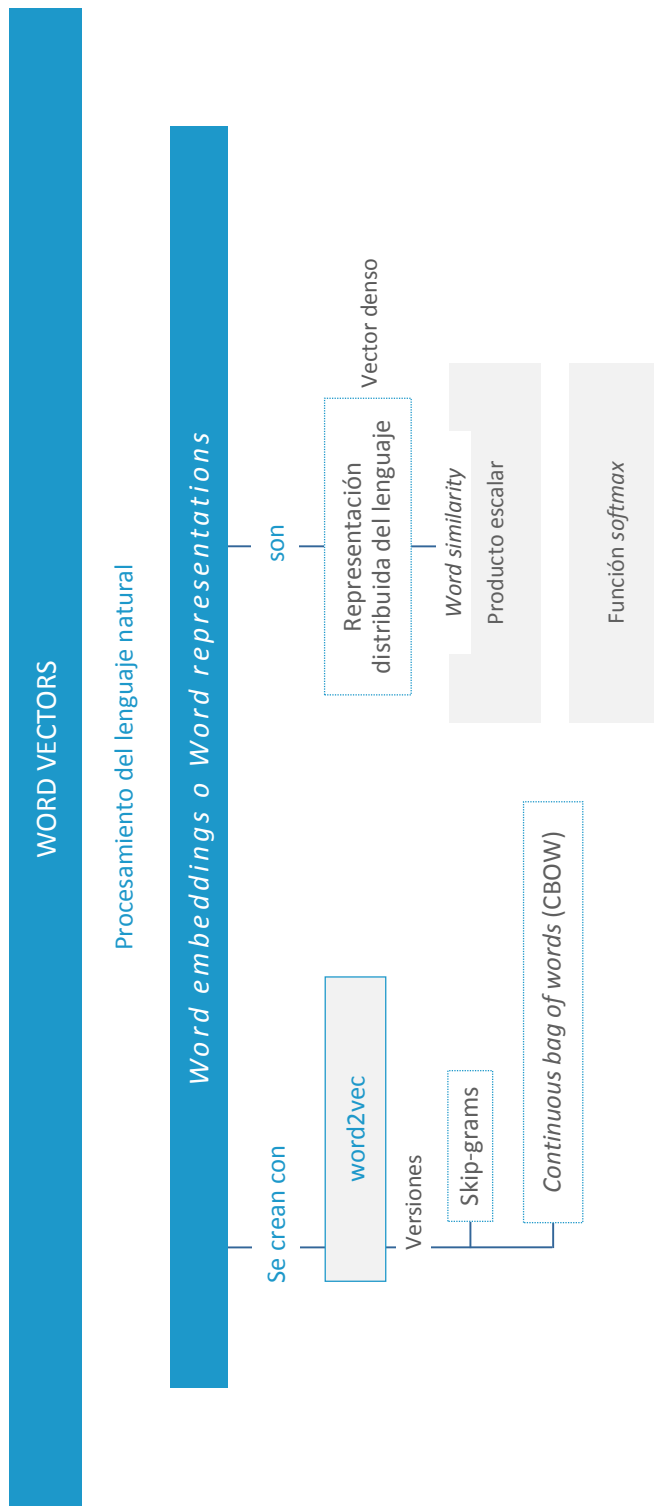


Sistemas Cognitivos Artificiales

Word Vectors

Índice

Esquema	3
Ideas clave	4
6.1. ¿Cómo estudiar este tema?	4
6.2. Representaciones del lenguaje	4
6.3. Word2Vec	7
6.4. Referencias bibliográficas	18
Lo + recomendado	19
+ Información	22
Test	23



6.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se desarrollan a continuación.

En este tema nos adentraremos en el mundo del PLN o procesamiento del lenguaje natural (NLP en inglés: *natural language processing*). En particular, veremos un tipo de representación de palabras llamado *word vectors*, que se ha hecho muy popular en los últimos años y ha propulsado la utilización del *deep learning* en problemas de lenguaje natural, así como un algoritmo para obtener estos vectores, **word2vec**.

Es importante comprender en este tema las distintas formas de representación del lenguaje y por qué los *word vectors* tienen más capacidad de representación que otros métodos anteriores. También es importante entender el funcionamiento de word2vec y las propiedades de los *word vectors* resultantes.

6.2. Representaciones del lenguaje

El lenguaje natural ha sido siempre un problema esquivo para la inteligencia artificial. Los matices y ambigüedades presentes en la comunicación humana a través del lenguaje han hecho que los sistemas de inteligencia artificial no sean capaces en muchas instancias de entender cosas que son triviales para un humano.

Uno de los problemas fundamentales es la representación en un ordenador del significado de una palabra. Una solución clásica al problema ha sido la creación de

bases de datos con palabras agrupadas por significado y reglas que definen las relaciones semánticas entre ellas. WordNet, por ejemplo, es un recurso de este tipo en inglés que permite obtener sinónimos y antónimos:

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')]
```

Figura 1. Información disponible en WordNet.

Fuente: <http://web.stanford.edu/class/cs224n/>

Este tipo de recursos funciona bien pero tiene una serie de **inconvenientes**:

- ▶ Muchas veces los matices se pierden. Por ejemplo, «*honorable*» aparece como sinónimo de «*good*», pero en muchos contextos una palabra no podría cambiarse por la otra.
- ▶ El lenguaje está en constante cambio con nuevos significados de palabras, lo que hace que este tipo de recursos pierdan ciertas acepciones modernas.
- ▶ Requiere mucho trabajo manual de un gran número de personas para recopilar una base de datos tan grande.
- ▶ Es complicado obtener una medida numérica de similitud entre palabras.

Representaciones discretas del lenguaje

Tradicionalmente, el texto se ha representado computacionalmente mediante el uso de **representaciones discretas**. Empezando con un vocabulario V , se representa el texto mediante un vector del tamaño del número de elementos de V , donde cada elemento del vector representa una palabra. Una representación común es utilizar *term frequency*, donde se asigna el número de veces que aparece cada palabra en ese vector.

Ejemplo 1. Para un vocabulario $V = [\text{perro}, \text{gato}, \text{mi}, \text{tu}, \text{simpático}, \text{ladrador}, \text{es}, \text{era}]$, la representación de la frase «mi perro es ladrador» sería:

$$[1, 0, 1, 0, 0, 1, 1, 0]$$

Esto nos dice el número de veces que aparece cada palabra en el texto y nos da una representación discreta del mismo, donde cada palabra se asocia a un valor del vector. En particular, esta representación se conoce como ***bag-of-words***. En *bag-of-words* no nos interesa el orden de las palabras, solo su aparición o no en el texto (de ahí la idea de que metemos las palabras en una «bolsa»).

Ejemplo 2. Con esta representación, una palabra sería expresada mediante un vector sencillo. Por ejemplo, una palabra como «hotel» podría ser representada utilizando un vocabulario distinto al del ejemplo anterior, como $[0, 0, 0, 0, 1, 0]$.

Este tipo de representación discreta **pierde una gran cantidad de información** sobre la palabra. Por ejemplo, la palabra «motel» guarda una gran relación con «hotel». Si estamos buscando hoteles por Internet, los resultados con «motel» también serían relevantes. Sin embargo, la representación discreta de «motel» sería un vector del tipo $[1, 0, 0, 0, 0, 0]$, que es totalmente ortogonal al vector de «hotel», haciendo imposible ver cualquier tipo de similitud entre ellos.

El objetivo sería pues, obtener una representación de una palabra que nos **aporte información semántica** sobre ella y que nos permita obtener un concepto de similitud entre representaciones de palabras. La idea será intentar codificar toda esa información en un vector denso, llamado **word vector**, lo que nos permitiría comparar palabras y utilizar esas representaciones en nuestros modelos de aprendizaje automático.

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

Figura 2. Representación distribuida de una palabra en un vector.

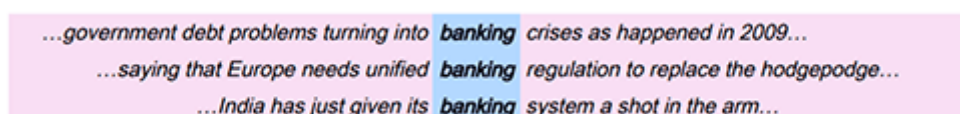
Fuente: <http://web.stanford.edu/class/cs224n/>

La representación de una palabra mediante un vector denso se conoce como **representación distribuida**.

6.3. Word2Vec

Una vez que el objetivo que buscamos está claro, ¿cómo construimos un vector que represente el significado de una palabra? La idea clave que utilizaremos es que el significado de una palabra viene dado por las palabras que aparecen frecuentemente junto a ella.

Esta idea nos permite definir un procedimiento estadístico para construir el vector de una palabra. Si tenemos una gran cantidad de texto disponible, podemos extraer el significado de las palabras mediante el contexto en el que están presentes. Definiremos el **contexto de una palabra** como el conjunto de palabras que aparecen cerca de ella, dentro de un rango de tamaño determinado por nosotros, que definiremos como *window* o ventana.



...government debt problems turning into **banking** crises as happened in 2009...
...saying that Europe needs unified **banking** regulation to replace the hodgepodge...
...India has just given its **banking** system a shot in the arm...

Figura 3. Contexto de la palabra «*banking*».

Fuente <http://web.stanford.edu/class/cs224n/syllabus.html>

En la imagen anterior podemos ver el contexto de la palabra «*banking*» en varios textos. Estadísticamente, esta palabra aparecerá frecuentemente en contextos con palabras y conceptos parecidos. Sus sinónimos o palabras relacionadas aparecerán del mismo modo en el mismo tipo de contextos.

La idea aquí es obtener un vector denso para cada palabra de manera que sea similar a los vectores de palabras que aparecen en contextos similares, lo que indica que las palabras guardan una relación semántica. Estos vectores serán nuestros *word vectors*, también conocidos como ***word embeddings* o *word representations***.

Algoritmo word2vec

El algoritmo que utilizaremos para construir los *word vectors* es conocido como word2vec y fue presentado en 2013 (Mikolov, Chen, Corrado y Dean, 2013). El algoritmo se emplea sobre un corpus grande de texto (por ejemplo, todo el texto en un idioma de Wikipedia), sobre el que se obtiene un vocabulario de palabras disponibles. El *output* será un vector por cada palabra en el vocabulario.

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

- ▶ Se recorre cada posición t en el texto, obteniendo una palabra central o y un contexto de palabras c .
- ▶ Utilizando las similitudes de los vectores calculados hasta el momento, se obtiene la probabilidad de c dado o .
- ▶ Se ajustan los vectores para maximizar esta probabilidad.

Por ejemplo, veamos las probabilidades que existen en una ventana (contexto) de tamaño 2:

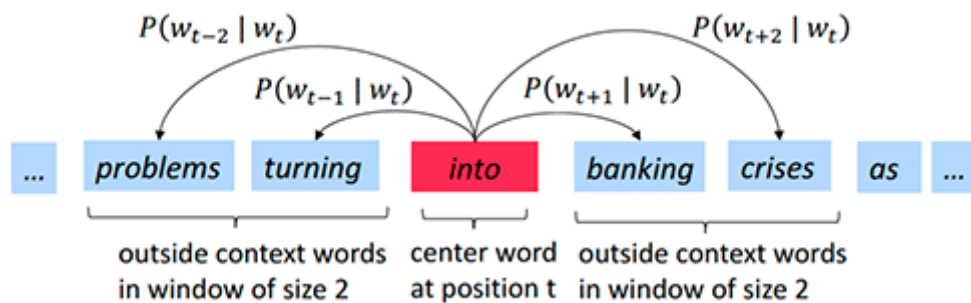


Figura 4. Ejemplo en una ventana de tamaño 2.

Fuente: <http://web.stanford.edu/class/cs224n/>

En la siguiente posición t en el texto tendremos:

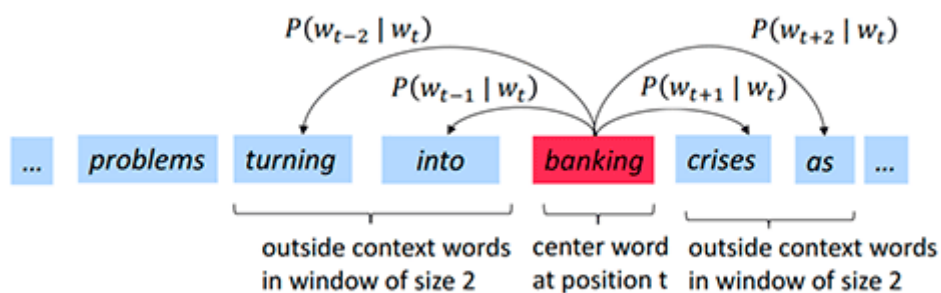


Figura 5. Ejemplo en una ventana de tamaño 2 en otra posición t .

Fuente: <http://web.stanford.edu/class/cs224n/>

De este modo, podemos obtener para todas las posiciones en el texto una verosimilitud o *likelihood* total a partir de las probabilidades de que una palabra esté

en el contexto dada una palabra central. Los parámetros de esta distribución de probabilidad serán todos los *word vectors* que queremos calcular:

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

Donde:

- ▶ T es el número total de palabras en los textos;
- ▶ t es la posición actual;
- ▶ m es el tamaño de ventana elegido.

La idea es maximizar esta verosimilitud o *likelihood*. Si maximizamos estas probabilidades vistas en los datos mediante la elección de unos *word vectors* adecuados, seremos capaces de, a partir de una palabra central, predecir las palabras del contexto.

Para maximizar esta probabilidad, obtendremos, como ya hemos visto en otros problemas de *machine learning*, una **función de coste a minimizar**. Como queremos maximizar L , la función de coste a minimizar será el valor negativo de L .

Por otro lado, en vez de minimizar $-L$ directamente, se aplica el logaritmo de L . Esto es un truco para evitar problemas numéricos en los cálculos, ya que las probabilidades son números pequeños y el producto de números pequeños se vuelve cada vez menor. Al aplicar el logaritmo, los productos se convierten en sumas.

De este modo, la función de coste a minimizar es:

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

El único ingrediente que nos falta por ver es **cómo obtener la probabilidad** dentro de la suma. Esta se deriva a partir de la similitud de vectores. Durante el proceso de aprendizaje, se definen dos vectores v y u para cada palabra (en vez de uno):

- ▶ v_w es el vector de la palabra w cuando esta actúa como **palabra central**;
- ▶ u_w es el vector de la palabra w cuando esta actúa como **palabra de contexto**.

Con esto, la probabilidad a calcular con respecto a la palabra central c y la palabra de contexto o se modela de la siguiente manera:

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

Esto quedaría así en el ejemplo visto en las figuras 4 y 5:

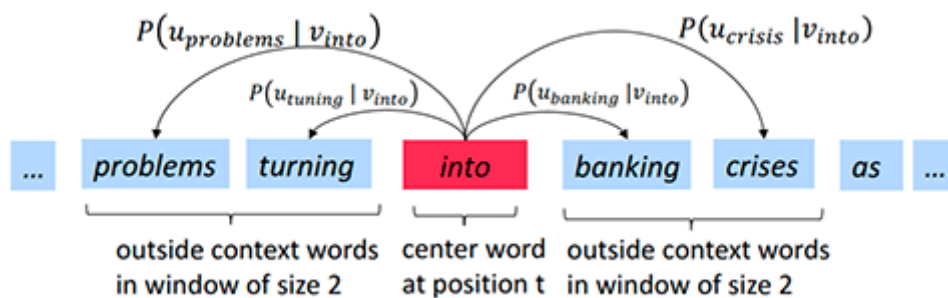


Figura 6. Probabilidad a calcular.

Fuente: <http://web.stanford.edu/class/cs224n/>

Similitud entre palabras (*word similarity*)

Para entender mejor cómo se obtiene esta probabilidad, veamos qué entendemos por similitud entre vectores. Para dos *word vectors*, una forma de obtener una medida de similitud es calcular su **producto escalar**. Este producto será mayor tanto para vectores que apuntan en la misma dirección como para valores absolutos grandes en direcciones similares.

La similitud mediante producto escalar puede hacerse arbitrariamente grande mediante valores mayores en los vectores, por eso es común utilizar el coseno del ángulo entre dos vectores como medida de similitud. El coseno de un ángulo tiene valores entre 0 y 1, y para dos vectores apuntando en la misma dirección, esto es, con ángulo 0°, tiene valor 1.

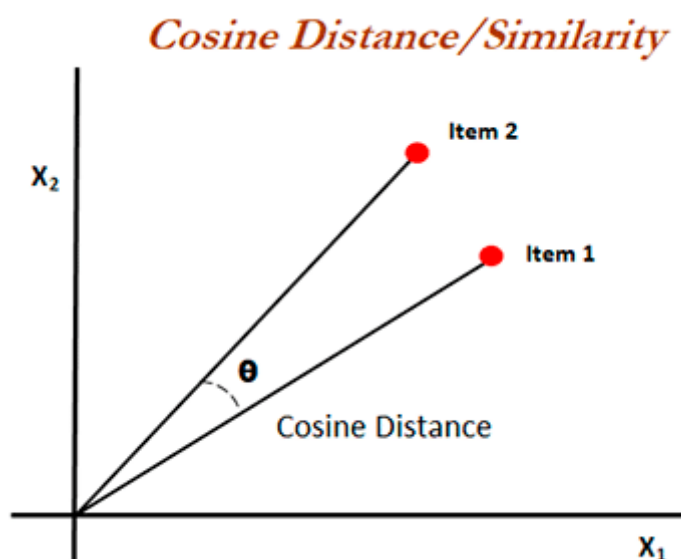


Figura 7. *Cosine Similarity*.

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

La similitud por coseno o *cosine similarity* se calcula mediante la fórmula:

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

Si bien lo común es utilizar *cosine similarity* como medida de *word similarity*, esto no es así en el caso de la probabilidad que estamos calculando en word2vec. En esta probabilidad se utiliza el producto escalar. La forma en la que el problema está dispuesto sobre un gran número de palabras impide, en cierta manera, que las similitudes se hagan arbitrariamente grandes mediante mayores valores en los vectores.

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

De este modo, volviendo a la fórmula podemos entender que la probabilidad de la palabra o dada la palabra central c es más grande cuanto más similares son.

Función softmax

Igualmente, la función que se utiliza para modelar la probabilidad de una palabra en el contexto dada otra central no es casual. Si bien esta es una elección del modelo (se podría haber modelado la probabilidad de otro modo), se utiliza una función muy común en *machine learning* llamada *softmax*. Esta convierte cualquier serie de números en una distribución de probabilidad (haciendo que los valores de la distribución sumen 1).

Para un vector (x_1, \dots, x_n) , la función *softmax* es:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

Como ejercicio para el alumno, se puede comprobar que los p_i suman 1.

El nombre de la función *softmax* viene de que se parece en cierta medida a un máximo, ya que la exponencial del numerador, el valor más grande del *input* (x_1, \dots, x_n) tiende a amplificarse. Sin embargo, como el resto de x_i aún tienen cierta probabilidad de salida, el máximo es «soft».

Calculando los *word vectors*

Tenemos ya todos los ingredientes para calcular los *word vectors*. Hemos definido una función de coste basada en unas probabilidades modelizadas por dos tipos de

vectores por cada palabra. Una vez llegados aquí, sabemos que la solución al problema es aplicar *gradient descent* o SGD para obtener los valores de nuestros parámetros. En esta ocasión, nuestros parámetros no son los pesos y *biases* de una red neuronal, sino todos los valores de los vectores u y v :

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

Figura 8. Cálculo de los *word vectors*.
Fuente: <http://web.stanford.edu/class/cs224n/>

El número de parámetros es $2dV$, donde:

- ▶ V es el número de palabras en nuestro vocabulario.
- ▶ d es la dimensión de los vectores.
- ▶ El número de parámetros se multiplica por 2 debido a que tenemos los vectores u y v por cada palabra.

Una vez resuelto el problema de optimización, se hace la media de los vectores u y v para obtener el *word vector* definitivo de la palabra. Podríamos preguntarnos para qué es necesario entonces tener dos vectores por palabra. Esto es así porque el problema de optimización es más sencillo de resolver de este modo y, además, los resultados experimentales son mejores.

Negative sampling

En la práctica, no se suele aplicar *gradient descent* o SGD directamente sobre el problema definido arriba, ya que el número de cálculos a realizar se complica ante el tamaño del vocabulario, que puede ser muy grande. La solución consiste en aplicar *negative sampling*, una técnica en la que se toman las palabras que aparecen juntas en el contexto y se eligen al azar un pequeño número de palabras que no. El objetivo es «acercar» los vectores de las palabras que aparecen juntas y «alejar» los vectores de las que no sin tener que recurrir a tener que hacer cuentas con todas las palabras del vocabulario.

Si bien los detalles no son importantes para esta clase, la **función de coste** a implementar sería:

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

Figura 9. Función coste a implementar.

Donde:

- ▶ El primer término son las palabras que aparecen juntas.
- ▶ El segundo es una suma (*expectation*) sobre valores negativos elegidos al azar entre el total de palabras que no aparecen en la ventana o *window*.

Skip-grams y CBOW models

Existen dos versiones de word2vec según cómo se calculen las probabilidades:

Skip-grams: es el modelo que hemos utilizado en esta clase. La idea es predecir las palabras del contexto a partir de la palabra central. Modelamos probabilidades sobre las palabras del contexto dada una palabra central.

Continuous Bag of Words (CBOW): la idea aquí es predecir la palabra central a partir de una «*bag*» de palabras contexto. Normalmente, los vectores de las palabras contexto se suman y modelamos la probabilidad de la palabra central.

Resultados

Hemos empezado este tema buscando una representación de las palabras en forma de vector, de manera que las características semánticas de la palabra estuvieran presentes y pudiéramos obtener medidas de similitud entre palabras. La idea de word2vec es buscar este significado en el contexto en el que las palabras suelen aparecer.

El objetivo es ver de manera estadística qué palabras aparecen normalmente juntas o en contextos similares para extraer ese concepto de similitud.

Los resultados en forma de *word vectors* funcionan tal y como se espera. Si aplicamos una reducción de dimensionalidad para representar los vectores que se obtienen en dos dimensiones, obtenemos un mapa donde se ve cómo palabras similares aparecen juntas en clusters. Por ejemplo, Nokia con Samsung en el centro o una serie de nombres en la esquina superior derecha.



Figura 10. Mapa de *word vectors*.
Fuente: <http://web.stanford.edu/class/cs224n/>

La representación mediante *word vectors* también permite hacer cierta aritmética de vectores muy interesante. La diferencia entre los vectores de palabras como «king» y «queen» tiende a ser la misma que la de vectores como «man» y «woman». Este tipo de relaciones es común en el espacio vectorial resultante.

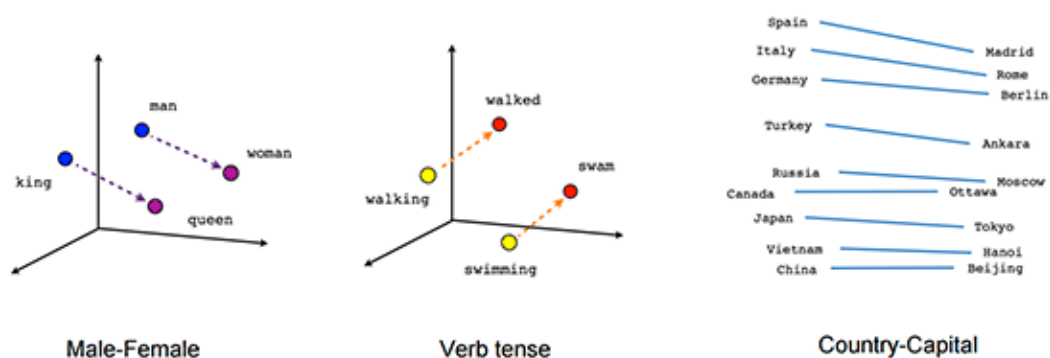


Figura 11. Ejemplos de representaciones de *word vectors*.
Fuente: <https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12>

Es importante mencionar, sin embargo, que esto no tiene por qué ocurrir siempre así. Al final, la calidad de los *word vectors* depende en gran medida de la **calidad y cantidad del texto** sobre el que se entrenan. Para entrenarlos, se suelen obtener enormes cantidades de texto, tales como *dumps* completos de la Wikipedia o Google News.

Los *word vectors* son una pieza clave a la hora de entrenar modelos de *deep learning* para PLN. Por ello, existe una gran cantidad de vectores de palabras ya entrenados disponibles en Internet (*pre-trained vectors*). Estos se pueden utilizar directamente en nuestros modelos sin necesidad de entrenarlos nosotros mismos.

6.4. Referencias bibliográficas

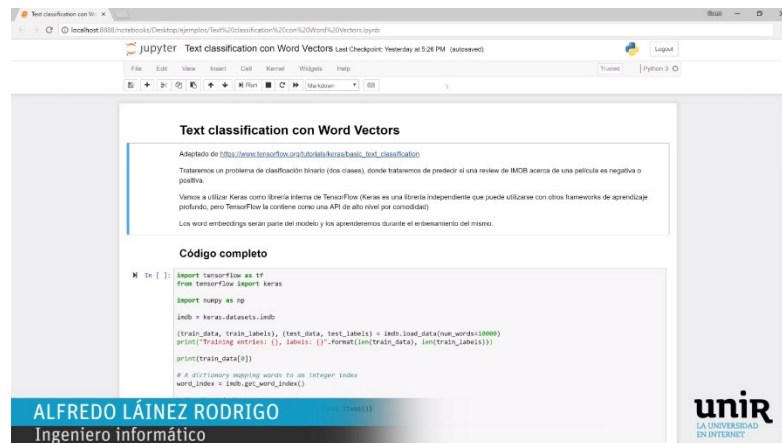
Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. Recuperado de <https://arxiv.org/abs/1301.3781>

Lo + recomendado

Lecciones magistrales

Clasificación de texto con Word Vectors

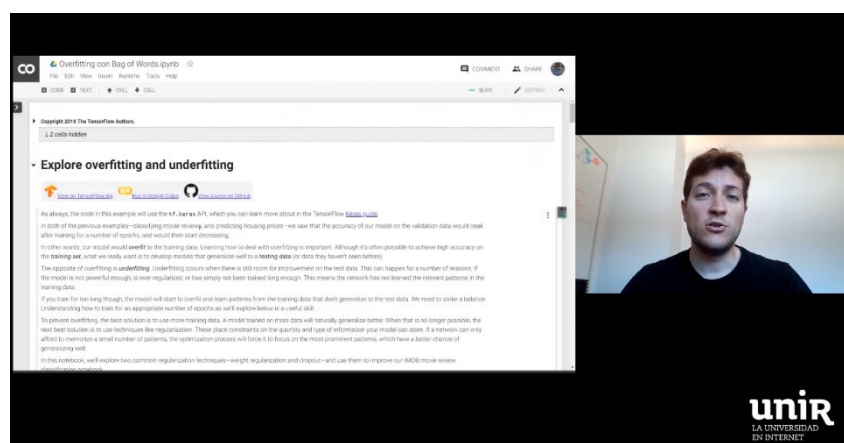
En este vídeo veremos el problema de clasificación binario de textos mediante Word Vectors en el que trataremos de predecir si una *review* de IMDB acerca de una película es negativa o positiva. Se utilizará Keras como librería interna de TensorFlow.



Accede a la lección magistral a través del aula virtual

Análisis de *overfitting* con un modelo *bag of words*

En esta magistral continuaremos con el problema de clasificación de textos y abordaremos cómo programar una red neuronal y entrenar un modelo tipo *bag of words*. Visto esto, exploraremos el fenómeno de *overfitting*.



Accede a la lección magistral a través del aula virtual

No dejes de leer

Word2Vec tutorial: the Skip-Gram model

McCormick, C. (19 de abril de 2016). Word2Vec tutorial: the Skip-Gram model [Blog post].

Explicación tutorial sobre Word2Vec por parte de Chris McCormick, de Nearist.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

Distributed Representations of Words and Phrases and their Compositionality

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. y Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2, 3111-3119.

Paper original sobre word2vec y las representaciones distribuidas.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>

A fondo

GloVe: Global Vectors for Word Representation

Pennington, J., Socher, R. y Manning, C. D. (2014). GloVe: Global Vectors for Word Representation [Archivo de datos y libro de códigos].

GloVe, otro algoritmo para obtener *word vectors*. Se puede acceder tanto al propio código como a documentación relacionada.

Accede a la documentación a través del aula virtual o desde la siguiente dirección:

<https://nlp.stanford.edu/projects/glove/>

A Primer on Neural Network Models for Natural Language Processing

Goldberg, Y. (2015). *A Primer on Neural Network Models for Natural Language Processing*. Manuscrito en preparación [En línea].

Un texto muy completo sobre *deep learning* aplicado a problemas de PLN. Incluye explicaciones acerca de *word vectors*.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<http://u.cs.biu.ac.il/~yogo/nlp.pdf>

1. Una base de datos de palabras del estilo de WordNet (marca todas las respuestas correctas):
 - A. Está construida por humanos.
 - B. Permite obtener sinónimos y antónimos de palabras mediante grupos y reglas definidos por humanos.
 - C. Contiene *word vectors*.

2. *Bag-of-words* (marca todas las respuestas correctas):
 - A. Proporciona una representación discreta del texto.
 - B. Asigna 0 a los elementos no presentes del vocabulario en el vector resultante.
 - C. Asigna 1 a los elementos no presentes del vocabulario en el vector resultante.

3. Un *word vector* (marca la respuesta correcta):
 - A. Es una representación discreta de una palabra.
 - B. No permite obtener similitudes con otros *word vectors*.
 - C. Es una representación distribuida de una palabra.

4. Los *word vectors* también se conocen como (marca la respuesta correcta):
 - A. *Word embeddings*.
 - B. *Word discrete vectors*.
 - C. *Discrete representations*.
 - D. *Word contexts*.

5. El tamaño de *window* o contexto m (marca la respuesta correcta):
- A. Es un parámetro del modelo word2vec y se aprende durante el entrenamiento.
 - B. Es un hiperparámetro del modelo word2vec y se elige antes de entrenar el modelo.
 - C. Un valor lógico para este parámetro sería $m = T$.
 - D. Un valor lógico para este parámetro sería $m = 0$.
6. La dimensión d de los *word vectors* (marca la respuesta correcta):
- A. Es un parámetro del modelo word2vec y se aprende durante el entrenamiento.
 - B. Es un hiperparámetro del modelo word2vec y se elige antes de entrenar el modelo.
 - C. No es un parámetro ni un hiperparámetro y su valor no reviste importancia.
7. La similitud mediante cosenos (marca todas las respuestas correctas):
- A. Se basa en el ángulo entre dos vectores.
 - B. Varía entre -1 y 1, con 1 siendo la máxima similitud y -1, la mínima.
 - C. Varía entre 0 y 1, con 1 siendo la máxima similitud y 0 la mínima.
 - D. Se obtiene a partir de un producto escalar normalizado mediante el módulo de los vectores.
8. Word2vec utiliza como datos (marca la respuesta correcta):
- A. Una representación discreta de un conjunto de textos.
 - B. Un conjunto de textos.
 - C. Un conjunto de textos, pero necesitamos a humanos que asignen a cada texto una clase correcta, como en un problema de clasificación.
 - D. Word2vec no necesita datos.

9. Los *word vectors* resultantes de word2vec (marca la respuesta correcta):
- A. Codifican una representación de las palabras con ciertos valores semánticos, de manera que pueden calcularse palabras similares a otras y detectar relaciones entre ellas.
 - B. Codifican una representación de las palabras con valores puramente estructurales, de manera que solo podemos saber palabras que se suelen comportar como sustantivos, verbos, adverbios, etc.
 - C. Codifican una representación discreta de las palabras.
 - D. Ninguna de las anteriores.
10. *Negative sampling* (marca la respuesta correcta):
- A. Es una técnica que no se usa en la práctica, ya que basta con aplicar *gradient descent* sobre el problema completo de word2vec.
 - B. Consiste en utilizar solo una pequeña parte aleatoria de los textos de los que disponemos.
 - C. Es una aproximación al problema de optimización completo de word2vec que resulta mucho más eficiente computacionalmente.
 - D. Ninguna de las anteriores.