

Eliane Martins de Aguiar

Aplicação do *Word2vec* e do Gradiente Descendente Estocástico em Tradução Automática

Rio de Janeiro

Maio 2016

Eliane Martins de Aguiar

Aplicação do *Word2vec* e do Gradiente Descendente Estocástico em Tradução Automática

Fundação Getulio Vargas -FGV

Escola de Matemática Aplicada -EMAp

Programa de Pós-Graduação

Orientador: Renato Rocha Souza

Coorientador: Flávio Codeço Coelho

Rio de Janeiro

Maio 2016

Aguiar, Eliane Martins de

Aplicação do Word2vec e do Gradiente descendente estocástico
em tradução automática / Eliane Martins de Aguiar. - 2016.
78 f.

Dissertação (mestrado) – Fundação Getulio Vargas, Escola de Matemática
Aplicada.

Orientador: Renato Rocha Souza.
Coorientador: Flávio Codeço Coelho
Inclui bibliografia.

1. Redes neurais (Computação). 2. Processamento da linguagem natural
(Computação). I. Souza, Renato Rocha. II. Coelho, Flávio Codeço. III. Fundação
Getulio Vargas. Escola de Matemática Aplicada. IV. Título.

CDD – 006.3



ELIANE MARTINS DE AGUIAR

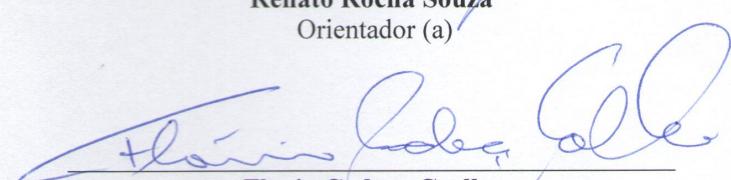
**“APLICAÇÃO DO WORD2VEC E DO GRADIENTE DESCENDENTE ESTOCÁSTICO
EM TRADUÇÃO AUTOMÁTICA.”**

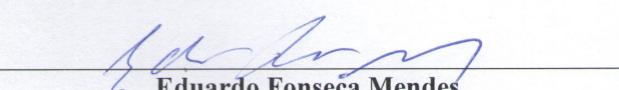
Dissertação apresentada ao Curso de Mestrado em Modelagem Matemática da Informação da Escola de Matemática Aplicada da Fundação Getulio Vargas para obtenção do grau de Mestre em Modelagem Matemática da Informação.

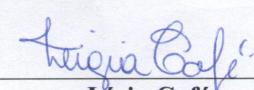
Data da defesa: 30/05/2016.

ASSINATURA DOS MEMBROS DA BANCA EXAMINADORA


Renato Rocha Souza
Orientador (a)


Flávio Codeço Coelho
Co-Orientador (a)


Eduardo Fonseca Mendes


Ligia Café

À minha mãe, Neide.

Agradecimentos

Ao meu orientador Renato Rocha por todo apoio e por ter acreditado no meu potencial.

Ao meu coorientador Flávio Codeço por ter me ajudado diversas vezes e por ter me sugerido o tema deste trabalho.

À CAPES por ter acreditado em mim e investido nos meus estudos.

Aos professores da banca pelo carinho e pela atenção.

Aos professores e funcionários da EMAp por toda atenção e suporte ao longo desta jornada.

Aos meus colegas de curso por terem me feito sentir como realmente parte da turma. Em especial ao Leon Diniz e ao Luis Eduardo Craizer por sempre terem sentado ao meu lado e feito os trabalhos de grupo comigo.

À minha mãe por todo seu apoio emocional, financeiro e moral. Sem você este mestrado nunca teria sido possível.

Aos meus parentes e amigos pelo apoio, por terem acreditado em mim, e por serem compreensivos com meus sumiços.

Ao Felipe Bottega Diniz pelo apoio, paciência, compreensão, aulas e toda ajuda durante todo o mestrado. Você foi fundamental para eu manter minha sanidade.

E também às pessoas que ajudaram revisando minha dissertação ou dando ideias.

'Wir müssen wissen. Wir werden wissen.' - 'Nós devemos saber. Nós saberemos'
(David Hilbert)

Resumo

O *word2vec* é um sistema baseado em redes neurais que processa textos e representa palavras como vetores, utilizando uma representação distribuída. Uma propriedade notável são as relações semânticas encontradas nos modelos gerados. Este trabalho tem como objetivo treinar dois modelos utilizando o *word2vec*, um para o Português e outro para o Inglês, e utilizar o gradiente descendente estocástico para encontrar uma matriz de tradução entre esses dois espaços.

Palavras-chaves: Processamento de Linguagem Natural, Redes Neurais, Word2vec, Continuous Bag-of-Words, Gradiente Descendente Estocástico, Tradução Automática.

Abstract

Word2vec is a neural network that processes texts and represents words as vectors, using a distributed representation. An important feature are the semantics relationships found in the models generated by it. The aim of this work is to train two models using Word2vec, one for Portuguese and the other for English, and use the stochastic gradient descent to find a translation matrix between those spaces.

Key-words: Natural Language Processing, Neural Networks, Word2vec, Continuous Bag-of-Words, Stochastic Gradient Descent, Machine Translation

Listas de ilustrações

Figura 1 – Exemplo de arquitetura de rede neural com uma camada oculta	4
Figura 2 – Representação da arquitetura do <i>CBOW</i> e <i>Skip-Gram</i> - Figura retirada de um artigo do Mikolov [13]	6
Figura 3 – Exemplo com animais usando <i>PCA</i>	27
Figura 4 – Exemplo com números usando <i>PCA</i>	27
Figura 5 – Exemplo com gêneros usando <i>PCA</i>	28
Figura 6 – Exemplo com opositos usando <i>PCA</i>	28
Figura 7 – Erros de W_1	31
Figura 8 – Erros de W_2	32
Figura 9 – Erros de W_3	32
Figura 10 – Erros de W_4	33
Figura 11 – Erros de W_5	33
Figura 12 – Erros de W_6	34
Figura 13 – Posição nos ranks de W_1	36
Figura 14 – Posição nos ranks de W_2	37
Figura 15 – Posição nos ranks de W_3	37
Figura 16 – Posição nos ranks de W_4	38
Figura 17 – Posição nos ranks de W_5	38
Figura 18 – Posição nos ranks de W_6	39
Figura 19 – Similaridade entre pivôs e alvos - W_1	40
Figura 20 – Similaridade entre pivôs e alvos - W_2	40
Figura 21 – Similaridade entre pivôs e alvos - W_3	40
Figura 22 – Similaridade entre pivôs e alvos - W_4	41
Figura 23 – Similaridade entre pivôs e alvos - W_5	41
Figura 24 – Similaridade entre pivôs e alvos - W_6	41
Figura 25 – Similaridade relativa - W_1	42
Figura 26 – Similaridade relativa - W_2	42
Figura 27 – Similaridade relativa - W_3	43
Figura 28 – Similaridade relativa - W_4	43
Figura 29 – Similaridade relativa - W_5	43
Figura 30 – Similaridade relativa - W_6	44
Figura 31 – Rank invertido - W_1	45
Figura 32 – Rank invertido - W_2	45
Figura 33 – Rank invertido - W_3	45
Figura 34 – Rank invertido - W_4	46
Figura 35 – Rank invertido - W_5	46

Figura 36 – Rank invertido - W_6	46
Figura 37 – Histograma das distâncias de palavras, usando o método <i>edit distance</i> , entre pivôs gerados por cada uma das 6 matrizes W e as palavras alvos correspondentes.	47
Figura 38 – Histograma das distâncias de palavras, usando o método <i>edit distance</i> , entre palavras geradas utilizando o primeiro método de Dinu e as palavras alvos correspondentes.	48
Figura 39 – Histograma das distâncias de palavras, usando o método <i>edit distance</i> , entre palavras geradas utilizando o segundo método de Dinu e as palavras alvos correspondentes.	48
Figura 40 – Norma euclidiana entre os pivôs gerados com W_1 e os vetores alvos.	49
Figura 41 – Norma euclidiana entre os vetores gerados com as palavras obtidas no teste 1 de Dinu com W_1 e os vetores alvos.	49
Figura 42 – Norma euclidiana entre os vetores gerados com as palavras obtidas no teste 2 de Dinu com W_1 e os vetores alvos.	50

Lista de tabelas

Tabela 1 – Frequências observadas e frequências esperadas	7
Tabela 2 – Tabela de coocorrências de w_1 e w_2	10
Tabela 3 – Dados e dados ajustados com a média	12
Tabela 4 – Exemplos de bigramas extraídos no Português	19
Tabela 5 – Exemplos de trigramas extraídos no Português	19
Tabela 6 – Exemplos de quadrigramas extraídos no Português	19
Tabela 7 – Exemplos de bigramas extraídos no Inglês	19
Tabela 8 – Exemplos de trigramas extraídos no Inglês	19
Tabela 9 – Exemplos de quadrigramas extraídos no Inglês	20
Tabela 10 – Exemplos de stopwords retiradas do Inglês	21
Tabela 11 – Exemplos de stopwords retiradas do Português	21
Tabela 12 – Teste de correspondência	29
Tabela 13 – Número de Hubs com K=5	35
Tabela 14 – Número de Hubs com K=10	35
Tabela 15 – Número de Hubs com K=20	35
Tabela 16 – Número de Hubs com K=50	35

Sumário

1	INTRODUÇÃO	1
1.1	Objetivo do trabalho	1
2	REFERENCIAL TEÓRICO	3
2.1	Processamento de linguagem natural	3
2.2	Modelo de linguagem de redes neurais	3
2.2.1	Vetores de palavras e relações semânticas	4
2.2.2	<i>Continuous bag-of-words (CBOW) e Skip-gram</i>	5
2.2.3	<i>Word2vec</i>	5
2.2.4	Tradução automática	6
2.3	Procedimentos estatísticos	7
2.3.1	Métricas	7
2.3.1.1	Raw-freq	8
2.3.1.2	Chi-quadrada	8
2.3.1.3	Jaccard	8
2.3.1.4	Razão de Verossimilhança (<i>Likelihood ratio</i>)	8
2.3.1.5	Mi-like	9
2.3.1.6	PMI	9
2.3.1.7	Poisson-Stirling	10
2.3.1.8	T-Student	10
2.3.2	<i>Principal component analysis (PCA)</i>	11
2.3.3	Gradiente descendente estocástico	13
3	METODOLOGIA	15
3.1	Etapas do experimento	15
3.2	Obtenção dos corpus	16
3.3	Remoção dos docs	16
3.4	Tokenização	16
3.5	Obtenção de <i>n-grams</i>	17
3.6	Criação do dicionário bilíngue	20
3.7	Gerando os modelos com o <i>word2vec</i>	22
3.8	Gradiente descendente estocástico	23
3.9	Hubs	23
4	RESULTADOS	27
4.1	Teste com <i>PCA</i>	27

4.2	Teste de correspondência	28
4.3	Relações entre as palavras	29
4.4	Testes dos alphas	31
4.5	Teste de número de hubs em K vizinhos	34
4.6	Teste da posição da palavra correta nos ranks dos pivôs	36
4.7	Teste de similaridade dos pivôs com as traduções	39
4.8	Tratamento de hubs: Normalizar os vetores de similaridade	42
4.9	Tratamento de hubs: Inverter a ordem da fila	44
4.10	Teste das distâncias	47
4.10.1	<i>Edit distance</i>	47
4.10.2	Norma euclidiana	48
5	CONCLUSÃO	51
5.1	Trabalhos futuros	51
	Referências	55

1 Introdução

O trabalho de tradução de um texto não consiste em traduzir uma palavra de cada vez de uma língua A para uma língua B, mas sim lidar com a morfologia, sintaxe e semântica de duas línguas distintas para fazer uma ligação entre elas. A ideia de ter máquinas que fossem capazes de traduzir automaticamente um texto remota ao século XVII [5], ideia a qual começou a se tornar realidade no século XX. Hoje em dia temos tradutores automáticos, mas eles ainda não conseguem fazer um trabalho tão bom quanto um tradutor profissional.

Os sistemas de tradução automática, geralmente, são desenvolvidos para usar informações apenas a nível de frase, ignorando o documento como um todo. Trabalhos recentes alegam que o contexto ajuda a traduzir palavras de maneira mais coerente [8], seguindo a hipótese distributiva de Harris: o significado das palavras é evidenciado pelo contexto em que ocorre.[4]

Esses sistemas, normalmente, usam modelos *n-gram* para representar uma palavra na língua alvo. No entanto, há outras formas de representar palavras, tais como vetores, por exemplo. A ideia de representação distribuída não é recente [14], e há modelos que a utilizam, como o *word2vec*. Modelos como o *word2vec* se mostraram robustos e poderosos em prever relações semânticas entre palavras, mesmo em diferentes línguas. Porém eles não lidam muito bem com os diferentes significados de palavras polissêmicas em uma única representação.

1.1 Objetivo do trabalho

O objetivo deste trabalho é realizar um experimento baseado no trabalho de Mikolov [13], e avaliar seus resultados. O experimento original propõe um modelo que automatiza o processo de gerar e estender dicionários e tabelas de frases, utilizando a arquitetura do *word2vec*.

No capítulo 2 apresentamos o referencial teórico, no capítulo 3 é explicada a metodologia utilizada, no capítulo 4 temos os resultados do experimento e no capítulo 5 concluimos e apresentamos sugestões de trabalhos futuros.

2 Referencial Teórico

Neste capítulo são apresentadas as noções teóricas estudadas para a compreensão e realização deste trabalho. Algumas das áreas abrangidas são: Processamento de Linguagem Natural, Redes Neurais, Estatística e Álgebra Linear.

2.1 Processamento de linguagem natural

O processamento de linguagem natural (*natural language processing*), conhecido também pela sigla PLN, é o conjunto de técnicas computacionais que lidam com problemas ligados à linguagem humana, tanto escrita como falada. Seus principais desafios englobam o entendimento e extração de significado da linguagem. O PLN tem importantes aplicações como extração de informação, correção ortográfica, recuperação de informação, reconhecimento de voz, tradução automática dentre outras.

Nas últimas décadas o avanço da tecnologia tornou possível implementar ideias que antes eram apenas vislumbradas. Os trabalhos relacionados ao PLN surgiram no final da década de 1940, com o foco em tradução automática (*machine translation*). Uma das primeiras pesquisas no campo, foi um trabalho de tradução automática do Russo para o Inglês, em um experimento rudimentar e limitado, o qual foi exibido na demonstração *IBM-Georgetown* em 1954.

Outro evento muito importante dessa época foi o *Teddington International Conference on Machine Translation of Languages and Applied Language Analysis* em 1961, onde trabalhos de vários países foram apresentados. Esse período foi de entusiasmo e otimismo. Os que tentaram resolver tarefas de PLN encontraram problemas com o processamento semântico e sintático e com variedades linguísticas.[6]

Essas questões ainda representam um grande desafio no campo de tradução automática. A tradução automática tem aparecido em trabalhos recentes [8][13] utilizando redes neurais. Este trabalho reproduziu um experimento similar ao de Mikolov [13], com tradução do Português para o Inglês.

2.2 Modelo de linguagem de redes neurais

Os modelos de linguagem de redes neurais (*neural network language model*) estão hoje em dia entre as técnicas mais bem-sucedidas para modelagem estatística de linguagem (*statistical language modeling*). Um modelo estatístico de linguagem é uma distribuição de probabilidade sobre *strings* (sequência de caracteres), que tenta refletir o quanto frequente

uma *string* ocorre como uma frase.

Esses modelos foram inspirados no funcionamento do cérebro, na capacidade de reconhecer padrões e aprender com erros e acertos. Os modelos utilizam neurônios artificiais, os quais são uma abstração bem simplificada de um neurônio biológico.

Uma rede neural pode ser vista como um sistema de neurônios, interconectados que processam informações em resposta a entradas externas. Geralmente os modelos de redes neurais são organizados em três camadas: uma camada como input, uma camada oculta e uma camada como output. Na figura abaixo temos um exemplo¹ da estrutura de uma rede neural, nela cada nó representa um neurônio artificial e as setas representam a conexão do output de um neurônio com o input de outro.

A entrada de cada neurônio é um somatório ponderado, e esse valor é usado como entrada de uma função de ativação. As redes neurais precisam de muitas informações para treino e também precisam de informações para teste.

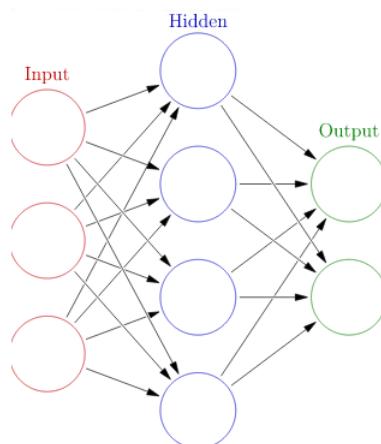


Figura 1 – Exemplo de arquitetura de rede neural com uma camada oculta

2.2.1 Vetores de palavras e relações semânticas

Em modelos de redes neurais podemos representar palavras como vetores, e há duas formas principais de se fazer isso: pela representação explícita e pela representação distribuída. No primeiro caso cada palavra é associada a um vetor que representa os contextos nos quais a palavra ocorre [9], e no segundo caso os vetores são obtidos em um modelo de rede neural como o *continuous bag-of-words* e o *skip-gram*.

Mikolov [12] mostrou que é possível averiguar relações de semelhanças entre os vetores de palavras, gerados utilizando uma rede neural recursiva, com operações lineares

¹ Imagem do artigo https://en.wikipedia.org/wiki/Artificial_neural_network

simples. Também é possível verificar similaridades entre pares de palavras, o exemplo mais conhecido é: ‘King’ – ‘Men’ + ‘Women’ = ‘Queen’. A igualdade aqui encontrada não significa que a operação dê exatamente igual ao vetor ‘Queen’, mas sim que a palavra ‘Queen’ é a mais próxima do vetor encontrado. Essa similaridade corresponde a dizer que *men* está para *king* da mesma forma que *woman* está para *queen*. Essas similaridades também são chamadas de regularidades linguísticas e não se restringem somente ao Inglês. Elas são capazes de capturar diferentes tipos de relações, segue alguns exemplos encontrados por Lopes[2]:

1. Rei + Mulher - Homem = Rainha
2. Areia + Montanha - Praia = Neve
3. Inglaterra + Paris - França = Londres
4. Maior + Pequeno - Grande = Menor
5. Vencer + Perdeu - Venceu = Perder

2.2.2 Continuous bag-of-words (CBOW) e Skip-gram

No modelo *CBOW* o input é um contexto e o output uma palavra omitida, para isso se combina as representações das palavras do em torno para prever a palavra do meio. É considerado como contexto as palavras que estão ao redor da palavra alvo w . Um exemplo é usar como contexto as duas palavras anteriores a w e as duas palavras posteriores. No caso de uma frase $w_1 w_2 w_3 w_4 w_5$ o contexto da palavra w_3 é $w_1 w_2 w_4 w_5$.

No modelo *skip-gram* o objetivo é: ao ser dada uma palavra, poder prever o seu contexto. Nesse caso, o input, é uma palavra que passa por uma camada oculta e tem como output o contexto mais provável.

Os dois modelos se utilizam de uma rede neural com uma camada oculta para gerar o output, e do algoritmo de *back propagation* para atualizar, ou apenas pesar, os valores dos parâmetros conforme observam novos para exemplos.

2.2.3 Word2vec

O *word2vec* é um ferramenta open-source, feita por Mikolov et al. [11], que é utilizada para calcular representações de palavras como vetores. Ele oferece duas arquiteturas como opção: *continuous bag-of-words* (CBOW) e *skip-gram*. O *word2vec* funciona tomando um corpus como input e devolvendo vetores de palavras como output. No processo é construído um vocabulário, e ocorre o aprendizado das representações do vocabulário como vetores. Pode-se escolher qual das duas arquiteturas será utilizada. Na prática o *skip-gram* oferece uma representação melhor de palavras quando a base de dados é menor. *CBOW* é mais rápido e aplicável em grandes bases de dados [13]. Na figura abaixo temos um modelo simplificado de como funcionam as arquiteturas. No *CBOW* se prevê uma palavra

baseado no contexto, no *skip-gram* se prevê as palavras em torno de uma palavra dada [10].

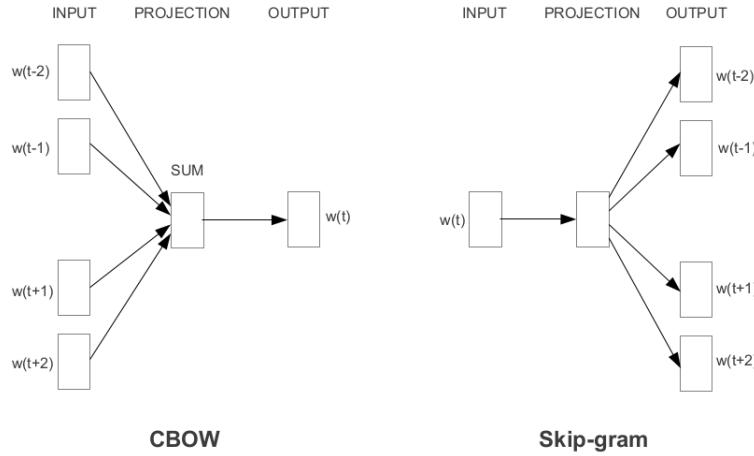


Figura 2 – Representação da arquitetura do *CBOW* e *Skip-Gram* - Figura retirada de um artigo do Mikolov [13]

2.2.4 Tradução automática

Mikolov e Wolf propuseram extensões bilíngues da arquitetura do *word2vec*. A base dos sistemas modernos de tradução automática são dicionários e tabelas de frases. Mikolov [13] gerou e utilizou um dicionário bilíngue para treino e teste, e teve como objetivo resolver o seguinte problema de otimização:

$$\min_W \sum_{i=1}^n \|x_i \cdot W - z_i\|^2,$$

onde W é uma matriz de tradução, que começa com valores aleatórios e será ajustada, $\{x_i, z_i\}_{i=1}^n$ são pares de vetores de palavras associados, onde x_i é a representação distribuída da palavra i na língua fonte e z_i é o vetor representação da sua tradução na língua alvo.

Para construir esse modelo é necessário construir modelos monolíngues usando grande quantidade de textos, depois usa-se um dicionário bilíngue para aprender a projeção entre as línguas. Tratando cada língua como um espaço vetorial pode-se capturar o relacionamento entre eles por um mapeamento. Esse método é baseia na premissa que palavras com os mesmos significados, em línguas distintas, apresentam um arranjo geométrico similar.

trico semelhante. Isso ocorre pelo fato de as línguas compartilharem conceitos semelhantes, como por exemplo que o gato é um animal menor que um cachorro e que alto é o oposto de baixo.

2.3 Procedimentos estatísticos

2.3.1 Métricas

Collocations são as combinações de duas ou mais palavras, e que ocorrem naturalmente dentro de uma língua. Neste trabalho as *collocations* serão tratadas por *n-grams*, sendo n o número de palavras seguidas que, juntas, formam uma unidade de sentido, como ‘São Paulo’ e ‘Rio de Janeiro’. Para se extrair, automaticamente, candidatos a *n-grams* é necessário usar alguma métrica de associação. As métricas de associação são fórmulas matemáticas que interpretam dados que tem uma coocorrência frequente. Para cada par, ou n -upla (se forem n palavras), de palavras, as métricas vão associar uma nota e vão retornar como resultado os candidatos em ordem.

Os modelos estatísticos de coocorrências vão levar em consideração as frequências observadas e as frequências esperadas, exemplificadas na tabela a seguir.

V representa a primeira posição em um bigrama, e U representa a segunda posição. A tabela de frequência observada pode ser vista da seguinte maneira: na primeira linha, à esquerda, teremos quantas vezes o bigrama vu foi observado no texto, e à direita quantas vezes tivemos um bigrama que a primeira palavra não fosse v mas a segunda era u . Na segunda linha temos, à esquerda, quantos bigramas tinham v na primeira posição e uma palavra diferente de u na segunda, e por último temos o caso em que a primeira palavra não é v e a segunda não é u .

Tabela 1 – Frequências observadas e frequências esperadas

	$V = v$	$V \neq v$	Total
$U = u$	O_{11}	O_{12}	$= R_1$
$U \neq u$	O_{21}	O_{22}	$= R_2$
Total	$= C_1$	$= C_2$	$= N$

	$V = v$	$V \neq v$
$U = u$	$E_{11} = \frac{R_1 C_1}{N}$	$E_{12} = \frac{R_1 C_2}{N}$
$U \neq u$	$E_{21} = \frac{R_2 C_1}{N}$	$E_{22} = \frac{R_2 C_2}{N}$

Tanto o experimento como as métricas foram feitos com trigramas, mas as explicações e exemplos de seus funcionamentos serão com bigramas. O experimento consistiu em analisar um dos textos do português. Nesse texto foi aplicado a métrica *raw-freq* igual a 100, explicado logo abaixo, e isso resultou em 1126 trigramas que foram analisados manualmente. Desses trigramas, foram considerados 42 bigramas válidos, 20 trigramas e 16 quadrigramas. Os testes utilizando as métricas abaixo consistiram em analisar quantos

desses trigramas e quadrigramas válidos apareciam, após se usar o filtro de frequência, dentro os 10% mais relevantes encontrados. As métricas estão disponíveis pelo NLTK ²

As métricas testadas foram as seguintes:

2.3.1.1 Raw-freq

Essa é a métrica mais simples, os *n-grams* vão ser classificados pelo número de vezes que aparecem no texto, os que tem uma frequência muito baixa serão desconsiderados. Essa métrica foi utilizada como um primeiro filtro no trabalho, isto é, só foram considerados *n-grams* que apareceram pelo menos 100 vezes no texto.

2.3.1.2 Chi-quadrada

Essa métrica avalia os *n-grams* usando a chi-quadrada de Pearson. Essa técnica não assume que os eventos observados tenham uma distribuição normal de probabilidade. A essência dele é comparar as frequências observadas com as frequências esperadas, se a diferença entre essas duas frequências for grande, temos evidência contra a hipótese nula de independência.

A chi-quadrada é calculada como sendo o quadrado da diferença entre a frequência observada menos a esperada, sobre a frequência esperada.

2.3.1.3 Jaccard

Os *n-grams* são classificados usando o index de Jaccard, também são conhecidos por coeficiente de similaridade de Jaccard. Esse método é usado para comparar a similaridade e diversidade de conjuntos, ele é definido como o tamanho da interseção dividido pelo tamanho da união dos conjuntos.

2.3.1.4 Razão de Verossimilhança (*Likelihood ratio*)

As razões de verossimilhança são mais apropriadas para dados esparsos do que a chi-quadrada de Pearson. As razões de verossimilhança nos diz o quão mais provável é uma hipótese em relação a outra.

Usando o bigrama ' $w_1 w_2$ ' como exemplo: a primeira hipótese (hipótese nula) é de independência, a ocorrência de w_2 é independente de w_1 ter ocorrido, já a segunda hipótese é a formalização da dependência,

Hipótese 1. $P(w_2|w_1) = p = P(w_2|\neg w_1)$,

Hipótese 2. $P(w_2|w_1) = p_1 \neq p_2 = P(w_2|\neg w_1)$.

² <http://www.nltk.org/api/nltk.metrics.html#nltk.metrics.association.NgramAssocMeasures>

Seja c_1 , c_2 e c_{12} o número de ocorrências de w_1 , w_2 e w_1w_2 respectivamente no texto, temos

$$p = \frac{c_2}{N} \quad p_1 = \frac{c_{12}}{c_1} \quad p_2 = \frac{c_2 - c_{12}}{N - c_1}$$

Assumindo uma distribuição binomial $b(k; n, x) = C_n^k x^k (1-x)^{n-k}$ para as palavras, a verossimilhança que de fato vai ser observada é

$$L(H_1) = b(c_{12}, c_1, p) \cdot b(c_2 - c_{12}; N - c_1, p) \text{ para hipótese 1,}$$

$$L(H_2) = b(c_{12}, c_1, p_1) \cdot b(c_2 - c_{12}; N - c_1, p_2) \text{ para hipótese 2.}$$

O \log da razão de verossimilhança será

$$\log(\lambda) = \log\left(\frac{L(H_1)}{L(H_2)}\right).$$

Se obtivermos um valor pequeno para $\log(\lambda)$ significa que a hipótese 2 explica melhor os dados que a hipótese 1, então o par de palavras ' $w_1 w_2$ ' tende a ser um bigrama.

2.3.1.5 Mi-like

Classifica os *n-grams* com uma variação da informação mutua (mutual information). A Mi é calculada da seguinte maneira:

$$Mi = \log\left(\frac{O_{11}}{E_{11}}\right),$$

sendo O_{11} = a frequência conjunta de w_1w_2 e $E_{11} = (n_{p1} \cdot n_{1p})/n$, sua frequência esperada sobre independência, onde n_{p1} é o total de bigramas onde w_2 aparece como segunda palavra, n_{1p} total de bigramas onde w_1 aparece como primeira palavra e n o total de bigramas.

O Mi-like é uma variação do Mi, e a diferença dele é um expoente no valor de três, a fórmula dele é

$$Mi^3 = \log\left(\frac{O_{11}^3}{E_{11}}\right)$$

2.3.1.6 PMI

Pointwise mutual information (PMI) é uma medida de associação, que mede o quanto duas palavras estão associadas, em caso de bigrama. Por exemplo, se o $PMI(w_1, w_2) = 18$, significa que ao saber que na posição i do texto temos a palavra w_1 , então as chances de termos w_2 na posição $i + 1$ aumentará em 18 vezes. Calcula-se a razão entre a probabilidade conjunta dos eventos e suas probabilidades separadas

$$pmi(w_1; w_2) := \log\left(\frac{p(w_1, w_2)}{p(w_1)p(w_2)}\right).$$

2.3.1.7 Poisson-Stirling

A métrica de Poisson-Stirling é uma aproximação logarítmica negativa da medida poisson-verossimilhança.

Tabela 2 – Tabela de coocorrências de w_1 e w_2

	w_2	$\neg w_2$	Total
w_1	n_{11}	n_{12}	n_{1p}
$\neg w_1$	n_{21}	n_{22}	n_{2p}
Total	n_{p1}	n_{p2}	n_{pp}

Conforme a tabela, vamos considerar que n_{11} seja o número de vezes que w_1 e w_2 ocorreram juntas, n_{12} é o número de vezes que w_1 ocorreu com outra palavra que não seja w_2 . O n_{21} é o número de vezes que w_2 apareceu como segunda palavra no bigrama, sendo que a primeira era diferente de w_1 , e n_{22} é o caso que a primeira palavra não é w_1 e a segunda não é w_2 . O n_{p1} é o número total de vezes que w_2 ocorreu como sendo a segunda palavra nos bigrama, n_{1p} o total de vezes que w_1 foi a primeira palavra dos bigramas, e n_{pp} o total de bigramas. Considere

$$M = \frac{n_{p1} \cdot n_{1p}}{n_{pp}}$$

A fórmula dessa medida será:

$$\text{Poisson-Stirling} = n_{11} \cdot (\log(n_{11}) - \log(M) - 1)$$

2.3.1.8 T-Student

Essa métrica observa a média e variância de uma amostra, ao olhar a diferença entre uma média esperada e uma observada, alterada com a variância, ela nos informa o quão provável é retirar uma amostra com aquela média e variância.

Consideremos as fórmulas

$$P(w_1) = \frac{\text{número de vezes que } w_1 \text{ apareceu no corpus}}{\text{número de tokens no corpus}},$$

$$P(w_2) = \frac{\text{número de vezes que } w_2 \text{ apareceu no corpus}}{\text{número de tokens no corpus}}.$$

A hipótese nula é a que a ocorrência entre elas duas palavras são independentes.

$$H_o = P(w_1 w_2) = P(w_1) \cdot P(w_2)$$

Se a hipótese nula for verdadeira, então se gerarmos bigramas aleatoriamente e atribuir o valor 1 quando aparecer $w_1 w_2$ e 0 nos demais casos, teremos uma distribuição Bernoulli com $p = p(w_1) \cdot p(w_2)$, a média será p e a variância $p(1 - p) \approx p$. Seja

$$t = \frac{\bar{x} - u}{\sqrt{\frac{s^2}{N}}},$$

onde \bar{x} é a média observada, u a média esperada, a variância $s^2 \approx x$ e N o total de tokens. Se observarmos na tabela da distribuição e olharmos o valor corresponde a um nível de significância de $\alpha = 0.005$, e esse valor for menor do que o t calculado, então poderemos rejeitar a hipótese nula. No caso de ser maior, a hipótese nula não pode ser rejeitada e w_1 e w_2 não são independentes.

2.3.2 Principal component analysis (PCA)

PCA é a sigla utilizada para *principal component analysis* (análise do componente principal). *PCA* é uma técnica estatística que tem aplicações em áreas como reconhecimento facial e compreensão de imagens. É uma técnica comum para encontrar padrões em dados de alta dimensão. O *PCA* foi utilizado nesse trabalho para reduzir as dimensões dos vetores, de 100 para 2, para ser assim possível a observação do arranjo geométrico de alguns vetores em duas dimensões. O *PCA* pode ser resumido em alguns passos:

- Colher os dados
- Subtrair a média
- Calcular a matriz de covariância
- Calcular autovetores e autovalores
- Escolher componentes e formar vetor de característica
- Gerar novos dados

Se considerarmos, por exemplo, que temos dados em 3 dimensões, x, y e z , e que a média de x é $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$, podemos ver os dados e os dados ajustados (dados menos a média) como tabelas:

Tabela 3 – Dados e dados ajustados com a média

X	Y	Z	X	Y	Z
x_1	y_1	z_1	$x_1 - \bar{x}$	$y_1 - \bar{y}$	$z_1 - \bar{z}$
x_2	y_2	z_2	$x_2 - \bar{x}$	$y_2 - \bar{y}$	$z_2 - \bar{z}$
x_3	y_3	z_3	$x_3 - \bar{x}$	$y_3 - \bar{y}$	$z_3 - \bar{z}$
x_4	y_4	z_4	$x_4 - \bar{x}$	$y_4 - \bar{y}$	$z_4 - \bar{z}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

Lembrando que a covariância entre duas variáveis aleatórias é

$$Cov(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

Iremos então gerar uma matriz de covariância \mathbf{C} , $n \times n$, sendo n o número de dimensões. Aqui no exemplo $n = 3$, logo a matriz de covariância será 3×3 .

$$\mathbf{C} = \begin{bmatrix} Cov(x, x) & Cov(x, y) & Cov(x, z) \\ Cov(y, x) & Cov(y, y) & Cov(y, z) \\ Cov(z, x) & Cov(z, y) & Cov(z, z) \end{bmatrix}.$$

Depois de ter calculado a matriz de covariância, é necessário achar autovetores (v_1, v_2, v_3) e autovalores $(\lambda_1, \lambda_2, \lambda_3)$ correspondentes.

Considere $\lambda_1 > \lambda_2 > \lambda_3$. Como desejamos diminuir para duas dimensões, vamos pegar dois autovetores, v_1 e v_2 , correspondentes a λ_1 e a λ_2 , respectivamente.

Devemos pegar os autovetores escolhidos e tratá-los como vetores linha, e em seguida colocá-los juntos em uma matriz. Nesta matriz, a primeira linha é o autovetor associado com o maior autovalor, na segunda linha o autovetor associado ao segundo autovalor e assim por diante.

Para obter os dados finais, ou seja, a representação em duas dimensões dos nossos vetores, basta realizar a seguinte operação:

$$\begin{bmatrix} \text{---} & v_1 & \text{---} \\ \text{---} & v_2 & \text{---} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

Obtendo assim x_1 e y_1 , as duas novas coordenadas do primeiro dado (que originalmente tinha as coordenadas x_1, y_1, z_1 em três dimensões).

2.3.3 Gradiente descendente estocástico

Tanto problemas da área de estimativa estatística, como da área de aprendizado de máquina (*machine learning*), se utilizam da minimização de uma função objetiva. Neste trabalho foi utilizado a técnica de estimativa conhecida como gradiente descendente estocástico (*stochastic gradient descent*), pois esta foi utilizada no trabalho de Mikolov [13].

O gradiente descendente estocástico é um método de otimização, onde o parâmetro desejado é atualizado a cada iteração, como na maioria dos algoritmos de otimização.

Nesse trabalho ele foi utilizado para podermos calcular a matriz de tradução, usada para achar uma correspondência entre as palavras, do espaço vetorial do treino em Português, no espaço vetorial do treino em Inglês.

O espaço do Português será referido como fonte e o do Inglês como alvo. Então queremos achar W , que é uma matriz de pesos, sendo X_i vetores treino do espaço fonte e Z_i sua tradução correspondente no espaço alvo. O problema de otimização é:

$$\min_W \sum_{i=1}^n \|X_i \cdot W - Z_i\|^2$$

Para isso a cada iteração, com cada um dos vetores do treino, as entradas de W serão atualizadas da seguinte forma:

$$W_{ij}^{new} = W_{ij}^{old} - \alpha(z_j - y_j)x_i$$

onde α representa a taxa de aprendizado da rede neural, tradicionalmente é desejável utilizar um valor pequeno. Vamos ilustrar um exemplo de como funciona a primeira iteração desse método.

Considere X o primeiro vetor de treino do Português e Z o primeiro vetor do treino do Inglês. Seja $X = [x_1, x_2, x_3]$, $Z = [z_1, z_2, z_3]$, W uma matriz 3×3 e $Y = X \cdot W = [y_1, y_2, y_3]$.

As entradas iniciais de W são geradas aleatoriamente. Consideremos a seguinte função de perda:

$$loss = \frac{\sum_{i=1}^3 \|z_i - y_i\|^2}{2}.$$

Logo estamos interessados na derivada parcial de $loss$ com relação a cada um dos W_{ij} .

$$\frac{\partial loss}{\partial W_{ij}} = (z_j - y_j)x_i$$

Alguns modelos utilizam a função sigmoide neste método como função de ativação, que é a responsável pela ativação das saídas. Nesses casos teremos $K = X \cdot W$ e

$$y = \frac{1}{1 + e^{-K}}$$

e a derivada de y com relação a W é

$$\frac{\partial y}{\partial W} = \frac{Xe^{-K}}{(1 + e^{-K})^2}.$$

Portanto a atualização de cada entrada de W será

$$W_{ij}^{new} = W_{ij}^{old} - \alpha(z_j - y_j) \frac{\partial y}{\partial W_j} x_i.$$

3 Metodologia

Neste capítulo são apresentados os passos necessários para a realização do experimento, abrangendo as etapas de limpeza e preparo do Corpus para utilizá-lo no modelo do *word2vec*, além de apresentar a geração da matriz de tradução e algumas dificuldades encontradas.

3.1 Etapas do experimento

O trabalho aqui descrito foi baseado no experimento que Mikolov fez com as línguas Inglesa, Espanhola e Tcheca [13], e aqui será com Inglês e Português. O objetivo do trabalho é utilizar dois corpus monolíngues, tratar os dois corpus, em seguida utilizar o *word2vec* para gerar os vetores de palavras em cada uma das línguas e calcular uma matriz de tradução que consiga transpor um vetor de uma língua A para uma língua B. Aqui serão mostrados os passos para poder gerar o modelo.

Os passos a serem feitos antes de se utilizar o *word2vec* foram:

- Obtenção do Corpus
- Diminuição do número de documentos
- Remoção das linhas com <doc>
- Remoção dos endereços web (qualquer token que contenha http ou www)
- Tokenização
- Remoção de caracteres de pontuação (como !?,) restando apenas os pontos finais
- *N-gram* representados como um token
- Remoção de stopwords

Depois diversas tentativas verificou-se que seguir os passos, nessa ordem, era a melhor maneira de preparar o Corpus. Essa tentativas incluiram testar os passos em ordens diferentes e tentar fazer os procedimentos com um único texto para cada um das línguas. Também foi contemplada a ideia de identificar automaticamente *n-grams* números e representa-los como um token, mas como seria necessário um algoritmo melhor do que o imaginado, logo essa ideia ficou para trabalhos futuros.

3.2 Obtenção dos corpus

Os corpus utilizados para reproduzir os experimentos foram os dumps (dados de uma base de dados) da Wikipédia em Inglês¹ e em Português². A parte utilizada foi a que contém a descrição *Recombine articles, templates, media/file descriptions, and primary meta-pages*.

Para extrair os dados foi utilizado um script³ chamado 'WikiExtractor.py'. Essa ação pode ser executada rodando o código abaixo no terminal

```
bzcat nome_do_arquivo | python WikiExtractor.py.
```

O dump da enwiki resultou em 4.200 arquivos de texto, distribuídos em 42 pastas, e da ptwiki 2.513 arquivos, distribuídos em 26 pastas. No Português foram 881.027 artigos da Wikipédia distribuídos nas 26 pastas, com uma média de 33.885 artigos por pasta. O que continha mais artigos foram 56.216 e o que continha menos foram 5.368. No Inglês foram 317.612 artigos, com média de 7.562 por pasta.

A primeira etapa foi reduzir o número de arquivos trabalhados, através de um programa feito em Python, o qual juntou todos os textos de uma pasta em um único arquivo de texto. Restaram 42 arquivos de texto para o Inglês e 26 para o Português, os quais somam 2.1 Gb e 1.3 Gb respectivamente. Isso foi preferível ao invés de ter apenas um arquivo para cada língua, para assim não sobrecarregar a máquina ao processar esses textos nas próximas etapas.

3.3 Remoção dos docs

Há vários artigos da Wikipédia em cada um dos textos trabalhos. No começo de cada artigo temos o sua representação com um ‘doc id’, como por exemplo:

```
<doc id="412475" url="https://pt.wikipedia.org/wiki?curid=412475" title="Cinemateca Brasileira">
```

Nesta etapa o foco foi retirar as linhas que continham essas informações. Então toda linha em que se encontrava um <doc id=...> ou </doc> foi eliminada. Em seguida foram retirados todos os endereços de páginas na web, isto é, qualquer token que contivesse http ou www.

3.4 Tokenização

Na etapa da tokenização foi utilizado o script do Statmt⁴ nos 68 textos, preparando-os assim para separação das palavras com as pontuações.

¹ <http://dumps.wikimedia.org/enwiki/20150901/>

² <http://dumps.wikimedia.org/ptwiki/20150901/>

³ https://github.com/NAMD/ptwp_tagger

⁴ <http://www.statmt.org/wmt09/scripts.tgz>

Para utilizar o script foi rodada uma linha de código no terminal no seguinte modelo:

```
scripts/tokenizer.perl -l en < wiki-en/AA/wiki_00 > working-dir/parte1/wikien-AA00
```

Esse script separa as palavras das pontuações, deixando um espaço entre elas. Por exemplo a frase

‘‘Este artigo é sobre cálculo estocástico em geral, para equações diferenciais estocásticas.’’

seria separada nos seguintes tokens: ‘‘‘Este’ ‘artigo’ ‘é’ ‘sobre’ ‘cálculo’ ‘estocástico’ ‘em’ ‘geral,’ ‘para’ ‘equações’ ‘diferenciais’ ‘estocásticas.’’’.

Ou seja ‘‘‘Este’ seria considerado um único token, com as aspas assim inclusas, e o mesmo ocorre com ‘estocásticas.’’’.

Com o script sendo utilizado antes, a separação de tokens fica:

‘‘‘‘Este’ ‘artigo’ ‘é’ ‘sobre’ ‘cálculo’ ‘estocástico’ ‘em’ ‘geral’ ‘,’ ‘para’ ‘equações’ ‘diferenciais’ ‘estocásticas’ ‘.’ ’’’

Dessa forma fica muito mais fácil a remoção das pontuações e a separação de frases, que ocorrerão nas etapas seguintes. A retirada dos endereços web tive que ser antes, justamente por representarem apenas um token antes do script, e após o mesmo seria mais trabalhoso retirar o endereço inteiro.

Em seguida, foi a etapa de retirar toda a pontuação dos textos, e todos os tokens ficaram com um espaço de distância. Nesse processo foram feitas duas cópias para cada arquivo: uma sem pontuação nenhuma e a outra quase sem pontuação, restando apenas as de final de frase (‘.’, ‘!’ e ‘?’) que foram transformadas em ponto final.

Essa distinção foi necessária, pois para a etapa de obtenção dos *n-grams* foram utilizados os arquivos sem pontuação, e para o processo de gerar o input para o modelo *word2vec* foi necessário delimitar aonde as frases terminavam, a demarcação utilizada foi o ponto final.

3.5 Obtenção de *n-grams*

A etapa seguinte foi o processo de obter os *n-grams*. Foi utilizada a biblioteca fornecida pelo NLTK⁵, dessa forma foram levantados *n-grams* automaticamente. Ao processar cada texto (sem nenhuma pontuação) obtivemos alguns milhões de trigramas possíveis, e um número maior ainda de bigramas. Devido ao tempo que seria gasto para analisar todos, manualmente, se faz necessária a implementação de filtros.

O primeiro filtro utilizado foi a frequência, pois um conjunto de *n-grams* que

⁵ <http://www.nltk.org/howto/collocations.html>

aparecem muitas vezes, tende a ser uma unidade única de sentido. O experimento foi feito procurando trigramas, ou seja, três palavras seguidas que representem um único sentido.

O valor imposto para o filtro de frequência foi igual a 100, ou seja, só passarão para próximo filtro aqueles conjuntos de *n-grams* que apareceram juntos mais de 100 vezes. Essa etapa reduz o número de *n-grams* para a casa dos milhares, algo em torno de 1000 e 2000 possível trigramas. O segundo filtro foi a utilização de uma das métricas disponíveis pelo NLTK.

Para decidir qual das métricas seria a utilizada foi realizado um teste com uma das seleções de trigramas depois da primeira etapa, que continha 1126 trigramas. Eles foram avaliados, manualmente, individualmente, e classificados como sendo válidos como trigramas, como *n-gram* (onde n maior que três), bigrama, casos dúbios ou descartados.

Em seguida, esse texto passou pela segunda etapa em 7 métricas distintas, e os resultados foram comparados para observar a que retornava o maior número de trigramas e quadrígramas do conjunto anterior. As métricas testadas foram:

- Chi-quadrada
- Jaccard
- Verossimilhança (Likelihood ratios)
- Mi-like
- Pointwise mutual information (PMI)
- Poisson-Stirling
- T Student

A métrica que mostrou um melhor resultado foi a Mi-like. No levantamento manual foram averiguados 20 trigramas, 40 bigramas e 16 trigramas que fazem parte de *n-grams* maiores, por exemplo, ‘rio grande do sul’ é um quadrígrama, e em trigramas deve aparecer como ‘rio grande do’ e ‘grande do sul’. O melhor resultado foi o da Mi-like, que retornou 7 dos 16 quadrígramas e 12 do trigramas.

Na etapa seguinte todos os textos foram colocados em letras minúsculas, e passaram por esses dois filtros, o de frequência e o mi-like, para retornar candidatos a trigramas. Em seguida foram analisados manualmente 10 % do total de candidatos de cada texto, e eles foram descartados ou classificados como sendo um bigrama, um trígrama ou um *n-gram* maior. No final todos *n-grams* aprovados foram agrupados. Nas tabelas abaixo temos exemplos dos bigramas, trigramas e quadrígramas extraídos nos modelos.

Tabela 4 – Exemplos de bigramas extraídos no Português

buenos aires	ar livre	altos pirenéus	alta normandia
alto reno	américa latina	animal kingdom	arranha céus
assembléia legislativa	baixa normandia	santa catarina	segunda guerra
serra leoa	sistema solar	são paulo	nova york

Tabela 5 – Exemplos de trigramas extraídos no Português

áfrica do sul	angra do heroísmo	américa do norte
américa do sul	artes marciais mistas	billboard hot 100
campeonato sul americano	cidade do méxico	copa do mundo
hall of fame	comitê olímpico nacional	d joão V
rio de janeiro	guns n roses	música popular brasileira

Tabela 6 – Exemplos de quadrigramas extraídos no Português

campeonato mundial de atletismo	estados unidos da américa
índice de desenvolvimento humano	jogos olímpicos de verão
rio grande do sul	the new york times
united states census bureau	united states of america
nord pas de calais	rendimento médio per capita

Tabela 7 – Exemplos de bigramas extraídos no Inglês

able to	academy award	african american	air force
all star	milky way	native american	new york
new testament	new zealand	nobel prize	spread out
summer olympics	supreme court	soviet union	united kingdom

Tabela 8 – Exemplos de trigramas extraídos no Inglês

as part of	as well as	bank of america	new york city
new york times	new south wales	papua new guinea	tom and jerry
summer olympic games	tour de france	u s army	u s navy
world trade center	world war i	world war ii	world wide web

Tabela 9 – Exemplos de quadrigamas extraídos no Inglês

at the same time	massachusetts institute of technology
national academy of sciences	united states of america
united states postal service	world championships in athletics

Depois de ter feito o levantamento dos *n-grams*, e ter mantido apenas os considerados válidos, utilizamos os textos obtidos após a utilização do script, nos quais a maioria das pontuações foram removidas e os caracteres ‘?’ e ‘!’ foram substituídos por ‘.’. Nesses textos foram feitas as substituições dos *n-grams* encontrados. Após os textos estarem nesse formato final poderemos separá-lo em frases.

3.6 Criação do dicionário bilíngue

A etapa seguinte foi a criação de um dicionário das palavras mais frequentes, como nesse experimento foi do Português para o Inglês, foram consideradas as palavras mais frequentes do Português. Para isso foram retiradas as stopwords, utilizando a lista de stopwords que existe no NLTK. Stopwords são palavras que possuem pouco ou nenhum valor semântico, a retirada de stopwords é uma tarefa comum em análise de textos, pois achar relações entre palavras como ‘e’ e ‘de’ não acrescenta nenhuma informação relevante.

Em seguida foram levantadas as 1000 palavras mais frequentes de cada um dos 26 textos do Português, sem considerar as repetições. Foram levantados 6730 palavras, das quais muitas foram desconsideradas por serem nomes próprios (lisa, jimmy, stephan, anna, laura), símbolos (†, <, >, ↔), palavras estrangeiras (little, were, stone, girls, life, plzeň, pčinja), números (6,0, 5,8, 254, 1987), algarismos romano (xv, viii, ii), abreviações (t., v., h., sc, rj, b., atp), letras soltas (b, r, ss, g) algumas palavras do Português que não existem no Inglês ou são iguais (catarinense, carioca, paulista, paraguaio, baiano) e alguns nomes de lugares (meurthe-et-moselle, haute-saône, maine-et-loire, haute-loire).

Essas escolhas foram necessárias para manter apenas as palavras que sejam relevantes para a língua portuguesa e que tenham um correspondente que seja significativo no Inglês. As traduções foram feitas uma a uma, manualmente, utilizando o *Google Translate* como auxiliar, e algumas escolhas tiveram que ser feitas. Algumas palavras poderiam ser traduzidas de mais de uma forma, com em casos de sinônimos, e foi necessário escolher qual das traduções utilizar. Outras palavras como ‘estaçao’, ‘mata’, ‘cobre’, ‘cobertura’ e ‘tomada’ tem mais de um significado, e dependendo do contexto podem significar coisas bem distintas. Estaçao pode ser traduzida como *station* ou *season*, mata pode ser do verbo matar ou de mata relacionada com floresta, por exemplo.

Além disso, algumas palavras que possuem uma grafia diferente em Português

Europeu e Português Brasileiro foram modificadas no corpus posteriormente, como actual, facto, actor e actriz para atual, fato, ator e atriz respectivamente.

Durante as traduções alguns bigramas e trigramas de verbos e substantivos foram gerados para o Inglês, como por exemplo: new_zealand, south_bohemian, european_union, stand_out. Depois de ter peneirado as palavras e ter feito as traduções, obtivemos um dicionário com um pouco mais de 5000 palavras.

Como mencionado anteriormente, foi necessário remover stopwords antes de gerar os modelos com o *word2vec*. As palavras escolhidas como stopwords foram baseadas na lista disponibilizadas pelo NLTK, com algumas adaptações, uma vez que estamos interessados em diferenças de gênero para o treino do modelo, palavras como ‘ele’, ‘ela’ (no Português) e ‘he’, ‘she’ (no Inglês) foram mantidas no modelo, além de conjugações de verbos.

Tabela 10 – Exemplos de stopwords retiradas do Inglês

our	ours	it	its	itself
who	whom	this	that	these
an	the	and	but	if
as	until	while	of	at
with	about	against	between	into
before	after	above	below	to
down	in	out	on	off
again	further	then	once	here

Tabela 11 – Exemplos de stopwords retiradas do Português

de	os	da	e	o
se	por	as	aquele	entre
nos	pela	havia	me	como
pelos	estes	depois	este	para
isto	do	mesmo	num	a
no	à	em	esses	pelas
que	na	te	aos	dos
das	esta	até	esse	mas

3.7 Gerando os modelos com o *word2vec*

Os modelos foram gerados com o *word2vec*. Para poder rodar o *word2vec* basta instalar o Gensim⁶, e rodar:

```
>>> model = Word2Vec(sentences, size=100, window=5, min_count=5, workers=4)
```

O parâmetro ‘sentences’ é o input. Todos os textos da base de dados devem ser colocadas no formato de frases, como por exemplo:

```
sentences = [['primeira', 'frase'], ['segunda', 'frase']]
```

O argumento ‘size’ é o tamanho dos vetores do nosso modelo, ‘window’ é o tamanho máximo das frases que o modelo vai analisar por vez, e ‘min count’ é quantas vezes uma palavra precisa aparecer, no mínimo, para ser incluída no modelo. ‘Workers’ é o número de *threads* usados para o treinar o modelo. É possível gravar o modelo e depois rodá-lo, sem precisar treina-lo novamente.

Inicialmente haviam 333M tokens no Inglês e 191M no Português. Após a retirada do doc, html e pontuação a quantidade passou a ser 332M no Inglês e 185M no Português. Com as mudanças dos *n-grams* e a retiradas dos stopwords o total de tokens que foram utilizadas para gerar os modelos foram: 235M para o Inglês e 133M para o Português. Nesse total são considerados todos os tokens dos textos, independente de quem são e quantas vezes se repetem. No vocabulário gerado pelo *word2vec* são considerados quantos tokens diferentes existem no modelo, removendo repetições. No modelo em Inglês obtivemos um vocabulário de 513k tokens e no Português um vocabulário de 447k tokens. Os vetores de palavras tem 100 dimensões.

Pode-se testar os modelos de algumas formas, um exemplo é o teste de correspondência que, de forma genérica, é representado por ‘A está para B assim como C está para D’. Outro exemplo é o teste para identificar qual a palavra que não pertence a um grupo, por exemplo no grupo ‘gato cachorro vaca cadeira’ o elemento que não pertence a ele é cadeira, por não ser um animal.

O Google disponibiliza um conjunto de teste⁷ com aproximadamente 20.000 exemplos sintáticos e semânticos, que segue a lógica:

‘‘A está para B assim como C está para D’’.

Esse teste foi adaptado e traduzido manualmente para o Português. A instrução para executar esse teste aparenta ser simples⁸, bastando uma linha de comando:

```
model.accuracy('/tmp/questions-words.txt')
```

Mas, infelizmente, a tentativa de utilizar essa linha de comando não ocorreu como o esperado. Logo, foi necessário programar esse teste para analisar as correspondências.

⁶ <https://radimrehurek.com/gensim/install.html>

⁷ <http://word2vec.googlecode.com/svn/trunk/questions-words.txt>

⁸ <http://rare-technologies.com/word2vec-tutorial/>

Com o dicionário em mãos, ainda é necessário verificar a cobertura do vocabulário no Inglês, pois podem haver palavras que estão no dicionário e não estão no modelo. As palavras do Inglês que não se encontravam no modelo tiveram seus pares descartados do dicionário, restando ao todo 4811 pares. Desse total 3311 foram utilizadas para treino e 1500 para teste.

3.8 Gradiente descendente estocástico

No experimento de Mikolov foram usadas as 5000 palavras mais frequentes da língua fonte, e suas traduções geradas automaticamente pelo *Google Translate*, como treino. 1000 palavras na língua fonte e suas traduções foram deixadas para teste. Foram reportados os 5 candidatos mais prováveis (top 5) para tradução ao invés de uma só palavra como candidata, pois o top 1 (apenas uma tradução) é altamente superestimado devido a traduções de sinônimos serem contadas como um erro.

Utilizando o dicionário, mapeamos as palavras da língua fonte (Português) para a língua alvo (Inglês) com o método do gradiente descendente estocástico. Foram utilizadas seis variações desse mapeamento para comparações:

- W_1 - Sem a função sigmoide e com 3311 palavras para treino
- W_2 - Com função sigmoide e com 3311 palavras para treino
- W_3 - Sem função sigmoide e repetindo as 3311 palavras 10 vezes
- W_4 - Com função sigmoide e repetindo as 3311 palavras 10 vezes
- W_5 - Sem a função sigmoide, com 3311 palavras para treino e distribuição uniforme em um intervalo pequeno (menor que $[0, 1]$)
- W_6 - Sem a função sigmoide, com 3311 palavras para treino e distribuição normal em um intervalo pequeno (menor que $[-1, 1]$)

Cada entrada das quatro primeiras matrizes foi gerado inicialmente por uma distribuição uniforme $[0, 1]$. As entradas das duas últimas tiveram o tamanho do intervalo da distribuição testado, e a média, de cada entrada de W_6 , foi mantida com o valor zero. Cada uma das variações acima gerou uma matriz W de tradução diferente. No algoritmo do gradiente descendente é necessário determinar previamente a taxa de aprendizado, denotado por α . Para isso foram feitos testes para calcular qual seria o valor que gera o menor erro relativo.

3.9 Hubs

O Gensim⁹ disponibiliza algumas funções que podem ser utilizadas em modelo. Dentre essas funções, existe a função *most similar*, que pode ser usada para procurar os

⁹ <https://radimrehurek.com/gensim/models/word2vec.html>

vizinhos mais próximos de uma palavra, usando o valor do cosseno entre seus vetores como medida, ou para procurar a palavra mais similar em um teste de correspondência.

Um dos primeiros testes realizados com os modelos foi a busca dos k vizinhos mais próximos das palavras mapeadas, e este não funcionou como esperado, as traduções corretas não se encontravam com k igual a cinco, dez ou cem. Após análise e pesquisa na literatura, surgiu a suspeita de existirem hubs no espaço. Hubs são vetores que tendem a estar próximos de vários outros ao mesmo tempo. A presença desses vetores é um problema intrínseco de espaços com altas dimensões.

No trabalho de Dinu [3] foi apresentada uma forma de medir o quanto um vetor é um hub, isto é, quanto mais alta sua pontuação mais *hubby* ele é. É utilizada a nomenclatura *hubby* para dizer se referir a qualidade de ser hub. Também foram propostos dois métodos para corrigir globalmente a recuperação de vizinhos.

A partir desse momento os vetores mapeados com as matrizes W , do Português para o Inglês, serão chamadas de pivôs neste trabalho. Para medir o quanto *hubby* um vetor é com respeito a um conjunto de pivôs basta contar quantas vezes ele aparece como um dos k -vizinhos de cada um dos pivôs.

Existem alguns métodos para corrigir a recuperação a vetores, diminuindo a importância dos hubs. Um método proposto por Radovanovic et al [14] e Tomasev et al [16] foi o de calcular as pontuações de quanto *hubby* cada um dos elementos no espaço alvo eram. Assim dado um rank de vizinhos próximos a um vetor podemos refazer esse rank diminuindo a importância dos elementos com uma pontuação alta.

No trabalho de Dinu foram propostos dois métodos, que neste trabalho foram reproduzidos. O primeiro método consiste em normalizar os vetores de similaridade. Para isso é necessário criar vetores de similaridade para cada um dos elemento do espaço alvo, neste caso são todos os vetores do vocabulário do modelo do Inglês, pois todos tem potencial de ser uma tradução correta.

Um vetor de similaridade de um dos vetores do espaço, possui 1500 dimensões, pois cada uma das entradas é associada com um dos pivôs. Por exemplo, na primeira entrada vai estar o valor do cosseno entre os dois vetores, o vetor do espaço alvo em questão e o primeiro dos 1500 pivôs. A normalização é necessária para penalizar os vetores que tem alta similaridade com muitos pivôs. Esta foi calculada usando a função *linalg.norm* em *Python*. Para normalizar um vetor divide-se cada uma de suas entradas pelo raiz da soma dos quadrados de todas as entradas.

O segundo método proposto não modifica os pesos das pontuações de similaridade, porém modifica o rank dos elementos alvos ao invés de retornar os vizinhos mais próximos da forma tradicional. Desta forma é usada uma abordagem de correção global. A ideia dessa correção global é mudar a forma de listar: ela retorna o elemento do espaço alvo

no qual o pivô aparece com um rank maior. Como o pivô não aparece explicitamente no rank, é possível avaliar sua posição, comparando o valor de similaridade do elemento com o pivô e as demais similaridades que aparecem no rank.

4 Resultados

Após recolher todos os dados do experimento aplicamos testes, que foram registrados nesse capítulo assim como os resultados obtidos. Na maioria deles foram utilizadas as 6 matrizes de tradução, mencionadas previamente, para fins de comparação. Os testes, assim como os códigos para limpar os dados e gerar o modelo, foram feito em Python e estão disponível no GitHub¹.

4.1 Teste com *PCA*

A técnica do *PCA* foi utilizada logo após a geração do modelo usando o *word2vec*. Com ela podemos notar alguma semelhança no arranjo geométrico das palavras dos dois espaços aqui estudados. Isso ocorre pois palavras com significados parecidos tendem a ocorrer no mesmo contexto em línguas distintas.

A ideia de poder achar uma matriz de tradução entre dois espaço deriva da ideia deles possuírem um arranjo geométrico semelhante. Nos gráficos a seguir temos alguns exemplos de representações vetoriais, projetados em duas dimensões usando a técnica do *PCA* e, rotacionados manualmente para acentuar as similaridades entre as duas línguas.

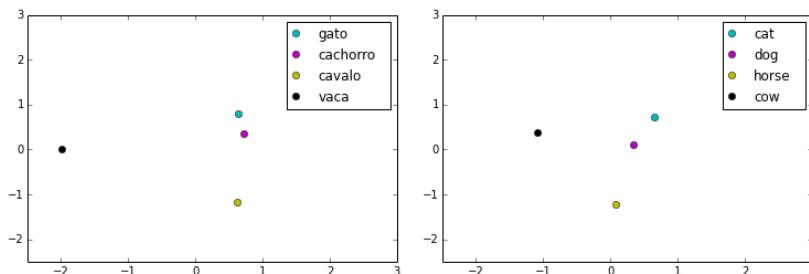


Figura 3 – Exemplo com animais usando *PCA*

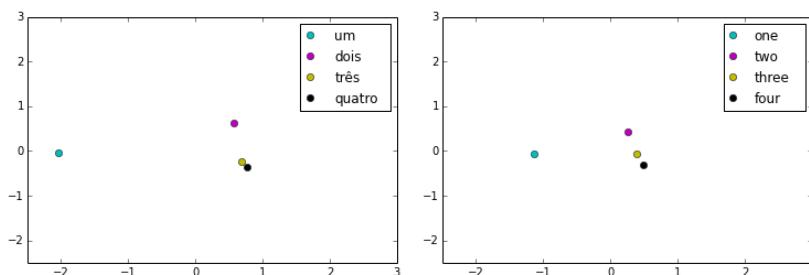
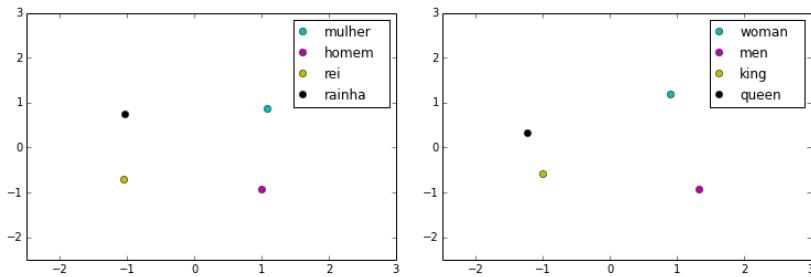
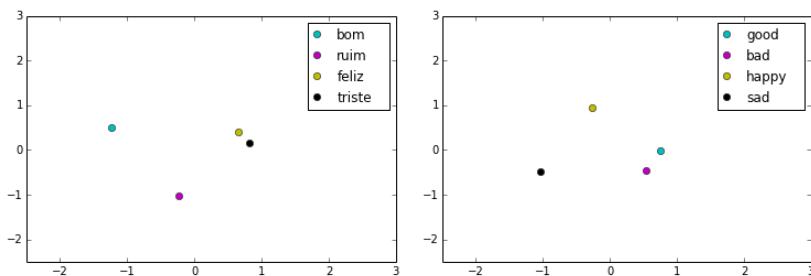


Figura 4 – Exemplo com números usando *PCA*

¹ <https://github.com/elianemart/Projeto>

Figura 5 – Exemplo com gêneros usando *PCA*Figura 6 – Exemplo com opositos usando *PCA*

4.2 Teste de correspondência

O teste realizado com o modelo de correspondência disponibilizado pela Google, em Inglês, não funcionou conforme as instruções fornecidas e foi necessário adaptá-lo, além de ter sido feita uma versão em Português para este trabalho.

Foram testadas 12 combinações das correspondências ($A + B - C = D$), sendo A,B,C e D cada uma das palavras a serem testadas. Com quatro palavras obteríamos 24 combinações, mas levando em conta que $A + B = B + A$, o total passa a ser 12.

Na tabela abaixo são mostrados os melhores resultados obtidos em cada um dos testes. Ocorreram vários casos em que uma palavra do teste não aparecia no Corpus, logo isso afetou a verificação da acurácia dos modelos.

Tabela 12 – Teste de correspondência

Categoria	Português	Inglês
adjetivo para advérbio	4.53%	12.5%
capitais do mundo	25.69%	67.19%
cidades e estados	12.16%	26.02%
comparativos	16.97%	52.70%
moeda	1.04%	11.43%
família	27.42%	54.94%
nacionalidades-adjetivos	3.46%	76.99%
opostos	6.54%	16.26%
capital países comuns	46.84%	38.02%
passado simples	27.43%	42.17%
plural	9.16%	40.01%
plural- verbos	31.38%	42.99%
particípio	1.04%	44.70%

4.3 Relações entre as palavras

Utilizando a função *most similar* disponível pelo Gensim² foram levantadas algumas relações entre palavras do Português e do Inglês. Essa função retorna ao usuário qual é o vetor de palavra mais próximo do encontrado em uma operação. Nessa seção serão mostradas 9 relações de correspondência, tanto no Português quanto no Inglês, com os resultados divididos em 3 categorias. As relações testadas nas duas línguas são equivalentes.

As categorias dos resultados são: (1) correto; (2) similar; (3) errada. Em todas relações existiam palavras que esperávamos encontrar, logo se no teste encontrávamos exatamente a palavra esperada, a relação era classificada na primeira categoria. Se a palavra encontrada não era exatamente a esperada, porém era uma palavra similar a esta, a relação era classificada na segunda categoria. E por último, se a palavra encontrada era completamente diferente da esperada, a relação é classificada na terceira categoria.

No Português 3 resultados ficaram no tipo 1, 4 no tipo 2 e 2 no tipo 3. No Inglês 2 resultados ficaram no tipo 1, 1 no tipo 2 e 6 no tipo 3.

Do primeiro tipo (resultado correto) no Português:

- Ele + Atriz - Ela = Ator

- Bonito + Sujo - Feio = Limpo

² <https://radimrehurek.com/gensim/models/word2vec.html>

- Maior + Pequeno - Grande = Menor

E no Inglês:

- Woman + King - Man = Queen
- He + Actress - She = Actor

Do segundo tipo (resultado parcialmente correto) no Português:

- Mulher + Rei - Homem = Consorte
- Menina + Menino - Homem = Moça
- Menina + Menino - Mulher = Garoto
- Bom + Ruim - Triste = Ótimo

E no Inglês:

- Good + Happy - Sad = Pleased

Do terceiro tipo (resultados incorretos) no Português:

- França + Brasil - Francês = Espanha
- França + Itália - Paris = Holanda

E no Inglês:

- Girl + Boy - Woman = Kid
- Girl + Boy - Man = Baby
- France + Brazil - French = Argentina
- France + Italy - Paris = Spain
- Beautiful + Dirty - Ugly = Filthy
- Bigger + Small - Big = Larger

4.4 Testes dos alphas

Para utilizar o método do gradiente descendente estocástico é necessário definir o valor de α , a taxa de aprendizado. Para podermos definir valores adequados para α em cada uma das matrizes de tradução (os W), foi realizado um teste com cada uma delas, onde o parâmetro de escolha foram os erros relativos.

Foram testados 12 valores de α para cada uma das quatro primeiras matrizes W , onde ele variou de 10^{-6} até 10^{-17} . Os valores de 10^{-1} a 10^{-5} foram desconsiderados por darem um erro muito maior que os demais.

Com as duas últimas matrizes foram testadas combinações entre os valores de α e um índice que foi chamado de β . O índice β representa a cota superior do intervalo da distribuição, $[0, \beta]$. Para o cálculo desse índice foram testados os valores de 10^{-1} a 10^{-5} , e do α os valores de 10^{-6} a 10^{-10} . Com cada uma das matrizes W foram gerados 1500 vetores testes y_i , logo, utilizando eles e os vetores z_i correspondentes (os vetores das traduções corretas), os erros relativos puderam ser calculados da seguinte forma:

$$Total_j = \sum_{i=1}^{1500} \frac{\|y_i - z_i\|}{\|z_i\|}$$

e

$$Error_j = \frac{Total_j}{1500}$$

Nos gráficos abaixo temos os valores de α no eixo x , e os valores dos $error_j$ no eixo y , para cada uma das matrizes W . Os gráficos estão em escala logarítmica.

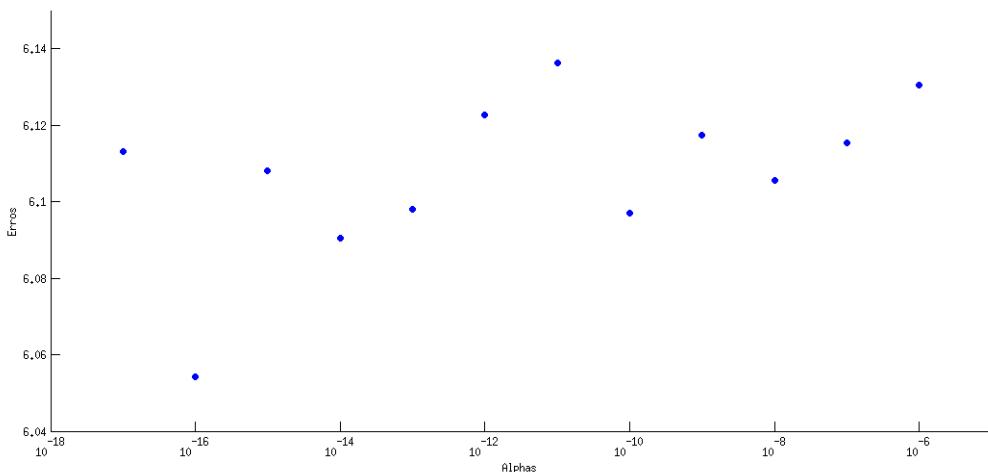
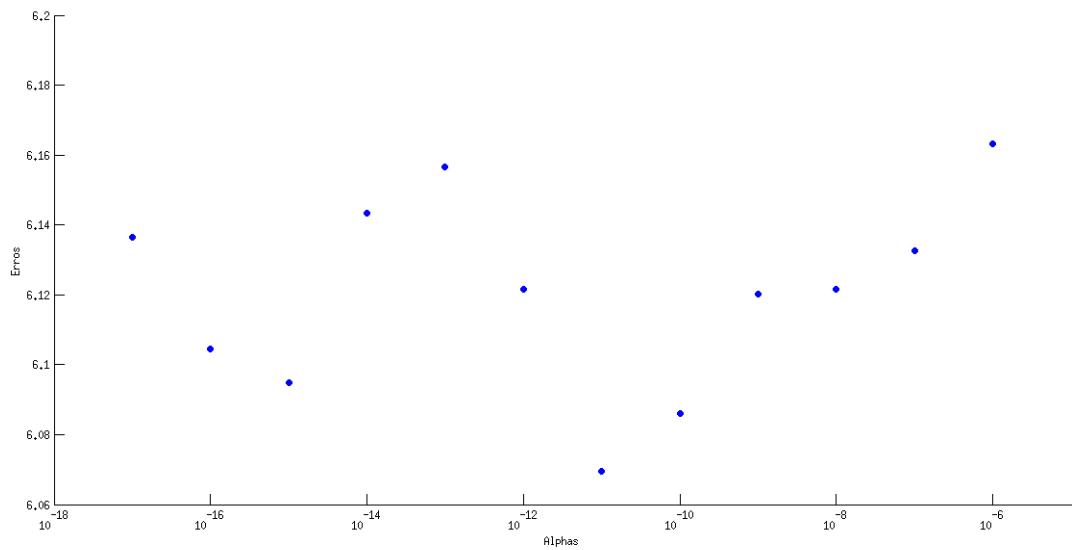
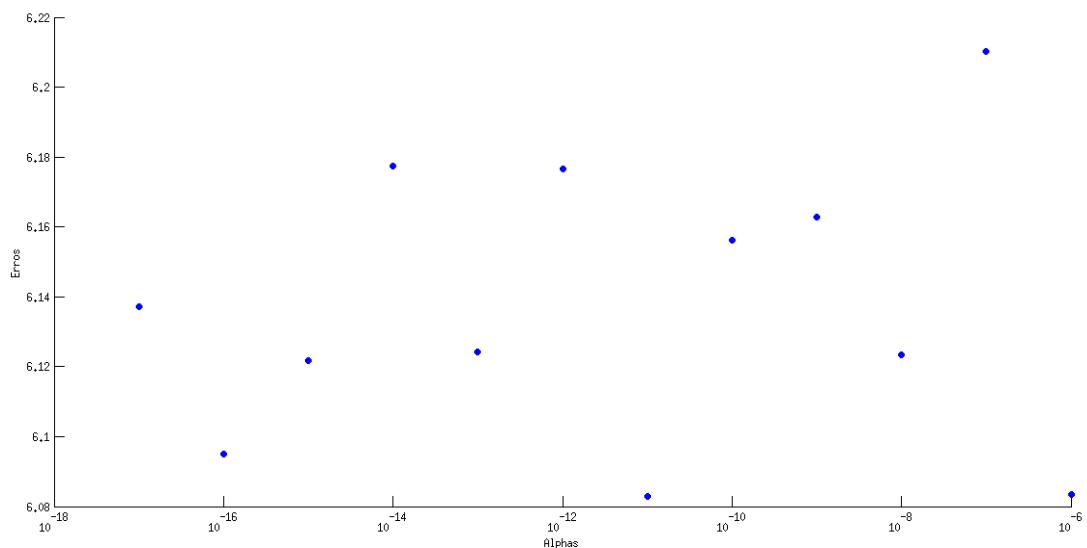
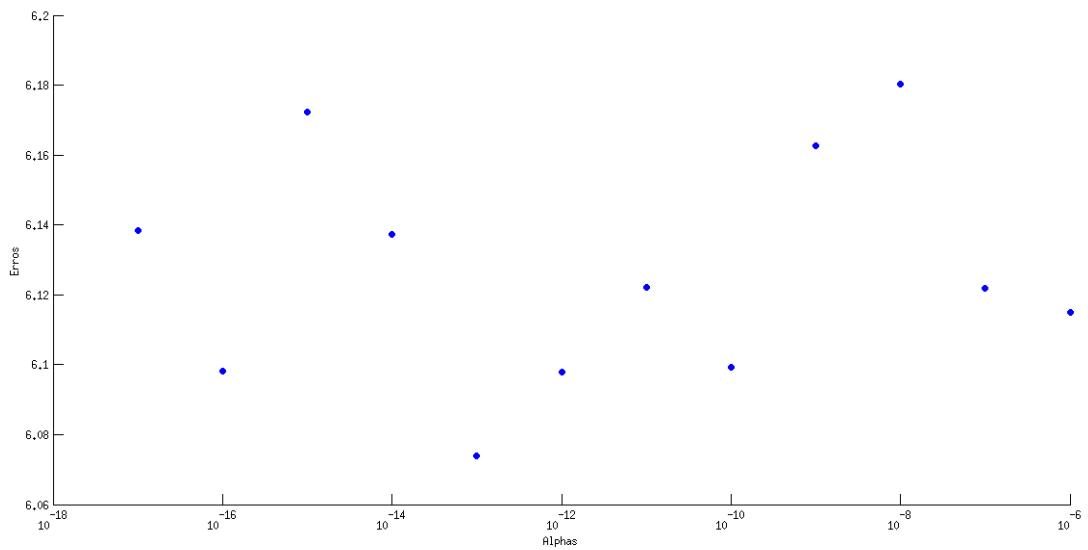
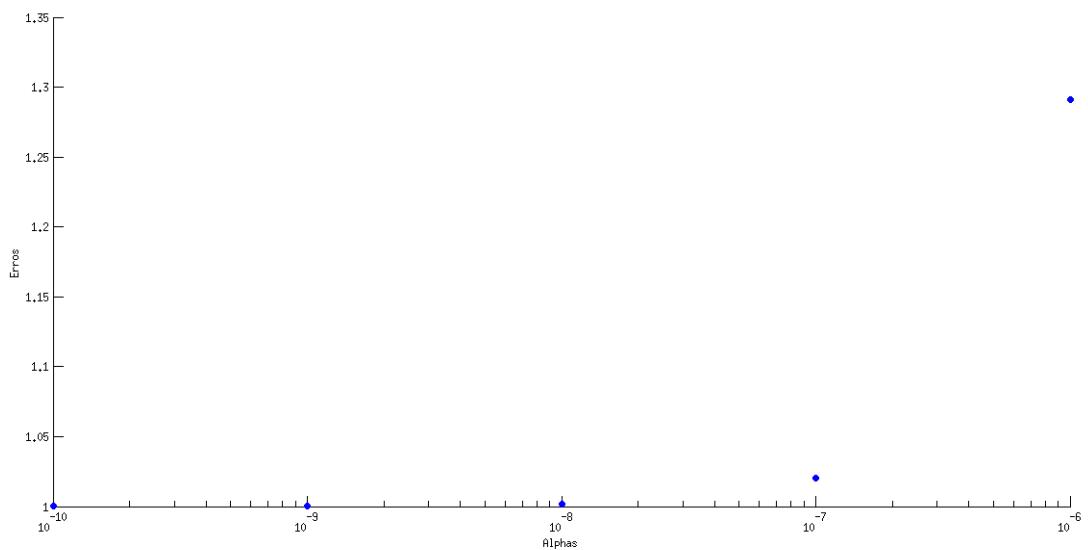
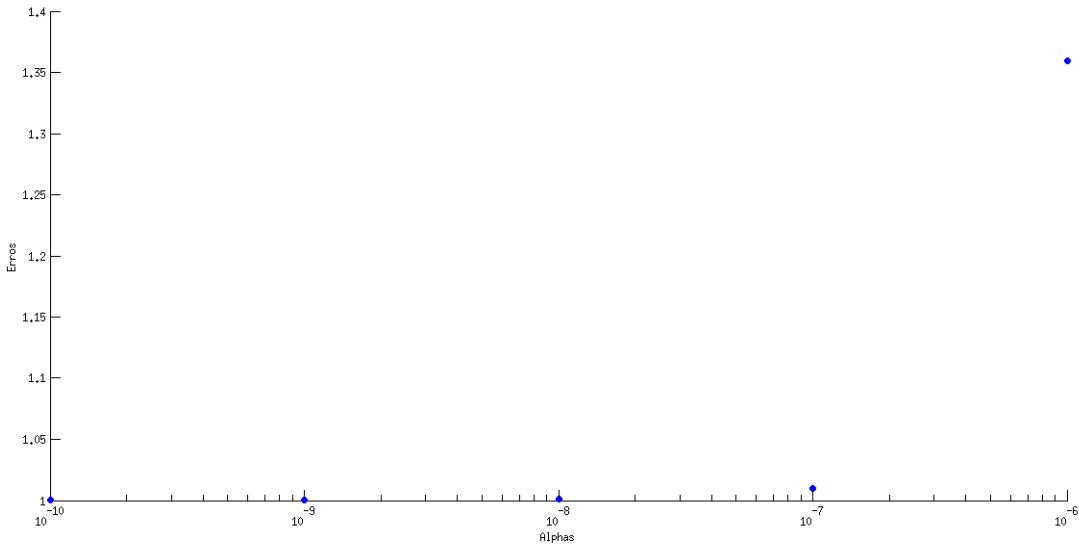


Figura 7 – Erros de W_1

Figura 8 – Erros de W_2 Figura 9 – Erros de W_3

Figura 10 – Erros de W_4 Figura 11 – Erros de W_5

Figura 12 – Erros de W_6

As diferenças dos erros encontrados foram bem pequenas, pois a ordem de grandeza destes erros foi bem próximas apesar da variação nos valores de α . Nos gráficos de W_5 e W_6 o valor de β foi fixado previamente em 10^{-5} , pois foi o que gerou os menores erros, enquanto o valor de α foi variado. Os valores escolhidos de α foram os que resultaram em menor erro relativo. Para W_1 foi 10^{-16} , para W_2 foi 10^{-11} , para W_3 foi 10^{-11} e para W_4 foi 10^{-13} . Para W_5 e W_6 foi 10^{-10} para α e 10^{-5} para β . De W_1 a W_4 as diferenças entre os erros foram menores que 0,16. É possível observar que os valores dos erros, nas primeiras 4 matrizes, oscilam dentro de uma margem pequena, em geral essa margem é um valor próximo a 0,1. Os erros encontrados para as duas últimas matrizes foram bem menores que os encontrados para as matrizes anteriores.

4.5 Teste de número de hubs em K vizinhos

Este teste consiste em analisar K vizinhos de cada um dos pivôs e registrar todas as palavras que aparecem como vizinhas e observar quantas vezes uma mesma palavra aparece. Neste teste foi utilizada a função *most similar*, mencionada previamente, para encontrar as palavras mais próximas.

Vamos considerar que o total de vezes que uma palavra aparece nos ranks nos indique o quão *hubby* uma palavra é, ou seja, palavras que obtenham uma pontuação alta nesse teste são consideradas hubs. O teste foi realizado com $K = 5$, $K = 10$, $K = 20$ e $K = 50$.

Nas tabelas abaixo temos quantos vizinhos distintos foram gerados em cada caso, quantos desses vizinhos aparecem entre 10 e 100 vezes, e quantos aparecem mais de 100.

Tabela 13 – Número de Hubs com K=5

Matriz	Total	Entre 10 e 100	Mais de 100
W_1	3154	75	10
W_2	3138	69	10
W_3	3162	64	12
W_4	3004	76	7
W_5	3120	68	9
W_6	6345	0	5

Tabela 14 – Número de Hubs com K=10

Matriz	Total	Entre 10 e 100	Mais de 100
W_1	5946	139	22
W_2	5823	153	20
W_3	5924	149	20
W_4	5635	138	22
W_5	5811	157	19
W_6	11999	5	10

Tabela 15 – Número de Hubs com K=20

Matriz	Total	Entre 10 e 100	Mais de 100
W_1	11089	174	46
W_2	10845	305	43
W_3	11123	313	41
W_4	10455	277	46
W_5	10629	313	46
W_6	22298	15	20

Tabela 16 – Número de Hubs com K=50

Matriz	Total	Entre 10 e 100	Mais de 100
W_1	24294	761	116
W_2	24408	739	116
W_3	24009	738	126
W_4	23596	757	119
W_5	23180	741	123
W_6	48787	105	50

Como pode ser observado nas tabelas a cima, há um número considerável de palavras que apareceram repetidas vezes como vizinhas de várias palavras distintas e este é um indicativo de que existem hubs no espaço. Conforme o valor de K aumenta, aumentam também o número de hubs existentes em valor absoluto. Porém, proporcionalmente ao crescimento de K , os hubs não aumentaram tanto. Menos de 1% dos vizinhos encontrados obtiveram um valor muito alto (maior que cem) no teste. W_6 foi quem obteve os melhores resultados, apresentando os menores números de hubs em todos os testes.

4.6 Teste da posição da palavra correta nos ranks dos pivôs

No *word2vec* existe a função *most similar*, que aqui será chamada de rank, que permite escolhermos um valor K , onde K é um número natural, ela nos retorna os K vizinhos mais próximos de um vetor, por ordem de proximidade (ordem decrescente). Isto é, ao usar a função rank em uma palavra ou vetor do modelo, ela nos retorna quais vetores são seus vizinhos. A similaridade utilizada pelo método é o valor do cosseno entre os dois vetores, logo ela pode variar de -1 a 1 .

Nosso interesse é analisar os ranks dos pivôs (os vetores calculados com as matrizes W) para verificar em qual posição do rank se encontra a palavra alvo (a palavra correta). Nos gráficos a seguir os pivôs foram ordenados de forma que ficassem com as posições no rank em ordem crescente.

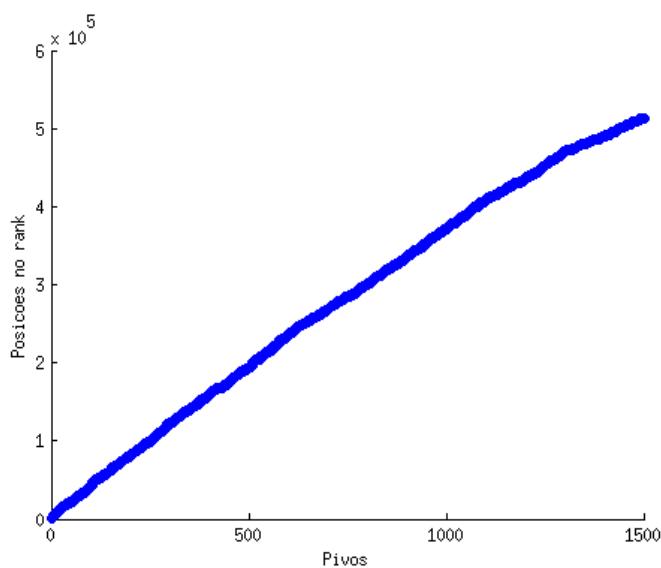
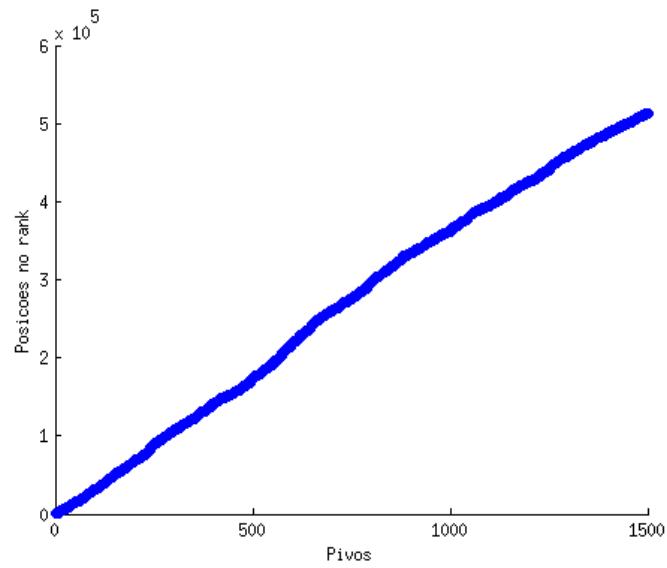
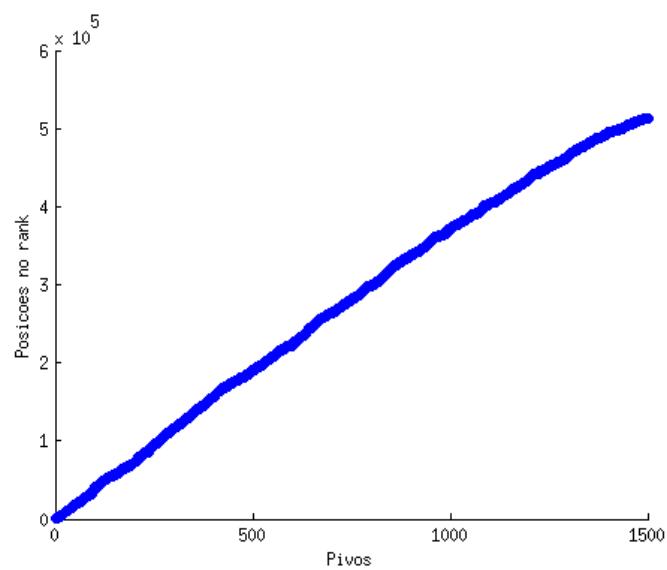
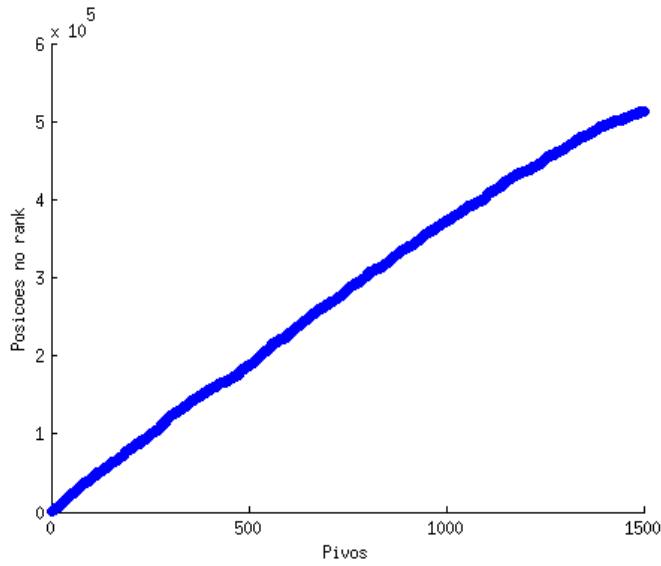
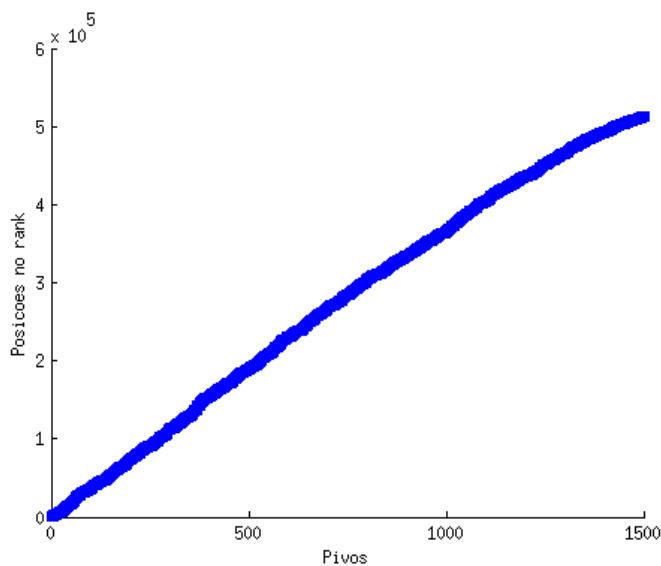
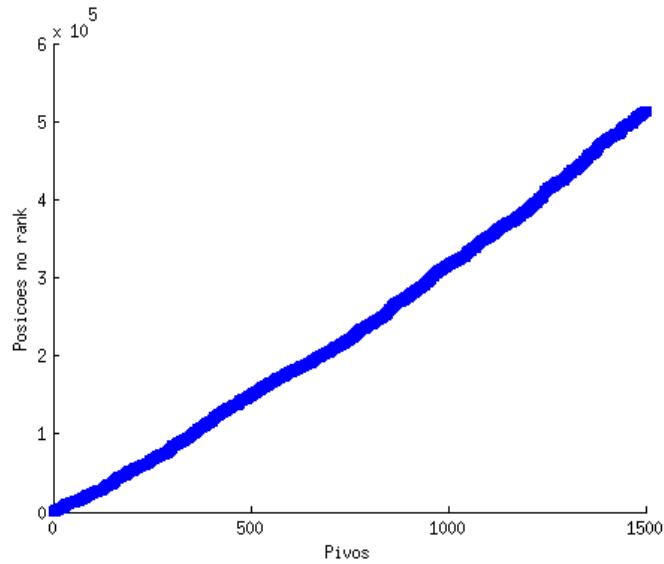


Figura 13 – Posição nos ranks de W_1

Figura 14 – Posição nos ranks de W_2 Figura 15 – Posição nos ranks de W_3

Figura 16 – Posição nos ranks de W_4 Figura 17 – Posição nos ranks de W_5

Figura 18 – Posição nos ranks de W_6

Podemos observar que a distribuição das posições nos ranks é muito similar em todas as matrizes W , apresentando um crescimento linear. Alguns pivôs ficaram abaixo da posição 1000, e a grande maioria ficou bem acima disso. O resultado ideal seria obter valores pequenos, menores que 10 por exemplo, para todas as posições no rank, pois isso significaria que as traduções estariam muito boas visto que os pivôs estariam muito próximos dos alvos. O que ocorreu foi bem diferente do cenário ideal, as posições encontradas foram muito superiores aos valores esperados, indicando que as traduções estão muito distantes do desejável.

4.7 Teste de similaridade dos pivôs com as traduções

Vetores de palavras com alta similaridade tendem a serem próximos semanticamente. Logo, desejamos que nossos pivôs sejam próximos das palavras alvo correspondentes. Portanto, os gráficos ideias seriam com todas as similaridades próximas de 1.

Para medir essas similaridades foram calculados os cossenos entre os pivôs e as palavras alvo correspondentes.

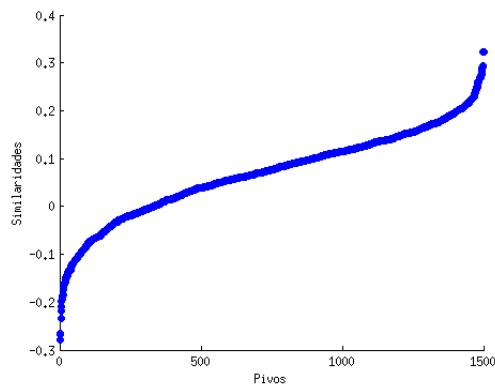


Figura 19 – Similaridade entre pivôs e alvos - W_1

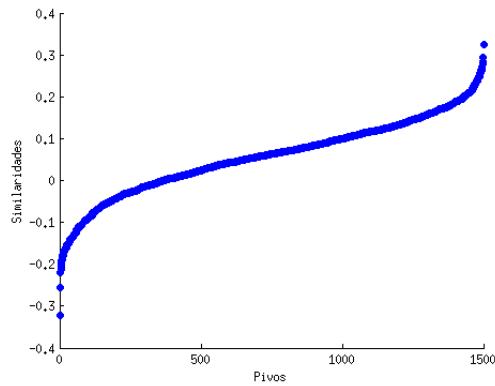


Figura 20 – Similaridade entre pivôs e alvos - W_2

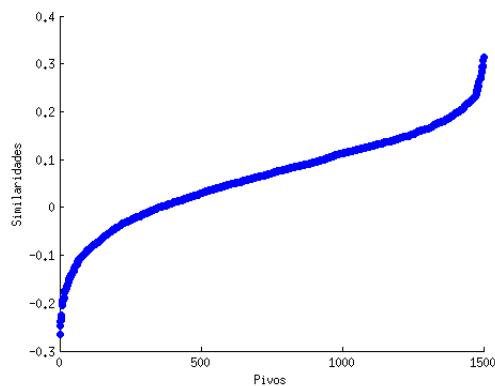
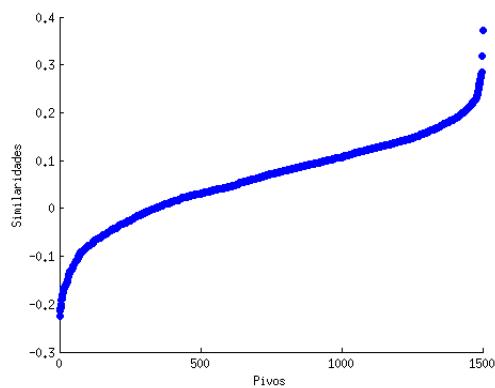
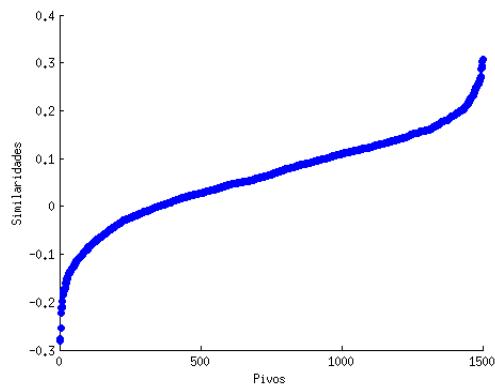
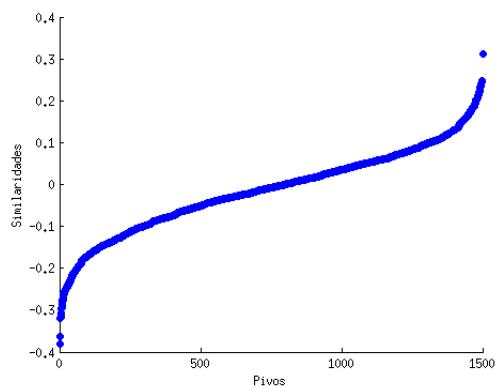


Figura 21 – Similaridade entre pivôs e alvos - W_3

Figura 22 – Similaridade entre pivôs e alvos - W_4 Figura 23 – Similaridade entre pivôs e alvos - W_5 Figura 24 – Similaridade entre pivôs e alvos - W_6

Os gráficos foram parecidos para cada um dos W , apresentando uma inflexão próxima do ponto zero. Os valores oscilaram entre -0,4 e 0,4. Aproximadamente um terço dos pivôs apresentaram similaridade negativa.

4.8 Tratamento de hubs: Normalizar os vetores de similaridade

Nesse teste foram calculados vetores de similaridade de todas as palavras do vocabulário com os vetores gerados utilizando as matrizes W . Os vetores de similaridade são os vetores, normalizados, cuja entrada w_{ij} é o cosseno entre o vetor correspondente à palavra i do vocabulário, e o pivô j . São 513k palavras no vocabulário e 1500 pivôs.

Para cada um dos pivôs é analisado qual palavra do vocabulário possui maior similaridade relativa, a similaridade depois de ser normalizada. As similaridades relativas encontradas foram todas positivas e entre 0.04 e 0.2. O crescimento a princípio parece ser linear mas tem um salto de alguns pontos na extremidade direita.

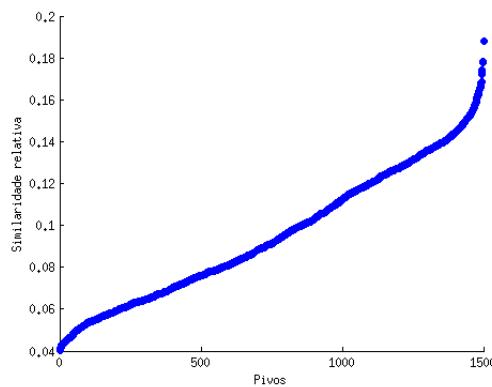


Figura 25 – Similaridade relativa - W_1

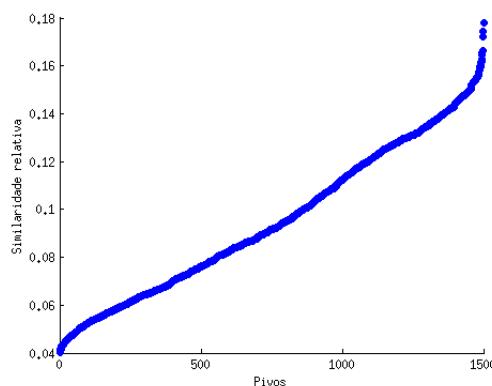
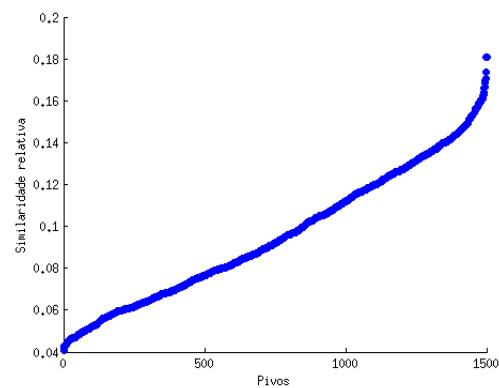
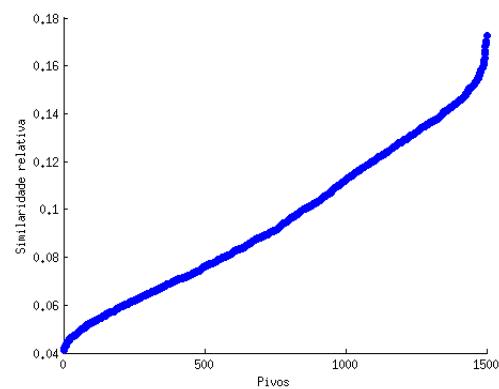
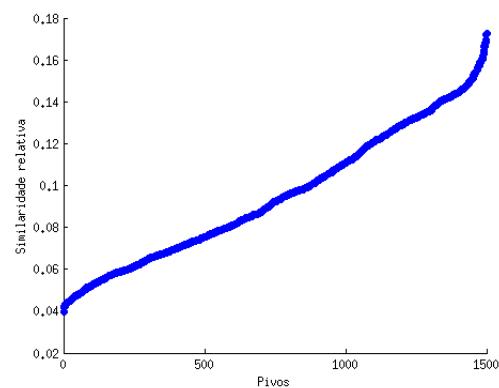


Figura 26 – Similaridade relativa - W_2

Figura 27 – Similaridade relativa - W_3 Figura 28 – Similaridade relativa - W_4 Figura 29 – Similaridade relativa - W_5

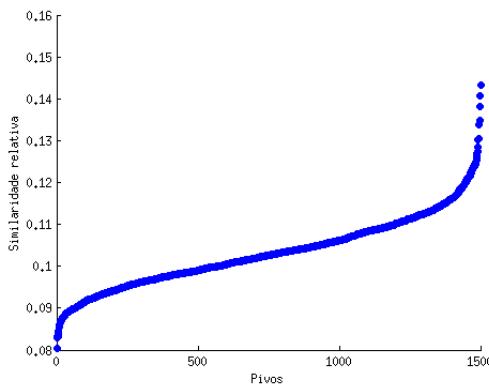


Figura 30 – Similaridade relativa - W_6

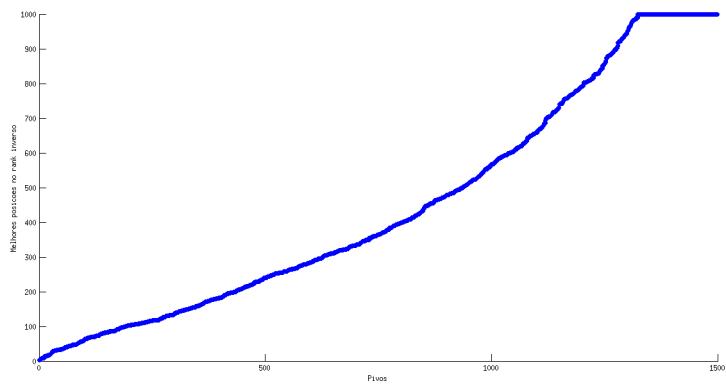
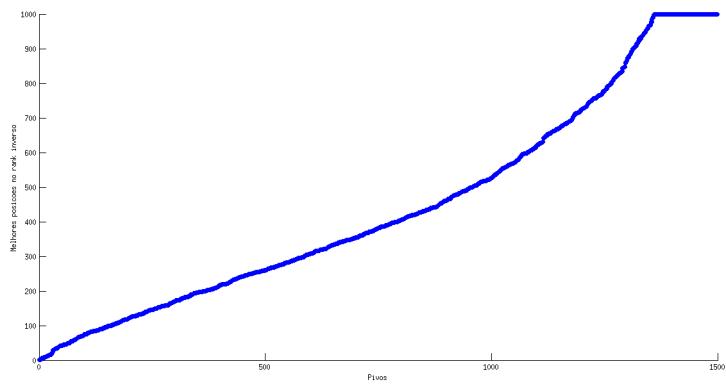
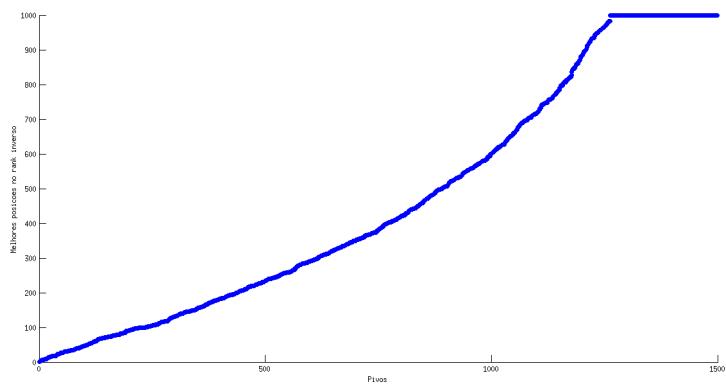
Nessas figuras são mostradas as similaridades relativas, isto é, as maiores similaridades encontradas, depois da normalização dos vetores, para cada um dos pivôs. Era esperado obter apenas similaridades positivas nos gráficos, pois só foi armazenada a de maior valor encontrada para cada um dos pivôs. Quanto aos valores das similaridades em si, não havia uma expectativa, pois estes eram relativos às entradas de vetores normalizados.

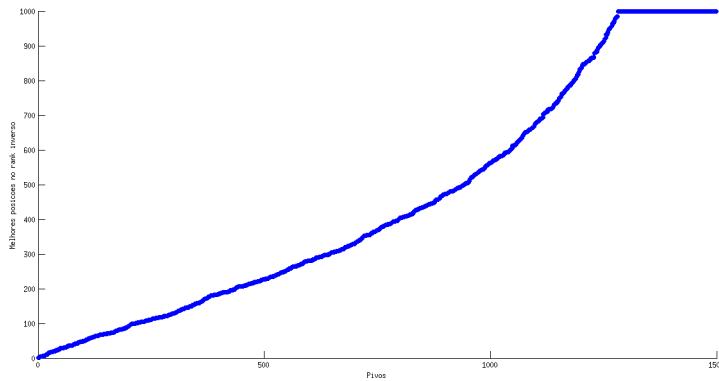
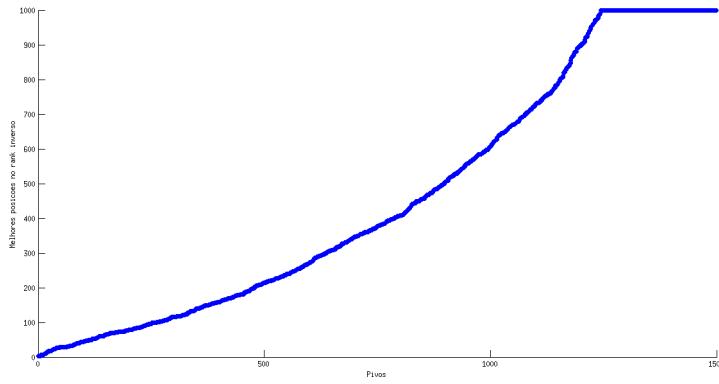
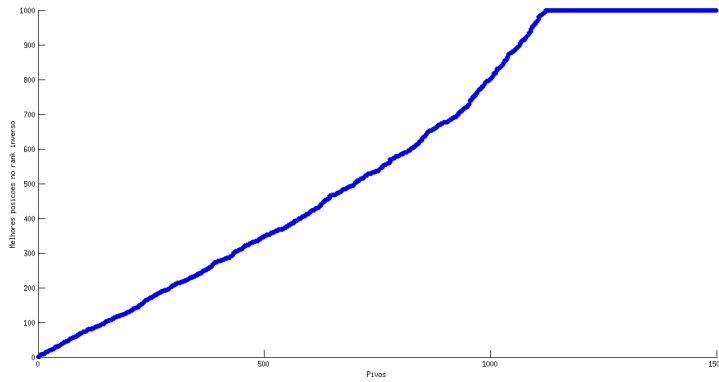
Verificou-se que, as palavras que obtiveram as maiores similaridades relativas, em cada uma das matrizes W , eram distintas das palavras alvos.

4.9 Tratamento de hubs: Inverter a ordem da fila

Esse teste originalmente propõe inverter a ordem fila, ou seja, ao invés de olharmos que palavras aparecem mais alto no rank de um determinado pivô, olharemos em que palavra esse pivô aparece em uma posição mais alta. Não é possível verificar com a função rank em qual posição um pivô aparece, pois os pivôs não estão associados a uma palavra no espaço. Porém, é possível estimar em que posição ele estaria, ao comparar o cosseno entre o pivô e a palavra alvo e os cossenos das palavras que estão no rank.

É inviável verificar para todas as palavras do vocabulário, pois essa verificação seria muito custosa. Logo esse teste foi adaptado da seguinte forma: iremos gerar o rank-100 para um determinado pivô, e para cada uma desses 100 vizinhos iremos gerar o rank-1000 e iremos verificar se o pivô apareceria na lista e em qual posição.

Figura 31 – Rank invertido - W_1 Figura 32 – Rank invertido - W_2 Figura 33 – Rank invertido - W_3

Figura 34 – Rank invertido - W_4 Figura 35 – Rank invertido - W_5 Figura 36 – Rank invertido - W_6

Nas 6 matrizes W obtivemos alguns pivôs considerados na posição 1000. A posição 1000 representa os pivôs não encontrados no rank. Em W_1 obtivemos 176 pivôs nessa posição, em W_2 foram 141, em W_3 foram 237, em W_4 foram 219, em W_5 foram 257 e em W_6 foram 379. A quantidade de pivôs encontrados foi bem maior em W_6 do que nas demais

matrizes, e a que obteve menos foi W_2 . Com exceção de W_6 , que apresenta um crescimento bem rápido com relação às posições, as demais matrizes têm menos da metade dos pivôs em posição abaixo de 500.

4.10 Teste das distâncias

Nesse teste podemos comparar a alterações sofridas nas distâncias entre os pivôs e as palavras alvos em 3 casos:

- 1. Com os pivôs calculados com as matrizes de tradução e os vetores alvos
- 2. Com as palavras obtidas pelo teste 1 proposto por Dinu e os vetores alvos
- 3. Com as palavras obtidas pelo teste 2 proposto por Dinu e os vetores alvos

Foram testadas duas medidas de distância. A primeira delas foi a *edit distance*, que é uma maneira de quantificar quão similar morfologicamente são duas palavras. Nesse método é contado o número mínimo de operações necessárias para transformar uma palavra em outra, como por exemplo, de *pato* para *gato* é necessário trocar apenas uma letra, então é contado como 1 de distância. As operações são inserção, exclusão e substituição de letras. A segunda medida de distância é a norma eucliana.

4.10.1 *Edit distance*

Nas três figuras abaixo temos os histogramas do método *edit distance*. Cada uma das colunas de uma cor corresponde a uma das matrizes.

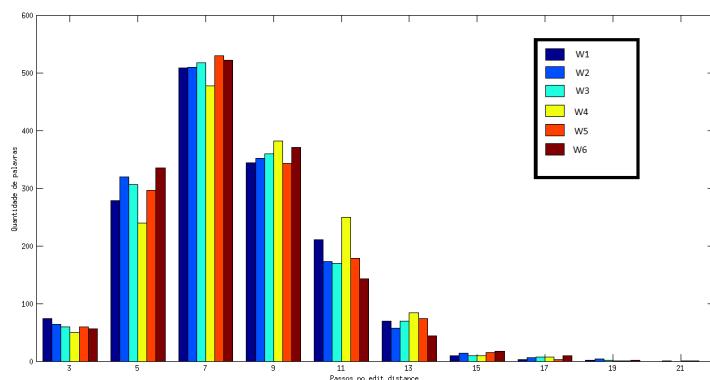


Figura 37 – Histograma das distâncias de palavras, usando o método *edit distance*, entre pivôs gerados por cada uma das 6 matrizes W e as palavras alvos correspondentes.

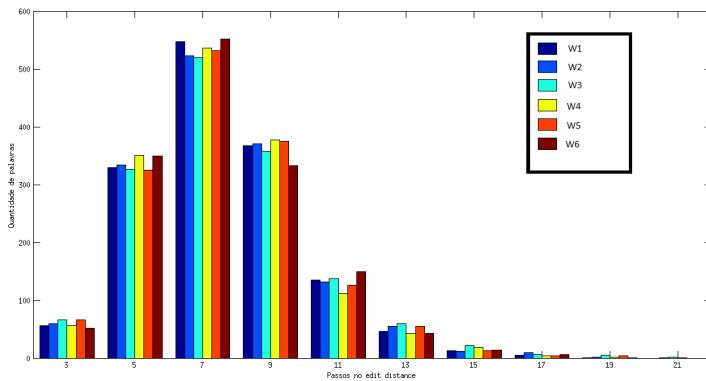


Figura 38 – Histograma das distâncias de palavras, usando o método *edit distance*, entre palavras geradas utilizando o primeiro método de Dinu e as palavras alvos correspondentes.

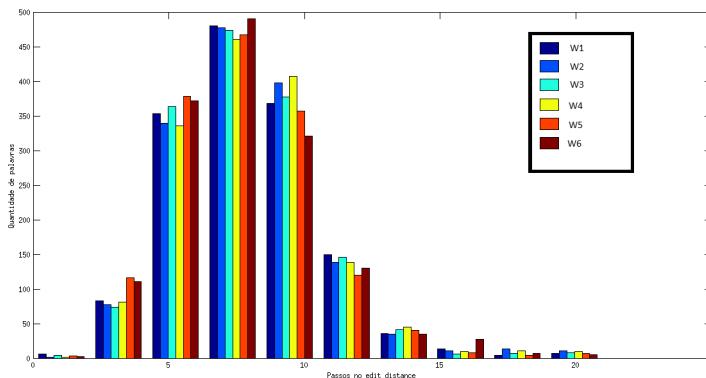


Figura 39 – Histograma das distâncias de palavras, usando o método *edit distance*, entre palavras geradas utilizando o segundo método de Dinu e as palavras alvos correspondentes.

Nos 3 testes obtivemos resultados similares. Na maioria dos casos, para todas as matrizes W , o *edit distance* encontrado foi 7, e a partir de 13 passavam a não serem muito comuns. E em pouquíssimos casos o *edit distance* foi um valor bem baixo, como menor ou igual a 2.

4.10.2 Norma euclidiana

Como as matrizes W tem gráficos similares, a seguir serão mostradas as normas euclidianas nos 3 casos apenas para W_1 .

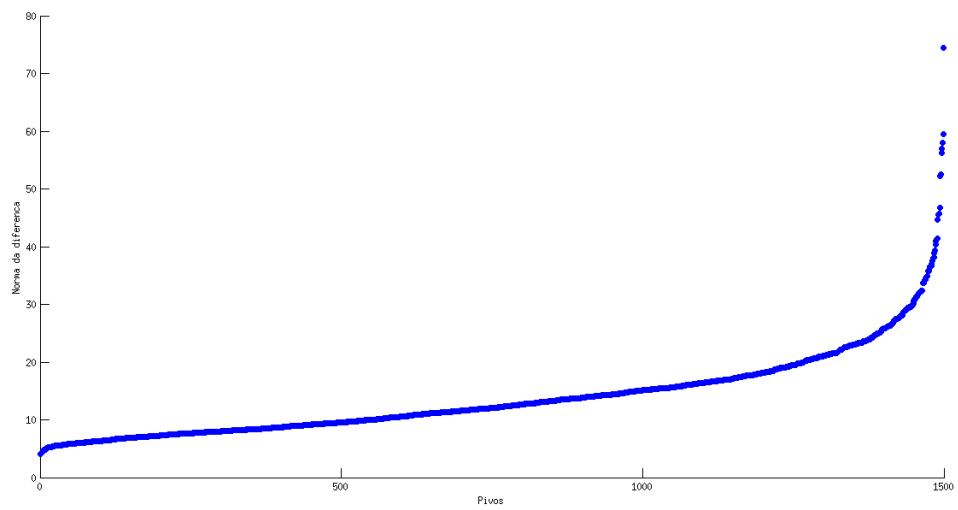


Figura 40 – Norma euclidiana entre os pivôs gerados com W_1 e os vetores alvos.

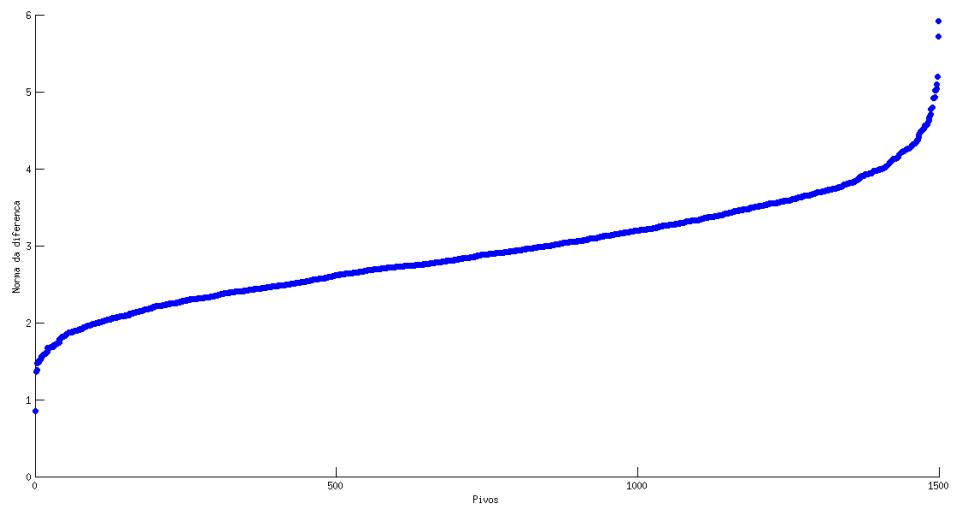


Figura 41 – Norma euclidiana entre os vetores gerados com as palavras obtidas no teste 1 de Dinu com W_1 e os vetores alvos.

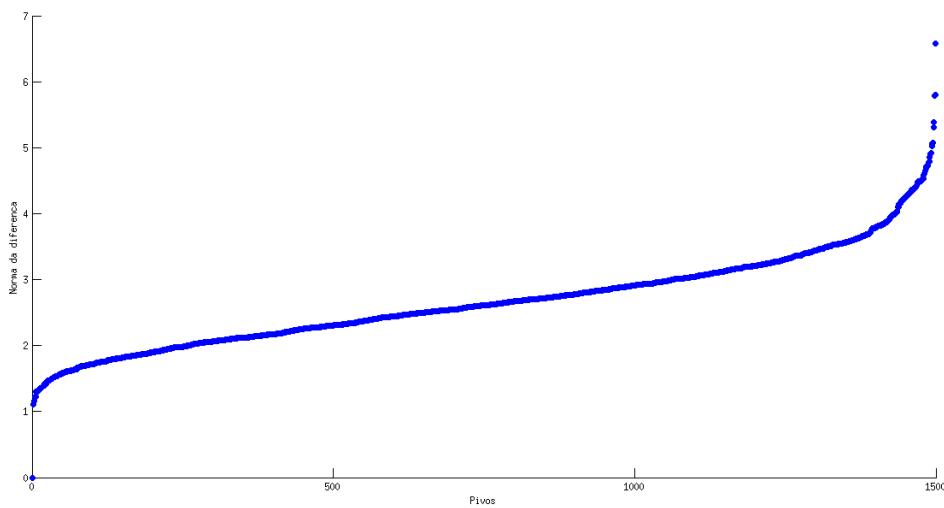


Figura 42 – Norma euclidiana entre os vetores gerados com as palavras obtidas no teste 2 de Dinu com W_1 e os vetores alvos.

No primeiro gráfico o crescimento a princípio apresentou um comportamento linear e no final exponencial, mais de dois terços dos pivôs teve uma norma menor que 20. No segundo gráfico a maioria dos pivôs apresentou uma norma entre 2 e 4, e uma parcela menor obteve a norma entre 4 e 6. O terceiro gráfico foi parecido com o segundo, com a norma indo até 7 ao invés de 6.

O caso ideal seria obter três gráficos parecidos e na mesma ordem de grandeza. Porém, como já vimos que os valores das similaridades entre os pivôs e os alvos não eram muito altos, era de se esperar que os vetores não estivessem tão próximos, e portanto, apresentassem uma norma maior que nos demais casos.

5 Conclusão

O presente trabalho propôs um modelo de tradução automática do Português para o Inglês, utilizando o *word2vec* e o gradiente descendente estocástico. A área de tradução automática é uma área que apresenta grandes desafios, pois é extremamente complexo capturar sutilezas de línguas distintas. Ao retirar preposições e ao analisar palavras fora de seu contexto dificilmente teremos boas traduções, além de ser complicada a escolha de sinônimos por não existirem sinônimos absolutos.

O modelo foi baseado no trabalho de Mikolov, com a variante do estudo na exploração da existência de hubs no experimento, medindo-os e testando métodos para suavizá-los. A principal contribuição deste trabalho é a reprodução deste experimento utilizando a língua portuguesa, abrindo assim um caminho com um grande potencial da tradução automática. O trabalho mostrou limitações ao tentarmos a mesma técnica em Português, e seus resultados apontam para possíveis continuidades para o trabalho.

No teste utilizando a técnica *PCA*, foi possível verificar que existe uma similaridade no arranjo geométrico nos dois espaços, portanto há muito o que explorar. O modelo do Inglês obteve, em geral, um melhor resultado do que o Português no teste de correspondência, mas o Português apresentou um melhor resultado nas relações de palavras. Foi possível averiguar que diferentes abordagens na geração da matriz de tradução obtiveram comportamentos similares, mesmo com diferença significativa no erro absoluto. O trabalho demonstrou ter bastante potencial de aprimoramento, com diversos testes para análise.

5.1 Trabalhos futuros

No decorrer do experimento alguns pontos foram notados como possíveis ramificações, e ficam como sugestões para trabalhos futuros.

Corpus maior

Os dumps da Wikipédia geraram 513k tokens para a língua inglesa e 447k para a língua portuguesa. Seria desejável ter um corpus maior para ter mais tokens de treino. Além disso um dicionário maior poderia ser interessante. O dicionário utilizado por Mikolov continha 5K palavras para treino, o deste experimento continha 3K.

Uma opção desejável também é fazer o experimento do Inglês para o Português e comparar o desempenho das matrizes nas duas direções (PT -> EN e EN -> PT).

Mais limpeza

Antes de rodar o *word2vec* é necessário fazer várias limpezas no Corpus, e existem algumas que seriam aconselháveis acrescentar. Uma é a remoção de frases estrangeiras. Diversas vezes no corpus em Português foram encontradas palavras ou frases em Inglês, Francês, Espanhol e outras línguas. Algumas palavras de fato foram incorporadas na nossa língua, e são empréstimos linguísticos, como *web* por exemplo. Mas quanto as que não o são, seria desejável retirá-las.

Outra mudança é reescrever valores numéricos como um único token após a retirada das stopwords. Dessa forma números escritos por extenso não serão considerados palavras diferentes. Como por exemplo o número 31 por extenso fica **trinta um** (o ‘e’ é uma stopword e por isso já foi retirado) e na divisão de tokens viram dois tokens distintos o ‘trinta’ e o ‘um’. O desejável é que ele vire um token só: ‘**trinta_um**’

Deve-se ter o cuidado de avaliar que combinações numéricas são válidas para não considerar números como **dois_três** e **três_dez** ao tentar realizar esse passo de forma automática.

Corpus anotado

No *word2vec* em Inglês a palavra **book** poderia ter uma representação como verbo (de reservar) e uma representação como substantivo (de livro), e podemos imaginar que elas teriam uma representação distintas por se tratarem de classes gramaticais distintas. No Português o mesmo ocorreria como, por exemplo, com **casa**, pois temos **casa** como substantivo e **casa** como verbo.

Seria aconselhável realizar esse experimento utilizando corpus anotados, pois dessa forma diversas palavras polissêmicas poderiam ser tratadas de forma distintas. Como foi mencionado anteriormente, na criação do dicionário, um dos desafios era a escolha de uma tradução adequada, em especial para palavras polissêmicas, pois a escolha dependia de fixar um dos significados e ignorar os demais. Com as palavras já sendo diferenciadas por suas classes, a escolha das traduções se torna mais simplificada. Porém, em alguns casos, ainda haverá problema, como por exemplo a palavra **planta** pode ser do verbo plantar, mas também tem dois significados distintos como substantivo: o de botânica e o de arquitetura.

Uma possibilidade também é acrescentar nos testes mais vetores, utilizando as demais possíveis traduções. Assim poderiam ser testados diferentes vetores como pivôs.

Utilização de outros métodos

O gradiente descendente estocástico foi escolhido como método para o cálculo da

matriz de tradução por ter sido mencionado como o método utilizado por Mikolov em seu trabalho. Porém existem outros métodos que podem ser significativos para gerar essa matriz como o *gradient descent* e o *conjugate gradient*.

Comparação com outra abordagem

Em um trabalho feito por Garcia [8] é apresentado uma outra abordagem de tradução automática utilizando o word2vec. Ao invés de treinar dois modelos de línguas distintas separadamente, ele treina um modelo com um corpus bilíngue. Neste trabalho é construído um modelo de representação de palavras bilíngue baseado em uma corpora paralela, isto é, um conjunto de corpus bilíngues. Uma ideia de trabalho interessante é comparar os resultados dessas duas abordagens: a utilizada neste trabalho com a utilizada por Garcia.

Referências

- [1] Dean Allemang and James Hendler. Semantic web for the working ontologist: effective modeling in RDFS and OWL. Elsevier, 2011.
- [2] Evandro Dalbem Lopes. Utilização do modelo skip-gram para representação distribuída de palavras no projeto Media Cloud Brasil. Fundação Getulio Vargas, Rio de Janeiro 2015.
- [3] Georgiana Dinu, Angeliki Lazaridou, and Marco Baroni.. Improving zero-shot learning by mitigating the hubness problem. arXiv preprint arXiv:1412.6568, 2014.
- [4] Harris, Zellig S. Distributional structure. Word 10.2-3 (1954): 146-162. 1954
- [5] Hutchins, W. John. Machine translation: A brief history. Concise history of the language sciences: from the Sumerians to the cognitivists (1995): 431-445, 1995
- [6] Karen Sparck Jones. Natural language processing: a historical review. Current issues in computational linguistics: in honour of Don Walker, pages 3–16. Springer, 1994.
- [7] Luiz Antônio Lopes Mesquita. Sintagmas nominais na indexação automática: uma análise estrutural da distribuição de termos relevantes em teses de doutorado da UFMG Universidade Federal de Minas Gerais, Minas Gerais, 2012.
- [8] Martínez Garcia, Eva, et al. Word’s vector representations meet machine translation. Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8): EMNLP 2014, SIGMT, SIGLEX Workshop: 25 Oct 2014, Doha, Qatar. Association for Computational Linguistics, 2014
- [9] Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. Linguistic regularities in sparse and explicit word representations. CoNLL-2014, page 171, 2014.
- [10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In arXiv preprint arXiv:1301.3781, 2013
- [11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013
- [12] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In HLT-NAACL, pages 746–751, 2013
- [13] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation arXiv preprint arXiv:1309.4168, 2013.

-
- [14] Milos Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. On the existence of obstinate results in vector space models. In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, pages 186–193. ACM, 2010
 - [15] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. Cognitive modeling, v. 5, n. 3, p. 1, 1988.
 - [16] Tomašev, Nenad, et al. A probabilistic approach to nearest-neighbor classification: Naive hubness bayesian kNN, Proc. 20th ACM Int. Conf. on Information and Knowledge Management (CIKM). 2011.