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

Desenvolvimento de um sistema NER em textos jurídicos Brasileiros utilizando BERTimbau

Caio Túlio de Deus Andrade

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Marcelo Finger

São Paulo
25 de Fevereiro de 2022

Agradecimentos

Nas profundezas do inverno, finalmente aprendi
que no meu interior vive um verão invencível.
— Albert Camus

Primeiramente, agradeço à minha família. Ao meu pai, Lídio Andrade, por sempre me incentivar nos estudos, me acudir nos momentos de crise, me apoiar em momentos de mudança e fomentar em mim um espírito de curiosidade desde criança. Tudo que consegui até hoje se deve a ele. Agradeço à minha mãe, Valéria Carvalho, cujo amor me acompanhou em todo momento de desafio, tristeza, e alegria. Mesmo não estando mais presente em vida, carrego seu amor em mim para o resto da vida. Agradeço também à minha irmã, Pâmela Carvalho, por ser a minha amiga mais próxima e me dar conselhos e ensinamentos que certamente me tornaram uma pessoa melhor.

Agradeço aos professores que pavimentaram cada centímetro da minha jornada acadêmica até aqui. Em especial, agradeço o professor Dr. Marcelo Finger pela orientação e apoio neste trabalho.

Agradeço aos amigos de faculdade, tão importantes durante os meus quatro anos de graduação. Sentirei tremenda falta das discussões sobre qual bandejão escolher para o almoço, as horas desesperadas de estudos pré provas e as escapadas das aulas para a cantina tomar um café e jogar conversa fora. Agradeço em especial ao Leandro Rodrigues e Caio Fontes, pelas caminhadas do Portão 1 da USP até o IME, conversas sobre música e apoio durante momentos difíceis que vivi este ano. Espero levar todos eles comigo pelo resto da vida.

Por fim, agradeço à Priscila Cravo Vianna, minha psicóloga, que me atendeu durante a elaboração desse trabalho. A minha boa saúde mental certamente se deve a ela e eu não conseguiria escrever essas palavras de fechamento sem o apoio dela. Meus mais sinceros agradecimentos.

Resumo

Caio Túlio de Deus Andrade. **Desenvolvimento de um sistema NER em textos jurídicos Brasileiros utilizando BERTimbau**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

REN é uma subárea de Processamento de Linguagem Natural que consiste de classificar palavras ou frases em entidades pré determinadas. Neste trabalho, estudamos os desafios em realizar o refinamento de um modelo BERTimbau para executar essa tarefa. Utilizamos como objeto de estudo o *dataset* LeNERBr. Analisamos a distribuição de etiquetas no conjunto de dados e constatamos desequilibrio de etiquetas, um problema comum em REN. A performance de refinar um modelo nesse *dataset* foi analisada em 1 época e 8 épocas, onde constatamos *overfitting* na classe desbalanceada e saturação de desempenho. Também comparamos a performance de diferentes modelos BERT, em português e inglês, e mostramos como todos eles apresentaram performance similar, sugerindo que o texto jurídico é suficientemente diferente linguisticamente tanto de português quanto de inglês. Propomos como possível solução a esse problema a inclusão de palavras específicas no vocabulário do tokenizador, ou o pré treino de um modelo BERT em texto jurídico. Durante o trabalho também foi desenvolvido um canal de refinamento que esconde as dificuldades de treino expostas nesse trabalho. O canal de refinamento é agnóstico o bastante a conjuntos de dados de maneira que possa ser adaptado a outros conjuntos.

Palavras-chave: Processamento de Linguagem Natural. Reconhecimento de Entidades Nomeadas. Aprendizado Profundo. Aprendizado de Máquina. Transferência de Aprendizado. Modelos baseados em Transformadores. Textos Jurídicos. Processamento de Linguagem Natural. Reconhecimento de Entidades Nomeadas. Aprendizado Profundo. Aprendizado de Máquina. Transferência de Aprendizado. Modelos baseados em Transformadores. Textos Jurídicos.

Abstract

Caio Túlio de Deus Andrade. **Development of a NER model in brazilian legal text using BERTimbau**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

NER is a Natural Language Processing subfield which consists of classifying words or phrases into pre-defined entities. In this capstone project, we have studied the challenges in refining BERTimbau in this downstream task, specifically using Brazilian legal text data. We have used as an object of study the dataset LeNERBr. We have analyzed the label distribution in the dataset and have found that it's unbalanced, as is common in NER datasets. The models performance during refinement was measured throughout 1 epoch and 8 epochs, during which we found that the model overfits around the unbalanced label and its performance plateaus. We have also compared different model checkpoints, in Portuguese and English, and have shown that all of them present similar performance after 1 epoch of training. This finding suggests that Brazilian legal text is linguistically different enough from both english and portuguese so that both classes of models perform almost with the same performance. We propose, as a solution to this problem, the inclusion of domain specific words in the tokenizers vocabulary, or to pre-train the model using Brazilian Legal text. We also have developed a refinement pipeline that hides away some of the complexities of dealing with REN and BERT. The pipeline is agnostic enough to dataset so that it can be reused in the future in other projects with different datasets.

Keywords: Natural Language Processing. Named Entity Recognition. Deep Learning. Machine Learning. Transfer Learning. Transformer based models. Legal text. Natural Language Processing. Named Entity Recognition. Deep Learning. Machine Learning. Transfer Learning. Transformer based models. Legal text.

Lista de Abreviaturas

API	*Aplication programming interface* (Interface de programação de interfaces)
BERT	*Bidirectional Encoder Representation from Transformers*
LSTM	Memória de Curto-longo prazo (*Long Short Term Memory*)
BiLSTM	Memória bidirecional de Curto-longo prazo (*Bidirectional Long Short Term Memory*)
RNN	Rede Neural Recorrente (*Recurrent Neural Networks*)
FFNN	Rede Neural *Feed Forward*(*Feed Forward Neural Network*)
LM	Modelo de linguagem (*Language Model*)
OOV	Fora de vocabulário (*Out Of Vocabulary*)
GPT	Transformador Generativo pré-treinado (*Generative Pre-trained Transformer*)
IOB	Dentro-fora-começo (*Inside-Outside-Beggining*)
HAREM	HAREM: uma Avaliação de Reconhecimento de Entidades Mencionadas
CBOW	*Continuous Bag of Words*
SGD	*Stochastic Gradient Descent* (Gradiente descendente estocástico)
ADAM	*Adaptative Moment estimation* (Estimação de momento adaptativa)
i.i.d	Independentes e identicamente distribuídos
REN	Reconhecimento de entidades nomeadas
NMT	Traduçao de maquina neural (*Neural Machine Translation*)
GPU	Unidade de processamento gráfica (*Graphics processing unit*)
TPU	Unidade de processamento de tensores (*Tensors Processing Unit*)
ML	Aprendizado de Máquina (*Machine Learning*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo
PUC	Pontífice Universidade Católica

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

O campo de Processamento de Linguagem Natural (PLN) tem evoluído rapidamente desde 2017. Com a introdução das arquiteturas neurais baseadas em transformadores (Vaswani *et al.*, 2017), uma verdadeira revolução ocorreu na área com o nascimento de modelos como GPT-1 (Radford e Sutskever, 2018) e BERT (Devlin *et al.*, 2019), substituindo parcialmente arquiteturas tradicionais para resolução de problemas de linguagens como as LSTMs (Hochreiter e Schmidhuber, 1997). O BERT teve como principais contribuições a introdução de um modelo de linguagem verdadeiramente bidirecional, e um modelo pré treinado em enormes volumes de dados, introduzindo um modelo de linguagem potente que atinge performance de estado da arte em 11 tarefas clássicas em PLN, dentre elas, a de reconhecimento de entidades nomeadas (REN).

O processo de treinamento do BERT é divido em duas etapas: pré-treino e refinamento. O pré treino é realizado utilizando tarefas genéricas de modelagem de linguagem em grandes volumes de textos extraídos da Wikipedia em inglês e do *BookCorpus* (Zhu *et al.*, 2015).O processo de pré treino é extremamente custoso, e, segundo os autores do modelo, só deve ser repetido quando há interesse em extrair conhecimento linguístico de textos muito específicos - comumente referidos como textos de cunho ou domínio especialista - , como textos científicos (Beltagy *et al.*, 2019); O refinamento, por sua vez, é uma etapa adicional de treino na qual o modelo é treinado em uma tarefa específica de PLN (como análise de sentimento de texto, REN, similaridade de textos, dentre outras), adicionando uma camada extra ao modelo. O conhecimento adquirido no pré treino é carregado ao refinamento e garante um ganho de performance se comparado ao treino partindo de um modelo que não foi pré treinado, sendo assim um modelo que se caracteriza como um caso de uso de Transferência de Aprendizado (*Transfer Learning*).

No fim de 2020 foi publicado o modelo BERTimbau: um modelo baseado em BERT que, ao contrário de seu predecessor, foi pré treinado utilizando textos em português. Os autores em Souza *et al.*, 2020 mostraram que o modelo treinado por eles conseguiu uma performance mais significativa se comparado ao BERT original em tarefas clássicas de PLN utilizando textos em português, corroborando assim que a natureza do texto de pré treino tem impacto significativo na performance do modelo.

Neste trabalho utilizamos um modelo baseado em BERTimbau para refinamento na

tarefa de REN especificamente em textos de natureza jurídica em português.

1.1 Contextualização

A tarefa de Reconhecimento de Entidades Nomeadas consiste em classificar palavras ou frases no texto em categorias pré determinadas, levando em conta o seu contexto. Um sistema como este é particularmente útil em cenários nos quais há o interesse em otimizar e automatizar a categorização de trechos ou de um documento como um todo de acordo com a ocorrência de certas entidades. Um exemplo de uso para REN é a classificação automática de textos jurídicos de acordo com a legislação utilizada no texto, permitindo agrupamento de documentos diversos. As entidades utilizadas são particulares de cada projeto e pertinentes ao objetivo do classificador. Geralmente, utiliza-se o padrão IOB, que indica se um token está dentro, fora, ou é o início de uma entidade (Ramshaw e Marcus, 1995)

Um Modelo de REN, fundamentalmente, deve ser capaz de identificar corretamente em que posição do texto uma entidade nomeada se inicia, aonde ela termina e, naturalmente, quais palavras (ou *tokens*) estão contidas nela. Assim, o modelo deve ser capaz de compreender como um texto se estrutura para executar sua classificação de maneira eficaz. A essa tarefa de compreensão da estrutura do texto se dá o nome de modelagem de linguagem, que nada mais é do que a formalização de um aspecto linguístico do texto: Bengio *et al.*, 2003 define um modelo de linguagem como uma aproximação de uma distribuição de probabilidade que descreve a ocorrência de uma sequência de palavras ou categorias que pertencem a uma linguagem natural. Uma linguagem natural é uma forma de comunicação que surgiu de maneira espontânea para a comunicação de humanos por meio de repetição de sons e símbolos.

Um modelo de REN com boa modelagem de linguagem é capaz de discernir ambiguidades de uso de palavras ou frases de acordo com o contexto: os *tokens* "São"e "Paulo", mesmo tendo o mesmo significado morfológico nestes usos, podem representar, quando ocorrendo contiguamente, uma cidade, um estado ou um time de Futebol. O contexto é fundamental para entender-se qual a intenção do uso de cada *token*.

Em REN, a manutenção e criação de conjuntos de dados é um processo custoso. Por se tratar de uma tarefa de aprendizado supervisionado, amostras de textos precisam ser etiquetados manualmente por humanos. Quando o domínio de aplicação do modelo é muito específico, o processo torna-se ainda mais custoso levando, assim, a um baixo número de conjunto de dados especialistas. Em Português, destacam-se os *datasets* HAREM (Freitas *et al.*, 2010) a Paramopama (Junior *et al.*, 2015), que são textos anotados em Português com propósito geral.

Modelos baseados em Aprendizado de Máquina tem obtido performances de estado da arte em tarefas de REN (Lample *et al.*, 2016). Uma arquitetura extremamente popular no campo é a de Redes Neurais BiLSTMs que são apropriadas para processamento de texto devido à sua natureza intrinsecamente sequencial. Essas redes são, na verdade, a concatenação de duas redes LSTMs: uma que processa o texto da esquerda para a direita, e outra que processa o texto da direita para a esquerda, criando assim uma visão rasa de bidirecionalidade em representação de texto. Um outro problema das BiLSTMs

é a dificuldade em criar dependências longas entre trechos do texto. As sessões 2.3.2 e 2.3.3 discorrem sobre problemas dessa arquitetura e limitações da sua modelagem de linguagem.

Em 2018, foi introduzido um modelo baseado em biLSTMs que atingiu o estado da arte em REN aplicadas a textos jurídicos em Português. Os autores, além do modelo, publicaram o LeNER-BR, um *dataset* de textos jurídicos em português anotados (Luz de Araujo *et al.*, 2018). Com ele, os autores atingiram performance de estado da arte em REN em textos jurídicos em Português.

Esta solução foi proposta durante um período de transformação no campo de PLN, protagonizada pela introdução do modelo BERT, lançado no mesmo ano. A introdução deste modelo representou uma mudança de paradigma por introduzir um modelo verdadeiramente bidirecional, altamente paralelizável, e com uma modelagem de linguagem de estado da arte. Um outro avanço foi o uso de técnicas de *transfer learning*: o modelo é pré treinado em um grande volume de dados e carrega esse conhecimento para a tarefa de refinamento, relaxando, assim, a hipótese fundamental de independência e distribuição idêntica(i.i.d) de dados de treino, validação e, nesse caso, refinamento (Tan *et al.*, 2018). Particularmente, o BERT é pré treinado em textos em inglês, o que, naturalmente, limitava o potencial que um modelo refinado em textos em português poderia atingir.

Ao fim de 2020 essa barreira é transposta com o lançamento do modelo BERTimbau: um modelo baseado em BERT mas pré treinado totalmente em textos em Português. A introdução deste modelo trouxe novas perspectivas para pesquisa e aplicação de modelos profundos de PLN em português, possibilitando uma modelagem mais sofisticada e apropriada de tarefas em português.

1.2 Motivação

Em seu livro de 1996, 'Estupro: Crime ou "Cortesia"?' (Pimentel *et al.*, 1998) A Professora Sylvia Pimentel da Faculdade de direito da PUC-SP traz à tona uma série de fatores que conectam os processos judiciais de violência contra a mulher a estereótipos de gênero identificados nas falas dos entes jurídicos participantes das decisões do processo judicial. Partindo de uma pesquisa de cunho sócio-jurídico e analisando processos judiciais em 5 regiões do país, as autoras concluíram que vieses e preconceitos dos entes jurídicos são frequentemente externalizados textualmente nos relatos dos processos, correlacionando, assim, decisões contra a vítima de acordo com sua etnia, identidade de gênero e orientação sexual com valores preconceituosos de juízes, advogados e policiais envolvidos.

Em 2019, nasce o projeto "Crime de Estupro No Sistema de Justiça Brasileiro - Abordagem sociojurídica de gênero", liderado novamente pela professora Pymentel. O projeto se propõe a retomar e atualizar as pesquisas de 96, desde uma perspectiva renovada, considerando as significativas mudanças legislativas e teóricas que ocorreram nas duas últimas décadas, no Brasil e no mundo para questionar o quanto elas foram, ou não, acompanhadas por mudanças jurisprudenciais significativas, mudanças de mentalidade e percepção dos operadores de direito. Ademais, o projeto de 2019 visa também trazer uma abordagem quantitativa, além da qualitativa, da análise dos processos judiciais. Analisando o avanço computacional dos 20 anos entre as duas fases da pesquisa e, em específico, o

desenvolvimento na área de PLN, naturalmente surgiu o interesse em aplicar técnicas de REN com Aprendizado de Máquina para a criação de um modelo responsável por automaticamente reconhecer e marcar entidades de interesse no texto jurídico, representando as frases preconceituosas nos processos judiciais.

Detecção de discurso de ódio em textos utilizando ferramentas modernas de PLN é uma área de grande interesse e tem rendido resultados interessantes (Caselli *et al.*, 2021). No entanto, não há um trabalho com esse cunho, mas voltado a textos jurídicos e em português, principalmente pela falta de dados: muitos desses processos judiciais correm sob segredo judicial e só podem ser liberados mediante um pedido especial por meio de convênios entre pesquisadores e Tribunais de Justiça. Essa liberação é um processo extremamente demorado e burocrático, motivo pelo qual este trabalho não seguiu seu objetivo inicial.

Tendo em vista essa dificuldade, este trabalho foi focado no estudo de Modelos baseados em BERT e sua performance de REN em textos jurídicos escritos em português. Ao contrário do objetivo inicial, foi utilizado um conjunto de dados de textos jurídicos previamente anotado com algumas entidades relevantes (entramos em mais detalhes sobre o conjunto de dados em 3.1.1).

Apesar de serem entidades diferentes, o processo de codificação de um canal de refinamento (ou *pipeline*) de um modelo BERT para REN pode ser reutilizado quando o conjunto de dados almejado inicialmente estiver disponível. Ademais, em ambos os casos, a tarefa do modelo é a mesma: classificação de *tokens* ou conjuntos de *tokens* em categorias pré determinadas. Portanto, há similaridade o bastante entre as duas atividades para a adaptação da motivação original para o projeto de fato realizado.

1.3 Objetivo

O objetivo desse trabalho, então, pode ser resumido em dois pontos:

1. Codificar um canal de refinamento de um modelo BERT aplicado a REN. Esperamos abstrair pontos de dificuldade ao adaptar *datasets* de REN para refinamento em sistemas BERT para facilitar o trabalho de elaborar um etiquetador de expressões sexistas na ocasião da eventual liberação dos dados que seriam utilizados inicialmente neste trabalho. Elaboramos algumas dessas dificuldades em 3.1.2.

2. Estudar e documentar desafios em refinar modelos baseados em Transformadores em REN, tendo em vista a ampla utilização dessa classe de modelos em aprendizado de máquina e, em particular, PLN.

Originalmente, o objetivo deste trabalho era refinar o BERTimbau em REN, mas utilizando um dataset de processos judiciais com falas sexistas anotadas por uma equipe especialista. Assim, o modelo se encarregaria em identificar automaticamente trechos contendo frases sexistas em falas de entes jurídicos, dando uma continuidade quantitativa à pesquisa de Pimentel *et al.*, 1998. No entanto, como mencionado anteriormente, o processo de liberação dos textos não anotados dos processos judiciais foi demasiado longo e mostrou-se incompatível com o cronograma deste trabalho. Portanto, optamos por alterar o escopo do trabalho em reconhecimento de entidades genéricas em textos jurídicos.

Por fim, esperamos que as discussões e implementações postas neste trabalho e modelo obtido facilitem o desenvolvimento posterior do sistema etiquetador de expressões sexistas e de outros trabalhos envolvendo PLN com modelos baseados em transformadores aplicados em textos jurídicos escritos em Português.

Capítulo 2

Revisão Técnica

O campo de Processamento de Linguagem Natural é uma área proeminente da Inteligência Artificial (IA) desde os anos 50. Os sistemas baseados em PLN sofreram grandes transformações ao longo das décadas, partindo de sistemas fortemente sustentados em ontologias e baseados em regras (*The good old fashioned AI*), para modelos centrados em regras estatísticas nos anos 80 para, enfim, adentrar na era do Aprendizado de Máquina nos anos 90. Com o avanço computacional e introdução de processadores multi-cores nos anos 2000, tornou-se possível o uso de arquiteturas ainda mais complexas, e assim as Redes Neurais começam a ser amplamente aplicadas como soluções em PLN e assim permanecem até hoje como a principal solução arquitetural.

A sofisticação arquitetural e crescimento exponencial em poder computacional proporcionou um crescimento em sistemas baseados em PLN no cotidiano, aplicados desde mecanismos de buscas até *chatbots* e assistentes de voz. Neste capítulo, irei explicar algumas dessas transformações marcantes na área e introduzir conceitos que fundamentam particularmente as técnicas utilizadas neste trabalho de conclusão. Para tanto, essa revisão técnica se preocupa principalmente em PLN com aprendizado profundo.

2.1 Como computadores entendem significados de palavras?

Um sistema lidando com linguagem precisa, fundamentalmente, compreender o significado por trás de um símbolo que representa uma palavra. As primeiras abordagens computacionais para representar o significado de palavras envolviam soluções estáticas e manuais como tabelas de similaridades de palavras e listas de usos frequentes. Uma dessas abordagens é o WordNet (Miller, 1995), que define o significado de uma palavra pelos seus usos mais frequentes. Abordagens manuais e com manutenção por humanos possuem problemas como a rigidez incompatível com a fluidez e evolução da língua e a dificuldade em estabelecer similaridade entre palavras.

Como sistemas baseados em aprendizado de máquina tipicamente recebem coleções de valores Reais na forma de vetores como entrada, convém-se pensar em um mapeamento de palavras para números. A técnica de *Word Embeddings* consiste em uma representação

densa de palavras num espaço n-dimensional. Assim, estabelece-se um mapeamento entre palavras e uma representação vetorial propícia a ser utilizada como entrada para um modelo neural. O processo de realizar *Embedding* de palavras é a técnica mais utilizada para realizar compreensão de palavras. O mesmo se aplica a frases ou documentos (no caso, *Sentence* ou *Document Embedding*), salvo as devidas particularidades. O interessante de transformar palavras em vetores vai além de apenas viabilizar o seu uso como características (ou *features*) em sistemas de aprendizado de máquina, pois vetores permitem o uso de operações de Álgebra Linear, como cálculos de similaridades de vetores. Uma técnica comum para calcular similaridade entre dois vetores é a **Similaridade de cossenos**: Sejam A e B vetores em $\mathbb{R}^n$. Medimos a similaridade entre A e B por meio do cosseno do ângulo θ formado entre eles, calculado como:

$$sim(A, B) = cos(\theta) = \frac{\mathbf{A}.\mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} \tag{2.1}$$

Onde **A**.**B** representa o produto interno entre A e B.

Se dois *embeddings* são semelhantes, então duas palavras são semelhantes. Assim, resolve-se o problema de sistemas baseados em tabelas de similaridades, que são rígidos por concepção.

Uma abordagem clássica de codificação de palavras em características é a técnica *1-hot vector*: constrói-se uma lista de palavras de tamanho V que o sistema irá entender, chamada de **vocabulário**. Tipicamente o vocabulário é ordenado em ordem alfabética. A i-ésima palavra no vocabulário então é representada como um vetor em $\mathbb{R}^V$, onde todas dimensões valem zero, exceto pela i-ésima dimensão, que vale 1. Assim, esse método gera uma representação esparsa para uma palavra, com vetores tão grandes quanto o tamanho do vocabulário, por consequência, uma abordagem ineficiente. Não somente isso, mas, por concepção, todos *1-hot vectors* de um vocabulário de dimensão V são linearmente independentes entre si, já que constituem uma base canônica de $\mathbb{R}^V$. Em outras palavras, a similaridade entre quaisquer dois *1-hot vectors* distintos é zero. Representação de palavras por *1-hot vectors* é uma primeira abordagem para mapeamento de palavras como *input* de sistemas baseados em ML, mas falha em estabelecer relações entre palavras.

2.1.1 Word2Vec

Em Mikolov *et al.*, 2013, pesquisadores do Google publicam o modelo Word2Vec. Esse modelo foi uma evolução em relação a abordagens passadas pois com ela, os autores conseguiram uma representação densa de palavras com uma redução significativa de dimensionalidade: utilizando um vocabulário de 30 mil palavras, os autores foram capazes de gerar representações com dimensão 300, reduzindo a dimensionalidade em 99%, em comparação a 1-hot vectors, mostrando os benefícios de uma representação densa em comparação a codificações esparsas.

O Word2Vec toma como base teórica a **semântica denotacional** que afirma que o significado de uma palavra é dado pelo seu contexto: ou seja, duas palavras utilizadas em contextos similares, terão o mesmo significado. Com isso, o modelo codifica essas palavras em vetores com similaridade alta. A semântica denotacional é a principal base

teórica linguística por trás da maioria dos modelos de PLN relacionados a compreensão de palavras.

Os autores treinaram uma rede neural na tarefa de prever palavras, dado uma palavra central e algumas palavras de contexto. O modelo deveria ser capaz de prever qual a palavra central, dado as palavras usadas de contexto (tarefa de CBOW) e quais as palavras de contexto, dado a palavra central (tarefa de Skip-gram). De maneira indireta, os autores foram capazes de extrair representações densas de cada palavra do vocabulário ao analisar a matriz de pesos que representa a camada oculta da rede neural.

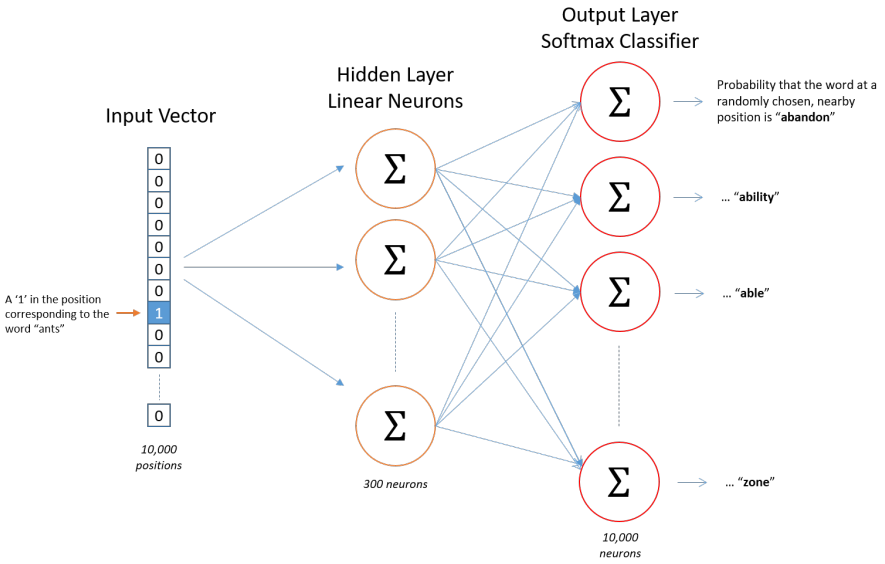


Figura 2.1: *Treino do Word2Vec em Skipgram. Imagem extraída de* McCormick, 2016

Como o input é um 1-hot vector, uma mesma palavra sempre ativa o mesmo conjunto de neurônios. No entanto, cada palavra possui um significado diferente, logo, é usada em contextos diferentes, ou seja, o objetivo de treino muda de acordo com o conjunto de palavras de contexto. Dessa forma, cada contexto diferente de uso de uma palavra contribui de maneira única para a alteração dos pesos ativados durante o treino. Consequentemente, o *embedding* de uma palavra pelo word2vec pondera todos seus usos. Assim, a representação é independente de contexto e não abrange polissemia. Chamamos esse tipo de representação de **livre de contexto**. Na sessão 2.5, iremos abordar o modelo BERT, que apresenta representação sensível a contexto.

Frequentemente, os *embeddings* são armazenados em matrizes, chamadas de **matrizes de *embeddings***. A matriz nada mais é do que a matriz de pesos da camada oculta da rede neural mencionada anteriormente. Nessa representação, cada linha da matriz de dimensão VxD - onde D é a dimensão de cada embedding, representa o embedding de uma palavra. O acesso ao embedding é feito por um simples produto do one-hot vector pela matriz de *embeddings*, extraindo a representação densa correspondente.

Word2vec é uma das primeiras abordagens de *Word Embedding* que atingiu altos níveis de adoção pela comunidade de PLN devido ao seu sucesso em representar similaridade entre palavras usadas em contextos similares com uma representação em baixa dimensionalidade.

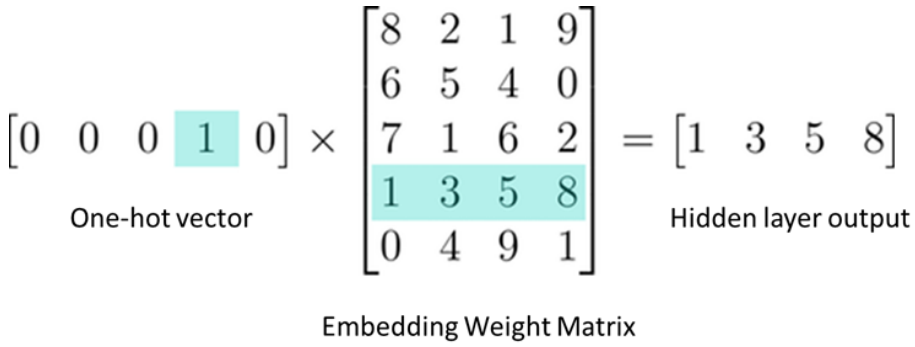


Figura 2.2: *Processo de extração de representação densa de uma matriz de embedding. Imagem extraída de https://towardsdatascience.com/what-the-heck-is-word-embedding-b30f67f01c81*

2.2 Modelos de Linguagem

Bengio *et al.*, 2003 define um modelo de linguagem como um modelo ou algoritmo que aprende a distribuição de probabilidade de sequências de palavras dentro de textos expressos em linguagem natural. Podemos modelar a probabilidade de ocorrência de uma sequência de palavras condicionando pelo contexto de palavras que a precedem:

$$P(w_1, w_2, \dots, w_t) = P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_t|w_{t-1}, \dots, w_2, w_1) \tag{2.2}$$

Podemos, alternativamente, modelar a ocorrência de uma sequencia de palavras de maneira bidirecional:

$$\begin{aligned} P(w_1, w_2, \dots, w_t) = G(&P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_t|w_{t-1}, \dots, w_2, w_1) \\ &, P(w_t)P(w_{t-1}|w_t) \dots P(w_1|w_2, w_3, \dots, w_t)) \end{aligned} \tag{2.3}$$

Onde $G : \mathbb{R} \longrightarrow [0, 1]$. Note que modelamos a probabilidade de uma sequência de palavras como alguma função entre duas modelagens diferentes dessa ocorrência. Podemos pensar bidirecionalidade como uma combinação de modelar a linguagem lendo o texto da direita para a esquerda com a modelagem de ler o texto da esquerda para a direita. Essa discussão será retomada na seção 2.3.3.

A tarefa de um modelo de linguagem, então, é aprender uma função $F : V \longrightarrow [0, 1]$, onde V é o vocabulário, de maneira que $F \approx P$, ou seja, uma aproximação da distribuição de probabilidade.

Modelos de linguagem podem ser divididos em duas categorias: **estatísticos** e **neurais**. Modelos estatísticos utilizam uma abordagem frequentista para aproximar a função densidade de probabilidade P por meio da equação:

$$\begin{aligned} P(w_1, w_2, \dots, w_t) &\approx f(w_1, w_2, \dots, w_n) \\ &= P(w_1, w_2, \dots, w_n), \\ n < t, f &: V^n \longrightarrow [0, 1] \end{aligned} \tag{2.4}$$

Esse modelo é chamado de **n-gramas**. Em suma, o modelo analisa a ocorrência de n-tuplas de palavras em um *corpus* de treino e com isso constrói uma aproximação da função de probabilidade. Esses modelos não lidam bem com palavras desconhecidas no vocabulário: o modelo naturalmente associaria uma probabilidade 0 de ocorrência a uma sequência de palavras que contém uma palavra desconhecida. Além disso, o crescimento da quantidade de palavras no conjunto de treinos leva a um aumento exponencial na quantidade de n-gramas a serem aprendidos.Os autores em Bengio *et al.*, 2003 chamam esse problema de *curse of dimensionality* (maldição de dimensionalidade).

Modelos neurais, por sua vez, aprendem uma aproximação da função de probabilidade pelo uso de uma rede neural. Além disso, esses modelos codificam palavras como *word embeddings*. Seja $C(w) \in \mathbb{R}^d$ a codificação de uma palavra $w \in V$, onde d é a dimensão do embedding. O input de t palavras na rede neural pode ser definido como:

$$X = [C(w_1), C(w_2), \dots, C(w_t)] \tag{2.5}$$

Por simplicidade, vamos denotar a saída da rede neural como $\hat{y} = h(X), \hat{y} \in R^V$. Definiremos h de acordo com a arquitetura abordada. A aproximação da função de probabilidade é feita por qualquer função $f : \mathbb{R}^V \longrightarrow [0, 1]$. Tipicamente, usamos uma função Softmax. Dado um vetor X de entrada, calcula-se o softmax como:

$$P(w_t = k | w_{t-1}, \dots w_1) \approx \frac{e^{\hat{y}[k]}}{\sum_{i=1}^{t} e^{\hat{y}[i]}} \tag{2.6}$$

A função Softmax é comumente usada em sistemas de ML por receber um vetor e normaliza-lo em uma distribuição de probabilidade.

Por usar uma representação contínua de palavras, esses tipos de modelos neurais se saem bem com sequências de palavras desconhecidas durante o treino mas que possuem um *embedding* similar. Nesse trabalho, iremos focar nos modelos de linguagem neurais.

2.3 Modelos de linguagem Neurais: as principais soluções pré Transformadores

Com a adoção de *embeddings* palavras passam a ser compreendidas com um vetor, o que torna o uso de arquiteturas neurais interessante tendo em vista que o *embedding* é uma coleção de valores numéricos, o que pode representar uma coleção de *features* como entrada para um modelo.

2.3.1 Redes neurais *Feed Forward*

Redes neurais feed forward foram a primeira arquitetura sugerida para modelos de linguagem neurais por Bengio *et al.*, 2003. Nessa arquitetura neural, o fluxo de informação é unidirecional, partindo da entrada e passando pela camada oculta para, enfim, produzir uma saída.

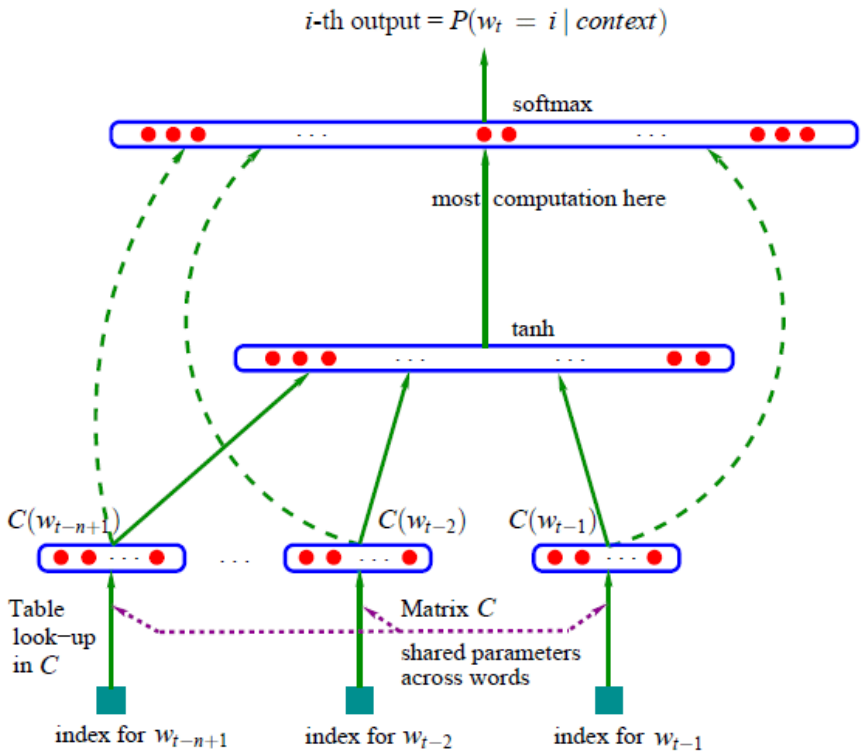


Figura 2.3: *Modelo de linguagem neural baseado em redes feed forwards. Imagem extraída de Bengio et al., 2003*

A figura 2.3 representa um modelo de linguagem neural usando redes *feed forward*. A quantidade de informação a ser processada pela rede neural precisa ser definida de antemão, já que a quantidade de neurônios na camada de *input* é fixa. Partindo de *1-hot vectors*, o modelo recupera a representação contínua de cada palavra da matriz C_{VxD}de *embedding*. Em um modelo com apenas uma camada oculta, podemos definir a função $h(X)$ dessa rede neural como:

$$h(X) = tanh(W^T X + b) \tag{2.7}$$

Onde W é uma matriz de pesos e b um vetor de viés.

Usar uma quantidade fixa de palavras como contexto cria uma limitação no poder do modelo de linguagem. Contextos muito pequenos podem gerar modelos de linguagem enviesados por localidade, enquanto que janelas grandes aumentam o tamanho da matriz W, o que gera um gargalo em tempo de treino.

2.3.2 Redes Neurais Recorrentes

Modelos de linguagem baseados em RNN tomam proveito da natureza sequencial do texto. RNNs são soluções neurais comumente aplicadas a qualquer tipo de dado classificado como **Série Temporal**, como preços de ações, sinais de áudio e o próprio texto. Esse tipo de modelo de linguagem neural processa palavra por palavra, sendo assim, não apresenta limitação de quantidades de palavras a serem consideradas .

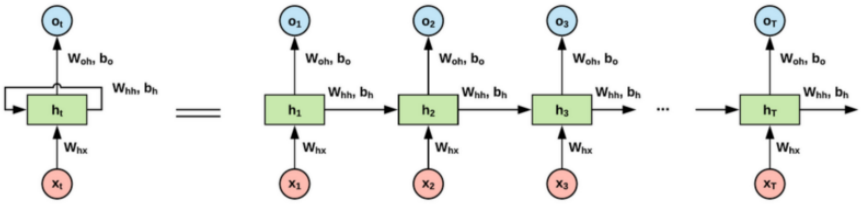


Figura 2.4: *Arquitetura de uma rede neural recorrente. Imagem extraída de https://towardsdatascience.com/introduction-to-recurrent-neural-networks-rnn-with-dinosaurs-790e74e3e6f6*

A principal diferença dessa arquitetura é a possibilidade de retroalimentação da saída da rede como uma entrada no próximo passo de tempo. Cada passo de tempo é sinalizado por um novo *input* sendo recebido pela rede.

Convém-se representar RNNs de duas maneiras: compacta e estendida. A figura 2.4 ilustra essas duas representações: na esquerda, a representação compacta. A rede neural produz uma saída a cada passo de tempo t, mas também a reutiliza como entrada. Na direita, cada passo de tempo é representado individualmente, o que gera uma representação mais didática do funcionamento de RNNs. Podemos calcular a ativação interna h_t da rede em um passo de tempo t como:

$$h_t = g_1(W_{hx}x_t + W_{hh}h_{t-1} + b_1) \tag{2.8}$$

Onde g_1 é uma função de ativação, b_1 é um viés, W_{hx} e W_{hh} são matrizes de pesos. As matrizes de peso são as mesmas para todo passo de tempo.

A saída da RNN para cada passo de tempo t, a_t, é obtida de maneira similar:

$$a_t = g_2(W_{ha}h_t + b_2) \tag{2.9}$$

Nem todas RNNs produzem uma saída a_t para cada passo de tempo t. RNNs que fazem classificação de *token* a *token*, como modelos de REN, produzem uma saída a cada passo de tempo. No entanto, RNNs interessadas em classificar uma frase produzem uma saída apenas no último passo de tempo da sequência.

Por poder processar uma quantidade indeterminada de palavras como entrada e apresentar um processamento sequencial apropriado a dados de série temporal, essa arquitetura se popularizou intensamente como escolha para modelos de linguagem neurais.

Analisando a equação da ativação interna e a representação extensa de uma RNN, fica claro que ela se comporta de maneira similar a redes neurais profundas. Consequentemente, RNNs apresentam o mesmo problema que arquiteturas muito profundas, como o de **esquecimento** gerado pelo problema do *Vanishing Gradient*.

RNNs aprendem por meio de uma variação do algoritmo clássico de *Backpropagation*, chamado de *Backpropagation through Time* (BPTT). Em suma, o BPTT calcula a derivada parcial da função *Loss* em relação a matriz de peso em cada passo de tempo t, sendo calculado a cada T passos de tempo:

$$\frac{\partial \mathcal{L}^T}{\partial W} = \sum_{i=1}^{T} \frac{\partial \mathcal{L}^T}{\partial W}\Big|_{(t)} \tag{2.10}$$

Como a função de perda no passo T depende da função de perda no passo $T-1$, pela regra da cadeia, a derivação acima se torna um produtório. Assim, gradientes pequenos sucessivos na multiplicação geram um produto muito pequeno, causando uma atualização de pesos tão menor quanto mais longa for a dependência.

Explicar o BPTT em profundidade foge ao escopo dessa sessão, no entanto, a compreensão de como ele é calculado é importante para entender o problema do *Vanishing Gradient*. Werbos, 1990 apresenta a técnica em detalhes e mostra as derivações relevantes.

Com o problema do esquecimento, surgiu a necessidade de um modelo capaz de memorizar dependências de longo prazo conforme fosse necessário, isto é, algum estado interno antigo que, se memorizado, resulta em um ganho de performance durante o treino

2.3.3 LSTMs

As redes ***L**ong **S**Hort **T**erm **M**emory* (Redes de memória longo-curto prazo, em tradução livre) foram introduzidas por Hochreiter e Schmidhuber, 1997 para atenuar o problema do *Vanishing Gradient*. LSTMs são um caso especial de RNNs: elas mantém ainda a estrutura linear e boa parte do funcionamento das redes recorrentes, mas elas são capazes de armazenar dependências longas no texto com mais eficiência, sendo capazes de controlar o que é armazenado internamente. Esse controle é definido de acordo com o contexto que a LSTM está analisando.

A figura 2.5 representa 3 passos temporais de uma LSTM e o funcionamento interno da rede. Em essência, LSTMs possuem dois estados internos: o estado oculto h_t e o estado de célula c_t, que é responsável por manter a memória de longo prazo internamente. Ambos são vetores de tamanho m. O controle do fluxo de informação interno à LSTM é feito por **portões**, vetores também de tamanho m. A LSTM possui três portões:

1. Forget gate (portão de esquecimento): controla se a informação em c_{t-1} deve ser mantida

2. Input gate (portão da entrada): define se c_t deve ser atualizado com a informação em h_t

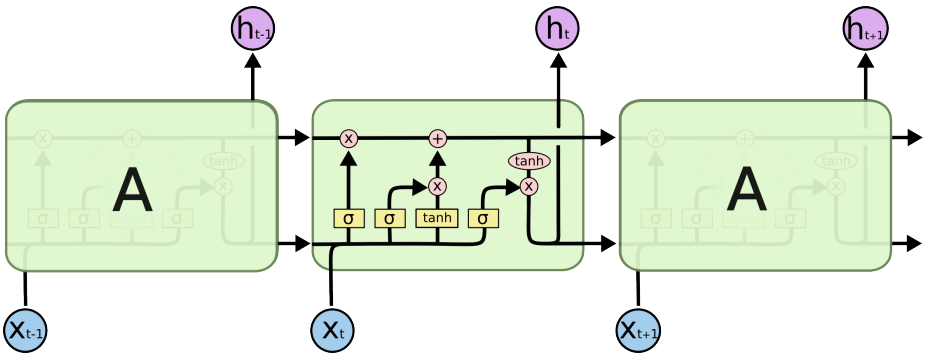


Figura 2.5: *Arquitetura de uma LSTM. Imagem extraída de http://colah.github.io/posts/2015-08-Understanding-LSTMs/*

3. Output gate (portão de saída): define se h_t irá receber a informação armazenada em c_t.

Hochreiter e Schmidhuber, 1997 apresenta uma explicação mais detalhada de como cada portão funciona e como isso interfere no aprendizado de longo prazo das LSTMs.

LSTMs se tornaram a opção dominante em PLN para modelagem de linguagem neural de 2013 a 2017 Bojar *et al.*, 2018. Um dos casos de uso de destaque foi para modelagem de modelos bidirecionais de linguagem, com as BiLSTMs. Esse modelo concatena a saída de duas redes LSTMs: uma que processa o texto da direita para a esquerda, e outra que processa o texto da esquerda para a direita. O modelo de linguagem aproximado por BiLSTMs pode ser descrito como:

$$\begin{aligned} & P(w_t|w_{t-1}, w_{t-2} \dots w_1, w_{t+1}, w_{t+2}, \dots) \\ \approx\ & P(w_t|w_{t-1}, \dots, w_1) * P(w_t|w_{t+1}, \dots, w_{t+n}) \end{aligned} \tag{2.11}$$

Note que a modelagem toma como hipótese que o modelo de linguagem da esquerda para a direita é independente do modelo de linguagem da direita para a esquerda. Essa aproximação de bidirecionalidade apresenta ganhos de performance em relação a modelos unidirecionais mas apresenta uma bidirecionalidade rasa: ao compor um texto, a presença de um termo no início de um período pode impactar a presença de uma palavra ao fim da frase, da mesma maneira que um termo no fim da frase impacta qual termo é mais apropriado no início do período. Um modelo verdadeiramente bidirecional foi proposto por Devlin *et al.*, 2019 com o BERT. O modelo será explicado em detalhes na sessão 2.5.

2.3.4 Modelos encoder/decoder

Uma aplicação interessante da arquitetura LSTM foi em modelos *Encoder/Decoder* (codificador/decodificador). Essa classe de modelos é uma combinação de dois modelos de linguagem: O modelo codificador é responsável por receber uma sentença de entrada e codifica-la num *embedding*, simbolizando a frase inteira. O modelo decodificador, por sua vez, recebe a codificação e gera uma sentença a partir dela. Em outras palavras, o modelo codificador é especialista em abstrair uma frase em um vetor, enquanto que o

modelo decodificador é especialista em geração de texto. Esse tipo de arquitetura foi particularmente muito utilizada para tradução de texto (Cho *et al.*, 2014). A área de PLN que se preocupa em usar arquiteturas neurais para tradução de texto é chamada de *Neural Machine Translation* (NMT) - Tradução neural de máquina -.

O uso de arquiteturas neurais para tradução de texto só passou a ser usado amplamente com o uso do mecanismo de atenção, introduzido em Bahdanau *et al.*, 2016. Atenção é uma técnica utilizada em diversas áreas de Aprendizado de Máquina além de PLN, como Visão Computacional. O Mecanismo de Atenção serve para enfatizar a importância de certos trechos do texto. Isso é feito simplesmente atribuindo pesos a cada estado interno da rede neural de entrada. Bahdanau *et al.*, 2016 critica a abordagem classica de sistemas NMT por comprimirem uma frase de origem em uma codificação de tamanho único. Dessa maneira, a codificação de frases muito longas ficaria comprimida demasiadamente e informação seria perdida. Os autores propuseram então um modelo de codificadores usando RNNs bidirecionais onde a codificação é feita usando fortemente o mecanismo de atenção, dessa maneira porções importantes do texto de origem são reforçados pelo seu peso de atenção. O uso de sistemas Encoder/decoder com atenção trouxe um enorme ganho de performance em sistemas NMT

Vaswani *et al.*, 2017 parte da idéia de usar atenção para incrementar a performance em modelos *encoder/decoder* e vai além, usando somente atenção como mecanismo de aprendizado e abandonando redes recorrentes.

2.4 Modelos baseados em Transformadores

O modelo de Transformadores foi introduzido em Vaswani *et al.*, 2017 com a motivação de criar um modelo voltado a tarefas de NMT com performance superior e tempo de treino reduzido em relação a modelos baseados em LSTMs com atenção. Não somente o Transformador cumpre esses processos, como é possível colocá-lo como uma das grandes transformações na área de PLN nos últimos 10 anos. Mesmo sendo um modelo mais complexo do que as soluções correntes até então, o tempo reduzido de treino se deu pelo uso intenso de operações altamente paralelizáveis em seu funcionamento interno, benefício que as arquiteturas sequenciais não usufruíam.

O Transformador também atenua mais intensamente o problema do esquecimento. Isso se deve ao uso do mecanismo de auto-atenção ao analisar uma janela fixa de entrada: em linhas gerais, o modelo analisa uma porção fixa do texto (o artigo analisa 512 *tokens* por padrão) e realiza correlações entre todas as palavras de entradas em relação a elas mesmas, gerando uma matriz de correlações.

Podemos separar o Transformador em duas porções: a codificadora e a decodificadora. Essas duas porções se comunicam de maneira similar a modelos *encoder/decoder*, com ligeiras diferenças no decodificador e grandes mudanças no funcionamento interno de cada sessão.

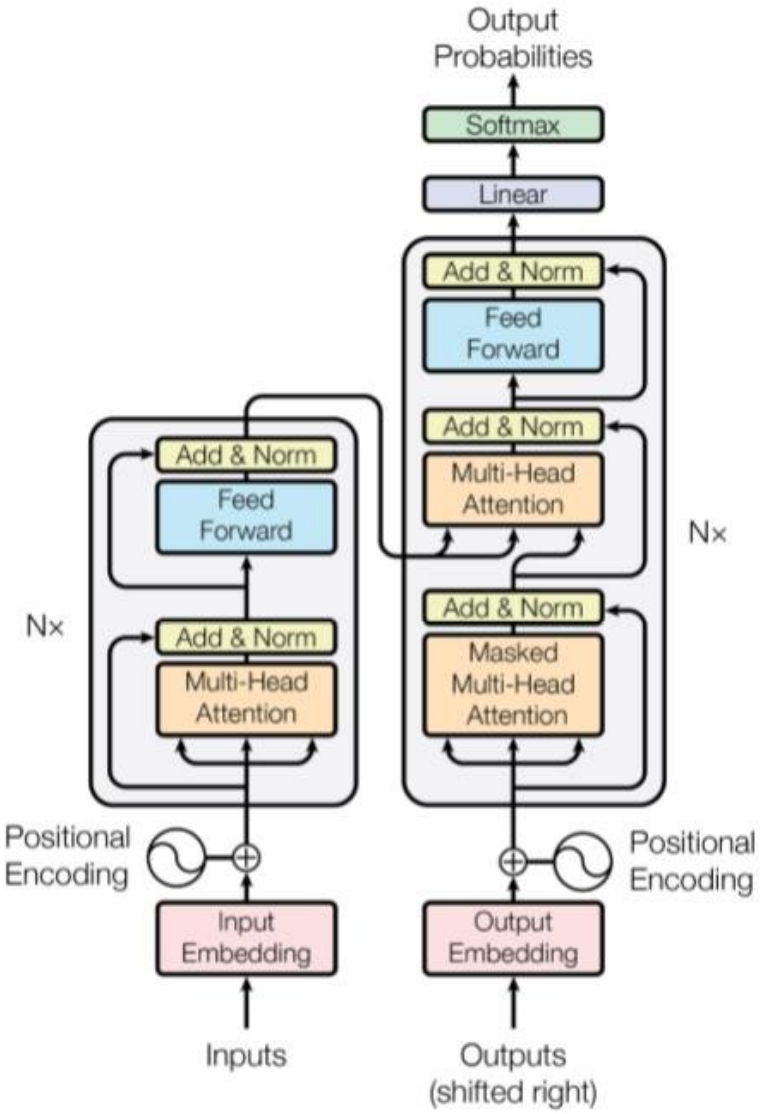


Figura 2.6: *O transformador. A estrutura à esquerda da figura representa o codificador, enquanto que a estrutura à direita representa o decodificador. Imagem originalmente em* VASWANI *et al., 2017*

2.4.1 Estrutura do transformador

Nessa seção, vamos explicar brevemente a estrutura do transformador. Focaremos majoritariamente na porção codificadora pois o modelo usado neste trabalho é uma extensão do codificador do transformador. O leitor que estiver em busca de uma explicação mais detalhada da estrutura como um todo pode recorrer a Vaswani *et al.*, 2017 ou a Alammar, 2018. O funcionamento deste modelo é relativamente complexo, mas a sua compreensão é fundamental para entender o funcionamento de modelos modernos de PLN como BERT e GPT.

Como o Transformador possui uma porção fixa de texto e correlaciona toda palavra entre si, o modelo perde a noção de sequencialidade entre palavras. Mesmo esse tipo de modelagem já ter se mostrado problemática devido ao problema de esquecimento, sequencialidade é fundamental na compreensão de texto. Para tanto, os autores introduziram uma unidade tanto no codificador quando no decodificador responsável por realizar uma codificação da entrada posicional. Portanto, além de realizar o *embedding* tradicional de cada *token* de entrada, os autores introduzem um conhecimento posicional extra de acordo com a posição do *token* na entrada.

O codificador do transformador é composto por uma pilha de 6 camadas de codificadores (os autores usam 6 como um número experimental). A primeira camada do codificador recebe os *embeddings* do *input* e gera uma codificação. Desse ponto em diante, cada camada recebe a saída da camada anterior, gerando uma representação profunda da entrada. Cada camada do codificador é composta por duas unidades lógicas: uma unidade de auto-atenção de múltiplas cabeças, e uma unidade de rede neural *feed forward*. A auto-atenção de múltiplas cabeças nada mais é do que diversas unidades de auto-atenção processando a entrada em paralelo, com pesos inicializados de maneira aleatória. Vamos detalhar o funcionamento de uma única cabeça a seguir.

A unidade de auto-atenção codifica uma frase em uma matriz, onde cada *token* recebe uma codificação. Essa codificação nada mais é do que a correlação da palavra em questão em relação a toda palavra na sentença de entrada. A figura 2.7 ilustra como a auto-atenção funciona: pesos maiores (em laranja escuro) são atribuídos a palavras que possuem um grau elevado de relação com a palavra sendo analisada. Ao usar diversas cabeças de atenção treinadas em paralelo, os autores propuseram codificar diferentes relações entre os termos de entrada, e com isso capturar diferentes relações semânticas e sintáticas. Vaswani *et al.*, 2017 exemplifica visualmente as diferentes relações que cada cabeça faz para uma mesma entrada.

Cada representação densa é passada por uma rede neural em paralelo. Todas saídas são combinadas e normalizadas, gerando a saída da camada do codificador. Assim, o codificador é capaz de capturar relações sutis na frase de entrada e gerar uma representação densa e profunda para uma entrada.

O decodificador, por sua vez, recebe a saída de cada camada do codificador e gera token a token a saída decodificada. Para gerar a próxima palavra, o decodificador recorrentemente realiza a operação de atenção, similarmente feito em Cho *et al.*, 2014 em tarefas de atenção de NMT, entre as saídas de cada camada do codificador e a saída do decodificador no passo de tempo anterior. Com isso, o modelo espera gerar um próximo *token* que faça sentido de

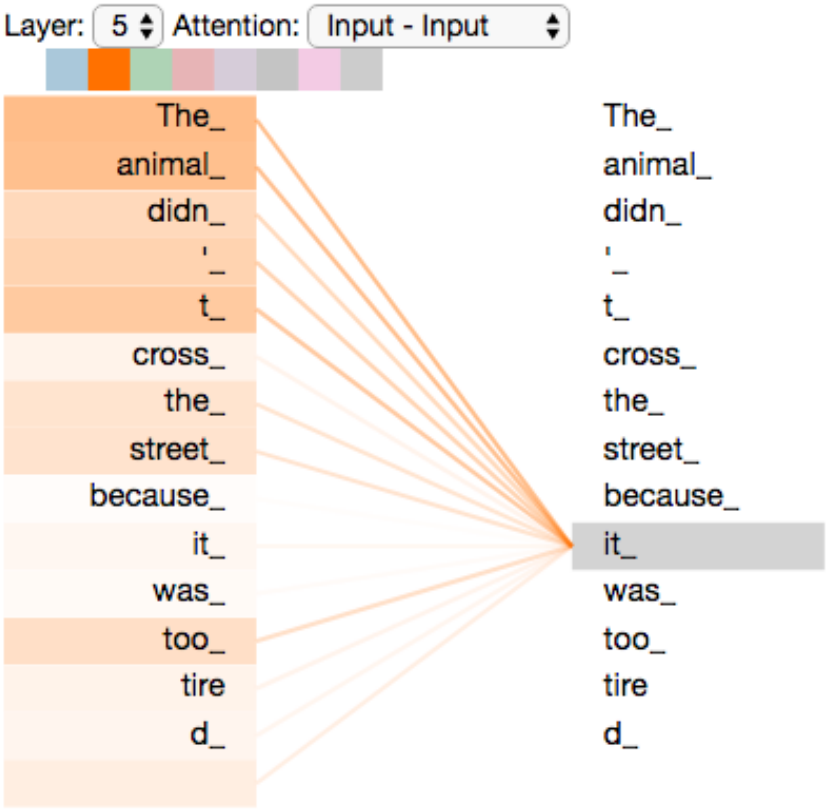


Figura 2.7: *O mecanismo de auto-atenção. Imagem originalmente em* Alammar, 2018.

acordo com o que foi gerado até então ao mesmo tempo que seja algo que faça sentido com partes chaves do *input* do codificador.

Em geral, modelos baseados em transformadores focados em tarefas de compreensão de texto e codificação fazem uso da porção codificadora do transformador, enquanto que modelos voltados a geração de texto e processamento sequencial fazem uso da porção decodificadora. A estrutura do transformador foi adaptada em dois modelos populares: o BERT faz uso da porção codificadora do transformador, enquanto que o GPT usa a porção decodificadora.

2.5 BERT

O modelo BERT foi desenvolvido pela equipe do Google A.I e publicado em Devlin *et al.*, 2019. O modelo faz uso da porção codificadora dos transformadores e, devido ao seu processo de pré-treino, gerou um modelo de linguagem verdadeiramente bidirecional. O Modelo é predominantemente utilizado para tarefas atreladas a codificação e Compreensão de Linguagem Natural. Convém-se destacar as tarefas de Análise de sentimento de texto, vinculação textual, Respostas a perguntas e REN.

Em tarefas como Respostas a perguntas e vinculação textual (tarefa que consiste em verificar se duas frases estão logicamente relacionadas), o modelo recebe duas sentenças,

enquanto que em tarefas como análise de sentimento de texto e REN o modelo recebe uma sentença; como o modelo é capaz de processar uma sentença ou duas sentenças, iremos chamar a entrada do modelo como "sequência", independentemente da quantidade de sentenças recebidas.

2.5.1 Arquitetura do modelo

O modelo possui duas porções: a base e a cabeça. A base é uma pilha de codificadores de Transformadores que compõe o modelo de linguagem bidirecional. A cabeça é uma porção destacável do modelo, cujo formato muda de acordo com a tarefa alvo do modelo. Essa cabeça nada mais é do que uma camada de output responsável por classificação a nível de *token* ou a nível de sentença, dependendo da função do modelo.

Os autores publicaram dois modelos: *base* e *large*, onde a principal diferença entre os dois é o número de parâmetros usados, logo, o tamanho do modelo: O modelo base apresenta 12 camadas de codificadores, uma *Hidden layer* na FFNN de tamanho 768 e 12 cabeças de auto-atenção, totalizando 110 Milhões de parâmetros; O modelo *large* apresenta 24 camadas de codificadores, *Hidden Layer* da FFNN de tamanho 1024 e 16 cabeças de atenção, totalizando 340 milhões de parâmetros. Os autores mostraram que modelos maiores apresentam ganho de performance mesmo lidando com *datasets* pequenos.

2.5.2 Contribuições

Podemos citar como principal contribuição do modelo a construção de um modelo de linguagem verdadeiramente bidirecional e com *embedding* contextual, introduzindo polissemia. Essa contribuição se deu principalmente pelo processo de pré treino, que será explicado em mais detalhes a seguir. O treino do modelo é separado em dois passos: pré treino e refinamento.

2.5.3 Pré treino

Durante o pré treino, o modelo é treinado usando dados não etiquetados onde o objetivo é criar uma modelagem de linguagem robusta. Essa modelagem de linguagem é agnóstica à aplicação posterior do modelo, então as tarefas de pré treino são genéricas às tarefas de aplicação do modelo.

O modelo é pré treinado em duas tarefas: *Masked Language Modeling* e *Next sentence Prediction*. A tarefa de *Masked Language Modeling* é inspirada na tarefa de Cloze (Taylor, 1953): nessa tarefa, um leitor recebe um trecho de um texto com algumas palavras censuradas. O leitor deve, então, tentar substituir a palavra censurada por outra que faça sentido no contexto . No caso do modelo, a tarefa consiste em bloquear (ou mascarar) *tokens* do texto de entrada e o modelo deve tentar prever qual é a palavra mascarada. Por exemplo:

O homem foi à [MASK] para comprar uma [MASK] de leite

O modelo deve prever que o primeiro *token* provavelmente é "Loja"e o segundo "caixa". Para preencher a primeira lacuna, analisa-se a porção à esquerda do termo para entender

que o homem foi a uma localização. Não somente isso, mas a uma localização com artigo feminino, devido ao uso da crase. Se olharmos à direita, nota-se que o homem comprou algo de leite. Portanto, é possível que o primeiro *token* seja um estabelecimento comercial no qual se vende leite. Usando tanto o contexto da esquerda quanto da direita em conjunto, podemos descartar opções como "banco"e "mercado"e considerar como bons candidatos ao primeiro *token* mascarado os termos "padaria"e "mercearia". Como o BERT é construído sobre transformadores e faz uso de auto-atenção, o modelo é capaz de enxergar toda a frase de entrada em tempo de treino. Assim, aliando uma estrutura de auto-atenção com um pré-treino propício, o modelo é capaz de aprender uma modelagem de linguagem verdadeiramente bidirecional, em contraste com a modelagem rasa de BiLSTMs.

Os autores ressaltam que, a princípio, esse processo de treino introduz um viés, já que o token [MASK] nunca será visto em validação ou em teste, apenas em treino.Para atenuar esse problema, em 80% das vezes o mascaramento é feito como planejado. Em 10% das vezes, o modelo substitui o *token* por outra palavra do vocabulário, e não por [MASK]. Por fim, em 10% das vezes o modelo não altera a palavra trocada e o modelo deve apontar que a palavra está bem encaixada em seu contexto. Assim, os autores evitaram viés de treino ensinando o modelo a entender se a palavra realmente faz sentido no contexto analisado.

A outra tarefa de pré treino é predição de próxima sentença. Essa tarefa é responsável por modelar relações entre sentenças: dado sentenças A e B, o modelo deve verificar se B segue A ou se é uma ordem aleatória.

O pré treino realizado em Devlin *et al.*, 2019 foi feito utilizando 2.5 bilhões de *tokens* extraídos da Wikipédia em inglês e 800 milhões de *tokens* extraídos do *Book Corpus*. O processo de pré treino gera um excelente modelo de linguagem em inglês (surpreendentemente, o modelo em inglês possui um desempenho razoável em tarefas usando texto em português). Os autores só recomendam refazer o processo de pré-treino se há o interesse em especializar o modelo em outra linguagem ou textos de domínio muito específico: Souza *et al.*, 2020 retreinaram o BERT usando textos em português, introduzindo o BERTimbau, um modelo de linguagem com excelente desempenho em português.

O leitor interessado em utilizar o BERT não precisa realizar o pré treino, podendo baixar o BERT já pré treinado. Um bom recurso para acessar modelos baseados em BERT pré treinados é o *HuggingFace* (https://huggingface.co/)

2.5.4 BERTimbau

Em Souza *et al.*, 2020, foi introduzido o BERTimbau: um modelo baseado em BERT pré treinado integralmente em português. Devido ao seu pré treino, o modelo superou a performance do BERT original e de outros modelos pré treinados utilizando textos de diversas línguas, como o Multi-lingual BERT. Por causa da sua performance superior em modelagem de linguagem e outras tarefas de PLN, esse foi o modelo pré-treinado utilizado neste trabalho de conclusão.

2.5.5 Refinamento

Na literatura, existem duas abordagens para uso de modelos de linguagem pré-treinados: *feature based* e *fine tuning based* (Baseado em características e baseado em refinamento, respectivamente, em tradução livre). A primeira abordagem usa o LM como uma *feature* separada do modelo especialista: um sistema *feature based* interessado em realizar análise de sentimento de texto usa o LM pré treinado como uma etapa de pré processamento do dado de entrada, e o modelo de análise de sentimento recebe esse dado pre processado e é treinado de maneira separada. Um LM popular que segue essa abordagem é o **ELMo** (Peters *et al.*, 2018).

Já abordagens *fine tuning based* acoplam o LM pré treinado e a cabeça de maneira mais próxima; o LM ainda é responsável por preprocessar o texto de entrada, mas, ao treinar o modelo especialista, o refinamento de parâmetros é feito de ponta a ponta: ou seja, o LM também é treinado mais uma vez, mas agora com um objetivo de aprendizado pertinente ao objetivo do modelo na ponta. O BERT é um modelo que segue essa técnica.

O processe de refinamento consiste em alimentar o BERT com dados e etiquetas pertinentes à tarefa considerada, além de adicionar uma camada de classificação ao modelo.

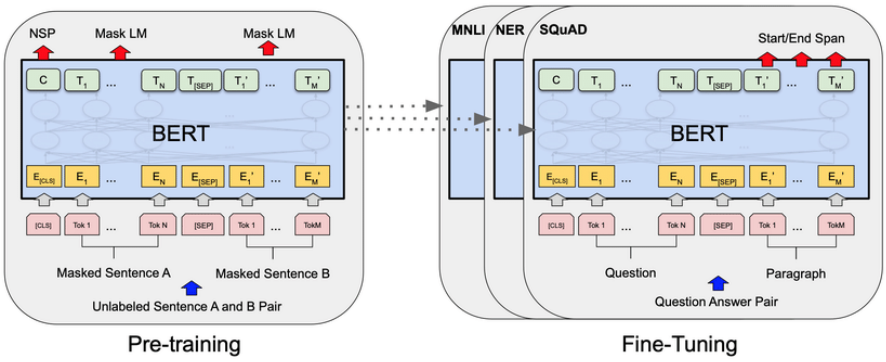


Figura 2.8: *Pré treino e refinamento no BERT. Imagem extraída de* Devlin *et al., 2019*

A figura 2.8 ilustra uma comparação entre o processo de pré treino, na esquerda, e o processo de refinamento, na direita. A figura mostra, em particular, o processo de refinamento na tarefa de *Question Answering*. Nessa tarefa, o modelo recebe uma sequência de entrada representando uma pergunta e um parágrafo que pode conter a resposta da pergunta. O modelo deve ser capaz de inserir, no parágrafo, dois *tokens* especiais que simbolizam o início e fim da região do parágrafo que responde a pergunta posta. Durante o processo de treino, todos os parâmetros do BERT são atualizados de maneira a obter um modelo de linguagem que é melhor em *Question Answering* do que o modelo apenas pré treinado. O processo de refinamento, no entanto, é muito menos custoso do que o processo de pré treino: enquanto que o processo de pré treino pode demorar alguns dias treinando em TPUs potentes, o processo de refinamento pode ser treinados em poucas horas usando GPUs básicas.

2.5.6 *Tokens* especiais

A figura 2.8 mostra uma sequência de entrada. Nela, vemos alguns *tokens* especiais. Nessa subseção iremos discorrer rapidamente sobre eles e sobre o processo de tokenização do BERT.

CLS : *token* de classificação. Quando o BERT é usado para classificar uma sentença, este token é usado para representar a sentença inteira

SEP : representa a separação entre duas sentenças em uma sequência. Não apresenta significado semântico.

PAD : o BERT processa sentenças de até 512 *tokens*. Pode ser interessante padronizar o tamanho das sequências de entradas para processamento paralelo. Para tal finalidade, usa-se este token de *padding*. Ele também não apresenta significado semântico.

UNK : Quando o BERT falha em encontrar um embedding para um *token*, ele atribui este *token* especial.

2.5.7 Tokenização

Um problema muito comum em PLN é como lidar com palavras OOV. Ao construir um vocabulário, é impossível contemplar todas palavras possíveis nessa lista finita, tendo em vista que se levarmos em conta processos de sufixação e prefixação, o número de palavras possíveis torna-se potencialmente infinito. A abordagem que o BERT e outros sistemas de PLN usam é o de tokenização: o processo de tokenização, em linhas gerais, consiste em quebrar uma *string* de entrada em *tokens* (tipicamente separando pelo carácter de espaço), unidades atômicas de processamento textual. Após essa quebra, o sistema precisa identificar se o *token* é conhecido internamente no vocabulário do modelo: se for conhecido, recebe o *embedding* correspondente; se não for, aplica-se uma heurística para lidar com o dado faltante. A heurística mais simples é atribuir um *embedding* padrão para todo token OOV.

O BERT usa o algoritmo **WordPiece** para lidar com tokens OOV, introduzido por Schuster e Nakajima, 2012. Esse processo de tokenização consiste em quebrar uma palavra em sub palavras de maneira gulosa até encontrar uma sub palavra que estiver no vocabulário. Se o WordPiece falha em encontrar qualquer *subtoken* no vocabulário, atribui-se o token especial [UNK] desconhecido para a entrada. Um exemplo de tokenização usando o tokenizador em português introduzido por Souza *et al.*, 2020:

Andando no calçadão de Osasco
And ##ando no cal ##ça ##dão de Osa ##s ##co

No exemplo, os *tokens* "Andando", "Calçadão"e "Osasco"não estão no vocabulário, então são quebrados. Note que "Andando"é quebrado em "And"e "ando", evidenciando o processo de sufixação. O Wordpiece minimiza drasticamente o problema de *tokens* OOV.

2.5.8 Refinamento BERT em REN

REN é uma tarefa de classificação a nível de *token*: o modelo deve ser capaz de classificar token a token em categorias pré definidas. *Datasets* de REN consistem em pares palavra-entidade, onde as palavras tipicamente não estão tokenizadas. Tome como exemplo a seguinte entrada presente no dataset utilizado nos experimentos:

```
Apelação   - B-JURISPRUDENCIA
cível      - I-JURISPRUDENCIA
pelo       - O
Ministério - I-ORGANIZAÇÃO
```

No exemplo, três tipos de entidades são representados: "O"(entidade para palavras sem categoria definida) , "ORGANIZAÇÃO"e "JURISPRUDÊNCIA". O programa 2.1 mostra o processo de tokenização de uma frase utilizando o tokenizador do BERTimbau.

Programa 2.1 *Script* Python representando o processo de tokenização de uma frase utilizando o tokenizador BERTimbau

```
1   >>> from transformers import AutoTokenizer
2   >>> checkpoint = 'neuralmind/bert-base-portuguese-cased'
3   >>> tokenizer = AutoTokenizer.from_pretrained(checkpoint)
4   >>> tokenizer.tokenize("Apelação cível pelo Ministério Público")
5
6   ['Ap', '##ela', '##ção', 'cí', '##vel', 'pelo', 'Ministério', 'Público']
7   ]
```

Como discorrido em 2.5.7, o BERT tokeniza todo dado de entrada. Por consequência existiria uma inconsistência entre a quantidade de tokens classificados e a quantidade de etiquetas fornecidas: no exemplo acima, o número de *tokens* resultantes foi 8, mas o número de etiquetas foi 5. Para contornar essa dificuldade, Devlin *et al.*, 2019 aconselham tomar como decisão de projeto considerar apenas o primeiro *token* resultante de uma palavra tokenizada.

Devido ao uso de auto-atenção o modelo ainda é capaz de compreender diferentes usos do mesmo radical mas a quantidade de informação carregada apenas pela raiz de uma palavra é muito reduzida, o que pode acarretar em um menor poder do modelo em fazer classificação de tokens altamente segmentados. Khan, 2021 explora a possibilidade em incluir palavras de domínio específico que não estavam incluídas originalmente no vocabulário como uma forma de reduzir a segmentação de entidades em tokens menores. Os autores verificaram um ganho de performance em REN ao realizar este procedimento.

Capítulo 3

Desenvolvimento

Dada e elaboração teórica exposta no capítulo anterior, iremos explicar neste capítulo o desenvolvimento do trabalho de conclusão e alguns experimentos realizados orientados pelos objetivos iniciais.

3.1 Metodologia

Durante o trabalho, comparamos a performance de alguns modelos baseados em BERT na tarefa de REN aplicada a textos jurídicos em português. Os modelos foram obtidos do Hugging Face e foram treinados utilizando o Dataset LenerBR.

3.1.1 *Dataset* LeNERBr

Uma das primeiras tarefas do trabalho foi uma análise estatística do conjunto de dados. Como mencionado em 1.1, a disponibilidade de conjunto de dados de REN em português é escassa, ainda mais conjuntos de domínios especialistas. Assim, não encontramos outro conjunto de dados em português de cunho jurídico para realizar uma comparação. O Dataset também foi obtido via HuggingFace e pode ser explorado em https://huggingface.co/datasets/lener_br

Como mencionado anteriormente, as etiquetas do conjunto seguem o padrão IOB, então cada entidade é simbolizada por duas etiquetas: uma que sinaliza o início da entidade e outra que sinaliza que a palavra está dentro de uma entidade, mas não é o seu início.

O conjunto apresenta treze etiquetas, representando sete entidades: "O", "B-ORGANIZACAO", "I-ORGANIZACAO", "B-PESSOA", "I-PESSOA", "B-TEMPO", "I-TEMPO", "B-LOCAL", "I-LOCAL", "B-LEGISLACAO", "I-LEGISLACAO", "B-JURISPRUDENCIA", "I-JURISPRUDENCIA".

A entidade "O"simboliza palavras que não se enquadram em nenhuma das outras doze categorias, e é útil para analisar se uma entidade foi identificada, independentemente se foi classificada corretamente. Naturalmente, ao etiquetarmos um texto, a maior parte das palavras não vão se enquadrar em um número reduzido de categorias pré definidas

pela equipe etiquetadora. Assim, é um problema comum na área de REN a presença de conjuntos de dados desbalanceados. Iremos discorrer sobre esse problema em 3.1.3

O conjunto de dados consiste de 10.395 frases anotadas, separadas em 7828 frases para treino, 1177 frases para validação e 1390 para teste, seguindo um esquema de separação de dados aproximadamente em 70%, /15%, 15%.

Um dos primeiros desafios encontrados no trabalho de conclusão de curso foi a adaptação do conjunto de dados para um formato que faça sentido para a tokenização utilizada pelo modelo. Uma entrada do conjunto de dados está representada no programa 3.1.

Programa 3.1 Representação de uma entrada no conjunto de dados

```
1   {
2     "id": "0",
3     "ner_tags": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0,
          0],
4     "tokens": [
5       "EMENTA", ":", "APELAÇÃO", "CÍVEL", "-", "AÇÃO", "DE", "INDENIZAÇÃO", "
            POR", "DANOS", "MORAIS", "-", "PRELIMINAR", "-", "ARGUIDA", "PELO", "
            MINISTÉRIO", "PÚBLICO", "EM", "GRAU", "RECURSAL"]
6   }
```

Acima, praticamente todas entidades estão etiquetadas como "O", exceto por "MINISTÉRIO"e "PÚBLICO", etiquetadas como "B-ORGANIZAÇÃO"e "I-ORGANIZAÇÃO". Como mencionado em 2.5.7, O BERT usa tokenização sub-word, ou seja, foi necessário retokenizar o dataset inteiro. Um dos primeiros experimentos realizados foi entender se a retokenização realmente seria necessária e a conclusão foi de que, sem a retokenização, mais de 23% dos tokens são classificados como ‘[UNK]‘ e perdidos.

3.1.2 Estrategia de retokenização

Neste trabalho, decidimos seguir a mesma estratégia que os autores em Devlin *et al.*, 2019 adotaram para REN: para cada palavra w anotada com etiqueta l e tokenizada em $w_1, w_2, \ldots, w_t$ *subtokens*, etiqueta-se w_1 - referido como *subtoken* titular - com l, $w_2, \ldots w_t$ recebem uma etiqueta especial ‘IGNORE‘ que diz ao modelo para ignorar quaisquer predições para esses tokens: isto é, ao calcular métricas de acurácia, o modelo não deve considerar acertos ou erros de etiquetagem para qualquer *subtoken* exceto para o titular. A implementação desse processo foi exposto na sub sessão 3.1.6.

Essencialmente, essa estratégia permite ao modelo utilizar informações contextuais ao redor do *subtoken* titular mas ignora predições feitas para os outros *subtokens*. Outras estratégias comuns consistem em simplesmente estender a etiqueta original sobre os *subtokens*, mas decidimos utilizar a estratégia exposta acima por ter sido proposta pelos autores do artigo que introduziu o modelo utilizado.

3.1.3 Análise do conjunto de dados

Devido ao uso de auto-atenção, o BERT deve receber, como parâmetro, o tamanho máximo das sentenças de entrada. Por padrão, o modelo processa sentenças de, no máximo, 512 tokens. Como processamos dados em lote, é importante que todos dados de entrada do modelo tenham o mesmo formato, para otimizar os recursos do processamento paralelo de execução em GPU. A definição de tamanho máximo de sentença tem duas consequências:

- Sentenças com mais tokens do que a quantidade máxima são cortadas ao atingirem o limite de tokens.
- Sentenças com menos tokens do que o máximo recebem o token PAD sem significado semântico para padronizar o formato do lote.

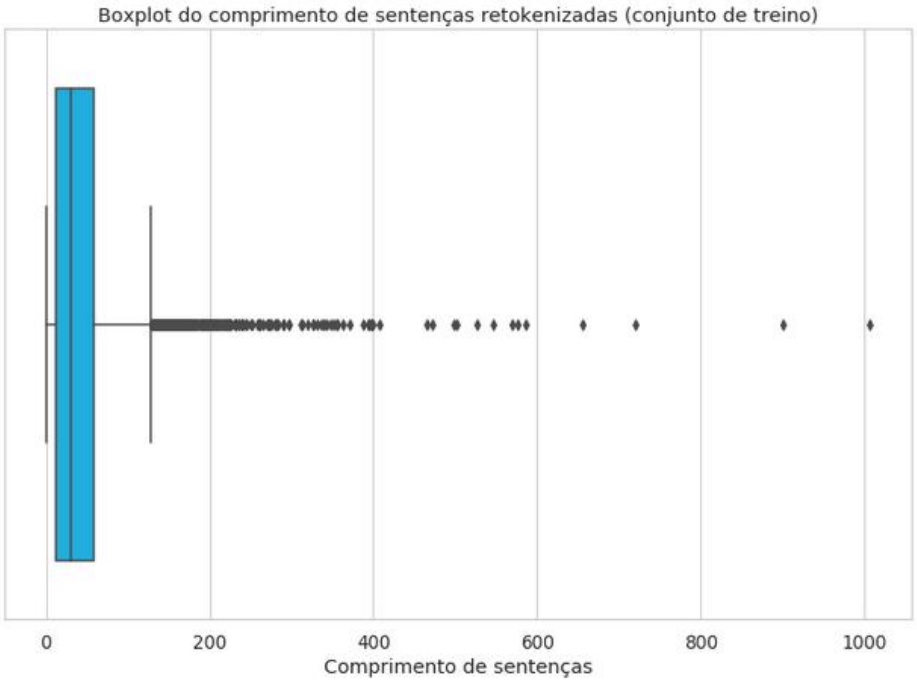
Assim, foi realizada uma análise do conjunto de dados para entender a quantidade média de tokens por frase após retokenizadas para definir o hiper parâmetro de tamanho máximo de sentenças ao BERT, evitando a alocação de cabeças de atenção demasiadamente largas (criando um gargalo em memoria) e evitando que muita informação seja perdida por definir um tamanho máximo muito pequeno.

Para analisar o número de *tokens* no conjunto de dados, foi realizado uma retokenização no conjunto de treino, teste e validação e plotamos histogramas para analisarmos a distribuição do comprimento de sentenças em número de *tokens* nos tres conjuntos. O resultado pode ser visto, respectivamente, nas figuras 3.1, 3.3 e 3.2.

Os histogramas já mostram que sentenças com mais de 200 tokens são *outliers* para os três conjuntos. O conjunto de validação e teste possuem uma distribuição de sentenças mais abrangente, enquanto que o conjunto de treino apresenta uma grande quantidade de dados *outliers*. Tentando aproveitar dados *outliers*, mas de comprimento razoável, analisamos qual o quantil 99% do conjunto de treino e concluimos que mais de 99% das sentenças possuem menos de 223 *tokens* no conjunto de treino, quando retokenizadas. Escolhemos, então, definir 256 como o tamanho máximo de sentença que o modelo: esse valor foi escolhido tanto pela análise dos dados quanto pelo fato de ser exatamente metade da quantidade padrão de *tokens* que o BERT processa.

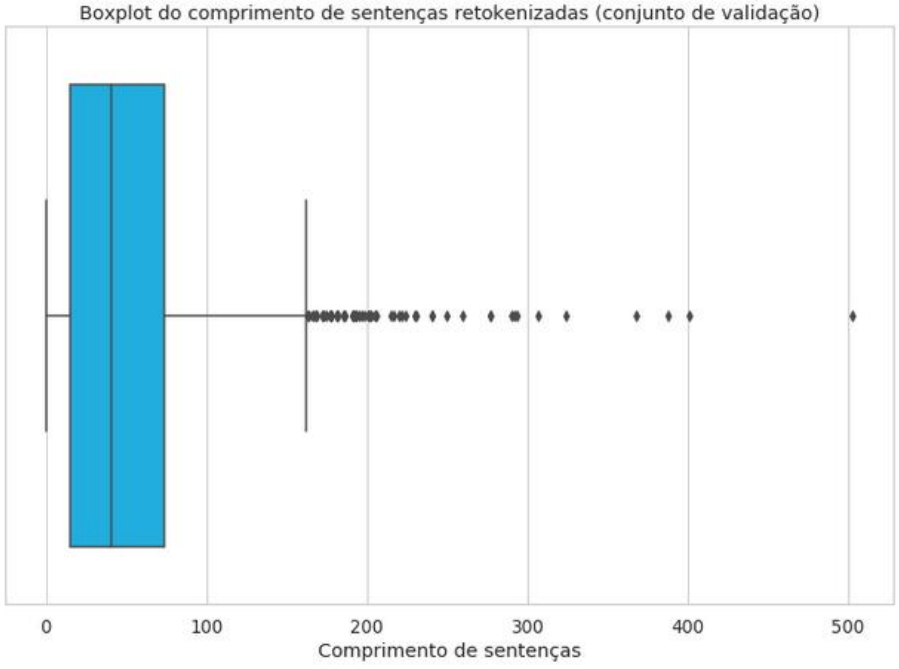
Uma outra análise feita foi entender a distribuição de etiquetas no conjunto de dados pois a distribuição de etiquetas pode afetar diretamente a validade do uso de certas métricas de avaliação de modelo, como acurácia: em conjuntos de dados desbalanceados (isto é, no qual a presença de uma etiqueta é maior do que de outras), a análise de performance do modelo deve ser feita com mais cuidado.

Na figura 3.4 podemos notar como o conjunto de dados é extremamente desbalanceado, sugerindo que, de fato, o uso de métricas como F1 sejam interessantes para avaliar um modelo treinado nesses dados. Também analisamos a ocorrência de etiquetas no conjunto de dados excluindo a etiqueta "O", visando entender a distribuição de *tokens* com uma entidade definida: Na figura 3.5 podemos ver que as outras etiquetas se distribuem de maneira ligeiramente desbalanceadas. Uma estratégia para mitigar o impacto de um conjunto de dados desbalanceado seria remover uma quantidade dos dados etiquetados que foram observados em excesso. No entanto, por se tratar de um modelo de linguagem neural



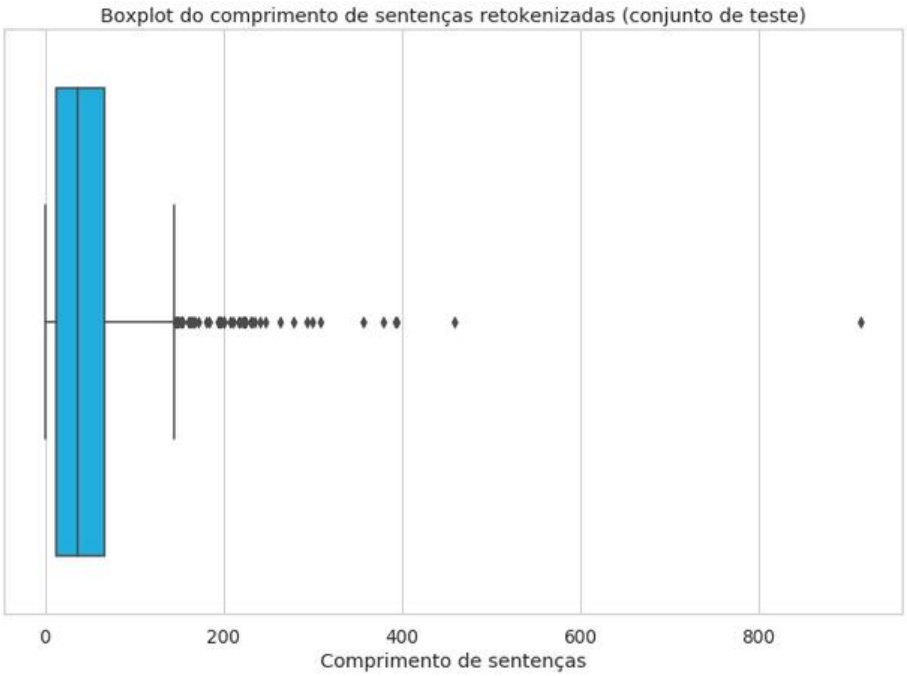
.

Figura 3.1: *Boxplot do número de tokens por sentença no conjunto de treino*



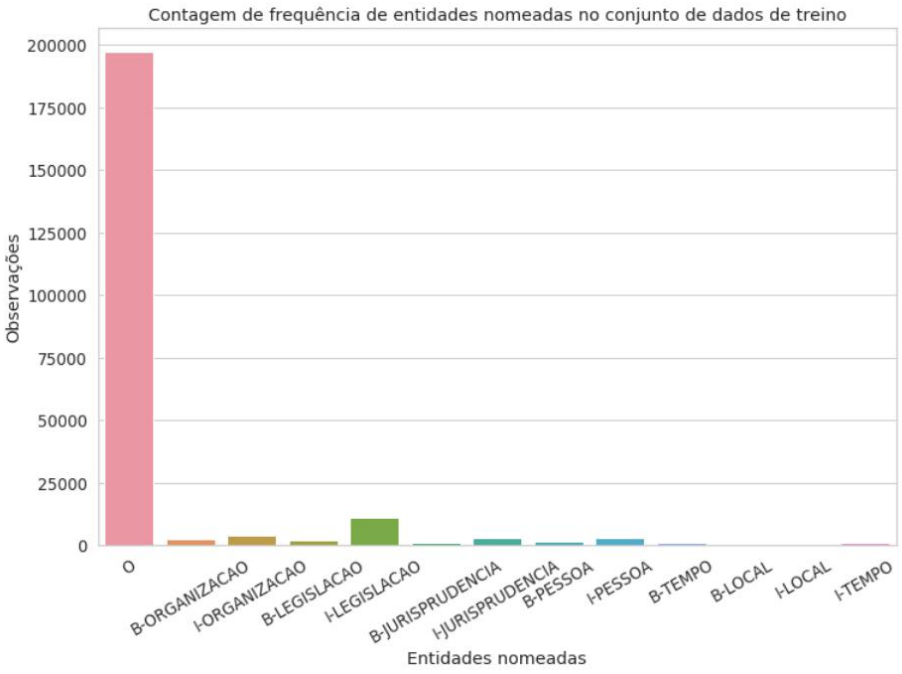
.

Figura 3.2: *Boxplot do número de tokens por sentença no conjunto de validação*



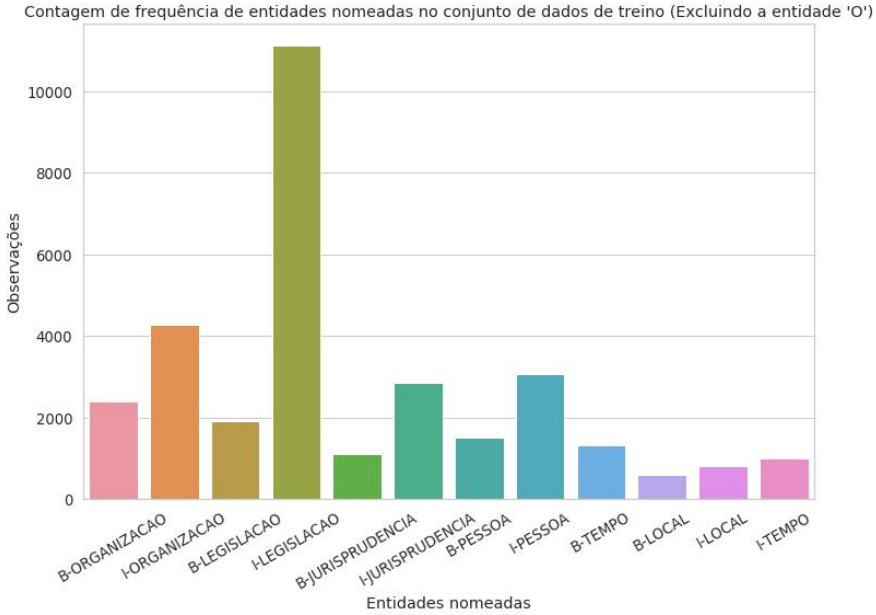
.

Figura 3.3: *Boxplot do número de tokens por sentença no conjunto de teste*



.

Figura 3.4: *Countplot das etiquetas do conjunto de dados de treino*



.

Figura 3.5: *Countplot das etiquetas do conjunto de dados de treino, excluindo a etiqueta "O".*

conexionista, a remoção de palavras de uma sentença iria impactar a capacidade do modelo para compreender uma frase, logo, impactaria a performance na tarefa de REN. Assim, para lidar com um conjunto de dados desbalanceados, analisamos a performance do modelo em F1.

3.1.4 Bibliotecas do *Hugging Face*

Hugging Face é um portal de modelos e conjuntos de dados, similar ao github mas voltado a soluções baseadas em transformadores e aplicações em PLN e Visão computacional. O uso desse portal foi fundamental nesse trabalho para o acesso a conjuntos de dados, tokenizadores e modelos pré treinados. Estes podem ser acessados por meio das biblioteca transformers e datasets via Python3. Os modelos disponibilizados são implementados ou em PyTorch, ou em Tensorflow, duas bibliotecas populares para aprendizado profundo em Python. Neste trabalho utilizamos apenas modelos implementados com PyTorch.

3.1.5 Linha de produção de treino

Um dos objetivos deste trabalho de conclusão foi a elaboração de um linha de produção (*pipeline*) de treino a ser utilizado futuramente para cumprir a motivação inicial. Um dos maiores desafios constatados ao longo deste trabalho foi a dificuldade de lidar com dados etiquetados palavra a palavra enquanto que usamos uma *tokenização* a nível sub-palavra. Assim, um dos pontos de interesse ao escrever um pipeline de treino foi de esconder as complexidades de adaptar o conjunto de dados ao modelo.

Além disso, ao escrevermos o código de treino de modelo, tivemos o objetivo de torna-lo o mais agnóstico possível quanto ao dado alimentado e qual modelo pré treinado escolhido. Assim, garantimos a facilidade de reaproveitamento de código. Modelos pré treinados são

referido como ponto de controle(ou *checkpoint*) no contexto da biblioteca *Transformers*. Um checkpoint nada mais é do que uma string que representa um caminho no Hugging Face. Utilizando um checkpoint é possível baixar modelos pré treinados facilmente.

O pipeline de código abrange três abstrações no processo de treino de um modelo: a adaptação do conjunto de dados, o modelo em si e o treino. Cada abstração é implementada em um módulo Python e exposta por uma classe

3.1.6 Adaptação do conjunto de dados

A adaptação do conjunto de dados é feita no módulo `preprocess_dataset.py` e exposta pela classe `NERDataset`. Essa classe recebe um *Dataset Pytorch* separado em conjuntos de treino, teste e validação. A estrutura de cada entrada no conjunto foi abordada em 3.1.1. A classe remodela o conjunto de dados de entrada para um formato conveniente mas mantendo a assinatura de um *Dataset Pytorch*, pois este formato é conveniente para realizar treinos em lotes. Para garantirmos a imutabilidade de tipo apesar das transformações internas, a classe `NERDataset` implementa os métodos `__init__`, `__len__ e __getitem__` da classe mãe `torch.data.utils.Dataset`;. Os dois primeiros são simples e apenas são responsáveis por inicializar um objeto da classe e determinar o seu comprimento. O método `__getitem__` é responsável por implementar como um item é recuperado do conjunto de dados e nele concentramos a maior parte da adaptação do dado ao formato mais conveniente para o BERT. Cada item retornado é um dicionário contendo um identificador único da entrada, a lista de *token* ids, a máscara de atenção e o vetor de etiquetas.

Ao garantirmos que `NERDataset` implementa a classe `torch.data.utils.Dataset`, garantimos que um modelo pode utilizar qualquer uma das duas classes sem precisar ser refatorado pelo fato de ambos apresentarem o mesmo contrato de interface.

A preparação do conjunto de dados é resumida em três etapas:

1. Preprocessamento
2. Retokenização
3. Adequação das etiquetas aos *subtokens*

O preprocessamento consistiu na remoção de sentenças vazias no conjunto de dados. Apesar de estar localizado no arquivo `preprocess_dataset.py`, as funções de preprocessamento não estão dentro da classe NERDataset para respeitar o princípio de responsabilidade única.

Como elaborado em 2.5.8, um etiquetador humano classifica cada palavra de um texto em uma entidade nomeada ao criar um conjunto de dados. No entanto, o BERT espera palavras tokenizadas seguindo o método introduzido por Schuster e Nakajima, 2012. Esse processo de adequação foi abordado na etapa de retokenização e adequação de etiquetas.

O processo de retokenização consistiu em, primeiramente, alterar a representação de uma frase: no conjunto de dados, cada frase é uma lista de *tokens*. Tornamos essa lista em uma única *string*, unindo cada *token* por um caracter de espaço. Após alterar a

representação de uma frase, aplicamos o tokenizador do BERT à *string*, obtendo, assim, uma nova segmentação da frase mas agora em *subtokens*.

Por fim, o processo de adequação das etiquetas aos *subtokens*. Como mencionado em 2.5.8, os autores em Devlin *et al.*, 2019 escolheram analisar apenas os titulares de cada entidade ao treinar o BERT para REN, ignorando todos outros *subtokens*, isto é, aqueles que começam com "##". Ao mesmo tempo, gostaríamos de manter os *subtokens* pois eles possuem valor semântico contextual, e remove-los diminuiria a capacidade de modelagem de linguagem do modelo. Estes dois pontos são conciliados ao simplesmente ignorarmos as classificações que o modelo faz para os *subtokens* não titulares ao calcularmos a perda e medidas de precisão do modelo.

Como utilizamos modelos baseados em PyTorch, pudemos fazer uso de algumas implementações da biblioteca relevantes para esse objetivo: ao calcular o erro entre um tensor de saída e um tensor de etiquetas, a biblioteca permite mascarar certas etiquetas que devem ser ignoradas. Isto é feito utilizando um valor especial que, por padrão, é -100. O programa 3.2 é responsável por implementar esse processo.

Programa 3.2 Trecho de código responsável por marcar tokens irrelevantes para métricas de avaliação.

```
1    special_tokens = ['[CLS]', '[SEP]', '[MASK]', '[PAD]']
2    input_ids = encoded_input['input_ids']
3    i = 0
4    encoded_labels = np.ones(self.max_len, dtype=int) * -100
5    for indx, input_id in enumerate(input_ids.squeeze()):
6       token = self.tokenizer.convert_ids_to_tokens(input_id.item())
7       if not token.startswith('##') and not token in special_tokens:
8          encoded_labels[indx] = labels[i]
9          i += 1
```

O trecho de código acima cria um vetor de etiquetas adaptadas chamado `encoded_labels` de tamanho `self.max_len` preenchido com o valor -100, que deverá ser ignorado internamente pelo modelo. Utilizamos um vetor de tamanho `self.max_len` pois, como explicado em 3.1.3, processamos dados em lote então todas entradas do modelo precisam possuir um tamanho padronizado. As etiquetas, por sua vez, necessariamente possuem o mesmo tamanho que o dado etiquetado.

O *script*, então, converte cada id para um token para analisar quais *tokens* devem ser considerados: apenas sobrescrevemos o valor -100 com a label do *token* se ele for titular. Do contrário, ou seja, se for um sub*token* ou um *token* especial, será ignorado. Para uma explicação sobre o que cada token especial representa, o tópico é discorrido brevemente na sub seção 2.5.6.

3.1.7 Modelo

A entidade de modelo é implementada em `model.py` na classe `NERClassifier`, no programa 3.3. Herdamos da classe `nn.Module`. Essa classe é a base para implementações de Redes Neurais baseadas em PyTorch. Classes filhas de `nn.Module` apresentam no mínimo

Programa 3.3 Implementação da entidade modelo em model.py

```
1   from torch import nn
2   from transformers import BertForTokenClassification
3
4   import torch
5
6   class NERClassifier(nn.Module):
7      def __init__(self, n_labels, checkpoint):
8         super(NERClassifier, self).__init__()
9         self.bert = BertForTokenClassification.from_pretrained(checkpoint,
              num_labels=n_labels)
10
11     def forward(self, ids, mask, labels=None):
12        return self.bert(ids, mask, labels=labels)
```

dois métodos: `__init__`, onde as camadas da rede neural são definidas, e `forward`, aonde o *forward pass* é definido, isto é, como um dado de saída é calculado utilizando o dado de entrada e as camadas da rede. No nosso caso, aplicamos uma única camada: um BERT especializado em REN. Internamente, este tipo de BERT é, na verdade, duas camadas: o modelo pré treinado e uma cabeça especializada em classificação token a token.

O modelo recebe um *checkpoint* de modelo pré treinado e o número de etiquetas usadas na tarefa de REN. Com isso, o construtor instancia um modelo pré treinado BERT e acopla uma camada de classificação de REN em `n_labels` categorias. Utilizamos a classe `BertForTokenClassification` da biblioteca *transformers* para instanciar o modelo com a cabeça de REN. A classe também define um método `forward`, que implementa o funcionamento da passagem de dados pelo modelo. Neste caso, alimentamos o modelo BERT com os ids de entrada, a mascara de atenção e as etiquetas. A máscara de atenção foi passada para informar ao modelo de linguagem quais *tokens* devem ser ignorados. As etiquetas foram passadas pois, segundo a API fornecida pela biblioteca *transformers*, isso informa ao modelo para retornar o cálculo de *loss* para a entrada, segundo as etiquetas. O método `forward` retorna então uma tupla composta pela predição do modelo e o *loss* correspondente.

3.1.8 Treino

O treino do modelo é realizado no arquivo `trainner.py` exposto pela classe Trainner,. A classe recebe no seu construtor o modelo, um *DataLoader*, o dispositivo no qual será realizado o treino (por exemplo, GPU) e alguns hiper-parâmetros: otimizador, tamanho do lote, número de exemplos e número de epocas.

O treino é implementado nos método `train` e `_train_epoch`. O primeiro método roda o treino para o número de epocas especificadas, utilizando o segundo método como auxiliar. Calculamos *loss* e métricas de acurácia para cada epoch de treino. O calculo de *loss* é feito internamente pelo modelo, então os tokens marcados com -100 são automaticamente ignorados. Este processo precisou ser manualmente implementado para o cálculo de pontuações de acurácia. Para tanto, utilizamos a função `torch.masked_select` que recebe um tensor alvo e um tensor de booleanos, e seleciona apenas as entradas nas quais o tensor

booleano apresenta o valor `True`. O processo foi implementado no programa 3.4.

Programa 3.4 Trecho de código para filtragem de *outputs* e *targets* pertinentes aos *tokens* relevantes para métricas de avaliação

```
1    for idx, sample in enumerate(self.dataLoader):
2       target = sample["targets"].to(self.dev, dtype = torch.long)
3       out = self.model(input_tensor, att_mask, labels=target)
4       logits = out['logits']
5       loss = out['loss']
6
7       flat_targets = target.view(-1)
8       flat_logits = logits.view(-1)
9       active_accuracy = flat_targets != -100
10      relevant_labels = torch.masked_select(flat_targets, active_accuracy)
11      relevant_logits = torch.masked_select(flat_logits, active_accuracy)
```

Achatamos o tensor de target pois estamos lidando com treino em batches, então cada tensor de target possui formato `[batch_size, max_len]`. O mesmo processo é feito para as predições do modelo. Dessa maneira, aproveitamos o valor contextual de *subtokens* mas ignoramos as predições para estes *tokens*

3.2 Experimentos

3.2.1 Refinamento de um BERTimbau em REN

Utilizando o pipeline elaborado em 3.1.5 fomos capazes de refinar um modelo baseado em BERTimbau na tarefa de REN. Os experimentos de refinamento foram realizados numa máquina 64gb de memória RAM, um processador i7 3.2ghz com 16 núcleos e uma GPU GeForce GTX 1070 com 4gb de memória interna. Recomendamos fortemente o uso de máquinas com GPU para reproduzir os experimentos de refinamento. É recomendado, também, reproduzir os experimentos em máquinas com pelo menos 4gb de RAM, devido ao grande espaço ocupado em memória pelo modelo e carregadores de dados. Na falta de uma máquina com configurações compatíveis, recomenda-se o uso de máquinas collab com GPU disponibilizadas pelo Google em https://research.google.com/colaboratory/.

Ao refinar o modelo, estivemos mais interessados em validar a abordagem com o pipeline de treino e explorar desafios de refinar um modelo em REN do que otimizar um modelo para estado de arte. Acreditamos que tal objetivo foge ao escopo do trabalho, mas o pipeline pode ser facilmente utilizado para a tarefa de otimização de hiper parâmetros.

Em Devlin *et al.*, 2019, os autores sugerem um conjunto de hiper parâmetros para refinar o BERT:

- Taxa de aprendizagem: $3 * 10^{-4}, 10^{-4}, 5 * 10^{-5}, 3 * 10^{-5}$
- Otimizador: AdamW, Adam, SGD
- Tamanho do batch: 8, 16, 32, 64, 128

Utilizamos taxa de aprendizagem $3 * 10^{-4}$, AdamW como otimizador e 8 exemplos de treino como tamanho de lote. Treinamos o modelo por 8 épocas e avaliamos a pontuação F1 do modelo a cada época para o conjunto de treino e teste.

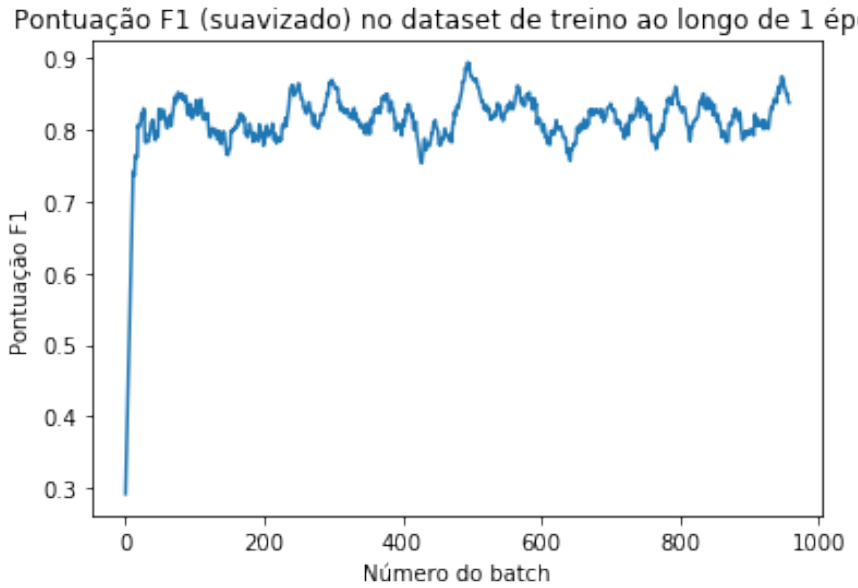


Figura 3.6: *Pontuação F1 do modelo em 1 época de treino.*

A figura 3.6 mostra uma evolução da pontuação F1 do modelo no conjunto de treino em uma época, suavizada por uma média móvel de 20 amostras. O que chamou atenção nesse experimento foi a rápida evolução do modelo. A pontuação F1 atinge um patamar de 0.8 em torno do décimo *batch*, tendendo a estabilizar nesse valor apesar de oscilações pontuais. A rápida evolução do modelo se deve principalmente a dois fatores:

1. Grande poder computacional do BERT: devido ao seu alto poder computacional, o modelo é capaz de aprender algum padrão de maneira rápida.

2. Tamanho do *batch*: cada *batch* possui 8 frases, e cada frase apresenta em torno de 128 *tokens* etiquetados. Assim, cada *batch* apresenta grande número de exemplos de treino.

Nota-se também que a performance do modelo torna-se estagnada. Apesar de existir um grande número de exemplos de treino por *batch*, a variedade é baixa: como mostramos no gráfico 3.4, a distribuição do conjunto de dados é extremamente desbalanceada para a etiqueta "O", sugerindo que o modelo apresenta *overfit* para esta etiqueta e não consegue aprender muito para as outras etiquetas.

Este fato e a dispersão apresentada no gráfico 3.5 sugeram que talvez possa ser vantajoso fazer uma amostra do conjunto de treino pequena para o modelo aprender a identificar etiquetas "O"e, posteriormente, analisar o resto do conjunto de dados de treino mas ignorar a etiqueta "O", forçando o modelo a aprender outras etiquetas.

Visando explorar a hipótese do modelo precisar de mais épocas para aprender, também treinamos e avaliamos o modelo em 8 épocas.

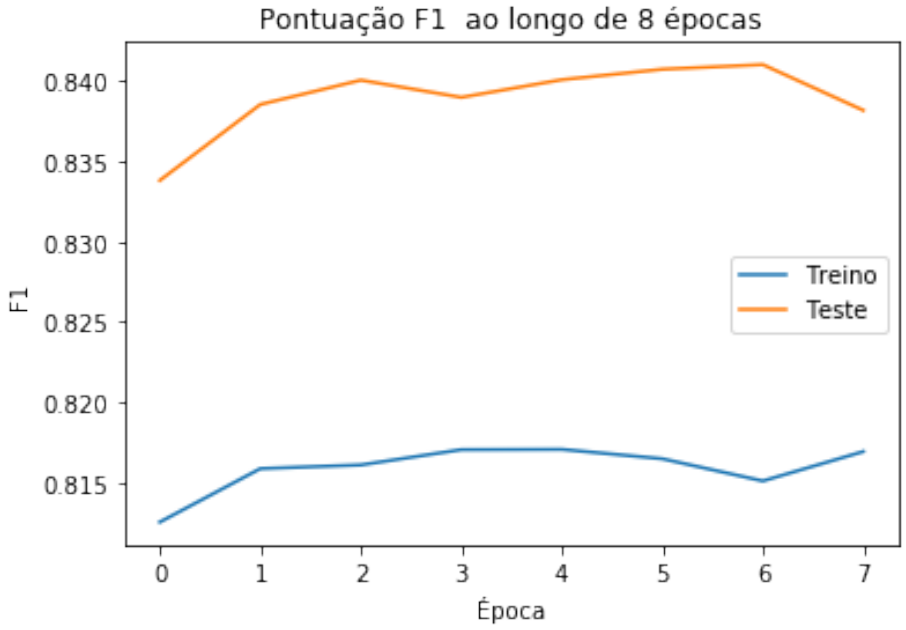


Figura 3.7: *Pontuação F1 do modelo avaliado no conjunto de treino e teste ao longo de 8 épocas.*

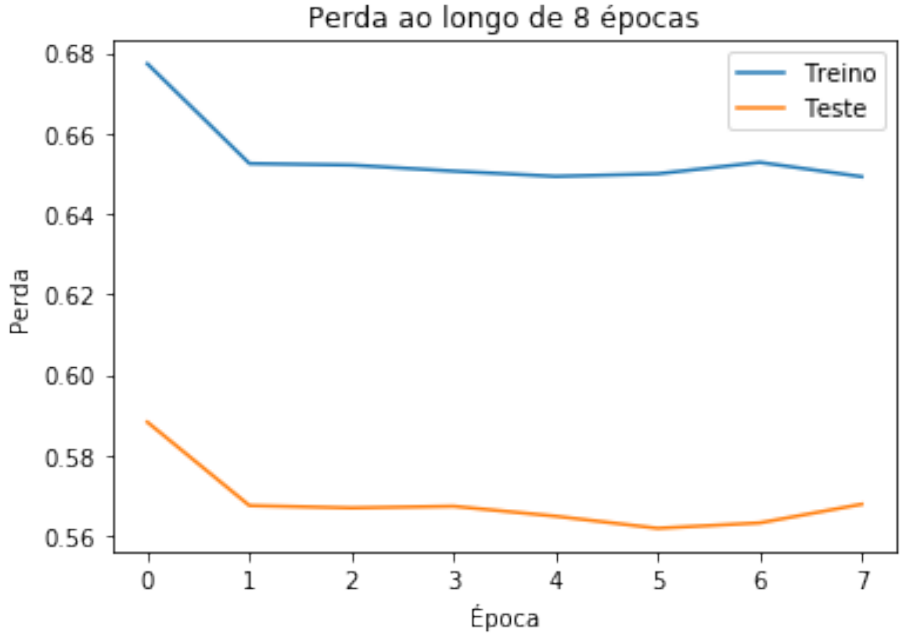


Figura 3.8: *Perda do modelo avaliado no conjunto de treino e teste ao longo de 8 épocas*

Os gráficos 3.7 e 3.8 mostram a pontuação F1 e perda em 8 épocas. Note que não há diferença significativa ao longo dos treinos. Isso sugere que o modelo de fato satura.

Também foi realizada uma depuração extensiva do código de treino do modelo verificando possíveis erros de codificação no aprendizado, mas nenhum erro foi encontrado, mostrando que o problema é de fato de generalização dos dados.

Um fato curioso constatado foi um comportamento consistente de performance superior no conjunto de teste em relação ao conjunto de teste. Como a avaliação é feita logo após uma passada de treino, a primeira hipótese abordada foi de que o modelo estava, de fato, treinando no conjunto de teste. No entanto, o ganho de performance para uma mesma época entre conjuntos de dados é superior entre o ganho de performance por treino entre duas épocas no conjunto de teste. Esse fato sugere, na verdade, que o conjunto de dados não está igualmente distribuído entre treino e teste, sugerindo que para REN pode ser valioso re-sortear amostras entre os conjuntos, visando uma distribuição igual de etiquetas entre os conjuntos.

3.2.2 Comparação de tamanhos de modelo

Outro experimento explorado foi de comparar diferentes tamanhos de modelo. Como elaborado na sessão 3.1.3, o BERT é um modelo que analisa uma porção fixa de uma frase e, analisando o conjunto de dados, definimos 256 como tamanho do modelo, devido à distribuição de tamanho de sentenças no conjunto.

Decidimos analisar a performance do modelo se escolhermos modelos ainda menores. Sabemos, pela sessão 3.1.3, que não estamos perdendo informação ao definirmos o tamanho máximo em 256, mas não sabíamos, nesse ponto, se um modelo mais enxuto obteria uma performance superior.

Foi realizada uma etapa de treino em uma época para os seguintes tamanhos máximos de entrada: 64, 128 e 256. Os resultados estão relatados nas figuras 3.9 e 3.10.

Nota-se pouca diferença de performance do modelo entre diferentes tamanhos de modelo, com vantagem ligeira do modelo de tamanho 256. Por um lado, isso sugere que podemos utilizar modelos menores para o treino, o que geraria menos carga durante o processo de treino. Por outro, o modelo, seguindo a configuração e metodologia de treino adotados, não está apresentando uma performance ideal, sugerindo que esse experimento pode retornar um resultado mais confiável se repetido corrigindo os problemas de treino apontados neste texto.

3.2.3 Comparação de diferentes pontos de verificação

O último experimento realizado foi o de comparar diferentes *checkpoints* disponíveis no *Hugging Face*. Comparou-se os seguintes *checkpoints*:

1. Bertimbau base cased:
2. Bertimbau Large cased
3. Bert Base Cased

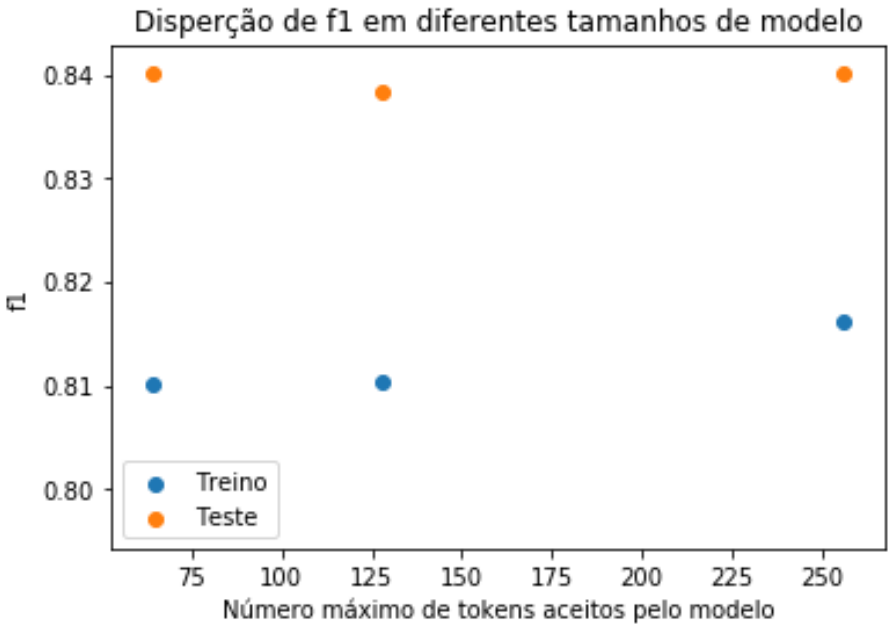


Figura 3.9: *Pontuação f1 em conjunto de teste e treino com diferentes tamanho de modelo*

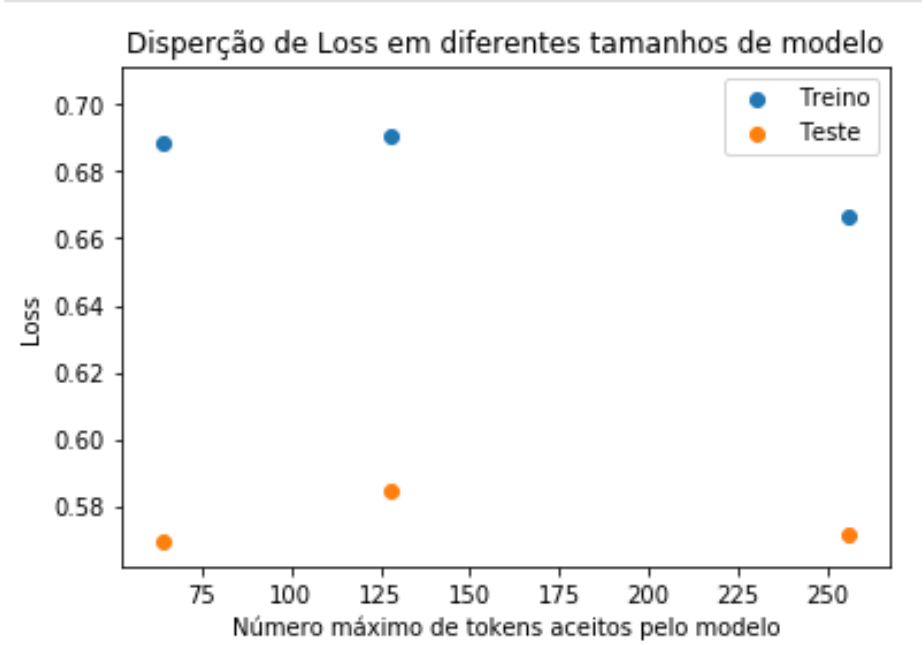


Figura 3.10: *Perda em conjunto de teste e treino com diferentes tamanho de modelo*

4. Bert Base Uncased
5. Bert Large Cased
6. Bert Large Uncased

Cada variante foi treinada em uma época no conjunto de treino e avaliada no conjunto de teste.

Com essa comparação, esperava-se verificar uma performance superior de modelos em português devido ao pré treino em textos em português, providenciando melhor modelagem de linguagem.

Modelos *Cased* também apresentam, segundo a literatura, performance superior em REN, já que o uso de letras maiúsculas pode definir a classificação de uma palavra em uma entidade.

Como explicado em 2.5.1, os modelos Base apresentam *110 milhões de parâmetros enquanto que os modelos *large* apresentam 340 milhões de parâmetros. Apesar da versão *Large* apresentar maior potencial de aprendizado, não nos interessamos em comparação de performance entre as variantes de tamanho de modelo, já que treinamos cada variante em uma única época e os modelos *large*, devido ao seu tamanho, podem precisar de mais tempo de treino para atingir uma performance ideal.

Somente a variante *cased* do BERTimbau estava disponível no *Hugging Face* durante o período de elaboração de experimentos deste trabalho, assim, não analisamos uma versão do BERT em português que não diferencie maiúsculas de minúsculas.

Os resultados estão representados na tabela 3.1

Checkpoint	F1 em treino	F1 em teste	Perda em treino	Perda em Teste
Bert Base Portuguese Cased	0,814	0,816	0,666	0,661
Bert Large Portuguese Cased	0,812	0,816	0,691	0,660
Bert Base Cased	0,811	0,814	0,681	0,668
Bert Base Uncased	0,807	0,813	0,699	0,662
Bert Large Uncased	0,807	0,813	0,708	0,664
Bert Large Cased	0,811	0,812	0,692	0,660

Tabela 3.1: *Comparação de performance de diferentes checkpoints do BERT no conjunto de teste e treino*

O resultado mais interessante observado é como todos *checkpoints* apresentam uma performance similar em REN, independentemente da linguagem de pré treino. Os modelos em português apresentam uma performance ligeiramente superior aos modelos em inglês, o que acompanha as conclusões de Souza *et al.*, 2020 mostrando a superioridade do BERTimbau em relação ao BERT em diversas tarefas, inclusive NER.

Em conjunto com a observação de estagnação de aprendizado do modelo, tomamos como possível hipótese de que os modelos apresentam performance similares pois a linguagem do texto é suficientemente distinta do português e do inglês, impedindo que o BERTimbau atinja o seu potencial desempenho.

Como explorado em 3.1.2, o processo de retokenização é fundamental ao trabalhar com REN e quanto mais específico o vocabulário de um tokenizador a uma língua, menos tokenizações são necessárias. Assim, palavras jurídicas - que ,pela sua natureza específica, são desconhecidas ao tokenizador do BERTImbau e do BERT - são excessivamente tokenizadas em diversos tokens. Em 3.1.2 tomamos como estratégia de retokenização tomar o primeiro subtoken como titular. Assim, um termo jurídico - excessivamente tokenizado - acaba recebendo como representante um token pequeno, possivelmente um radical muito comum. Assim, a informação disponível em um único token cai, minando a capacidade preditiva do modelo.

Os autores em Khan, 2021 estudaram técnicas de REN em textos jurídicos, nele propõe-se duas estratégias para aumentar a performance do sistema devido à natureza específica da linguagem do texto:

1. Expandir o vocabulário do tokenizador com termos específicos de natureza jurídica, reduzindo a quantidade de tokenizações feitas por termo
2. Retreinar o modelo utilizando textos jurídicos

Como o processo de retreinar o modelo é extremamente custoso computacionalmente, convém-se em trabalhos futuros expandir o vocabulário do modelo e avaliar sua performance.

Capítulo 4

Conclusão

Este trabalho se propunha a contextualizar de maneira didática o uso de transformadores em PLN, em específico com o modelo BERT, apresentando desafios de refinar em REN e possíveis abstrações que facilitariam o refinamento em tal tarefa. O objetivo tinha tanto um papel didático quanto prático, tendo em vista a mudança de foco dessa monografia desde o início da sua elaboração. Por um lado, a documentação e contextualização de técnicas e modelos necessários para compreender as inovações introduzidas pelo BERT pode auxiliar um aluno interessado no tópico. Por outro, a elaboração de um *pipeline* de treino que abstrai incompatibilidades entre a metodologia de etiquetagem de dados e a técnica de tokenização do modelo poderia auxiliar pesquisadores interessados em elaborar um sistema etiquetador de expressões sexistas em textos judiciais.

Ao longo da monografia, apresentamos um panorama de modelos de linguagem e soluções de PLN neurais, fundamentando o necessário para um leitor interessado se aprofundar em soluções baseadas em Transformadores. Também elaboramos sobre desafios do uso de modelos baseados em BERT para REN, tanto pelas diferenças em metodologia de etiquetagem e de tokenização quanto pela escassez de dados.

Propusemos também um *pipeline* de dados que abstrai as dificuldades expostas, fazendo uso da API do *HuggingFace* para facilitar instanciação do modelo pré treinado. No processo, documentamos alguns desafios e dores ao se trabalhar com BERT para reconhecimentos de entidades.

Um dos maiores desafios abordados no trabalho foi o de trabalhar com um conjunto de dados desbalanceado, como é frequente em REN. Notamos que um modelo BERT treinado em REN satura seu conhecimento rapidamente e possivelmente apresenta *overfitting* em uma das categorias. Uma solução proposta foi de quebrar o treino em duas etapas, onde a segunda etapa não apresenta dados com a etiqueta da categoria desbalanceada.

Constatou-se neste trabalho que a linguagem usada no texto jurídico é suficientemente diferente de português e inglês para impactar a performance de modelos BERT pré treinado nas duas linguas, demandando, possivelmente, técnicas de normalização e processamento de texto para permitir o modelo em português uma performance superior, como constatado em Souza *et al.*, 2020. Alternativamente, pode-se seguir soluções propostas por Khan, 2021 e expandir o vocabulário do tokenizador para incluir palavras jurídicas.

Por fim, espera-se que as conclusões e principalmente as dificuldades expostas neste trabalho sirvam como auxílio para que pesquisadores possam, no futuro, retomar o objetivo inicial de pesquisa deste trabalho de conclusão de curso.

Referências

[Alammar 2018] Jay Alammar. *The Illustrated Transformer*. 2018. URL: https://jalammar.github.io/illustrated-transformer/ (citado nas pgs. 18, 19).

[Bahdanau *et al.* 2016] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL] (citado na pg. 16).

[Beltagy *et al.* 2019] Iz Beltagy, Kyle Lo e Arman Cohan. *SciBERT: A Pretrained Language Model for Scientific Text*. 2019. arXiv: 1903.10676 [cs.CL] (citado na pg. 1).

[Bengio *et al.* 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent e Christian Janvin. "A neural probabilistic language model". Em: *J. Mach. Learn. Res.* 3 (mar. de 2003), pgs. 1137–1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966 (citado nas pgs. 2, 10–12).

[Bojar *et al.* 2018] Ondřej Bojar *et al.* "Findings of the 2018 conference on machine translation (wmt18)". Em: jan. de 2018, pgs. 272–303. DOI: 10.18653/v1/W18-6401 (citado na pg. 15).

[Caselli *et al.* 2021] Tommaso Caselli, Valerio Basile, Jelena Mitrović e Michael Granitzer. *HateBERT: Retraining BERT for Abusive Language Detection in English*. 2021. arXiv: 2010.12472 [cs.CL] (citado na pg. 4).

[Cho *et al.* 2014] Kyunghyun Cho *et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406 . 1078 [cs.CL] (citado nas pgs. 16, 18).

[Devlin *et al.* 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee e Kristina Toutanova. "Bert: pre-training of deep bidirectional transformers for language understanding". Em: (2019). arXiv: 1810.04805 [cs.CL] (citado nas pgs. 1, 15, 19, 21, 22, 24, 26, 32, 34).

[Freitas *et al.* 2010] Cláudia Freitas, Cristina Mota, Diana Santos, Hugo Gonçalo Oliveira e Paula Carvalho. "Second harem: advancing the state of the art of named entity recognition in portuguese." Em: jan. de 2010 (citado na pg. 2).

[Hochreiter e Schmidhuber 1997] Sepp Hochreiter e Jürgen Schmidhuber. "Long short-term memory". Em: *Neural computation* 9 (dez. de 1997), pgs. 1735–80. doi: 10.1162/neco.1997.9.8.1735 (citado nas pgs. 1, 14, 15).

[Junior *et al.* 2015] Carlos Junior, Hendrik Macedo, Flavio Oliveira Thiago Bispo, Nayara Silva e Luciano Barbosa. "Paramopama: a brazilian-portuguese corpus for named entity recognition." Em: 2015 (citado na pg. 2).

[Khan 2021] Muhammad Zohaib Khan. "Comparing the performance of NLP toolkits and evaluation measures in legal tech". Em: *CoRR* abs/2103.11792 (2021). arXiv: 2103.11792. url: https://arxiv.org/abs/2103.11792 (citado nas pgs. 24, 40, 41).

[Lample *et al.* 2016] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami e Chris Dyer. *Neural Architectures for Named Entity Recognition*. 2016. arXiv: 1603.01360 [cs.CL] (citado na pg. 2).

[Luz de Araujo *et al.* 2018] Pedro H. Luz de Araujo *et al.* "LeNER-Br: a dataset for named entity recognition in Brazilian legal text". Em: *International Conference on the Computational Processing of Portuguese (PROPOR)*. Lecture Notes on Computer Science (LNCS). Canela, RS, Brazil: Springer, set. de 2018, pgs. 313–323. doi: 10.1007/978-3-319-99722-3_32. url: https://cic.unb.br/~teodecampos/LeNER-Br/ (citado na pg. 3).

[McCormick 2016] Chris McCormick. *Word2Vec Tutorial - The Skip-Gram Model*. 2016. url: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ (citado na pg. 9).

[Mikolov *et al.* 2013] Tomas Mikolov, Kai Chen, Greg Corrado e Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL] (citado na pg. 8).

[Miller 1995] George A. Miller. "Wordnet: a lexical database for english". Em: *Commun. ACM* 38.11 (nov. de 1995), pgs. 39–41. issn: 0001-0782. doi: 10.1145/219717.219748. url: https://doi.org/10.1145/219717.219748 (citado na pg. 7).

[Peters *et al.* 2018] Matthew E. Peters *et al.* "Deep contextualized word representations". Em: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. url: http://arxiv.org/abs/1802.05365 (citado na pg. 22).

[Pimentel *et al.* 1998] Silvia Pimentel, Ana Lucia P. Schritzmeyer e Valeria Pandjiarjian. *Estupro:crime ou "cortesia"? Abordagem Sociojurídica de gênero*. Porto Alegre, S.a. Fabris, 1998 (citado nas pgs. 3, 4).

[Radford e Sutskever 2018] Alec Radford e Ilya Sutskever. "Improving language understanding by generative pre-training". Em: (2018) (citado na pg. 1).

[Ramshaw e Marcus 1995] Lance A. Ramshaw e Mitchell P. Marcus. "Text chunking using transformation-based learning". Em: *ArXiv* cmp-lg/9505040 (1995) (citado na pg. 2).

[Schuster e Nakajima 2012] Mike Schuster e Kaisuke Nakajima. "Japanese and korean voice search". Em: mar. de 2012, pgs. 5149–5152. ISBN: 978-1-4673-0045-2. DOI: 10.1109/ICASSP.2012.6289079 (citado nas pgs. 23, 31).

[Souza *et al.* 2020] Fábio Souza, Rodrigo Nogueira e Roberto Lotufo. "BERTimbau: pretrained BERT models for Brazilian Portuguese". Em: *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*. 2020 (citado nas pgs. 1, 21, 23, 39, 41).

[Tan *et al.* 2018] Chuanqi Tan *et al. A Survey on Deep Transfer Learning*. 2018. arXiv: 1808.01974 [cs.LG] (citado na pg. 3).

[Taylor 1953] Wilson L. Taylor. "Cloze procedure: a new tool for measuring readability". Em: (1953). eprint: https://doi.org/10.1177/107769905303000401 (citado na pg. 20).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". Em: (2017). arXiv: 1706.03762 [cs.CL] (citado nas pgs. 1, 16–18).

[Werbos 1990] Paul Werbos. "Backpropagation through time: what it does and how to do it". Em: *Proceedings of the IEEE* 78 (nov. de 1990), pgs. 1550–1560. DOI: 10.1109/5.58337 (citado na pg. 14).

[Zhu *et al.* 2015] Yukun Zhu *et al. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. arXiv: 1506.06724 [cs.CV] (citado na pg. 1).