Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 6: Word Vectors

Word Vectors

- Representaciones del lenguaje
- Word2Vec

Word vectors

- También conocidos como ***word embeddings***.
- Un tipo de representación de palabras que ha ayudado en gran medida a la aplicación del deep learning en problemas de lenguaje natural.

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 6.1: Representaciones del lenguaje

Representaciones del lenguaje

- Entender el lenguaje natural es un problema complejo. La comunicación humana está llena de matices y ambigüedades.
- ¿Cómo representar computacionalmente el significado de una palabra?
- Una solución clásica ha sido la creación de bases de datos de palabras agrupadas por significado, con reglas que definen las relaciones semánticas entre ellas. Ejemplo: WordNet.

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Fuente de la imagen: Stanford CS224N

Representaciones del lenguaje

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

- Este tipo de recursos tiene una serie de inconvenientes:
  - Los matices se pierden en muchas ocasiones. Ejemplo: *honorable* sinónimo de *good*.
  - Requiere de mucho trabajo manual para recopilar toda la información.
  - El lenguaje cambia continuamente, con nuevas acepciones.
  - Es complicado obtener una medida numérica de similitud entre palabras.

Fuente de la imagen: Stanford CS224N

Representaciones discretas

- Tradicionalmente, el texto se ha representado computacionalmente mediante el uso de **representaciones discretas**.

- A partir de un vocabulario V, se representa el texto mediante un vector de tamaño del número de elementos de V, donde cada elemento del vector representa una palabra. Una forma común de hacer esto es representar el número de veces que aparece una palabra en el texto (*term frequency*):

```
V = [perro, gato, mi, tu, simpático, ladrador, es, era]

"Mi perro es ladrador" -> [1, 0, 1, 0, 0, 1, 1, 0]
```

- Este tipo de representación se conoce como **bag-of-words**. En bag-of-words no nos interesa el orden de las palabras, solo si aparecen o no en el texto y cuántas veces lo hacen.

Representaciones discretas

- Una palabra se representa mediante un vector sencillo.

```
V = [perro, gato, mi, tu, simpático, ladrador, es, era]

"perro" -> [1, 0, 0, 0, 0, 0, 0, 0]
"gato"  -> [0, 1, 0, 0, 0, 0, 0, 0]
```

- Con este tipo de representación perdemos una gran cantidad de información. Las palabras perro y gato tienen una gran relación (ambas definen animales de compañía) y sin embargo sus representaciones son totalmente ortogonales.

- Nos gustaría obtener una representación que nos aporte información semántica y que nos permita obtener un concepto de similitud entre palabras.

Representaciones distribuidas

- La idea será intentar codificar esta información semántica en un vector denso llamado **word vector**.
- La representación de una palabra mediante un vector denso se conoce como **representación distribuida**.
- Estas representaciones podrán ser utilizadas en nuestros modelos de machine learning.

$$linguistics = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Fuente de la imagen: Stanford CS224N

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 6.2: Word2Vec

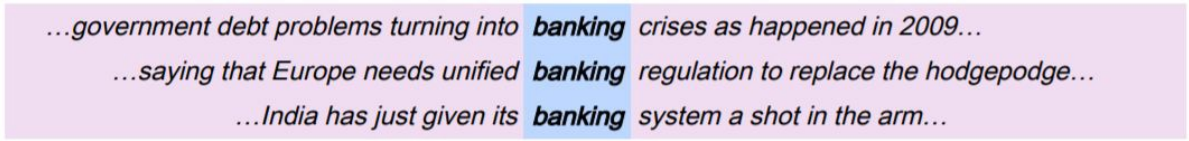
Universidad Internacional de La Rioja

Word2Vec

- Idea clave para construir word vectors:

El significado de una palabra viene dado por las palabras que aparecen frecuentemente junto a ella.

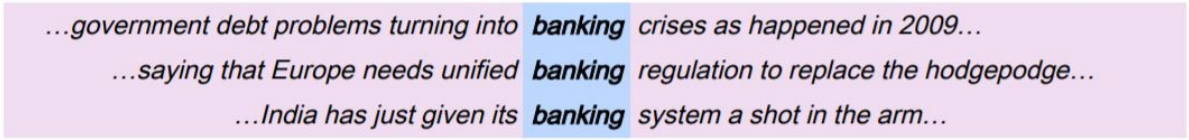
- Podemos extraer el significado de una palabra estadísticamente a partir del contexto en el que la palabra está presente.



- El contexto de una palabra será el conjunto de palabras que aparecen cerca de ésta, dentro de un rango de tamaño determinado (*window* o ventana)

Fuente de la imagen: Stanford CS224N

Word2Vec



- A partir de una gran cantidad de texto, obtendremos un vector denso para cada palabra que sea similar a los vectores de palabras que aparecen en contextos similares.
- Aparecer en contextos similares significa que las palabras guardan una relación semántica.
- Los vectores obtenidos serán los **word vectors**, también conocidos como **word embeddings** o **word representations**.

Fuente de la imagen: Stanford CS224N

Word2Vec

- El algoritmo que utilizaremos para construir los word vectors es conocido como **word2vec** y data de 2013.
- La idea es utilizar un corpus grande de texto (por ejemplo, todo el texto en un idioma de la Wikipedia), que es tratado estadísticamente para obtener los vectores de cada palabra.
- El output del algoritmo es un vector por cada palabra en el vocabulario.
- Existen otras alternativas para calcular word vectors. Una de las más famosas es GloVe.

Fuente de la imagen: Stanford CS224N

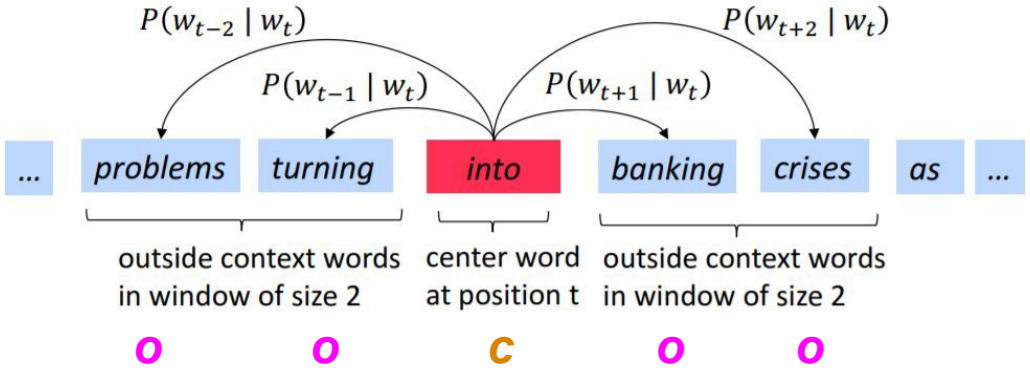
Word2Vec

- El algoritmo funciona, a grandes rasgos, de la siguiente manera:

1. Se recorre cada posición *t* en el texto, obteniendo una palabra central ***c*** y un contexto de palabras ***o***.

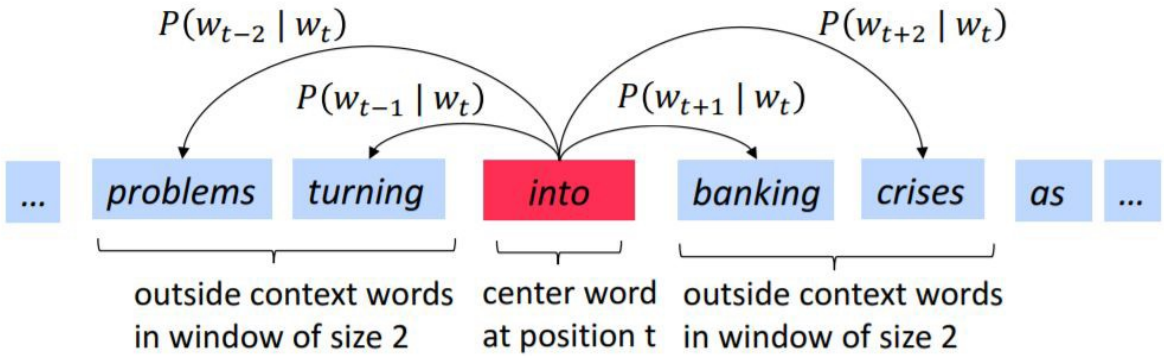
2. Utilizando las similitudes de los vectores calculados hasta el momento, se obtiene la probabilidad de cada palabra del contexto ***o*** dado ***c***.

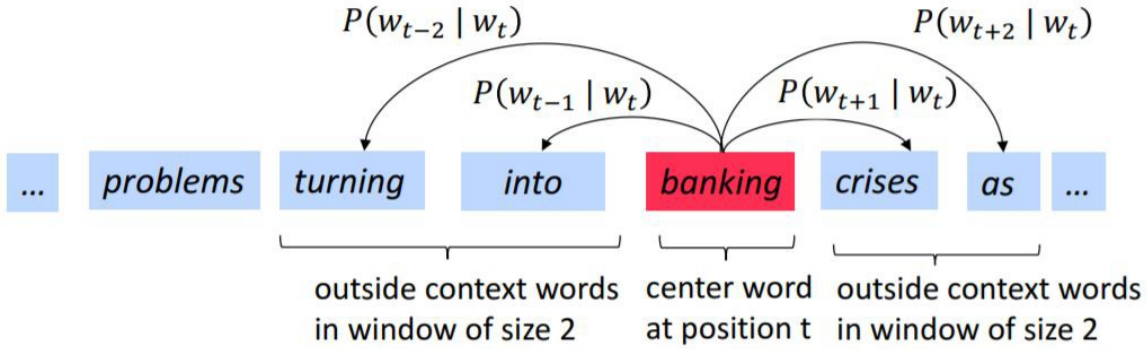
3. Se ajustan los vectores para maximizar la probabilidad de lo visto en los datos (*maximum likelihood*).



Fuente de la imagen: Stanford CS224N

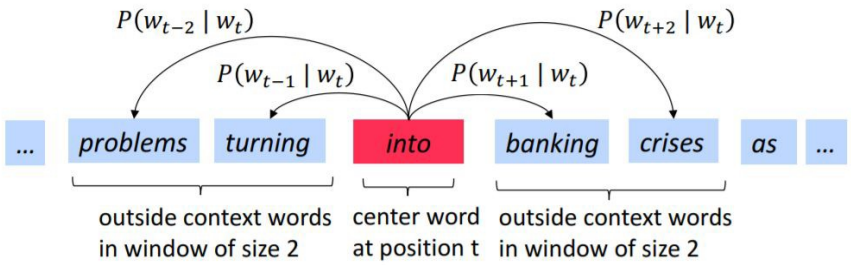
Word2Vec





Fuente de la imagen: Stanford CS224N

Word2Vec



- A partir de todas las posiciones vistas en el texto y todas las probabilidades obtenidas, podemos calcular una verosimilitud o *likelihood*.

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

donde *T* es el número total de palabras y *m* es el tamaño de ventana elegido.

- El objetivo es maximizar esta verosimilitud. Si maximizamos las probabilidades vistas en los datos mediante la elección de unos word vectors adecuados, seremos capaces de predecir las palabras del contexto a partir de una palabra central.

Fuente de la imagen: Stanford CS224N

Word2Vec - Función de coste

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

- Queremos obtener una función de coste a minimizar.
- Maximizar *L* es equivalente a minimizar *-L*. Asimismo, aplicamos logaritmos, de modo que los productos se convierten en sumas. La **función de coste** resultante a minimizar es:

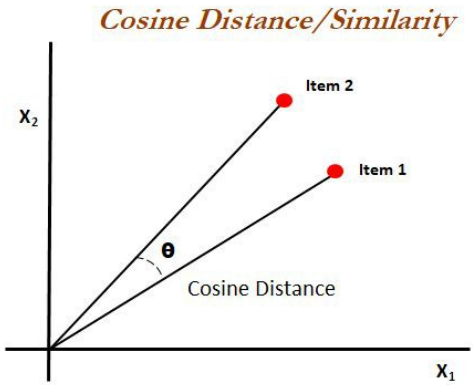
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- Los parámetros de esta función son los word vectors. Nos falta ver cómo calcular las probabilidades a partir de estos, que se modelan a partir de similitudes de vectores.

Word2Vec - Similitud de vectores

- El **producto escalar** es una medida de similitud de dos vectores. Es la que utilizaremos en word2vec.
- Otra forma tradicional de calcular la similitud entre dos vectores es mediante la **cosine similarity**.

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



Fuente de la imagen: https://www.safaribooksonline.com/library/view/statistics-for-machine/9781788295758/eb9cd609-e44a-40a2-9c3a-f16fc4f5289a.xhtml

Word2Vec - Probabilidades a partir de vectores

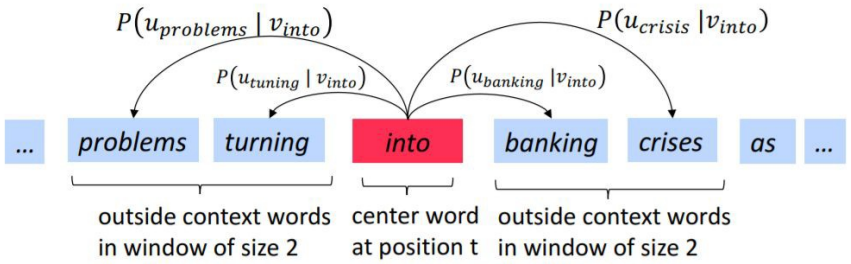
- Vamos ahora a obtener la probabilidad en nuestra función de coste:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- Modelamos la probabilidad de tener una palabra contexto **o** junto a una palabra central **c** a partir de una similitud de vectores. Se define:
  - v_w como el vector de la palabra *w* cuando ésta actúa como palabra central.
  - u_w como el vector de la palabra *w* cuando ésta actúa como palabra de contexto.

- La probabilidad viene dada mediante un softmax de similitudes dadas por producto escalar (define una distribución de probabilidad de cómo de parecidas son dos palabras):

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)}$$



Fuente de la imagen: Stanford CS224N

Word2Vec - Recapitulando

- A partir de un corpus de texto y un vocabulario hemos definido una función de coste

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P(w_{t+j} \mid w_t;\theta) \qquad P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)}$$

cuyos parámetros vienen dados por dos vectores por cada palabra *u* y *v*. En total, tenemos un total de 2dV parámetros:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

donde V es el tamaño del vocabulario y d es la dimensión de los vectores.

Fuente de la imagen: Stanford CS224N

Word2Vec - Recapitulando

- A partir de un corpus de texto y un vocabulario hemos definido una función de coste

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P(w_{t+j} \mid w_t; \theta) \qquad P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)}$$

- Una vez definido el problema como un problema de optimización, podemos obtener los vectores *u* y *v* a partir de gradient descent o SGD.

- Para obtener el vector final de cada palabra, se hace la media de los vectores *u* y *v*.

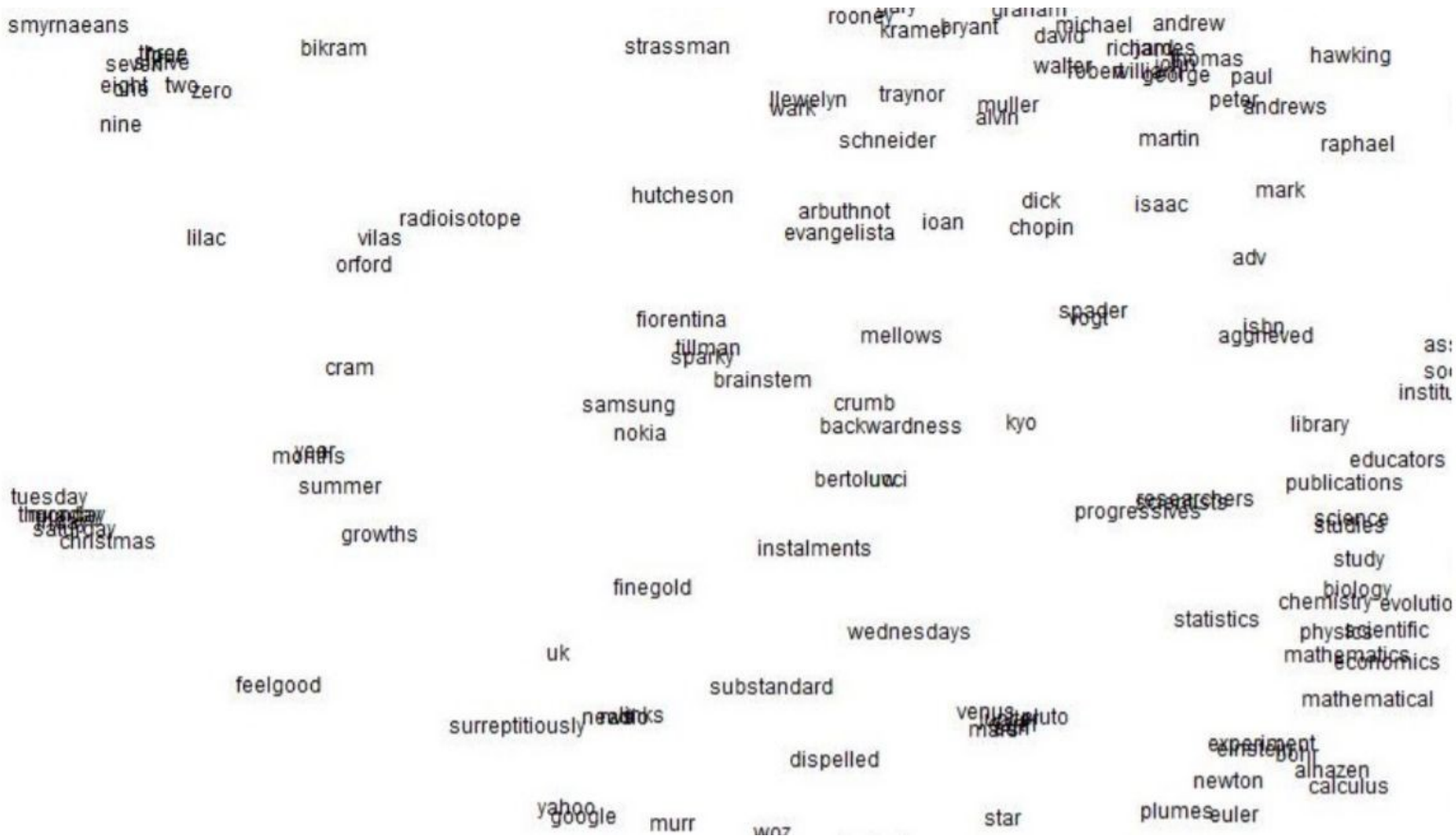
- ¿Por qué el algoritmo utiliza dos vectores por cada palabra en vez de uno? El problema de optimización es más sencillo de resolver de este modo y, además, los resultados experimentales son mejores.

Word2Vec - Negative Sampling

- Aplicar gradient descent o SGD directamente sobre el problema definido es demasiado complicado computacionalmente.
- Al calcular derivadas, tenemos que obtener gradientes respecto a los vectores de todas las palabras del vocabulario, no sólo las del contexto y la palabra central. Esto es muy costoso.
- En la práctica, se aplica una técnica conocida como **negative sampling**. Se toman las palabras que aparecen juntas y se elige al azar un pequeño número de palabras que no. De este modo, no se calculan gradientes respecto a todas las palabras del vocabulario.
- El objetivo es acercar los vectores de las palabras que aparecen juntas y alejar los vectores de los que no, sin tener que recurrir a tener que obtener gradientes para todas las palabras del vocabulario.
- Esta técnica es una aproximación al problema que es computacionalmente tratable y obtiene muy buenos resultados.

Word2Vec - Resultados

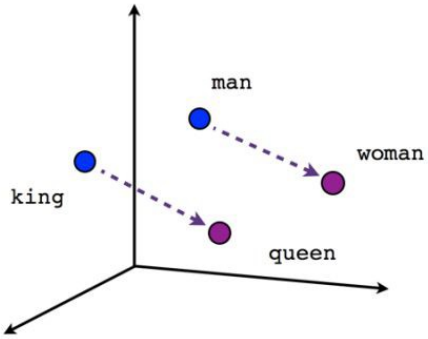
Clústeres de palabras:



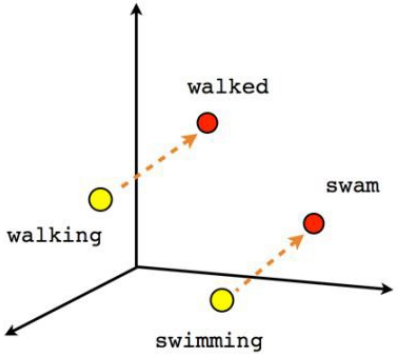
Fuente de la imagen: Stanford CS224N

Word2Vec - Resultados

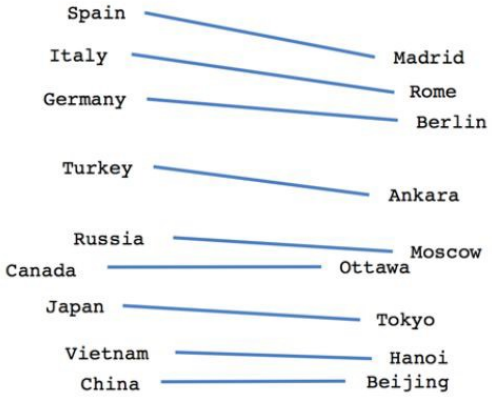
Aritmética de vectores:



Male-Female



Verb tense



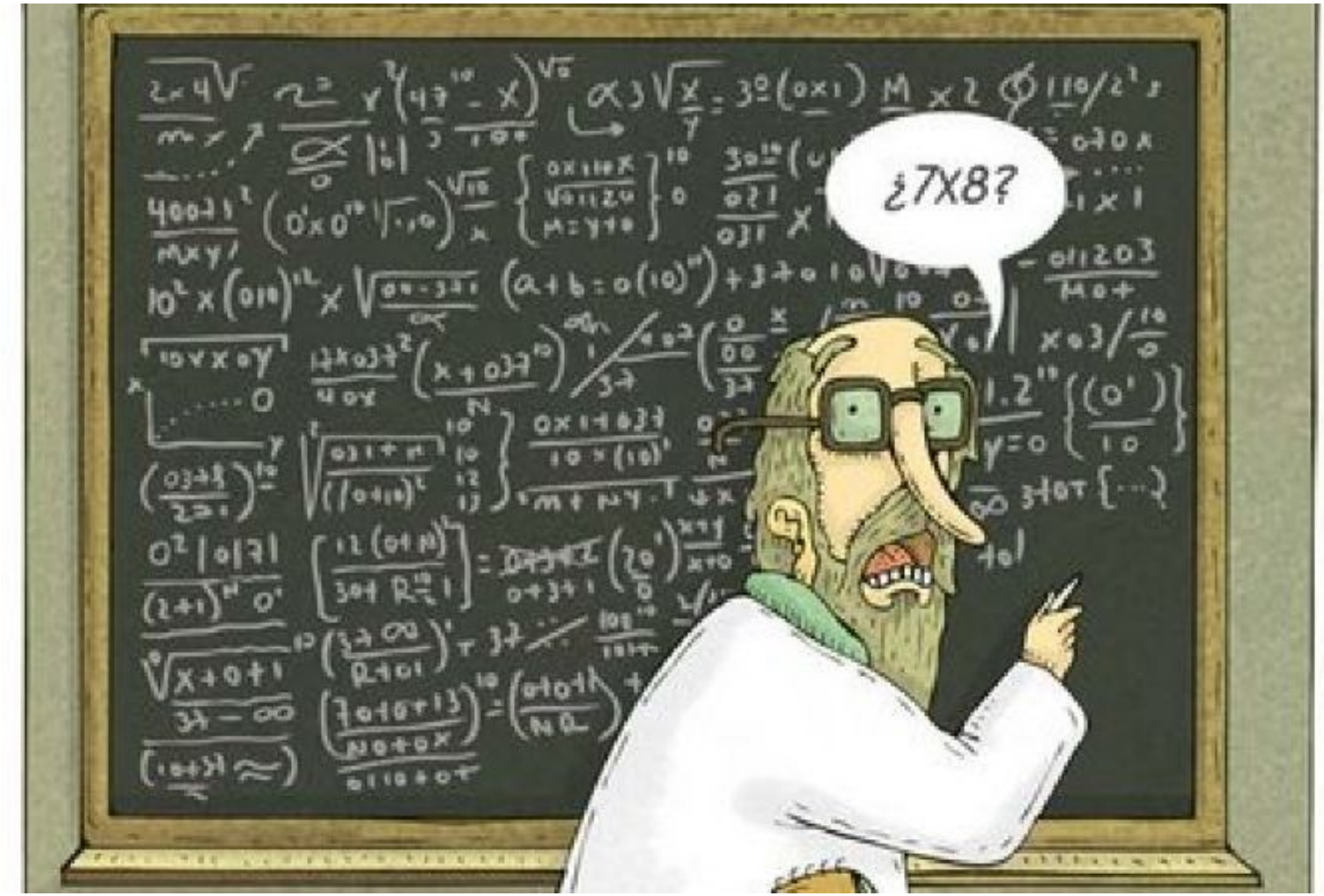
Country-Capital

Fuente de la imagen: https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12

Word2Vec

- Los word vectors son una pieza clave a la hora de entrenar modelos de deep learning para procesamiento del lenguaje natural.
- Es frecuente utilizar **word vectors pre-entrenados**. Es sencillo obtener conjuntos de vectores ya entrenados para utilizar en nuestros modelos.
- Igualmente, es posible utilizar word vectors como parte de nuestro modelo de deep learning y **entrenarlos a la vez que se entrena el modelo**, sin tener que aplicar word2vec con anterioridad.

¿Dudas?



www.unir.net