Análise da influência dos hiperparâmetros no treinamento de *word embeddings* para a língua portuguesa

P. A. Lelis*, **F. I. Fazanaro***, **J. R. Sato***
*Universidade Federal do ABC (UFABC), Santo André, SP, Brazil
e-mail: pedro.lelis@ufabc.edu.br

Abstract - *This work analyses the use of the TensorFlow framework and the word2vec algorithms in order to train word embeddings. Wikipedia pages are used as corpus and training is performed employing the Google Cloud Platform. Future works will use these embeddings as input for deep neural models to perform extrinsic tasks such as Entity Recognition and Sentiment Analysis on on-line reviews. The results of training word2vec with 10 epochs show accuracy of 47.9% at analogical reasoning task.*
Keywords: *Natural Language Processing, Word Embeddings, word2vec.*

Introdução

Prever o comportamento de compra do cliente tornou-se muito importante hoje em dia. No entanto, muitos filtros de recomendação dependem basicamente de dados objetivos, como histórico de compras e dados demográficos dos clientes. Embora esses dados objetivos sejam confiáveis e fáceis de medir, eles não representam efetivamente o sentimento do consumidor sobre uma compra específica. Segundo os resultados de Abbasi [1], analisar os dados perceptivos, como o sentimento dos clientes em avaliações *on-line*, resulta em melhor precisão e recall para a predição do comportamento, se comparado à abordagens que analisam apenas dados objetivos. E o melhor resultado para predição de comportamento é alcançado quando ambas abordagens são combinadas, isto é, análise de dados objetivos e perceptivos.

Porém, a extração dos sentimentos dos clientes a partir das revisões *on-line* não pode ser considerada uma tarefa trivial como a coleta de dados objetivos. Para se realizar essa tarefa, o procedimento no estado da arte usa conceitos de *deep learning* para processar a linguagem natural. Irsoy e Cardie [2] usaram a *Bidirectional Recurrent Neural Network* (BRNN), a qual incorpora simultaneamente informações tanto sobre a ordem normal bem como sobre a ordem inversa com que as palavras aparecem na sentença, a fim de analisar a hierarquia temporal das palavras na tarefa de extração de opinião. O problema da análise de sentimentos foi abordado por Socher [3] com a *Recursive Neural Tensor Network* (RNTN), a qual retrata a natureza recursiva das linguagens naturais. Nesta abordagem, a RNTN foi treinada no *Stanford Sentiment Treebank* e alcançou o estado da arte em classificação positiva / negativa de uma única frase e na predição de rótulos de sentimento *fine-grained* para todas as frases. Kim [4] aplicou a *Convolutional Neural Network* (CNN) para tarefas de classificação em nível de sentença e implementou o ajuste fino com o objetivo da CNN aprender vetores específicos de diferentes tarefas de Processamento de Linguagem Natural (NLP), a fim de alcançar novos ganhos no desempenho.

Todos os trabalhos citados acima utilizaram vetores densos de D dimensões para representar cada palavra de seus respectivos *corpus*. Estes vetores são conhecidos como *embedding* e são capazes de mapear o significado sintático e semântico das palavras. O presente trabalho tem o objetivo de analisar o uso do *framework* TensorFlow e a influência de diferentes hiperparâmetros do algoritmo word2vec durante o treinamento de tais *embeddings* para as palavras da língua Portuguesa. Futuramente, esses *embeddings* irão "alimentar" uma arquitetura de rede neural profunda, como a *Long Short-Term Memory* (LSTM). Este tipo de arquitetura de rede neural profunda permite reconhecer as entidades (produtos, serviços, negócios, etc.) sobre as quais uma dada revisão *on-line* está se referindo e direcionar cada um dos diferentes sentimentos presentes no texto da revisão com a respectiva entidade, a qual ele se refere [2].

Word Embeddings

Tradicionalmente, o *one-hot vector* é utilizado para representar palavras em um dicionário através de vetores. Este método representa as palavras através de vetores com N dimensões, onde N é igual ao tamanho do dicionário. Nesta abordagem, todos os elementos do vetor são definidos iguais a 0, exceto o i-ésimo elemento, o qual é definido com valor igual a 1 e representa a i-ésima palavra no dicionário. Desta forma, a matriz que representa todas as palavras tem $N \times N$

dimensões, é esparsa e incapaz de representar as relações que existem entre as palavras.

Métodos como word2vec representam palavras através de vetores com dimensões geralmente entre 25 e 1000. Seus vetores são densos e são capazes de representar relações sintáticas e semânticas entre as palavras. Esse método assume que as palavras, as quais normalmente aparecem próximas em um dado contexto, devem possuir vetores próximos.

Skip-gram

Skip-gram é uma das maneiras de se implementar o word2vec, a qual prevê a probabilidade de palavras aparecerem no contexto dado uma palavra-alvo. Neste modelo, cada palavra é inicialmente representada por um *one-hot vector* com dimensão N igual ao tamanho do dicionário. A palavra-alvo é representada pelo *one-hot vector* w_t. O *embedding* da palavra-alvo (ou vetor de entrada), v_t, é pesquisado em $\mathbf{V}$ a partir do w_t, onde $\mathbf{V}$ é a matriz de peso de entrada. $\mathbf{V}$ possui dimensão $N \times D$, onde D é o valor do hiperparâmetro referente ao tamanho do *embedding*. Um vetor de pontuação com dimensão N é gerado através do produto interno entre a transposta de $\mathbf{U}$ e v_t, onde $\mathbf{U}$ é a matriz de peso de saída com dimensão $D \times N$. Finalmente, o vetor de pontuação passa por uma função Softmax a fim de gerar um vetor de probabilidades, $\hat{\mathrm{y}}$, com dimensão igual a N. As probabilidades de se observar cada palavra do contexto em um tamanho de janela de contexto m são dada por $\hat{\mathrm{y}}_{t\text{-}m}$, ..., $\hat{\mathrm{y}}_{t\text{-}1}$, $\hat{\mathrm{y}}_{t+1}$, ..., $\hat{\mathrm{y}}_{t+m}$. O modelo Skip-gram possui como objetivo maximizar a probabilidade média do log [5]:

$$J_{SG} = \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq +m, \\ j \neq 0}} \log p(w_{t+j}|w_t) \qquad (1)$$

$$p(w_{t+j}|w_t) = \frac{\exp(u_{w_{t+j}}^{\top} v_{w_t})}{\sum_{w=1}^{W} \exp(u_w^{\top} v_{w_t})} \qquad (2)$$

onde $\mathrm{p}(\mathrm{w}_{t+j}|\mathrm{w}_t)$ representa a probabilidade da palavra de contexto w_{t+j} aparecer, dado que a palavra-alvo w_t apareceu; $\mathrm{u}_{w_{t+j}}$ representa o *embedding* da palavra de contexto w_{t+j}; v_{w_t} representa o *embedding* da palavra-alvo w_t e u_w representa o *embedding* de cada palavra w do vocabulário.

No entanto, a função Softmax precisa calcular em cada iteração o produto interno sobre todas as palavras do dicionário, o que é computacionalmente custoso e não escalável. Como alternativa, uma estimativa das probabilidades pode ser dada pela função de custo *Noise Contrastive Estimation* (NCE) ou pela função de custo Amostragem Negativa. Essas funções amostram k palavras no dicionário, as quais necessariamente não estão dentro da janela de contexto, e usam o produto interno entre elas e a palavra-alvo w_t para substituir a normalização realizada na função Softmax. Desta forma, a probabilidade de uma palavra do contexto w_{t+j} co-ocorrer dentro da janela de contexto da palavra-alvo w_t é aproximada por [5]:

$$\log \sigma(u_{w_{t+j}}^{\top} v_{w_t}) + \sum_{t=1}^{k} \log \sigma(-u_k^{\top} v_{w_t}) \qquad (3)$$

onde σ representa a função sigmoide e u_k representa o *embedding* de cada palavra k amostrada do vocabulário.

Durante o treinamento da rede neural, o *Back-propagation* utiliza o Método do Gradiente para atualizar as matrizes de peso de entrada $\mathbf{V}$ e de saída $\mathbf{U}$. De fato, o Método do Gradiente *Mini-Batch* (MB-GD) ou o Método do Gradiente Estocástico (SGD) são usados para otimizar os parâmetros da rede neural, uma vez que estes são mais eficientes do que sua versão original [5]. Os vetores resultantes do treinamento realizado pelo Skip-gram são utilizados como parâmetros iniciais em arquiteturas de rede neural profunda, como a LSTM.

Metodologia

O banco de dados com todas as páginas da Wikipédia em Português até o dia 20 de Agosto de 2017 foi baixado da Wikimedia [6], resultando em um arquivo compactado de 1,4 GB. Este arquivo foi descompactado e filtrado utilizando o *script* do WikiExtractor [7], a fim de gerar o *corpus* que foi utilizado para treinar o word2vec. O *script* do WikiExtractor descarta anotações presentes nas páginas da Wikipédia como *hyperlinks*, imagens, tabelas, referências e listas; resultando em um arquivo com apenas o texto simples das páginas.

Na sequência, o *corpus* passou por dois pré-processamentos. O primeiro pré-processamento adiciona um caractere de espaço antes e depois de cada caractere de pontuação. Por exemplo, "Será que vai chover?" torna-se "Será que vai chover ? ". O segundo pré-processamento escreve por extenso o nome dos dígitos de todos os números presentes no *corpus*, sempre com um caractere espacial entre

o nome de cada dígito. Por exemplo, "10" se torna "um zero".

O word2vec considera cada *string* separada por um caractere de espaço como um *token*. Desta forma, o primeiro pré-processamento tem como objetivo representar cada caractere de pontuação como um *token* independente. No caso deste pré-processamento não ter sido realizado, cada palavra seguida por um caractere de pontuação seria considerada pelo word2vec como um *token* independente. Por exemplo, "chover" é um *token* diferente de "chover?", o qual por sua vez é um *token* diferente de "chover," e assim por diante.

O segundo pré-processamento tem como objetivo representar todos os números possíveis como a combinação de apenas dez *tokens* diferentes, ou seja, os nomes por extenso dos dez dígitos. No caso deste pré-processamento não ter sido realizado, cada número diferente seria considerado um *token* diferente.

Após estes pré-processamentos, o *corpus* resultou em um documento com cerca de 300 milhões de palavras. Apenas palavras que se repetiram mais de 5 vezes foram adicionadas ao dicionário. As demais palavras foram substituídas pelo *token* "UNK", resultando em um dicionário de aproximadamente 500 mil palavras.

Para a construção do algoritmo word2vec, foi utilizado o TensorFlow 1.3, um *framework* de código aberto criado pelo Google para computação numérica. TensorFlow utiliza grafos de fluxo de dados, nos quais os nós representam operações matemáticas e as arestas representam matrizes de dados multidimensionais (tensores) que fluem entre os nós [8]. Foi simulada a arquitetura Skip-Gram do word2vec e utilizado o Método do Gradiente Estocástico para otimizar os parâmetros com mais eficiência. O custo foi avaliado com a função Amostragem Negativa e foi implementado um decaimento linear da taxa de aprendizagem em relação ao número total de palavras processadas. Foram utilizados os seguintes hiperparâmetros:

A) Taxa de aprendizagem inicial: 0,025;

B) Tamanho da janela: 2, 4, 6, 8 e 10;

C) Tamanho dos *embeddings*: 100, 200, 300 e 400.

A acurácia foi calculada de acordo com a "*analogical reasoning task*" introduzida por Mikolov [5] e traduzida para o português por Hartmann [9]. Os gráficos foram gerados com a acurácia de diferentes hiperparâmetros para entender a influência de cada um, a fim de escolher alguns dos melhores hiperparâmetros para serem utilizados em tarefas extrínsecas, como Reconhecimento de Entidades e Análise de Sentimentos.

Resultados

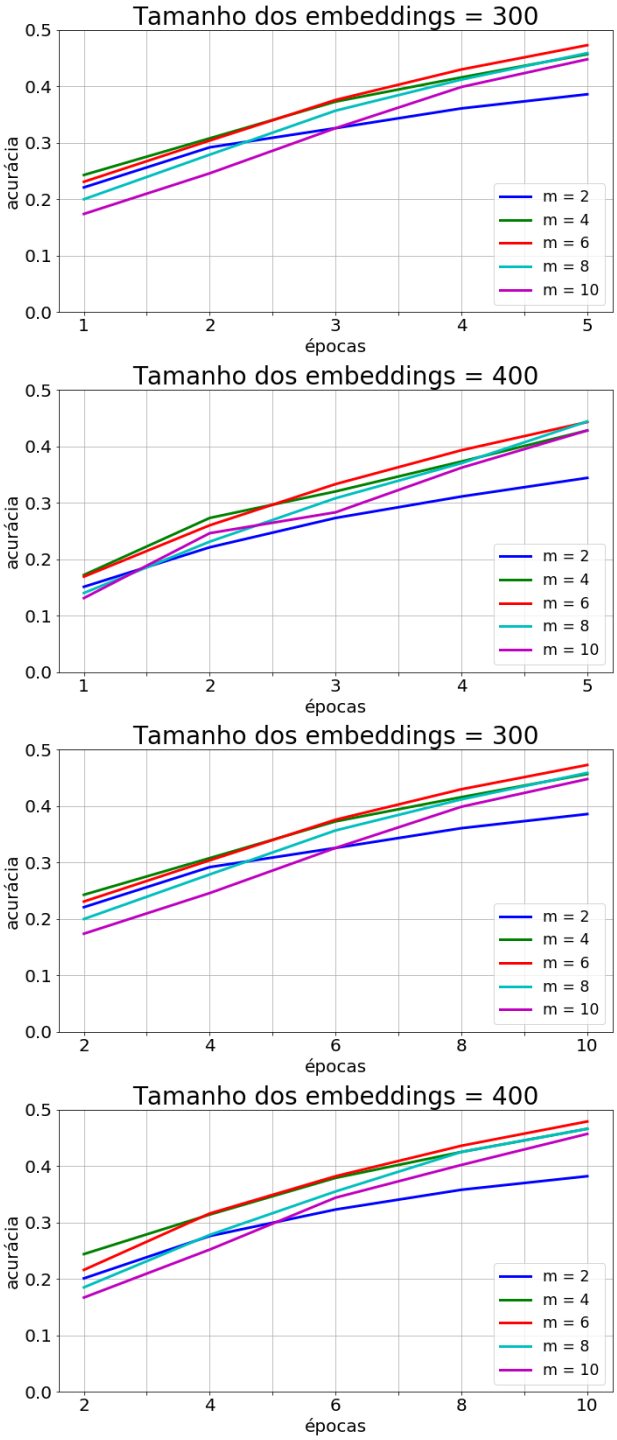

Figura 1 – Acurácia da "*analogical reasoning task*" para diferentes hiperparâmetros.

A Figura 1 mostra os resultados dos treinamentos para, respectivamente, dimensões dos *embeddings* de 300 e 400 e épocas de treinamento de 5

e 10. Em cada um dos casos, variou-se o tamanho da janela de contexto entre 2, 4, 6, 8 e 10. Cada época significa que o algoritmo viu todo o conjunto de treinamento, ou seja, passou por todas as palavras do *corpus*.

É possível notar que a acurácia cresce de forma aproximadamente linear conforme o número de épocas aumenta. Exceto para o tamanho da janela de contexto igual a 2, no qual a acurácia começa a se estabilizar em torno de 35%.

Analisando os resultados obtidos para o mesmo tamanho dos *embeddings* e variando-se as épocas treinadas entre 5 e 10, mas nota-se um ganho de acurácia maior para *embeddings* de tamanho 400. Em outras palavras, o word2vec está aprendendo os melhores parâmetros aproximadamente duas vezes mais lento para épocas igual à 10, se comparado com o resultado para épocas igual à 5.

Finalmente, nota-se que os melhores resultados são obtidos para dimensão igual a 400 e para tamanho de janela igual a 6.

Conclusões

Analisando os resultados obtidos para épocas iguais a 5 e a 10, nota-se que não houve diferenças significativas na acurácia. Esse resultado é devido a taxa de aprendizagem estar muito alta. Portanto a taxa de aprendizagem está limitando o desempenho do word2vec. Como consequência, a acurácia melhora proporcionalmente ao decaimento da taxa de aprendizagem e não proporcionalmente ao número de épocas treinado.

Quanto à estabilização nas curvas de acurácia para o tamanho de janela de contexto igual a 2, pode-se concluir que esta estabilização significa que o aprendizado está saturado para esse tamanho de janela e não é viável aumentar a dimensão dos *embeddings* para conseguir maior acurácia. Neste caso, é mais eficiente aumentar o tamanho da janela de contexto. Espera-se, ao utilizar uma taxa de aprendizagem menor, que a acurácia para todos os tamanhos de janela comecem a saturar após determinado número de épocas treinado.

Os próximos passos serão treinar o word2vec com taxas de aprendizagem menor e com decaimento da taxa de aprendizagem exponencial, a fim de avaliar quais hiperparâmetros produzem *embeddings* que desempenham melhor no "*analogical reasoning task*". Estes *embeddings* serão então utilizados em tarefas extrínsecas como Reconhecimento de Entidades e Análise de Sentimentos.

Agradecimentos

Os autores agradecem à UFABC, à CAPES e ao CNPq (processo 449699/2014-5) pelo apoio financeiro.

Referências

[1] A. Abbasi, R. Y. K. Lau, and D. E. Brown, "Predicting Behavior," *IEEE Intelligent Systems*, 2015.

[2] O. Irsoy and C. Cardie, "Opinion Mining with Deep Recurrent Neural Networks," *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[3] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[4] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[6] Wikimedia, "Wikimedia downloads," 2017.

[7] G. Attardi, "Wikipedia extractor," 2016.

[8] TensorFlow, "An open-source software library for machine intelligence," 2017.

[9] N. S. Hartmann, E. Fonseca, C. D. Shulby, M. V. Treviso, J. S. Rodrigues, and S. M. Aluísio, "Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks," *Symposium in Information and Human Language Technology (STIL)*, 2017.