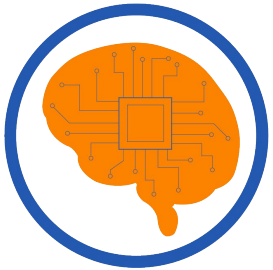


Word2Vec

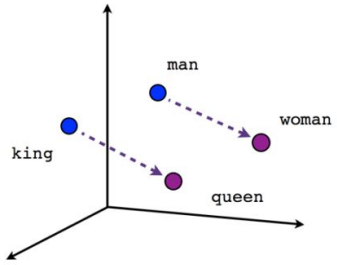
Overview Inicial

Overview

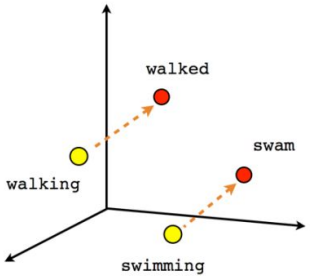
- Questão principal → como representar palavras em vetores, para processamento em modelos?
  - One-hot vectors (?)
  - Bag of Words (?)
- Problema: capturar significado entre palavras nos vetores
  - Como?
    - Contexto - o significado de uma palavra está intimamente relacionado às palavras que, em geral, aparecem com esta.

Overview

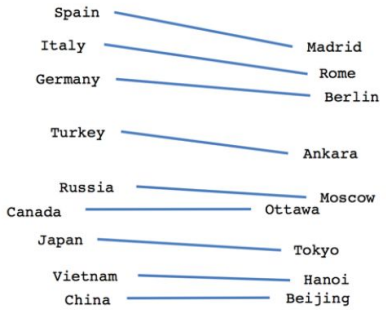
- Tarefa artificial: complete o puzzle: “eu fui ao ___ e comprei abacaxi” ou me diga quais palavras apareceriam junto com “supermercado”.
- Tarefa “self-supervised”, já que os labels estão contidos no próprio corpus base, basta escolher a frase e as palavras de contexto e palavra alvo.
- Capacidade de gerar vetores com valor semântico



Male-Female



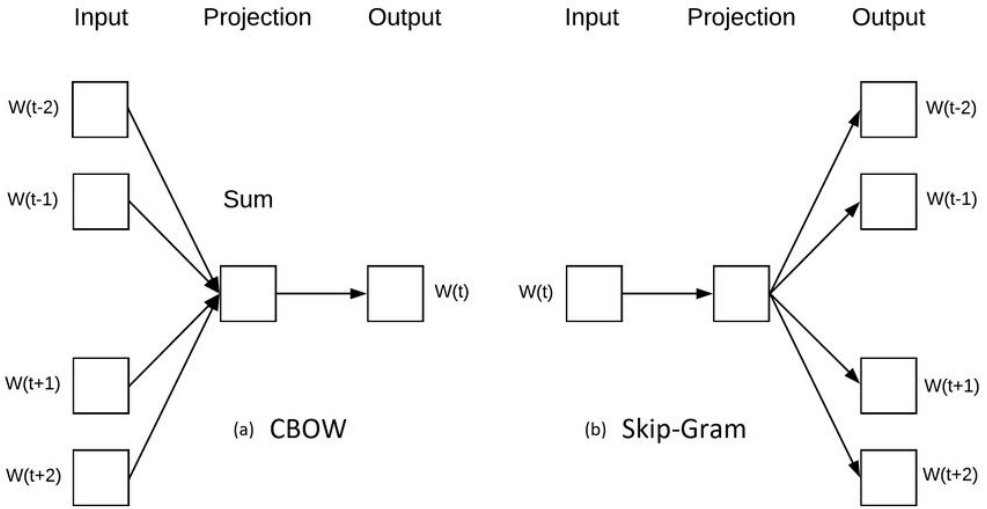
Verb tense



Country-Capital

Overview

- Dois modelos (CBOW vs Skip-Gram)
  - complete o puzzle: “eu fui ao ___ e comprei abacaxi” - CBOW
  - me diga quais palavras apareceriam junto com “supermercado” - Skip-gram



Similaridade entre vetores

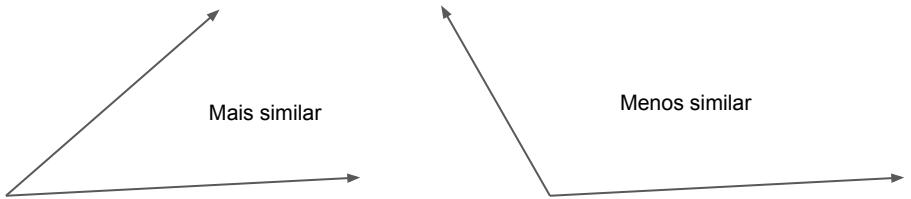
A semântica e os vetores

	Rainha	Rei
Gênero	-0.95	0.789
Realeza	0.89	0.96
	...	...
Fruta	0.015	-0.05
Violência	0.56	0.8

Em geral, cada componente do vetor corresponde a uma dimensão semântica, como “gênero” , “fruta”, “realeza”

Similaridade de vetores de palavras

- Existem várias formas de medir similaridade ou dissimilaridade.
- A distância euclidiana é uma forma de medir dissimilaridade (quanto maior o valor mais dissimilar), ou seja:
  - O vetor (1,0) é mais similar ao vetor (2,0) do que ao vetor (2,1), já que as distâncias são 1 e raiz de 2.
- Para vetores de palavras é mais interessante saber o ângulo entre dois vetores do que a distância euclidiana entre eles. Já que são as direções que contém significado

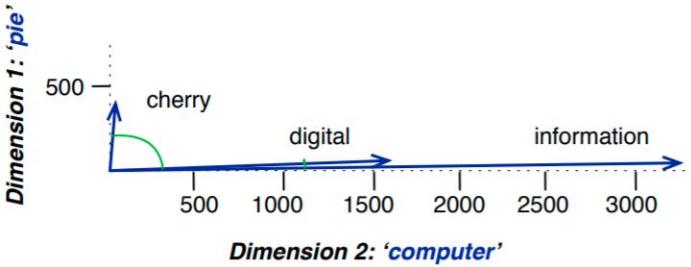


Similaridade de cossenos

- A similaridade de cossenos é um “produto interno normalizado”. Ela permite calcular o cosseno do ângulo entre dois vetores
  - Esse valor varia entre -1 e 1, sendo 0 se o ângulo for de 90°

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

```
def cossine_similarity(x,y):
    return np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))
```



https://web.stanford.edu/~jurafsky/slp3/6.pdf

Avaliação de Embeddings

Avaliação intrínseca vs extrínseca

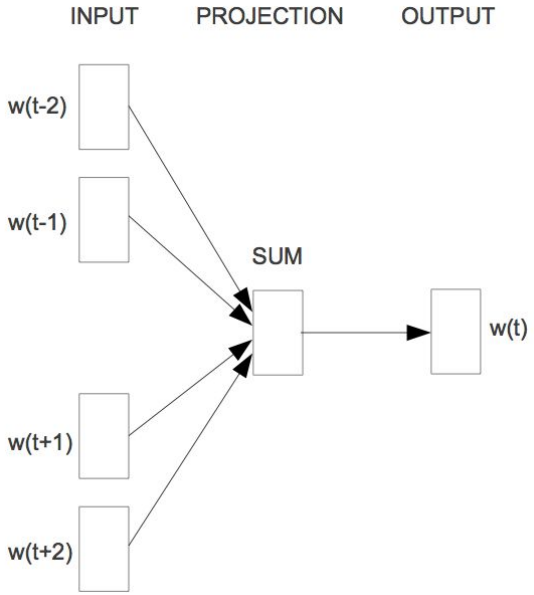
- Intrínseca: analogias, similaridades, plots para ver se os embeddings capturaram informações semânticas do jeito esperado
- Extrínseca: Verificar se os embeddings melhoram a performance na tarefa de interesse

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

https://arxiv.org/pdf/1301.3781.pdf

CBOW

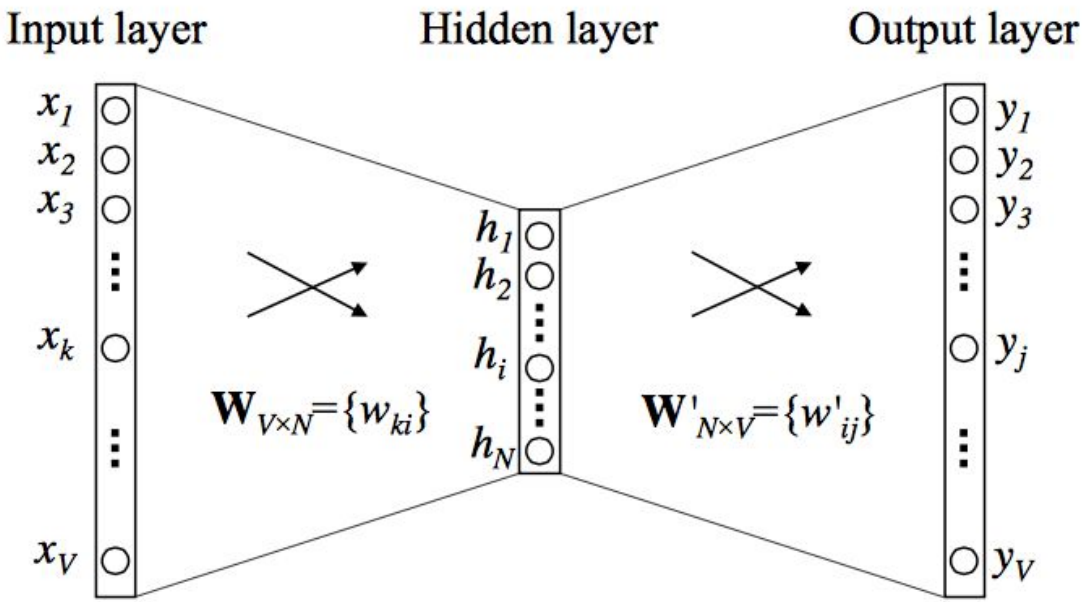
Ideia do CBOW (Continuous Bag of Words)



Obter os embeddings treinando uma rede para prever uma palavra central recebendo palavras de contexto em uma janela C ao redor dela.

Ex: Com a frase “No princípio criou Deus os céus e a terra”, se tomarmos C=2 e a palavra central “Deus”, o contexto seria [‘princípio’, ‘criou’, ‘os’, ‘céus’].

Arquitetura original



V → tamanho do vocabulário

N → tamanho do embedding (hiperparâmetro)

Input → vetor **x** de tamanho V correspondente a média dos vetores one-hot das palavras de contexto

Output → vetor **ŷ** de tamanho V com a probabilidade de cada palavra do vocabulário ser a palavra central

Input

O vetor one-hot é um vetor cujos elementos são todos 0, com exceção de um com valor 1, representando uma única palavra do vocabulário. Vamos voltar ao exemplo “No princípio criou Deus os céus e a terra” (V = 9), com C=2, palavra central “Deus” e contexto [‘princípio’, ‘criou’, ‘os’, ‘céus’].

princípio → [0,1,0,0,0,0,0,0,0]

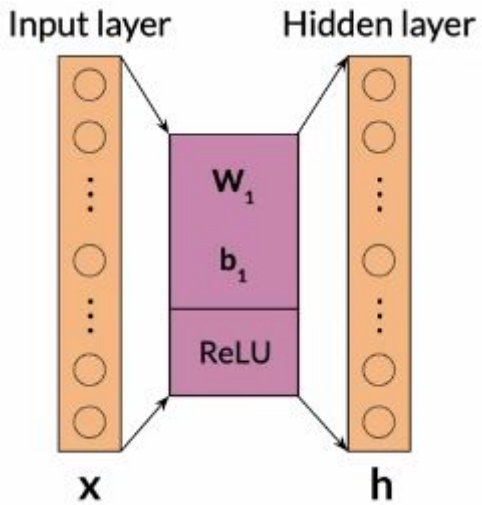
criou → [0,0,1,0,0,0,0,0,0]

os → [0,0,0,0,1,0,0,0,0]

céus → [0,0,0,0,0,1,0,0,0]

média dos vetores (input):
[0,¼,¼,0,¼,¼,0,0,0]

Hidden Layer



Obtemos o vetor **h** da hidden layer por:

$$z_1 = W_1 x + b_1$$

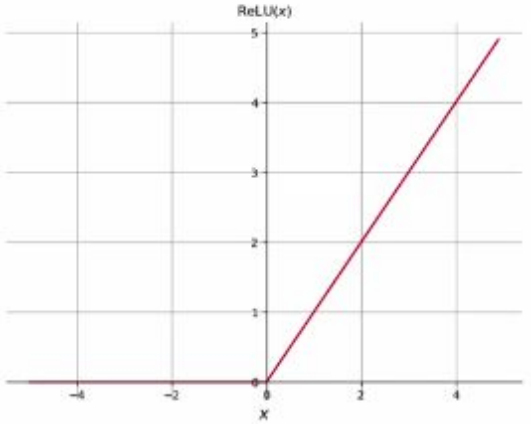
$$h = \text{ReLU}(z_1)$$

Sendo W1 a matriz de pesos NxV e b1 um vetor bias Nx1.

A multiplicação de uma matriz NxV e um vetor Vx1 resulta em um vetor Nx1.

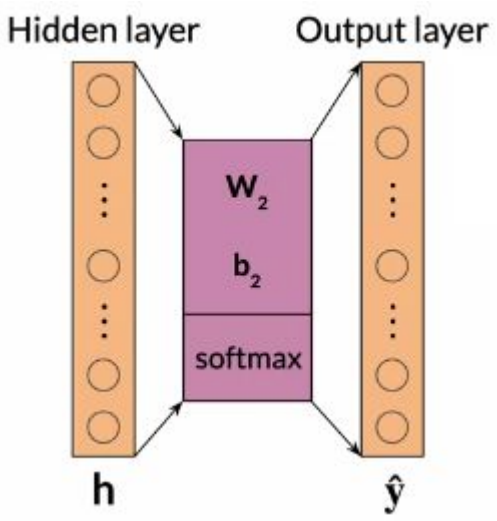
Assim, z1 e h serão vetores Nx1.

$$\text{ReLU}(x) = \max(0, x)$$



Obs: Na arquitetura original, não se usa uma função de ativação na hidden layer! Ou seja, não tem a ReLU e h=z1.

Output



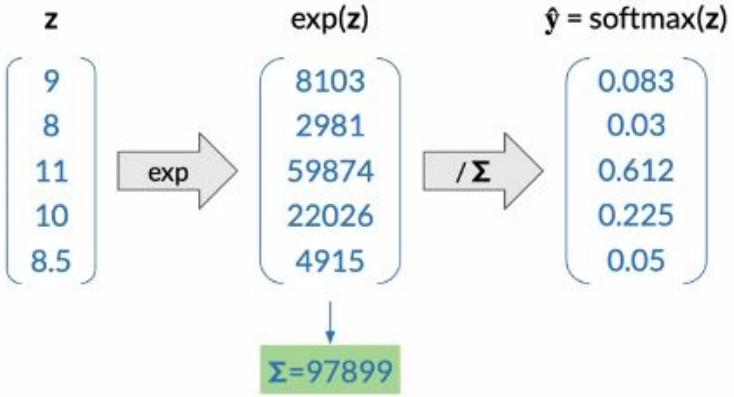
Obtemos o vetor **ŷ** por:

$$z = W_2 h + b_2$$

$$\hat{y} = \text{softmax}(z)$$

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Sendo W2 uma matriz VxN e b2 o vetor bias Vx1.



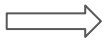
Assim, a soma de todos os valores de ŷ é 1 e cada valor y_i está entre 0 e 1, representando a probabilidade de cada palavra ser a palavra central.

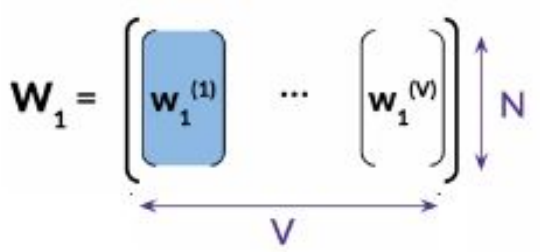
Embeddings

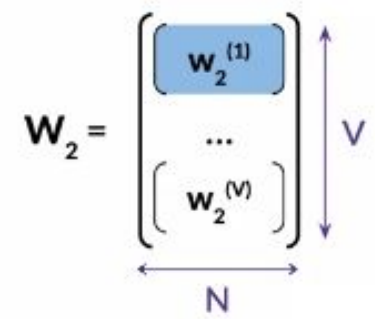
Algumas opções:

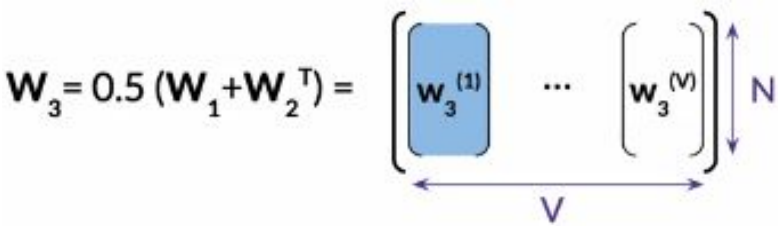
$x =$

1	no
0	princípio
0	criou
0	deus
0	os
0	céus
0	e
0	a
0	terra



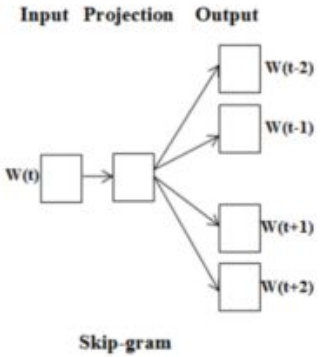






Skip-Gram

Ideia do skip gram



Input: 1 palavra

Output: probabilidade para cada palavra do vocabulário de fazer parte da janela do input (=contexto)

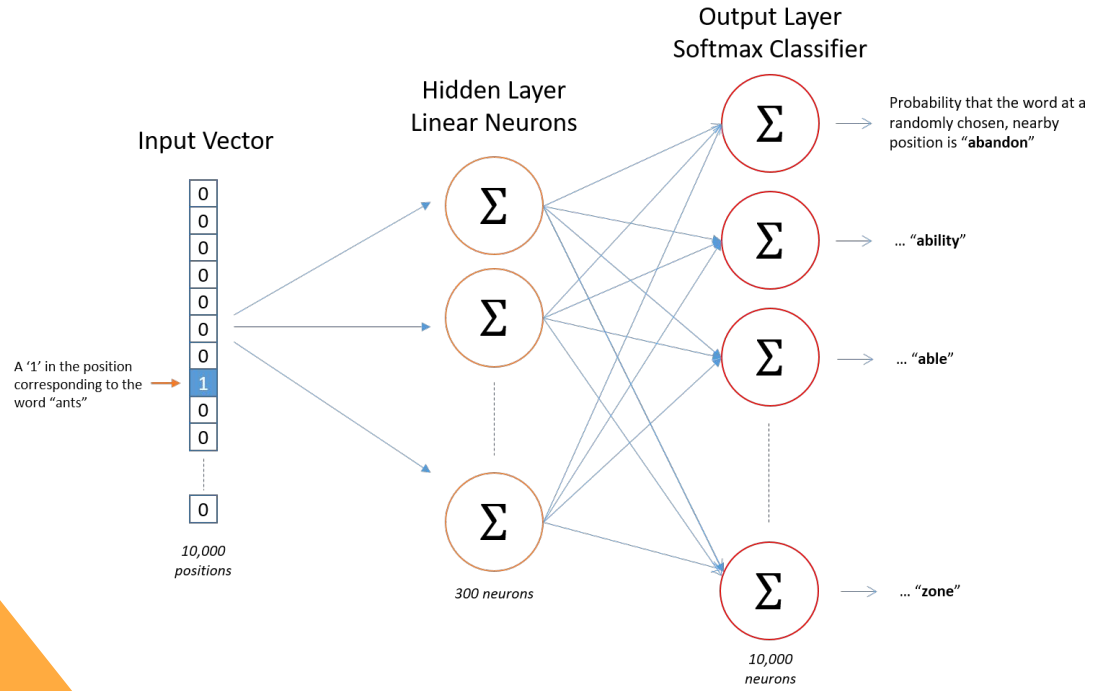
Ex: Se a rede receber como input a palavra "Estados", o output de probabilidades será muito maior para palavras como "Unidos" do que para palavras como "uva" e "jabuti".

Thou shalt not make a machine in the likeness of a human mind

input word	target word
not	thou
not	shalt
not	make
not	a

A rede aprende as estatísticas de acordo com o número de vezes que cada par aparece. Ou seja, para o nosso exemplo, o par ("Estados", "Unidos") apareceria muito mais vezes do que ("Estados", "jabuti").

Arquitetura original



Input → one-hot vector da palavra central

Output → 1 vetor do mesmo tamanho do vocabulário contendo a probabilidade de cada palavra do nosso vocabulário ser selecionada aleatoriamente.

Input

abelha, amora, jabuti, morango, zumbi (5 palavras)

Vetor one-hot para "abelha" → [1, 0, 0, 0, 0]

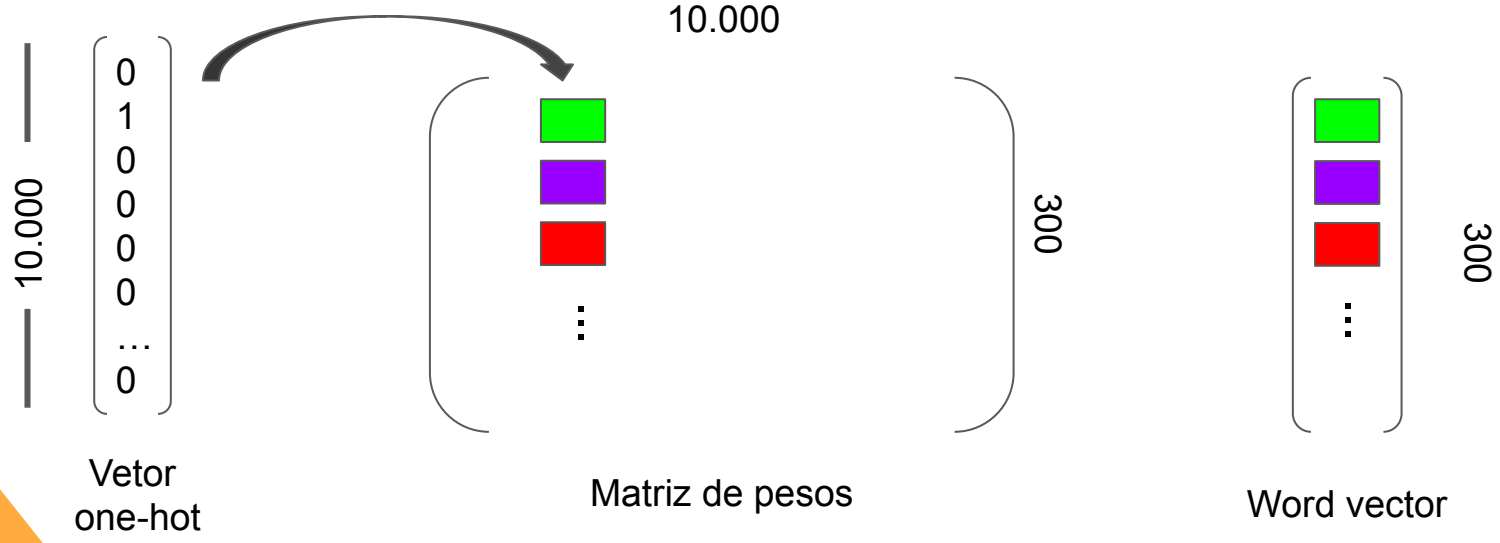
Vetor one-hot para "zumbi" → [0, 0, 0, 0, 1]

*para o resto do exemplo, imagine que estamos trabalhando com um vocabulário de 10.000 palavras

Hidden Layer

Nesse exemplo, aprenderemos word vectors com 300 features (esse número é um hiperparâmetro).

A hidden layer é representada por uma matriz de pesos composta por 300 linhas (1 para cada neurônio dessa camada) X 10.000 colunas (1 para cada palavra do vocabulário).



Hidden Layer

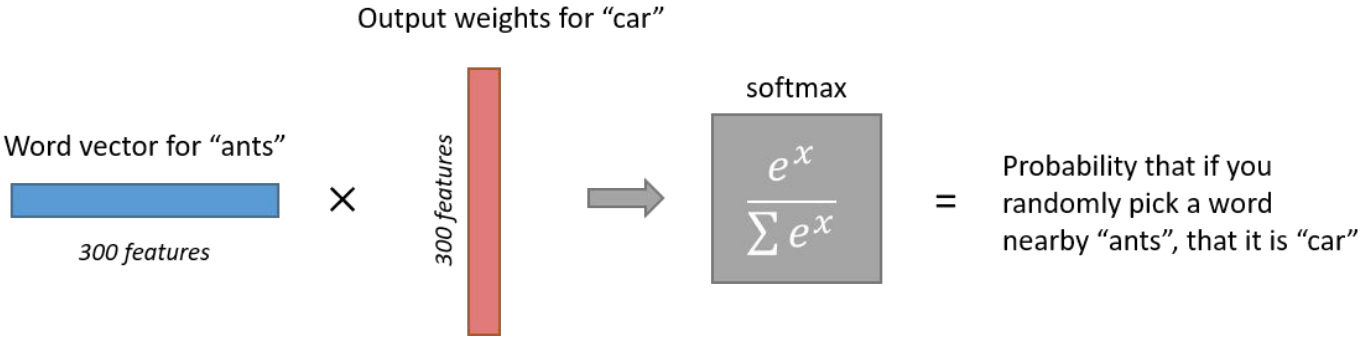
abelha, amora, jabuti, morango, zumbi (5 palavras)

Vetor one-hot para "morango" → [0, 0, 0, 1, 0]

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

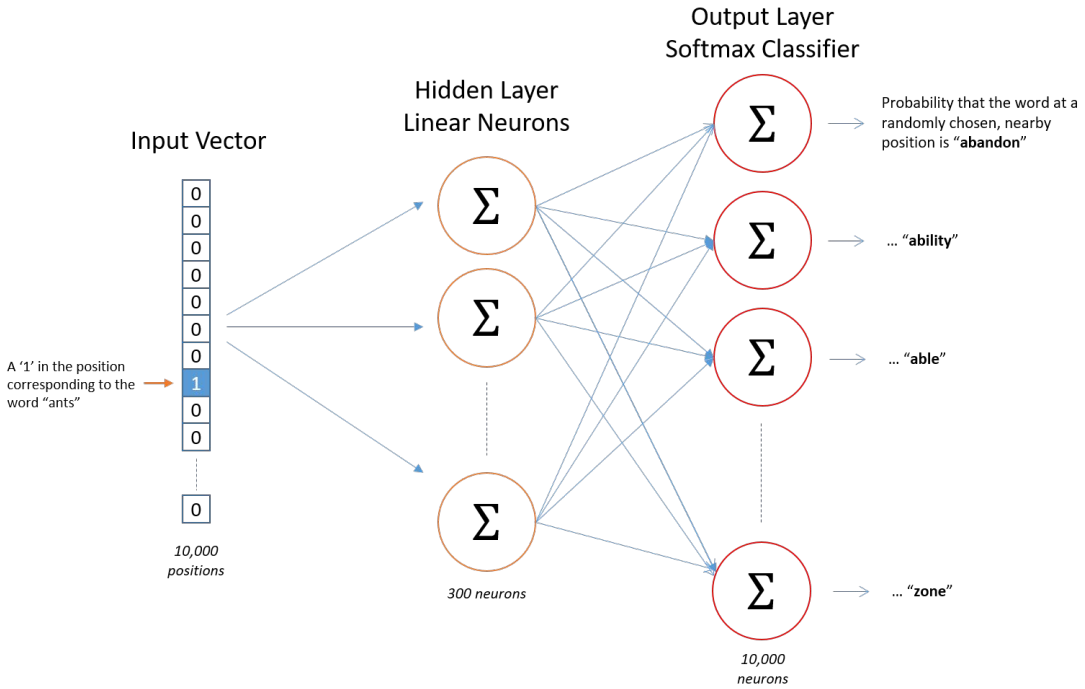
Output Layer

Output layer usa uma softmax. Cada neurônio do output (1 palavra do nosso vocabulário) irá produzir um output entre 0 e 1, e a soma desses output terá soma igual a 1



Para tudo ser somado para 1, dividimos esse resultado pela soma dos resultados de todos os 10.000 outputs

Recapitulando…



one hot vector → embedding → word vector → softmax → $\hat{y}$

Negative Sampling e classificação binária

- Problema: softmax é cara
  - Para vocabulários maiores, o custo computacional é elevado
- Solução: Distributed Negative Sampling e classificação binária
  - Ideia: embeddings palavras que aparecem no mesmo contexto (similares) devem ter um produto interno (dot product) elevado.
  - Nova tarefa do algoritmo: dadas duas palavras, elas foram retiradas do mesmo contexto (janela de palavras)?
    - Para descobrir: calcule o produto interno dos embeddings
    - Passe esse número numa sigmoid
    - Classifique
  - Assim é possível treinar uma regressão logística para fazer tal tarefa e atualizar os embeddings com base na performance na classificação

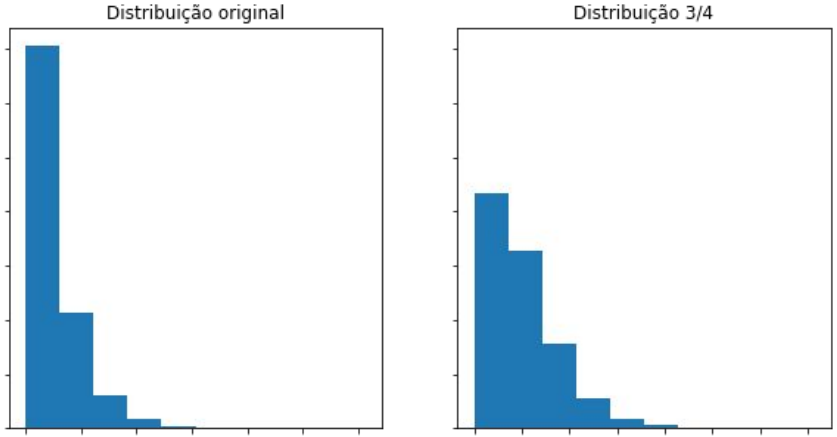
Negative Sampling e classificação binária

- Embeddings:
  - A palavra enquanto contexto tem um embedding diferente da palavra como target
  - No final, vamos ficar apenas com a matriz referente às targets
- Negative Sampling em si
  - Para cada par positivo de palavras do mesmo contexto, vamos gerar k pares falsos
  - Esses pares falsos vão ser obtidos através da seleção de uma palavra aleatória do vocabulário
  - A probabilidade de uma palavra ser selecionada nesse passo tem a ver com a sua frequência, mais especificamente:
    - Com a frequência de uma palavra calculamos sua probabilidade
    - Depois, elevamos todas as probabilidades a ¾ e normalizamos

Negative Sampling e classificação binária

Binomial Classification

(regression, logistic)		(regression, zebra)
(regression, machine)		(regression, pimples)
(regression, sigmoid)	VS	(regression, Gangnam-Style)
(regression, supervised)		(regression, toothpaste)
(regression, neural)		(regression, idiot)
Likely to observe		Unlikely to observe
$p(D = 1\|w, c_{pos}) \approx 1$		$p(D = 1\|w, c_{neg}) \approx 0$



https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling

Negative Sampling e classificação binária

- Função de custo

$$J(\theta; w, c_{pos}) = -log\,\sigma(\bar{c}_{pos} \cdot \bar{w}) - \sum_{c_{neg} \in W_{neg}} log\,\sigma(-\bar{c}_{neg} \cdot \bar{w})$$

- Intuição: termo dentro dos dois logs, idealmente vai pra 1, com isso, o log iria para 0.

Negative Sampling e classificação binária

Algorithm 1 SGNS Word2Vec

function UPDATE(η)
  for all skip-grams w_I, w_O **do**
    $W_{neg} \leftarrow \{\}$
    for $i \in [1, N]$ **do**
      $W_{neg} \leftarrow W_{neg} \cup sample(P_n)$
    $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$ ▷ get gradients
    $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_O}$
    for all $w_j \in W_{neg}$ **do**
      $\mathbf{g}_{w_j} \leftarrow \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}'_{w_I}$
      $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}'_{w_j}$
    $\mathbf{v}'_{w_O} \leftarrow \mathbf{v}'_{w_O} - \eta \cdot \mathbf{g}_{w_O}$ ▷ update vectors
    $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}$
    for all $w_j \in W_n$ **do**
      $\mathbf{v}'_{w_j} \leftarrow \mathbf{v}'_{w_I} - \eta \cdot \mathbf{g}_{w_j}$

https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewFile/14956/14446

Pseudocódigo para a implementação da classificação binária com negative sampling

Referências

- https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewFile/14956/14446
- https://www.geeksforgeeks.org/implement-your-own-word2vecskip-gram-model-in-python/
- http://jalammar.github.io/illustrated-word2vec/
- https://towardsdatascience.com/word2vec-from-scratch-with-numpy-8786ddd49e72
- https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- https://arxiv.org/pdf/1301.3781.pdf
- https://web.stanford.edu/~jurafsky/slp3/6.pdf
- https://ruder.io/word-embeddings-1/index.html#skipgram
- https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling