

Word Embeddings

Víctor Mijangos

Facultad de Ciencias

Lingüística computacional



Representación del significado

En PLN es una tarea complicada lograr que una máquina **entienda** el lenguaje humano.

Se busca **representar el significado** de las unidades lingüísticas, de tal forma que la máquina pueda procesarlos.

Algunas propuestas para realizar esta representación son:

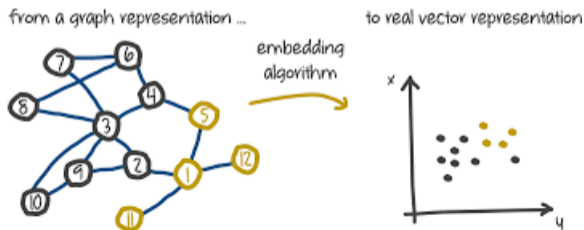
- ▶ Metalenguajes y lenguajes lógicos, como cálculo lambda.
- ▶ Ontologías, representaciones gráficas de las relaciones entre palabras.
- ▶ Taxonomías, representaciones jerárquicas de relaciones entre palabras.

Embedding

Si asumimos que el significado de las unidades lingüísticas está determinado por ciertas operaciones y propiedades, podemos pensar que estas propiedades pueden **embederse** (o incrustarse) en otro tipo de estructuras matemáticas:

Definición (Embedding)

Se dice que una estructura S es embebida en un espacio V si las propiedades algebraicas relevantes de S se preservan en V .



Word embeddings

El objetivo de los llamados **word embeddings** es ‘embeder’ las palabras en espacios vectoriales que representen el significado de estas.

La estructura latente del significado, que no conocemos a priori, va a emerger en el espacio vectorial.

En este espacio podremos determinar operaciones (como la similitud, la composición, analogías, etc.) que desconocemos cómo realizar en el espacio textual original.



Hipótesis distribucional

Las perspectivas clásicas y actuales para representar las palabras en espacios vectoriales se basan en la llamada **hipótesis distribucional**, planteada por Zellig Harris [3, 6]:

Definición (Hipótesis distribucional)

Palabras similares aparecen en contextos similares:

Mi mascota es un gato.

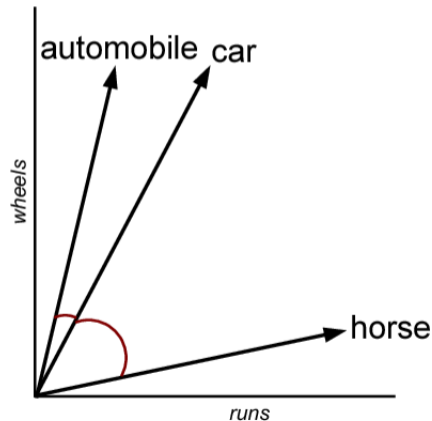
Mi mascota es un perro.

Mi mascota es (·).

Semántica distribucional

La **semántica distribucional** [1] se basa en la hipótesis distribucional. Busca representar las palabras a partir de las **co-ocurrencias** con otras palabras.

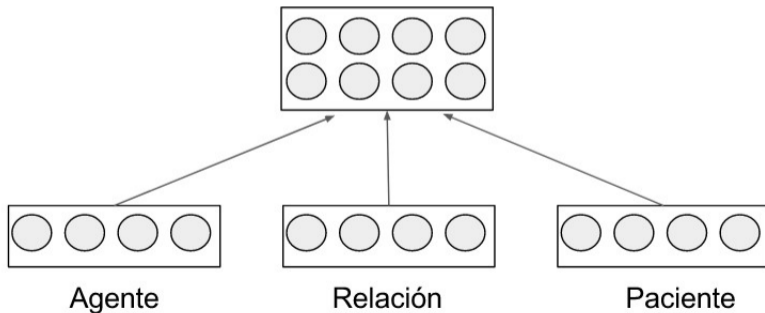
	<i>runs</i>	<i>wheels</i>
automobile	1	4
car	2	4
horse	4	1



Modelos distribuidos

Los modelos de la semántica distribucional representan los conceptos a través de la frecuencia. Pero ya en los 80s se propone representar los conceptos a partir de redes neuronales.

G. Hinton (1986) propone representar los conceptos a partir de los “roles” de las palabras: agente (sujeto) y paciente (objeto) relacionados por una relación (verbo).



Modelos del lenguaje y embeddings

Con la propuesta de Bengio (2003) se observó que dentro de la capa oculta, las representaciones distribuidas de las palabras capturaban nociones sobre el significado.

Sin embargo, el cálculo de estos embeddings era costoso y requería cálculos que no eran necesarios.

Mikolov et al. (2013) [4] propone una arquitectura neuronal enfocada al aprendizaje de word embeddings. A esta arquitectura la llama **word2vec**.

Contextos

Dado que estos modelos se basan en la **hipótesis distribucional**, se basan en la extracción de **contextos**.

El **contexto** de una palabra w_i puede entenderse como las palabras que co-ocurren con ésta. $\mathcal{N}(w_i)$ son elementos w_k , $w_j \in \Sigma$ que aparecen con w_i .

El contexto se puede determinar de distintas formas:

- ▶ Por medio de la presencia de una palabra en una construcción lingüística (oración, párrafo, documento).
- ▶ Por medio de una ventana de palabras.

Contextos

Supóngase el siguiente corpus:

*La presencia humana en **México** se remonta a 14 000 años antes del presente. Como fruto de miles de años de desarrollo cultural surgieron en el territorio mexicano las culturas mesoamericanas, arido-americanas y oasisamericanas. El actual territorio central de **México** fue el principal y mayor escenario del pueblo mexica y, en parte, del pueblo maya, dos de las civilizaciones más importantes de la América precolombina. Durante 300 años, la totalidad del actual territorio formó parte del Virreinato de Nueva España, con capital en la Ciudad de **México**, siendo una de las entidades más importantes del Imperio Español en América.*

Los contextos de «México» con ventanas de 2×2 son:

«humana en _ se remonta a»,

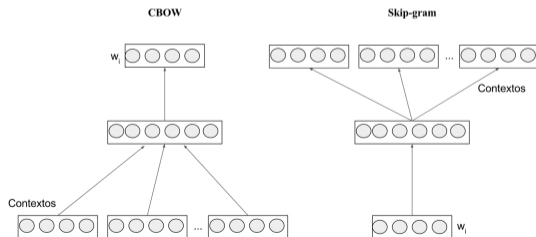
«central de _ fue el»,

«Ciudad de _ siendo una».

CBOW y Skip-gram

En el modelo de word2vec se proponen dos arquitecturas según se prediga el contexto o a partir del contexto:

- ▶ CBOW (Continuous Bag Of Words): la entrada de la red es el contexto y se predice una palabra objetivo $p(w_i|w_c)$.
- ▶ Skip-gram: La entrada de la red es una palabra objetivo y se predice el contexto en el que puede aparecer $p(w_c|w_i)$.



Arquitectura de la red

La arquitectura del modelo de word2vec es una simplificación de la propuesta de Bengio (2013). Esta se compone de:

Entrada: Determinada por los vectores one-hot $x(i) \in \mathbb{R}^N$ para cada palabra w_i .

Capa de embedding: La capa central, determinada por una función lineal:

$$C(i) := Cx(i)$$

donde C es una matriz de $d \times N$ (d hiperparámetro).

Capa de salida: Definida por:

► Pre-activación: Dada la matriz $W \in \mathbb{R}^{N \times d}$ (no hay bias):

$$a(i) := WC(i)$$

Activación: Dada por la función Softmax (asumimos skip-gram):

$$p(w_c | w_i) = \frac{e^{a(i)_c}}{\sum_{k=1}^N e^{a(i)_k}}$$

Arquitectura de la red

La arquitectura de word2vec sólo cuenta con los parámetros:

$$\theta = \{C, W\}$$

y la salida puede verse como un modelo del lenguaje. Sin embargo, el objetivo aquí no es el modelo, sino los embeddings guardados en la matriz C .

La función de salida puede reescribirse como:

$$\begin{aligned} p(w_c | w_i) &= \frac{e^{a(i)_c}}{\sum_{k=1}^N e^{a(i)_k}} \\ &= \frac{e^{W_c \cdot C(i)}}{\sum_{k=1}^N e^{W_k \cdot C(i)}} \end{aligned}$$

Arquitectura de la red

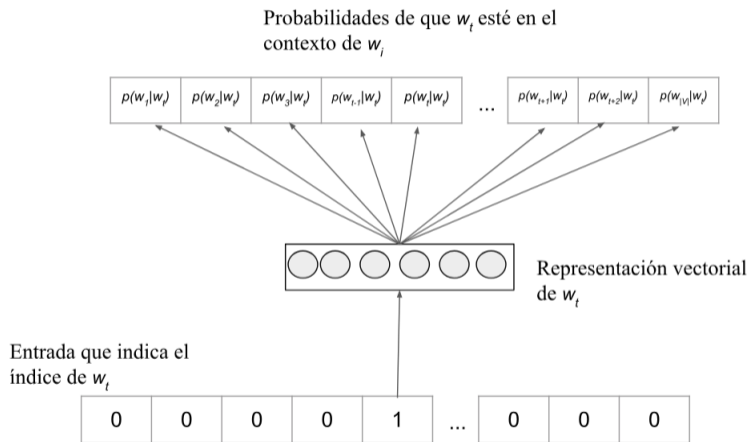


Figura: Arquitectura de word2vec (Skip-gram)

Aprendizaje self-supervised

El aprendizaje se dice que es *self-supervised* (auto-supervisado), pues los pares de entrenamiento son obtenidos automáticamente por medio de ventanas de contexto.

Supóngase el texto:

Y diciendo esto, dio de espuelas a su caballo Rocinante, sin atender las voces que su escudero Sancho le daba...

Si la palabra objetivo es «caballo» en una ventana de 2×2 , su único contexto es: «a su _ rocinante sin».

Los pares de entrenamiento (para el modelo skip-gram) serán:

$(caballo, a), (caballo, su), (caballo, rocinante), (caballo, sin)$

NOTA: Ya que se crearán pares para cada palabra, los pares simétricos (a, caballo), (rocinante, caballo), etc. también estarán en el conjunto de entrenamiento.

Aprendizaje self-supervised

Source Text	Training Samples			
<table border="1"><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. →	The	quick	brown	(the, quick) (the, brown)
The	quick	brown		
The <table border="1"><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. →	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)
quick	brown	fox		
The quick <table border="1"><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. →	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
brown	fox	jumps		
The quick brown <table border="1"><tr><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. →	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
fox	jumps	over		

Conjunto supervisado

El conjunto de entrenamiento, entonces, estará dado por los pares:

$$(w_i, w_c)$$

Al igual que en el modelo de Bengio (2003), la entrada será representada por un one-hot $x(i)$ y la salida, por una clase y_c . El conjunto será:

$$\mathcal{S} = \{(x(i), y_c) : x(i) \in \mathbb{R}^N, y_c \in \{0, 1\}\}$$

Si w_c está en el contexto de w_i entonces $y_c = 1$, de lo contrario su valor será 0.

Retropropagación

La función de riesgo estará dada por la entropía cruzada:

$$R(\theta) = - \sum_i \sum_c y_c \ln p(w_c | w_i)$$

Por tanto, la variable del método de backpropagation en la capa de salida es:

$$\begin{aligned} d_{out}(k) &= \frac{\partial R}{\partial \phi_k} \frac{\phi_k}{a(i)_k} \\ &= \phi_k - y_k \\ &= p(w_k | w_i) - y_k \end{aligned}$$

Retropropagación

La variable para la capa oculta, está determinada como:

$$\begin{aligned}d_c(k) &= \frac{\partial C(i)}{\partial C_k} \sum_q W_{k,q} d_{out}(k) \\ &= \sum_q W_{k,q} (\phi_q - y_q)\end{aligned}$$

Finalmente, las derivadas para cada capa son:

- ▶ Capa de salida:

$$\frac{\partial R}{\partial W_{j,k}} = (\phi_k - y_k) C(i)_j$$

- ▶ Capa de embedding:

$$\frac{\partial R}{\partial C_{j,k}} = \left[\sum_q W_{k,q} (\phi_q - y_q) \right] x(i)_j$$

Algoritmo de Word2Vec

Input: Pares (w_i, w_c) de palabras w_i con contextos w_c ; T iteraciones; rango de aprendizaje η

Inicialización: $C \in \mathbb{R}^{d \times N}$ y $W \in \mathbb{R}^{N \times d}$ se inicializan aleatoriamente.

Entrenamiento: Por T iteraciones y para cada ejemplo $x(i)$ hacer:

1. **Forward:**

$$\phi_c = \frac{e^{W_c \cdot C(i)}}{\sum_k e^{W_k \cdot C(i)}}$$

2. **Backward:**

2.1 Para la capa de salida, tenemos que:

$$W_{j,k} \leftarrow W_{j,k} - \eta d_{out}(k) C(i)_j$$

2.2 Para la capa oculta, tenemos que:

$$C_{j,k} \leftarrow C_{j,k} - \eta d_h(k) x(i)_j$$

Finalización: El algoritmo termina después de las T iteraciones.

Output: Vectores $C(i)$ (las columnas de C).

Generalización de CBOW

El modelo neuronal CBOW puede generalizarse para tomar en cuenta m contextos de entrada.

Dado una cadena de entrada de longitud m , w_c, \dots, w_{c+m-1} , se obtienen los one-hot vectors:

$$x(c), \dots, x(c + m - 1)$$

Y se obtendrá una representación distribuida de estos vectores, dada por:

$$C(c, \dots, c + m - 1) = \frac{1}{m} \sum_{l=0}^{m-1} Cx(c + l)$$

Esta expresión se puede reducir a:

$$C(c, \dots, c + m - 1) = \frac{C}{m} \sum_{l=0}^{m-1} x(c + l)$$

Retropropagación

Puede notarse que la variable de backpropagation en la salida no cambia. En la capa de embedding, por su parte, tenemos:

$$\begin{aligned}d_C(k) &= \frac{\partial C(c, \dots, c + m - 1)}{\partial C_k} \sum_q W_{k,q} d_{out}(q) \\ &= \frac{1}{m} \sum_q W_{k,q} d_{out}(q)\end{aligned}$$

Y en general, la derivada puede verse como:

$$\frac{\partial R}{\partial C_{j,k}} = d_C(k) x(c, \dots, c + m - 1)_j$$

Sub-muestreo de ejemplos negativos

Un par de entrenamiento **positivo** es $(x(i), y_c)$ tal que $y_c = 1$. Es decir, donde w_c es contexto de w_i

Para cada ejemplo en el entrenamiento, la salida tendrá una neurona $y_c = 1$, mientras que las $N - 1$ neuronas restantes serán igual a 0 (ejemplos negativos).

Ya que el interés está puesto en la capa de embedding (y no en la de salida), Mikolov et al. (2013) proponen simplificar el cómputo de las salidas a través de un **sub-muestreo** de ejemplos negativos.

La idea del **negative sampling** es tomar, además del ejemplo positivo, $s < N - 1$ ejemplos negativos, elegidos aleatoriamente a partir del vocabulario restante.

Negative sampling

Con el negative sampling, se obtendrán $s + 1$ salidas, en lugar de N salidas, por lo que el número de cálculos en la capa de salida se reduce.

Para elegir los ejemplos negativos en cada par de entrenamiento, estos se seleccionan a partir de la distribución:

$$q(w_j) := \frac{fr(w_j)^{3/4}}{\sum_{r=1}^N fr(w_r)^{3/4}}$$

Dado que la probabilidad Softmax pierde sentido, se propone que cada neurona salida sea una variable independiente $D \in \{0, 1\}$ (son o no son contextos), y se utiliza la función sigmoide:

$$p(D = 1 | w_i, w_j) = \frac{1}{1 + e^{w_j c(i)}} \quad (1)$$

Retropropagación

La función de riesgo sigue siendo la entropía cruzada; sin embargo, debido a los cambios de la función de salida y el número de neuronas, el backpropagation varía.

La función de riesgo puede escribirse como:

$$\begin{aligned}R(\theta) &= - \sum_{j=1}^{s+1} \mathbb{E}[\ln p(D = y | w_i, w_j)] \\&= - \sum_{j=1}^{s+1} \mathbb{E}[\ln p(D = 1 | w_i, w_j)] - \sum_{j=1}^{s+1} \mathbb{E}[\ln p(D = 0 | w_i, w_j)] \\&= - \ln \sigma(W_c \cdot C(i)) - \sum_{j=1}^s \mathbb{E}[\ln(1 - \sigma(W_j \cdot C(i)))] \\&= - \ln \sigma(W_c \cdot C(i)) - \sum_{j=1}^s \mathbb{E}[\ln \sigma(-W_j \cdot C(i))]\end{aligned}$$

Retropropagación

Cuando derivamos sobre el ejemplo positivo, tenemos que:

$$\begin{aligned}d_{out}(c) &= \frac{\partial R}{\partial \sigma} \frac{\partial \sigma}{\partial a(i)_c} \\ &= (1 - \sigma(W_c \cdot C(i)))\end{aligned}$$

Si la neurona sobre la que derivamos es un ejemplo negativo, por su parte, tenemos que:

$$d_{out}(j) = -\sigma(W_j \cdot C(i))$$

Por lo que podemos escribir la derivada como:

$$d_{out}(k) = y_k - \sigma(W_k \cdot C(i))$$

La derivada de la capa de embedding no cambia más que en que ahora toma esta variable.

Similitud semántica

La similitud entre dos palabras puede entonces trasladarse al cálculo de similitud entre vectores.

Una función de similitud equivale a una función de distancia $d : \mathbb{R}^d \rightarrow \mathbb{R}^+$ en el espacio de embeddings.

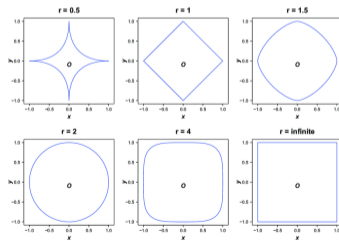
Diferentes métricas y distancias pueden usarse para calcular esta similitud:

- ▶ Distancia coseno: $d(x, y) = \cos(x, y) = \frac{\langle x, y \rangle}{\|x\| \|y\|}$
- ▶ Métrica euclídeana: $d(x, y) = \|x - y\|_2 = \sqrt{\sum_i (x_i - y_i)^2}$
- ▶ Distancia Manhattan (del taxista): $d(x, y) = \|x - y\|_1 = \sum_i |x_i - y_i|$

Métrica Minkowsky

Las últimas dos medidas de similitud son métricas. En general, estas pueden derivarse de la llamada métrica Minkowsky, dada por ($1 \leq p \leq \infty$):

$$d(x, y) = \|x - y\|_p = \left(\sum_i |x_i - y_i|^p \right)^{\frac{1}{p}}$$

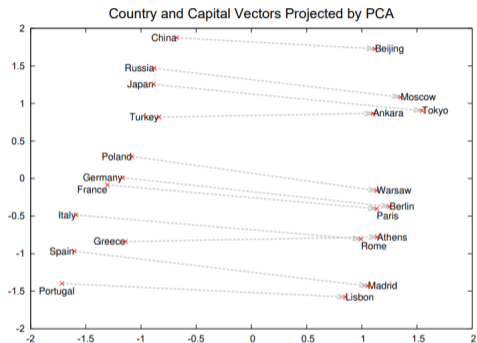


Aplicaciones

Las aplicaciones de los word embeddings son múltiples, algunas son:

- ▶ Como entrada para otras aplicaciones de ML.
- ▶ En búsqueda de palabras similares.
- ▶ Para realizar analogías: Una analogía puede establecerse a partir de resolver la ecuación:
$$C(i) - C(j) = C(m) - C(k)$$
- ▶ Para representación de unidades lingüísticas mayores, como frases, oraciones, etc.

Analogías



La analogía es encontrar la palabra (correspondiente al vector) x que cumpla:

$$\arg \min_x d(C(i) - C(j) + C(m), x)$$

Composicionalidad

La **composicionalidad** asume que el significado de unidades complejas del lenguaje (como frase, oraciones, documentos) es la suma de los significados de sus elementos componenciales.

Si μ es una función de significado, entonces tenemos que:

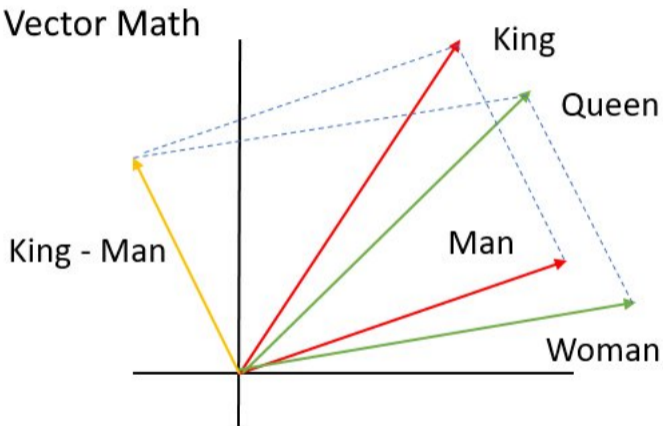
$$\mu(w_1 \cdot w_2) = \mu(w_1) + \mu(w_2)$$

De esta forma, si asumimos que el significado de las palabras está representado por los embeddings, el significado de una unidad compleja será la suma de los embeddings de sus componentes:

$$C(i, j) = C(i) + C(j)$$

Composicionalidad

Vector Math

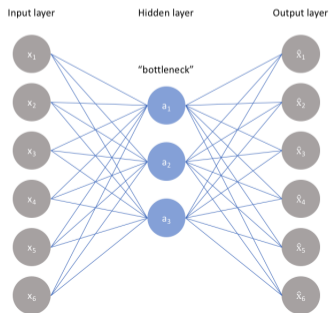


Otros tipos de composición

Además de la suma, la composición de los vectores se puede hacer por medio de una transformación de la forma:

$$T[u; v] \in \mathbb{R}^d$$

Donde $[u; v] \in \mathbb{R}^{2d}$ es la concatenación de los vectores. Esta transformación puede ser aprendida por un **AutoEncoder**.



AutoEncoder

Un AutoEncoder es un tipo de red neuronal no-supervisada, donde existe una capa oculta, tal que la función anterior a esta se conoce como **encoder** y la función superior como **decoder**.

La salida del AutoEncoder debe ser la entrada reconstruida. Así, tenemos dos funciones tal que:

$$[\hat{u}; \hat{v}] = T^{-1}T[u; v]$$

Donde \hat{u} y \hat{v} son las versiones reconstruidas. En su forma más simple, el AutoEncoder puede estar determinado por dos matrices $W \in \mathbb{R}^{d \times 2d}$ (encoder) y $U \in \mathbb{R}^{2d \times d}$ (decoder). La función de riesgo será entonces:

$$R(W, U) = \|[u; v] - [\hat{u}; \hat{v}]\|_2^2 = \|[u; v] - UW[u; v]\|_2^2$$

Otros métodos de word embeddings

El método de Word2Vec no es el único que se ha propuesto. Actualmente, existen muchos métodos neuronales para creación de word embeddings.

En este módulo, revisaremos dos métodos basados en redes FeedForward:

- ▶ FastText [2] propuesto por el *Facebook IA Research Group*.
- ▶ GloVe (Global Vectors) [5] propuesto por la Universidad de Stanford.

Otros métodos actuales se basan en otro tipo de arquitecturas neuronales, como redes recurrentes y transformers.

FastText

El método de FastText consta con la misma arquitectura que el método de Word2Vec, pero su diferencia radica en el vocabulario.

Mientras que Word2Vec toma palabras como parte del vocabulario, FastText toma **subwords** como su vocabulario.

Las subwords son determinadas como **n-gramas de caracteres** (comúnmente trigramas), así, una palabra como «gatitos» se descompone en:

$$\text{gatitos} \rightarrow \{ \langle \text{ga}, \text{gat}, \text{ati}, \text{tit}, \text{ito}, \text{tos}, \text{os} \rangle \}$$

Los símbolos < y > se agregan para indicar inicio y fin de la palabra.

FastText: Representación de palabras

Para representar las palabras como vectores distribuidos, el método de FastText hace uso de la **composición**: la representación de la palabra es la suma de la representación de sus partes:

$$F(\text{gatitos}) = F(< ga) + F(\text{gat}) + F(\text{ati}) + F(\text{tit}) + (\text{ito}) + F(\text{tos}) + F(\text{os} >)$$

Donde $F : \Sigma \rightarrow \mathbb{R}^d$ es la función de embedding (arquitectura word2vec).

De forma general, para representar una palabra w , se tiene que:

$$F(w) = \sum_{i=2}^{m+1} F(a_{i-2}, a_{i-1}, a_i)$$

Las **ventajas** de este método es que puede representar OOVs, además de que puede trabajar eficientemente con corpus más pequeños.

GloVe

El método de GloVe (Global Vectors) retoma la hipótesis distribucional buscando encontrar representaciones vectoriales que respondan a la co-ocurrencia de palabras.

De esta forma, define una **matriz de co-ocurrencias** X , cuyas entradas son:

$$X_{i,j} = |\{w_j : w_j \in \mathcal{N}(w_i)\}|$$

donde $\mathcal{N}(w_i)$ representa los contextos de w_i . Esto es **el número de contextos que comparten** w_i y w_j .

Claramente $X_{i,j} = X_{j,i}$.

GloVe: probabilidad de coocurrencia

La **probabilidad de coocurrencia** de dos palabras, estará dada como:

$$p(w_j|w_i) = \frac{X_{i,j}}{\sum_j X_{i,j}}$$

Dos palabras w_i y w_j serán similares entre más contextos compartan; es decir, serán más similares en cuanto el coeficiente:

$$\frac{p(w_i|w_k)}{p(w_j|w_k)}$$

sea más cercano a 1.

GloVe: Isomorfismo

El objetivo de GloVe es encontrar un **isomorfismo** entre el espacio \mathbb{R}^d y Σ .

Sean $u_i, u_j, \hat{u}_k \in \mathbb{R}^d$ vectores que representan las palabras w_i, w_j, w_k . Sea F una función tal que cumpla:

$$F[(u_i - u_j) \cdot \hat{u}_k] = F[u_i \cdot \hat{u}_k - u_j \cdot \hat{u}_k] = \frac{p(w_k | w_i)}{p(w_k | w_j)}$$

Ya que $F[(u_i - u_j) \cdot \hat{u}_k] = \frac{F[u_i \cdot \hat{u}_k]}{F[u_j \cdot \hat{u}_k]}$, entonces:

$$F[u_i \cdot \hat{u}_k] = p(w_k | w_i) = \frac{X_{i,j}}{\sum_j X_{i,j}}$$

De donde es claro que $F(x) = e^x$.

GloVe: Representación vectorial

Dado que $e^{u_i \cdot \hat{u}_k} = \frac{X_{i,j}}{\sum_j X_{i,j}}$, podemos despejar los vectores, tal que:

$$u_i \cdot \hat{u}_k = \log X_{i,j} - \log X_i$$

Buscamos **conservar la simetría**, por lo que el factor $-\log X_i$ puede pasar como bias del lado izquierdo, tal que:

$$u_i \cdot \hat{u}_k + b_i + b_k = \log X_{i,j}$$

De esta forma, la parte izquierda de la ecuación responde a una red neuronal. Podemos definir la función de riesgo como:

$$R(\theta) = \sum_{i,j} (u_i \cdot \hat{u}_k + b_i + b_k - \log X_{i,j})^2$$

GloVe: Corrección de la función de riesgo

Sin embargo, si $X_{i,j} = 0$, entonces $\log X_{i,j} = \infty$, por lo que no podremos alcanzar el mínimo.

Para solucionar esto, se multiplica el riesgo por una función f :

$$R(\theta) = \sum_{i,j} f(X_{i,j})(u_i \cdot \hat{u}_k + b_i + b_k - \log X_{i,j})^2$$

Tal que:

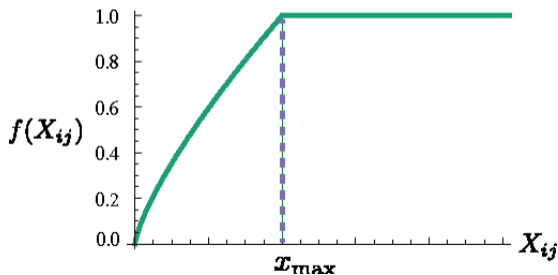
1. Si $X_{i,j} = 0$, entonces $f(X_{i,j}) = 0$.
2. f no decreciente.
3. f pequeño para valores grandes.

GloVe: Corrección de la función de riesgo

Para la función f entonces se propone:

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^\alpha & \text{si } x < x_{max} \\ 1 & \text{en otro caso} \end{cases}$$

Donde $\alpha = \frac{3}{4}$ (relacionado con Zipf).



Embeddings estáticos

Las representaciones que hemos discutido se consideran **word embeddings estáticos**. Se dice que son estáticos porque a cada token del vocabulario le corresponde un vector que no varía.

Estos embeddings son:

- ▶ Word2Vec
- ▶ GloVe
- ▶ FastText

Existen otros métodos de embeddings estáticos como Node2Vec, Doc2Vec, etc.






Embeddings pre-entrenados

Es común que los embeddings de los tokens se pre-entrenen; es decir, que se creen los embeddings a partir de una cantidad grande de corpus para obtener vectores del mayor número de palabras. Estas representaciones se usan en otras tareas.

Se pueden proponer dos formas en cómo utilizar estos embeddings pre-entrenados:

- Feature based:** Las representaciones son utilizadas como features (rasgos) en un modelo de aprendizaje. Por ejemplo, se pueden tomar estas representaciones como los vectores de entrada de un clasificador o un método de agrupamiento.
- Fine tuning:** Las representaciones se ajustan a la tarea final (clasificación, agrupamiento, etc.). Se pueden utilizar estas representaciones para inicializar los parámetros del modelo de aprendizaje.

References

-  Marco Baroni, Raffaella Bernardi, and Roberto Zamparelli.
Frege in space: A program of compositional distributional semantics.
LiLT (Linguistic Issues in Language Technology), 9, 2014.
-  Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov.
Enriching word vectors with subword information.
Transactions of the Association for Computational Linguistics, 5:135–146, 2017.
-  Zellig S Harris.
Distributional structure.
Word, 10(2-3):146–162, 1954.
-  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.
-  Jeffrey Pennington, Richard Socher, and Christopher Manning.
Glove: Global vectors for word representation

The End