INVESTIGANDO CODIFICAÇÕES VETORIAIS CONTÍNUAS DE PALAVRAS NO CONTEXTO DE DEEP LEARNING

Daniel Teixeira Rodrigues e Orlando Bisacchi Coelho

Apoio: PIVIC Mackenzie

RESUMO

No contexto atual de Processamento de Linguagem Natural (PLN), na intersecção dos paradigmas de Aprendizagem de Máquina e Semântica Vetorial, existe o desafio de codificar uma palavra com o máximo de valor linguístico possível. Esse artigo apresenta uma análise das soluções disponíveis para codificações vetoriais contínuas de palavras (*word embeddings*) – um modelo de codificação onde cada palavra é representada por um vetor de grande dimensão composto por números *float*, cada número codificando informações importantes sobre a palavra e o contexto no qual está inserida. Em específico, são exploradas as *word embeddings* desenvolvidas no contexto de *Deep Learning*, que são obtidas por meio de treinamento sobre grandes *corpora*. Foi realizado um experimento validando uma dessas representações (BERT), utilizando um modelo no estado-da-arte de PLN, o Transformer. Esse modelo foi treinado por meio de *Transfer Learning*, de modo que são reaproveitados seus milhões de parâmetros pré-treinados, e se faz um ajuste apenas nas últimas camadas para atender a tarefa específica. Para avaliação da eficácia do modelo, foi escolhida uma tarefa tal que a riqueza linguística capturada pela codificação escolhida para as palavras seja essencial para que a tarefa seja realizada a contento: a Pontuação Automática de Ensaios. O modelo treinado acabou obtendo uma acurácia bastante boa para a realização da tarefa, validando a capacidade de BERT para capturar e representar informação linguística. São também discutidos possíveis passos futuros para a possível melhoria do modelo.

Palavras-chave: *Deep Learning*. Processamento de Linguagem Natural. BERT.

ABSTRACT

In the current context of Natural Language Processing (NLP), in the intersection of Machine Learning and Vector Semantics, encoding a word with the maximum linguistic value possible is a big challenge. This article presents an analysis of some of the main approaches available for developing word embeddings – a form of encoding where each word is represented by an array with high dimensionality composed by float numbers, each number encoding important information about the word and the context in which it is located. Specifically, word embeddings developed using Deep Learning are explored, which are obtained by training networks with large amounts of text. It also presents an experiment validating one of those representations, BERT, carried out with a state-of-the-art NLP model, the Transformer. The model was trained

by Transfer Learning, in such a way that most of the model's pretrained parameters were reused, and only their last layers were fine tuned to attend to the specific task. To evaluate the efficacy of the model, a task where the linguistic richness captured by the chosen encodings is essential for the task to be completed successfully was chosen: Automated Essay Scoring. The model trained obtained a good level of accuracy for the given task, which validates the power of BERT to capture and represent linguistic information. Future steps that can lead to the improvement of the model are also presented.

Keywords: Deep Learning. Natural Language Processing. BERT.

1. INTRODUÇÃO

Um dos conceitos mais fundamentais em *Deep Learning* (LECUN; BENGIO; HINTON, 2015) é o de aprendizagem de representação (*representation learning*). O objetivo básico de todo processo de aprendizagem neural (em redes multicamadas) é o desenvolvimento de representações, nas camadas escondidas da rede, que permitam mapear as codificações recebidas pela rede, na entrada, nas codificações produzidas na saída. O que um algoritmo de aprendizagem neural, como *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986), por exemplo, faz é permitir o aprendizado dessas representações internas, ao aprender um conjunto adequado de valores para os pesos e *biases* da rede. Nesse sentido, fica claro que as codificações usadas nas entradas e alvos da rede podem facilitar ou dificultar o aprendizado da rede, o que é largamente constatado na prática (RUMELHART; HINTON; WILLIAMS, 1986).

Já na era *Deep Learning* da história das Redes Neurais Artificiais (a partir da metade dos anos 2000), enquanto as áreas de aplicação de *Deep Learning* em Processamento de Imagem e em Visão Computacional avançavam rapidamente, o progresso nas aplicações voltadas a Processamento de Linguagem Natural era mais lento. Constata-se, atualmente, em retrospecto, que uma das razões para isso foi a dificuldade em se construir representações dos elementos textuais que fossem ricas em informação e facilmente utilizáveis no contexto da aprendizagem neural. A entrada típica para uma rede que processa imagens é uma representação, num certo nível de resolução, similar à imagem (mesmo que separada em seus três canais de cores, se for o caso). Em suma, a imagem é um sinal, um símile do "real". Já em linguagem, o que é esse sinal? No reconhecimento de fala, sim, também é um sinal, uma onda sonora. Mas quando se trabalha com textos, não existe um tal sinal. Qual é a unidade

de informação: a letra, a palavra, as expressões compostas, as sentenças, o texto como um todo? Os símbolos são necessariamente arbitrários: os caracteres são arbitrários e dependentes do alfabeto usado; os sentidos às vezes estão nas palavras, mas nem sempre: numa expressão como “quem espera sempre alcança” há um sentido, inclusive cultural, que transcende as palavras per si. Além do mais, não só as palavras, mas a própria linguagem é polissêmica.

O grande desafio para o Processamento de Linguagem Natural baseado em *Deep Learning* (GOLDBERG, 2017) foi desenvolver uma representação para linguagem que fosse semanticamente rica e computacionalmente tratável; o que só foi alcançado ao longo desta última década. Mais precisamente, um processo que, pode-se dizer, começou em 2013 e ainda está em curso. Ao longo desse período uma série de codificações vetoriais contínuas de palavras (*word embeddings*) (GOLDBERG, 2017) foram desenvolvidas (como descrito a seguir, na seção 2). Cada um desses modelos, concebidos ao longo do tempo, ao ser usado como entrada para as respectivas redes *Deep Learning*, foi melhorando os resultados obtidos em benchmarks usados pela comunidade de Linguística Computacional (JURAFSKY; MARTIN, 2008) para medir a performance de solução computacionais em tarefas linguísticas fundamentais, como análise de sentimento, criação de *chatbots*, resposta automática a questões, resumo e tradução por máquina, entre outras. A riqueza linguística intrinsecamente capturada nessas codificações se traduz no ganho de performance medido, extrinsecamente, nessas tarefas de Linguística Computacional (WANG et al., 2019).

Uma tarefa de Processamento de Linguagem Natural que vêm recebendo a atenção de pesquisadores da área de *Deep Learning*, nos últimos anos, é a Pontuação Automática de Ensaios (AES – *automated essay scoring*) (SHERMIS; BURSTEIN; BURSKY, 2013). Essa tarefa, bastante sofisticada, consiste na análise de ensaios produzidos por aprendizes e sua pontuação, por um algoritmo de Aprendizagem de Máquina (RASCHKA; MIRJALILI, 2017), de modo a reproduzir a pontuação que seria atribuída por um ou mais corretores humanos.

No sentido de contribuir evidência de que a escolha de uma representação semanticamente rica para palavras pode facilitar o aprendizado de tarefas linguísticas sofisticadas, o presente trabalho buscará utilizar uma representação baseada em BERT para codificar os ensaios que serão, posteriormente, pontuados por meio de um classificador baseado em Transformer.

2. REFERENCIAL TEÓRICO

2.1. Deep Learning

O conceito mais básico por trás de Deep Learning é uma Rede Neural Artificial (RNA) (HAYKIN, 2008). Uma rede neural é uma replicação do modelo de sistema nervoso encontrado no corpo humano, de maneira extremamente simplificada. Cada neurônio é representado artificialmente como um vértice da rede e armazena um determinado valor. Cada neurônio é ligado a todos os neurônios da camada anterior da rede, e todos os neurônios da camada seguinte da rede, havendo um peso para cada conexão com cada outro neurônio. Dessa forma, cada neurônio tem determinada força de ligação com seus vizinhos, o que compromete na maneira como passa sua informação para eles. Um algoritmo de treinamento de RNA consegue atualizar esses valores dos neurônios e todos os pesos de todas as conexões da rede, de forma que a rede consiga codificar informações relevantes para a tarefa de treinamento, transformando a informação de entrada da rede numa informação com valor agregado na saída.

A maior parte da pesquisa e das aplicações em RNA centrou-se em aprendizagem supervisionada. Nesse paradigma, um algoritmo de aprendizagem pode usar a informação do erro obtido para cada entrada, computado pela diferença entre a saída produzida e a que se desejava produzir. Na maioria dos modelos desenvolvidos nesse paradigma, foi usado o algoritmo *backpropagation* (RUMELHART; HINTON; WILLIAMS,1986), um algoritmo que usa o sinal de erro gerado para cada padrão de treinamento para adaptar os pesos da rede, e se baseia na descida do gradiente de erro.

A partir de meados da década de 2000, com o surgimento de novos algoritmos de treinamento e arquiteturas de rede e, principalmente, novos desenvolvimentos em termos de hardware, tornou-se possível o treinamento de redes profundas (LECUN; BENGIO; HINTON, 2015), com várias dezenas de camadas e centenas de milhões de pesos, as quais aprendem tarefas descritas por um conjunto de treinamento de até milhões de exemplos. Esse novo paradigma de rede neural artificial foi denominado de *Deep Learning* (DL) (SCHMIDHUBER, 2014).

Essas redes profundas, com muitas camadas, só podem ser treinadas a partir de conjuntos de treinamento tão grandes graças ao desenvolvimento das unidades de processamento gráficas (GPUs) e à possibilidade de rodar esses algoritmos diretamente nas GPUs (STEINKRAUS; BUCK; SIMARD, 2005; RAINA; MADHAVAN;

NG, 2009). Para isso contribuem a existência de tecnologias e ambientes de programação como TensorFlow/Keras (TENSORFLOW; KERAS), e PyTorch (PYTORCH).

O avanço das redes de *Deep Learning* trouxe novas maneiras de enfrentar problemas com Inteligência Artificial. A aplicação de redes neurais ou *Deep Learning* ao Processamento Sequencial e ao Processamento de Linguagem Natural se baseava, até há muitos poucos anos, na arquitetura LSTM (HOCHREITER; SCHMIDHUBER, 1997). Um desenvolvimento recente em termos de arquiteturas *Deep Learning* para o processamento de linguagem natural é a classe de redes conhecida por *Transformer* (VASWANI et al., 2017). Essa arquitetura consiste numa rede profunda do tipo *encoder-decoder* (GOODFELLOW, BENGIO; COURVILLE, 2016). A rede *encoder-decoder* é uma rede *feedforward* treinada para reproduzir nas unidades de saída os padrões que recebe como entrada. Ela tem grande aplicação na aprendizagem de sequências. Daí advém suas aplicações em Processamento de Linguagem Natural, dado o aspecto sequencial da linguagem – como é o caso da arquitetura BERT (DEVLIN et al, 2018). A arquitetura *Transformer* estende a arquitetura *encoder-decoder* ao dotar a rede de um mecanismo de atenção, que aprende automaticamente a que elemento(s) da sequência atentar a cada etapa do processamento. BERT utiliza a rede *Transformer* para gerar representações vetoriais contínuas de palavras (*word embeddings*) (MIKOLOV et al., 2013) que capturam profunda riqueza semântica.

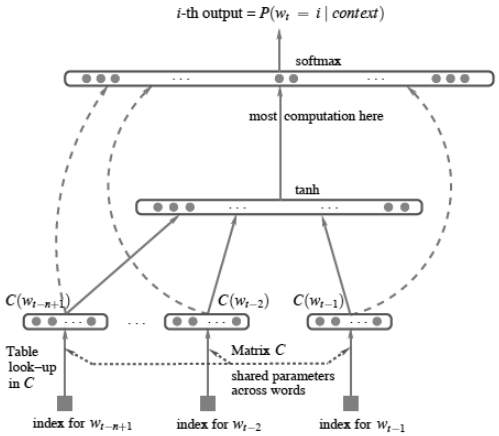
2.2. Codificações Vetoriais Contínuas de Palavras

Originalmente, a representação para palavras, em Redes Neurais Artificias (HAYKIN, 2008), era uma representação localista (RUMELHART; HINTON; WILLIAMS, 1986), também conhecida por 1-dentre-N ou *one-hot encoding*. Uma vez que se definisse o *corpus* com que se pretendia trabalhar e, eventualmente, se eliminasse as palavras e símbolos de pontuação que se desejava descartar, chegava-se a um total de N palavras distintas. Então, ordenava-se as palavras (na ordem lexicográfica, por exemplo) e associava-se a cada palavra um vetor N-dimensional, da seguinte forma: à primeira palavra se associava o vetor (1, 0, 0, ..., 0); à segunda palavra (0, 1, 0, 0, ..., 0) era o vetor associado; e assim por diante, até chegar à última palavra, à qual se associava o vetor (0, 0, 0, ..., 0, 1). Claramente, essa codificação é arbitrária, não havendo nela nada que capture o sentido ou o uso que se faz das palavras. Em particular, as codificações de todas as palavras têm uma distância

vetorial constante entre si, independente de algumas serem muito próximas e outras muito distantes entre si, ortográfica, sintática, semântica ou pragmaticamente. Em suma, não há nada de linguagem nessa representação linguística. Computacionalmente essa codificação também introduzia um sério problema: à medida que o tamanho do *corpus* aumenta, a dimensão dos vetores de entrada também aumenta. O que aumenta o número de parâmetros livres (pesos e *biases*) a serem ajustados pelo algoritmo de aprendizagem. O que requer mais exemplos para treinar a rede, e um *corpus* maior. Com essa codificação seria impossível trabalhar com *corpora* na classe de bilhões de palavras, como se faz hoje (MIKOLOV et al., 2018), os quais contém um número elevado de palavras distintas.

A Hipótese Distribucional assume que palavras que aparecem nos mesmos contextos tendem a ter os mesmos sentidos (GOLDBERG, 2017). Dessa forma, os sentidos das palavras podem ser apreendidos de seus contextos. Dar uma versão em termos de codificação neural foi o insight fundamental por trás do desenvolvimento das codificações vetoriais de palavras. A tarefa linguística de modelagem de linguagem (*language modeling*) (GOLDBERG, 2017) consiste em atribuir probabilidades à ocorrência de determinadas sentenças em uma dada linguagem; ou, alternativamente, calcular a probabilidade de uma dada palavra seguir uma certa sequência de palavras numa linguagem. Bengio e colegas (2003) usaram essa tarefa para desenvolver uma representação neural para palavras que capturasse seus respectivos contextos de uso. A ideia fundamental do trabalho foi: (i) associar a cada palavra no vocabulário um vetor de características de palavras, um vetor multidimensional com valores reais em suas várias coordenadas; (ii) expressar a probabilidade conjunta de certas palavras aparecerem numa certa sequência no *corpus* com os valores das representações vetoriais das palavras na sequência e (iii) aprender ao mesmo tempo as representações vetoriais e os parâmetros da função de probabilidade conjunta. A dimensão desses vetores é extremamente menor que a do vocabulário, o que representa uma clara vantagem computacional dessa abordagem. O vetor que codifica uma dada palavra é obtido como o vetor cuja i-ésima dimensão é o valor de um peso numa rede neural, o peso que liga o índice da palavra numa sequência de palavras à uma determinada unidade na camada escondida de uma rede *feedforward* como na Figura 1.

Figura 1. Codificação vetorial para palavras proposta por Bengio e colegas (2003).



Fonte: (Bengio et al., 2003).

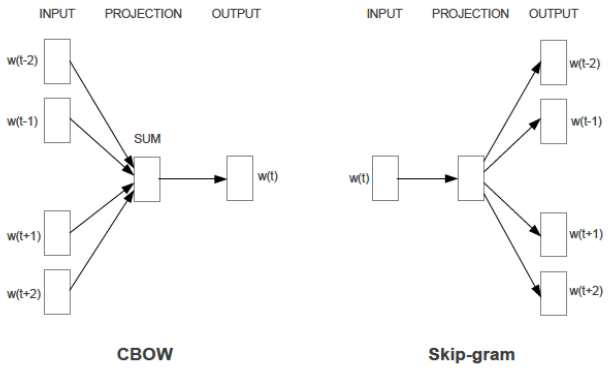
Se temos, num dado instante *t* de uso da linguagem, uma sequência de palavras w_{t-n+1}, w_{t-n+2}, ..., w_{t-2}, w_{t-1} e queremos calcular a probabilidade de que a palavra w_t seja a próxima, dentro de um conjunto de palavras que podem se seguir a essa sequência, podemos aprender os pesos de uma rede como a representada na Figura 1, onde a ativação da camada de saída seja dada pela função *softmax* e a função que mede o erro seja a entropia cruzada (GOODFELLOW; BENGIO; COURVILLE, 2016). Os valores dos *n* pesos que ligam o índice de uma palavra w_{t-i} às *n* unidades da camada escondida constituem a codificação vetorial proposta para a palavra w_{t-i}.

Collobert e Weston (2008) desenvolveram um modelo semelhante, só que baseado numa rede convolucional. No seu modelo as codificações das palavras são treinadas ao mesmo tempo em que a tarefa linguística nas quais as palavras serão usadas é aprendida.

A computação dessas codificações vetoriais era muito custosa para um *corpus* de porte razoável. A utilidade prática dessa proposta só surgiu a partir do trabalho de Mikolov e colegas (MIKOLOV et al., 2013). Eles introduziram uma série de simplificações que tornaram a aprendizagem da rede muito mais eficiente. Uma das simplificações introduzidas foi abandonar o cálculo preciso das probabilidades – o que exigia o cômputo da função entropia cruzada. Seguindo (COLLOBERT; WESTON, 2008), eles substituíram o cálculo da probabilidade de uma certa palavra ser a próxima

palavra num dado contexto pela distinção do que seriam contextos apropriados ou contextos inapropriados para uma dada palavra. A representação desenvolvida, dita word2vec, vem em dois sabores – CBOW e Skip-gram –, das quais a última mostrou-se mais apropriada. Conforme ilustrado na Figura 2, a tarefa a ser aprendida, na versão Skip-gram é identificar se o contexto representado pelas palavras $w_{(t-2)}$, $w_{(t-1)}$, ..., $w_{(t+1)}$, $w_{(t+2)}$ é um contexto apropriado para a palavra $w_{(t)}$.

Figura 2. Arquitetura de rede que gera a codificação word2vec.



Fonte: (MIKOLOV et al., 2013).

GloVe (*Global Vectors*) (PENNINGTON; SOCHER; MANNING, 2014) é uma abordagem alternativa a word2vec. Conforme os autores bem caracterizam, existem duas abordagens principais para aprender representações vetoriais de palavras. A mais usada, da qual vord2vec é paradigmática, utiliza contextos locais para as palavras. Uma possível limitação dessa abordagem é que ela não utiliza a totalidade das estatísticas presentes no *corpus*, já que o treinamento para o aprendizado da codificação de cada palavra é feito em uma série de contextos locais. Uma outra abordagem, que GloVe representa, pertence à tradição da análise da semântica latente (LSA – *8nalysi semantic 8nalysis*) (DEERWESTER et al., 1990), que busca capturar as co-ocorrências entre palavras que aparecem na totalidade do *corpus* (isto é, globalmente). Para isso, GloVe constrói uma matriz de co-ocorrências entre todas as palavras no *corpus*. Essa matriz é muito grande e esparsa, basicamente composta por zeros. Os autores introduzem uma série de manobras de forma a tornar o processamento dessa matriz mais eficiente. Uma delas consiste em só levar em consideração os elementos não-nulos da matriz. Se w é a palavra cuja codificação se quer aprender e c é o vetor que representa o contexto global em que ela aparece então GloVe busca minimizar

$$\sum_{i,j=1}^{V} f(X_{ij})\ (w\ c + \ b_w + \ b_c - \log X_{ij})^2 \qquad (1)$$

onde V é o tamanho do vocabulário do *corpus*, X_{ij} é o número de vezes que a j-ésima palavra aparece no contexto da i-ésima palavra (tal como representado na matriz de co-ocorrências), b_w e b_c são *biases* para a palavra e seu contexto, respectivamente, valores a serem aprendidos, e f é uma função de ponderação que atribui mais peso aos itens que mais aprecem no *corpus*.

De modo geral, word2vec e GloVe produzem representações equivalentes em termo de adequação às tarefas em que são usadas, e em termos de esforço computacional no treinamento sobre o *corpus*.

fastText (MIKOLOV et al., 2018) é uma evolução de word2vec, que opera no nível dos caracteres que formam as palavras. Com isso a representação consegue capturar informação sobre prefixos e sufixos; e também sobre palavras raras ou compostas (como “New York Times”) que não pertencem ao *dataset* sobre o qual o modelo foi treinado, mas podem eventualmente ser decompostas em palavras que estavam presentes no *dataset*.

BERT (*Bidirectional Encoder Representations from Transformers*) (DEVLIN et al, 2018; GOOGLE-RESEARCH, 2019) e suas variantes posteriores representa o estado-da-arte em codificações vetoriais para palavras. BERT usa, para gerar as codificações, uma arquitetura de DL relativamente nova, o *Transformer* (VASWANI et al., 2017). BERT funciona sobre o *Transformer*, de modo que treinar uma tarefa sobre palavras codificadas em BERT consiste em adicionar um nível final, específico para a tarefa, ao *Transformer* já treinado. Por exemplo, se a tarefa que se pretende realizar é uma classificação, como no caso de reconhecimento de entidade mencionada (NER – *name-entity recognition*), por exemplo, basta adicionar ao *Transformer* no qual as codificações BERT foram geradas uma camada de unidades de saída com ativação *softmax* e treinar a rede usando a medida de erro entropia cruzada (GOODFELLOW; BENGIO; COURVILLE, 2016). Por outro lado, existem métodos para extrair codificações vetoriais para palavras, treinadas sobre *corpora* de grande porte, a partir das ativações geradas em uma das primeiras camadas do *Transformer*.

Cumpre observar que há um certo número de outras representações vetoriais contínuas de palavras foi proposto além das descritas acima, dentre as quais vale destacar ULMFit (HOWARD; RUDER, 2018) e ELMo (ELMO, 2019).

O conceito de modelos pré-treinados em linguagem é um exemplo de transferência de aprendizagem (*transfer learning*) (BENGIO, 2012), que consiste em usar o conhecimento adquirido numa determinada tarefa específica dentro de um domínio de aplicação para uma tarefa distinta, mas ainda no mesmo domínio ou modalidade. No caso de modelos pré-treinados em linguagem (DAI e LE, 2015), trata-se de primeiro treinar codificações vetoriais de palavras em um contexto bastante geral, sobre um *corpus* extremamente amplo, e então usar essas codificações de palavras como entrada para uma tarefa linguística bem específica. Hoje existem vários *corpora* de grande porte disponíveis, e as soluções que geram as codificações vetoriais podem ser treinadas de forma semiautomática nesses *corpora*, sem nenhuma necessidade de se rotular palavras ou expressões; isto é, de forma não supervisionada (semi-supervisionada, na verdade). Atualmente, uma série de codificações vetoriais de palavras, geradas por vários métodos distintos – word2vec, GloVe, fastText, BERT e ELMo, entre outros –, treinadas em *corpora* de grande porte, em vários idiomas, usando recursos computacionais que estão fora do alcance da maioria dos pesquisadores, estão disponibilizados para *download* gratuito.

2.3. Contexto do *Corpus* Utilizado

Para treinar e avaliar a acurácia de um modelo de codificações vetoriais contínuas de palavras, se faz necessário escolher uma tarefa de processamento de linguagem natural. Para a pesquisa, a tarefa escolhida foi a Pontuação Automática de Ensaios (AES – *automated essay scoring*) (SHERMIS; BURSTEIN; BURSKY, 2013), por ser uma tarefa de ampla relevância atual e por ser possível se realizar experimentos de aprendizagem supervisionada com dados para a tarefa.

Pontuação Automática de Ensaios (AES – *automated essay scoring*) (SHERMIS; BURSTEIN; BURSKY, 2013) consiste na análise de ensaios produzidos por estudantes e a obtenção de sua pontuação por meio de um algoritmo de Aprendizagem de Máquina, de modo a reproduzir a pontuação que seria atribuída por um ou mais corretores humanos. Tendo lançado a ideia de AES em 1966, Page liderou a construção do primeiro sistema para correção automática de ensaios, em 1973. Em 1999, o produto comercial IntelliMetric (ELLIOT, 1999) foi lançado. Ele media mais de 300 características sintáticas, semânticas e discursivas para construir sua pontuação. Intelligent Essay Assessor (LANDAUER; FOLTZ; LAHAM, 2003), em contraste, usava Análise de Semântica Latente (LSA – *Latent Semantic Analysis*) (LANDAUER; LAHAM; FOLTZ, 1998) para criar uma representação semântica dos tópicos do ensaio

e poder, então, identificar os ensaios já avaliados que são mais parecidos com aquele sob análise, de modo a poder lhe atribuir uma nota. Já o *e-rater* (ATTALI; BURSTEIN, 2006) conseguiu incorporar conceitos mais próximos aos usados por avaliadores humanos, tais como variedade de sentenças, sofisticação de vocabulário e organização do texto.

A pesquisa e desenvolvimento em correção automática de ensaios recentemente se beneficiou do Automated Essay Assessment Prize (ASAP-AES, 2012; SHERMIS, 2014). A terceira melhor solução desenvolvida nesse contexto foi o EASE, que é a base da solução automatizada ora usada pelo edX (MITROS et al., 2013).

Ao longo dos anos de 2016 a 2018, alguns trabalhos baseados em técnicas de *Deep Learning* foram desenvolvidos sobre essa base. Alikaniotis e colegas (ALIKANIOTIS; YANNAKOUDAKIS: REI, 2016) exploraram as arquiteturas LSTM unidirecionais, LSTM bidirecionais e *Stacked* LSTMs (LSTMs empilhadas) para desenvolver um AES. Para representar as palavras contidas em cada ensaio eles exploraram duas representações vetoriais: a desenvolvida por Collobert e Weston (2008) e a word2vec. A melhor arquitetura que eles encontraram em seus experimentos foi a BLSTM com duas camadas de células LSTM. Tang, Peterson e Pardos (2016) usaram o mesmo conjunto de dados para desenvolver uma solução para uma tarefa distinta: prever uma palavra seguinte adequada, após um trecho do ensaio. Porém, a acurácia obtida na tarefa foi muito baixa (21%).

De volta à aplicação AES, Dong e Zhang (2016) desenvolveram uma rede CNN empilhada hierárquica (GOODFELLOW, BENGIO; COURVILLE, 2016), com duas camadas, onde numa camada mais inicial do processamento representando a estrutura da sentença e uma camada mais superior representando a estrutura do ensaio. Taghipour e Ng (2016) partiram de representações *one-hot*, não contínuas, para as palavras no *corpus*. Eles implementaram uma camada que funciona como uma *look-up table*, fazendo uma transformação das representações *one-hot* para representações contínuas, a partir de uma matriz de pesos a ser aprendida. E implementaram uma camada convolucional, que permite extrair características locais dos vetores produzidos na camada *look-up table*. Uma camada LSTM foi usada para integrar a informação do ensaio como um todo. Antes de enviar as ativações das células LSTM para uma única unidade com ativação logística, a saída das células LSTM era projetada numa camada que calcula a média ao longo do tempo (*mean-over-time layer*) das ativações das células LSTM até o respectivo instante de tempo. Os resultados obtidos foram um pouco melhores dos que os reportados em

(ALIKANIOTIS; YANNAKOUDAKIS: REI, 2016), para o caso de ensaios não muito curtos. Para os ensaios relativamente curtos não foram obtidos bons resultados.

3. METODOLOGIA

Para treinar codificações vetoriais de palavras no contexto de Pontuação Automática de Ensaios, foi escolhido o *dataset* The Hewlett Foundation: Automated Essay Scoring (HEWLETT; 2012), publicado como parte de uma competição no portal de competições de Ciência de Dados Kaggle (KAGGLE). A competição, que ocorreu de fevereiro a março de 2012, ofereceu prêmios de US$ 60,000 à equipe vencedora, U$ 30,000 à segunda colocada e U$ 10,000 à terceira. O *dataset* oferece a maior quantidade publicamente disponível de dados rotulados para a tarefa até o momento, contando com cerca de 13000 ensaios no total, tendo de 150 a 550 palavras cada, e todos com notas de pelo menos três avaliadores diferentes. Todos os ensaios são anonimizados. O conjunto de dados consiste em oito séries de ensaios diferentes, cada série sendo sobre um tema específico, de tal forma que o experimento de codificação de palavras fica especialmente interessante por haver uma forte homogeneidade linguística presente nos seus textos.

O modelo escolhido para o treinamento das codificações de palavras foi um da família dos *Transformers*, em específico o DistilBERT (SANH et al., 2019), uma versão mais leve do modelo do atual estado-da-arte para processamento de linguagem natural, o BERT. Com o poder computacional disponível para a pesquisa, não seria viável realizar um treinamento de uma rede neural com o BERT. O DistilBERT mostra que é possível destilar o conhecimento presente no BERT na fase de pré-treino para obter um modelo 40% menor, que mantém 97% de suas capacidades de processamento de linguagem natural, e que pode ser treinado 60% mais rápido. A implementação do DistilBERT escolhida para uso foi a implementação em Python, publicada pela comunidade de inteligência artificial e processamento de linguagem natural Hugging Face (HUGGING FACE). Em específico, uma variação da implementação base que já foi previamente ajustada para realizar classificações entre mais de uma classe, como é o caso da tarefa que foi escolhida, uma classificação do ensaio entre diferentes notas. A Hugging Face disponibiliza implementações do modelo nos dois principais *frameworks* de inteligência artificial para Python: PyTorch (PYTORCH) e TensorFlow (TENSORFLOW). Foi decidido utilizar a implementação em PyTorch, por já estar sendo muito mais utilizada na comunidade em comparação com o TensorFlow. Além disso, há muito mais conteúdos gratuitos, como artigos e

blogs, disponíveis *online* para ensino de implementações de modelos de PLN com PyTorch em comparação com TensorFlow.

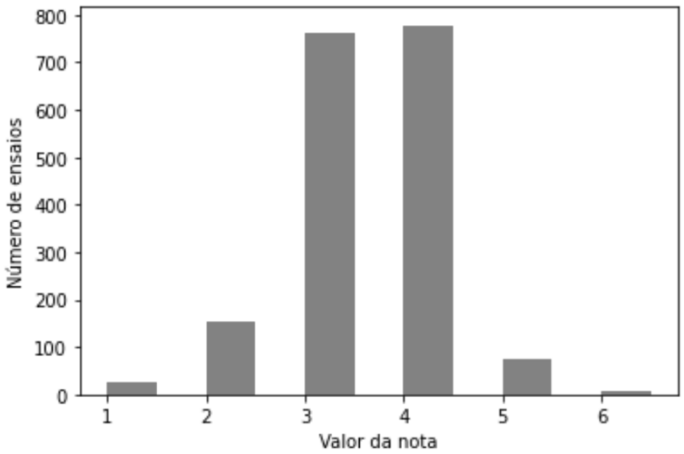
Para que o modelo seja treinado com ensaios sobre o mesmo tema, uma das oito séries de ensaios disponíveis no *dataset* da Hewlett Foundation foi escolhida para ser utilizada no experimento, contando com cerca de 1800 ensaios. Essa série foi escolhida pois seus ensaios têm a menor dependência da proposta de tema de ensaio. Como a informação da proposta de tema não é fornecida para o modelo em momento algum, não é interessante que os ensaios tenham alta dependência ou façam referência à proposta de tema, pois assim parte da riqueza linguística dos ensaios seria perdida durante a codificação de palavras. No caso dessa série, a proposta era redigir um ensaio sobre censura em bibliotecas – foi pedido um texto de opinião sobre a aplicação de censura em materiais como livros, músicas e filmes, e se deveriam ser removidos das prateleiras caso sejam considerados ofensivos. Os ensaios dessa série possuem notas discretas variando de 1 a 6 no âmbito de aplicação da escrita, e notas variando de 1 a 4 no âmbito de convenção linguística. Foi decidido utilizar apenas as notas de aplicação da escrita, pois as convenções não são especialmente interessantes para a captura de valor linguístico das palavras. Houve um trabalho prévio de tratar os dados, separar as notas a serem utilizadas, e juntar tudo em uma planilha em formato CSV que seria futuramente usada para o treinamento. Uma visão do *dataset* final e da distribuição de suas notas está disponível nas figuras 3 e 4, respectivamente.

Figura 3. Detalhe do *dataset* utilizado.

grade	essay
4	Certain materials being removed from libraries...
1	Write a persuasive essay to a newspaper reflec...
2	Do you think that libraries should remove cert...
4	In @DATE1's world, there are many things found...
4	In life you have the 'offensive things'. The l...

Fonte: (Os autores).

Figura 4. Distribuição das notas do *dataset* utilizado.



Fonte: (Os autores).

Para avaliar a acurácia de classificação do modelo treinado, é preciso verificar se o modelo classifica corretamente dados que nunca foram vistos por ele antes. Durante os experimentos, a acurácia foi avaliada em duas etapas, e para isso o *dataset* disponível foi particionado em três partes: 70% (1260) dos dados para treinamento, 15% (270) para validação, e 15% (270) para teste. Os dados de validação são utilizados durante o próprio treinamento: realiza-se uma classificação com cada dado de validação após cada época de treinamento, armazenando a acurácia da classificação durante o treinamento. Isso é feito para que não ocorra *overfitting*, ou seja, para que o modelo treinado não fique sobre ajustado ao conjunto de treinamento, perdendo assim parte de sua capacidade de generalização, deixando de funcionar a contento para novos exemplos nunca vistos durante a fase de treinamento. Já os dados de teste, também conhecido como teste cego, são utilizados no final de todo o processo de treinamento, avaliando a acurácia final do modelo.

4. RESULTADO E DISCUSSÃO

As melhores acurácias obtidas com os experimentos foram de em torno de 70%, o que não é um número ideal, mas já é um grande avanço quando comparado com a acurácia de uma classificação totalmente aleatória entre 6 classes, que seria de 16,6%. Os parâmetros utilizados para o treinamento se encontram na Figura 5, e uma visão do valor da função de perda de treinamento e validação, e da acurácia de validação se encontra na Tabela 1.

Figura 5. Melhores valores para os hiperparâmetros utilizados no treinamento do modelo.

```
training_args = TrainingArguments(
    num_train_epochs=11,               # total number of training epochs
    per_device_train_batch_size=8,  # batch size per device during training
    per_device_eval_batch_size=4,   # batch size for evaluation
    warmup_steps=5,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    load_best_model_at_end=True,     # load the best model when finished training (default metric is loss)
    logging_steps=3,                # log & save weights each logging_steps
    evaluation_strategy="steps",     # evaluate each `logging_steps`
)
```

Fonte: (Os autores).

Tabela 1. Resultados obtidos durante o treinamento.

Step	Training Loss	Validation Loss	Accuracy
100	1.134300	0.980601	0.570370
200	0.858500	0.810366	0.670370
300	0.771900	0.805410	0.696296
400	0.665400	0.952057	0.622222
500	0.623200	1.104286	0.577778
600	0.487400	0.996032	0.651852
700	0.386700	1.202809	0.633333
800	0.295600	1.302490	0.677778
900	0.148000	1.566715	0.637037
1000	0.120200	1.787845	0.637037
1100	0.094000	1.837299	0.662963
1200	0.074500	1.832226	0.674074
1300	0.015500	1.925615	0.685185
1400	0.036100	1.944972	0.700000
1500	0.020500	1.988922	0.696296
1600	0.007700	2.004767	0.692593
1700	0.014300	2.027094	0.688889

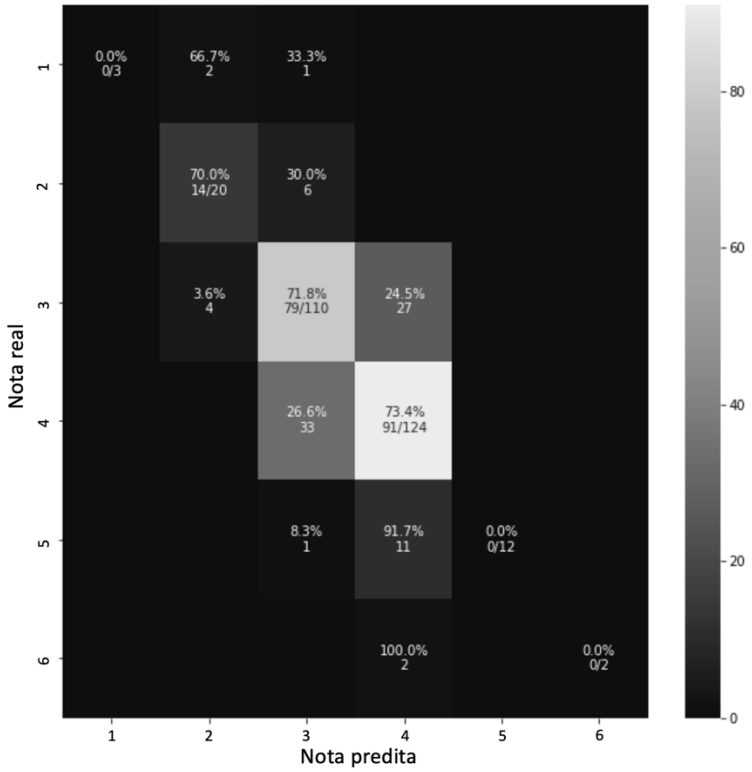
Fonte: (Os autores).

Foi escolhido o número de 11 épocas para treinamento do modelo, com levas de 8 ensaios sendo classificados antes de calcular a função de perda em cada época até totalizar o número total de ensaios, e levas de 4 ensaios para validações após cada época. A cada 100 passos de atualizações dos pesos do modelo, foi salva uma versão dos pesos do modelo, para que ao final do treinamento seja utilizada a versão com menor perda encontrada. Foi possível constatar que após a terceira época de treinamento, a perda de validação começou a aumentar, e a acurácia diminuir, o que significa que o modelo começou a ficar viciado nos dados de treinamento, fazendo predições muito específicas para ele e que não são generalizáveis para outros dados – um problema muito comum em *Deep Learning* conhecido como *overfitting*. Portanto, o modelo final escolhido após o treinamento foi o salvo após 300 passos de

atualizações de pesos, o qual teve a menor perda, e a acurácia de validação de 69,6%. Quando submetido aos dados de teste, o modelo apresentou uma acurácia de 67,9%.

No entanto, ao se gerar uma matriz de confusão (Figura 6) – uma visualização de como o modelo treinado realiza as predições – para as predições de teste, foi possível constatar um sério problema com o modelo: ele ficou enviesado para nunca tentar classificar um ensaio com as notas 1, 5 ou 6. O problema da não classificação das determinadas notas está relacionado com a pouca quantidade de ensaios com essas notas presente no *dataset* escolhido. Havendo pouca chance de uma predição para tais notas ser certeira, a função de perda e atualização de parâmetros fez com que o modelo fosse mais recompensado por nem sequer tentar predizer que um ensaio receba tais notas. Para as notas 3 e 4, o modelo apresentou a melhor acurácia, o que pode ser similarmente relacionado com a maior quantidade de ensaios com tais notas.

Figura 6. Matriz de confusão obtida com o teste final.



Fonte: (Os autores).

5. CONSIDERAÇÕES FINAIS

Com os experimentos realizados, foi possível utilizar a técnica do estado-da-arte em *Deep Learning* para codificações vetoriais de palavras, os *Transformers*. Por meio de Transfer Learning, foram reaproveitados os pesos do modelo DistilBERT, uma das implementações com mais eficácia de *Transformers*, e que é um pouco mais leve e, portanto, mais fácil de se treinar com menos recursos computacionais. Foi feito um

ajuste fino das últimas camadas do modelo, as camadas de classificação multi-classe, para que ele se adeque à tarefa de Pontuação Automática de Ensaios. Por fim, foi feita uma análise dos resultados do treinamento desse modelo e do que pode ainda ser melhorado para tal tarefa.

Dentre os resultados dos experimentos, podemos constatar que os modelos de *Transformers* pré-treinados podem ser facilmente adaptados para tarefas novas e nunca vistas antes, poupando um grande trabalho computacional. Isso mostra que, com o pré-treinamento de suas camadas iniciais, os modelos já possuem uma grande capacidade de generalização e entendimento da linguística. A parte de ajustes finos ainda demanda um grande processamento para tais modelos, mas já é algo bem mais factível, e que pode ser feito em poucos minutos ou horas com uma GPU popular compatível com o modelo e plataforma de programação paralela CUDA (CUDA).

O principal problema encontrado foi o da má distribuição das notas dos ensaios utilizados, o que gerou um modelo incapaz de fazer uma classificação para todas as notas, nunca classificando um ensaio com a nota 1, 5 ou 6.

Para trabalhos futuros, pode-se explorar técnicas de balanceamento de *datasets*, de meio a reduzir o problema obtido, ou utilizar uma função de perda que penalize mais as classificações errôneas para as classes com menos dados. Ademais, em busca de otimizar a acurácia seria interessante a realização de experimentos com outras variações de modelos de *Transformers*, como o próprio BERT, ou o RoBERTa (LIU et al., 2019).

6. REFERÊNCIAS

ALIKANIOTIS, Dimitrios; YANNAKOUDAKIS, Helen; REI, Marek. **Automatic Text Scoring Using Neural Network,** 2016. Disponível em: https://arxiv.org/abs/1606.04289. Acesso em: 6 set. 2021.

CUDA. **CUDA Zone**. https://developer.nvidia.com/cuda-zone. Acesso em: 7 set. 2021.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. In: 57th. Annual Meeting of the Association for Computational Linguistics (ACL 2019), Florence. Proceedings, Florence, 2019.

GOLDBERG, Yoav. **Neural Network Methods for Natural Language Processing**. Williston: Morgan & Claypool Publishers, 2017.

HAYKIN, Simon. **Neural Networks and Learning Machines**. 3. ed, Pearson, 2008.

HEWLETT, The Foundation. **Automated Essay Scoring**. Disponível em: https://www.kaggle.com/c/asap-aes/. Acesso em: 6 set. 2021.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation** v. 9, p. 1735–1780, 1997.

HOWARD, Jeremy; RUDER, Sebastian. **Universal Language Model Fine-tuning for Text Classification**. Proc. 56th. Ann. Meeting Association for Computational Linguistics, Melbourne, jul. 2018, p. 328-339.

HUGGING FACE. **The AI Community building the future**. https://huggingface.co/. Acesso em: 7 set. 2021.

JURAFSKY, Dan; MARTIN, James H. **Speech and Language Processing**. 2 ed., Upper Saddle River: Prentice Hall, 2008.

KAGGLE. **Kaggle**. https://www.kaggle.com/. Acesso em: 6 set. 2021.

KERAS. **Keras: Deep Learning library for Theano and TensorFlow**. https://keras.io/. Acesso em 21 mar. 2021.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey E. Deep Learning. **Nature**, v. 521, p. 436-444, 2015.

LIU, Yinhan; OTT, Myle; GOYAL, Naman; DU, Jingfei; JOSHI, Mandar; CHEN, Danqi; LEVY, Omer; LEWIS, Mike; ZETTLEMOYER, Luke; STOYANOV, Veselin. **RoBERTa: A Robustly Optimized BERT Pretraining Approach**, 2019. Disponível em: https://arxiv.org/abs/1907.11692. Acesso em: 7 set. 2021.

MIKOLOV, Tomas; CHEN, Kai; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. **Efficient Estimation of Word Representations in Vector Space**, 2013. Disponível em: https://arxiv.org/pdf/1301.3781.pdf. Acesso em: 18 mai. 2021.

PYTORCH. **Tensors and Dynamic neural networks in Python with strong GPU acceleration**. http://pytorch.org/. Acesso em: 25 mar. 2021.

RAINA, Rajat; MADHAVAN, Anand; NG, Andrew Y. **Large-scale deep unsupervised learning using graphics processors**. In: 26TH ANNUAL INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2009, Montreal. Proceedings, Montreal, p. 873–880, 2009.

RASCHKA, Sebastian; MIRJALILI, Vahid. **Python Machine Learning**. 2 ed. Birmingham: Packt, 2017.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning representations by back-propagating errors**. Nature, v. 323, p. 533–536,1986.

SANH, Victor; DEBUT, Lysandre; CHAUMOND, Julien; WOLF, Thomas. **DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter**, 2019. Disponível em: https://arxiv.org/abs/1910.01108. Acesso em: 7 set. 2021.

SCHMIDHUBER, Juergen. **Deep Learning in Neural Networks**: An Overview. Technical Report IDSIA-03-14, 2014.

SHERMIS, Mark D. **State-of-the-art automated essay scoring: Competition, results and future directions from a United States demonstration**. Assessing Writing, v. 20, p. 53-76, 2014.

SHERMIS, Mark D.; BURSTEIN, Jill; BURSKY, Sharon A. **Introduction to Automated Essay Evaluation**. In: M. D. Shermis & J. Burstein (eds.), Handbook of Automated Essay Evaluation, London: Routledge, 2013.

STEINKRAUS, D.; BUCK, I.; SIMARD, P. T. **Using GPUs for Machine Learning Algorithms**. EIGHT INTERNATIONAL CONFERENCE ON DOCUMENT ANALYSIS AND RECOGNITION (ICDAR'05), 2005, Seoul. Proceedings, 2005, Seoul, p. 958-963.

TENSORFLOW. **TensorFlow**. https://www.tensorflow.org/. Acesso em: 19 mar. 2021.

VASWANI, Ashish et al., **Attention Is All You Need**. Proc. 31st. Conf. Neural Information Processing Systems (NeurIPS 2017), Long Beach, dez. 2017.

WANG, Bin; WANG, Angela; CHEN, Fenxiao; WANG, Yuncheng; KUO, C.-C. Jay. **Evaluating word embedding models: methods and experimental results**. SIP, v. 8, e19, p. 1-1414, 2019. https://arxiv.org/pdf/1901.09785v1.pdf. Acesso em: 31 mai. 2021.

Contatos: danitrod@protonmail.com e orlandobc@gmail.com