BERT E WORD2VEC: UMA ANÁLISE INFERENCIAL E COMPUTACIONAL NA CLASSIFICAÇÃO DE TEXTOS COM REDES NEURAIS CONVOLUCIONAIS

Bernardo Cardoso Cordeiro

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Heraldo Luís Silveira de Almeida

Rio de Janeiro

Setembro de 2019

BERT E WORD2VEC: UMA ANÁLISE INFERENCIAL E COMPUTACIONAL NA CLASSIFICAÇÃO DE TEXTOS COM REDES NEURAIS CONVOLUCIONAIS

Bernardo Cardoso Cordeiro

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO

Autor:

Bernardo Cardoso Cordeiro

Orientador:

Prof. Heraldo Luís Silveira de Almeida, D.Sc.

Examinador:

Prof. João Baptista de Oliveira e Souza Filho, D.Sc.

Examinador:

Prof. Natanael Nunes de Moura Junior, D.Sc.

Rio de Janeiro

Setembro de 2019

Declaração de Autoria e de Direitos

Eu, Bernardo Cardoso Cordeiro CPF 149.389.157-07, autor da monografia BERT E WORD2VEC: UMA ANÁLISE INFERENCIAL E COMPUTACIONAL NA CLASSIFICAÇÃO DE TEXTOS COM REDES NEURAIS CONVOLUCIONAIS, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.

2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.

3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.

4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.

5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.

6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.

7. Por ser verdade, firmo a presente declaração.

Bernardo Cardoso Cordeiro

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

AGRADECIMENTO

Dedico este trabalho ao povo brasileiro que contribuiu de forma significativa a minha formação e estadia nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e a confiança em mim depositados.

RESUMO

Este trabalho consiste na aplicação de técnicas de representação de palavras (*word embeddings*) atuais, considerando arquiteturas de redes neurais convolucionais, voltadas para a classificação de documentos textuais.

Camadas de convolução são tradicionalmente utilizadas no processamento de imagens, mas recentemente, elas têm ganhado peso também no processamento de textos. Em um dos primeiros artigos abordando este uso desta técnica, foi considerada a utilização de um dos métodos mais conhecidos de *word embeddings*, o word2vec. Para este projeto, visa-se replicar este trabalho utilizando um dos métodos mais recentes de *word embeddings*, o BERT, de modo a comparar os efeitos de cada um no desempenho do modelo.

Além disso, alguns conjuntos de documentos em português também serão utilizados. A maioria das pesquisas realizadas neste campo utilizam corpus de documentos em língua inglesa, o que acarreta em uma escassez deste tipo de trabalho no idioma lusófono. Desta forma, este projeto se coloca, também, como uma contribuição para o avanço destas soluções no campo da língua portuguesa.

Por último, visa-se realizar não só uma análise inferencial do desempenho, como também uma análise computacional. O que isto quer dizer é que, além de utilizar métricas ligadas ao acerto e erro estatístico das técnicas, também serão levados em consideração aspectos computacionais, especificamente o tempo e a memória necessários nas soluções analisadas.
Palavras-Chave: classificação de texto, redes neurais convolucionais, BERT, word embeddings, word2vec.

ABSTRACT

This work consists in using current word embedding techniques on top of convolutional neural network architectures, applied to the classification of textual documents.

Convolutional layers are traditionally used in the context of image processing. Recently, however, they have gained popularity in the field of text processing. In one of the first papers covering the use of this architecture in this domain, one of the most well-known methods of word embeddings was employed, namely word2vec. For this project, I aim to replicate this paper utilizing a more recent method of building word embedding, BERT, in order to compare the effects of each method on the model's performance.

Moreover, some datasets in Portuguese will also be used. Most research done in this field make use of corpora in the English language, which means the amount of work available in the Portuguese language is quite scarce. As such, this project also has the goal of contributing to the advancement of the domain in the Portuguese language.

Last but not least, on top of performing an inferential analysis, a computational analysis will also be made. This means that, in addition to using metrics tied to the statistical error of these techniques, the computational aspects will also be taken into consideration, specifically the time and memory necessary to run the models.

Key-words: text classification, convolutional neural networks, BERT, word embeddings, word2vec.

SIGLAS

UFRJ - Universidade Federal do Rio de Janeiro

BERT - *Bidirectional Encoder Representations from Transformers*

CNN - *Convolutional Neural Network* - Rede Neural Convolucional

NLP - *Natural Language Processing* - Processamento de Linguagem Natural

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

O tema principal deste trabalho é o da classificação de documentos textuais, dentro da área de Processamento de Linguagem Natural, analisado sob um aspecto inferencial e computacional. Este problema se refere à construção de um modelo capaz de atribuir diferentes classes a documentos de texto. Para isso, o modelo requer um conjunto de treinamento, composto por documentos que já foram marcados com as classes desejadas.

1.2 Delimitação

A demanda por este tipo de tecnologia é vasta, com aplicações nos campos mais variados. Tanto [1] quanto [2] e [3] citam algumas aplicações comuns. Uma das mais antigas e conhecidas é a filtragem de *spam*, onde novos *emails* são classificados como spam ou não. Outra aplicação é a classificação de artigos de jornal, de forma a selecionar melhores recomendações para grupos de usuários relevantes. Na área de suporte ao consumidor, pode-se classificar as reclamações de acordo com temas determinados, de modo a agrupá-las e analisá-las mais rapidamente. No campo do comércio eletrônico (entre outros), uma aplicação comum é a análise de sentimentos, em que *feedbacks* de usuários podem ser classificados como positivos ou negativos, assim podem ser identificados produtos que estão sendo bem ou mal recebidos. Outra aplicação é a atribuição de autor [4], cujo objetivo é identificar o autor de

um determinado texto (que pode ser usado tanto para textos históricos, quanto para análise forense). Atualmente, também se tem explorado o potencial desta área na classificação de documentos legais [5]. Em muitos destes campos, o aumento exponencial dos dados disponíveis impossibilitou a realização manual destas tarefas, o que torna necessário o uso de algoritmos para automatizar o processo. A escolha de algoritmos, no entanto, é longe de ser trivial. Além dos aspectos estatísticos, ligados à taxa de acertos de cada modelo, também devem ser levados em conta aspectos computacionais, sobretudo o tempo e a memória necessários, que podem ser impeditivos em determinados cenários.

1.3 Justificativa

Como demonstrado anteriormente, o tema possui diversas aplicações. Para cada uma destas aplicações, as razões para realizá-las pode variar, mas os interesses costumam ser os mesmos. Em algumas, classificar textos pode levar a uma maior personalização da experiência do usuário (por exemplo, no caso de recomendação de determinados artigos de jornal). Em outros, essa classificação é necessária para se entender os tipos de problemas recorrentes que uma empresa está enfrentando (como na aplicação voltada ao suporte ao consumidor ou a *feedbacks* de produtos). Em todos os casos, no entanto, o principal interesse é automatizar e agilizar um processo muitas vezes custoso e demorado quando realizado manualmente.

1.4 Objetivos

O objetivo principal deste trabalho é comparar o desempenho do BERT, com técnicas mais tradicionais, aqui sob a forma da solução *word2vec*. De acordo com os criadores do método BERT, ele é um método mais flexível que aumenta a performance dos modelos nos quais ele é inserido, em diversos tipos de tarefas. Verificar até que ponto ele é acoplável a modelos já existentes, e o aumento do desempenho trazido em relação aos maiores requisitos computacionais (para diferentes conjuntos de dados), é altamente desejável.

Assim, para contrapor este aumento do desempenho ao tempo e memória necessários, também serão medidos os tempos de treinamento e predição dos mo-

delos e suas configurações. Em [6], o autor realiza uma discussão sobre o que ele chama de *inferential thinking*, interessado no desempenho estatística dos modelos, contrapondo-o com o *computational thinking*, relacionado aos requisitos computacionais como tempo e memória. O segundo objetivo deste projeto é, portanto, de explorar a necessidade de considerar tanto o desempenho de um modelo quanto a sua complexidade computacional ao realizar trabalhos no campo da ciência de dados.

1.5 Metodologia

Para atender aos objetivos supracitados, será feita a replicação do artigo *Convolutional Neural Networks for Sentence Classification* [7], em seguida modificando-o para utilizar o BERT.

Primeiramente, o modelo original será replicado. Aqui, espera-se encontrar resultados iguais ou similares aos encontrados no artigo, o que servirá também para revisar o trabalho original, e identificar pontos que não tenham ficado claros. Outros conjuntos de dados também serão utilizados para obter resultados complementares aos obtidos pelo autor.

Em seguida, com esses resultados em mãos, o modelo será modificado para utilizar o método BERT para gerar os novos *word embeddings*. Isso permitirá avaliar não apenas a diferença de desempenho obtida pelos dois modelos de um ponto de vista estatístico, mas também medir a diferença de complexidade entre os dois, principalmente através do tempo necessário para as etapas de treinamento e também de predição de cada um dos modelos.

1.6 Descrição

No Capítulo 2 será feito um resumo não-extensivo de alguns dos diferentes métodos que existem para realizar a tarefa de classificação de documentos.

O Capítulo 3 começará explicando os conceitos de *word embeddings* e camadas de convolução, importantes para a compreensão do resto do trabalho. Em seguida, será apresentado em mais profundidade o modelo utilizado, tanto em sua versão original, quanto modificada, com detalhes mais técnicos sobre o funcionamento da arquitetura.

Após isso, os conjuntos de dados utilizados, assim como o preprocessamento realizado, o processo de avaliação escolhido, e a configuração experimental usada serão mostrados no Capítulo 4.

Depois disso, o Capítulo 5 fará um breve resumo das métricas que costumam ser utilizadas no campo, assim como as métricas escolhidas e os resultados obtidos. Nele serão explicitados as diferenças de performance, tempo de treinamento, tempo de predição e memória utilizada entre o modelo original e modificado.

Por último, no Capítulo 6 serão apresentadas as conclusões obtidas, pontos de discussão importantes, assim como sugestões para aprofundar o tema e melhorar os modelos futuramente.

Nem todos os detalhes serão explicados minuciosamente, já que isso desviaria do foco principal do trabalho. No entanto, para os pontos técnicos que, porventura, necessitem de um maior aprofundamento, será fornecida literatura para que o leitor possa encontrá-los em outras fontes.

Capítulo 2

Revisão da Literatura

A classificação de um texto de acordo com um conjunto de classes não é algo novo. Inclusive, é uma tarefa que pode aparecer em várias situações, tais como a identificação de idioma, a classificação em tópicos, em análise de sentimento e na detecção de *spam*, entre outros [3]. Ao longo dos anos, diversos métodos foram pensados para atacar este problema. Estes métodos podem ser agrupados em duas categorias principais: métodos baseados em regras (do inglês *rule-based*, às vezes também chamado de *knowledge-based*), e métodos estatísticos (ou de Aprendizado de Máquina). Nesta seção, iremos fazer um breve resumo destas diferentes abordagens..

2.1 Abordagens Baseadas em Regras

É possível classificar um texto em diversas categorias seguindo regras definidas manualmente. Por exemplo, se estamos classificando artigos de jornal em diferentes tópicos, poderiamos definir a regra de que, se as palavras *economia*, *bolsa* ou *PIB* aparecem no texto, então o artigo pertence ao tópico de economia. Caso as palavras *política*, *presidente* ou *senado* aparecem, então o artigo é sobre política, e assim por diante.

Métodos deste tipo são referidos como baseados em regras. Um exemplo de método desta categoria pode ser visto em [8].

Uma discussão sobre as vantagens e limitações desta abordagem é feita em [9]. Uma das vantagens desta abordagem é que é muito fácil para um ser humano entender a lógica que está sendo utilizada para classificar novos textos. Em outras

palavras, são métodos altamente interpretáveis. Além disso, é muito fácil expandi-los, adicionando novas regras que reflitam novos conhecimentos obtidos. De fato, a possibilidade de integrar conhecimentos de domínio diretamente no sistema de classificação por meio destas regras é uma das maiores vantagens dos métodos desta categoria. No entanto, esta melhora é muito difícil de manter (já que, por vezes, as novas regras podem afetar as regras antigas), além de consumir muito tempo no caso de sistemas muito complexos, que necessitam de um tempo considerável de análise e teste de cada regra, tanto individualmente quanto em conjunto. Outro ponto é que as regras costumam ser muito sensíveis a pequenas nuances no texto, que, embora não mudem o significado final do que está escrito, deixam, muitas vezes, de serem capturadas pelas regras determinadas.

2.2 Abordagens de Aprendizado de Máquina

Uma outra categoria de abordagem são as baseadas em Aprendizado de Máquina. Neste caso, o texto a ser tratado deve, primeiramente, ser transformado em atributos os quais alimentarão um modelo de aprendizado tradicional.

Uma das maneiras mais populares de realizar a transformação do texto em atributos é o chamado *Bag-of-Words* (BoW), transformação citada, por exemplo, em [10]. Nesta técnica, o texto a ser analisado é transformado em um vetor de tamanho n. Cada dimensão deste vetor está associada a uma determinada palavra, e o valor de uma dada dimensão é o número de vezes que aquela palavra associada apareceu no texto em questão. Se há m textos a serem analisados, eles podem ser representados por uma matriz m x n ao utilizar esta técnica.

Uma variação desta transformação utiliza a chamada TF-IDF, que além de contar a frequência de cada termo (Term Frequency - TF) nos textos, também considera o inverso da frequência do termo nos documentos (Inverse Document Frequency - IDF). Isto faz com que um termo que apareça muito em um texto e pouco em outros tenha uma importância grande naquele texto. Enquanto isso, termos que aparecem em todos os textos muitas vezes não têm um peso tão elevado.

Ao obter estes atributos, é possível aplicar diversos algoritmos de Aprendizado de Máquina para então classificar os textos, como por exemplo o Naive Bayes,

Regressão Logística, SVMs, ou, no nosso caso, Redes Neurais.

Cada modelo não será detalhado aqui, pois foge do escopo deste projeto, mas uma vasta literatura pode ser encontrada acerca deles, assim como de inúmeros outros, como por exemplo em [10], [11] ou [12].

Mais recentemente, com o surgimento do *word2vec* [13] e dos *word embeddings*, tem-se conseguido um aumento de desempenho significativo ao utilizar diferentes arquiteturas de redes neurais tais como as redes neurais convolucionais (que utilizaremos neste projeto) [7], mas também as arquiteturas de redes neurais recorrentes [14], e mesmo recorrentes e convolucionais [15]. Além disso, outras técnicas, como os mecanismos de atenção [16], têm aparecido, e estão ajudando a avançar rapidamente o campo de NLP nas mais diversas tarefas, como a de tradução automática de textos, sistemas de perguntas e respostas, entre outros.

Um outro tipo de abordagem, chamada híbrida, também pode ser utilizada. Ela é basicamente uma mistura das duas abordagens anteriores: usa-se um modelo de Aprendizado de Máquina junto com regras estabelecidas (seja para a obtenção de atributos, ou para refinar a saída dos modelos) de forma a melhorar o desempenho final.

Capítulo 3

Metodologia

Nesta seção, será detalhada a metodologia utilizada. Primeiramente, os conceitos de *word embeddings* e de redes neurais convolucionais serão explicados brevemente, já que são conceitos essenciais para o entendimento das etapas seguintes.

Em seguida, o modelo do artigo de Kim, Y. [7] será detalhado, por se tratar do modelo sobre o qual este trabalho se baseia. Também serão detalhadas as modificações feitas à arquitetura original, utilizando o BERT, e como ela se acopla no modelo.

3.1 *Word Embeddings*

Um *word embedding* pode ser entendido como uma representação de uma palavra dentro de um espaço vetorial. Essa representação deve ser feita de tal forma que, idealmente, palavras com sentidos similares estejam próximas umas das outras neste espaço. Por exemplo, a palavra *homem* deve ficar mais próxima da palavra *mulher* do que da palavra *avião*.

Uma das técnicas mais conhecidas e utilizadas para construir *word embeddings* é o word2vec [13]. Nesta técnica, as representações são construídas a partir de um corpus de muitos documentos, que leva em consideração a palavra dentro de seu contexto (ou seja, a co-ocorrência de diferentes palavras). Isso quer dizer que, se *pai* e *filho* costumam aparecer em contexto similares, eles terão representações similares no espaço vetorial.

Um detalhe importante a ser mencionado quando falamos do word2vec, é

que cada palavra possui uma única representação no espaço. Ou seja, ao treinar o *embedding* de uma palavra, o seu contexto é levado em consideração. Mas ao final deste processo (na fase de predição, podemos dizer), cada palavra possuirá um único *embedding* associado. Isso quer dizer que a palavra *banco* possuirá a mesma representação na frase "fui ao banco hoje"e na frase "sentei-me no banco do parque". Para mais detalhes sobre o algoritmo do word2vec, favor referir-se ao artigo original [13].

Uma das maiores vantagens de utilizar *word embeddings* é que eles oferecem uma maneira prática de realizar transformações e outras operações matemáticas em cima de um conjunto de palavras, comparado com maneiras mais tradicionais de tratar documentos, como a representação por BoW (*Bag of Words*).

Outros métodos de *word embeddings* conhecidos na literatura atualmente são o GloVe [17], ELMo [18], ULMFit [19], e mais atualmente, o BERT [20], que será usado neste projeto.

3.2 Redes Neurais Convolucionais

Outro conceito que será importante no decorrer deste projeto, são as Redes Neurais Convolucionais. Primeiro, é necessário explicar o que é uma camada (ou filtro) de convolução. Uma camada de convolução pode ser entendida como uma função que é aplicada sobre uma janela de valores.

Se considerarmos, por exemplo, uma imagem, onde cada pixel pode ser entendido como um valor entre 0 e 255, a imagem nada mais seria então do que uma matriz de valores. Camadas convolucionais podem ser aplicadas sobre esta matriz para realizar uma miríade de operações, como por exemplo, detectar bordas, desfocar uma imagem ou deixá-la mais nítida, entre outros. A Figura 3.1 mostra um exemplo aplicado à detecção de bordas.

Redes neurais que se utilizam de camadas de convolução são largamente utilizadas em aplicações em imagem, como por exemplo em [21]. Digamos que estamos interessados em detectar cães e gatos num conjunto de imagens. Seria interessante definir filtros de convolução capazes de detectar características relativas a estes animais, tais como os olhos, focinho, pelo, entre outros, que nos ajudem no objetivo

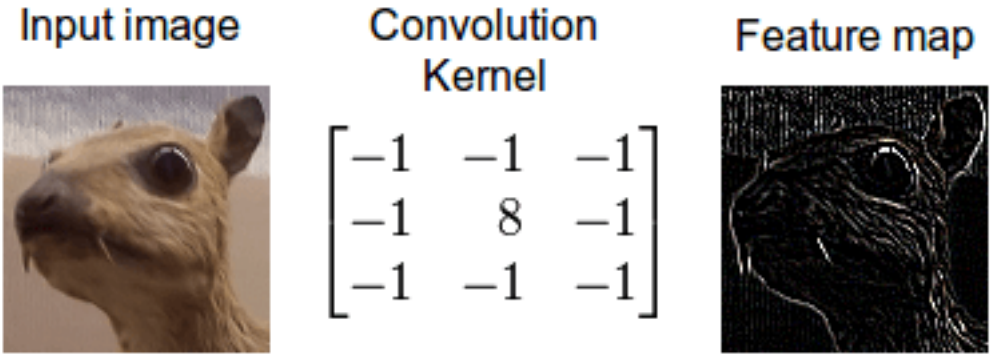

Figura 3.1: Exemplo de um filtro de convolução de tamanho 3x3 usado para detectar as bordas de uma imagem. Fonte: https://timdettmers.com/2015/03/26/convolution-deep-learning/, link acessado em 05/05/2019.

final que é a detecção do animal. No entanto, definir estes filtros manualmente seria uma tarefa quase impossível. Ao treinar muitos filtros de convolução em uma rede neural, no entanto, isso se torna possível, já que deixamos o modelo descobrir os melhores valores a serem usados nas diversas camadas para classificar a imagem em "cão"ou "gato".

3.3 Modelo Original

Com os conceitos de *word embeddings* e filtros de convolução em mãos, podemos agora partir para a explicação do modelo utilizado, já que o artigo de Kim [7] utiliza os *word embeddings* em conjunto com camadas de convolução.

Ao obter *embeddings* para cada palavra no nosso conjunto de documentos, é possível construir uma matriz de tamanho m x N para cada documento, onde m é o número máximo de palavras no documento, e N é o número de dimensões dos *word embeddings*. A ordem em que estes *word embeddings* aparecem é ditada pela ordem das palavras no documento.

Por exemplo, consideremos a seguinte frase: "Eu fui ao banco hoje de manhã". Esta frase possui 7 palavras. Se considerarmos que cada palavra pode ser representada por um *embedding* de dimensão 5, então é possível construir uma matriz de

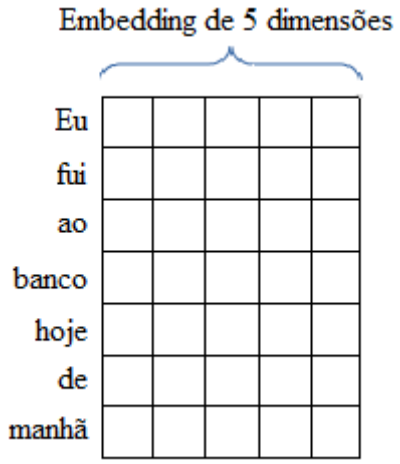

Figura 3.2: Exemplo de uma frase representada por um *embedding* de cinco dimensões. Fonte: Imagem criada para este projeto.

tamanho 5x7 correspondente a esta frase, como na Figura 3.2.

Esta representação do documento através de uma matriz é muito parecida com a representação de uma imagem através de uma matriz dada pelo valor de seus pixels. Em outras palavras, a sequência de palavras, representadas por seus respectivos *word embeddings*, são como uma "imagem"em 2D do texto. Desta forma, a utilização de redes neurais convolucionais sobre estas representações se torna possível.

No artigo considerado, o autor propõe o uso de uma janela que contemple o *word embedding* em sua plenitude, apenas deslizando em cima da sequência de palavras. Assim, para cada k palavras, obtemos um valor dado pelo filtro de convolução (que será treinado para obter a classificação correta daquele documento). Este valor também passa por uma função de ativação não-linear chamada ReLU (do inglês, *Rectified Linear Unit*), que irá retornar o $max(0, z)$, onde z é o valor obtido pela convolução, eliminando assim valores negativos.

Em seguida, estes valores serão agregados através de uma função de *max pooling*. Esta função retorna o maior valor presente em um conjunto de valores, visando identificar as características dominantes, e desconsiderando a localização destas características (invariantes ao deslocamento). Esta operação é repetida com diferentes filtros de convolução aplicados a esta sequência de *word embeddings*.

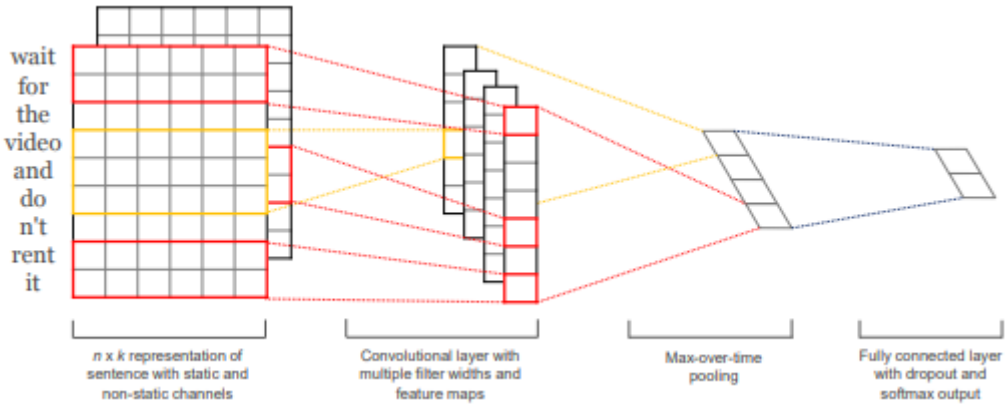

Figura 3.3: Arquitetura do modelo CNN. Fonte: Convolutional Neural Networks for Sentence Classification [7].

Ao final desta camada, um vetor de x valores é extraído, onde x é o número de filtros escolhido. Estes x valores passarão, então, por uma camada densa, que ao final retornará um vetor y de tamanho n (onde n é o número de classes do problema), seguida de uma função de ativação *softmax*. Esta função retorna a probabilidade do documento pertencer a cada uma das classes. A fórmula correspondente pode ser vista na Equação 3.1, onde i indica a classe para a qual a probabilidade está sendo calculada. É este valor que será usado para obter a classe final do documento. O modelo pode ser visto na Figura 3.3.

$$softmax(y_i) = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)} \tag{3.1}$$

Este modelo, assim como todos os modelos que se utilizam de redes neurais, é treinado a partir de um algoritmo chamado de *backpropagation*. Basicamente, este algoritmo consiste em calcular o gradiente de modo a minimizar a perda causada por uma má classificação dos textos. Este gradiente é propagado, começando das camadas finais, voltando para as camadas iniciais, ou seja, propagado para trás, de onde vem o nome *backpropagation*. A explicação completa do *backpropagation* junto com as equações correspondentes foge ao escopo deste trabalho, e não será abordada, mas pode ser encontrada em [22], assim como uma variedade de outras explicações de conceitos relacionados a redes neurais.

Ao final deste treinamento, espera-se obter valores para os parâmetros da

rede neural que maximizem a performance da mesma. Isto inclui os parâmetros da camada densa, assim como os valores usados nos filtros de convolução, e nos *word embeddings* de cada palavra.

No artigo de Kim [7], ele define quatro configurações para o modelo: multi-canal, estática, não-estática, e aleatória (no artigo: *multichannel, static, nonstatic* e *rand*). Estas quatro configurações dizem respeito a se os *word embeddings* devem ou não ser treinados pelo modelo ou serem mantidos fixos:

- **Rand:** O word2vec não é utilizado, sendo os valores dos *embeddings* iniciados de forma aleatória e modificados durante a fase de treinamento através do *backpropagation*.
- **Static:** O word2vec é utilizado, mas seus valores não são modificados durante a fase de treinamento, logo mantidos fixos.
- **Non-Static:** O word2vec é utilizado, e seus valores são modificados durante a fase de treinamento através do *backpropagation*, sendo otimizados para cada conjunto de treinamento.
- **Multichannel:** O word2vec é utilizado, e dois canais diferentes são passados para os filtros de convolução - um canal com os *embeddings* fixos, e o outro com *embeddings* treináveis. Esta configuração específica é a representada na Figura 3.3.

Iremos, assim como no artigo original, implementar as quatro configurações para o modelo com o word2vec.

3.4 Modelo modificado

O BERT, ao contrário do word2vec, não gera *word embeddings* únicos para cada palavra ao final de seu treinamento. O que ele faz é fornecer um modelo que, dada uma frase inteira, irá gerar um *word embedding* para cada palavra dentro do contexto da frase. Isso quer dizer que, no exemplo das frases "fui ao banco hoje"e "sentei-me no banco do parque"dado anteriormente, a palavra *banco* possuirá representações distintas em cada uma delas.

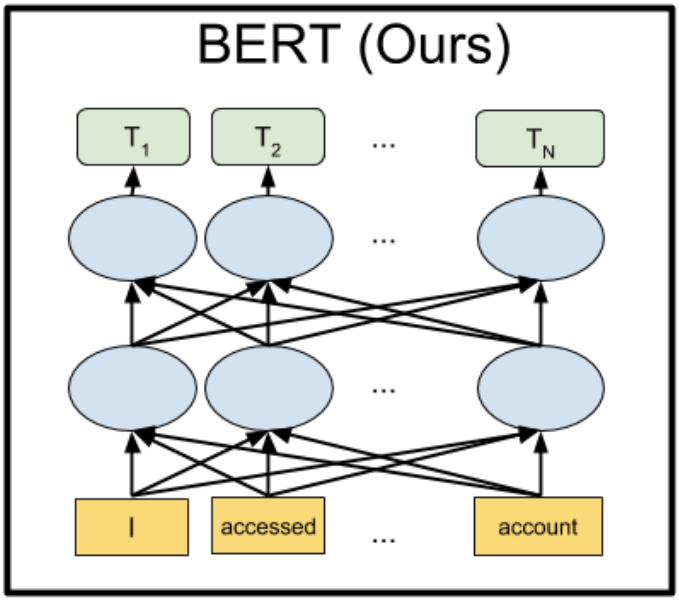

Figura 3.4: Representação da arquitetura do BERT. Fonte: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html, link acessado em 05/05/2019.

A arquitetura utilizada para o BERT é construída de tal forma que, ao treinar e ao predizer os *embeddings*, tanto as palavras anteriores quanto seguintes são levadas em conta, e mecanismos de atenção [16] são utilizados para guardar a informação das palavras mais e menos relevantes na frase. Uma representação da maneira como o BERT funciona está ilustrada na Figura 3.4. O funcionamento completo do BERT está fora do escopo deste trabalho, mas pode ser visto no seu artigo original [20].

Para o modelo modificado, um acoplamento do modelo anterior e do modelo do BERT (após a camada que fornece os *embeddings* finais) é realizado. Isso quer dizer que, ao invés de definir uma camada de *embeddings* como ponto de partida do modelo, onde cada palavra possui uma única representação (como estávamos fazendo com o *word2vec*), o ponto de partida agora será o modelo do BERT, e os *embeddings* gerados por este serão utilizados nas subsequentes camadas de convolução, função de *max pooling* e camada densa mencionadas na Seção 3.3.

Da mesma forma como foi feito com o *word2vec*, diversas configurações serão testadas, especificamente as configurações *multichannel*, *static* e *non-static*. O equivalente da versão *rand* para o BERT não será utilizada por uma questão de difi-

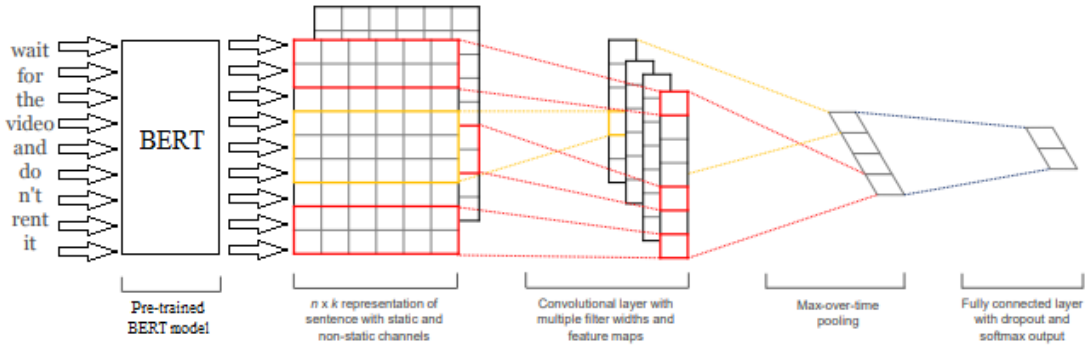

Figura 3.5: Arquitetura do modelo CNN utilizando os *embeddings* gerados pelo BERT. Fonte: Criado manualmente, utilizando a imagem da arquitetura de [7] como base.

culdade técnica: a Google fornece uma interface para facilitar o uso do BERT, mas esta interface não parece ter nenhum meio de utilizar o modelo sem *word embeddings* pré-treinados de forma simples.

Capítulo 4

Datasets e Configuração Experimental

Nesta seção, serão apresentados os conjuntos de dados que foram utilizados, o preprocessamento e a tokenização realizados nos dados, o processo de avaliação escolhido e a configuração de *hardware* utilizada que, embora não afete o desempenho do modelo, tem uma grande influência no tempo de execução destes.

4.1 Conjuntos de Dados

Dois conjuntos de dados que foram utilizados são os mesmos do artigo [7], especificamente: MR e TREC. Isso permite não apenas uma comparação entre a performance dos dois modelos, mas também uma comparação com os resultados obtidos no artigo original. Além disso, um conjunto de dados em português também foi utilizado, considerando a análise de sentimentos no Twitter [23].

- **MR:** *Reviews* de filmes, classificados como bons ou ruins. As classes estão balanceadas, com 5332 casos positivos e 5332 casos negativos [24].
- **TREC:** Conjunto de dados de perguntas classificadas em 6 tópicos diferentes. Possui 5452 perguntas no conjunto de treino, e um conjunto de teste de 500 perguntas. Suas classes não estão balanceadas [25].
- **PortTwitter:** Conjunto de dados de análise de sentimentos no Twitter, classificados como positivos, negativos ou neutros. Foi utilizado o conjunto des-

balanceado, em que 1299 são negativos, 553 casos são neutros e 168 casos são positivos [23].

4.2 Tokenização

Para o modelo baseado no artigo original [7], o preprocessamento e a tokenização realizados em cima dos textos é mínimo. Separam-se as palavras por espaço, e certas pontuações, como vírgulas, pontos de exclamação e interrogação são considerados tokens por si só. No caso de textos em inglês, junções de palavras (como "don't", "he'll", etc.) também são separados em dois tokens (por exemplo, "don't"se torna "do"e "n't").

Já no caso do BERT, o processo de tokenização utilizado foi o mesmo do código fornecido pela Google para a utilização da técnica. Neste caso, a tokenização é mais complexa. Aqui, são consideradas não só as palavras separadas por espaço e certos itens de pontuação, mas também se separam certas palavras em vários tokens (por exemplo, a palavra "running"se torna dois tokens: "runn"e "ing"). Isto faz com que as nuances de certas derivações de palavras sejam capturadas em seus determinados tokens. Esse tokenizador utilizado pelo BERT está disponível no Github do próprio BERT [26].

4.3 Teste e Validação

Alguns conjuntos de dados estão disponíveis com um conjunto de teste previamente estabelecido. Nestes casos, a avaliação do modelo é feita em cima deste conjunto de teste. Nos casos em que só dispomos de um conjunto de treino, a validação é ligeiramente diferente entre o *word2vec* e o BERT. Nos modelos que utilizam o *word2vec*, é realizado um KFold onde K = 10 (ou seja, 10 *splits* onde 10% dos dados é utilizado como teste), e a performance final é a média do desempenho em cada um dos *splits*. Já no caso onde o BERT é utilizado, por restrições de tempo, seria extremamente demorado rodar um KFold com 10 *splits* (mesmo com a configuração de *hardware* explicitada abaixo). Por isso, nos casos onde o BERT é utilizado, 10% dos dados são separados aleatoriamente para compôr o teste. Cada modelo é treinado por 10 épocas (ou seja, 10 iterações do conjunto de dados). Um

número maior de épocas pode influenciar nas performances finais obtidas, sendo algo a ser considerado em trabalhos futuros.

4.4 Código e Hardware

Os modelos apresentados neste trabalho foram implementados utilizando a linguagem Python, mais especificamente a biblioteca Tensorflow, para a construção dos modelos de Redes Neurais.

Para rodar o treinamento e avaliação dos modelos, foi feito uso da Google Cloud Platform (GCP), utilizando uma máquina com uma placa de vídeo NVIDIA Tesla K80, com 24GB de RAM. A utilização de uma máquina mais potente foi algo necessário devido, principalmente, à complexidade do BERT. É possível rodar o modelo com *word2vec* em uma máquina local, mesmo com uma CPU (embora seja altamente recomendado utilizar uma GPU para acelerar o processo). Mas ao tentar rodar o modelo com BERT localmente, simplesmente não foi possível, principalmente por questões de tempo de execução.

Para os novos usuários, a GCP disponibiliza uma quantia de $300 que possibilitou a realização deste trabalho sem nenhuma preocupação financeira, embora também representasse um fator limitador em relação a quanto poderia ser feito sem ultrapassar este valor.

O código do projeto pode ser encontrado no github[1].

[1] https://github.com/bernardoccordeiro/BERT-CNN

Capítulo 5

Resultados

Nesta seção, serão apresentados os resultados obtidos para os diferentes conjuntos de dados, nas diferentes configurações utilizadas para cada um dos modelos.

5.1 Métricas

Para avaliar os modelos de um ponto de vista estatístico, é necessário definir qual métrica será utilizada. Existem diversas opções de métricas para a tarefa de classificação em geral. Uma das mais importantes etapas ao avaliar um modelo é justamente a definição de uma métrica adequada. Para entender as métricas mais comuns deste domínio, precisamos começar definindo alguns conceitos centrais.

Para ajudar na explicação destes conceitos, vamos imaginar um problema hipotético. Suponha que queiramos classificar um conjunto de 100 *emails* em *spam* e não-*spam*. Para os *emails* de *spam*, daremos a classe 1, ou classe positiva. Para os *emails* que não são *spam*, daremos a classe 0, ou classe negativa. Quando fazemos uma predição sobre a classe de determinado *email*, existem quatro possibilidades em relação à corretude desta, que são:

- **Verdadeiro Positivo:** A classe correta era a classe 1 e a predição feita indicou a classe 1, ou seja, o *email* era *spam* e o classificamos corretamente como *spam*.
- **Verdadeiro Negativo:** A classe correta era a classe 0 e a predição feita indicou a classe 0, logo, o *email* não era *spam* e o classificamos corretamente como não-*spam*.

- **Falso Positivo:** A classe correta era a classe 0 e a predição feita indicou a classe 1, um *email* que não era *spam* que foi classificado como *spam*. Erros deste tipo também são chamados na literatura de erros do tipo I.

- **Falso Negativo:** A classe correta era a classe 1 e a predição feita indicou a classe 0. Um *email* de *spam* que foi classificado como não-*spam*. Erros deste tipo também são chamados na literatura de erros de tipo II.

Assim, o objetivo de qualquer algoritmo de classificação será maximizar o número de verdadeiros positivos e verdadeiros negativos, ao mesmo tempo em que minimiza o número de falsos positivos e falsos negativos. Uma maneira simples de visualizar conjuntamente estas informações é através da chamada **matriz de confusão**.

	Predição 0	Predição 1
Classe 0	40	10
Classe 1	20	30

Tabela 5.1: Exemplo de uma matriz de confusão para o problema hipotético da classificação de *emails*.

Na Tabela 5.1 temos o exemplo de uma matriz de confusão. Utilizando o exemplo dos emails nesta matriz, podemos ver que 100 emails foram classificados ao todo, onde 40 foram verdadeiros negativos, 30 foram verdadeiros positivos, 10 foram falsos positivos e 20 foram falsos negativos. Em um problema de classificação de múltiplas classes, essa matriz pode ser aumentada para examinar quais classes estão sendo confundidas umas com as outras, permitindo uma melhor percepção dos casos nos quais o algoritmo está cometendo erros.

Ainda utilizando a Tabela 5.1 como exemplo, podemos definir os conceitos de **acurácia**, **precisão** e **sensibilidade** (na literatura muitas vezes chamada de **recall**).

A **acurácia** nada mais é do que a porcentagem de predições corretas. Matematicamente falando, isso pode ser expresso como:

$$Acc = \frac{VP + VN}{VP + VN + FP + FN} \tag{5.1}$$

Onde VP representa o número de verdadeiros positivos; VN, o número de verdadeiros negativos; FP, o número de falsos positivos, e FN, o número de falsos negativos.

A Tabela 5.1 indica que o algoritmo obteve uma acurácia de 70% no problema de classificação de *emails*, pois, dos 100 *emails*, 70 foram corretamente classificados.

Por ser uma métrica de fácil entendimento, ela é largamente utilizada na literatura. Porém, existem outras métricas importantes, tais como a **precisão** e o ***recall***. Para explicar o que essas métricas querem dizer, vamos voltar ao exemplo da classificação de *emails*. A precisão irá responder a pergunta: "dos *emails* classificados como *spam*, quais de fato o eram"? Por sua vez, o *recall* se preocupa em dizer: "dos *emails* que são *spam*, quantos foram identificados"?

Matematicamente, isto é expresso através das seguintes fórmulas:

$$Precisão = \frac{VP}{VP + FP} \tag{5.2}$$

$$Recall = \frac{VP}{VP + FN} \tag{5.3}$$

Com os valores da Tabela 5.1, obtemos uma precisão de 75% (dos 40 *emails* classificados como *spam*, 30 de fato o eram), enquanto que o *recall* foi de 60% (dos 50 *emails* que eram *spam*, 30 foram identificados como *spam*). Idealmente, gostaríamos que tanto precisão quanto *recall* sejam iguais a 100% (o que também levaria a uma acurácia de 100%), mas, na prática, muitas vezes um aumento da precisão acarreta em uma redução do *recall* e vice-versa.

Uma última métrica que vale a pena mencionar é o **F1 score**. Este nada mais é do que a média harmônica entre a precisão e o *recall*. Em muitos conjuntos de dados, é possível obter uma precisão alta e um *recall* baixo (ou vice-versa), e para avaliar se um algoritmo está com um desempenho bom ou não, é necessário olhar ambas as métricas. Utilizando o F1, podemos agregar a informação destas duas métricas em uma única, através de:

$$F1 = 2 \cdot \frac{precisão \cdot recall}{precisão + recall} \tag{5.4}$$

Claro que cada caso é um caso, e para muitos problemas, a acurácia pode ser mais do que suficiente para avaliar de forma razoável a desempenho de um

algoritmo. Já em outros casos, o *recall* ou a precisão podem ter precedência sobre outras métricas.

Para o problema tratado neste trabalho, como em muitos casos onde se lida com problemas multi-classe, iremos utilizar, especificamente, a média ponderada do F1 de cada classe:

$$F1_{modelo} = \frac{\sum_{c \in C} sup_c \cdot F1_c}{\sum_{c \in C} sup_c} \tag{5.5}$$

Onde sup_c representa o número de amostras do conjunto de dados pertencentes à classe c. Desta maneira, podemos olhar para o F1 de maneira global, levando em consideração também possíveis desbalanceamentos na distribuição das classes.

Além do ponto de vista estatístico, também precisamos levar em consideração o ponto de vista computacional. Para isto, iremos considerar os dois fatores mais importantes quando falamos de desempenho na área da computação: memória e tempo. Para analisar a memória, será considerada tanto a memória total utilizada pelos parâmetros dos modelos, assim como a memória apenas dos parâmetros treináveis. Já quando analisarmos o tempo, iremos olhar tanto para o tempo de treinamento (em média, por época), quanto para o tempo de predição (em média, por amostra).

É importante levar em conta todos esses fatores pois, em um caso de uso real, todos eles podem ser impeditivos de uma maneira ou de outra. Por exemplo, em determinadas aplicações, podemos precisar de um desempenho estatístico mais alta, mesmo que para isso seja necessário sacrificar tempo e memória. Já em outros casos, a quantidade de recursos (tanto em poder de processamento quanto de memória) pode ser limitada, sendo preferível a escolha de modelos mais simples.

5.2 Resultados

Na Tabela 5.2, podemos perceber que no caso do BERT, o uso de memória aumenta muito em relação ao *word2vec*. Existem diversas razões para tal, como por exemplo o maior número de dimensões utilizados nos *embeddings* do BERT, assim como a maior quantidade de camadas, além da camada de *embeddings*, presente nesse modelo. No entanto, a maior razão para o uso de memória neste caso é o

	MR	TREC	PortTwitter
W2V Multichannel	46.5MB/23MB	20MB/10MB	24MB/12MB
W2V Static	23MB/70KB	10MB/71KB	12MB/70KB
W2V Non-Static	23MB/23MB	10MB/10MB	12MB/12MB
Rand	23MB/23MB	10MB/10MB	12MB/12MB
BERT Multichannel	1.362GB/681MB	1.362GB/681MB	1.362GB/681MB
BERT Static	681MB/180KB	681MB/180KB	681MB/180KB
BERT Non-Static	681MB/681MB	681MB/681MB	681MB/681MB

Tabela 5.2: Memória dos modelos para os diferentes conjuntos de dados e algoritmos. O primeiro valor de memória corresponde ao número total de parâmetros. Por sua vez, o segundo valor corresponde à memória dos parâmetros treináveis.

fato de que, no BERT, é difícil controlar o vocabulário que será usado pela camada de *embedding*, já que a interface disponibilizada pela Google não permite este nível de customização ao construir o modelo. Em outras palavras, no *word2vec* podemos facilmente construir o modelo utilizando apenas as palavras presentes no conjunto de dados, enquanto que no BERT, todas as palavras, mesmo aquelas de outras línguas e não-presentes no conjunto de dados sendo tratado, estão presentes no modelo (embora não-utilizadas).

Um ponto a ser lembrado ao analisar esses valores de memória é que eles correspondem apenas aos parâmetros necessários para guardar o modelo em si. As etapas de treinamento em geral utilizam memória de 3 fontes diferentes:

- Os valores de saída das camadas intermediárias necessários para realizar o *backpropagation*. Algumas implementações podem armazenar apenas o valor da saída que está sendo analisado em um dado momento, reduzindo drasticamente a memória utilizada por essa fonte.

- Os parâmetros (que são os valores da Tabela 5.2), assim como seus gradientes, e fatores como *momentum* que são usados por determinados otimizadores. Na prática, os valores da tabela 5.2 devem ser multiplicados por 3 ou mais, dependendo do otimizador.

- Outros dados que aumentem o uso de memória, como por exemplo o conjunto

de dados que está carregado em memória.

Assim, para ter uma estimativa razoável da memória necessária para o treinamento do modelo, considerando uma implementação inteligente que aloque apenas a memória necessária para as saídas intermediárias nos momentos em que são requisitadas, e sem contar a memória utilizada pelo conjunto de dados, multiplicam-se estes valores por um fator de 3, formado pela memória dos parâmetros, seus gradientes, e o *momentum* (ou outro fator similar) do otimizador.

As Tabelas 5.3 e 5.4 a seguir mostram os resultados obtidos para os três problemas mencionados (MR, TREC e PortTwitter) para cada técnica de *word embedding* testada (W2V para o *word2vec*, e BERT) e configuração de rede neural utilizada. Antes de começar a interpretar os resultados das tabelas, é importante lembrar que os ensaios do BERT, por limitações de tempo, só consideraram um único conjunto de teste, constituído por 10% dos dados disponíveis. Já o TREC possui um conjunto de teste disponível, e por isso também foi treinado e avaliado uma única vez. Por esse motivo, estes dois casos não possuem desvio padrão.

	MR	**TREC**	**PortTwitter**
W2V Multichannel	0,765 $\pm 0,021$	0,769	0,621 $\pm 0,066$
W2V Static	0,762 $\pm 0,016$	0,823	0,630 $\pm 0,032$
W2V Non-Static	0,778 $\pm 0,011$	0,896	0,601 $\pm 0,049$
Rand	0,748 $\pm 0,013$	0,878	0,723 $\pm 0,026$
BERT Multichannel	**0,813**	0,912	**0,804**
BERT Static	0,627	0,335	0,541
BERT Non-Static	0,812	**0,913**	0,718

Tabela 5.3: Resultados do F1 score dos modelos para os diferentes conjuntos de dados.

Pelos resultados obtidos nas Tabelas 5.3 e 5.4, percebemos que em muitos casos o uso do BERT melhora bastante o desempenho do modelo. Além disso, com relação às configurações, percebe-se que, em geral, a configuração "Static"costuma ter o pior desempenho quando comparada a outras configurações (exceto no caso do PortTwitter utilizando *word2vec*). Já o "Multichannel"e o "Non-static"podem ser melhores ou piores, dependendo do conjunto de dados.

	MR	**TREC**	**PortTwitter**
W2V Multichannel	0,766 $\pm 0,019$	0,772	0,695 $\pm 0,049$
W2V Static	0,762 $\pm 0,016$	0,832	0,698 $\pm 0,028$
W2V Non-Static	0,779 $\pm 0,011$	0,898	0,684 $\pm 0,040$
Rand	0,748 $\pm 0,012$	0,879	0,749 $\pm 0,020$
BERT Multichannel	**0,813**	0,912	**0,821**
BERT Static	0,627	0,358	0,638
BERT Non-Static	0,812	**0,914**	0,722

Tabela 5.4: Resultados da acurácia dos modelos para os diferentes conjuntos de dados.

Curiosamente, a configuração "Static", utilizando BERT, apresenta os menores desempenhos nos conjuntos de dados testados, apesar dos melhores resultados também serem obtidos através do BERT. Isso sugere que o BERT não é adequado para ser utilizado sem ajuste, mas que ao ajustá-lo minimamente ao conjunto de dados, este é bem sucedido, em poucas épocas, em aumentar significativamente seu desempenho. A questão da diferença de desempenho de um *embedding* ajustado e não-ajustado pode ser aprofundada em [27].

Também podemos notar que, no caso do PortTwitter, o desempenho é quase sempre pior do que nos conjuntos de dados em inglês, possuindo também um desvio padrão maior. A causa disso não é clara, podendo ser atribuída tanto a uma diferença entre as línguas, quanto a uma menor quantidade de dados deste conjunto em relação aos outros, ou um possível maior desbalanceamento entre as classes.

Uma outra causa possível é o fato do modelo *word2vec* pré-treinado que foi utilizado não incluir uma grande quantidade de termos presentes neste conjunto de dados. Para o PortTwitter, um número alarmante de 87% das palavras não possuem um *embedding* no modelo pré-treinado, e, portanto, são inicializados de forma aleatória. Essa porcentagem é de 19% e 18% para o MR e o TREC, respectivamente. Todavia, isto não explica de imediato por qual motivo a configuração totalmente aleatória (RAND) bate todas as outras, com exceção do BERT Multichannel.

Também vale ressaltar que, para este mesmo conjunto de dados em português, foram obtidas desempenhos melhores ao se utilizar técnicas mais simples, como visto em [23]. Uma maior experimentação nestes dados deve ser conduzida para se

descobrir a causa disso.

	MR	**TREC**	**PortTwitter**
W2V Multichannel	18,69s	7,57s	3,77s
W2V Static	7,08s	4,08s	2,01s
W2V Non-Static	17,07s	6,48s	3,36s
Rand	17,07s	6,54s	3,40s
BERT Multichannel	30m16s	13m09s	5m11s
BERT Static	7m37s	2m57s	1m07s
BERT Non-Static	22m10s	9m53s	4m01s

Tabela 5.5: Tempo de treinamento dos modelos para os diferentes conjuntos de dados em segundos/epoca.

	MR	**TREC**	**PortTwitter**
W2V Multichannel	1,07s	0,5s	0,23s
W2V Static	0,76s	0,375s	0,18s
W2V Non-Static	0,66s	0,33s	0,14s
Rand	0,66s	0,33s	0,14s
BERT Multichannel	1m57s	59s	45s
BERT Static	50s	22s	15s
BERT Non-Static	1m06s	37s	30s

Tabela 5.6: Tempo de predição dos modelos para os diferentes conjuntos de dados em segundos. A predição para o conjunto de dados inteiro é contabilizada.

As Tabelas 5.5 e 5.6 mostram, respectivamente, os tempos de treinamento e predição dos modelos. Como era de se esperar, o BERT, por se tratar de um modelo mais complexo, necessita de uma quantidade de tempo consideravelmente maior do que o *word2vec*. Isso pode fazer com que, em determinados casos onde o tempo é um fator crucial, não seja viável a utilização do BERT, apesar de seu desempenho estatístico ser superior.

Capítulo 6

Discussão e Conclusões

Relativamente à proposta inicial do trabalho, conclui-se que o uso do BERT ao invés do clássico *word2vec* ajuda a aumentar, nos casos analisados, o desempenho estatístico da rede neural convolucional - embora com um custo maior de tempo e memória. As duas técnicas aplicadas a diferentes conjuntos de dados e configurações do modelo apresentam não apenas mudanças em relação às métricas escolhidas, mas também diferenças substanciais relativamente ao tempo e ao quantitativo de memória necessários para o treinamento e predição dos modelos.

Em determinados casos de uso, pode-se dar um peso maior para um fator do que a outro. Por exemplo, para aplicações onde as predições ou o treinamento devem ser feitos em tempo real, é importante que o tempo de predição/treinamento seja baixo, mesmo que para isso o desempenho do modelo escolhido seja menor. Já em outros casos, onde a performance do modelo é primordial (por exemplo, em aplicações médicas), deve-se escolher um modelo que apresente uma menor taxa de erros, mesmo que ele não seja o mais rápido, contanto que ele realize seus cálculos em um tempo ainda viável.

A maior parte dos trabalhos não analisa estes dois aspectos, se contentando, muitas vezes, em reportar apenas o desempenho estatístico. Neste contexto, este trabalho se apresenta como uma iniciativa em se comparar as soluções também considerando o custo computacional dos algoritmos no campo da ciência de dados.

Ademais, a literatura muitas vezes deixa de citar aspectos importantes para a análise (e replicabilidade) desses modelos, como por exemplo o número de épocas de treinamento utilizadas, ou até mesmo a métrica escolhida para avaliar os resultados.

Para expandir este trabalho, alguns pontos podem ser mais explorados. Primeiramente, poder-se-ia realizar um maior trabalho de validação cruzada para a escolha dos hiper-parâmetros dos modelos, o número de filtros de convolução, entre outros, que não foram otimizados. Possivelmente, ao escolher valores mais sintonizados com o problema, poderíamos ver uma diminuição (ou mesmo um aumento) na diferença de desempenho entre os modelos.

Também nesse sentido, no caso do BERT, pode valer a pena realizar uma validação cruzada utilizando 10-*Folds* (como foi feito com *word2vec* neste trabalho), para que os resultados sejam melhor comparáveis entre as duas técnicas, e tenhamos uma melhor ideia da variabilidade dos resultados, mas para isso deve-se visar a utilização de *hardware* ainda mais potente. Nesta mesma linha, também poderíamos aumentar o número de épocas pelas quais os modelos são treinados e assim, idealmente, chegar mais próximo ao ponto de convergência destes.

Além disso, testes de modelos envolvendo uma quantidade maior de conjuntos de dados, de diferentes naturezas, poderia evidenciar os domínios no quais tal ou tal modelo são melhores. Conjuntos de documentos maiores, onde uma mesma palavra pode ter significados completamente distintos, provavelmente seriam melhor analisados pelo BERT, enquanto que conjuntos restritos, onde cada palavra tem um sentido mais ou menos único, a diferença talvez não compense o custo computacional associado.

Referências Bibliográficas

[1] LIU, H., MOTODA, H., *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007.

[2] SEBASTIANI, F., "Text Categorization", , 10 2003.

[3] RUSSELL, S., NORVIG, P., *Artificial Intelligence: A Modern Approach*. 3 ed. Upper Saddle River, NJ, USA, Prentice Hall Press, 2009.

[4] STAMATATOS, E., "A survey of modern authorship attribution methods", *Journal of the American Society for Information Science and Technology*, v. 60, n. 3, pp. 538–556, 2009.

[5] SULEA, O., ZAMPIERI, M., MALMASI, S., *et al.*, "Exploring the Use of Text Classification in the Legal Domain", *CoRR*, v. abs/1710.09306, 2017.

[6] I. JORDAN, M., "On Computational Thinking, Inferential Thinking and Data Science", Disponível em:https://bids.berkeley.edu/events/computational-thinking-inferential-thinking-and-data-science, acessado em 05/05/2019.

[7] KIM, Y., "Convolutional Neural Networks for Sentence Classification", *CoRR*, v. abs/1408.5882, 2014.

[8] APTé, C., WEISS, S., "Data Mining with Decision Trees and Decision Rules", *Future Generation Computer Systems*, v. 13, pp. 197–210, 11 1997.

[9] WALTL, B., BONCZEK, G., MATTHES, F., "Rule-based Information Extraction: Advantages, Limitations, and Perspectives", *Jusletter IT, 22*, , 2 2018.

[10] MURPHY, K. P., *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.], MIT Press, 2013.

[11] BISHOP, C. M., *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg, Springer-Verlag, 2006.

[12] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., *The Elements of Statistical Learning*, Springer Series in Statistics. New York, NY, USA, Springer New York Inc., 2001.

[13] MIKOLOV, T., CHEN, K., CORRADO, G., *et al.*, "Efficient Estimation of Word Representations in Vector Space", *CoRR*, v. abs/1301.3781, 2013.

[14] LIU, P., QIU, X., HUANG, X., "Recurrent Neural Network for Text Classification with Multi-Task Learning", *CoRR*, v. abs/1605.05101, 2016.

[15] LAI, S., XU, L., LIU, K., *et al.*, "Recurrent Convolutional Neural Networks for Text Classification". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 2267–2273, 2015.

[16] VASWANI, A., SHAZEER, N., PARMAR, N., *et al.*, "Attention Is All You Need", *CoRR*, v. abs/1706.03762, 2017.

[17] PENNINGTON, J., SOCHER, R., MANNING, C. D., "Glove: Global vectors for word representation". In: *In EMNLP*, 2014.

[18] PETERS, M. E., NEUMANN, M., IYYER, M., *et al.*, "Deep contextualized word representations". In: *Proc. of NAACL*, 2018.

[19] HOWARD, J., RUDER, S., "Fine-tuned Language Models for Text Classification", *CoRR*, v. abs/1801.06146, 2018.

[20] DEVLIN, J., CHANG, M., LEE, K., *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *CoRR*, v. abs/1810.04805, 2018.

[21] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C. J. C., Bottou, L., *et al.* (eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1097–1105, 2012.

[22] GOODFELLOW, I., BENGIO, Y., COURVILLE, A., *Deep Learning*. MIT Press, 2016. Disponível em: http://www.deeplearningbook.org, acessado em 05/05/2019.

[23] AGUIAR, E. J. D., FAIçAL, B. S., UEYAMA, J., *et al.*, "Análise de Sentimento em Redes Sociais para a Língua Portuguesa Utilizando Algoritmos de Classificação", .

[24] "Dataset MR", Disponível em: https://www.cs.cornell.edu/people/pabo/movie-review-data/, acessado em 05/05/2019.

[25] "Dataset TREC", Disponível em: http://cogcomp.org/Data/QA/QC/, acessado em 05/05/2019.

[26] "BERT Github Page", Disponível em: https://github.com/google-research/bert, acessado em 05/05/2019.

[27] PETERS, M., RUDER, S., SMITH, N. A., "To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks", *CoRR*, v. abs/1903.05987, 2019.

Apêndice A

Apêndice A

Matrizes de Confusão

Para fins de referência, as matrizes de confusão para cada um dos conjuntos de dados, técnica de *word embedding*, e configuração da rede neural serão colocadas aqui. Em algumas matrizes, os valores podem estar quebrados. Isso acontece porque, nos casos em que uma validação cruzada de 10-Fold é utilizada, a matriz de confusão gerada é a média das matrizes de confusão de cada um dos *folds*.

A.1 Matrizes para o conjunto de dados MR

412,2	120,9
147,2	385,9

Tabela A.1: Matriz de confusão para o conjunto de dados MR, utilizando *word2vec*, na configuração RAND.

392,7	140,4
112,5	420,6

Tabela A.2: Matriz de confusão para o conjunto de dados MR, utilizando *word2vec*, na configuração STATIC.

347,0	203,0
194,0	323,0

Tabela A.3: Matriz de confusão para o conjunto de dados MR, utilizando BERT, na configuração STATIC.

395,1	138,0
111,2	421,9

Tabela A.4: Matriz de confusão para o conjunto de dados MR, utilizando *word2vec*, na configuração MULTICHANNEL.

424,0	99,0
100,0	444,0

Tabela A.5: Matriz de confusão para o conjunto de dados MR, utilizando BERT, na configuração MULTICHANNEL.

421,3	111,8
123,8	409,3

Tabela A.6: Matriz de confusão para o conjunto de dados MR, utilizando *word2vec*, na configuração NONSTATIC.

436,0	102,0
98,0	431,0

Tabela A.7: Matriz de confusão para o conjunto de dados MR, utilizando BERT, na configuração NONSTATIC.

A.2 Matrizes para o conjunto de dados TREC

135	2	0	0	1	0
11	73	0	3	0	7
3	0	4	0	0	2
0	5	0	60	0	0
6	8	0	0	95	4
2	5	0	1	0	73

Tabela A.8: Matriz de confusão para o conjunto de dados TREC, utilizando *word2vec*, na configuração RAND.

138	0	0	0	0	0
35	53	0	3	1	2
8	1	0	0	0	0
0	7	0	57	0	1
9	2	0	0	102	0
12	2	0	1	0	66

Tabela A.9: Matriz de confusão para o conjunto de dados TREC, utilizando *word2vec*, na configuração STATIC.

74	36	0	18	7	3
26	36	0	24	2	6
4	1	0	4	0	0
9	12	0	40	1	3
29	27	0	34	20	3
12	18	0	32	10	9

Tabela A.10: Matriz de confusão para o conjunto de dados TREC, utilizando BERT, na configuração STATIC.

69	68	0	0	1	0
12	76	0	3	0	3
4	5	0	0	0	0
0	5	0	59	0	1
3	2	0	1	105	2
1	3	0	0	0	77

Tabela A.11: Matriz de confusão para o conjunto de dados TREC, utilizando *word2vec*, na configuração MULTICHANNEL.

133	3	0	0	0	2
6	79	0	3	4	2
0	1	8	0	0	0
1	3	0	58	0	3
0	0	0	0	101	12
0	2	0	2	0	77

Tabela A.12: Matriz de confusão para o conjunto de dados TREC, utilizando BERT, na configuração MULTICHANNEL.

135	2	0	0	1	0
17	68	0	4	1	4
3	0	6	0	0	0
0	4	0	60	0	1
3	3	0	0	106	1
3	4	0	0	0	74

Tabela A.13: Matriz de confusão para o conjunto de dados TREC, utilizando *word2vec*, na configuração NONSTATIC.

138	0	0	0	0	0
5	83	0	2	1	3
3	0	5	0	0	1
7	1	0	56	1	0
3	1	2	0	104	3
4	1	0	5	0	71

Tabela A.14: Matriz de confusão para o conjunto de dados TREC, utilizando BERT, na configuração NONSTATIC.

A.3 Matrizes para o conjunto de dados PortTwitter

2,3	6,6	7,7
0,4	29,3	25,6
0,8	9,4	119,7

Tabela A.15: Matriz de confusão para o conjunto de dados PortTwitter, utilizando *word2vec*, na configuração RAND.

0,0	2,4	14,2
0,0	15,5	39,8
0,0	4,4	125,5

Tabela A.16: Matriz de confusão para o conjunto de dados PortTwitter, utilizando *word2vec*, na configuração STATIC.

0,0	1,0	18,0
0,0	6,0	48,0
0,0	6,0	123,0

Tabela A.17: Matriz de confusão para o conjunto de dados PortTwitter, utilizando BERT, na configuração STATIC.

0,0	1,9	14,7
0,0	14,3	41,0
0,0	3,8	126,1

Tabela A.18: Matriz de confusão para o conjunto de dados PortTwitter, utilizando *word2vec*, na configuração MULTICHANNEL.

4,0	6,0	9,0
1,0	40,0	9,0
0,0	11,0	122,0

Tabela A.19: Matriz de confusão para o conjunto de dados PortTwitter, utilizando BERT, na configuração MULTICHANNEL.

0,0	1,5	15,1
0,0	10,8	44,5
0,1	2,5	127,3

Tabela A.20: Matriz de confusão para o conjunto de dados PortTwitter, utilizando *word2vec*, na configuração NONSTATIC.

5,0	5,0	6,0
5,0	35,0	19,0
3,0	18,0	106,0

Tabela A.21: Matriz de confusão para o conjunto de dados PortTwitter, utilizando BERT, na configuração NONSTATIC.