size = batch_size # Evaluate with this batch size.
    )
```

de memória, definimos lotes de tamanho 16. Definimos ainda que a seleção dos lotes de treinamento deve ser aleatória a cada época com o uso da classe `RandomSampler` do *pytorch*. Já os lotes de validação são sempre selecionados de forma sequencial pela classe `SequentialSampler`.

4.1.4 Configurações para o treinamento

A arquitetura da nossa rede é definida na classe `CustomModel`, que herda de `nn.Module`, a classe base para implementação de redes neurais em *pytorch* (Programa 4.3). Em `__init__`, são inicializadas todas as camadas que irão compor a nossa rede, enquanto no método `forward` são implementadas as operações que serão aplicadas nos dados de entrada. Implementamos uma rede com 3 camadas principais:

1. A primeira camada é a camada do BERTimbau, instanciada pela classe `BertModel`, da biblioteca `transformers`, com os parâmetros pré-treinados e carregados a partir do `checkpoint` da versão `base` do BERTimbau. Ela recebe os *input_ids* e a *attention_mask* dos textos e retorna as representações contextuais dos *tokens* e a representação agregada dos textos. Como a tarefa a ser realizada é de classificação, somente a representação agregada é passada para a próxima camada.

2. A segunda camada é simplesmente uma camada de aplicação de *dropout*, uma estratégia de regularização já abordada na seção 3.1. A probabilidade de *dropout* foi definida em 0.3.

3. A terceira camada compõe junto com a camada anterior o que chamamos de "cabeça de classificação". Nessa última camada, é realizada uma transformação linear, que reduz o tamanho da representação produzida pelo BERTimbau de 768 para 6 valores, correspondentes a cada uma das 6 classes do problema. Esses valores devolvidos são chamados de *logits* e podem ser interpretados como predições brutas das classes. A classe com maior *logit* é a que o modelo está indicando como mais provável.

É necessário ainda definir mais alguns dispositivos importantes para o treinamento, como a função de perda, um otimizador e um cronograma de aprendizado. A função

Programa 4.3 Implementação da classe CustomModel que define a arquitetura da rede neural para o refinamento.

```
1    class CustomModel(torch.nn.Module):
2    def __init__(self):
3       super(CustomModel, self).__init__()
4       self.l1 = transformers.BertModel.from_pretrained(checkpoint)
5       self.l2 = torch.nn.Dropout(0.3)
6       self.l3 = torch.nn.Linear(768, 6)
7
8    def forward(self, ids, attention_mask):
9       _, output_1 = self.l1(ids, attention_mask, return_dict=False)
10      output_2 = self.l2(output_1)
11      output = self.l3(output_2)
12      return output
```

de perda é usada para calcular quanto as predições do modelo treinado estão distantes dos valores corretos. Instanciamos a função usada no nosso modelo a partir da classe `CrossEntropyLoss`, do *pytorch*, que recebe os *logits* brutos produzidos pelo modelo e os rótulos corretos das classes, normaliza os *logits* e calcula a função de entropia cruzada, definida pela seguinte fórmula:

$$L_{CE}(\hat{y}, y) = -log\, p(y|x) = -[y log \hat{y} + (1-y) log(1-\hat{y})] \tag{4.1}$$

onde x é o texto a ser classificado, y é o rótulo correto e $\hat{y}$ é o rótulo predito pelo modelo.

O otimizador, por sua vez, é o principal algoritmo responsável pelo aprendizado. A otimização é o processo de ajuste dos parâmetros da rede de modo a reduzir o valor da função de perda a cada passo do treinamento. Para isso, precisamos calcular o gradiente da função de perda em relação a todos os parâmetros da rede, o que é feito através de um algoritmo chamado de *backpropagation*, ou retropropagação (Rumelhart *et al.*, 1986). O gradiente é essencialmente um vetor que aponta para a direção de maior crescimento da função; logo, se seguirmos na direção oposta, conseguimos diminuir a perda.

Para criar o otimizador, foi utilizado o objeto `optimizer` do *pytorch*, que encapsula toda a lógica do otimização (Programa 4.4). Para inicializá-lo, escolhemos o algoritmo `AdamW` (Loshchilov e Hutter, 2019). Ele recebe os parâmetros do modelo que devem ser aprendidos, um parâmetro especial *epsilon (=1e-8)*, que previne divisões por zero, e uma taxa de aprendizado que define o quanto esses parâmetros serão atualizados a cada iteração - valores menores resultam em um treinamento mais lento, enquanto valores muito grandes podem resultar em comportamentos não-esperados. Seguindo a faixa de valores sugeridos pelos autores do BERT, que varia de 3e-4 a 3e-5, foi utilizada uma taxa de aprendizado igual a 2e-5.

Em conjunto com o otimizador, utilizamos também um cronograma de aprendizado linear. A função dele é reduzir a taxa de aprendizado a cada iteração sobre um lote de dados no treinamento, do valor inicial até 0, na última iteração. A ideia é que quanto mais nos aproximamos do final do treinamento, menor queremos que seja o passo de otimização,

Programa 4.4 Trecho de inicialização do otimizador.

```
learning_rate = 2e-5

optimizer = AdamW(model.parameters(),
          lr = learning_rate,
          eps = 1e-8
        )
```

visto que estamos próximos de atingir a melhor performance do modelo e uma taxa de aprendizado maior poderia provocar uma mudança de percurso muito drástica.

Resta somente uma última definição para iniciarmos o treinamento: o número de épocas. Uma época consiste em uma passagem completa dos dados de entrada pela rede. Usualmente, uma rede é treinada em várias épocas para que o modelo consiga um bom ajuste de parâmetros, visto que uma única época se mostra insuficiente para provocar uma queda substancial no valor calculado pela função de perda. Definimos inicialmente 6 épocas de treinamento, porém, ao longo dos testes do modelo, verificou-se que somente 4 épocas já seriam suficientes para otimizar a rede, como sugeriram os autores do BERT.

4.1.5 Treinamento

Definidos todos os hiperparâmetros (tamanho dos lotes, taxa de aprendizado, número de épocas) e os dispositivos auxiliares (função de perda, otimizador, cronograma de aprendizado), podemos finalmente inicializar o nosso modelo a partir da classe `CustomModel` que definimos e movê-lo para a GPU para maior agilidade de processamento. Em cada época de treinamento, iteramos sobre os lotes amostrados pelo `DataLoader` e as seguintes ações são executadas:

1. Os dados de entrada do lote amostrado, que incluem os *input_ids*, a *attention_mask* e os rótulos das classes, são carregados na GPU junto com o modelo.

2. Os gradientes calculados para o lote anterior são zerados para evitar que eles sejam acumulados ao longo das iterações.

3. O *forward pass* é aplicado, ou seja, os dados são enviados ao modelo, que retorna as predições obtidas no formato de *logits*.

4. A função de perda é calculada sobre os *logits* para verificar o quão distantes as predições estão dos rótulos reais.

5. O valor retornado pela função de perda é acumulado para permitir o cálculo da média da perda no final de cada época.

6. O *backward pass* é aplicado, ou seja, os gradientes de todos os parâmetros em relação à função de perda são calculados através do algoritmo de retropropagação.

7. Os gradientes calculados são normalizados para evitar o problema da explosão de gradientes.

8. Os parâmetros da rede são atualizados pelo otimizador em função dos gradientes calculados e considerando a taxa de aprendizado atual.

9. A taxa de aprendizado é reduzida pelo cronograma de aprendizado.

4.1.6 Validação

A cada época, além das etapas enumeradas na seção acima, que compõe o chamado *loop* de treinamento, é executado também um *loop* de validação logo em seguida. Esse *loop* é bastante semelhante ao *loop* de treinamento, sendo executadas as seguintes ações:

1. Os dados de entrada do lote amostrado, que incluem os *input_ids*, a *attention_mask* e os rótulos das classes, são carregados na GPU junto com o modelo.

2. O *forward pass* é aplicado, ou seja, os dados são enviados ao modelo, que retorna as predições obtidas no formato de *logits*.

3. A função de perda é calculada sobre os *logits* para verificar o quão distantes as predições estão dos rótulos reais.

4. O valor retornado pela função de perda é acumulado para permitir o cálculo da média da perda no final de cada época.

5. A acurácia do modelo é também calculada e acumulada para fornecer mais dados sobre a evolução da performance do modelo a cada época.

Em comparação à etapa de treinamento, essa é uma etapa expressivamente mais rápida e que consome muito menos memória, pois não é realizado qualquer cálculo de gradientes ou atualização de pesos, já que o objetivo dela é somente a captura de dados da performance do modelo sobre dados externos ao treinamento. Ao final de cada época, caso o valor da perda no conjunto de validação seja menor que o da época anterior, os parâmetros do modelos são salvos para serem recuperados ao final de todas as épocas. Isso é importante, pois a tendência é que, ao passar de algumas épocas, o modelo comece a se ajustar demais aos dados de treinamento e perca a capacidade de generalização, como mostra o gráfico da figura 4.2. Ao salvar os parâmetros da época com menor perda de validação, garantimos portanto que teremos o modelo com a melhor performance ao final do treinamento.

4.1.7 Teste

Salvado e carregado o modelo com menor perda no conjunto de validação, chegou o momento de utilizar o conjunto de testes para avaliar a performance final do nosso modelo refinado. Para isso, criamos um *DataLoader* com amostragem sequencial para iterar uma vez sobre os lotes do *dataset* de testes e registrar as predições feitas pelo modelo. Assim, com as predições e os rótulos verdadeiros de cada classe, calculamos as seguintes métricas de avaliação: acurácia, *quadratic weighted kappa (QWK)*, root-mean-square error (RMSE) e discrepância horizontal.

A acurácia é a medida mais intuitiva de medida de performance, calculada pela razão entre o total de acertos das predições do modelo e o total de observações. Todavia, essa métrica é demasiadamente simples e não é muito informativa sobre como o modelo

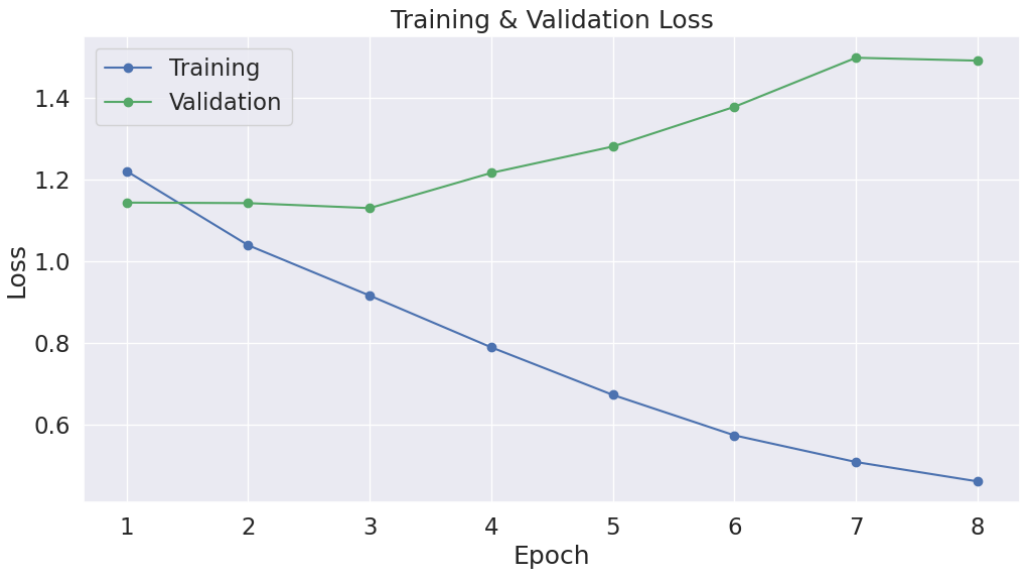


Figura 4.2: *Gráfico do comportamento da evolução da perda calculada durante o treino e a validação ao longo das épocas.*

está errando, o que demanda que calculemos também algumas outras métricas mais informativas.

O QWK (Cohen, 1968), por sua vez, é a métrica mais usada e considerada na área de ARR para a avaliação da performance de modelos, porém é um pouco mais difícil de ser interpretada. Basicamente ela mede a concordância entre o modelo e os dados reais, mas levando em consideração o quão dependente é a distribuição das predições do modelo em relação à verdadeira distribuição dos rótulos dos dados. Em outras palavras, essa medição tenta avaliar se as predições do modelo estão condicionadas a distribuição real dos dados, o que é especialmente relevante quando essa distribuição é desbalanceada. A tabela 4.1 mostra a interpretação dos valores do QWK proposta por Altman (1990).

valor do QWK	nível de concordância
<0.20	pobre
0.21 - 0.40	razoável
0.41 - 0.60	moderado
0.61 - 0.80	bom
0.81 - 1.00	muito bom

Tabela 4.1: *Níveis de concordância do QWK*

Já o RMSE é uma métrica que nos diz a distância média entre os valores preditos e os valores reais. Dessa forma, essa é uma métrica útil para avaliar não só a proporção de erros do modelo, mas também a dimensão desses erros. Um RMSE igual a 1, por exemplo, no caso do nosso modelo, indicaria que em média o modelo está errando as predições por uma classe (ex: a correta era 3 e a predição feita foi 4). Logo, quanto menor o RMSE calculado sobre as predições, melhor é o modelo.

Por fim, a discrepância horizontal (DH) é um índice que mede a taxa de erro do modelo em mais de duas classes. Essa métrica é especialmente útil na avaliação das competências da Redação do ENEM, pois, de acordo com a Cartilha do Participante, disponibilizada pelo INEP (Inep, 2023), considera-se que houve discrepância nas correções de dois avaliadores quando a nota total diferir em mais de 100 pontos ou a diferença da nota atribuída a qualquer um das competências for superior a 80, ou seja, divergir por dois níveis de avaliação.

4.2 Experimentos

Um dos principais objetivos desse trabalho, além de implementar e executar um fluxo de refinamento do BERTimbau na tarefa de avaliação automática de redações do ENEM, é explorar estratégias e abordagens que possam enriquecer esse processo de aprendizado e, com efeito, promover alguma melhora nos resultados obtidos pelo procedimento de refinamento padrão. Mais especificamente, no contexto de avaliação da competência III, buscou-se analisar como as informações disponibilizadas sobre a redação ou mesmo aspectos extraídos dos próprios textos podem ser usados para alimentar o modelo com mais dados relevantes auxiliem nas predições. A seguir, detalhamos quais foram as estratégias adotadas, como elas foram implementadas e por que ela foram escolhidas.

4.2.1 Modelo Base

Para que sejamos capazes de fazer uma avaliação comparativa das estratégias adotadas, é necessário antes definir uma base para a comparação, ou seja, um modelo que adote a abordagem padrão para esse tipo de tarefa e defina a chamada *baseline*, que consiste em um valor mínimo obtido para cada uma das métricas métricas, o qual buscaremos melhorar.

O modelo base definido segue exatamente a estrutura e os passos detalhados no capítulo 4. Em resumo, o modelo recebe os textos das redações, aplica a tokenização para gerar os *input_ids* e a *attention_mask* e processa-os através da camada do BERTimbau para gerar uma representação agregada dos textos. Essa representação então alimenta a cabeça de classificação - contendo uma camada de *dropout* e uma camada de transformação linear - que devolve os *logits*, dos quais podemos obter o rótulo da classe predita identificando o índice do maior elemento do tensor. O diagrama apresentado na figura 4.3 ilustra esse processo, mostrando as camadas da rede e as transformações aplicadas nos dados.

4.2.2 Primeira Estratégia: Concatenar o tema com a redação

A primeira estratégia implementada a partir do modelo base é bastante simples: alimentar o modelo não só com os textos da redações, mas também com os temas das redações, ou seja, o título da proposta de texto correspondente. Em um primeiro momento, essa ideia pode parecer trivial, porém surpreendentemente a grande maioria dos trabalhos já realizados que propõe a avaliação automática de redações do ENEM, desconsidera totalmente essa informação no treinamento dos modelos.

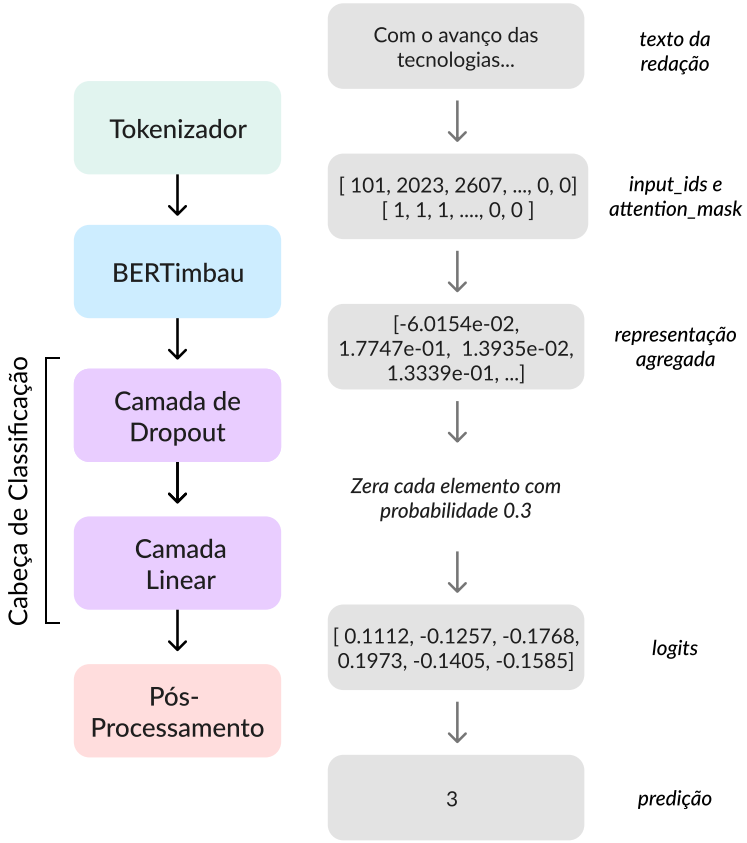


Figura 4.3: *Diagrama do processo de refinamento do modelo base.*

A ideia por trás dessa estratégia é de que a informação sobre o tema trabalhado no texto é fundamental para que o corretor possa fazer uma avaliação completa da competência III, haja vista que um dos aspectos considerados para definir o nível de avaliação é se as informações, fatos e opiniões apresentados estão relacionados com o tema, como foi mostrado na figura 3.10. Dessa forma, é esperado que a adição do tema nos dados de entrada contribua significativamente para a melhora dos resultados da abordagem base.

A implementação dessa estratégia é simples, pois basicamente só precisamos fazer modificações na etapa de pré-processamento do modelo base. Antes, dentre os atributos disponíveis no *dataset* usado, removíamos todas as colunas, mantendo somente o texto da redação e a nota da competência III. Agora, devemos manter também a coluna com o índice da proposta de texto usada e carregar o conjunto de dados que contém as propostas em um *dataframe* para que possamos extrair o título do tema correspondente. Em seguida, basta então concatenar os respectivos temas em todas as redações para que tenhamos uma única *string* a ser processada pelo tokenizador. Como mostra a figura 4.4, essa concatenação é realizada adicionando-se o *token* especial **[SEP]** entre os trechos, de forma a indicar ao modelo que o tema e a redação são sequências de texto distintas, cada uma com limites bem definidos.

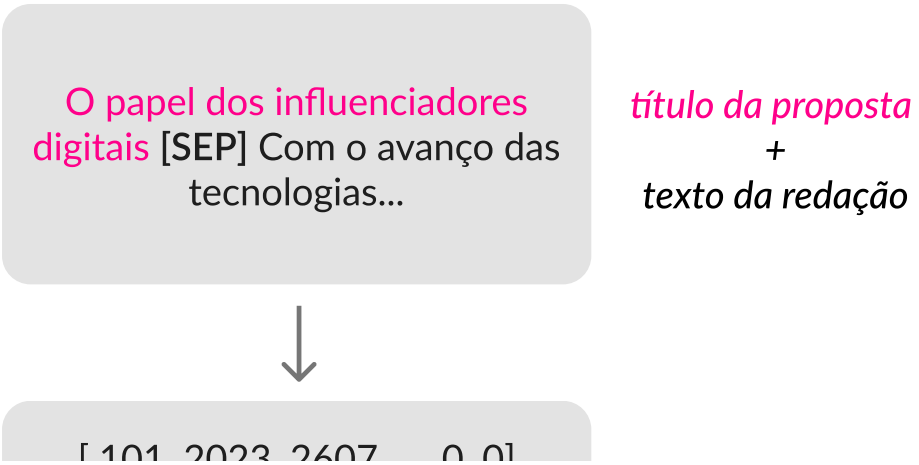


Figura 4.4: *Concatenação do título da proposta com a redação (primeira estratégia).*

4.2.3 Segunda Estratégia: Concatenar o tema e algumas características extraídas da redação com a redação

A segunda estratégia testada incrementa mais um pouco a primeira. Agora, além de concatenar o tema com a redação, concatenamos também algumas características extraídas do próprio texto da redação que podem ser consideradas relevantes na avaliação da competência III. A ideia por trás dessa estratégia é tentar retomar um pouco das abordagens clássicas da área de AAR, em que são aplicadas técnicas de engenharia de *features* para extrair características dos textos e usá-las para treinar modelos regressores e classificadores.

Outra inspiração para essa estratégia foi o trabalho de McCormick (2021). Nessa publicação, o autor apresenta um tutorial em que ele propõe a combinação de características categóricas e numéricas com o BERT para a classificação de avaliações de usuários em um site de compra de roupas. As avaliações estão todas em texto e as características disponibilizadas são o departamento da roupa, a divisão a que pertence o departamento, o tipo de roupa, a idade do comprador e a nota de 0 a 5. O objetivo do trabalho era usar o BERT para predizer, a partir do texto, se avaliação na plataforma dada pelo comprador tinha sido positiva ou negativa. Os resultados apresentados mostraram que a adição das características ao texto na sequência de entrada do BERT contribuiu para uma leve melhora de performance.

Contudo, há um grande desafio ao tentar aplicar esse tipo de estratégia para a competência escolhida nesse trabalho, visto que não há características claras e fáceis de extrair que possam ser ditas determinantes para avaliá-la, diferentemente de outras competências que possuem critérios mais objetivos. No caso da competência I, por exemplo, que avalia domínio da língua, é claro que a quantidade de erros gramaticais é determinante para definir a nota. Já no caso da IV, que avalia conhecimento de mecanismos linguísticos, a quantidade elementos de coesão presentes no texto (ex: contudo, portanto, ademais), é fundamental para determinar o nível de avaliação.

Como o projeto de texto e o desenvolvimento da argumentação avaliados na competência III não podem ser expressos em características desse tipo, decidimos extrair algumas

características que de alguma forma se relacionem com esses critérios e possam servir como heurísticas para o problema, tendo como base o Manual de Correção elaborado pelo INEP (Inep, 2019). As características extraídas foram:

1. **Quantidade de parágrafos:** Essa característica busca capturar indícios da existência de projeto de texto. Ainda que o foco maior da avaliação do projeto de texto seja o conteúdo e não a forma, um desenvolvimento argumentativo bem planejado e coeso passa necessariamente pela estruturação do texto em partes bem definidas e organizadas. Em geral, um texto com um bom projeto deve apresentar 4 parágrafos: um parágrafo de introdução, dois parágrafos de desenvolvimento e um de conclusão. Logo, pode-se considerar, por exemplo que um texto com apenas 2 parágrafos apresenta indícios de falha no projeto de texto.

2. **Diversidade lexical:** A diversidade lexical é um indicador de desenvolvimento linguístico associado à quantificação da variação de palavras empregadas no texto. Uma boa argumentação, em geral, se utiliza de um amplo repertório lexical que seja capaz de comunicar com clareza as ideias apresentadas. Nesse sentido, um texto com muita repetição de palavras e conhecimento raso da língua dificilmente será capaz de desenvolver uma argumentação produtiva e articulada.

3. **Quantidade de sentenças:** Além da organização dos parágrafos, o projeto de texto também pode ser evidenciado na estruturação interna desses parágrafos. Ainda que tenha 4 parágrafos bem definidos, um texto que apresente somente um período por parágrafo provavelmente não será capaz de desenvolver um fluxo de ideias coeso, claro e articulado, e são casos como esse que essa característica pode ajudar a capturar.

4. **Similaridade entre redação e textos motivadores:** Um dos aspectos de avaliação descritos nos níveis de desempenho da competência III é se as informações, fatos e opiniões apresentados pelo candidato estão limitados aos textos motivadores, caso em que a nota é limitada em no máximo 120 pontos. Sendo assim, partimos da hipótese de que, calculando a similaridade da redação com os textos motivadores, podemos ter algum indicativo desse tipo de ocorrência.

5. **Entidades nomeadas:** Entidades nomeadas, nesse contexto, são quaisquer citações no texto a localidades, pessoas, organizações, entidades e até eventos. Em Inep (2017), é pontuado que o participante pode lançar mão de alguns recursos, como o uso de informações estatísticas, exemplos, argumentos de autoridade, entre outros meios, a fim de que ele convença o leitor de que seu ponto de vista é pertinente. De modo geral, ao utilizar esses recursos, é frequente que apareçam no texto entidades nomeadas, sejam elas um autor, uma obra de ficção ou um órgão público, de modo que, capturando essas informações temos mais um indício da qualidade do desenvolvimento argumentativo.

Todas as características listadas foram extraídas utilizando a API do *spaCy* (apresentado na seção 3.5) e concatenadas da mesma forma que o título da proposta na primeira estratégia, como mostra a figura 4.5.

No caso da similiridade entre redação e textos motivadores, o *spaCy* compara as representações e devolve um valor em ponto flutuante entre 0 e 1. Como o BERT não

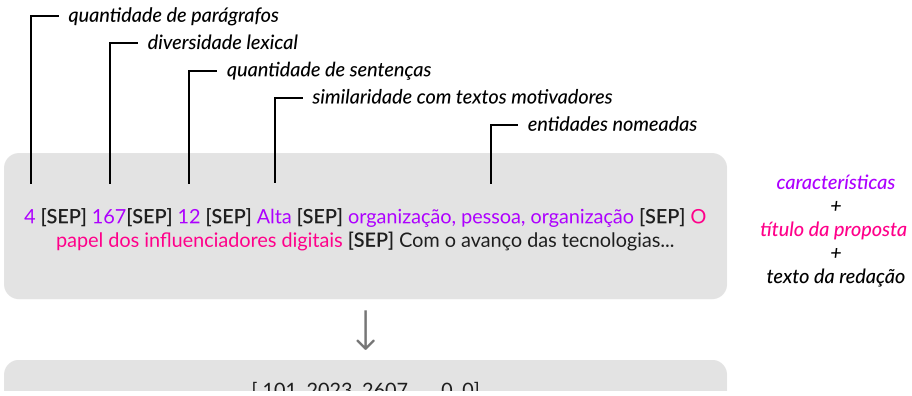


Figura 4.5: *Concatenação das característcas extraídas e do título da proposta com a redação (segunda estratégia).*

é capaz de interpretar valores decimais, as pontuações obtidas foram discretizadas em faixas de valores, definidas a partir de uma análise dos dados de treino, e classificadas em similaridade "Muito Alta", "Alta", "Média", "Baixa" e "Muito Baixa".

Já no caso da extração das entidades nomeadas, haja vista que a simples concatenação das entidades presentes no texto seria redundante, decidimos concatenar as categorias das entidades. O reconhecedor de entidades da *pipeline* para língua portuguesa do *spaCy* categoriza as entidades de acordo com os seguintes rótulos:

- LOC: Definido como rótulo de localidade, mas que verificou-se que identifica também entidades públicas e privadas. Rótulo tranformado para "entidade".
- MISC: Classifica categorias diversas, como eventos, nacionalidades, produtos ou obras de arte. Rótulo tranformado para "miscelânea".
- ORG: Semelhante à categoria LOC, identifica empresas, agências, instituições, etc. Rótulo tranformado para "organização".
- PER: Identifica nomes de pessoas ou famílias. Rótulo tranformado para "pessoa".

Já as características de quantidade de parágrafos, diversidade lexical e quantidade de sentenças foram mantidas como valores inteiros.

4.2.4 Terceira Estratégia: Concatenar o tema com a redação e unir a representação do BERT com um tensor de *features*

A terceira estratégia buscou, assim como a segunda, utilizar as características extraídas dos textos para alimentar o modelo com mais informações. Nesse experimento, o título da proposta foi concatenado com a redação assim como nos anteriores. Contudo, ao invés de incluir as características na entrada de texto, a ideia foi inseri-las em um tensor e concatená-lo ao tensor da representação agregada produzida pelo BERTimbau, como mostra a figura 4.6.

Nesse caso, para que seja possível incluir todas as características em um tensor, é necessário que todas elas estejam em formato numérico. No caso da quantidade de parágrafos, diversidade lexical e quantidade de sentenças nada muda em relação a segunda

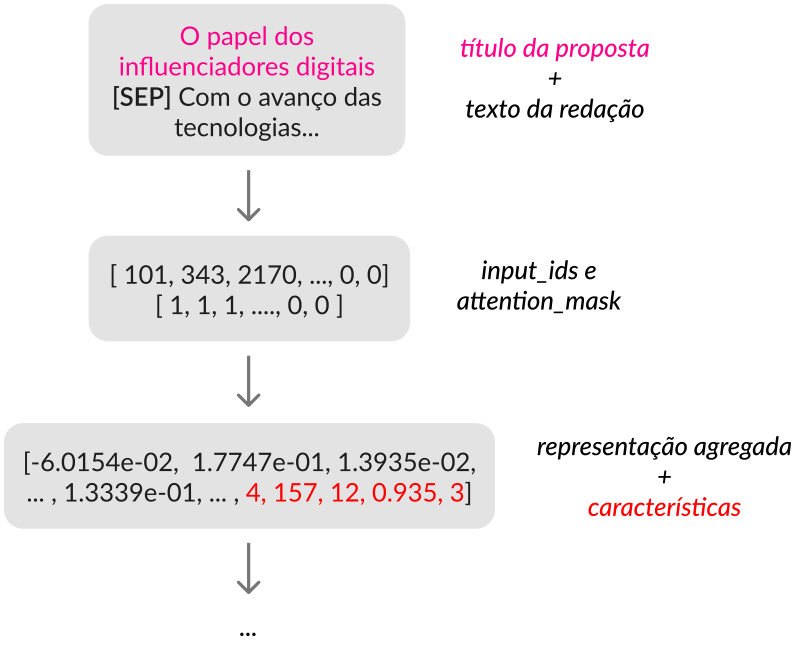


Figura 4.6: *Concatenação das características extraídas com a representação gerada pelo BERTImbau (terceira estratégia).*

estratégia, elas continua sendo representadas como valores inteiros. Já para a similaridade da redação com os textos motivadores, não é mais necessário discretizá-la e, portanto, podemos utilizar o valor em ponto flutuante. Finalmente, para as entidades nomeadas, o valor utilizado foi a quantidade de entidades mencionadas no texto, independentemente da categoria.

O programa 4.5 abaixo mostra a principal alteração necessária na implementação dessa estratégia. Além dos *input_ids* e da *attention_mask*, precisamos também passar o tensor de características para o método `forward` para que elas sejam concatenadas à representação gerada pela camada do BERTimbau antes de passar pela cabeça de classificação. Veja que, na linha 6, a camada linear recebe agora um tensor de tamanho 773 (768 elementos da representação do BERT + 5 elementos correspondentes às características).

Programa 4.5 Implementação da classe CustomModel na terceira estratégia.

```
1   class CustomModel(torch.nn.Module):
2     def __init__(self):
3        super(CustomModel, self).__init__()
4        self.l1 = transformers.BertModel.from_pretrained(checkpoint)
5        self.l2 = torch.nn.Dropout(0.3)
6        self.l3 = torch.nn.Linear(773, 6)
7
8     def forward(self, ids, attention_mask, features):
9        _, output_1 = self.l1(ids, attention_mask, return_dict=False)
10       output_2 = self.l2(torch.cat([output_1, features], dim=1))
11       output = self.l3(output_2)
12       return output
```

Capítulo 5

Resultados

Tendo em vista que no refinamento do modelo BERTimbau os parâmetros da cabeça de classificação são sempre inicializados de maneira aleatória e, portanto, os resultados finais obtidos no refinamento de uma instância do modelo podem variar a depender desses parâmetros iniciais, nos experimentos realizados foram refinadas 10 instâncias de modelos para cada uma das abordagens introduzidas no capítulo anterior. Ao final, foi possível então coletar dados de 40 modelos refinados e obter o desempenho médio de cada estratégia em relação às métricas definidas na seção 4.1.7.

As tabelas 5.1, 5.2, 5.3 e 5.4 contém os dados coletados dos modelos refinados com a abordagem base e com as estratégias E.1, E.2 e E.3, respectivamente. Foram registrados, para cada modelo, a melhor época de treinamento, o valor da perda de validação correspondente, além das métricas de acurácia, *quadratic weighted kappa* (QWK), *root-mean-score error* (RMSE) e discrepância horizontal (DH), calculadas sobre o conjunto de dados de teste, que tem tamanho igual a 987.

Modelo	Melhor época	Perda Validação	Acurácia	QWK	RMSE	DH
1	2	1.066081	0.598784	0.534555	0.763265	0.007092
2	1	1.129304	0.547112	0.445292	0.810268	0.012158
3	2	1.108072	0.583587	0.530191	0.820828	0.012158
4	2	1.084650	0.560284	0.545608	0.818356	0.011145
5	2	1.092180	0.580547	0.558157	0.778381	0.008105
6	2	1.051165	0.590679	0.551313	0.777730	0.008105
7	2	1.109238	0.569402	0.489470	0.825137	0.016211
8	2	1.073789	0.566363	0.557861	0.798301	0.007092
9	2	1.090437	0.562310	0.547229	0.797666	0.009119
10	2	1.094894	0.586626	0.518413	0.775120	0.008105

Tabela 5.1: *Dados coletados dos modelos com a abordagem base*

Modelo	Melhor época	Perda Validação	Acurácia	QWK	RMSE	DH
1	2	1.041205	0.608916	0.580892	0.754587	0.005066
2	3	1.082664	0.594732	0.551383	0.775120	0.009119
3	3	1.058412	0.581560	0.546199	0.793847	0.011145
4	3	1.046897	0.595745	0.581263	0.751223	0.005066
5	2	1.051583	0.589666	0.575752	0.779681	0.007092
6	2	0.980524	0.622087	0.582637	0.747844	0.008105
7	2	0.985846	0.632219	0.594687	0.738299	0.006079
8	2	1.017336	0.603850	0.559650	0.771188	0.009119
9	2	1.005075	0.618034	0.587541	0.743768	0.005066
10	2	0.991861	0.604863	0.574283	0.753916	0.006079

Tabela 5.2: *Dados coletados dos modelos com a estratégia 1*

Modelo	Melhor época	Perda Validação	Acurácia	QWK	RMSE	DH
1	1	1.070765	0.553191	0.556377	0.825751	0.007092
2	3	1.042659	0.590679	0.587598	0.770531	0.002026
3	2	1.028923	0.595745	0.604445	0.765915	0.004053
4	3	1.035665	0.605876	0.574269	0.758605	0.007092
5	3	1.058083	0.590679	0.552156	0.805251	0.011145
6	2	1.027226	0.604863	0.576646	0.776426	0.009119
7	2	1.033911	0.603850	0.539307	0.783570	0.009119
8	3	1.031944	0.606890	0.578350	0.769215	0.009119
9	2	1.004221	0.589666	0.559872	0.782923	0.007092
10	3	1.110993	0.582573	0.556778	0.795122	0.011145

Tabela 5.3: *Dados coletados dos modelos com a estratégia 2*

Modelo	Melhor época	Perda Validação	Acurácia	QWK	RMSE	DH
1	3	1.115078	0.584600	0.556780	0.795122	0.012158
2	2	1.053920	0.566363	0.467413	0.798935	0.008105
3	4	1.286429	0.587639	0.567957	0.765254	0.006079
4	3	1.038534	0.601824	0.564431	0.762601	0.009119
5	2	1.115110	0.607903	0.565647	0.757937	0.007092
6	2	1.280622	0.545086	0.409879	0.831254	0.007092
7	3	1.340020	0.624113	0.550536	0.769215	0.007092
8	2	1.196723	0.597771	0.568188	0.758605	0.007092
9	3	1.048505	0.598784	0.565514	0.760606	0.009119
10	3	1.217350	0.598784	0.594388	0.757268	0.007092

Tabela 5.4: *Dados coletados dos modelos com a estratégia 3*

Analisando as tabelas produzidas, um dos primeiros dados a se observar é a melhor época de treinamento dos modelos, selecionada pelo critério de menor perda calculada no *loop* de validação. Como esperado, em média, todos os modelos convergiram em por volta de 2 a 4 épocas, a partir de quando a perda de validação começou a aumentar com o ajuste excessivo dos parâmetros aos dados de treino. Contudo, é interessante notar que, à medida que fomos incrementando e modificando o modelo em cada estratégia e se distanciando mais da abordagem padrão, a convergência no refinamento, em média, demorou um pouco mais, como enfatizado abaixo:

1. **Abordagem base:** convergência média em **1,9** épocas
2. **Estratégia 1:** convergência média em **2,1** épocas
3. **Estratégia 2:** convergência média em **2,4** épocas
4. **Estratégia 3:** convergência média em **2,7** épocas

Quanto às métricas de acurácia, QWK, RMSE e DH, foi construída uma tabela com as médias dos valores calculados para os 10 modelos refinados de acordo com cada uma das quatro abordagens (Tabela 5.5) e esses mesmos valores foram também representados graficamente de modo a facilitar a análise comparativa das estratégias (Gráficos 5.1, 5.2, 5.3, 5.4).

Estratégia	Acurácia	QWK	RMSE	Discrepância horizontal
Base	57.4%	0.529	0.796	0.09%
E.1	60.5%	0.573	0.761	0.07%
E.2	58.9%	0.567	0.787	0.07%
E.3	59.1%	0.541	0.775	0.08%

Tabela 5.5: *Médias das métricas de desempenho de cada estratégia.*

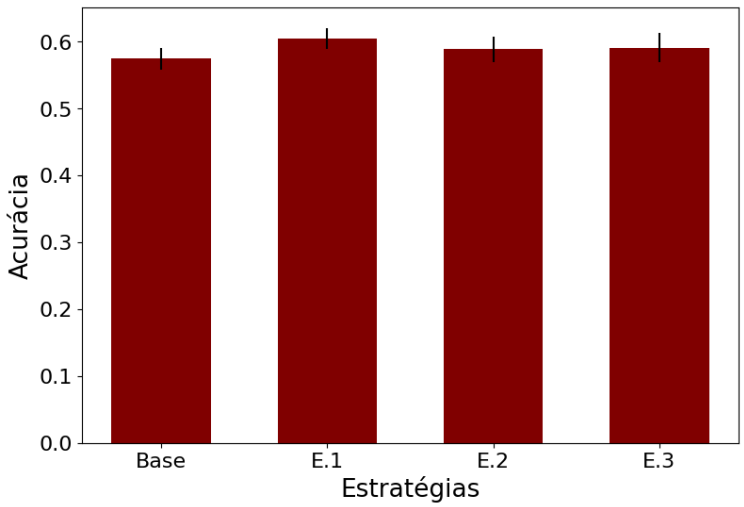


Figura 5.1: *Gráfico da acurácia por abordagem.*

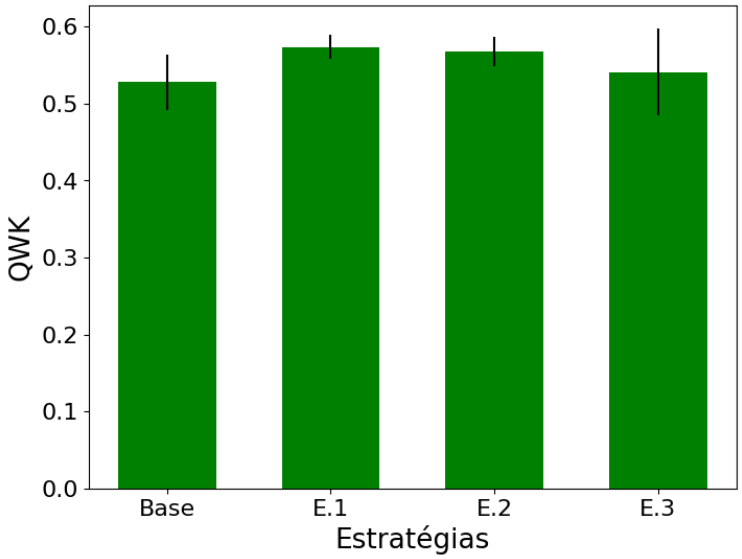


Figura 5.2: *Gráfico do QWK por abordagem.*

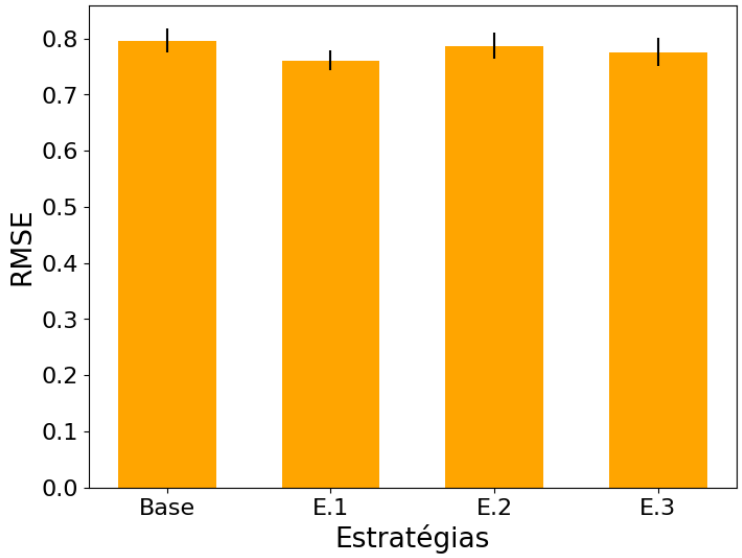


Figura 5.3: *Gráfico do RMSE por abordagem.*

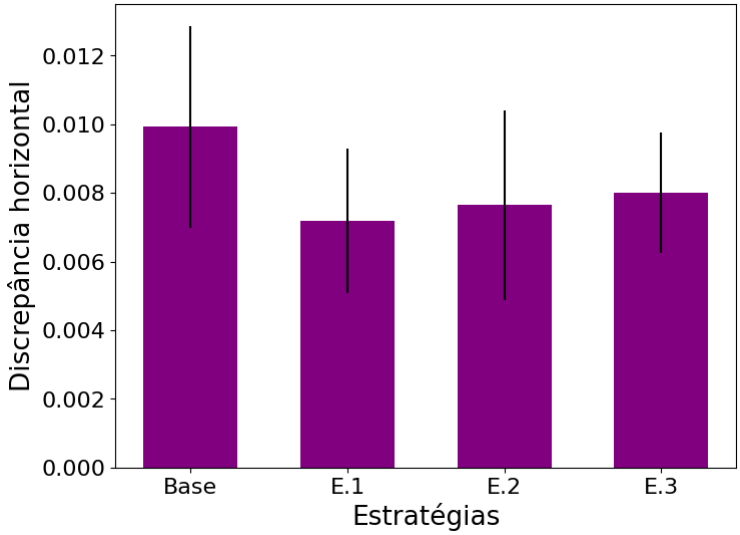


Figura 5.4: *Gráfico da discrepância horizontal por abordagem.*

Notamos que, de maneira geral, todos os modelos obtiveram uma acurácia próxima de 60%, um QWK moderado, um RMSE abaixo de 0.8 e uma discrepância horizontal inferior a 0.1%. Além disso, o desvio padrão verificado em todas as métricas, indicado pelas linhas nos limites superiores das barras dos gráficos, se mostrou consideravelmente baixo, à exceção da discrepância horizontal que apresentou um desvio proporcionalmente grande, mas bastante pequeno em escala, já que os valores de discrepância não passaram de 0.2%. Isso demonstra, portanto, uma certa consistência no processo de refinamento, mesmo com o fator de aleatoriedade na inicialização dos parâmetros da cabeça de classificação.

Analisando mais atentamente os dados, ainda que se observem resultados bastante próximos nas quatro abordagens, é possível notar uma pequena vantagem da estratégia 1 (E.1) em relação às demais, tendo essa obtido o melhor desempenho para todas as métricas avaliadas. Considerando especialmente o QWK, a principal medida de performance de modelos de AAR, a simples inclusão do título do tema nos dados de entrada do BERTimbau produziu um salto de um QWK de aproximadamente 0.53 para um QWK de quase 0.58, um ganho bastante relevante e que se aproxima de um nível de concordância considerado "bom" pela interpretação proposta por Altman (1990) que foi apresentada na tabela 4.1.

Nas demais estratégias (E.2 e E.3), convém salientar que, apesar de ambas terem obtido melhores resultados que a abordagem base em todas as métricas, se considerarmos que elas são incrementais em relação a E.1, ou seja, implementam a estratégia de E.1 (concatenar o tema com a redação) junto com outra potencial melhoria, não podemos dizer que elas foram bem sucedidas em suas propostas. Em outras palavras, a concatenação do tema com a redação foi positiva para a melhora das predições do modelo, mas a inclusão adicional de características extraídas do texto na sequência de entrada e na representação agregada devolvida pelo BERT, ao contrário do esperado, pioraram os resultados já atingidos pela estratégia 1.

No caso da piora registrada na estratégia 2, podemos interpretar como tendo havido uma maior dificuldade do BERTimbau em compreender os valores numéricos fornecidos, referentes à diversidade lexical, à quantidade de parágrafos e à quantidade de sentenças.

No trabalho de McCormick (2021), que serviu de inspiração para a implementação dessa estratégia, ele já sugeria que a inclusão de características no texto de entrada havia funcionado para o *dataset* em particular que ele utilizou, pois elas podiam ser facilmente entendidas como texto pelo BERT, como é o caso das informações sobre o departamento da roupa e da categoria a que ela pertence. Até mesmo no caso da avaliação numérica, este dado também é razoavelmente fácil de ser interpretado haja vista que só há cinco notas possíveis. Por outro lado, as características utilizadas nesse trabalho, tanto no caso das textuais, como as categoriais de entidades nomeadas, quanto no caso das numéricas, como a diversidade lexical, eram nitidamente mais difíceis de serem interpretadas.

Já no caso da estratégia 3, a piora da performance pode ter sido consequência da alteração mais drástica implementada no processo de refinamento padrão, já que este não prevê modificações diretas na representação gerada como foi feito aqui. Ainda sim, isso não exclui a possibilidade de que a realização de operações com os *embeddings* antes da entrada na camada de classificação possa beneficiar o procedimento de refinamento.

Por fim, para verificar o comportamento das predições de forma mais detalhada, foram construídas matrizes de confusão a partir das predições de um dos modelos de cada abordagem (Figura 5.5). O modelo escolhido para representar cada uma foi o que obteve um valor de QWK mais próximo da média de todas as 10 instâncias refinadas para aquela abordagem.

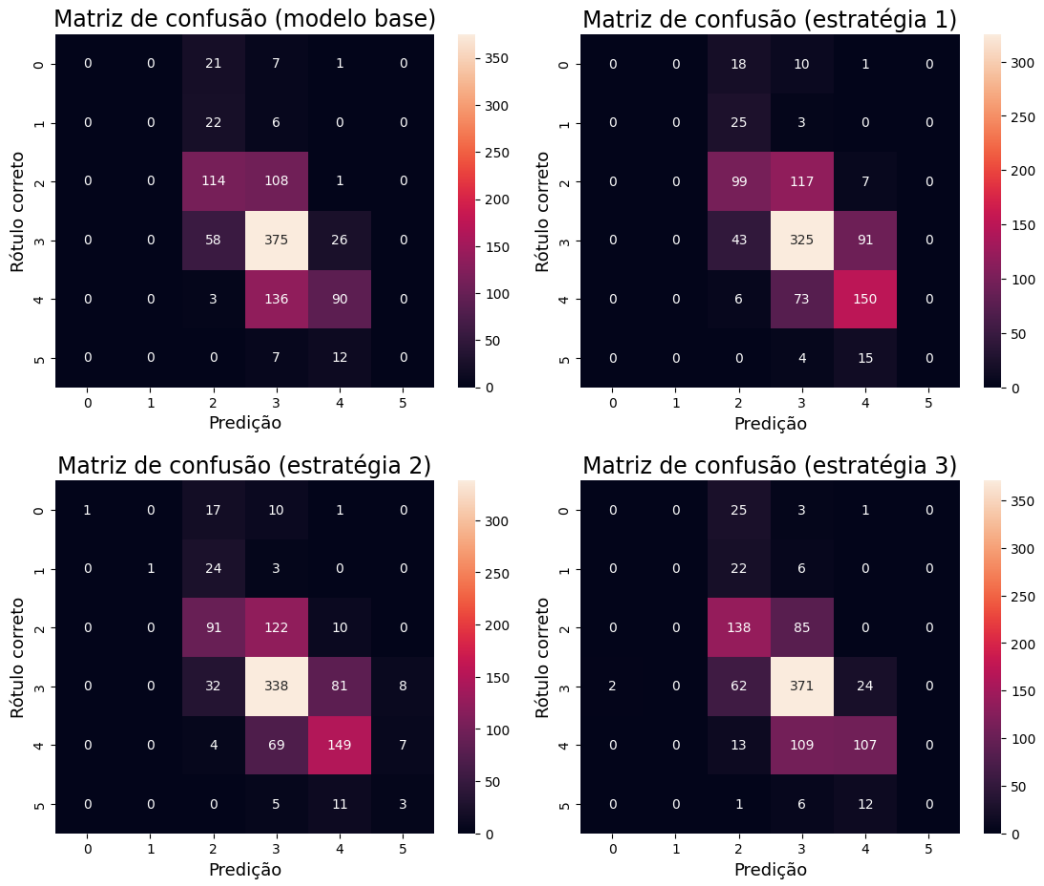


Figura 5.5: *Matrizes de confusão das quatro abordagens.*

Vemos que todas as matrizes construídas são bastante parecidas e bastante dependentes da distribuição desbalanceada das notas nos *dataset*, como mostrado na seção 3.4. A maioria das predições está concentrada na faixa de 80 a 160 pontos, correspondentes aos níveis de avaliação 2 a 4. Entretanto, há um detalhe sutil que pode ser percebido na matriz da estratégia 2. Essa estratégia foi a única que que foi capaz de predizer corretamente a nota de redações avaliadas nos dois níveis mais extremos e menos frequentes do *dataset* de treino. As demais estratégias sequer tiveram alguma predição, nem mesmo errada, para esses níveis de desempenho.

Capítulo 6

Conclusão

Esse trabalho de conclusão de curso, de maneira geral, teve como objetivo estudar a área de processamento de linguagem natural, o funcionamento redes neurais e o uso de modelos baseados em *Transformers* para classificação de textos. Por conseguinte, buscamos apresentar os conhecimentos adquiridos a outras pessoas interessadas por meio da produção de imagens, diagramas e gráficos que possam auxiliar na compreensão de mecanismos complexos como a auto-atenção de múltiplas cabeças e de processos sofisticados como o treinamento de modelos de linguagem bidirecionais.

Além disso, no campo prático, definiu-se como foco específico desse trabalho a tarefa de avaliação automática de redações. Por muitos anos, essa área foi notadamente dominada pelo uso de modelos que implementam técnicas de engenharia de *features* e métodos de regressão. Contudo, dado que nos últimos anos, modelos de linguagem como o BERT vêm dominando a área de processamento de linguagem natural, propusemos nesse trabalho o refinamento do BERTimbau, um modelo baseado na arquitetura BERT e pré-treinado com textos em português, para realizar a correção de redações do ENEM ou, mais especificamente, da competência III de avaliação da redação.

Os experimentos realizados no desenvolvimento do trabalho visaram testar abordagens híbridas, que se beneficiassem tanto da capacidade elevada de aprendizado contextual do BERTimbau quanto do uso de técnicas de engenharia de características que pudessem auxiliar no processo de refinamento do modelo com a inclusão de mais informações relevantes sobre os textos. Foram definidas três estratégias nesse sentido a partir do procedimento de refinamento padrão do BERTimbau. Na primeira, o tema da redação foi incluído junto com a redação na entrada do modelo. Na segunda, além do tema, características extraídas da redação foram também incluídas como texto na sequência de entrada. Na terceira e última, as características extraídas foram transformadas em valores numéricos, inseridas em um tensor e concatenadas à representação gerada pelo BERTimbau antes da entrada na cabeça de classificação.

Os resultados finais revelaram valores para as métricas de acurácia, *quadratic weighted kappa* (QWK), *root-mean-score error* (RMSE) e discrepância horizontal (DH) bem próximos em todas as abordagens. Um dos principais problemas identificados na análise desses resultados foi o desbalanceamento do *dataset* utilizado, o qual gerou previsões bastante

condicionadas às classes mais frequentes do conjunto de dados de treinamento.

Ainda que isso tenha ocorrido, pudemos notar que a estratégia 1 conseguiu se destacar sobre as outras, obtendo os melhores desempenhos em todas as métricas. A simples concatenação do título do tema à redação nessa estratégia foi capaz de proporcionar um salto do QWK de 0.53, na abordagem padrão, para 0.57, nessa abordagem, confirmando a relevância do tema para a avaliação da competência III.

Por outro lado, as estratégias 2 e 3, que usaram as características das redações para alimentar o modelo com mais dados além do título do tema, não conseguiram melhorar os resultados já obtidos pela estratégia 1. Entretanto, na análise das matrizes de confusão dos modelos implementados para cada estratégia, verificou-se que somente a estratégia 2 foi capaz de predizer notas nos níveis de desempenho mais extremos e menos frequentes no conjunto de dados, o que mostra que há algum ganho possível na inclusão de características junto à redação na entrada do BERTimbau.

Nesse sentido, entendemos que esse trabalho se trata apenas de um estudo exploratório preliminar de algumas abordagens que podem ser usadas, em conjunto com modelos baseados em BERT, para a avaliação automática de redações. Sendo assim, propomos que trabalhos futuros realizem estudos mais aprofundados sobre o uso desse tipo de estratégia híbrida, de modo a entender com maior clareza qual a melhor maneira de selecionar características escolhidas, qual é a importância que cada uma delas exerce nas predições e qual é a melhor forma de incluí-las no procedimento de refinamento. Além disso, é interessante que futuramente seja explorada também a aplicação das estratégias apresentadas para a avaliação das demais competências que ficaram fora do escopo desse trabalho.

Esperamos que os resultados desse trabalho de conclusão de curso possam auxiliar o desenvolvimento de novos sistemas de avaliação de automática de redações e promover mais pesquisas que se utilizem de modelos baseados em *Transformers* nesse contexto de aplicação.

Referências

[Altman 1990] Douglas G. Altman. "Practical statistics for medical research". In: 1990. URL: https://api.semanticscholar.org/CorpusID:62683118 (citado nas pgs. 29, 41).

[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, 2017, pp. 94–102. URL: https://aclanthology.org/E17-4010 (citado nas pgs. 3, 4).

[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. 29).

[Devlin *et al.* 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee e Kristina Toutanova. "Bert: pre-training of deep bidirectional transformers for language understanding". In: *North American Chapter of the Association for Computational Linguistics*. 2019. URL: https://api.semanticscholar.org/CorpusID:52967399 (citado nas pgs. 1, 5, 13, 15, 23).

[Fonseca *et al.* 2018] Erick Rocha Fonseca, Ivo Medeiros, Dayse Kamikawachi e Alessandro Bokan. "Automatically grading brazilian student essays". In: *International Conference on Computational Processing of the Portuguese Language*. 2018. URL: https://api.semanticscholar.org/CorpusID:52276565 (citado nas pgs. 3, 4).

[Hochreiter e Bengio 2001] Sepp Hochreiter e Yoshua Bengio. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In: 2001. URL: https://api.semanticscholar.org/CorpusID:17278462 (citado na pg. 12).

[Honnibal *et al.* 2020] Matthew Honnibal, Ines Montani, Sofie Van Landeghem e Adriane Boyd. "spaCy: Industrial-strength Natural Language Processing in Python" (2020). DOI: 10.5281/zenodo.1212303 (citado na pg. 20).

[Inep 2017] INEP. *Textos Dissertativo-Argumentativos : Subsídios para qualificação de avaliadores*. 2017. URL: https://www.gov.br/inep/pt-br/centrais-de-conteudo/acervo-linha-editorial/publicacoes-institucionais/avaliacoes-e-exames-da-educacao-basica/textos-dissertativo-argumentativos-subsidios-para-qualificacao-de-avaliadores (citado na pg. 33).

[Inep 2019] INEP. *Manual de correção da redação – Competência 3*. 2019. URL: https://download.inep.gov.br/educacao_basica/enem/downloads/2020/Competencia_3.pdf (citado na pg. 33).

[Inep 2023] INEP. *A Redação do Enem 2023: cartilha do participante*. 2023. URL: https://www.gov.br/inep/pt-br/centrais-de-conteudo/acervo-linha-editorial/publicacoes-institucionais/avaliacoes-e-exames-da-educacao-basica/a-redacao-no-enem-2023-cartilha-do-participante (citado nas pgs. 16–18, 30).

[Junior *et al.* 2017] Celso Junior, Marcos Spalenza e Elias Oliveira. "Proposta de um sistema de avaliação automática de redações do enem utilizando técnicas de aprendizagem de máquina e processamento de linguagem natural". In: mai. de 2017. DOI: 10.14210/cotb.v0n0.p474-483 (citado na pg. 3).

[Júnior 2020] José Adenaldo Santos Bittencourt Júnior. "Avaliação automática de redação em língua portuguesa empregando redes neurais profundas". In: fev. de 2020. URL: http://repositorio.bc.ufg.br/tede/handle/tede/10411 (citado na pg. 3).

[Ke e Ng 2019] Zixuan Ke e Vincent Ng. "Automated essay scoring: a survey of the state of the art". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2019, pp. 6300–6308 (citado na pg. 22).

[Loshchilov e Hutter 2019] Ilya Loshchilov e Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG] (citado na pg. 26).

[Marinho, Anchiêta *et al.* 2021] Jeziel Marinho, Rafael Anchiêta e Raimundo Moura. "Essay-br: a brazilian corpus of essays". In: out. de 2021, pp. 53–64. DOI: 10.5753/dsw.2021.17414 (citado nas pgs. 2, 4, 17).

[Marinho, Cordeiro *et al.* 2022] Jeziel Marinho, Fábio Cordeiro, Rafael Anchiêta e Raimundo Moura. "Automated essay scoring: an approach based on enem competencies". In: nov. de 2022, pp. 49–60. DOI: 10.5753/eniac.2022.227202 (citado nas pgs. 2, 4).

[McCormick 2021] Chris McCormick. *Combining Categorical and Numerical Features with Text in BERT*. 2021. URL: https://mccormickml.com/2021/06/29/combining-categorical-numerical-features-with-bert/#24-bert-on-review-text-only (citado nas pgs. 32, 42).

[Page 1966] Ellis B. Page. "The imminence of... grading essays by computer". *The Phi Delta Kappan* 47.5 (1966), pp. 238–243. ISSN: 00317217. URL: http://www.jstor.org/stable/20371545 (citado na pg. 3).

[Pedregosa *et al.* 2011] F. Pedregosa *et al.* "Scikit-learn: machine learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (citado na pg. 21).

[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. URL: https://api.semanticscholar.org/CorpusID:205001834 (citado na pg. 26).

[Souza *et al.* 2020] Fábio Souza, Rodrigo Nogueira e Roberto Lotufo. "Bertimbau: pretrained bert models for brazilian portuguese". In: *Intelligent Systems*. Ed. por Ricardo Cerri e Ronaldo C. Prati. Cham: Springer International Publishing, 2020, pp. 403–417 (citado nas pgs. 2, 16).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". In: *Neural Information Processing Systems*. 2017. URL: https://api.semanticscholar.org/CorpusID:13756489 (citado na pg. 1).

[Weizenbaum 1966] Joseph Weizenbaum. "Eliza—a computer program for the study of natural language communication between man and machine". *Communications of the ACM* 9 (1966), pp. 36–45. URL: https://api.semanticscholar.org/CorpusID:1896290 (citado na pg. 1).

[Zhu *et al.* 2015] Yukun Zhu *et al.* "Aligning books and movies: towards story-like visual explanations by watching movies and reading books". In: *The IEEE International Conference on Computer Vision (ICCV)*. Dez. de 2015 (citado na pg. 15).