Laura Abreu Schulz Vieira - 2017015061

Aprendizado Profundo em Processamento de Linguagens Naturais
Trabalho Prático 1 - Estudo de modelos de linguagem

Introdução:

Este trabalho tem como finalidade principal avaliar diferentes construções de modelos de linguagens neurais e o impacto da variação de seus hiperparâmetros no desempenho obtido por eles. Para tal análise, utilizamos o modelo word2vec e variamos os parâmetros: size (dimensionalidade do vetor de embedding), window (contexto analisado para predição da palavra) e escolha do algoritmo (CBOW ou Skip-Gram). Treinamos o modelo a partir de um corpus fornecido e avaliamos utilizando o conhecido "questions-words.txt", a partir da métrica distância dos cossenos entre a palavra predita e a palavra correta.

Metodologia[1]:

Apesar do modelo fornecido em C, optei por utilizar a biblioteca Gensim do python, devido à maior facilidade de manipulação e familiaridade com a linguagem. O preprocessamento foi feito também em python, utilizando a biblioteca nltk, onde apenas removi as stop-words.

A escolha do modelo se dá a partir da função Word2Vec, na qual os parâmetros citados na introdução são selecionados. As configurações escolhidas foram as seguintes:

- Foram executadas 24 diferentes configurações do modelo, organizadas da seguinte forma:
- 12 para cada algoritmo (CBOW/Skip-Gram).
- As 12 execuções de cada foram divididas em 3, nas quais varia-se o parâmetro window, sendo cada grupo com a window fixa em 2, 5 ou 10.
- Para cada uma dentro de cada grupo citado acima varia-se o parâmetro size entre 10, 50, 100 e 300.
    - Obs: o parâmetro size modificado define a dimensionalidade do vetor de embedding, e não o tamanho do corpus. O tamanho do corpus foi mantido fixo em seu valor integral - havia feito alterando esse parâmetro e não houve tempo para mudar.

Os demais parâmetros se mantiveram fixos em seu valor default em todas execuções (exceto pelo epochs, em training, que alterei para 5 buscando melhorar o desempenho global).

Após o treino do modelo, partimos para análise dos resultados. Como especificado, o método foi: dado 3 palavras (provenientes do questions-words.txt) deve-se prever a quarta.

Tal avaliação foi feita utilizando-se duas principais funções: most_similarity, que busca a palavra mais próxima, segundo a métrica distância de cossenos, da resultante da operação entre as 3 palavras dadas, e, em seguida, a função distance, que verifica qual a

[1] https://github.com/lauraabreusv/TP1-NLP

distância de cossenos entre a palavra fornecida pelo modelo e a palavra correta. Isso ocorre para cada instância de questions-words.txt, portanto, foi feita uma média das distâncias para cada execução.

Resultados:

Primeiramente, vamos apresentar os resultados em uma tabela:

Algoritmo	**Window**	**Size**	**Distâncias**
CBOW	2	10	0.20476320954203897
CBOW	2	50	0.27634588779700797
CBOW	2	100	0.2869785999488343
CBOW	2	300	0.2903761353368646
CBOW	5	10	0.2617761158514321
CBOW	5	50	0.3098384166523586
CBOW	5	100	0.31875563769285015
CBOW	5	300	0.31545074662678313
CBOW	10	10	0.29427563943360824
CBOW	10	50	0.33511682373405005
CBOW	10	100	0.33751694005064165
CBOW	10	300	0.33709973056905707
SKIP-GRAM	2	10	0.14210703559661236
SKIP-GRAM	2	50	0.22531428161337552
SKIP-GRAM	2	100	0.25914187979330533
SKIP-GRAM	2	300	0.3051277067905244
SKIP-GRAM	5	10	0.1473230571076388
SKIP-GRAM	5	50	0.23242848244026254
SKIP-GRAM	5	100	0.2667765601364307
SKIP-GRAM	5	300	0.33431945179494954
SKIP-GRAM	10	10	0.15630007275827984
SKIP-GRAM	10	50	0.24047563183108447

SKIP-GRAM	10	100	0.2782633101021789
SKIP-GRAM	10	300	0.36488825093233146

A partir dos resultados observados nessa tabela, foram gerados os seguintes gráficos contendo os erros (distâncias) de acordo com cada métrica:

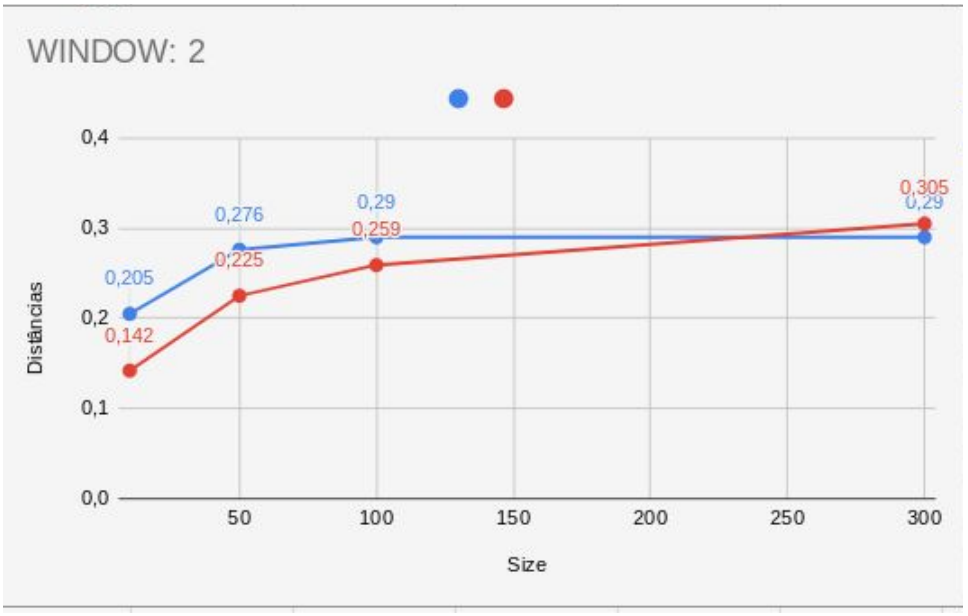

Gráfico 1: CBOW (linha azul) e SKIP-GRAM (linha vermelha) quando o contexto foi setado em 2. Valores de erros computados para os sizes 10, 50, 100 e 300 (demarcados no gráfico)

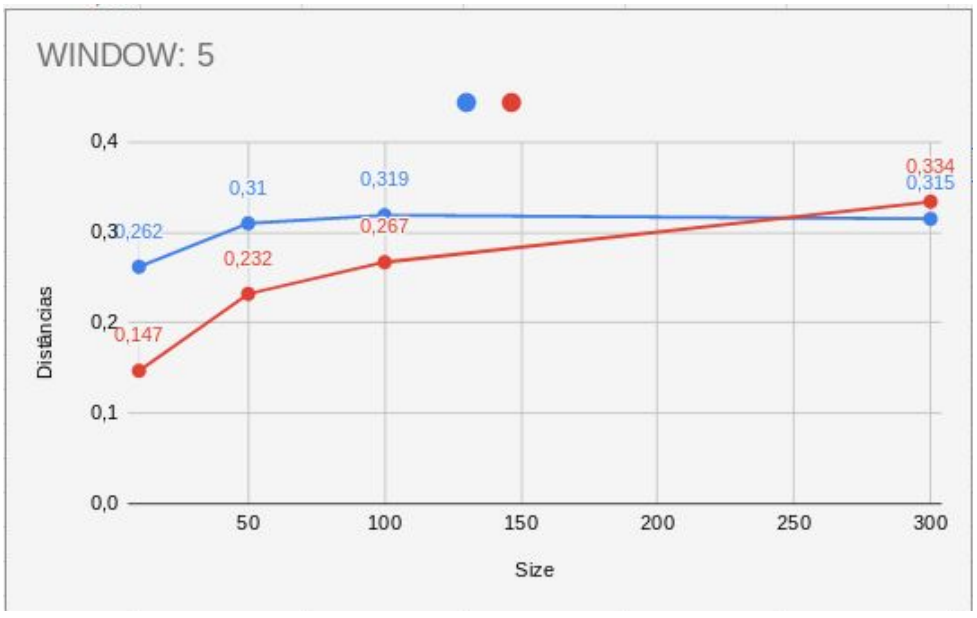

Gráfico 2: CBOW (linha azul) e SKIP-GRAM (linha vermelha) quando o contexto foi setado em 5. Valores de erros computados para os sizes 10, 50, 100 e 300 (demarcados no gráfico)

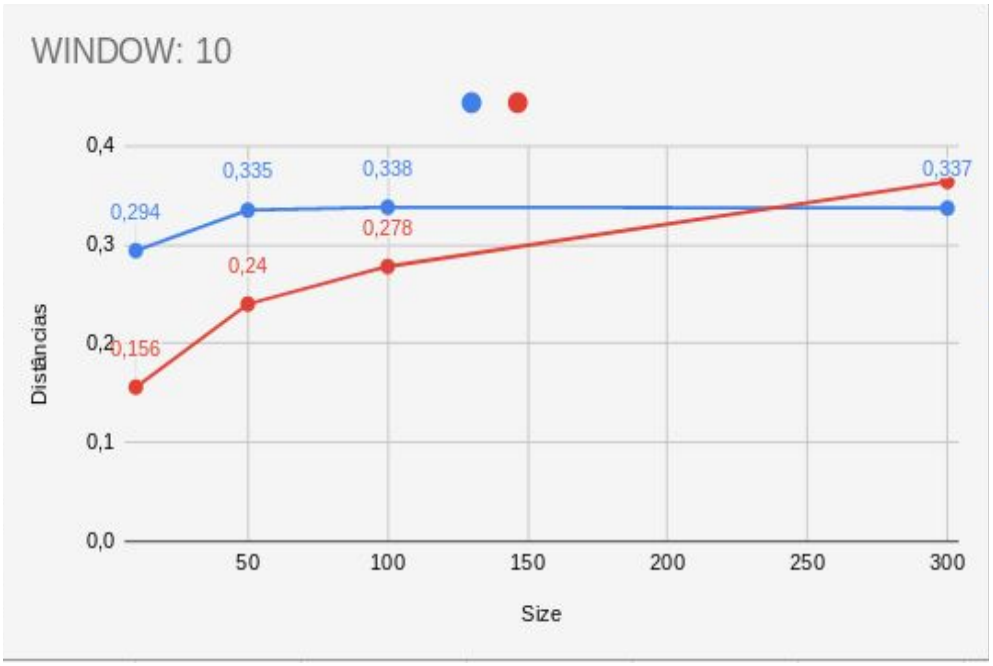

Gráfico 3: CBOW (linha azul) e SKIP-GRAM (linha vermelha) quando o contexto foi setado em 10. Valores de erros computados para os sizes 10, 50, 100 e 300 (demarcados no gráfico). Valor faltante skip-gram (para size = 100): 0.364.

Conclusões:

A partir da análise dos gráficos gerados e da tabela contendo os resultados, é possível observar alguns pontos importantes:

1. Ambos os algoritmos foram piorando seu desempenho à medida em que aumentava-se o contexto e a dimensionalidade do vetor, o que não é esperado, pois acreditava-se que aumentando até certo ponto tais parâmetros o desempenho tenderia a aumentar também. Uma possível explicação para esse resultado é o *overfitting*, que é quando acredita-se, pelo treino, que o aprendizado foi bom e não precisa ser melhorado, mas quando se altera o conjunto de dados (para teste) é verificado um desempenho baixo.
2. O algoritmo do skip-gram, como costumam apontar as pesquisas, teve um desempenho superior para as configurações que se apresentaram melhores para ambos (menor contexto e dimensionalidade). Entretanto, se mostra pior que o CBOW para as configurações que foram inferiores para ambos. No CBOW, observa-se que do size 100 para o 300 já não há tanta diferença de performance.
3. A performance do algoritmo não pode ser prevista analisando apenas um único parâmetro, e sim um conjunto de vários, além de outros fatores não analisados neste trabalho. Portanto, para escolher um modelo devem ser analisadas diversas variáveis.

Referências:

Word2Vec, Wikipedia https://en.wikipedia.org/wiki/Word2vec#CBOW_and_skip_grams
Doc Gensim, radimrehurek https://radimrehurek.com/gensim/models/word2vec.html
Tutorial Gensim, Towards Data Science
https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92
Meu código: https://github.com/lauraabreusv/TP1-NLP